# University of Alberta

Simulating Strategic Rationality

by

John Simpson

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Philosophy


Spring 2010
Edmonton, Alberta

# Examining Committee

Adam Morton, Philosophy

Wes Cooper, Philosophy

Geoffrey Rockwell, Philosophy & Humanities Computing

Harvey Quamen, English & Humanities Computing

Duncan MacIntosh, Philosophy, University of Dalhousie

For my parents, Val and Phil, and my sister, Kate,
who have always believed in me and have never once asked:
"What are you going to do with a degree in Philosophy?"

This project explores the intersection of three topics: games, rationality and simulation. There are four major results produced by this exploration. First, it is argued that whether or not a particular process can count as a simulation for a particular person is entirely dependent on whether or not that person is able to see the process in questions as being relevantly and appropriately similar to the process that it is intended to be a simulation of. In short, whether or not something can count as a simulation is entirely observer relative. Second, a proof is offered that there are exactly 726 meaningfully different 2×2 games. This proof addresses a confusion in the game theory literature regarding exactly how many 2×2 games are possible. Third, an argument based on the statistical likelihood of encountering each of the various 2×2 games is advanced; presenting a serious challenge for the common practice of focusing on only a small number of games such as the Prisoner's Dilemma, Chicken, and Stag Hunt. From a small set of assumptions it follows these attention grabbing games are likely to be encountered at most eight percent of the time. This result has important implications for designing artificial intelligences and for understanding human behaviour. Finally, it is shown that under certain stipulated conditions playing games using the model of rationality presented by traditional economics, decision, and game theory, results in sub optimal outcomes. Specifically, there is at least one case preventing population inhabitants from having full information provides the population that they belong to with adaptive advantages. The populations used to show this behaviour choose from a status quo position and were inspired by the framing effect known as the status quo bias. In addition to these major results it is also shown that simply endowing artificial agents with theory of mind is not necessarily beneficial and an argument is made for creating/maintaining spaces for play within science.

# Acknowledgements

It is with the deepest thanks that I acknowledge the assistance of the following people in the preparation of this project:

- Adam Morton, my advisor, for his candid advice, accepting my need to tilt at windmills from time to time, and for always putting things in perspective.
- Wes Cooper for acting as my initial advisor when I first arrived at the University of Alberta, for suggesting that Adam Morton might be a better fit when he arrived, and for always asking all the right questions.
- The members of my committee for taking the time to read the (almost) final draft and for challenging me to think about the project in new ways.
- Don Ross for demanding more.
- David Crookall, editor of *Simulation & Gaming*, for believing in my work despite its running deeply counter to prevailing ideas.
- Lorraine Woollard for giving me the Community Service-Learning advantage.
- Vishaal Rajani for countless hours spent in pursuit of better ways to count and name games.
- Christ McTavish and Elizabeth Panasiuk for providing the counseling that only an office mate can and for Elizabeth in particular for providing me with the "thesis completion instructions" that still informs all my writing.
- Leah, Darren, Cassady, and Charlotte Spencer, for acting as an extended family when my real one was so far away.
- Garth Fitzner for removing the worry finding a place to live.
- Marnie Ferguson for being there, regardless.
- Jason Taylor for providing me with all the necessary real-world distractions.

- Melissa Manning for holding my hand through the end.
- Jim Shortt for being a good friend as long as the friendship lasted.
- Anna Kessler for challenging me to prove my beliefs regarding the intersection of faith and reason.
- The Ubuntu Community for quickly answering all my OS related issues.
- Xiao Guo for being brave enough to help with the statistical complexity that still riddles this entire project.
- Katie McDonough for her relentless commitment to editing even the longest of possible sentences.
- Vance Schamehorn for providing advice on initial pieces of code and for the use of an implementation of the Mersenne Twister random number generator.
- Anita Theroux, Wendy Minns, and Sussane McDonald, for making so many potential administrative nightmares disappear.

There are many more to whom I am indebted to but whom I cannot begin to list here. If this is you, your contributions are appreciated and not forgotten.

# Contents


**1 Playing with Simulations** **1**

1.1 Defining Simulation . . . . . 2
1.2 Defining Play . . . . . 4
1.3 Exploration vs. Proof . . . . . 8
1.4 Value to Science . . . . . 11
1.5 Value to Broader Aspects of Human Life . . . . . 13
1.5.1 Games . . . . . 14
1.5.2 Science as a Game . . . . . 17
1.5.3 Cheating & Spoilsporting . . . . . 18
1.6 Onward to Simulation . . . . . 21

**2 Identity Crisis: Simulations & Models** **22**

2.1 The Essential Nature of Models . . . . . 24
2.2 The Inherent Observer Relativity of Models . . . . . 25
2.3 The Essential Nature of Simulations . . . . . 27
2.4 The Relationship Between Simulations and Models . . . . . 30
2.5 What Counts as the Underlying Model? . . . . . 33
2.6 Two Paths to a Simulation . . . . . 36

**3 2×2 Games and Their Properties** **41**

3.1 Importance of 2×2 Games . . . . . 41
3.2 Properly Counting 2×2 Games . . . . . 44
3.3 2×2 Game Frequency . . . . . 50

**4 A Rational Agent Model for Simulation** **58**

4.1 Why Build Another Model? . . . . . 58
4.2 The Model . . . . . 63

4.2.1 The Agents . . . 63
4.2.2 Normal/Classic Agents . . . 65
4.2.3 Status Quo Agents . . . 66
4.2.4 Comparing Agents . . . 68
4.2.5 Environment . . . 70
4.3 More on Modelling . . . 73
4.3.1 Agent-based . . . 74
4.3.2 Ideal Type . . . 75
4.3.3 Evolutionary . . . 75
4.3.4 Non-spatial . . . 75
4.3.5 Varied-Model . . . 76
4.3.6 Benefits of this model . . . 78

**5 Comparing SQ and NC Agents 80**
5.1 Simulation Summary . . . 81
5.2 Results . . . 82
5.2.1 Possible Mistakes . . . 85
5.2.2 Relative Simplicity of SQ Agents . . . 86
5.2.3 Inherent Nature of SQ Agents . . . 87

**6 Results of Simulating Theory of Mind 95**
6.1 Modeling Theory of Mind . . . 95
6.2 Setting Aside a Philosophical Dispute . . . 96
6.3 Details of the Model . . . 98
6.4 Basic Results of Introducing ToM . . . 101
6.4.1 Extended Results of Introducing ToM . . . 104

**A 726 2×2 Games: Case-by-Case Proof 109**
A.1 The Proof . . . 113
A.1.1 Case 1 [1] . . . 113
A.1.2 Case 2 [2] . . . 114
A.1.3 Case 3 [3] . . . 114
A.1.4 Case 4 [9] . . . 115
A.1.5 Case 5 [6] . . . 115
A.1.6 Case 6 [10] . . . 116
A.1.7 Case 7 [12] . . . 118
A.1.8 Case 8 [72] . . . 118

A.1.9 Case 9 [48] . . . . . 119
A.1.10 Case 10 [8] . . . . . 120
A.1.11 Case 11 [54] . . . . . 121
A.1.12 Case 12 [36] . . . . . 122
A.1.13 Case 13 [171] . . . . . 123
A.1.14 Case 14 [216] . . . . . 124
A.1.15 Case 15 [78] . . . . . 124

**B 726 2×2 Games: Brute Force Proof 126**

**C Taxonomy: 726 2×2 Game 144**
C.1 Games of Type 1 . . . . . 145
C.2 Games of Type 2 . . . . . 145
C.3 Games of Type 3 . . . . . 145
C.4 Games of Type 4 . . . . . 146
C.5 Games of Type 5 . . . . . 146
C.6 Games of Type 6 . . . . . 147
C.7 Games of Type 7 . . . . . 148
C.8 Games of Type 8 . . . . . 150
C.9 Games of Type 9 . . . . . 158
C.10 Games of Type 10 . . . . . 163
C.11 Games of Type 11 . . . . . 164
C.12 Games of Type 12 . . . . . 170
C.13 Games of Type 13 . . . . . 174
C.14 Games of Type 14 . . . . . 193
C.15 Games of Type 15 . . . . . 217

**D Taxonomy: 729 2×2 Status Quo Game Positions 227**

**E SQ and NC Simulation Program Code 307**
E.1 Introduction . . . . . 307
E.2 Mid-Level Agent Description . . . . . 307
E.3 Program Framework . . . . . 314
E.3.1 Main Function . . . . . 322
E.3.2 Prepare Input and Output Files . . . . . 325
E.3.3 Trials Loop . . . . . 328

E.3.4 Population Loop and Beginning of Multi-Thread Processing . . . 328
E.3.5 Environment Loop . . . 329
E.3.6 Build Initial Population for Environment . . . 329
E.3.7 Collect Data on Initial Population . . . 331
E.3.8 Generation Loop . . . 331
E.3.9 Round Loop . . . 332
E.3.10 Build Games . . . 332
E.3.11 Match Agents and Games . . . 333
E.3.12 Play Games . . . 338
E.3.13 Update Agents . . . 339
E.3.14 Remove Poorly Performing Agents . . . 340
E.3.15 Spawning of Remaining Agents . . . 341
E.3.16 Collect Generation Data . . . 341
E.3.17 Additional Functions . . . 344
E.3.18 `agent.h` . . . 377
E.3.19 `game.h` . . . 410
E.3.20 `game.cpp` . . . 414
E.3.21 `match.h` . . . 437
E.3.22 `match.cpp` . . . 438

**Bibliography** **440**

# List of Tables


1.1 Comparing Players, Triflers, Cheats and Spoilsports . . . . . 16

3.1 Red Dress . . . . . 43
3.2 Four Ways to Represent the Prisoner's Dilemma in Strategic Form . . . . . 44
3.3 Eight Ways to Represent Red Dress in Strategic Form . . . . . 46
3.4 Comparing Game Definitions . . . . . 47

5.1 Chicken . . . . . 88
5.2 Bits Frequently Set for Playing ON the Status Quo . . . . . 93
5.3 Bits Frequently Set for Playing OFF the Status Quo . . . . . 93
5.4 SQ Positions No. 365 & 716 . . . . . 94

# List of Figures


2.1 Underlying Models . . . . . 36
2.2 Two Paths to a Simulation . . . . . 37

3.1 Count of Games Shared Between Count and Probability Groups 52
3.2 Frequency of Single Members with the 2x2 Probability Groups 53
3.3 Frequency of 2x2 Probability Groups . . . . . 54

5.1 Population Growth for NC Agents by Generational Ante and Initial Population Size . . . . . 83
5.2 Population Growth for SQ Agents by Generational Ante and Initial Population Size . . . . . 83
5.3 Average Payoffs Offered and Received by Agent Type and Generational Ante . . . . . 84
5.4 SQ Agents, bits set to "ON" . . . . . 89
5.5 SQ Agents, bits set to "ON" . . . . . 90
5.6 SQ Agents, bits set to "ON"–Closeup: 1360 - 1388 . . . . . 91
5.7 SQ Agents, bits set to "ON"–Closeup: 1616 - 1644 . . . . . 91
5.8 SQ Agents, bits set to "OFF"–Closeup: 2708 - 2732 . . . . . 92
5.9 SQ Agents, bits set to "OFF"–Closeup: 2450 - 2476 . . . . . 92

6.1 Population Growth for NC Agents by Generational Ante and Initial Population Size, AfterPlay=ON . . . . . 102
6.2 Population Growth for SQ Agents by Generational Ante and Initial Population Size, AfterPlay=ON . . . . . 102
6.3 Average Payoffs Offered and Received by Agent Type and Generational Ante, AfterPlay=ON . . . . . 103
6.4 Population Growth for NC Agents by Generational Ante and Initial Population Size, AfterPlay=ON, Discrimination=ON . 105

6.5 Population Growth for NC Agents by Generational Ante and Initial Population Size, AfterPlay=ON, Discrimination=ON . 106
6.6 Average Payoffs Offered and Received by Agent Type and Generational Ante, AfterPlay=ON . . . . . 107

E.1 Example of NC Type Agent Decision Procedure . . . . . 311

# Chapter 1

# Playing with Simulations

Science relies heavily on the propagation of, and continued adherence to, ideas, processes, and standards. This standardization is so important that it counts as one of the defining characteristics of science. In addition to standardization, science is also directed towards developing new ideas and new methods that resolve anomalies. Thomas Kuhn (1977) refers to the commensuration of these two directions as "the essential tension." This tension is not constantly balanced; it is possible for one side to overcome the other, even if only temporarily. When current theories about the nature and appropriate methods of scientific practice wins the tug-o-war the result is dogmatism, as scientists cease to be merely skeptical towards alternative ideas and methods and instead refuse them any fair consideration. When challenges to the established views and methods win, the result is a new set of standards, methods, or understandings that will (ideally) coalesce over time. If the change of perspective happens too often then it will be difficult for progress to be made towards answering the questions that science deals is directed towards. Frequent perspectival change in science is as debilitating as it is in politics; a recipe for long-term instability.

Judging by recent scientific history, it is dogmatism that must be purposefully guarded against; it is the most persistent of the two extremes and, given its insidious nature, it takes hold unnoticed. Dogmatism inhibits progress by hampering the ability of scientists to be openly creative while also blinding them to the value in the creativity of others.

Dogmatism is not just associated with particular theories or disciplines, but also with scientific practice in general. What counts as legitimate research or as a waste of research dollars is determined in part by the views of the

scientific community and so may become subject to dogmatism. Let it not be forgotten that a rigidity against the willie-nillie acceptance of new things continues to be a very important enabler of progress in science, the issue at hand is when this rigidity goes too far.

The costs of dogmatism may be grouped into two wide categories: the stunting of scientific progress and the failure to realize the goals for which science is being carried out in the first place. The history of science is full of cases where good ideas were held back by a refusal to give them due consideration (e.g. plate tectonics, sideways genetic influence). This potential cost is well-known and documented.[1] The effect of dogmatism on the ability of science to realize the goals that it is undertaken to achieve in the first place—the improvement of human life—is subtler and correspondingly less well studied.

Computer-based simulations are a relatively new technique in the toolbox of scientists. In the past half-century years since they first arose as a plausible option for more than a few specialists, they have become important additions to the tool boxes of scientists. Work on weather predication, climate change, structural integrity, and population dynamics are outstanding examples of what simulations have to contribute to scientific progress. Besides these direct practical returns, simulations also offer an opportunity to protect science from the consequences of dogmatism, but only if conscious effort is made to ensure that simulations do not become the next bastion of dogmatic proceduralism. They can do so because they are free from many of the restrictions that hobble traditional experiments, operating as a blank canvas rather than a page from a paint-by-numbers book. If appropriately respected, the sort of play encouraged by this open intellectual space can provide a fruitful landscape for exploring new ideas that can benefit both science and broader aspects of human life.

## 1.1 Defining Simulation

What counts as a simulation is still being sorted out by the philosophy of science community. It is argued elsewhere (Chapter 2) that there is an observer relativity that is inherent in the concept of a simulation which determines whether a process counts as a simulation and to whom it counts as a simulation.

[1] This is true at least for the positive side. The number of solutions to research problems sitting on the desks of the discouraged remains an unknown, and likely unknowable, element. The work of Gregor Mendel is a case in point.

A simulation is simply a process that is taken to be an imitation of some other process, real or imaginary, and will count as a simulation only for those people for whom this similarity relation is apparent. The most important class of simulations for science in recent years are *computer* simulations. Computer simulations are processes carried out on electronic computational machines that are seen to be appropriately similar to other processes, such as those programs that investigate the carrying capacity of geographic areas, stress assessments on aeronautical engineering components, or the predicted effects of combining various chemicals.

It should be noted that what counts as a simulation, computer or otherwise, is more a matter of a difference of degree rather than a difference of kind. There are parallels that can be drawn between any given process and any other process (as processes they both take time, if nothing else), but it is only after a sufficient threshold of similarity has been attained that a simulation may be claimed to exist. It is for this reason that most computer games, even those that are intensely realistic in their portrayal of human interactions, politics, and technological achievements, do not count as simulations for most of the people who play them. Consider the class of games known as real-time strategy (RTS) games. These often provide a top down view of a landscape and give the player the ability to create and direct an army over this represented surface. Despite the ease with which RTS games could be seen as qualifying as simulations of actual or possible military scenarios—some are even historically themed, with great attention to detail—they are not simulations for most players. Instead, they remain simply games or an attempt to illuminate historical curiosities. Put in the hands of others these games cease to be simply fun and take on a more informative role, showing what could have happened or what would happen in similar scenarios; it is for these people that RTS games become simulations. A similar perspective is required to transform any other process carried out on a computer, game or otherwise, into a simulation. Perception is all important here, a person may well be playing what others consider to be a simulation, because *they* draw the relevant similarity relations, but only be playing a game from her own perspective. This makes determining exactly what counts as a simulation a tricky prospect about which more will be said in Chapter 2.

## 1.2 Defining Play

Following Suits (1977) we will begin defining 'play' by identifying cases where "play" is often used in common language as a synonym for some activity other than the sort that interests us here. Thus, we "ought not concern ourselves with the word "play" when it is used as equivalent in meaning to such words as "perform," "operate," or "participate in" " (Suits, 1977, p. 120).

What is play then? "Play is not work" is certainly a common response to the question and points in the right direction.[2] This does not mean that play is anything not considered to be work (sleeping does not typically count as play although it can; it can also count as work). Rather, play is importantly distinct from work in such a way that the two concepts are best understood in contrast to each other. The most commonly accepted distinction between woork and play is found in the sources of their perceived value. Work is seen as a means to an end; its value is thus primarily instrumental—a task undertaken to attain something of value beyond the activity itself. Conversely, play is its own end, an end in itself and so its value is intrinsic. However, examples of activities having both sorts of value are readily available and stand as strong evidence that the distinction between work and play is more a matter of degree than a matter of kind.

The most significant piece of writing on play in the twentieth century was historian John Huizinga's 1931 *Homo Ludens* or "Man the Player", a book that examines the play element of culture. Huizinga offers the following outline of the necessary and sufficient characteristics of play:

> Summing up the formal characteristics of play we might call it a free activity standing quite consciously outside "ordinary" life as being "not serious", but at the same time absorbing the player intensely and utterly. It is an activity connected with no material interest, and no profit can be gained by it. It proceeds within its own proper boundaries of time and space according to fixed rules in an ordinary manner. It promotes the formation of social groupings which tend to surround themselves with secrecy and to

[2]Play as effort not aimed at any goal beyond the enjoyable experience of that effort. ARNESON, RICHARD (1998). Work, philosophy of. In E. Craig (Ed.), Routledge Encyclopedia of Philosophy. London: Routledge. Retrieved January 04, 2010, from http://www.rep.routledge.com.login.ezproxy.library.ualberta.ca/article/S068SECT1

> stress their difference from the common world by disguise or other means. (Huizinga, 1950, p. 13)

This definition enriches the work-play/instrumental-intrinsic distinction by explicitly calling out the autonomy of play and the contribution of play to the formation of social groups. Huizinga goes on to argue that play can be found across cultures in almost any activity; law, art, and war all have elements of play. He summarizes his investigation and position as follows:

> The spirit of playful competition is, as a social impulse, older than culture itself and pervades all life like a veritable ferment. Ritual grew up in sacred poetry; poetry was born in play and nourished on play; music and dancing were pure play. Wisdom and philosophy found expression in words and forms derived from religious contests. The rules of warfare, the conventions of noble living were built-up on play patterns. We have to conclude, therefore, that civilization is, in its earliest phases, played. It does not come *from* play like a babe detaching itself from the womb: it arises *in* and *as* play, and never leaves it. (Huizinga, 1950, p. 173)

As eloquent and persuasive as Huizinga's words on play are, they are not above controversy. Bernard Suits takes issue with the broad range of activities that Huizinga captures in his definition: "...ever since Huizinga began to find play under nearly every rock in the social landscape, quite a bit too much has been made of the notion. ...For there is little now that someone or other has not called play, from a cat chasing its tail to Aristotle contemplating the Unmoved Mover" (Suits, 1977, p. 117). The problem Suits finds with examination and attributions of the concept of play similar to and following from Huizinga's is that they count any autotelic activity whatsoever as an act of play.[3] This method of classifying activities is questionable because it amounts to a variation of the play-work dichotomy; the play-as-any-autotelic-activity-whatsoever criteria leads us to classify a wide range of activities as being acts of play when they are really first and foremost acts of a different sort, but not necessarily work. Generally speaking, contemplating God would count as a sort of "religious exeperience" (Suits, 1977, p. 118) and not as an act

[3] "Autotelic" is meant to convey the idea that an activity has value in and of itself rather than being carried out for some other ends. Autotelic value is thus value that is intrinsic to the carrying out of the activity rather than extrinsic.

of play, although it may have an element of play to it while also not qualifying as work.

As a way around this sort of problem, Suits offers his own definition of playing, "x is playing if and only if x has made a temporary reallocation to autotelic activities of resources primarily committed to instrumental purposes" (Suits, 1977, p. 124). This definition retains the intrinsic/autotelic definition of Huizinga's, but adds the additional criteria that play must involve the redirection of resources away from some other instrumental activity and in doing so makes explicit the tension that exists between work and play. Contemplating God now only counts as play when the resources involved in contemplating God, perhaps time, paper, pen, or a bench, were committed to other things that had instrumental value. With this change a great deal of the activities that count as play will be anything that is done when one should/could be working. Suits' definition also eliminates the question of whether professional athletes are playing or working; because they are making use of resources in the way in which they were intended—training and participating in sporting activities—the athletes are not playing.[4]

Suits' area of expertise is games; and given the close relationship between games and play his treatment of the two should be compatible. But Morgan (2008) argues that they are not compatible because of what he calls the "divided resource conundrum". This conundrum arises for Morgan because Suits claims that playing a sport like baseball with a baseball does not count as an act of play because baseball is the primary use for the ball. This seems to make it the case that in order to play baseball with a baseball that the ball must be used outside the parameters of a game of baseball by a player—"like seeing how much of the ball he could shove up his nose" (Suits, 1977, p, 125)—thereby destroying the game. Morgan believes that the solution to this problem lies in making a distinction between agent-neutral value and agent-relative value and then using this distinction to argue that the reallocation of resources required to play a game is inherent within the game and not outside it, thereby saving the game from destruction. Roughly put, games take purely instrumental ends or purely novel goals and make them intrinsically valuable goals. In doing this they satisfy the resource reallocation portion of Suits' definition of play when-

[4]Broadly speaking they can still be seen as playing by anyone who sees it as being the case that the resources involved in training or participating in the sport were meant for other things and have become co-opted for some autotelic activity that may be masquerading as something instrumentally valuable.

ever the game is undertaken since the game inherently requires the redirection of resources from instrumental to intrinsic ends.

Morgan's reasoning surrounding the intrinsic value inherent in games is interesting and quite possibly correct, but Suits' definition of playing does not require it. The conundrum that Morgan identifies is a result of a confusion about the extent of the claims that Suits makes when talking about whether or not baseball can be *played* with baseballs. Suits is not in danger of falling prey to the conundrum because he is not advocating the kind of reallocation of resources that Morgan believes him to be. To see this clearly, it will be helpful to consider Suits' extended parenthetical note on the subject in full:

> If Johnny is, for example, playing baseball, we must not say that he is *playing* just by virtue of the fact that he is playing *with* his baseball. That *with* is a different kind of *with* than the *with* that is required by the definition. For playing the game of baseball with a baseball is not a diversion or reallocation of the ball to a secondary use: playing baseball with the ball is the ball's primary use. In order to illustrate the kind of thing I mean by playing *with*, Johnny would have to be using the ball for some purpose other than baseball, and this other purpose would have to consist in some activity that Johnny would find intrinsically worth while, like seeing how much of the ball he could shove up his nose. (Suits, 1977, pp. 123-124)

When Morgan reads this passage he takes it to mean that baseball cannot be played with baseballs since it does not involve a reallocation of the ball away from its primary use, namely *playing baseball*. Reading the passage in this way results from focusing on the third sentence over the surrounding sentences, but this takes the comment out of context in a fundamental way. Taken in context the claim that Suits is making is much less troubling: using a baseball in a game of baseball is not enough to constitute *playing with* the ball. Since the ball is being used for its primary purpose this use does not constitute playing with the ball *simpliciter*. What is thus meant by saying that someone is playing baseball *with* a baseball is simply that the baseball is being used to carry-out the game, not that the baseball itself is being played with. Whether the *game of baseball* is being played or not will depend on whether, and for whom, the two-part definition put forward by Suits is satisfied. Where both the autotelic

and reallocation criteria are satisfied then the game is being *played* and where at least one of these conditions is not satisfied then the game is not being *played*.

One final note on play needs to be made before moving on, namely the relativity that will be present when using "play". Suits makes it clear "... that one and the same activity—e.g., listening to Beethoven, contemplating God—can correctly be designated "play" by designator x at the same time that it is correctly designated "not play" by designator y" (Suits, 1977, p, 126). The source of this relativity is that the criteria that make-up the definition of play are matters of degree and not subject to determination by some form of strict classification. The degree to which a person is undertaking an activity for its intrinsic versus its instrumental nature or whether or not the level or type of resource reallocation is sufficient to warrant a switch from considering the activity "not play" to "play" will change depending on the assessor and the context. This is not to say that there can be no agreement on such issues; meaningful discussion can typically be salvaged simply on the basis of a shared context of experience.

## 1.3 Exploration vs. Proof

Modern computer simulations have three characteristics that encourage play. First, they can be produced to imitate almost anything that can be conceived of and with great attention to detail. They are the electronic equivalent of the child's Erector Set or the blank canvas of the artist. Second, they are fast. Many very complicated simulations can be completed relatively quickly and it is easy to adjust the parameters used to direct the progression of the simulation to explore alternative possibilities. Third, the machines necessary to carry out many simulations are readily available. In combination these three features give the simulationist the ability to simply see what happens for a very wide set of systems under a very wide range of conditions. In short, the simulationist can create and observe the consequences of almost any system that meets his or her fancy without needing to worry about wasting limited resources; this is a powerful framework for inviting play.[5]

[5]One need look no farther for proof than the regular and consistent co-opting of new and more powerful computing resources for purposes other beyond their intended use, such as the creation of SpaceWars!

Not everyone will interpret this situation as an invitation to. The open possibilities that computer simulations provide will be seen as an invitation to play only by those with a predisposition to explore freely and widely. For others this openness will be a source of indifference, at best, and a source of anxiety and fear, at worst. In the same way that the existentialist finds that expressing the belief that we are alone and ultimately responsible for choosing the essences of our lives leads to fear and anxiety, so too can the open field of possibilities offered by computer programs create fear and a paralysis of choice. If anything is possible and it is the responsibility of the simulationist to decide what direction to take, then it can be difficult to find traction and a place for an idea to cling.

Much of this anxiety can be removed by directing simulations toward answering research questions within the disciplines with which the simulationist is already familiar. Such simulations are extensions of the models and theories that already exist in these disciplines, thereby providing a context for the simulation to function within. In this way simulations supplement alternative approaches to the puzzle solving of mathematical modeling (e.g. the replicator dynamics of evolutionary biology) or traditional experimentation (e.g. control group testing), providing a new media in which to experiment with our understandings and the consequences thereof. In these contexts simulations become tools used to advance proofs or disproofs.

Simulations are useful for more than proof and disproof, they can also count as acts of exploration on three three different fronts: the capabilities of the machines they are being carried out on, the capabilities of the simulationist(s), and what is possible (in the broadest sense of 'possible') given whatever system the simulationist can dream up that is within the capacity of the simulation device. Simulations that count as acts of exploration are often, but not always, acts of play. They obtain this status because, given the general nature of science, such exploratory simulations will easily count as a diversion of resources away from other purposes—perceived to be more useful by some—for ends that are clearly of an autotelic nature. Such acts are going to raise concerns within the scientific community, partially because of what will likely be deemed an inappropriate reallocation of resources but more poignantly because such acts are going to be seen as stepping outside the boundaries of what counts as science.

The contrast that arises between the use of simulations as proof and simu-

lations as exploratory play, amounts to a case in point for what Kuhn (1977) refers to as the "essential tension" within science. This tension results from the need of science to simultaneously standardize itself and yet remain open to divergent approaches. The use of simulations as tools that enable proofs or disproofs of ideas to take place, such as using computers to model weather patterns for prediction, is an act of reinforcing standard research practices. The only difference when the computer model is used is that the computer stands in for what the researcher would otherwise be left to do by hand (or not at all if the models and calculations are sufficiently complicated). As such, these simulations are directed by standard research methods and practices, including hypothesis formation, testing, accountability to established models, and so on. Simulations undertaken as explorations are not bound by the same standards (or at least not so tightly), becoming divergent research approaches as they move in directions not mandated by standard research programs. In moving outside of what might be considered "normal science" these exploratory simulations come under pressure to conform as the vast majority of the success of science has been as a result of this convergence (Kuhn (1977); Barnes (1985)).

As acts of exploration rather than (dis)proof these simulations are going to regularly fail to satisfy standard research practices. One particular place where this can happen is in the formation of hypotheses to direct the simulation. "Maybe something interesting will happen..." is hardly an acceptable hypothesis by most accounts and any attempt to clean it up to approximate something that would be considered appropriate would amount to an act of academic charades. The consequence of this is that exploratory simulations are going to be seen to be of dubious character by the general scientific community. As an example of this, consider a criticism articulated by O. V. S. Heath against traditional experiments of a similar exploratory character:

> ...I have found that my first hypotheses are more often wrong than correct (e.g. HEATH, 1950) and I believe that this is the common experience (DARWIN, 1958). Why then should we bother with a hypothesis until after we have discovered some new and startling phenomenon by trying this and that and observing what happens? It is a curious fact that such aimless experiments are almost always completely unproductive.[6] (Heath, 1970, p. 7)

[6]Heath goes on to say that this is likely due to some psychological mechanism that

Considering something to be "aimless" or "unproductive", even if it is only *almost* always so, is damning criticism within a discipline that prides itself on both giving direction to and being directed by productive pursuits. What then is the value of simulations that constitute acts of play and why should they be either pursued or allowed to be pursued within science? There are two answers to this, one within the context of science and one within the context of human life broadly construed. Both of these arguments are play specific, but in being made it must be remembered that simulations as play will be no less valuable than any other sort of play being considered.

## 1.4 Value to Science

Play, as an intrinsically valuable activity for anyone participating, is still able to provide instrumental value to science. This value may not be as directly tangible and trackable as the instrumental value of including a control group in an experiment, the peer review process, or hypothesis formation, but it is valuable nonetheless. The principle value that play brings to science is the new perspectives that it can bring to old problems, allowing for the boundaries of current knowledge to be challenged and possibly extended. Play is necessary for science to be the immensely productive activity that it is; without play to break-up the tedium of regular and prescribed scientific practice science will stall, lacking the spark to propel itself forward.

Play explores boundaries and generates new perspectives on old problems. Consider the case of what is often referred to as "playing with an idea". In such cases the first question to be asked is whether or not such an act will count as play under Suits' definition at all since it would seem to satisfy neither the autotelic nor the resource reallocation criteria. These are fair criticisms and when applied in many cases it will be shown that what is really meant by "playing" in such cases is something much more along the lines of "thinking creatively towards some directed end". However, this criticism will not apply in all cases though and it is likely that even in cases where it does apply it may do so only partially.

What is different in those cases where playing with an idea really does

predisposes us to be attentive in the appropriate ways when a hypothesis is in place. The metamessage is that for science to get done as effectively as possible, the scientific process must be followed.

constitute playing at least partially? Two things: the thinking involved is intrinsically valuable for the person doing the thinking and that thinking, as well as the time it takes to do it in conjunction with any other resources involved, like paper and pencil, is being redirected away from some other intended task. In short, the thinker is *playing* under Suits' definition. Such play might happen by innocently daydreaming while staining slides or through off-topic banter across a research desk that constitutes the neglect of a deadline or any similarly more important activity. Rather than being simply wastes of time these acts of play, at least when confined to scientific topics, provide fertile ground for the creativity that goes into seeing beyond the obvious. At this point it is merely a hypothesis, but it seems entirely unlikely that there are many cases of scientific progress that did not involve at least small acts of play of this sort, either during their initial conception or in their refinement.

There are also cases wherein specific acts of science constitute acts of play because they are being carried out for their own value and doing so involves resource reallocation away from "more important" tasks. The paradigmatic example of this is the young child who pilfers chemicals and other items from the kitchen, garage and workshop—all items with other intended uses—in order to simply do some "science." Even when such homegrown experiments are directed at specific tasks, such as the building of a rocket, which would seem to make the experiment an instrumental one, an element of play will often remain to the degree that the entire experience is rewarding in and of itself. The exploratory spirit of such children often remains when they become adults, allowing for at least some of their adult activities to be rightfully considered play. It should go without saying that even if such playing does not carry over into the career of the scientist it was clearly valuable in setting them on the path of science in the first place.

As a paradigmatic example of science as play, consider the case of rocket scientist Homer Hickam Jr. whose exploits at rocket making as a child have been popularized in the 1999 film *October Sky* which was based on his earlier autobiographical work *Rocket Boys*.[7] As a youth, Hickam followed a passion of rocket building despite the pressure from those around him to get his head out of the clouds and accept his fate in a mining town. As an adult he became an

[7]"October Sky" is an anagram of "Rocket Boys" and reference to the launch of Sputnik in October 1957. The change was apparently made to make the title more appealing to women.

engineer with the National Aeronautical and Space Administration, continuing at least partially in the spirit of the work which captured his imagination as a child.[8]

For some scientists it even seems that 'science' and 'play' are synonymous. Physicist Richard Feynman stands out as a particularly clear example of this. Aside from his reputation as a serious physicist who behaved in ways unbecoming of a serious physicist, play formed an importantly necessary part of the work that Feynman did. One particularly illustrative episode took place in a Cornell University cafeteria. One student threw a plate across the room and as it flew Feynman noticed that a wobble accompanying the spin and immediately set about calculating the ratio between the two. When asked by a colleague why he would undertake such an exercise he exclaimed, "It has no importance—it's just for fun!" and is quoted as citing this particular incident and the playfulness that it embodied as what allowed him to continue pursuing his work on quantum electrodynamics, work which earned him a Nobel Prize in 1965. (Ouellette, 2005, pp. 201-204)

For many scientists the difference between playing and doing science is going to be much more clearly defined and perhaps this is the norm. For others, science will simply be useful play. Regardless, play remains an important activity when considering the nature of science, not because it is science, but because of the value that it brings to science. Play allows scientists to make disparate connections that they would not otherwise see and allows them to find additional value in their jobs, keeping them interested and employed as scientists, and, consequently, propelling science forward.

## 1.5 Value to Broader Aspects of Human Life

Simply put, science can be understood as a game played within the broader context of human life. As such, there may be cases where exemptions to standard practice should be embraced and not merely considered. Acts of play within science, whether through simulations or otherwise, should be exempted from reproach in at least a limited way. This claim can be made to the extent that science may be considered a game in the sense described by Suits (1978). Seeing the goals of science subordinate to other goals that human beings may

[8]Current information on Homer Hickam Jr. may be easily found by visiting his personal website at http://www.homerhickam.com/about/bio.shtml (last checked April 11, 2010).

have, such as living a good life, will form a key part of this argument.

### 1.5.1 Games

To begin, let it be said that referring to science as a game is not meant to trivialize science (at least this is not the primary intent). In the most general sense, saying that science is a game amounts to claiming that the institution of games can be used as a model for science, allowing the role of science in our lives to be better understood. The simplification that necessarily follows from this claim is no more excessive than that carried out via any of the models that scientists employ in simplifying the world so that it is understandable; this trivialization counts as fair play.

There are two theories of games that stand as candidates with which to understand the claim "science is a game". The first is the theory of games originally put forward by von Neumann and Morgenstern (1967) who created the theory to analyze situations of strategic interdependence.[9] The second is the theory put forward by Suits (1978). Suits' theory, while not unrelated to Von Neumann and Morgenstern's, focuses on the act and nature of game playing as giving meaning and direction to our lives. Despite the wide use and popularity of the first candidate (it is what is recognized by most people as 'game theory'), it is Suits' theory that offers the most value in understanding science.[10]

#### Suits on Games

Suits defines a game as follows:

> To play a game is the attempt to achieve a specific state of affairs [prelusory goal], using only means permitted by rules [lusory goal],

[9]Such situations have at least two players whose actions are linked such that the choices of one player affects the outcomes of the other and vice versa.

[10]There is a third theory of games that could also be considered, the theory that there is nothing common about games at all that would allow us to build a theory of games in the first place. Such a theory is usually grounded in the claim that the use of "game" is so context dependent that it will be immune to any attempts to generalize it. This is the position typically attributed to Wittgenstein (2001). Unsurprisingly, Suits (1977) is opposed to this position, claiming in the preface to *The Grasshopper* that while Wittgenstein looked at games he did not give them as much attention as was warranted and so missed the fact that they really did all share something in common. Further, in an appendix added to a later edition, Suits remarks that "the question whether all things *called* games have something in common is very different from the question of whether all things that *are* games have something in common" (Suits, 2005, p. 163).

> where the rules prohibit use of more efficient in favour of less efficient means [constitutive rules], and where the rules are accepted just because they make possible such activity [lusory attitude]. I also offer the following simpler and, so to speak, more portable version of the above: playing a game is the voluntary attempt to overcome unnecessary obstacles. (Suits, 1978, p. 41) (Brackets are as they appear in the original)

Prelusory goals are "*... specific achievable states of affairs...*" with no limits on how such a state is to be brought about (Suits, 1978, p. 36).[11] *Pre*lusory goals stand in contrast to goals which are simply lusory. Lusory goals are also directed towards some specific state of affairs, but they have restrictions on how such states might be brought about. Within the context of game playing the lusory goal is simply to win the game, where winning is equivalent to achieving the prelusory goal via the rules that are in place. While it is possible to achieve the prelusory goal without following the rules, it is not possible to achieve the lusory goal in the same way since the rules are constitutive of the game itself and to break them is to cease to play the game at all.[12] The lusory attitude is simply the acceptance of the necessity of rules and one's acceptance of the rules for the particular game in question in order to make the game possible in the first place.

Attempting to apply this definition to the sorts of games that are most familiar, games like chess or checkers, is apt to lead quickly to a challenge regarding the necessity of prelusory goals. The challenge is simply that many games do not seem to have a prelusory goal at all, all their goals being tied up in constituting the game in the first place and Suits addresses this. This challenge only has teeth if there is no distinction made between the institution of a game, such as chess, and any individual act of playing that particular type of game (carrying out the institution of that game). The independence of the prelusory goal from any particular act of playing a particular game is what matters in understanding Suits' definition of games. This independence

[11] "Lusory" is taken from the Latin *ludus*, meaning "game". (Suits, 1978, p.35)

[12] Suits actually identifies three types of rules: constitutive rules; rules of skill; and penalty rules (Suits, 1978, pp. 37-38). The only rules of direct relevance for this discussion are the constitutive rules, which define how to play the game at all. Rules of skill determine what it means to play the game well. Penalty rules are a special subclass of the constitutive rules that outline what is to be done should the rules not be followed, thereby bringing the breaking of rules within the rules of the game.

| | *Recognize Rules* | *Recognize Goals* |
|---|---|---|
| *Players* | YES | YES |
| *Triflers* | YES | NO |
| *Cheats* | NO | YES |
| *Spoilsports* | NO | NO |

Table 1.1: Comparing Players, Triflers, Cheats and Spoilsports

can be seen in every case because it can be achieved through the violation of the procedural rules which are constitutive of the game.

Take the case of chess. Here the prelusory goal is to place the opposing player into checkmate. Checkmate is part of the institution of chess simply because it is a concept importantly related to chess, but it is not in and of itself a *rule* for how chess is played. To see this, simply note that there are lots of things that have to do with the institution of chess and the concept of checkmate that have nothing to do with playing chess at all. Simply arranging pieces on the board to create a checkmate scenario and what has come to be known as the "Queens Problem" are both examples of the use of the institution of chess without actually playing chess.[13] This distinction between the institution of a game and an individual act of playing the game gives rise to four possible types of "players" as through the various combinations of accepting and rejecting the constitutive rules and the prelusory goals.

Table 1.1 can be used to provide a clearer definition of what it means to have the lusory attitude necessary to make game playing possible—a person can only be counted as a *player* by recognizing both the rules and the goals of the game. The distinctions displayed in this table also illustrate just why it is that Suits feels justified in summarizing games as "...the voluntary attempt to overcome unnecessary obstacles" (Suits, 1978, p. 41): neither constitutive rules nor prelusory goals must necessarily be recognized and it is only through making this recognition that games become possible.

[13]A particularly excellent summary of the Queens Problem, including its history and expansion to boards of size $n \times m$ grids and toroidal surfaces, may be found at http://mathworld.wolfram.com/QueensProblem.html, last visited December 6, 2009.

### 1.5.2 Science as a Game

Science becomes a game as soon as its constitutive rules are combined with its prelusory goals. What is the prelusory goal of science? An initial answer might be "Acquiring knowledge, perhaps an understanding, of the world and all within it relevant to human life."[14] It is not this goal alone that defines science. In addition to this goal science has a myriad of rules, both explicit and implied, dictating what counts as an act of science and what does not. These rules include, but are in no way limited to a conception of what counts as an instance of the scientific process, the system of recognition, the accepted methods of education, the units of measure used in different fields, what counts as evidence, whose opinion is allowed to settle disputes, and what questions are important to pursue (Kuhn (1970); Kuhn (1977); Barnes (1985); Collins and Pinch (1998)). What these rules share in common is that they all reinforce a more important rule: *assume that all things can be rationalized and explained.* It is this rule that truly defines science. Without it, seances, personal anecdotes, etcetera, would all be admissible as evidence within science; that they are not is a testament to the power of this metarule. The lusory goal of science may thus be defined as the attainment of knowledge via an understanding that the world is mechanistic and rational.

None of the specifics of the rules or the lusory goal are necessary for the satisfaction of the prelusory goals of science.[15] This is true both generally and with regards to any specific rule. They define science as an institution and allow for any particular act to be evaluated within this institution. Were various rules to be added, changed, or dropped altogether we would cease to have the same institution of science that we currently recognize, but it is not the case that the institution would cease to exist altogether. As long as the prelusory goal and the metagoal of rational behaviour and explanation of all things are shared there is a case to be made for two differing sets of rules

[14] I recognize that this will be seen as a naive position by many, particularly those who are deeply critical of science as an institution of domination. While I do acknowledge these sympathies, I am not convinced that science is necessarily an institution of domination, rather that it becomes this when it is allowed to overstep our own, human, prelusory goals—whatever they may be.

[15] Binmore (1998) would refer to these sorts of rules as rules in "The Game of Morals". The game of morals is made up of all the rules that we have made up and that end up guiding our acts and decisions, whether other people are involved or not. Rules of the game of morals are distinguished from rules for the game of life. Rules for this game result directly for the nature of the world and include things like gravity and magnetism. Rules of the game of life *cannot be violated* whereas rules of the game of morals can.

constituting variations of the game of science. This is not something that Suits discusses, but the idea logically follows from the conception of a game that he forwards.

### 1.5.3 Cheating & Spoilsporting

If the prelusory goal of science is to give meaning to the world through the attainment of knowledge and understanding then why do the current rules of science—particularly the rationality metarule—for achieving knowledge and understanding? It is the result of accepting the metarule because the demands that it makes continually push towards totality; it is not simply that *science* should be carried out in a rational way but that *all aspects of the world* must be understood as rational, as fitting together in a mechanistic way. Whether or not we currently have a rationalized understanding of a particular event is not as important as simply presuming that it is in principle attainable for that particular event. The result of accepting the metarule of science is the acquisition of the lusory attitude necessary to for a scientific outlook, not simply in the lab or office, but everywhere.

This line of discussion will seem more than a little vacuous to anyone who would assert that the world really is the way that science says it is and then hold up a list of examples, likely consisting of discoveries and technologies, to show that not only is the scientific process valuable, but that it is also right. The implication of this protest amounts to something along the lines of "science is valuable, without it we would never have made these discoveries nor come to enjoy these technological advancements". Such a claim has the same character as Heath's assertion that when experiments without hypotheses are carried out alongside experiments with hypotheses it is clear that those without are as useless as they are aimless. What both these claims lack is evidence that goes beyond anecdotes. More than this, they exclude any non-scientific process from providing any value in the first place because the rationality metarule continues to operate for them. The fact that there seems to be only one option for attaining knowledge offers evidence for the totalitarian nature of science as strong as any protest that the successes of science make it obvious that science is the *only* way.[16]

Yes, science is useful, and yes, there are likely a great many discoveries and

[16]Deep internalization of the rules is equivalent to being mysticized in the terms of Sarte's existentialism.

technologies that would not exist without the scientific process. The challenge being made here is not that science is a terrible thing, nor that its claims are somehow false or unreal. Rather, the challenge is that the methods and viewpoints of science can be taken too far, and that it is very likely that they have.[17] The inverse of the argument that science has made many things possible is admitting that taking part in science precludes carrying out other activities, activities which might aid us in acheving either the prelusory goal of science or perhaps even other prelusory goals.

"What do we want from science?", "What should science be about?" and "How would we like to live our lives and what role will we allow science to play?" are the sorts of deep fundamental questions that we should be continually asking and answering because they remind us that the role science plays in our lives is not predetermined. These questions suggest that there are goals beyond science that direct and constitute it. They also suggest that science need not be constituted in the way that it is currently, that it might somehow be possible for it to be meaningfully different.

Dressing science in the suit of a game allows for the possibility of this meaningful difference. This possibility arises in part because claiming that science is a game trivializes it to a small degree and in doing so allows for distance to be gained from it, creating a chance to see the distinction between the separation of the lusuory and the preslusory goals. This realization allows for a number of actions to be taken to combat the totalitarian nature of science, actions that create the possibility of pursuing the prelusory goal on terms that serve broader human interests.

The separation of rules and goals allows for scientific dogmatism to be combated through conscientious cheating and spoilsporting. Recall that the discussion surrounding table 1.1 allowed for four different ways that people might stand in relation to a game. As anyone who has ever played games will know, cheats and spoilsports are antithetical to the very act of playing the game in question, whatever it may be. Cheats recognize the prelusory goal but do not abide by the rules while spoilsports simply disregard both rules and goals. It is this disregard for the rules that allows cheats and spoilsports to effectively catch the attention of players and in doing this create the possibility

[17]Kuhn (1977) certainly thought this was the case and I can find no reason to suppose that much has changed in the past thirty years, especially given the arguments put forward by Collins and Pinch (1998).

for change.[18]

Cheating would involve the recognition of the prelusory goal of science—acquiring knowledge—while disregarding the standard rules constraining exactly this goal is to be brought about. It is important to emphasize that this is not the sort of cheating associated with falsifying data, plagiarism, or industrial espionage. Cheating of these sorts is undertaken to achieve alternative goals that might be summarized as "Get recognized" or "Make money". The kind of cheating that is being advocated here amounts to recognizing that knowledge does not have to be sought through the scientific process nor is knowledge only valuable when it is either useful or rational. Cheating of this type involves allowing aesthetic, religious, or subjective experience to inform one's view of the world and that this information may be counted as knowledge on the same terms as the knowledge acquired through scientific processes is. Anyone claiming to have acquired knowledge in these nonscientific ways is dismissed. Since the rationality metarule implies that the only way to gain meaningful knowledge is through the scientific process then anyone who has not followed this rule cannot have real knowledge. Cheaters, after all, are said to never prosper and the dogmatic adherence to the metarule helps to ensure that this continues to be the case.

Spoilsports present a different challenge to science. These are people who do not recognize either the goal nor anything approaching an important subset of the rules of science, substituting instead different goals and different rules. Standing outside of science these spoilsports can pass as ghosts through the scientific world. As a result of following different goals and rules, spoilsports will frequently perform acts that will be surprising to those looking at the world through scientific lenses. Interacting with these people will amount to playing blocks with someone who is playing hopscotch, which is to say not at all. While it is possible that spoilsports might become ghosts to the scientific world this is not a likely scenario given that the metarule demands that all things be rational and mechanical, including these people. With this metarule in place conflicts become inevitable.

[18]The third type of non-player is the Trifler, who recognizes the rules, but not the goal. Such people allow the game mechanics to continue to operate despite having no interest in the goal. In this way they allow the game to continue for others in a way that does not interfere directly with the play of others, insofar as they still "play" by the rules. Such is the role of bureaucrats and paper shufflers who exist *sans* vision of any larger picture of their work.

While cheating and spoilsporting against the institution of science might simply be done for the sake of being annoying, there is perhaps a higher reason for doing so: challenging the rationality metarule of science. To the extent that this metarule is the source of the totalitarian nature of science it is necessary to fight against it should one of two things be the case:

1. a person shares the prelusory goal of science, in which case the metarule will bar access to knowledge and understanding that cannot be acquired through the methods of science, or

2. a person has their own prelusory goals, where the metarule will actively work to undermine and devalue these goals.

It is in the hope of either pursuing the prelusory goal of science or any other prelusory goal that cheating and spoilsporting should be undertaken and it is in this spirit that such acts count as conscientious. The realization and carrying out of acts such as these are tickets to the game that might be referred to as "living a good life".

## 1.6 Onward to Simulation

The arguments given in support of play as an important component of science extend directly to simulations that count as acts of play. A simulation undertaken with lab resources to simply see what happens may involve a more significant, and therefore more obvious, reallocation of resources, but it will be, in principle at least, no less valuable than any other sort of playing with ideas that might be undertaken. It may even be the case that the simulation will prove to be *more* valuable given that it is capable of carrying forward relationships and performing calculations with a precision that dwarfs most human capacities. This does not mean that simulations should be allowed to be undertaken completely willie-nillie or on a massive scale; for it must be remembered that much of the success of science lies in its methodological and homogenizing approach to the treatment of techniques and ideas. Rather, what should be taken away from this is an openness to the possibility that playing with simulations is something to be partially encouraged since it offers benefits to science—there is still much to be explored both about and with simulations and playing is an effective, albeit often inefficient way to do this—and in the intrinsic value of the act itself.

# Chapter 2

# Identity Crisis: Simulations & Models*,†

Simulations have been around for a very long time and have taken on various forms including mock battles, role plays, and other sorts of games, but they have recently received new attention due to their increasing use as scientific tools, particularly in cases of complicated systems such as those that give rise to social phenomena (Skyrms (2005); Namatame et al. (2002); Liebrand et al. (1998); Young (1998); Sober and Wilson (1996); Skyrms (1996); Axelrod (1984); etc.). The speed and flexibility of modern digital computers has made it possible for simulations to be used as both substitutes for and augmentations to traditional methods of scientific investigation. Despite increased attention by the philosophical community (Humphreys (2004); Winsberg (2001); Wins-

*The body of this chapter has been accepted for publication in the journal *Simulation & Gaming*. It is part of a special anniversary edition of the journal and will appear in print in 2010. As of April13, 2010, it is available online at http://sag.sagepub.com/cgi/rapidpdf/1046878109334007v1.

†This chapter was born out of a comment made to me by Professor Wes Cooper at the beginning of my thesis research. This comment amounted to, "...but why should we trust simulations at all?" As is often the case in the face of such questions the answer was not as obvious as it was before the question was asked. Throughout the development of this chapter I am indebted to Professor Adam Morton for advice and encouragement on many drafts. I also benefited from the help and assistance of Till Grüe-Yanoff. I am particularly thankful that Till took the time to suggest with some detail what ideas could profitably be kept and what could profitably be dropped. Finally, I would like to thank Katie McDonough for catching many typographical errors and grammatical atrocities and, more importantly, for forcing me to be both clear and concise. Any remaining errors are solely my own responsibility.

berg (2003); Hegselmann et al. (1996); Hartmann (1996); Epstein and Axtell (1996); Crookall et al. (1987); etc.) the extent to which computer simulations should be used as tools of scientific inquiry remains an open question. Specifically, while both the philosophic and scientific communities seem open to the use of simulations as tools of scientific investigation, exactly what counts as a simulation and the degree to which their use may be justified are questions that have yet to receive widespread agreement.

This paper puts forward three underappreciated ideas and consequences that must be taken into account should the philosophic community be serious about providing a full explication of the nature and legitimacy of simulations as tools of science. First, there can be no purely objective view regarding what counts as a simulation and what does not. Ultimately, the status of a process as a simulation or not is importantly observer relative since simulations are based on representational relationships drawn by individual observers. So in order to answer questions like, "Is x a simulation?" we need to know from who's perspective the question is being asked. Second, many simulations are not exactly what we think they are. This is particularly true of simulations run on digital computers as the process that ultimately takes place is likely not identical with the simulation that was intended due to the compounded impact of small changes during coding, compilation, and even execution. Consequently, anyone making use of computer simulations to explore reasonably complicated systems must be prepared for the possibility that the results ultimately produced represent a different system than was initially intended and they must qualify any results accordingly. Finally, processes become simulations through a combination of direct and indirect methods of recognizing the relationships between the actual process and the stand-in—the simulation—the relationships between which are observer relative.

To make these points in full it is necessary to explicate the relationship between models and simulations and, to a small degree, the nature of models themselves.

## 2.1 The Essential Nature of Models

Models are models because of how they are used. This is true regardless of the theory of models considered.[1] In order for any object, physical or mental, to be employed as a model the employing agent must draw the appropriate similarity relationships between the thing that is to be the model and the thing that is to be modeled. A similarity relation is any way in which a person may consider two things as being related to each other, such as size, shape, color, or function, so that one object may serve as a stand-in for another. Stand-ins, or models, are needed in various situations to overcome challenges with the original object, including complexity, size, delicacy, uniqueness, and cost. An object cannot be a model unless it can operate as one. This cannot happen unless what ever use it is put to is seen as being representative of similar uses in another object, which is to say that the appropriate similarity relations, whatever they might be, must be drawn. Once an appropriate number of these similarity relations are perceived one object becomes a model of the other. The following four consequences follow directly:

1. *Anything can be a model of anything else.* Clearly some objects will be better suited to bear similarity relations than others given the experiences and mental capacities of the observers involved, but in principle no object, mental or physical, is barred from acting as a model of any other object. Salt mills and pepper pots often stand for human beings during dinnertime discussions and there is no rule of logic that prevents anything else from being used in this way. Further, actual things can model hypothetical or imagined entities and vice versa, such as when an architect models a building that may never exist or when a mental model of a combat zone exists in the mind of a general.

2. *An object's status as a model depends on the existence of a set of similarity relationships, not on the strength or accuracy of this set.* Simply knowing that a collection of scribbles on a napkin is a model may be enough to make the scribbles a model, but not necessarily enough to make it a useful model (perhaps the agent is told that the scribbles are a

[1] For a very useful discussion on models see Mary Morgan and Margaret Morrisson's Models as Mediators (1999) which has its roots in earlier works, notably Cartwright (1980), Cartwright (1983), Nagel (1961), and Hesse (1966). For more information on syntactic and semantic models see Giere (1988) and Suppe (1977).

map of the city, but does not know which lines represent which streets). It is this consequence of the nature of models that might allow some people to see the wall behind Searle's desk as a model of the Wordstar program even if we do not agree that it is actually an implementation of it (Searle, 1992).

3. *Models are contingent on minds.* The necessity of drawing a set of similarity relations between the object that is being modeled and the object acting as the model means that models are principally mental entities—minds are required to give them their status as models and this status is contingent on the continued existence of such minds. This is not a claim that the object which is operating as a model is necessarily a mental entity, but simply that it could not exist as a model without a mind to make it one.

4. *The status of a thing as a model is observer relative, holding true for some agents but not necessarily for others.* This is the most important consequence of the general criteria for something to operate as a model for the purposes of this paper because simulations will ultimately inherit/parallel the observer relative nature of models. The next section is devoted entirely to making this idea clear.

## 2.2 The Inherent Observer Relativity of Models

To tease out the inherent observer relativity of models consider the case of an arrangement of prefabricated parts that someone might assemble together, ultimately claiming that they have built a model of a ship. If it is not clear to an observer that the assembled pieces are appropriately similar to a ship, actual or imaginary (perhaps the modeler did such a poor job during assembly that the arrangement is unrecognizable to anyone else as a ship in any way at all), then the arrangement is not a model of a ship for that observer. It could be a paperweight or a hammer or piece of modern art or a model of something other than a ship (a junk pile?), but it is not a *model* of a ship. It may even be the case that for other observers the assembled pieces bear the requisite similarity relations necessary for it to be recognized as a model of a ship, but this only makes the object a model of a ship for these observers and not for any observer who cannot see the similarity relationship. Often,

an observer may be helped to draw such similarity relations, but until this is done it remains the case that it is not a model for that observer.

Telling an observer that the item on the table is a model ship is not necessarily sufficient to make it a model ship. The observer may well have no idea what a ship is in the first place or simply not see how any ship, real or imaginary, and the collection of pieces might possibly be related. Further explanation may overcome these impasses, but success in this matter is not guaranteed.

A further complication is that it is possible that the same thing may model different things for different people or for the same person at different times. For example, two people may agree that the object in question is a model of a ship but disagree on the qualities that make it such. For one person it may be a model ship only insofar as its visual qualities are concerned (e.g. it *looks* sufficiently like a ship to them) while to the other it is a model of a ship only insofar as its physical characteristics are concerned (e.g. it *behaves* sufficiently like a ship to them when placed in water).

As a further consequence of the observer relativity inherent in the existence of models, it is possible for an object to be used as a model might be without the object becoming a model. Consider again the model ship. An observer who has never seen a ship and knows nothing about ships is given an object believed by others to be a model ship and the naïve observer takes it in the bath. The object floats in the bath as a model ship should, but it is not a model for the bather because it is not recognized by the bather as such. This is similar to the case where a newborn holding a stick cannot be said to be using the stick as a model sword. Things are not models until we use them as such, in the broadest possible sense of "use". The property of being a model is lost as soon as it ceases to be used as such—a stick may be used as a model of a ship, albeit a crude one, by children playing in a stream but it loses this status as soon as they forget about it and move on to other pursuits, perhaps using the stick to make a dam or a slingshot.

This claim holds for models that are more abstract than the ships that might be found in the hands of would-be hobbyists, such as the models that would be encountered in advanced classes in logic or mathematics. In the case of such abstract models it is not unreasonable to suppose that a person might stumble upon something that is referred to as model for a particular theory without that person having any knowledge of the original theory at all. If the

person finding the formula recognizes that it is intended to be a model or that others have used it as a model then it can be a model for them insofar as they use it as a model based on this recognition (they might refer to it casually in a paper). As their understanding of the formula and its connections to some phenomenon deepens so will the status of the formula as a model for the user. Again, the status of anything as a model is determined by the existence of a set of similarity relations as observed by the people involved.

The observer dependence of a model's existence should not be used to argue for an ultimately self-refuting collapse or loss of meaning regarding what counts as a model. There is clearly a great overlap in the things that different people consider to be and use as models: toy planes, instructive diagrams written on napkins, and many works of mathematics and logic are all commonly agreed to be models. This overlap counts as evidence of sufficient commonalities regarding what we consider a model to prevent claims of a complete loss of meaning. While issues surrounding what constitutes a model may raise problems in theory, they are not particularly troublesome in practice, likely due to our shared physiology and culture.

## 2.3 The Essential Nature of Simulations

Simulations "*... imitate one process by another...*" [emphasis in original] (Hartmann, 1996). Imitation requires that similarity relations be drawn between the object doing the imitation and the object being imitated. Insofar as simulations have the same relationship properties as models, as introduced previously, simulations are processes that model other processes. This makes simulations models while it is not the case that models are necessarily simulations unless there is a process involved. As a consequence of the need to have similarity relations drawn by individuals, what counts as a simulation will also inherently observer relative, a conclusion that is not immediately obvious by looking at some popular definitions of simulation. Consider the definition of a computer simulation given by Humphreys:

> System S provides a core simulation of an object or process B just in case S is a concrete computational device that produces, via temporal processes, solutions to a computational model... that correctly represents B, either dynamically or statically. If in addition the

> computational model used by S correctly represents the structure of the real system R, then S provides a core simulation of system R with respect to B... In order to get from a core simulation to a full simulation, it is important that the successive solutions which have been computed in the core simulation have been arranged in an order that, in the case of our example, correctly represents the successive positions of the planet in orbit ((Humphreys, 2004, pp. 110-111).[2]

While I believe that this definition correctly captures many notions relating to what should count as a simulation I am concerned that the distinctions made are not robust: the distinction between core and full simulations is arbitrary for cases where there is no single correct order; and pressing the distinction between core simulations and full simulations that correctly represent real systems will show that there is no distinction at all (the problems surrounding this distinction will be discussed following the groundwork laid out in the next section). Further, this definition carries with it the implication that what counts as a simulation can be settled objectively, which is not the case.

To illustrate the arbitrary nature of the distinctions between core and full simulation as well as the problem of trying to settle what counts as a simulation in some objective fashion, let us consider the case of the four color theorem—the conjecture that there is at least one coloring scheme for any planar map such that only four colors are needed to ensure that no adjacent cells share the same color—and the use of computers in "proving" it by generating and dismissing roughly 10,000 possible cases (Appel and Haken (1977); Appel et al. (1977)).[3,4] Taking Humphreys' definition at face value leaves it

[2]Humphreys (Humphreys, 2004, p. 109)defines a core simulation as "...the temporal part of the computation process."

[3]In addition to its use in questioning what can be counted as a simulation, the proof of the four-color theorem is also interesting for the controversy it brings regarding the nature of proof in general. There are two challenges in particular that it raises. First, it is unclear how it stands as a proof when compared to more traditional methods of proof (Staff (2005)). Checking computer based proofs by hand is typically impossible, possibly requiring even more time than it would take someone to complete the actual proof itself, and any journal referee should have a rare collection of skills to judge the success of the implementation—programming languages, computer science, philosophy of computing, and mastery of the relevant subject area. Second, proofs by simulation might best be seen as constituting brute force proofs (i.e. proofs that test every possibility in order to prove a conclusion) and these are typically considered less desirable by the mathematical community than proofs making use of more elegant and elaborate forms of reasoning.

[4]The four-color theorem is not unique in its explanatory value for this section, it is just rather well known. Other similar uses of computers, such as solving checkers by testing each

unclear whether the use of computers in this proof constitutes a simulation at all. First, it is unclear that the program used by Appel and Haken constitutes anything akin to the computational model demanded for there to be a core simulation. This is not a substantive challenge though, since Humphreys explicitly allows that mathematical objects, such as probability simulations, can be represented by core simulations (Humphreys, 2004, p. 111). More importantly, no matter how the proof is conceived of there is no clear or particularly correct order in which the core simulation components need to be placed (all that matters to the proof is simply checking every case) and so it is unclear that the distinction Humphreys makes between core and full simulations is meaningful.

The proof that Appel and Haken provide can be seen as either one very large computer program or a collection of small computer programs. The distinction made by Humphreys between core and full simulations does not apply to either. If the proof is seen as one large program then the core simulation components would be the tests of each individual case. The order of the tests has no bearing on the final result though, making it unclear how to order them so that they would qualify as a full simulation according to Humphreys. If any ordering is possible for full simulation status then the distinction between core and full simulations collapses for this case; so further revision of the definition is required. In the case where the proof is seen as a collection of programs, each one representing what a mathematician would do in each case, it is also not clear that there is only one way to order the components of the test and the same challenge arises.

Humphreys' definition implies that only processes with a clear temporal order count as simulations—the definition includes as an example a system with a clear temporal order—but does not explicitly demand that *only* systems with definite temporal orderings can bring about full simulations.[5] I suggest that the space of this silence be filled by recalling that simulations, like models, do not need to represent actual processes but might also represent hypothetical or ideal processes. In the case of the four-color conjecture the computer may

of the roughly $5\times1020$ possible positions Schaeffer et al. (2007), could easily be substituted.

[5]Humphreys' definition may be silent on this issue, but Humphreys is not. During the 2006 conference on Simulations and Models in Paris he made it clear during the question period following the presentation of an earlier draft of this paper that cases such as the four-color theorem are calculations, not simulations. His reason for disagreeing was (partially) that such uses of computers do not have the proper temporal character that simulations require, an argument which I hope is at least partially diffused in this new version.

be seen as simulating one of the many possible orderings of the process that a mathematician would go through *if she was willing and capable of devoting many lifetimes towards following the same procedures carried out by the computer*. Details aside, the computer, after all, just picks out a case, tests it via whatever process the programmer selected, records the result, puts the case aside and then moves to the next case, which is just what a mathematician would have to do were she to complete the proof herself. Of course, the computers used in the four-color theorem are performing a number crunching procedure, but this can be said of any computer process whether it is held to be a clear example of a simulation or not. Humphreys' definition is thus, on a charitable interpretation, compatible with more than simply processes that have singularly correct orderings of events.

The core oversight in the definition that Humphreys provides is that it is intended to be an objective one: it is given in the spirit of being a final definition that can be used to achieve unanimous agreement on what is a simulation and what is not. This puts his definition at odds with the view of simulations presented here. Recall that, simulations cannot exist *simpliciter*; which is to say, simulations come into existence only with respect to an observer and his or her ability to see a set of relationships between the process that is to be the simulation and the process that is to be simulated. Put another way, a process becomes a simulation when it is deemed to be appropriately similar to some other process by a given observer, not because it satisfies some formal definition.[6]

## 2.4 The Relationship Between Simulations and Models

The nature of the relationship between simulations and models is exposed directly by considering what happens to a simulation as it is halted and then resumed. Any kind of simulation will do for this thought experiment but, for the sake of continuity with the rest of the examples and general ease of explanation, suppose that the simulation is carried out via a computer, or, in terms used by Humphreys (2004), via a concrete computational device. What the simulation is of is also irrelevant just so long as you recognize it as a

[6]Unless, of course, it is the satisfaction of a given formal definition that makes the candidate simulation a parallel of another process.

simulation. Now, imagine that a simulation is taking place and consider what will remain if the simulation is halted; the computer simply ceases processing without the loss or degradation of any states involved in the simulation. Is this suspended process a simulation? No—it was, but in this halted state it is not.

We speak of simulations as processes; and since the process has been halted (you stopped it after all) no claim that a simulation is taking place should be made. Even though there is no simulation taking place within the hardware there remains a collection of states representing the data that make up both a description of a state of affairs and instructions for what to do with these descriptions. The instructional data can be traced back to the high-level programming code in which the program was designed.[7] Providing that some set of relations is believed to be represented by this code we can consider the resulting program to be a representation of this set of relations which makes it a model. The data that remains when stopping the simulation embodies this set of relations and, through its lineage, the representative character of these relations, qualifying it as a model as well. While modeling with code potentially has a host of functional differences from other types of modeling that could be linked to the same process (e.g. diagrams, written descriptions of processes, physical replicas, etc.), program code meets the basic criteria to be a model just so long as it bears appropriate similarity relations. Next, imagine that the originally halted process is allowed to resume. Now is there a simulation? Yes–stopping and starting a simulation is no different than it would be for the world if time was stopped and then restarted (since the entire frame of reference would be affected no knowledge could be had of the event from anywhere within the frame). Further, if we are allowed to start a simulation in the first place then resuming the process that originally counted as a simulation will still count as one since there is no technical difference between this restart and beginning an entirely new simulation from an identical initial state.

The data that resides within the computer is unaffected by the halting

[7]This is an oversimplification of the complexity and number of stages involved in translating back and forth between the high-level programming code that simulations are written in and the machine language that a computer is capable of operating on. More will be said on the possibility of tracing this lineage successfully in the next section. For a fuller discussion of the complexities involved in any claim that a piece of high-level programing and the final compiled code are "the same" see Fetzer (1987).

and restarting procedure that we have been imagining. We know the model remains during these interruptions because we could, at least in principle, extract it from the computer at any time as it is held within actual physical states of the computer. It is the consistency of this model under these pause and play conditions that allows the simulation to be resumed as if it was either started for the first time from that position or never halted at all. It is not merely coincidence that the model exists whether the simulation exists or not; rather, it is indicative of an important relationship between the model and the simulation—the model held in the instructional data is capable of endowing simulations with their requisite similarity relationships to other processes. These models are the basis for the simulation's existence in the first place and they direct the behavior of the simulation. Let us refer to a model that stands in this way to a simulation as an *underlying model* to distinguish it from models that are not connected to simulations in this way.

The attachment of a simulation to an underlying model is an important difference between simulations and other computer programs. If only the processes/behaviors of a computer program carrying out a simulation are compared to a program not carrying out a simulation it would be impossible to discern which was the simulation and which was not. Both the simulation and the regular program are simply collections of states inside the computer that shift themselves and other data around; and this is not enough to constitute a simulation. Again, the distinction between a simulation and a non-simulation does not rest in the process, but in the character that simulations gain from the connections that an observer might draw between the computational process and some other process. In other words, where one person might see a simulation another may only see a computer program and vice versa. Simulations exist when an observer links them to a process. Models and simulations are what they are because of how we see and use them, not because they have some special property inherent in themselves.

Unfortunately, knowing that every computer simulation is supported by an underlying model does not guarantee that said model can be found once a process has been recognized as a simulation. It may be difficult—even impossible—to clearly locate the underlying model. Furthermore, many candidates for this model are almost impossible to interpret. The two most likely places to find the underlying model are in the minds of those taking part in the simulation, in the case of role-plays and games, and in the physical make-up

of the objects of the simulation, as in miniaturizations and computer simulations. Unfortunately, suggesting locations for underlying models is relatively easy when compared to actually finding them which, in turn, seems relatively easy when compared to being able to express exactly what the underlying model is for any particular simulation.

## 2.5 What Counts as the Underlying Model?

On initial consideration there are two plausible candidates for the underlying model for any given computer simulation: the model that the simulation is intended to actualize and the model expressed in the written program intended to realize the model and create the simulation. In an ideal world this would be an inconsequential distinction, since these two would always be identical. Unfortunately, in the world we live in such models differ regularly. Setting aside cases where outright programming errors cause the identity relation to fail, the fact remains that computers are physical machines with physical limitations; and implementing any model that demands more than the computer is capable of will necessitate modification. In high-level programming code such changes are typically brought about by intentional tweaking on the part of the programmer in an attempt to increase the tractability or speed of the program. These intentional changes in conjunction with non-intentional ones, such as programming errors and misunderstandings, make it unlikely that the intended model and the final coded model end up the same. Given this likely failure of identity and the fact that the coded model is closer to producing the final simulation than the ideal model that inspired it, it is the coded model that is the stronger candidate for standing as the underlying model for any given computer simulation.

However, caution should be taken when considering the coded model to be the underlying model since it is unlikely that even this code is ultimately responsible for producing the simulation. Between the coded model and the simulation that will finally be output is a complicated series of translations that can change the original program into a form that can be implemented on a given class of machines. During this compilation process many things that are part of the intended model may be modified and approximated so that the computer can complete the task as accurately as possible. This can happen as straightforwardly as truncating real numbers to fit within the architecture

of the machine. More subtle changes can be made as well though, especially when coding libraries are used for performance increases. For example, direct exponentiation is a resource intensive process even for modern digital computers, so programmers often look for alternatives when a great many exponents must be dealt with. Coding libraries that have simplified routines are often turned to, but these sacrifice accuracy for speed, compounding any effects due to rounding already present.[8]

Besides the direct pragmatic differences that compilation brings to a model, there are two conceptual challenges that work to prevent equating the coded model and the compiled model, even under a qualification like "close-enough". First, the compiled model will *look* significantly different from the coded model. This may seem trivial since we can, at least in principle, interpret the data after compilation and then reconstruct the original code. Unfortunately, in many cases the disruption and distortion of the model would be so great that it would be akin to disassembling the model ship into its molecular components, laying the components in a row and insisting that they still constitute a model of a ship as they could, in principle, be reconstructed. After the compilation stage the ability to draw representational connections becomes sparse or non-existent to the point where anyone who believes that there is still an appropriate model for the task in the compiled code is making that assessment based on faith. This is the same faith that a traveler to a foreign land puts into trusting translations from others. Second, and more mundanely, although compilation is an algorithmic process, compilers are immensely complicated and expansive programs in their own right and there is nothing preventing the compiler from doing things that anyone attempting a reconstruction would not expect.

Perhaps unsurprisingly at this point, it is not necessarily the compiled model that ultimately produces the simulation either. What does ultimately produce the outputs that become interpreted as the simulation? A fusion of the model held in the compiled code and the idiosyncrasies of the specific

[8]In Fortran, programmers have the option of making use of vector libraries which can offer speed increases of five times. As a demonstration I offer the results of an example created by Masao Fujinaga of the University of Alberta Research Support Team. This code runs a computationally intensive process many times, recording the result of each run and the time taken to completion. The only difference? One run uses the formula $y^{2.4}$ for a repetitious calculation and the other replaces this formula with $\texttt{vexp}(2.4 \times \texttt{vlog}(y))$, a piece of code that makes use of the vector math library to save time. The end result is that the version that tackles exponentiation head-on takes 3.45 seconds to complete and returns 191699257756.619598 while the version that exploits the vector math library takes 0.68 seconds to complete and returns 191699257756.619629.

machine that implements the compiled code.[9] This result is inevitable because as much as we would like to believe that at the most basic level all computers are equal, this is not the case. Not only is it highly probable that two machines with identical specifications purchased from the same manufacturer at the same time will not be truly identical in their ability to perform a task, but the potential for differences grows over time and use. Simply put, automated assembly lines and meticulous testing cannot eradicate errors in machines that are assembled from a multitude of pieces.[10] Fortunately, when errors do occur there is a possibility that they will either be inconsequential, as when they result in altering the accuracy of a calculation at a level of precision well beyond our current requirements, or self-healing, such as when a bad disk sector is automatically detected and blocked from further use. Idiosyncrasies of particular machine tokens aside, implementing a simulation on one type of machine rather than another can have significant effects as well. The clearest case where this can happen—it is a simplified version of this scenario that led Lorenz to father the field of mathematical chaos in the first place—is when a simulation has at least a small nonlinear element within some floating-point calculation and is run on both a 64-bit machine and a 32-bit machine. The results of these two simulations will diverge simply because the 64-bit machine allows for greater precision in its calculations. Such differences will compound over time, particularly in a non-linear system, making a significant impact on the outcomes of the simulation across these two machine architectures.

This means that there are four possible candidates to consider as the underlying model of a given simulation (see Figure 2.1 on the following page). Of course, the ideal case is one where the implemented model and the intended model are identical; however, this ideal will be the exception rather than the rule. As much as the differences between these models can be miti-

[9]There is a technical sense in which the distinction between the compiled code and the machine this code runs on is a false one because the code is no more than physical states within the machine and so technically no different from any other component of the machine itself. Still, there is value in making the distinction and it is hoped that referring to what we will be calling the implemented model as a "fusion" of these two things will help maintain the spirit of the outright physical nature of the machine in the face of making this pragmatic distinction.

[10]As a specific example, consider that memory errors are particularly problematic and hard to test for. Memtest, a standard comprehensive memory testing program, can take over 24 hours to run on a machine with even just a gigabyte of RAM because of all the variations in read-write combinations that it needs to run. Even after it is complete there is no absolute guarantee that there are no errors, only that the program did not detect any errors given the read-write combinations that it attempted while it was running.

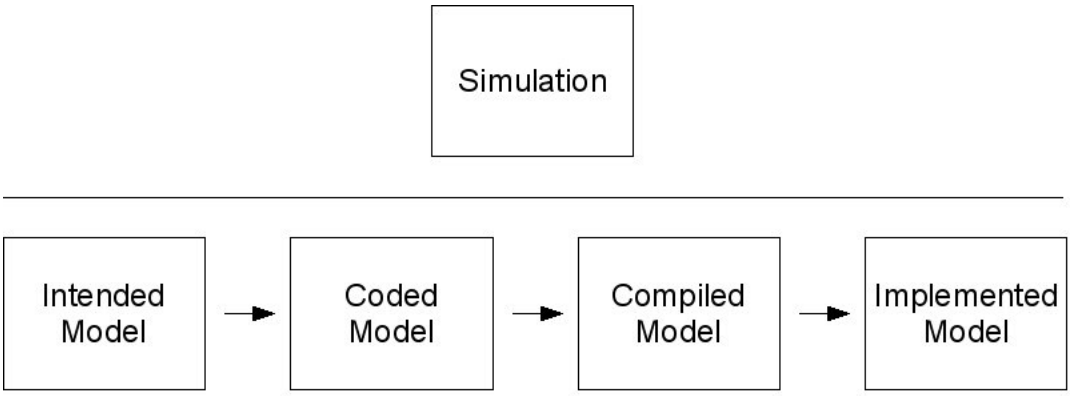


Figure 2.1: Underlying Models

gated through testing and tweaking, such practices cannot eradicate the possibility of significant yet unnoticed differences. This forces simulationists to face the fact that the most appropriate candidate for the model that underlies their simulation is the implemented model since it is what brings about what will (hopefully) be experienced as a simulation.

Accepting the implemented model as the underlying model brings with it the problem that, in all likelihood, the underlying model cannot be specified with the precision we might wish. Running simulations in ensembles across different machines as is done for weather prediction (Smith, 2007) and looking for robustness of phenomena across ranges of initial conditions (Skyrms, 2005) are certainly best practice candidates for living with this reality but they cannot completely override the possibility that perhaps the implemented model is subtly different from what a given experimenter believes it to be—the experimenter's regress looms in the background here just as in many other scientific techniques (Collins and Pinch, 1998, pp. 97-116). In the face of this I suggest the following silver lining: an awareness of this challenge will keep us cautious about claims made based on the results returned, and this caution may keep us from taking things too far too fast by ignoring the human context in which the simulations are produced, even if the results are ultimately "correct".

## 2.6 Two Paths to a Simulation

There are two paths through which a process can achieve the status of a simulation (see Figure 2.2 on the next page). The first is the *indirect* route, as described in the previous section. Representativeness will not necessarily be

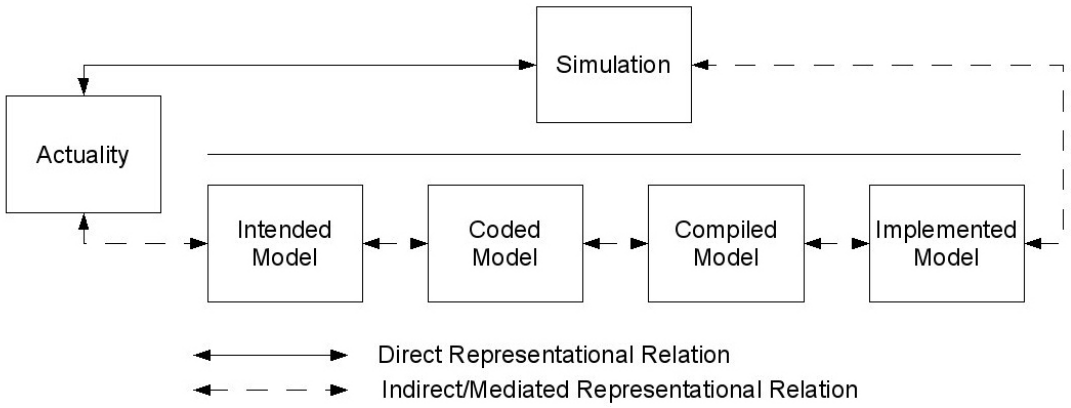


Figure 2.2: Two Paths to a Simulation

transmitted perfectly from intended model to implemented model due to the imperfections in the translation of the described model between the stages. Thus there will be cases where the model degrades under each transformation such that the implemented model is not even remotely representative of the appropriate process in the way the intended model is meant to be. Even worse, we may treat the intended model as if it was ideally representative. Of course, there will be cases where there is either no degradation or it is small enough that the representational relations remain remain sufficiently intact, but whether these are the exception or the rule remains an open question.

The second path to establishing a process as a simulation is via the *direct* representational relationship between the simulation and the process that it is meant to represent. This direct relation is based primarily on appearance: does the simulation seem to capture the relevant behavior (actual or inferred) of the thing in question? While visualizations may be very important components of simulations, there is no particular reason why senses other than sight may not be recruited when judging the success of a process as a simulation. As Paul Humphreys (Humphreys, 2004, pp. 112-114) observes, the form that the output of a computer program takes contributes in an important way to whether the entire process counts as a simulation or not. Both the direct and indirect relationship to the thing are important, but the extent to which each is important will be dictated by context. This is most easily seen by considering the class of simulations known as simulators.

In the case of a flight simulator or similar device, what counts most in determining whether the machine produces a simulation or not is the assessment of the pilots who use the machine. No matter how closely the underlying

model may map onto the behavior of an actual plane, the process will fail to produce a simulation if the final experience of the pilot does not match his or her expectations. Whether or not the simulator computes its outputs in a way that represents the inner workings of a plane and the actual physics of the world does not, in and of itself, constitute a simulation. Certainly, the underlying model should be accurate to some degree, but this is moot unless the end user recognizes the resulting processes of the machine as being sufficiently similar to the real thing. The computer generating the simulated experience may do very little in the way of model crunching at all, simply responding with images in response to various inputs in a way not significantly different from looking up values in a table. The creators of the simulator may not have even had much in the way of a model in mind when they assembled the simulator. Instead they may have observed workings of a real plane and then associated a series of images or other sensory stimuli for identical combinations within the simulator. In such cases there may still be models at work, but they are not as sophisticated as models that might actually *generate* outputs in response to inputs as the simulation moves from state-to-state (e.g. an animation of the movements within our solar system that simply flashes a set of predetermined images on the screen versus an animation that calculates each new screen from the set of previous states).

A case can be made that an index of outputs to inputs could constitute a simulation, whether generated by a sophisticated model or simply built through recording observations, but this is surely a stretch of common understanding—what makes the flight simulator a simulation is that its outputs are deemed appropriately similar to what would be experienced in a real plane. Were the fancy visualization, sound, and physical feedback mechanisms removed so that the computer spits out strings of 1's and 0's then the flight simulator would, for most, cease to be a simulation, becoming a number cruncher at best and useless at worst. Even the select few who might recognize straight binary output as a simulation would do so because they were still able to draw the appropriate similarity relationships between the processes to make a simulation and not because they were able to interpret the output directly.

There are cases though, where drawing the direct representational relation is tenuous because it is not clear just exactly what the actual thing being modeled or simulated is supposed to be. This happens when we are dealing with hypothetical cases such as when supposing that certain assumptions or

never observed states of affairs hold in the world. In cases such as these we cannot compare the outputs to the actuality since there is no actuality, only our ideas of what the actuality might look like. Cases where actual state of affairs are not known and require suppositions also fit into this grouping; examples include military strategies and weather forecasting. In all these cases it is the indirect relationship that matters most because it is our trust that the results of the simulation parallel what would happen were the process modeled empirically that ultimately makes the simulation. This trust is built on the construction and tuning process we either put the simulation through or believe that the simulation has gone through. Once we start to use the simulation we are implying that we are satisfied that everything from our initial intuitions to the implemented model are sufficiently free from errors to provide results that are good enough for the purpose at hand. It is in these cases where the accuracy of the underlying model matters the most since it is the model that we are ultimately trusting to guide us in the right direction.

The interplay between direct and indirect methods of connecting representations suggested here matches nicely with Winsberg's argument that "...simulation is a rich inferential process, and not simply a "number crunching" technique" (Winsberg, 2001, p. S442). Winsberg makes it clear that his reason for including the construction of a simulation under the name "simulation" is to highlight the entire simulation process—including: construction, use, and refinement—as being worthy of its own epistemology within the philosophy of science, an argument that I agree needs to be made. The construction process weaves the indirect and direct representational paths tightly together so that the final product can only be a simulation in the eyes of the constructors and anyone of similar mind.

The three main points raised here regarding simulations—that a simulation is only a simulation for sets of individuals and not necessarily for everyone; that the underlying models are often obscure making it the case that the final simulation may not behave exactly as we expect it to; and that processes become simulations in part via a combination of directly and indirectly arrived at sets of representational relations—are not commonly held or discussed by those concerned with the nature of simulations. I suspect the reason for this is that

relativism remains an unfashionable position within the philosophy of science because on naive interpretations it threatens to make projects meaningless; however, as with conclusions drawn from sociological and anthropological investigations into science (Latour and Woolgar, 1986) such initial worries are typically misplaced. Truths such as the inextricable intertwining of the social with the pursuit of science may create pressure to change our metaphysical stance with respect to science—for example, reflections on the realities of simulations are no doubt useful to both sides of the realism-antirealism debate. Nevertheless this does not mean that we are left without anything upon which we can justify our beliefs and actions. It just means that we cannot have the absolute certainty that we desire. For the majority of situations, given a shared physiology and common academic background there will be a broad consensus on what counts as a simulation or model and just how likely it is that the actual underlying model is sufficiently correct that the overlying simulation can be trusted. Still, consensus is no guarantee of being right and we must be wary not to let our own beliefs blind us to the perspectives of others. If history should have taught philosophers of science anything at all it is that things are rarely quite as right as the majority believes.

# Chapter 3

# 2×2 Games and Their Properties

## 3.1 Importance of 2×2 Games

A game is any situation where two or more players operate under a situation of strategic interdependence. The simplest models of agent interaction that can provide non-trivial multi-agent decision making scenarios are known as 2×2 games.[1] Simpler games result in at least one player not really being a player at all, but more of an observer. While there are certainly cases where people observe the decisions of others, it is rarely the case that such observers are without choices. It is simply that these choices do not matter from the perspective of generating outcomes that are meaningfully different and so simpler 1×1 and 1×2 games can be modeled by 2×2 games. More complicated games and scenarios can also be modeled by 2×2 games although such modeling can result in a loss of information through this simplification. Nevertheless there is reason to believe that human beings often perform similar simplifications when facing choice scenarios, reducing decisions into binary choice scenarios. In this way we present ourselves with simple choices such as "Should I stay or should I go?" rather than the more complicated choice scenario that this question

[1]This name follows the standard method of classifying games which makes explicit the number of players and the total number of options available to each player through numbers separated by multiplication signs. This information is presented in the form A×B×C×..., where each letter represents the total number of options available to a specific player and the total number of players is indiciated by a count of the values separated by the character '×'. The '×' is read as "by" and may be interpreted as a multiplication symbol. The product of all options available to each player is the total number of possible outcomes for the game. For example, a 4×4×4 game has three players, each of whom faces a choice between four options which can be combined into a total of $4^3 = 64$ different outcomes. So, a 2×2 game has two players each of whom can choose between two options for a total of four possible outcomes.

may simplify such as, "Should I stay and do 'w' or stay and do 'x' or stay and do... or go and do 'y' or go and do 'z' or...?". In this way 2×2 games are reasonable models for simplified real-world choices faced by real-world beings.

This simplification, in conjunction with a corresponding rigorous formality, has led to the widespread use of 2×2 to assess and analyze a wide range of human interactions, from nuclear deterrence to business partnerships to romantic relationships. Unfortunately, while there is a heavily explored core set of 2×2 games there is a larger penumbra of these games that have yet to be given much attention. With the notable exception of the Rapoport and Guyer (1966) taxonomy and a recent attempt at a topology by Robinson and Goforth (2005), an intense focus has been given to only a handful of 2×2 games—in particular The Prisoner's Dilemma, Stag Hunt, Battle of the Sexes, and Chicken. This narrow range of attention is problematic insofar as it suggests and promotes the idea that all meaningful human interaction can be captured with only a handful of games, 2×2 or otherwise.

Some of this inattention seems justified as there is a large number of 2×2 games that can uncontroversially be considered trivial. For example, consider the following subset of what might be termed "trivial" or "obvious choice" games:

| | |
|---|---|
| 0, 0 | 0, 0 |
| 0, 0 | 0, 0 |

| | |
|---|---|
| 1, 1 | 0, 0 |
| 0, 0 | 0, 0 |

| | |
|---|---|
| 1, 1 | 1, 0 |
| 0, 1 | 0, 0 |

In the first game what either player does makes no difference to the payoff that either player can receive. Such a state of affairs is ripe for being considered trivial because neither player cares how they or the other player behave. In the second and third cases playing for the top-left outcome is "obvious" to both players, so obvious that such games might also be labeled as trivial. Still, just because something is trivial does not mean that it is irrelevant and so such games should not be dismissed from consideration without further cause. Moreover, despite the attention on the core set of 2×2 games there are other games which are also interesting. Take for example a game called "Red Dress" displayed in Table 3.1.[2]

[2]The name and explanation arose in response to presenting a class I taught with the game shown and asking the students to come up with a state of affairs that would fit the game. I regret that I no longer remember the name of the student who presented it.

| | | Girl | |
|---|---|---|---|
| | | *Red* | |
| | | *Dress* | *Jeans* |
| Boy | *Suit* | 1, 1 | 0, 1 |
| | *Jeans* | 2, 0 | 0, 0 |

Table 3.1: Red Dress

A boy and a girl are going out on a date. They have talked on the phone about what to wear and could not come to an agreement. The girl wants the boy to wear a suit rather than jeans. The boy has a more complicated set of preferences and would really like to balance the comfort of showing off his girlfriend in her red dress with the comfort that jeans provide over his suit. Having his girlfriend wear jeans is the worst outcome from the boy's perspective since he is applying a double standard and is worried about her looking "trashy". Choosing to wear jeans is a weakly dominant strategy for the boy because if his girlfriend is going to wear her dress then he can improve the outcome and if she is going to wear jeans then he does no worse than if he had worn the suit. However the existence of this dominant strategy fails to provide a robust solution to the game. Why? The girl has similar motives for having the boy wear his suit and is in a position to make a credible threat in any single iteration of the game and is able to punish the boy in any iterative version. This threat exists because while the girl is indifferent to what she wears she is not indifferent to what the boy wears, giving her the leverage of threatening to wear jeans should the boy not agree in advance to wear his suit. However, should such an agreement be made the boy then has reason to wear his jeans as a weakly dominant strategy again and so begins a cycle not unlike that experienced in reasoning about the Prisoner's Dilemma. Yes, mixed strategies are often suggested as a way to solve such games, but it can take a long time for each player to discover the appropriate probabilities to assign each outcome and mixed strategies do not, at least in this case, take into account the credible threat available to the girl as a consequence of her indifference about what she wears.

Given the overuse of certain 2×2 games and the existence of alternatives that might either act as better models in certain interactions or which might have interesting properties in their own right, a natural question to ask is how

many 2×2 games are there and what they are. Unfortunately, the literature on 2×2 games is unclear on just how many 2×2 games there are: the original taxonomy by Rapoport and Guyer (1966) counts 78, a paper byWang and Yang (2003) claims that there are 6561, and a recent book dedicated to producing a topology of 2×2 games by Robinson and Goforth (2005) claims that there are 144. These differences are not simply a matter of restricting discussion to some particularly interesting or important subset of 2×2 games, although this certainly accounts for part of the difference. What is the main source of these wildly different values? An unspoken disagreement among these authors (and others) regarding *exactly what counts as a game*.

What follows is an account of this disagreement and a suggested "correction". The result of this is that there 726 meaningfully different 2×2 games. Given the large number of these games and the exhibited small frequency with which we are likely exposed to the games that traditionally receive the most attention it is suggested that the set of games considered to be of interest be broadened.

## 3.2 Properly Counting 2×2 Games

The core of the disagreement about just how many 2×2 games there are arises from a decision on the part of some authors to count what amount to different representations of the same game as different games, leading to anywhere from double to octuple counting. For example, each of the games in Table 3.2 is a Prisoner's Dilemma.[3]

| | |
|---|---|
| 2, 2 | 0, 3 |
| 3, 0 | 1, 1 |

(a)

| | |
|---|---|
| 3, 0 | 1, 1 |
| 2, 2 | 0, 3 |

(b)

| | |
|---|---|
| 0, 3 | 2, 2 |
| 1, 1 | 3, 0 |

(c)

| | |
|---|---|
| 1, 1 | 3, 0 |
| 0, 3 | 2, 2 |

(d)

Table 3.2: Four Ways to Represent the Prisoner's Dilemma in Strategic Form

Prisoner's Dilemma (a) is perhaps the most common way to represent the game, (b) represents a swapping of the rows from the representation shown in

[3]There is some inconsistency in the literature regarding what counts as a prisoner's dilemma. The structure given here is the one used by Rapoport and Guyer Rapoport and Guyer (1966) and has become the standard way to understand this 2×2 game. Luce and Raiffa (1989) use a different payoff structure and hence present an entirely different game as a PD.

(a), (c) represents a swapping of the columns from the representation shown in (a), and (d) represents a swapping of both rows and columns from the representation shown in (a). Despite the change of appearance the game being played remains the same for all ideally rational agents. Wang and Yang (2003) count each of these presentations as a different game while both Rapoport and Guyer (1966) and Robinson and Goforth (2005) agree that each representation is just that, a *representation* of the same game. This agreement between the earlier work of Rapoport and Guyer and the more recent work of Robinson and Goforth is not perfect, extending only to games that, like the Prisoner's Dilemma, have symmetry across the payoffs available to the players. Such symmetric games have the exact same payoff matrix when subject to R&G reflections.[4]

Table 3.3 provides eight ways of representing Red Dress. The lack of symmetry between the players allows for the full spectrum of ideas about what constitutes the game to be examined.

For Rapoport and Guyer each of the normal form game matrices in this table are simply alternative ways of representing the same game. With matrix (a) as the base case, matrices (b) through (c) can be created through row or column swaps (e.g. (b) can be attained from (a) simply by swapping the second column of outcomes with the first). The matrices (e) through (h) are the R&G reflections of the game immediately above (e.g. (e) is the R&G reflection of (a)). Robinson and Goforth are of a different opinion regarding what is captured in Table 3.3: rather than eight representations of one game they see two games, each with four representations. Why this discrepancy?

The discrepancy arises in part because Robinson and Goforth are interested in building a topology of games that will provide a tool for game theorists with a value similar to what the periodic table of the elements provides to chemists. By explicitly ignoring R&G reflections as alternative representations of the same game they are able to create a topological description of 2×2 games

[4]R&G reflections are "Rapoport and Guyer reflections" and are the equivalent of swapping the row player's payoffs with the column player's payoffs. If the same payoff arrangement is returned following the swap it is an indication that an ideally rational player whose von Neumann and Morgenstern utilities are fully captured by the current utility assignments for both players, would have no preference to choose to be either the row or the column player. The name "R&G reflection" is used by Robinson and Goforth (2005) and is a reference to Rapoport and Guyer (1966) who made use of the reflection to eliminate the number of payoff matrix combinations that they needed count when building their original taxonomy.

| | |
|---|---|
| 1, 1 | 0, 1 |
| 2, 0 | 0, 0 |

(a)

| | |
|---|---|
| 0, 1 | 1, 1 |
| 0, 0 | 2, 0 |

(b)

| | |
|---|---|
| 2, 0 | 0, 0 |
| 1, 1 | 0, 1 |

(c)

| | |
|---|---|
| 0, 0 | 2, 0 |
| 0, 1 | 1, 1 |

(d)

| | |
|---|---|
| 1, 1 | 0, 2 |
| 1, 0 | 0, 0 |

(e)

| | |
|---|---|
| 1, 0 | 0, 0 |
| 1, 1 | 0, 2 |

(f)

| | |
|---|---|
| 0, 2 | 1, 1 |
| 0, 0 | 1, 0 |

(g)

| | |
|---|---|
| 0, 0 | 1, 0 |
| 0, 2 | 1, 1 |

(h)

Table 3.3: Eight Ways to Represent Red Dress in Strategic Form

that illustrates a clear set of relationships between each game and that is both beautiful and useful. The problem is not their admirable topology but their insistence in referring to the products of R&G reflections as *different* games. Their reasoning for this position is that they interpret R&G reflections as involving the reassigning of roles, which:

> . . . is not conventional geometric or group-theoretic reflection. "Rapoport and Guyer reflections" (R&G reflections) are behaviourally equivalent if players facing the same payoff structure always behave the same way. In other words, reflections are indistinguishable if players are indistinguishable. (Robinson and Goforth, 2005, p. 17)

The strongly implied premise in their reasoning is that many possible players, especially human players, are clearly *not* indistinguishable.[5] Consequently, it seems that we should consider R&G reflections as different games; otherwise we will be failing to account for all the 2×2 games that non-indistinguishable beings might play.

Another reason for counting R&G reflections as different games is that, under their approach to producing the topology, including these reflections produces a stack of games consisting of four layers of games, six games wide by six games deep. This block of games and the specific ordering they provide allows for numerous relationships between the 2×2 games to be clearly

[5]As an example of how players fail to be indistinguishable consider that it is well known that human beings are not purely rational and so changes that should make no difference whatsoever to how the game is played, such as rearranging rows and columns or changing the names used to refer to various actions can influence human behaviour, and thus the play of games, significantly. See framing effects in Kahneman and Tversky (2000), Gilovich et al. (2002), and Dawson, Gilovich, and Regan (Dawson et al.).

illustrated in a way similar to how the periodic table of the elements allows elements with similar properties to be grouped and more easily understood.

Despite the pragmatic and aesthetic usefulness of counting R&G reflections and their non-conventional nature with regard to reflection in general, counting reflections of games as entirely new games is a mistake. The source of the mistake made by Robinson and Goforth rests in a confusion concerning what constitutes a game and how this is importantly separate from the way in which this game is represented.

John Von Neumann and Oskar Morgenstern, define games succinctly, "[A] *game* is simply the totality of rules which describe it" (von Neumann and Morgenstern, 1967, p.49).[6] More recent authoritative works include Luce and Raiffa (1989) and Fudenberg and Tirole (1991); the definitions these authors provide for normal (strategic) form games can be found in table 3.4.[7]

**Fudenberg and Tirole (1991)**

A game in strategic (or normal) form has three elements:

1. the set of players $i \in I$, which we take to be the finite set $\{1, 2, \ldots, I\}$,
2. the *pure-strategy space* $S_i$ for each player $i$, and
3. *payoff functions* $u_i$ that give player $i$'s von Neumann-Morgenstern utility $u_i(s)$ for each profile $s = (s_1, \ldots, s_I)$ of strategies.

**Luce and Raiffa (1989)**

. . . a *game in normal form* consists of:

1. The set of $n$ players.
2. $n$ sets of pure strategies $S_i$, one for each player.
3. $n$ linear payoff functions $M_i$, one for each player, whose values depend upon the strategy choices of all the players.

Table 3.4: Comparing Game Definitions

None of these definitions say anything at all about the method of representation that is to be used when presenting any game in a less abstract way. The

[6] Of course, after putting it succinctly they go to great lengths to spell out an axiomatic definition of games spanning roughly twenty pages and ending with a complete list of all axioms. See section 10 of von Neumann and Morgenstern (1967) for this list.

[7] The strategic/normal form definitions are used simply because they are simpler to write down. It should be recalled by the reader that well-formed normal form representations are completely translatable into an extensive form representation of the same game and vice-versa.

reason for this silence is that the representation is irrelevant to the constitution of a game. Robinson and Goforth go wrong when they claim, " 'Rapoport and Guyer reflections'... are behaviourally equivalent if players facing the same payoff structure always behave the same way [and they do not]". This position is wrong because R&G reflections are not about changing the options or outcomes available to a player but about changing the way that the current options and outcomes are represented. The representations of Red Dress given in table 3.3 are all *representations* of the same game because the same options are given to the boy and the girl regardless of whether they are displayed as either the row or column player and they maintain their preferences across the outcome regardless of presentation. Put another way, making the girl the row player in the representations of Red Dress given in table 3.1 does not somehow make her a boy; it is the representation of her preferences over the outcomes that changes, not her actual preferences. Robinson and Goforth's claim that there are 144 2×2 games is thus mistaken because they double count every 2×2 game with a non-symmetric payoff structure.

Wang and Yang (2003) make the same mistake, but on a much larger scale, arguing that there are $81^2 = 6561$ different 2×2 games. They begin their proof by showing that for each player there are four cases to consider for how a preference ordering may be applied across the four cells in a normal form representation of a 2×2 game: no outcomes are equally preferred; two and only two cells are equally preferred; two pairs of equally preferred cells or a single set of three equally preferred cells; or all cells being valued equally. From this they determine that there are 24, 36, 20, and 1 (respectively) possible ways of arranging the preferences for a player over the cells. The error comes into play when they reason that because there are 81 possible ways of distributing the preferences of one player over the four cells in a normal form distribution that each application of a payoff ordering from one player will produce a meaningfully different arrangement when combined with each payoff ordering of the other player. Thus, there are $81 \times 81 = 81^2 = 6561$ different 2×2 games. In making this jump they implicitly assume that different arrangements of preferences within the cells of a normal form distribution are meaningfully distinct under the definitions of a game given earlier, but this is not the case.

To see that this is a mistake consider again the eight representations of Red Dress given in table 3.3. According to Wang and Yang each of these

representations is a *different* game because each shows a different arrangement of preferences over the cells within each matrix. While it is true that each is a different arrangement of preferences across cells they fail to recognize that different arrangements of *preferences over cells* in a matrix can produce the same arrangement of *preferences over outcomes*. Their claim thus amounts to saying that the orientation of the playing surface of a board game such as Monopoly or Risk changes the game being played or that having every player in a game of poker shift one seat to the right before dealing the cards would somehow change the game.

So, if neither Wang and Yang nor Robinson and Goforth assert the correct number of 2×2 games, then just how many are there? Rapoport and Guyer's original 1966 taxonomy was restricted to only include those games that could be constructed when:

1. payoffs are taken into account on an ordinal scale,
2. transformations resulting in equivalent games are ignored, and
3. the payoff scale is strict (i.e. no items on the scale may be judged as equivalent).

Given these restrictions they correctly identified 78 meaningfully different 2×2 games. However, their restriction to strict payoff scales, while useful in constraining the number of games to be considered within their taxonomy, leaves out a significant number of 2×2 games; namely those within which at least one of the players is indifferent across at least one pair of outcomes. Removing the strict payoff scale restriction results in a significant expansion of the number of 2×2 games from 78 to 726.

Two methods of proof for this claim are offered. The first breaks the 2×2 games down into fifteen cases depending on the number of equivalent payoffs that are offered to each player and is provided in full in Appendix A. The second is carried out by a computer program explicitly for this purpose. This method tries every possible method of assigning each of the payoffs across the four outcomes one of four possible values, testing each result to see if it is the same as any previously identified game and then throwing away the duplicates. The code for this brute-force approach can be seen in Appendix B. This code also produces a full taxonomy of all 726 2×2 games which may be viewed in Appendix C.

## 3.3 2×2 Game Frequency

So, given that there are 726 2×2 games, which ones matter? Or, put another way, which ones do we expect to come up most frequently in our day-to-day existence so that we can focus any future analysis on those games? While the exact answer to this question will vary from person to person and context to context, there are a few major factors that will contribute to any such analysis, as follows:

1. The preference structure of human beings. In particular, the number of meaningfully different gradations of preference held.

2. The relative importance of small changes in the value of outcomes as determined by the environment.

3. The distribution of payoffs across outcomes as determined by the interaction of the environment and an individual's preferences.

The first of these is important simply because the more gradations a person has stands to change both the number and sorts of games that they might play. To see this in the most trivial sense note that the person who has only one level of preference and is thus indifferent across all things can only play one game (Game 1, Appendix C). Adding more gradations increases the number of possible 2×2 games that he or she might face until he or she reaches four gradations at which point all 726 games become available, providing that the other player is similarly endowed.

The environment stands to play an important role in just what games are important in a number of ways. The first is that an environment where resources are plentiful and preferences are easily satisfied will stand to reduce the number of conflicts that are likely to arise. Not only will there be less reason for conflict in the first place (why would anyone pick a fight over food when they can simply pick-up some food?) but those that do arise will have a very different character than they would otherwise as a result of the interaction between preferences and the environment (the interacting players are likely to have different preferences than they would in a harsher environment). The reverse stands to be true in environments where survival becomes increasingly difficult.

Allowing that human psychology is constructed such that, for the most part, the preferences that people have correspond to the survival value of the

payoffs associated with the outcomes of events and that there is no reason why the environment broadly construed should favour providing one payoff over another over each outcome of the game it can be seen as a reasonable assumption that the relative frequency with which each game is encountered is a function of the random distribution of payoff over some range of values. With this underlying assumption in place it becomes possible to calculate the frequency with which each of the 726 2×2 games might be encountered for a given range of values.

This calculation was carried out for a total number of possible payoffs ranging from 4 to 25 via a computer program written specifically for the task. This program attempts every payoff combination possible with the number of values allowed at each stage of the calculation. Each combination is checked to see which of the 726 possible 2×2 games it is a representation of and then a counter associated with that game is incremented. When all the combinations have been tried for a given number of possible values a payoff can take then the number of times each game was encountered was divided by the total number of combinations possible. The resulting probabilities were tracked over the entire range of values being explored.[8]

The first important insight from looking at the results of this process was that there were clearly 15 different distributions of frequency across the 726 2×2 games. While it was initially suspected that these 15 groups would overlap perfectly with the 15 groups used to count the games in the first place (See Appendix C) this did not turn out to be the case. Instead the games within the count groups split over the probability groups in various ways. The exact details of this combined distribution are shown in figure 3.1. The reason for the different frequencies of occurrence within each count group is the existence of symmetries within certain games that are not shared with neighbours in the count group. The existence of symmetries reduces the likelihood of a game occurring because it is no longer simply a matter of a certain collection of payoffs being on the board but a matter of their specific arrangement. These symmetries amount to the presentation of the game remaining unchanged through

[8]This brute force approach was necessary to gain some initial insight into this problem, but its limitations became apparent quickly—at 25 possible values the number of combinations to be checked and tracked was $25^8 \approx 1.5x10^11$, over 150 billion possibilities. With this insight into the frequency of games under an increasing number of possible payoff values it became possible to construct a formula to achieve this result more efficiently. This formula will be shared shortly.

| Counting Group \ Probability Group | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | Total |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | 1 | | | | | | | | | | | | | | | *1* |
| **2** | | 2 | | | | | | | | | | | | | | *2* |
| **3** | | | | 3 | | | | | | | | | | | | *3* |
| **4** | | | | | | | 9 | | | | | | | | | *9* |
| **5** | | | | | 6 | | | | | | | | | | | *6* |
| **6** | | | | | | 4 | | | | 6 | | | | | | *10* |
| **7** | | | | | | | | | | 12 | | | | | | *12* |
| **8** | | | | | | | | | | | | | 72 | | | *72* |
| **9** | | | | | | | | | | | | 48 | | | | *48* |
| **10** | | | 2 | | 4 | | | | | 2 | | | | | | *8* |
| **11** | | | | | | | | | | | | | 54 | | | *54* |
| **12** | | | | | | | | | | | | 36 | | | | *36* |
| **13** | | | | | | | | | 18 | | | | | 153 | | *171* |
| **14** | | | | | | | | | | | | | | | 216 | *216* |
| **15** | | | | | | | | 12 | | | 66 | | | | | *78* |
| Total | *1* | *2* | *2* | *3* | *10* | *4* | *9* | *12* | *18* | *20* | *66* | *84* | *126* | *153* | *216* | ***726*** |

Figure 3.1: Count of Games Shared Between Count and Probability Groups

any of the various transformations that are available with a normal form game (row swap, column swap, and player swap).

The behaviour of the different probability groups from four possible values for a payoff to take to twenty-five possible values are presented in figures 3.2 and 3.3. The first of these figures shows the behaviour of single members within each of the 15 probability groups. This graph shows that as the number of possible values that a payoff can take increases there are are only two probability groups that have members who occur with increasing frequency, groups 14 and 15. These two probability groups belong to count group 15 which is made up of those 2×2 games where neither player is indifferent between any combination of outcomes. Probability group 15 captures the games which have no symmetry and group 14 captures those with symmetry. Given that as the number of possible values a payoff can take increases the likelihood of two payoffs taking on the same value over a random distribution decreases, this result should not be surprising. What is surprising is that when the members of each probability grouping are added together and the results of this aggregation plotted, as in figure 3.3, groups 14 and 15 become much less significant when the number of values that a payoff may take are relatively small. To see this more clearly note that the *individual members* of probability group 15 become the dominant when as few as 6 possible values for a payoff to take are allowed, but that the *group as a whole* does not become dominant until 16

possible values. The situation is even more striking for probability group 14. Individually, its members become the second most dominant by 12 possible values for a payoff, but as a group it is barely able to make third place by 25 possible values. Most surprising of all, probability group 14 is where the the majority of the most famous 2×2 games can be found: The Prisoners' Dilemma, Stag Hunt, Chicken, Battle of the Sexes, etc.

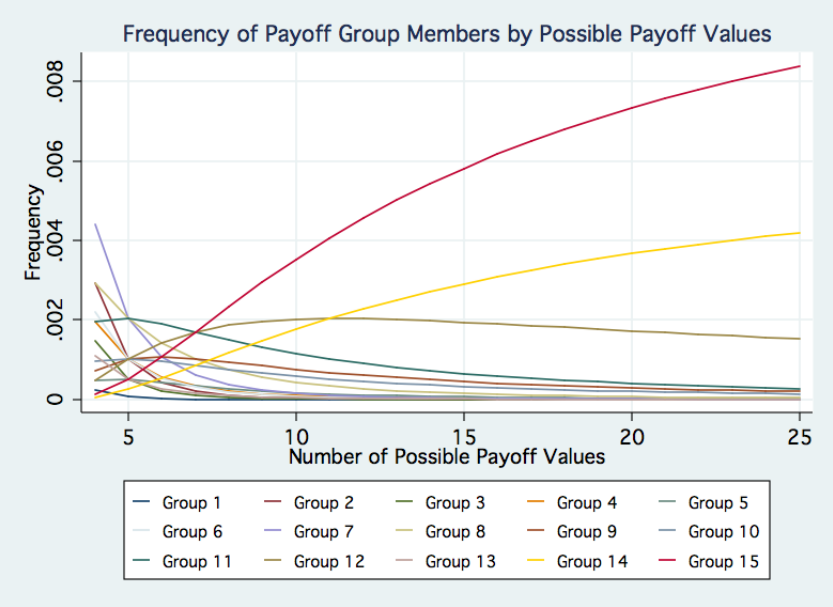


Figure 3.2: Frequency of Single Members with the 2x2 Probability Groups

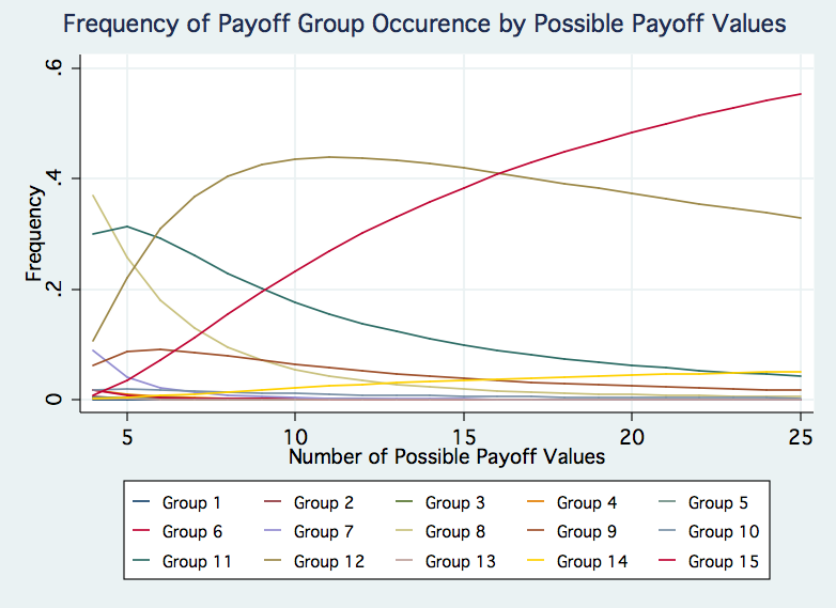


Figure 3.3: Frequency of 2x2 Probability Groups

The probability group that manages to beat out this grouping of famous games for second place, at least initially, is group 12. Looking at figure 3.1 it can be seen that this probability group is composed of all the games from count groups 9 and 12. Count group 9 is made up of games where one player values three of the four outcomes equally and the fourth either more or less than these while the second player values every outcome differently. Count group 12 is made up of games where one player's preferences across outcomes may be grouped into two pairs where the members within each pair are valued equally while the second player values every outcome differently. Combined, these two count groups have $48 + 36 = 92$ members, dwarfing the 12 members of probability group 14. This, in combination with simple probability account for the dominance of probability group 12 over group 14 despite any intuitions that may be had to the contrary.

Eventually, probability groups 15 and 14 win out over all other probability groups. This is most easily seen by noting that as the number of possible values that the payoffs can take increases the likelihood of the same value being assigned to two payoffs drops to zero. More formally, the probability of any 2×2 game being generated is given by the formula[9]:

$$P(x) = \frac{r\binom{x}{p}\binom{x}{q}}{x^8}$$

where

- $x$ is the number of integer values that each payoff might take on from 4 $\rightarrow \infty$
- $r$ is a measure of the symmetry within the game in question. There are four possible values that $r$ might take on. A value of 1 means that the game is symmetric through all possible changes in presentation. Only the game where all the payoffs to each player are equivalent receives a value of 1. A value of 2 means that there is strong symmetry that does not manage to be perfect. Such games maintain their presentation in both the case of an R&G reflection (player swap) and in the case of a

[9]I believe that this formula is generalizable such that it will provide the probability of *any* game of any class of games being randomly generated when the corresponding information regarding the relevant structural components is provided. It is my hope that this will be an important step towards being able to count the games of any class without making use of the clumsy method shared in the appendices. It may also aid in developing a non-arbitrary system for naming games, but this is likely a long way off.

row swap followed by a column swap. A value of 4 means that there is only one form of symmetry and the presentation cannot be maintained through more than one kind of change. A value of 8 means that there is no symmetry at all.

- $p$ is the number of different values that the row player is allowed to have appear as payoffs across the outcomes. $p$ must be an integer between 0 and 4.

- $q$ is the number of different values that the row player is allowed to have appear as payoffs across the outcomes. $q$ must be an integer between 0 and 4.

While the binomial nature of this formula in conjunction with the fact that it only takes integer values removes the possibility of simply taking the limit to find out which games survive at the highest values, it is possible to put the formula into a high-end math program that is able to perform the necessary calculation at increasingly larger values of $x$. At $x = 1000000$ Maple 13 approximates the probability of generating a game from probability group 15 ($r = 8$,$p = q = 4$) at 0.92 and the probability of generating a game from probability group 14 ($r = 4$,$p = q = 4$) at 0.08. All other groups have their probability of occurring go to zero. This happens because they require indifference of a player across a set of outcomes to exist and this becomes increasingly unlikely as the range of values randomly assigned to each payoff expands.

Given the attention that has been bestowed upon the games of probability group 14—and only a small portion of these 12 games at that—it should be surprising that they compose only 8 percent of the possible games that might be produced as the number of possible preferences approaches the limit. Granted, it may well be the case that they are more frequently encountered in the real world due to anything from a psychological disposition towards forcing symmetries to environmental structures that are unaccounted for in this simplistic treatment, but if such things exist surely it is the responsibility of those studying such games to carry the burden of proof regarding their existence.

Practically speaking, it is not likely the case that anyone has preferences along a smooth continuum. It is much more likely that people operate on a day-to-day basis with some much smaller range given limited perceptive

faculties and the limits of time. It would seem to be the case that there are even social conventions about being too picky. If there is any doubt about this try individually inspecting and trying on every item on the rack the next time you go shopping for clothes, *even if they are the same size*. It is on this practical level that we interact with most people most of the time and thus the area on which the majority of our focus concerning games should be placed.

# Chapter 4

# A Rational Agent Model for Simulation

Simulations require underlying models during their construction and for their continued existence.[1] Such models are also necessary for a thorough analysis of the results of any simulation. In the next two chapters simulations are used to investigate two questions about the nature of rational behaviour under various conditions. In both cases the same underlying model is used and the necessary data required to answer each question is drawn from the same data set produced by these simulations. The details of the underlying model constructed to aid in answering the questions and the motivations for building it are provided in this chapter.

## 4.1 Why Build Another Model?

There are many agent-based models being used to explore elements of human social systems, moral development, economic systems, and the like. It seems as though anyone in the business has their own custom-built agent-based simulation meant to explore a specific series of questions. This specialization leads to a great deal of progress on specific questions, but the drawback is a lack of cumulative progress in the agent-based approach when taken as a whole (Namatame et al., 2002). The degree to which it is believed that there is a lack of cumulative progress in the modelling of social systems is going to depend

[1] For an at length discussion about the relationship between simulations and models as well as a discussion of what should be understood to count as an *underlying* model see Chapter 2.

a great deal on what is seen to constitute a new model. In the most granular sense even the smallest of differences between two models, perhaps simply the inclusion of an obscure exogenous variable in one that does not appear in the other, is enough to make them distinct. Given such a granular perspective on models and modelling it should come as no surprise that there is a proliferation of models. This is not the only way to understand "model". Often what is meant is some broad class made up of those models that share core commonalities in their approach. This common approach can be referred to as a framework and each variation within the framework produces a new model. It is in this way of grouping models together via their common structures that a taxonomy of models might be constructed.

Given this state of affairs some justification is required to explain why a new model has been built when so many models are already available for use. Four reasons stand in defense of building a new model rather than directly co-opting or modifying an existing model:

1. There was no known specific model suitable to borrow.
2. There was no known general model suitable to build upon.
3. There is a great deal of value inherent in building a model from scratch.
4. The model chosen is intended to play different roles than most agent-based models.

Models are built with the intent of exploring specific phenomena or answering specific questions. Even slightly different questions require different models. Specialization gets results and when what matters is answering the question at hand, specialization is the order of the day. The result is that most models are not ideally suited to being applied to questions that they were not built to answer and this may be true even if the questions are closely related. Granted, an off-the-shelf model might be easily modified to accommodate a different question, but this is not necessarily the case; most models are built to utilize the most efficient and effective methods available. Such utilitarian construction goes beyond simple model parameters, including the capabilities of the machines that the resulting simulation will be carried out on, the software installed on these machines, and the programming and data analysis capabilities of the research team. These issues are particularly salient for agent-based

models because the questions that motivate agent-based approaches to solutions are inherently complicated, so much so that traditional approaches are not suitable. Such traditional approaches include inference, mathematical modelling, logical analysis, experimentation, personal anecdotes, and field research. Each of these fails to answer the relevant questions because the questions have characteristics that demand more than these approaches can provide.

Roughly put, the question motivating the models that exist already amounts to one of, "How did we come to be the way we are?" or "How are we?" The question that I am interested in answering is "How could we be?" or "What could we have become?"[2] The difference is subtle but important, amounting to a need for an increase in the range of possible variability that the agents in the new model will require over those of alternatives.

There is also a lack of agent-based models that are general enough that they might be seen as being pre-specialized blanks useful for exploring questions of a specific nature. I suspect that this is also the result of the need for specialized tuning before a model will be useful. This specialization happens at such an early stage of the entire process that building a generalized template that does not presuppose a specific set of questions is almost impossible.

Perhaps the most significant reason for building a new model—it would have trumped any model that proved to be a less than perfect match for the requirements—is the inherent value of building the model oneself. Winsberg (2001, p. S442) has stated that "...computer modelling is a rich inferential process." In addition to the inferences available, a great breadth of skills and knowledge is required to build an agent-based computer model and interpret the results of the simulations ultimately run, including math, logic, programming languages and conventions, and hardware and software limitations. Both the inferences and skills available through the process may be acquired in the fullest sense only when a researcher has actually been an integral part of building the model and simulation. As Winsberg emphasizes, simulation is a process from beginning to end and much of the insight is to be had from the construc-

[2]This shift from "How are we?" to "How could we be?" is an important change in the question motivating social behaviour research. I cannot claim responsibility for this shift (I was inspired to explore it by Ken Binmore's *Game Theory and the Social Contract*) but it seems to me to be on par with the shift made by Hobbes when he shifted the question of politics from idealizations in a heavenly sphere to pragmatic observations in a human realm (Lilla and CBC Radio (2008)). Whether this comparison is warranted or not will be revealed in time.

tion of the simulation itself. A full awareness of how small changes in the order of operations or variable assignments affect the performance of the simulation can only be had through hands-on toying with the model. This familiarity is similar to what scientists experience when constructing their own experimental apparati and instruments (Collins and Pinch, 1998).

I also believe that my hand was forced into creating a new model because my intent for the use of the model differs from what models are traditionally intended to do. My intent is different in two ways, both of which originate in my conception of the purposes of this project. The first amounts to an attempt to gain experience dealing with complexity while the second is akin to the shift from explanation to prediction.

The traditional approach to understanding social structure might be seen as two sets of researchers on either side of a large canyon trying to overcome the great divide. On one side are psychologists, anthropologists, and sociologists who focus first on examining and reporting the structure of the world as it is and second on devising generalized models that describe these structures. On the other side are philosophers and political theorists who first examine small cases and devise generalized models from their discoveries and subsequently look to apply their models to the structure of the broader world. Each side continually throws ideas to the other. There are models, observations, and experiences that span the gap as tiny threads but many of these either fail to reach the other side, slip from their grasp, or are not sufficiently robust to bear the weight of examination. Both sides have started building bridges, but neither side can reach very far across the chasm without additional support and so these bridge beginnings have amounted to observation decks, useful for peering into the abyss below. The abyss is known to have a bottom, it is just a very long way down and a complicated, messy environment to work in, filled with murky waters. If the chasm is to be spanned by more than threads the bridge must be supported by piers secured to the bottom. There is no other way around it. People must go down into the abyss and build these piers by whatever means necessary (even if it means manually labouring in caissons surrounded by dangerous currents) and in doing so tame the floor of the abyss and raise the piers (Sawyer, 2003, p. 263).

I see my model as an attempt to push into the gap between observation and theory, a wading into complexity and the complications this act brings with it. This, of course, means increased computational time, larger data sets, tenuous

and ephemeral relations between objects, and the lack of satisfaction that comes from building something that is neither simple nor obviously real. The model is thus as much an exploration of the difficulties of building complicated models as it is a model suitable for investigating possible social worlds.

This final reason for building a new model ties directly into the first two reasons because while many agent-based models/simulations force agents into a specific relational geography this new model does not. This relational geography is either presented via a simulated physical geography where the agents are able to associate only with those agents that they are physically close to, such as in adjacent squares on a grid (e.g. Skyrms (2005); Epstein and Axtell (1996); Schelling (1971)), or through predefined relationship channels which allow agents to associate with only prespecified agents such as networks that place each member on the edge of a ring and only allow interaction with a neighbour (Young, 1998). This approach is excellent for exploring simple predetermined relationship structures, either actual or possible, and that is what they are typically used for. The challenge comes in being able to map these simplified models onto behaviours of larger and more complicated systems. This is not to imply that this cannot be done without success, only to say that what simple models gain in explanatory power they lose in applicability to the real world system that they are intended to represent (Morgan and Morrison, 1999). What these simpler models are not necessarily good at is acting as tools for the exploration of large scale societies where the relationships formed by members are more fluid and numerous, and consequently much more complicated.

The direct consequence of these motivations towards model building is that the proliferation of models will continue for the foreseeable future. This is not perhaps as detrimental as is felt by some (Terano et al., 2003). Given the newness of agent-based models and the use of computer simulations in general it is quite possible that what is being witnessed here is a natural stage of development and exploration of a scientific method. Whether or not it constitutes anything in the way of a paradigm shift of the sort introduced by Kuhn (1970) remains an open question. More important than this, the field remains relatively immune to criticism on the grounds of group-think or scientific monoculture because such challenges are precluded by the variety of methods being employed by many researchers (Sunstein, 2003).

## 4.2 The Model

The model developed for simulation has three main parts: the agents that represent rational decision making beings, the games that represent the combination of social and situation specific environmental frameworks within which the agents interact, and the broader environment. Each will be discussed in turn.

### 4.2.1 The Agents

The most important core component of the model is the template used in representing rational decision making beings. Each agent amounts to a complete description of a set of responses to all the possible situations that they might face, given the environment models in which they operate. Note that there are actually two different types of agents, each representing a different choice procedure. Since both agent types share the same general architecture the details of their differences will temporarily be set aside.

At the core of each agent type is a base structure that enables the members of each type to perform three specific actions: they can play any of the 726 possible 2×2 games, they are capable of assessing the status of their relationship with other agents in a non-trivial way, and they are able to respond to the actions of others in conjunction with the outcomes of the games that they play with each other.

Designing agents capable of playing any 2×2 game allows them to be used to investigate a wide variety of scenarios. Such scenarios can be either strictly theoretical/hypothetical or more realistic representations of the social environment faced by actual decision making beings that exploring only the more traditional 2×2 games would ignore.

The way that agents are given the capacity to play any game is relatively simple. When each agent is created it is assigned a complete list of responses to all the possible scenarios that it might face (these assignments are random for the initial agents of a population and inherited with 50% probability from parents during the reproduction of new agents). Since each scenario is basically a choice within a 2×2 game an agent's behaviour when faced with that game can be represented by a single bit. Setting the bit to 1 ("on") or to 0 ("off") indicates which choice the agent will make. If there are variations to the environment in which the game is played, perhaps due to past experiences

with a player, then each of these variations requires a different bit.

The second important feature of agents within the model is that they are capable of assessing relationships that exist between them and other agents in a non-trivial way. The ability of people to assess various types of relationships that they might have with others was important to include because such assessments can lead to significant changes in behaviour. Most people simply do not provide the same level of help and support to every person in the world and the source of this discriminatory behaviour is what or how they think of the other person. To accommodate this reality every agent is capable of assigning all the agents with whom they interact into one of three categories. These categories can be roughly understood as 'Friend', 'Neutral', or 'Enemy', but it is important to understand that how agents assign other agents to these categories may not map onto our understanding of these terms. Agents simply have three categories and treat all agents within each category the same. Whether they are 'nice' to all the agents in the 'Enemy' category and 'mean' to everyone in the 'Friend' category or vice versa is something that is not explicitly stipulated in the program. Further, being nice or mean is not necessarily implemented across all possible situations within each category. It is much more likely the case that various actions that we as observers might consider to be "nice" or "mean" are distributed evenly with fifty-fifty randomization across each category during the start of each trial and so any attempt to anthropomorphize how agents like each other should be strongly resisted.

In addition to the basic relationship assessment that results from noting that people interact differently with other people depending on how they like them, I also take it as obvious that people further adjust their behaviour towards others by taking into account how they believe other people see them. People do not often treat another person the same way if they believe that the other person hates them as opposed to believing that the other person likes them or at least has a more neutral stance towards them. As a mirror to the three stances that each agent might take towards another agent there are three stances that each agent supposes that the other agent might take towards them. Taking into account what has been said earlier about resisting the attraction of overly anthropomorphizing these agents the three stances can be understood as mapping onto: "I think that you think that I am a Friend", "I think that you think that I am a Neutral Party", and "I think

that you think that I am an Enemy". These three stances in combination with the three listed immediately prior result in nine different possible relationship types that each player can assess whenever a choice is to be made. Each 2×2 game that a player is capable of playing must have nine bits to describe how that game is to be played.

Having each agent capable of assessing both how they perceive the relationship and how they perceive how the other agent perceives the relationship simulates empathetic preferences or Theory of Mind (Binmore (1994); Dunbar (1996); Cummins (2000)), roughly the ability to take into account how other people see a given situation and to include that in one's own decision making (of course, agents always see the other agent's perspective only from within their own perspective). This capacity is focused on in more detail in Chapter 6.

The final important defining characteristic of the agents within my model is their ability to respond to the outcomes brought about by their interaction with other agents and the mechanism that creates this ability. The ability to respond to outcomes simply means that after a game has been played an agent looks at what both he or she and the other player actually received versus what was available to be received. Based on this assessment the agent is capable of reassigning both the stance that he or she takes toward the other agent across the three available categories and the stance that he or she sees the agent he or she just interacted with as taking towards him or her. This behaviour is also intended to create the possibility of mimicking the behaviour of human beings who also respond to the outcomes that they experience. Reassignment to a different stance is not a necessity, only a possibility; not all outcomes lead people to change the way they see those with whom they interact.

### 4.2.2 Normal/Classic Agents

Normal/Classic (NC) agents are intended to represent the way that classical game or decision theory tells us that rational agents are supposed to choose. They see each situation without any personal attachment to it beyond the outcomes that they have available to both them and the other agent that they are playing with. In this way, they represent ideally rational agents who are free from the sorts of framing effects that are believed to often influence non-ideally rational decision makers. One clear example of a framing

effect that is widely believed to be detrimental is the counting of sunk costs (Steele (1996); Kelly (2004)); others arise from various psychological studies such as those done and inspired by Kahneman and Tversky (Kahneman and Tversky, 2000). NC agents achieve an objectivity of assessment by evaluating each game in its entirety; simply put, they know exactly which game they are playing and what player-position they have been assigned within the game. Given this knowledge, in conjunction with the knowledge of who they are playing against and what they believe the relationship status between them and the other agent to be, each agent makes a play as determined by their behavioural genetics.

Since they are playing a 2×2 game their choice amounts to a decision between one of two possible actions. As shown in Chapter 3 and displayed in Appendix C, there are 726 possible 2×2 games; however, there are more than 726 possible scenarios that each NC agent must be prepared to face. Each agent playing the game has a role assigned to them that amounts to knowing whether they are choosing between the row player's options or the column player's options, were the game to be represented as a normal form game matrix. This means that not only must every NC agent be prepared to play any 2×2 game, they must be prepared to play these games from either of the two positions available. As an upper limit each player needs to be able to play any of $726 \times 2 = 1452$ perspectives. Since some of the 726 2×2 games are symmetric this number may be reduced since the payoff structure in such games will be perfectly mirrored for both players. In total there are 39 2×2 games that are symmetric across players and so NC choosers are responsible for being able to play from $39 + (726 - 39) \times 2 = 1413$ different positions. With nine possible relationships that an agent can assess between themselves and another player NC agents must therefore have $1413 \times 9 = 12717$ bits to fully describe their behaviour set. Since each bit can take on only one of two possible values there are $2^{12717}$ uniquely different decision matrices that an NC agent can hold.

### 4.2.3 Status Quo Agents

Status-Quo (SQ) agents are intended to more accurately represent the way in which it is supposed that real humans make choices within a frame of reference. Rather than seeing the situation from a neutral or objective perspective, people

often see themselves as already being in one of the possible outcomes. Choice becomes less about making the choice that will bring about the best outcome than it is about choosing whether or not to step away from what is seen as the current state of affairs or "status quo." This status quo based choice might amount to a framing effect that may closely resemble what is known as the status quo bias ((Samuelson and Zeckhauser, 1988), (Kahneman et al., 1991), (Conesa and Garriga, 2003)), roughly providing undue weight to the current state of affairs during decision procedures simply because it is the current state of affairs. This said, SQ based agents do not necessarily suffer from this framing effect because their construction only changes how they see the game without forcing any preference for playing on the status quo over departing from it.

SQ agents play the exact same games as the NC agents, but they do not see the game in the way that NC agents do. What SQ agents see is how the payoffs available to both agents stand in relation to a randomly assigned outcome that functions as the status quo. Both agents are assigned the same status quo outcome and they know exactly which outcome this is. Each payoff outside this status quo appears to the agents playing the game as something that is better, worse, or the same as the payoff that they would receive on the status quo outcome. This modification of the presentation of the game that the agents are playing results in a reduction of the total number of possible scenarios that each agent must be able to navigate when compared to NC agents.

There are four possible outcomes in any 2×2 game. With one of them functioning as the status quo there are three other possible outcomes, each with two payoffs (one for the agent choosing and one for the other player) that must be evaluated in relation to the status quo outcome. This leads to a total of $3^6 = 729$ possible status quo assessments which each agent must have a built-in response to. Of course, SQ agents must also be able to account for the nine possible relationship assessments that they can make and thus there is a total of $729 \times 9 = 6561$ bits that describe the possible behaviours of any SQ class agent. Since each bit can take on only one of two possible values there are $2^{6561}$ uniquely different decision matrices that an SQ agent can hold.

Allowing agents to effectively choose from a status quo amounts to giving them an additional piece of information with which to play the game. This addition is meant to partially capture the difference between asking someone

what they would do in a hypothetical situation and what they would do when actually making a decision framed within a context equivalent to the status quo. But why create agents that play from a status quo position at all? Two reasons. First, there is evidence to suggest that people make choices from a status quo position on the basis of the well documented framing effect (Khaneman and Tversky, 2003:209-223). Explanations of social behaviour and systems development have already started to take this into account (Binmore, 1994 & 2002). Second, many of the games that people really play are not out there in the world in some tangible sense, even if there are payoffs for each outcome that are made up of tangible items like money, resources, or even points. Rather, each player looks at what is available and then reevaluates the worth of these outcomes based on their own preferences and values. As any poker player can tell you, there is often more than money that is being played for. Status quo playing agents amount to a variation on this important idea.

### 4.2.4 Comparing Agents

Epstein (2006) raises the issue of the lack of standards for model comparison and replication of results. The overall thrust of the concern is that subtle changes to models can have "momentous consequences" (Epstein, 2006, p. 29). The Huberman and Glance (1993) critique of Nowak and May (1992) is the most widely cited example of this. This critique shows that simply switching from synchronous updating to asynchronous updating is sufficient to produce very different outcomes. This does not make the primary claim made by Nowak and May—that iteration is sufficient to make cooperation rational in Prisoner's Dilemmas—false, but it does highlight the need for qualifications when making such claims; namely to make explicit the *entire* set of assumptions that are built into the model. The only way to do this without leaving out anything that might be of importance—even though it may not appear to be important—is to share the entire set of code responsible for producing the results, as is done in Appendix C. Still, even when this is done, complete assurance that results can be perfectly replicated are not guaranteed when hardware differences are possible, as discussed in Appendix B.

Even with the code provided there remains the need to compare "apples and oranges" in ways that are justifiable given the circumstances of the simulation trials and that do not import too many questionable interpretations. The

direct factor for comparison between NC and SQ class agents is population size. Population size is the closest thing to an endogenous variable in both of the models created by choosing one agent class over the other and so it is the natural factor of choice for further analysis.

While it may be more difficult to directly compare other factors, such as the expression of behaviours across a population because the NC and SQ agent classes do not share any common behaviours as a result of their different architectures, it is possible to make indirect comparisons. The process for making these comparisons of behaviours across agent classes amounts to grouping behaviours within each class on similar criteria, measuring the relative changes in the expression of the groups over time within each class, and then comparing these relative changes in the behavioural composition of the population across the classes. Pareto Optimality, Anti-Pareto Optimality, Martyrdom, and Anti-Martyrdom are the groupings used to make these indirect comparisons.

Adding an important degree of robustness to both of these comparisons is the aggregation of information in two ways. First, by focusing on entire populations the wide variability that can exist within individual agents can effectively be ignored and a working picture of an "average" agent substituted for all the individuals. Of course, it is *highly* unlikely that any agent within the population will be identical to the picture of the average agent that will be drawn. It may even be the case that this aggregation hides entire groups of agents that exhibit wide ranging differences. Such observations serve as a warning that population level statistics must be used with a degree of caution. Second, these comparisons are lent additional robustness through the ensemble approach to simulation that has been used. Recall that this amounts to running a large number of simulations that each start from different compositions and then examining the results as an aggregated whole. In this way it is possible to build a picture that represents the entire set of possibilities that may result from the model being used without actually exploring all these possibilities (an intractable task). This method ensures that no single outlying population is taken as being representative of the overall success of the population model that each class is a member of.

### 4.2.5 Environment

The environment within which both agent classes exist consists of two parts, one physical and the other social. The physical environment affects all agents and sets the rules within which they interact with each other. It is not physical in the sense that it is "real" or tangible in the way that the physical environment that we occupy is, but the constraints that it imposes on the agents within a simulation are just as real for those agents as our physical environment is for us; we must both live by these rules. Within the simulations the physical environment is composed of two things: games and a set of parameters that affects the agents outside games. The games agents play define how the agents interact with each other. They are a combination of both social and physical environmental conditions from the world we are used to. As mentioned earlier, these games are all 2×2 games so each agent is confined to a choice between two options within any game. Games also offer payoffs. Both players making a choice results in an outcome, one of the four possible combinations of choices between the two agents that are available, and each player receives a payoff on every outcome. Payoffs are in the form of points that agents can use to spend on things outside the context of the games. There are ten possible point values that an agent may receive, ranging from zero to nine.

There are four parameters that affect agents outside of the games they play:

1. Generational Ante - This is the point price that each agent must pay in order to survive into the next generation. Agents that can pay this price are given the chance to spawn and allowed to continue into the next generation.

2. Spawning Cost - This is the point price that each agent must pay in order to spawn. Agents that can pay this price and find another agent that is willing to spawn with them will produce a single offspring with their partner. This offspring is generated by a series of virtual coin flips to decide which of each characteristic is inherited from each parent. Spawning costs are paid after the generational ante. Agents can spawn as many times as they have the points to pay for. Agents are released into the next generation with whatever pocket change they have remaining after spawning as many times as possible.[3]

[3]Note in advance of the results that spawning cost proved to only be relevant when

3. Playing Discrimination - This sets the maximum number of attempts that any agent may have in trying to find a partner who is willing to play with them. When set to zero no discrimination is allowed and every agent must play with whomever they are randomly matched. Any number higher than zero allows every agent that many random matchings to find a mutually agreeable pairing for playing an unknown game.

4. Spawning Discrimination - This sets the maximum number of attempts that any agent may have in trying to find a partner who is willing to spawn with them. When set to zero no discrimination is allowed and every agent must play with whomever they are randomly matched. Any number higher than zero allows every agent that many random matchings to find a mutually agreeable pairing for spawning.

5. Number of Generations - This variable sets the total number of generations that a population is allowed to exist through before it is removed and a new population trial is begun.

6. Number of Rounds - This sets the number of times within a generation that the agents within the population are given the opportunity to find a partner and play a game before they are forced to pay the generational ante and given the opportunity to spawn.

The environment in which an agent exists has constraints that go beyond these physical-type restraints. Agents also exist within a social framework that arises from the interaction of all the agents that currently compose the population. Clearly, the physical environment should have a distinct effect on the shape of the social environment. Harsher physical conditions will lead to the rise of alternate strategies because not every behaviour can survive as the ante to live between generations and spawning costs increases.

Changes to the model that will have the most direct affect on social environment include:

1. Activating the "ForePlay" dispositions of each agent.[4] Recall that in order to play a game either agent type must have some information

the generational ante was relatively easy to come by. In the trials that were carried out the generational ante was much harder to acquire and analysis showed that spawning cost was not statistically relevant in these cases so it was dropped from analysis to simplify presentation and discussion.

[4] `AfterPlay` is a distinct module within the program that manages the behaviours of

about the game being played and an estimation of the relationship that they have with the other player. Since agents must assess some form of relationship before they can determine how they should play a given game, they must have a set of defaults that can be used when they meet an agent for the first time. These defaults can be implemented in one of two possible ways. In the most basic way, all agents within a population assess relationships with potential new partners in the same way; as an "N". A more complicated population is created when agents are randomly assigned one of the three possible relationship assessment values as a default. This allows for the creation of agents that perhaps more closely match onto personalities that human beings can be seen displaying, such as those people who feel that they like everyone, but that everyone hates them.

2. Activating the `AfterPlay` module within the agents. Recall that `AfterPlay` is the module containing the mechanism that causes agents to react to the outcomes of the games they play with other agents. When this mechanism is enabled agents evaluate outcomes and use the products of these evaluations to change either the category that they assign the agent they have played with to, and/or the category that they believe the agent that they just played with now assigns them to. This results in agents facing consequences for the outcomes that they bring about when playing with other agents; agents do not necessarily play the same way when they play with the same agents over and over again.

3. Agent Memory Cap. Agents have a set size that they occupy in memory with one exception: the number of past interactions that they remember. Initial trials of the simulation allowed agents an infinite memory capacity. However, this led to two problems that proved insurmountable: memory overflow beyond the RAM available which then led to severe performance decreases as chunks of memory had to continually be reallocated and moved as every agent's memory continued to grow. Two remedies were introduced to overcome these issues. The first involved removing from every agent any memory of any agent that dies; within

agents following the play of a game. There is no corresponding single module for what agents do before playing, but the set of all the pre-play behaviours can be thought of as a module in principle. It is the fact that `AfterPlay` is an actual circumscribed procedure within the code that it is called out by a special font while "ForePlay" is simply put in quotations.

the context of the simulation the memory in the computer that this information occupied was clearly being wasted since no one would ever be able to play with that agent again. The second involved putting a cap on the number of rounds and generations that an agent can go through without interacting with a particular agent before all relationship stance information associated with that agent is deleted. In combination with population size this cap has the potential to alter which agents prosper and which do not. For example, in a large population it is less likely that an agent will see another agent before the memory cap is hit. As a consequence, agents that treat other agents badly can do so with a smaller likelihood of consequences. Very large populations with a relatively small memory cap have the potential to result in a significant negation of any affects that `AfterPlay` might bring about while smaller populations would still be directed by the `AfterPlay` distribution across a population. Even though this restriction is motivated by pragmatic considerations these changes can be seen as bringing the model closer to real life; people forget, and they really only think about those with whom they regularly interact.

Besides the direct affects that modules like `AfterPlay` bring to the social environment, it must be remembered that there are consequences that arise from the physical environmental conditions as well. Anything that has the capacity to change which agents compose a population will affect the social environment in some way. Everything from the initial random distribution of characteristics across all the agents in an initial population, the size of the initial population, playing discrimination, and spawning discrimination.

## 4.3 More on Modelling

The model outlined here may be described with the following classifications: agent-based; ideal type; evolutionary; and non-spatial (Gilbert (2008); Billari et al. (2006)). It is also appropriate to classify this model as a "varied model" and/or "multi-model" given that the general approach resembles an ensemble forecast (Eckel and Mass, 2005, p. 1). Each of these is classifications is discussed in turn.

### 4.3.1 Agent-based

An agent-based model is primarily composed of objects intended to represent independent beings. These objects are referred to as "agents". Agents interact with each other against the background of the modelled environment. These agents are independent insofar as how each agent responds to various stimuli is solely the result of the construction of that individual agent. Since each agent is independent, the possibility of heterogenous populations rises, one of the principal benefits of agent-based modelling (Gilbert, 2008).

Agent-based modelling differs from traditional/mathematical modelling. Mathematical models consist of a series of functions describing the behaviour of each component of the system, making it clear that there is an equation capable of describing the behaviour of the entire system. In contrast, agent-based models (typically) do not even imply that such an equation exists. Such an equation does exist, if the Church-Turing thesis is accepted, but any attempts to try and produce it are likely to end in failure and frustration since for even very simple models such an equation is intractable (Epstein, 2006). The reason for this is that the progression of an agent-based simulation depends a great deal on the interactions of the agents within the population with both each other and the environment. The outcomes of these interactions cannot typically be easily deduced in advance, especially if the outcomes provide feedback that changes either the environment or the agents within the population. Further, agent-based models typically feature the characteristics exhibited by chaotic systems: non-linearity, sensitive dependance, etc. (Smith, 2007). This means that each run of the model through a simulation may produce different results.

Agent-based models offer a variety of benefits that compensate for the inability to fully describe them mathematically. These benefits include: ontological correspondence (it is easy to relate the agents to the objects they are meant to represent, regardless of whether these objects are groups or individuals); heterogeneity is easily accommodated (each agent can be different from all other agents in the simulation even if similarities are shared); and a capacity to directly represent the environment (it is relatively easy to build environmental factors and to see them as separate from the agents) (Gilbert, 2008).

### 4.3.2 Ideal Type

This simply means that the model is an idealization of the phenomena that it targets. As an idealization certain features of the target are emphasized while others are de-emphasized. For example, when building a model representing the interactions of wolves and rabbits details like the day to day weather and the effects of natural selection may be ignored leaving only key features of the target animals like movement, feeding, and reproduction.[5] The model described here is an ideal type model because it emphasizes idealizations of interactions (agents only play 2×2 games) and the corresponding abilities that the agents must have in order to participate in these interactions.

### 4.3.3 Evolutionary

My model features agents that interact within a framework of Darwinian natural selection. Agents interact with each other and the results of these interactions allows for an assessment of the fitness of each agent to be made. Agents below the requisite fitness threshold are removed while those above are kept. Agents that survive from one period to another while sustaining a sufficiently high fitness assessment are given the ability to reproduce, with offspring inheriting traits from both parents.

### 4.3.4 Non-spatial

The agents within my model are not constrained by a representation of a physical geography and do not have their interactions with other agents confined in ways that are typical of spatially bound agents (e.g. agents are not restricted to interacting only with agents that they are close to). Non-spatial models are a kind of ideal type model because spatial orientation is clearly a part of our existence; however, it is not significant in certain circumstances. Given the increasing ability that people around the world have to communicate with each other on an almost instantaneous basis, regardless of distance, non-spatial models are more appropriate now than previously. They are particularly well suited to analysing network formation and information flow.

[5] An excellent demonstration of a simple simulation depicting rabbits and wolves can be found at http://www.shodor.org/interactivate/activities/RabbitsAndWolves/.

### 4.3.5 Varied-Model

My approach to modelling populations is best classified as a varied-model and/or multi-model approach. This approach to ensemble forecasting varies the parameters of a model through multiple runs of a simulation as a form of sensitivity analysis, ensuring that the results obtained have an appropriate degree of dispersion (i.e. The results represent the range of outcomes that are possible). This approach is to be contrasted against multi-model and single-model approaches. Multi-model approaches use simulations produced from different underlying models to disperse results over an informative range while single-model approaches focus on a single model with little or no variance within that model. The difference between each of these classifications is ultimately one of degree and what counts as a different model is both open to interpretation and a matter of context (See Chapter 2). On all but the broadest interpretations of what counts as a model, the SQ and NC agent classes are different enough to constitute different agent-based models of human beings and thus qualify my approach to forecasting possible social worlds as multi-model. Given the large degree of variance of inputs each agent class is tested through and the number of times these tests are conducted with the same initial conditions but through varying the randomization within the progression, my model is almost certainly a varied-model approach as well.

A model with these features became necessary because of the question that I was interested in answering; roughly, "What possible social worlds, as defined by the behaviours people express and are predisposed to express, are possible?" The agent-based nature of the model I have developed follows directly from the fact that societies are heterogeneous, being composed of nonidentical individuals, and chaotic systems.

The ideal type feature comes out of necessity because it is simply not possible to capture all the features of a real-world society in a model (in all likelihood the social world is more complicated than the physical one by many orders of magnitude). The evolutionary nature of the model became necessary for two reasons. First, human beings appear to be on all scientific fronts the products of Darwinian natural selection so it makes sense that a model composed of agents meant to loosely represent all human-like beings should also

exist with this important feature in place. Second, testing all possible societies across all possible evolutionary paths becomes intractable as the complexity of the model increases. In the case of my model there is sufficient possible variability across individual agents that such research could not be completed within the remaining lifetime of our sun. What the evolutionary feature does is allow is for a series of randomly generated societies to be "grown" and the outcomes used to make predictions regarding the features of similar societies. The evolutionary process used is commonly referred to as a genetic algorithm (Axelrod (1997);Gilbert (2008)).

One important difference between my model and most implementations of the genetic algorithm is that in the trials reported here there is no explicitly coded possibility of mutation during reproduction.[6] Foregoing mutation was a conscious choice made to prevent additional complications from cluttering up this early version of the model. I also judged that given the random initial distribution of behaviours across agents within the initial population used in conjunction with the ensemble modelling approach, this would not be an overly detrimental decision.

The non-spatial feature was selected in order to keep the model as simple as possible. Within any human population it is common for people to have a multitude of interactions with various people over the course of a day. These interactions do take place within a spatial environment, but this environment is often not relevant to the interaction. For example, it typically makes little difference whether an exchange of money for goods between two people takes place at one side of the city or the other. Models that include spatial relations typically simplify these relationships to a very high degree, frequently using a checkerboard model of the world, and allowing agents to interact only with those in their immediate neighbourhood.[7] At the outset it was assumed that the predisposition of any given human being towards any particular social behaviour is the result of many interactions with many agents and since the simple geographic models typically used would fail to capture this, a non-spatial model would work best.

[6]Mutation is still possible, although unlikely, due to mistakes made by the physical machine on which the simulation is run or due to software errors in the operating system.

[7]Note there are three strong influences that direct people to use the checkerboard model. 1. A version of this was used by Schelling in what is credited as the first use of agent-based modelling (Schelling (1971); Gilbert (2008); Billari et al. (2006)). 2. It is now used so often that people cannot help but think of things within this framework. 3. It really is very simple.

### 4.3.6 Benefits of this model

What general benefits does this framework provide over and above alternatives? The major benefit of the modelling framework chosen is the easily scalable sophistication of the agents. This benefit is realized via three differences from similar models: multi-game capacity agents, two different agent classes, and theory of mind related decision making.

Scalable agent sophistication amounts to having a modelling framework that allows the capabilities of agents to be easily extended or retracted. This is accomplished by having each agent composed by a data set that fully describes how they will behave in the face of any possible assessment of the situation. This data set is best thought of as a multi-dimensional array, but it is implemented as a bitset (think of all the rows from a multi-page table arranged to form a single long line). The approach is similar in design to that used by Axelrod (1984) when further investigating the results of his famous tournaments through the use of the genetic algorithm.[8] In addition to being flexible, this approach allows programs with this type of behaviour modelling to run quickly on whatever architecture they are implemented on since it is extremely fast to execute searches within the bitset. Such "look-up" procedures allow us to find values stored in memory rather than by an act of computational acrobatics that a behavioural formula would require.

To understand the nature and value of this framework in more detail it will be helpful to spend a short time considering the difficulties that any decision procedure must address and how this framework overcomes them. Specifically, there are three significant difficulties when it comes to designing agents capable of making sophisticated decisions in ways analogous to actual human beings.

First, the procedure should represent and produce behaviours similar to the targets of the model. In the case of human beings the model should have two such capacities: it should allow for discontinuities and for new experiences/information to inform future behaviours. Evidence for decision making discontinuities in human beings is rampant (Kahneman and Tversky, 2000). People simply do not exhibit choices that can be plotted on a continuous curve, there often being cases where choices counter to all expectations are exhibited.

[8]Tit-for-tat was only victorious in the first two tournaments that Axelrod ran. In a third, anniversary, tournament the victor was a set of programs that "conspired" to elevate one of their members to victory. For a quick overview of the downfall of TFT please see http://www.wired.com/culture/lifestyle/news/2004/10/65317.

To see that this is so, consider the anecdotal case of a friend of mine who often goes to the effort to pick-up change off the ground but will not switch his or her savings from a regular to a high interest account until he or she has thousands of dollars to "make it worthwhile". This is despite the simplicity that the Internet brings to the process and explanations and advice. I am certain that I am not alone in this experience and am very likely unaware of similar behaviours that I exhibit. The ability to change behaviour in response to new experiences/information is also a well known human capacity. Obstinance and various documented rational failings aside, people learn and this needs to be reflected in any agent-based model with human beings as the target.

Second, the procedure itself should be simple enough to be understood. This is particularly true in cases where the purpose of the agent-based model is to investigate rationality or decision related behaviour. It would make little sense to perform an investigation that produced results that were as complicated as the system or process that prompted the investigation.

Finally, it must be possible to implement the procedure within a given population taking into consideration the resource limitations of both the computer that will host the simulation and the overall timeline of the project. There is little point beyond the inherent value of developing a model if it cannot tuned, tested, and then used to generate results that might form the basis of a deeper understanding.

The agents within the models used in this project satisfy these conditions by design. This is shown in Appendix E, which compliments the high-level overview given here with a mid-level description of the how the model was implemented followed by the complete code used to produce the simulations with full annotations.

# Chapter 5

# Comparing SQ and NC Agents

As outlined in Chapter 4 two different types of rational agents were modeled so that they could be compared via simulation: normal/classic (NC) agents who play all 2×2 games in an ideal and objective way and status quo (SQ) agents who play the same games, but who base their choices on a randomly preassigned status quo position within each game. The supposition at the outset of this project was that given that status quo decision making behaviour is documented as having an effect on actual human beings that it may provide some benefit even though it is often referred to as a bias and has been shown to lead to less than optimal decisions in many situations ((Samuelson and Zeckhauser, 1988), (Kahneman et al., 1991), (Conesa and Garriga, 2003)). This supposition was based on the reasoning that if this behaviour was not at least beneficial in some cases, it would have been eradicated long ago as evolution is notoriously unkind to behaviours that are not beneficial (or at least not harmful). With this in mind the simulations were constructed under the belief that some small but useful effect might be shown in some specific cases or that perhaps the performance of populations of SQ agents in comparison to those of NC agents would not prove to be significantly poorer. The results of simulating populations of these agents shows that this original supposition underestimated the capabilities of SQ agents—not only do they do as well as NC agents under the majority of environmental conditions imposed, but in certain specific cases they significantly outstrip the capacity of NC agent populations to coordinate their actions and ultimately grow the population.

In presenting the results of the simulation trials it must be remembered that each type of agent was built with the capacity to exhibit a number of traits or capacities and that these could be turned on or off. Of these traits,

the ability of agents to model Theory of Mind (ToM) is the most significant and it is worthy of being examined on its own. For this reason the results of ToM enabled agents are given their own chapter (Chapter 6) and not discussed here except to make passing reference as required for a complete comparison of the two agent types.

## 5.1 Simulation Summary

Early tests of the simulations showed that populations where the generation ante (i.e. the cost each agent must pay to live into the next round and have the chance to reproduce) and the cost of spawning (i.e. how much each agent much pay in order to reproduce a new agent be cooperating with another agent) were low resulted in rapidly growing populations that quickly outstripped the capacity of the simulating machines in terms of both memory requirements and speed of processing. With an eye to avoiding this unwanted scenario, only cases where it was relatively difficult for the agents within the population to survive across generations were examined. Since the payoffs available in any randomly generated 2×2 game varied from 0 to 9 points, the average payoff available in each game was 4.5. Since each generation consisted of one hundred rounds where each player had the opportunity to play one game a randomly playing agent should be able to pickup an average of 450 points. This formed the anchor around which the generational ante for the simulation trials would fall. Cases where the generational ante was 440, 445, 450, 455, and 460 were considered.

In addition to the generational ante both the cost of spawning and the initial population size were varied. The cost of spawning was set at one of 375, 400, or 425, on the expectation that these relatively high costs would ensure that only those agents who were able to survive more than a few rounds would be capable of reproduction. The initial population size was varied across three levels, 10, 100, and 1000 so that the effects of a wider pool of agents could be examined.

Combining the variations of these three options produces $5 \times 3 \times 3 = 30$ cases for examination in each of the two agent types. Within each case 144 populations were produced and simulated, resulting in a total of $30 \times 144 =$

4320 simulation trials for each type.[1,2]

## 5.2 Results

The most interesting results of the simulation trials in terms of comparing the two agent types is shown across figures 5.1 and 5.2. Each of these figures collects together a scatterplot of all the population sizes at the corresponding time/generation for the given conditions shown at the top of each graph within each collection. these conditions are given with the generational ante first and followed by the initial population size so a graph labeled "445, 100" indicates that the population results displayed are for a generational ante of 445 points and an initial population size of 100.

The major difference between these two sets of figures is the regular growth of the SQ population beyond what NC agents were able to achieve under the same environmental conditions. This contrast is starkest when the generational ante is smallest (440) and the initial population size is largest (1000). In this case the SQ agents are able to consistently grow their population size back to a level that is at least as great as the initial population size and up to three times this initial size. NC class agent populations, in contrast, struggle to maintain a population of roughly 500 over the same one hundred generation period. The size of the advantage that SQ agents have over NC agents can be seen in almost every combination of generational ante. Possible exceptions seem to be those simulation trials with the smallest initial population size (10). Given that such a small number of populations are actually represented in this apparently minor divergence and the small population sizes being dealt with it is likely that this is simply the result of the randomness of the model.

[1]Including all the ToM variations results in 64800 simulation trials.

[2]These simulations were both compiled and carried out on the University of Alberta's Linux Cluster. At the time the simulation was run this cluster offered a 64-bit environment over 188 AMP based cores with clock speeds ranging from 2.0GHz to 2.4GHz. The simulation code originally produced and tested on a stand-alone machine was optimized to take advantage of the increased computing power offered by this resource through the introduction of OpenMP modifications, allowing the simulation to take advantage of up to four cores at once. This reduced the overall runtime by roughly two-thirds. The compiler used was g++ and the O3 (maximum) optimization level was invoked.

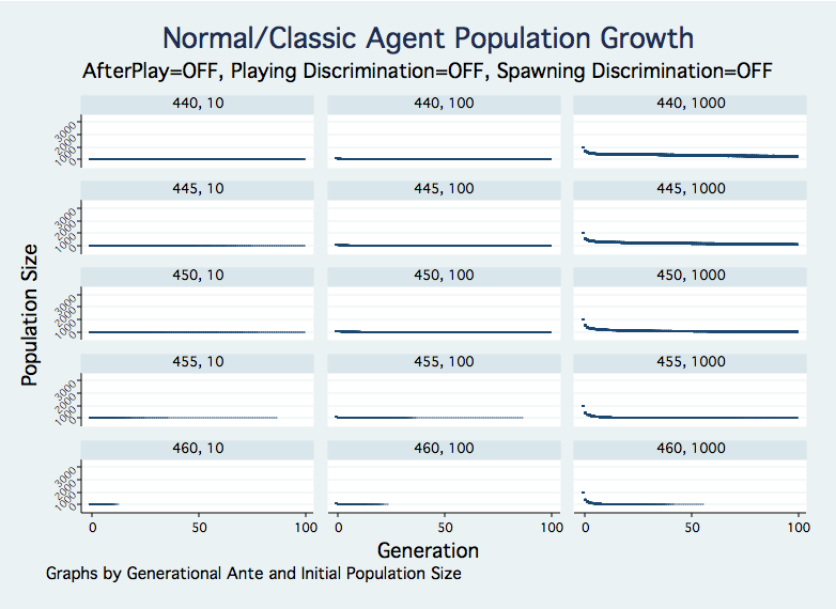


Figure 5.1: Population Growth for NC Agents by Generational Ante and Initial Population Size

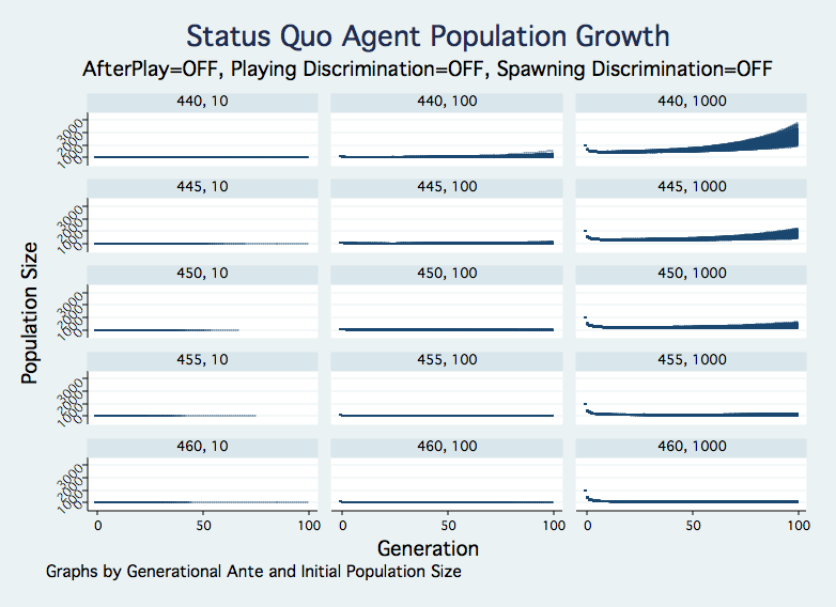


Figure 5.2: Population Growth for SQ Agents by Generational Ante and Initial Population Size

The source of this increased population size is the ability of SQ agents to adapt to environments more effectively than NC agents, acquiring higher payoffs more frequently in the 2×2 games that they play which results in the agents inside these populations being able to survive from generation to generation and reproduce more frequently. This is shown in figure 5.3 where the average payoff received within each SQ population (the dark scatterplot) quickly jumps above the average payoff offered per generation (the dark orange line) and remains high for most of the remaining generations.

This is not something that NC agents are able to accomplish with anything close to the success of SQ agents. Note that, instead of a band that rises over the average payoff offered, the average payoffs received by NC agent populations looks as though it has been split evenly in two by the average payoff offered. That this is not entirely the case is shown by the gold line which rises slightly above the average payoff offered over the course of each generation. This golden line is the mean of the average payoffs received across all the populations displayed in the scatterplot. The rise of this line above the average payoff offered shows that there is an evolutionary effect taking place within the NC agent populations, it is just much smaller than that displayed by SQ agents.

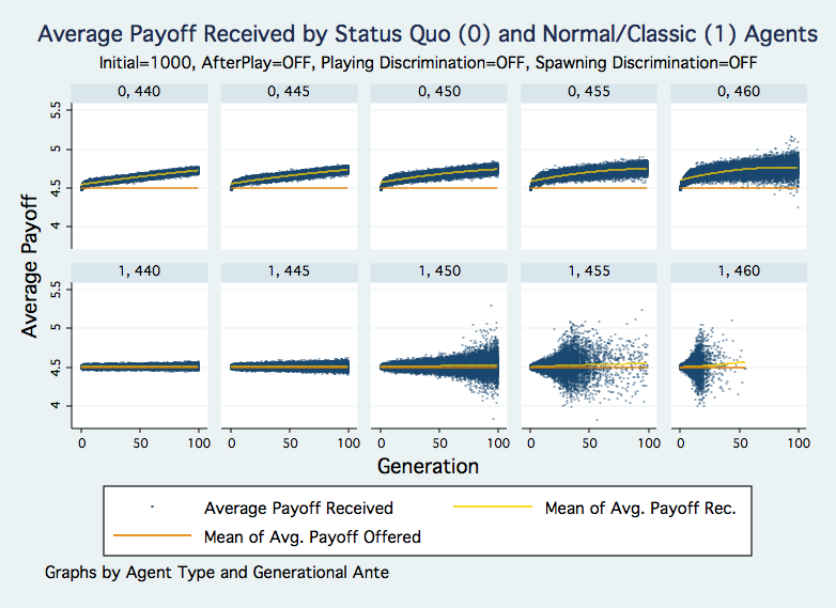


Figure 5.3: Average Payoffs Offered and Received by Agent Type and Generational Ante

Why is this happening? Why are the NC agents not able to survive and perform as adequately as the SQ agents? The interaction of the following three possibilities is likely responsible:

1. There is a mistake in the code used to generate and deploy NC agent simulations.

2. The relative simplicity of SQ agents over NC agents makes it much easier for them to coordinate effectively as a population.

3. SQ agents have an inherent advantage over NC agents given how their construction leads them to see/experience the environment they exist in.

Each of these will be discussed in turn.

### 5.2.1 Possible Mistakes

Despite a great amount of attention the possibility of a simple coding mistake leading to the displayed results remains a possibility. As discussed in Chapter 2 this is a real possibility in any simulation project. The likelihood of a coding error being the source of this result is reduced by the fact that both SQ and NC agents are written to take advantage of the object oriented nature of C++. This means that aside from the agent-objects the vast majority of the code between them is identical, having been written once and then copied over to the other agent-based simulation program and adapted as required by the agent. This is true of most of the functions used in the simulations as well as the objects.

In addition to this preventative measure both models have undergone extensive testing and revision throughout the simulation. The code used to generate these results is the fifth major revision since the project began roughly five years ago. While time spent does not necessarily equate to reduced errors the long term use of and immersion in a piece of code does stand to reduce the likelihood of error in conjunction with active testing.

Finally, when these results were discovered all the likely sources of a mistake that would bring about this result were double-checked. The most significant possible error sources were the game playing mechanisms of both agents (not allowing agents to play games correctly would easily lead to a distorted result) and the reproductive capacities and controls affecting both agents (if NC agents were prevented from effective spawning or SQ agents were receiving

something akin to a two-for-one deal then an advantage could be accounted for. Neither of these two investigations nor any of the smaller ones that were carried out turned up anything untoward.

The likelihood of an unfair advantage being held by SQ agents due to a coding mistake is mitigated somewhat by observing what happens when the environment in which they are living is made more complicated. The introduction of a Theory of Mind-like ability into SQ agents, as is discussed in Chapter 6, almost completely eliminates the advantage that SQ agents are displaying under the softer environmental conditions discussed here.

Regardless, the possibility of error remains. In order to maintain openness about this and to aid in hunting down the error should one exist, the entire code base has been shared in Appendix E.

### 5.2.2 Relative Simplicity of SQ Agents

Recall from Chapter 4 that in order to play games in a way that is akin to the way the standard economic theory advises, NC agents must be able to play from any of 1413 possible roles that might arise when living in an environment that offers each of the 726 games through the random production of the payoffs available in each outcome through the random generation of an integer value from 0 to 9. SQ agents are responsible for just over half this many situations, requiring the ability to manage just 729 different positions that they might find themselves in.

Put in terms of being able coordinate a set of actions in a small amount of time the SQ agents will have a significant advantage. Consider that each generation consists of only 100 rounds. Supposing that every game was encountered with equal probability (which is not the case, as shown in Chapter 3) SQ agents would, on average, get the opportunity to use every behaviour at least once every eight generations. NC agents would take fifteen generations to do this, by which point the SQ agents would have had the opportunity to test almost everyone of their behaviours twice. This extra time required by NC agents to test and then expel counterproductive behaviour from populations by removing the carrying agent allows the carrying agents more time to reproduce, making it even harder to ultimately remove them while setting more members of the population up for failure in future generations.

The possibility that the relative simplicity of SQ agents may play a role in

their success, not just in comparison to NC agents, but broadly speaking is also supported by comparing the results presented in Chapter 6 to those shared here. Adding the capacity for theory of mind like behaviour adds a significant layer of complexity to SQ agents while at the same time reducing their overall performance. Simplicity has been called out by Axelrod (1984) as one of the primary reasons for the success of Tit-For-Tat in the original tournaments that he ran, but it is widely believed to be a standard of excellence in everything from Occam's Razor to aesthetics. While there are effects related to the details of implementation that may well contribute to some of the superior growth rates seen by SQ agents, simplicity may have a role to play just the same.

### 5.2.3 Inherent Nature of SQ Agents

SQ agents have two particular characteristics that might go unnoticed/unvalued if not for the likelihood that they are contributing significantly to their ability to thrive even in environments where the more sophisticated NC agent populations do not: they are given an outcome which may act as a focal point for coordination, and a single status quo behaviour is often applicable across a wide number of games.

When a pair of SQ agents are given a game to play they receive an additional piece of shared information; they are told which of the four possible outcomes has been assigned to act as the status quo position. With this information in hand both agents make their decisions regarding whether to continue with the action that makes the status quo outcome possible or to switch to the only alternative. If research into the status quo bias observed in humans is correct then this information is a powerful coordinating factor. With this information in hand agents would be expected to be better able to remain at the status quo rather than attempt to acquire another payoff, even if it is tempting. For example, the 2×2 game known as "Chicken" becomes solved in an important sense as soon both of the players know that one of them is receiving their most preferred outcome (see the bolded payoffs in Table 5.1) .

The final benefit that SQ agents have over NC agents is more subtle and would easily have gone unnoticed were it not for an expansion to the program initially intended to produce a list of all the 726 2×2 games. This expansion produces a listing of all the status quo positions that exists across the set of all the 2×2 games. The listing that resulted from this expansion can be seen

| | | Player 1 | |
|---|---|---|---|
| | | *Straight* | *Swerve* |
| Player 2 | *Straight* | 0, 0 | **3, 1** |
| | *Swerve* | **1, 3** | 2, 2 |

Table 5.1: Chicken

in Appendix D and the code that produced it can be found in Appendix B.

What needs to be noted about the status quo positions shared in Appendix D is that while every 2×2 game has the potential to have up to eight meaningfully different status quo positions the distribution of status quo positions across games is much more varied. Some status quo positions, such as #19 and #22, appear in only a single game (games #2 and #9, respectively) while others occur in over 150 different games, such as #378 and #716; therefore, not only is it the case that SQ agents are simpler than NC agents generally speaking, but the way that this simplicity is realized allows them to take a few behaviours and apply them over a wide range of situations. The fact that certain types of games are more likely to be created within the simulation than others further compounds this effect, allowing SQ agents to cover a very wide range of possible situations with a much smaller number of behaviours.

Evidence that the ability of SQ behaviours to cover multiple scenarios is (at least partially) behind the success of SQ agents over NC agents is available by examining the final distribution of the status quo game playing behaviours within various SQ populations. To perform this examination 2,160 populations were simulated across a range of environmental conditions identical to those described earlier. Initial population size in all cases was left at 1000 randomly generated agents to provide the widest diversity of behaviours in the initial generation. In addition to the information collected in the simulation trials previously described, this series also collected information on the percentage of agents within each population expressing each behaviour. Behaviours are expressible by setting the corresponding bit in the behaviour coding bitset to either 1 ("ON" or "Stay with the status quo action") or 0 ("OFF" or "Switch from the status quo action"). To simplify analysis and speed-up the simulations only the top ten bits most frequently turned on and the top ten bits most frequently turned off after the one hundredth generation were collected for inclusion in this analysis and only from populations that were able to

successfully survive to the one hundredth generation. Figures 5.4 and 5.5 provide the results of this investigation. Figure 5.4 shows the percentage of surviving agents within populations that have the bits listed on the horizontal axis set to 1 or "ON" while Figure 5.5 shows the percentage of surviving populations that have the bits listed on the horizontal axis set to 0 or "OFF".

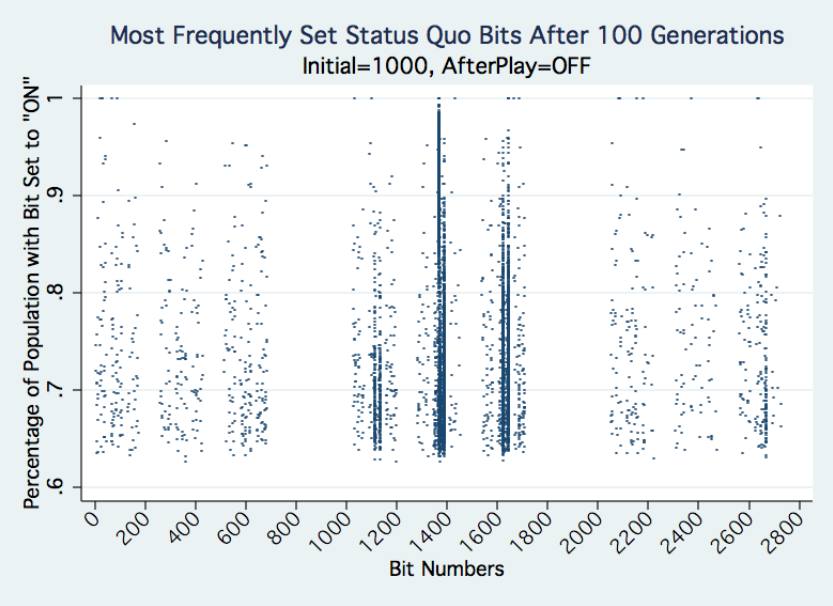


Figure 5.4: SQ Agents, bits set to "ON"

It is the sets of solid vertical bands that are important in each of these graphs. Each of these bands is a dense collections of points, indicating that the behaviours they are associated with appeared in a very large number of the populations from which the data was acquired. The density of these bands indicates that the behaviours they are associated with are important in contributing to the overall health of a population composed. While Figures 5.4 and 5.5 are evidence that there are status quo behaviours that are particularly important to SQ agents, they are not particularly helpful in indicating exactly which behaviours these are. Two things are required in order to connect these bars to the behaviours listed in Appendix D. First, the graphs within these figures need to be refined so that it is possible to determine exactly which bit number is associated with the bands. Second, the bit number that is reported in the graphs does not correspond directly to the index values provided in Appendix D and so a conversion from bit number to index number is required.[3]

[3]The bit number and the index number are different because the status quo positions

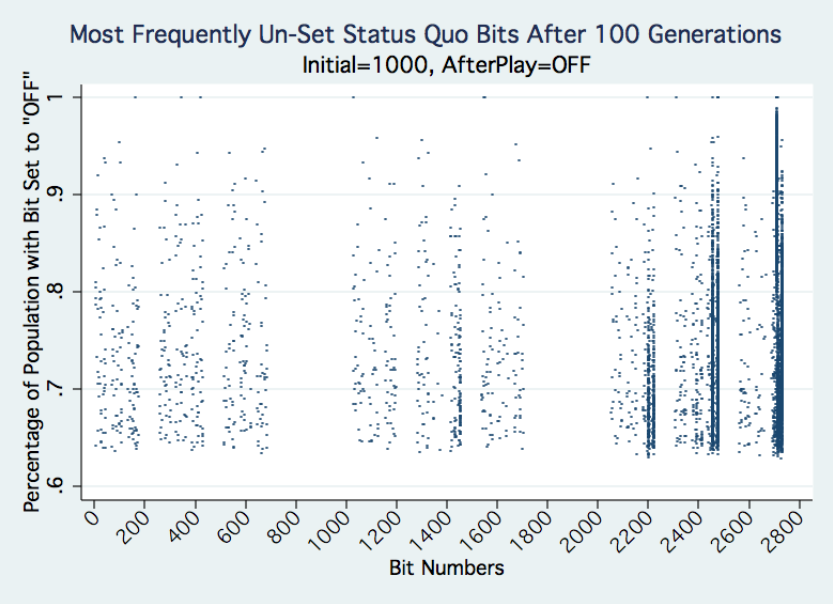


Figure 5.5: SQ Agents, bits set to "ON"

The required refinement is accomplished by looking only at the groups of bit numbers that contain the solids bands. Figures 5.6 and 5.7 show the results of doing this for the bits set to 1 or "ON" (5.4) while figures 5.8 and 5.9 show the results of doing this for the bits set to 0 or "OFF" (5.5).

---

are part of a trinary system that is actually counted and stored using binary. The result of this conversion is values in the binary count correspond to impossible combinations. For example, when making assessments in determining the value of an outcome in relation to the status quo, each agent makes an assessment as to whether or not the the alternative outcome is better, worse, or equivalent to the status quo outcome. This assessment requires two bits, one to record if the alternative is better than the status quo option and one to record if the alternative is worse. If the alternative is equivalent then neither of these two bits is set. On the current interpretation it would be nonsensical for both of these bits to be set at the same time since that would amount to the claim that the alternative was simultaneously *both better and worse* than the status quo outcome. It is such nonsensical bit combinations that create gaps that must be removed for efficient storage of the behaviour set of each agent.

The formula used for converting from what is referred to as the bit number in the graphs presented to the SQ behaviour index number as used in Appendix D is:

$P(x) = x - \frac{x}{4^1} - \frac{x}{4^2} * 3^1 - \frac{x}{4^3} * 3^2 - \frac{x}{4^4} * 3^3 - \frac{x}{4^5} * 3^4 - \frac{x}{4^6} * 3^5$

The formula for converting from the SQ behaviour index number to the bit number in the presented graphs is:

$P(x) = x + \frac{x}{3^1} + \frac{x}{3^2} * 4^1 + \frac{x}{3^3} * 4^2 + \frac{x}{3^4} * 4^3 + \frac{x}{3^5} * 4^4 + \frac{x}{3^6} * 4^5$

In both cases it must be understood that the division indicated is *integer division* (only the integer portion of the usual division operation is returned) and that it takes precedence over simply multiplication when determining order of operations.

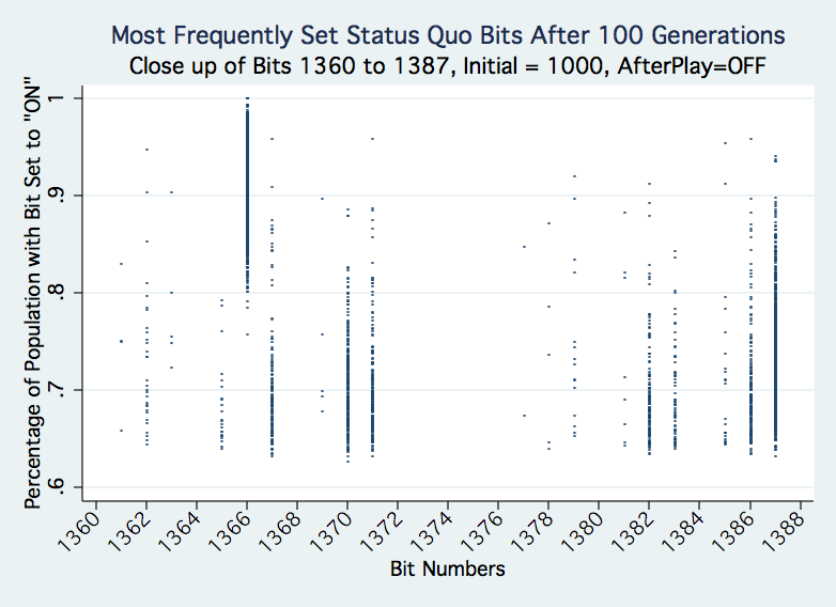


Figure 5.6: SQ Agents, bits set to "ON"–Closeup: 1360 - 1388

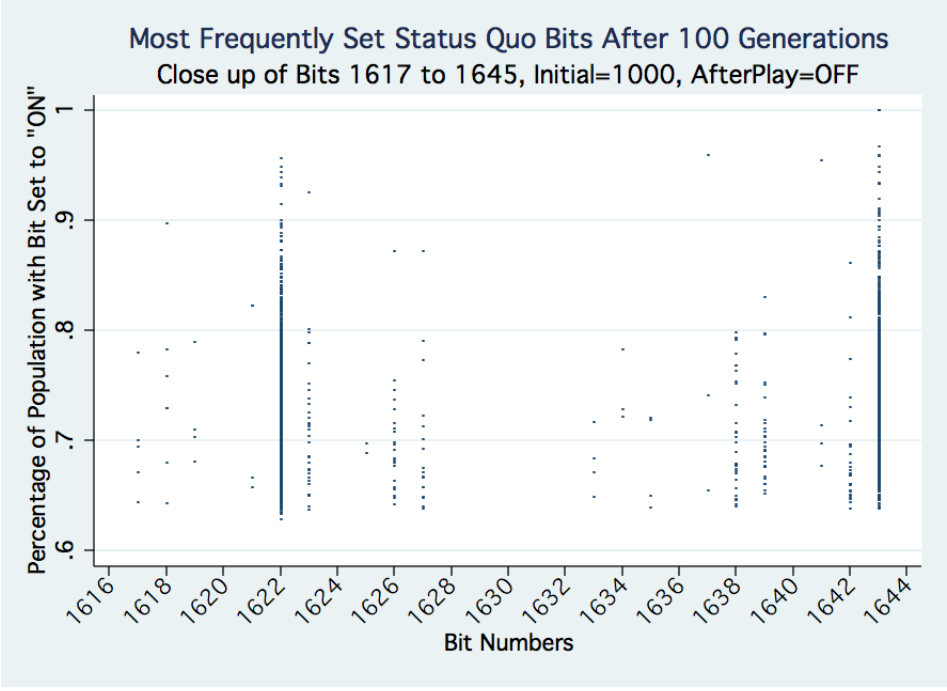


Figure 5.7: SQ Agents, bits set to "ON"–Closeup: 1616 - 1644

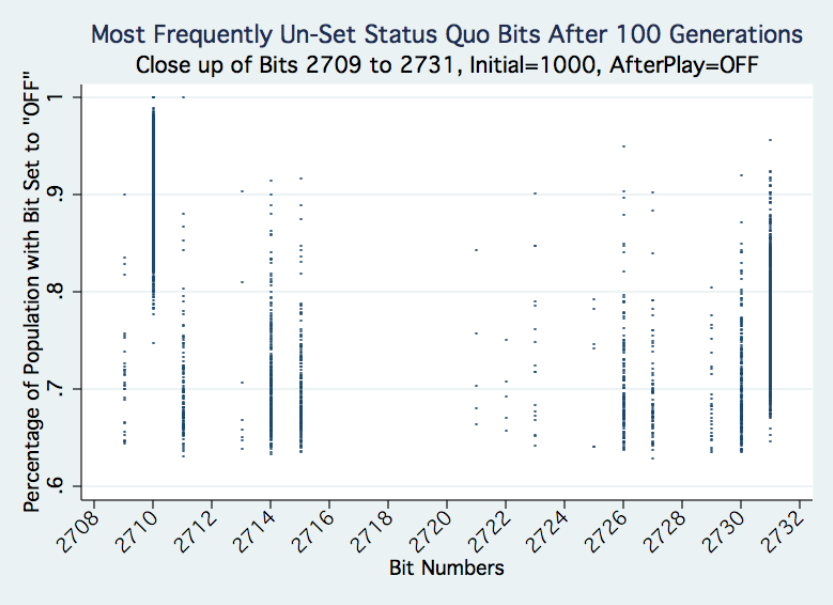


Figure 5.8: SQ Agents, bits set to "OFF"–Closeup: 2708 - 2732

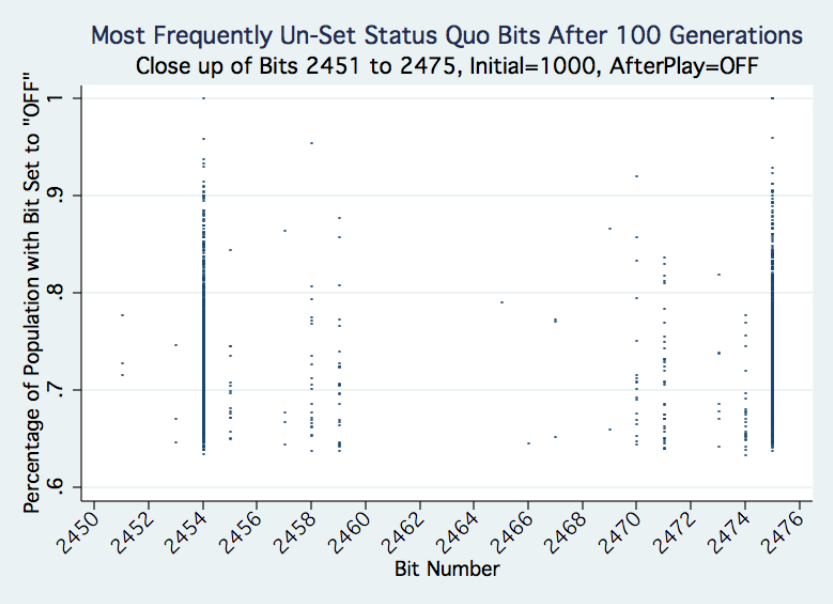


Figure 5.9: SQ Agents, bits set to "OFF"–Closeup: 2450 - 2476

| *Bit Number* | *Index Value* | *Position Count* |
|---|---|---|
| **1366** | **365** | **182** |
| 1387 | 378 | 172 |
| 1370 | 368 | 39 |
| 1622 | 446 | 39 |
| 1643 | 459 | 40 |

Table 5.2: Bits Frequently Set for Playing ON the Status Quo

| *Bit Number* | *Index Value* | *Position Count* |
|---|---|---|
| 2454 | 635 | 40 |
| 2475 | 648 | 39 |
| **2710** | **716** | **172** |
| 2714 | 719 | 40 |
| 2731 | 729 | 182 |

Table 5.3: Bits Frequently Set for Playing OFF the Status Quo

This refined perspective shows quite clearly which bit numbers are associated with the bands observed in the original figures. These bit numbers are summarized in Tables 5.2 and 5.3. Within each table the most significant behaviour as indicated by the strength of the bands within the figures that these behaviours are drawn from have been bolded. Beyond selecting the most significant behaviours from both those bits set to "ON" and those bits set to "OFF" it is difficult to make an assessment of overall importance so no attempt to do so will be made here.

Playing on the status quo most frequently occurs when conditions triggering SQ index position #365 exist and playing off the status quo occurs most frequently when conditions triggering SQ index position #716 exist. While it might initially be thought (hoped?) that these behaviours would correspond to "staying on the status quo position when everyone does worse everywhere else" and "moving off the status quo position when everyone does better everywhere else," this is not the case. The actual status quo assessments for both of these positions are given in Table 5.4.[4] While staying on the status quo position when everyone does worse elsewhere is indeed the number one

[4]Representations for the other values can be found in Appendix D through the use of the index values.

365

| | |
|---|---|
| **0**, **0** | -1, -1 |
| -1, -1 | -1, -1 |

716

| | |
|---|---|
| **0**, **0** | 1, -1 |
| 1, -1 | 1, -1 |

Table 5.4: SQ Positions No. 365 & 716

reason to stay, the number one reason to move off the status quo position for an agent is because they do better everywhere else while the agent they are playing with does worse everywhere else.

A deeper analysis that incorporates all the relevant bits will be required before any substantial claim might be made about exactly what this means. However, at this initial stage these results seem to suggest that under the harsh conditions the reporting populations survived through there is strong evolutionary value in doubling ones advantage by both improving ones own lot while worsening the lot of another. Following this opportunity in importance is the chance to simply improve one's own lot, even if another agent does better as a result of this pursuit as well. These generalizations are supported by the all the behaviours reported in Tables 5.2 and 5.3. Every behaviour in these sets is self-serving, but none of them give any indication of any particular value existing in either aiding the other party when there is no cost to oneself nor in injuring the other party when there is no cost to oneself. Thus it would seem that neither altruism nor maliciousness is a major contributing factor to the success of SQ agents over NC agents. Again, only the top five behaviours for both staying on and switching from the status quo position as indicated by the frequency bands reported in Figures 5.4 and 5.5 form the basis of this claim. There are 729 SQ agent behaviours requiring further analysis and comparison to make this claim with confidence. Still, at this stage it is reasonable to assert that the most likely reason for the success of SQ agents over NC agents is that they are able to bring obviously beneficial behaviours to bear in a wide variety of circumstance through simpler adaptations than are required by NC agents.

# Chapter 6

# Results of Simulating Theory of Mind

## 6.1 Modeling Theory of Mind

Theory of Mind (ToM) is the capacity to either represent, or otherwise account for, the mental states and perspectives of others during one's own thinking or planning processes. It may exist to various degrees with the simplest being the capacity to think about what others might be thinking about but it may be extended to include thinking about what others are thinking about what you may be thinking about what they may be thinking about what you may be thinking about and so on. ToM is an important emerging focus within artificial intelligence (AI) research because of the promising consequences that it may hold for any area involving either collaborative or competitive strategies between multiple agents, regardless of whether they are humans or machines. Such areas include military applications, computer gaming, financial trading, collision avoidance, and psychological analysis (Ficici and Pfeffer (2008a); Ficici and Pfeffer (2008b); Kim and Lipson (2009)). It seems likely that theory of mind has been a candidate for inclusion in AI since it was first postulated by Premack and Woodruff (1978). However, it has only been within the last decade that modeling ToM has received significant attention within the AI community.[1]

[1]This lag can be accounted for in two ways. First, AI research has had other priorities; it simply doesn't make sense to pursue ToM at level two or higher until the agents being worked with are able to carry out the basic sorts of tasks necessary to give ToM at these higher orders any relevance. Second, modeling ToM is a potentially computationally intensive task, requiring at least an arithmetical expansion of resources as each level is added. It has only

Despite the recent intersection of multi-agent systems and ToM the major focus of this research remains the cognitive capacities of the individual agents within the system rather than characteristics of the system itself. This is to be expected given that intelligence is still widely agreed to be localized within individual agents and since AI research is primarily carried out within this paradigm, attention to larger social consequences is understandably lacking.[2]

This leaves the broader social consequences of ToM open to exploration. The most important question to be answered is whether or not ToM is beneficial to populations and under what specific conditions (if any) this is the case. Given the benefits that it clearly has for individual agents it would seem that the answer to the first question is an obvious "Yes", but without further investigation such a claim is hopeful at best and a fallacy of composition at worst.[3] The work presented in this chapter was undertaken in order to begin answering this question. It proceeds by first setting aside an important philosophical debate that would unduly slow progress and then turns to an explication of the models and methods used in the simulations directed towards investigating the value of ToM for populations. In the final sections the results of these simulations are presented along with reflections on the consequences that are entailed.

## 6.2 Setting Aside a Philosophical Dispute

The rise of Theory of Mind as an explanation for the behavioural capacities of various sorts of creatures has been accompanied by a dispute among philosophers of mind. The essence of this dispute is the mechanism by which ToM actually arises and the catalyst is the appropriate way to explain the behaviour of children of various ages in the false belief task of Wimmer and Perner (1983). The most basic version of the false belief task has the child acting as the subject of the experiment observe a protagonist hide an object in one location

been relatively recently that computers have become cheap enough and plentiful enough to fully pursue this expansion.

[2]Investigations inspired by social insects such as ants and wasps that use masses of simple robots or agents to carry out complicated tasks are underway, but the simplicity of the agents used precludes anything approaching ToM.

[3]The capacity to outmaneuver other agents and take their resources is clearly beneficial to any *single* agent in the short term, but it is not at all clear that there are long term benefits or that any benefit would be had if this power was given to all the agents in the population.

and then, without the protagonist's awareness, observe the object moved to a different hidden location. The child is then asked where the protagonist will look for the object. In the original study, "None of the 3-4-year old, 57% of 4-6-year old, and 86% of 6-year old children pointed correctly" (Wimmer and Perner, 1983, p. 103) to the location where the object was originally hidden by the protagonist and further studies have provided similar results. These results are taken to show that there is a capacity that develops in children that allows them to attribute beliefs to other agents based on what they know about what the other agent knows. In short, sometime around 4 years of age children begin to develop ToM. Exactly how this ToM is manifest is an open question and the source of a divide.

On one side of the divide are the theory theorists, holding that ToM is a direct result of the development within children of a theory of just what it is to hold a belief and the conditions under which such beliefs may be true or false. Such a theory need not be formalized in any sense nor articulable by anyone who holds it. All that is required is that it exist in the mind in such a way that it can influence the beliefs, and ultimately the behaviour, of anyone with the capacity to form the theory.

The other side of the divide is occupied by the simulation theorists. The position they stake out is that the correct explanation for the ability of older children to succeed in the false belief task is not the rise of a theory of other minds or beliefs within children but the development of a capacity in children to suppose that they are in the situation of another being and reason under this supposition.

Volumes of material have been dedicated to both understanding this debate (Davies and Stone (1995a); Davies and Stone (1995b)) and defending one side or the other. Not knowing the exact mechanism by which children come to acquire and exercise ToM is a serious problem to further research only where the nature of the mechanism itself is at issue. Where the interest is in the consequences of ToM broadly construed, as is the case here, this lack of knowledge becomes less significant. In such cases it becomes permissible to "black box" the details of the inference making process and instead focus solely on the effect that results of including capacities that are sufficiently ToM-like within the populations, even if the methods used to program them are extremely simplistic to the point that their candidacy as acts of inference

would be questionable.[4] The effectiveness of simple models can be improved through the use of an evolutionary mechanism that puts selective pressure on those mechanisms that prove more beneficial for agents through an increased ability to survive and reproduce those mechanisms.

## 6.3 Details of the Model

In order to investigate some of the possible benefits from ToM a rudimentary ToM-like capacity was built into the agents described in Chapter 4 and already presented in Chapter 5. There exist at least two architectures dedicated specifically to modeling multi-agent systems where the agents have the capacity for ToM: PsychSim (Marsell et al., 2004) and CASE (Zhang et al., 2008). However, there are numerous general platforms that are capable of carrying out such simulations and the general practice is to custom build a simulation environment for the questions that are being asked.[5] None of these tools were used in the construction of the simulations that were used here for the reasons given in section 4.1.

The simplest models of ToM are discreet-state and limited to second level ToM. Discreet-state models are those that limit the possible number of perspectives that an agent can represent to a finite number such that all states must fall within one of the specified states when being modeled. This property, especially when the number of possible states are very few, makes agents easy to model and reduces the overall memory requirements on many architectures. Currently, discreet-state ToM is considered to be "state of the art" but there is at least one initiative to change this (Kim and Lipson, 2009). Second level ToM occurs when an agent not only takes account of its own internal states (first level ToM), but also the states of other agents as well. These acts of "taking account of other agents" amount to making observations about the behaviours and situations that these other agents either exhibit or can be found in and then inferring what their internal states are, and thus what their future actions will be, based on these observations. As the debate between simulation theorists and theory theorists makes clear, while it can be agreed that some sort of process must be taking place, the nature and details of this process

[4]The details of this black boxing are given in the following section.

[5]For example, the general PhysX simulation environment has been used by Kim and Lipson (2009) to build custom simulations.

remain unknown.

The ToM enabled agents within the model are capable of assessing relationships that exist between them and other agents in a non-trivial way. To accommodate this reality every agent is capable of assigning all the agents with whom they interact into one of three categories. These categories can be roughly understood as 'Friend', 'Neutral', or 'Enemy', but it is important to understand that how agents assign other agents to these categories may not match onto our understanding of these terms. Agents simply have three categories and treat all agents within each category the same. Whether they are 'nice' to all the agents in the 'Enemy' category and 'mean' to everyone in the 'Friend' category or vice versa is something that is not explicitly stipulated in the program. Further, being nice or mean is not necessarily implemented across all possible situations within each category. It is much more likely the case that various actions that we as observers might consider to be "nice" or "mean" are distributed evenly with fifty-fifty randomization across across each category during the start of each trial and so any attempt to anthropomorphize how agents like each other should be strongly resisted.

In addition to the basic relationship assessment that results from noting that people interact differently with other people depending on how they like them, it is also taken as obvious that people further adjust their behaviour towards others by taking into account how they believe other people see them. People do not often treat another person the same way if they believe that the other person hates them as opposed to believing that the other person likes them or at least has a more neutral stance towards them. As a mirror to the three stances that each agent might take towards another agent there are three stances that each agent supposes that the other agent might take towards them. Taking into account what has been said earlier about resisting the attraction of overly anthropomorphizing these agents these three stances can be understood as mapping onto: "I think that you think that I am a Friend", "I think that you think that I am a Neutral Party", and "I think that you think that I am an Enemy". These three stances, in combination with the three listed immediately prior, result in in nine different possible relationship types that each player can assess whenever a choice is to be made. Each $2\times2$ game that a player is capable of playing must have nine bits to describe how that game is to be played.

Agents respond to the outcomes brought about by their interaction with

other agents by reassessing their relationship with the agent they just interacted with. The ability to respond to outcomes simply means that after a game has been played an agent looks at what both he or she and the other player actually received versus what was available to be received. Based on this assessment the agent is capable of reassigning both the relationship category that they assigned the other agent to across the three available categories and the relationship category that they believe that the other agent assigned them to. It is by allowing agents to update their relationship assessments and their assessments of the assessments of other agents based on their interactions with these agents that they gain a rudimentary Theory of Mind. This behaviour is also intended to create the possibility of mimicking the behaviour of human beings who also respond to the outcomes that they experience. Reassignment to a different stance is not a necessity, only a possibility; not all outcomes lead people to change the way they see those with whom they interact.

The results of an assessment of outcomes is determined through a module called `AfterPlay`. The heart of the `AfterPlay` module is a three dimensional matrix, internal to each agent, that takes as index values an assessment of whether the agent doing the assessment did better, worse, or the same as some standard drawn from the structure of the 2×2 games being played, an assessment of whether the agent played against did better, worse, or the same as a standard also drawn from the game, and an assessment of the stance taken regarding the relationship between the two agents when they first entered the game. The standards vary between agent types and are one of the ways that the two agent types of rational agents modeled differ. This is certainly not a perfect mirror of the capabilities of actual human beings, but it is a necessary simplification at this early stage of building this model.

The matrix is used twice in the `AfterPlay` module by each agent. The first time is to see if the stance the assessing agent takes toward the other agent in the game will be different in the future. The second time is to see if the stance that the assessing agent believes the other agent in the game takes toward them will be different in the future. The assessments of better, worse, or no-change regarding the standards drawn from the game are used in both assessments and are simply switched from one assessment to the other to represent the change of perspectives. The third index, the stance taken from one agent to the other, changes between assessments. One matrix is

used for both assessments for two reasons. First, this is intended to be an extension of the notion of empathetic preferences. Agents assess the effect of outcomes on others, but must do so from their own view of the world. In short, they are doing what they would do if they were in the other agent's position and had just experienced the outcomes of the game just played. Second, it helps to keep the `AfterPlay` module reasonably simple. "Reasonably" is used loosely. As `AfterPlay` is currently implemented in SQ agents, the simplest of the two agent types, there are three ways that the relationship stance can be assessed by an agent, three ways that they can assess the stance from the perspective of the other agent, and three ways that the payoff received as an outcome of a game can be valued. This means that there are $3 \times 3 \times 3 = 27$ different possible cells within the `AfterPlay` matrix. Since each cell in the matrix can take on one of three possible values as well (switch to 'Friend', switch to 'Enemy', and switch to 'Neutral') there are $3^{27} \approx 7.6 \times 10^{12}$ uniquely different arrangements that can exist within the `AfterPlay` matrix for SQ agents. Normal/classic agents have ten ways of assessing outcomes, making it the case that there are $3^{300}$ uniquely different arrangements that can exist within the `AfterPlay` matrix for this more complicated agent class.

## 6.4 Basic Results of Introducing ToM

The results of introducing the ToM-like capacity into the agents as described above are significant. This can be seen most clearly by comparing Figures 6.1 and 6.2 from this chapter with Figures 5.1 and 5.2 from Chapter 5. The first thing to be noted is that introducing ToM has had a strong negative impact on population size and growth rates across both agent types. The second thing to note is that it has curtailed the ability of SQ agent populations to grow and survive much more extensively than NC agents.

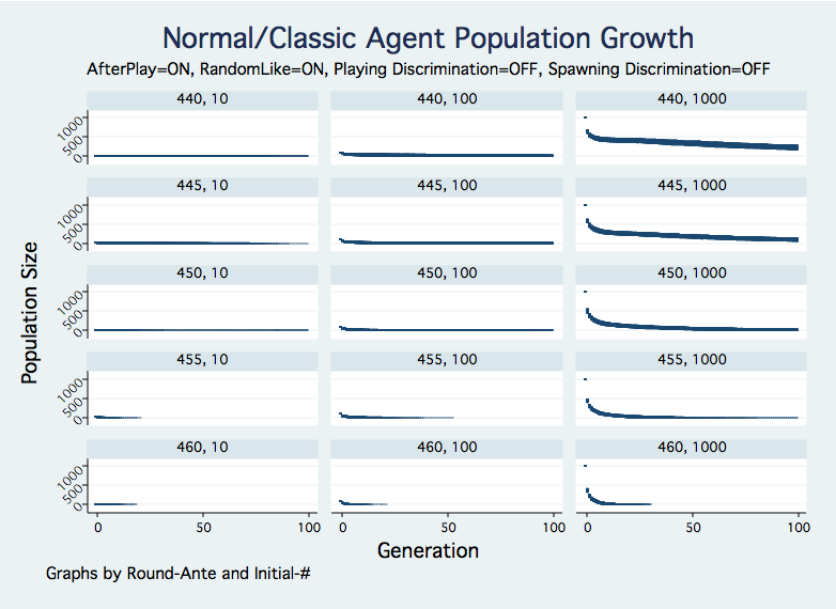


Figure 6.1: Population Growth for NC Agents by Generational Ante and Initial Population Size, AfterPlay=ON

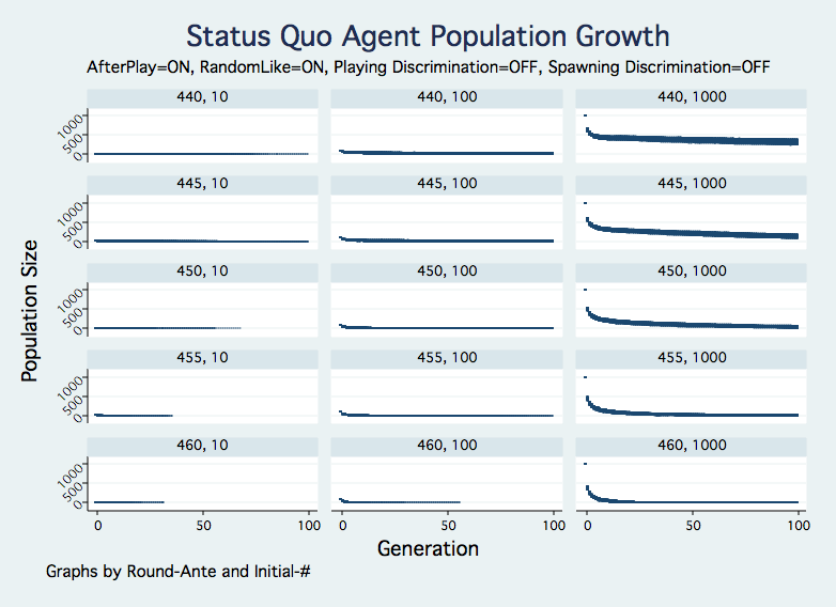


Figure 6.2: Population Growth for SQ Agents by Generational Ante and Initial Population Size, AfterPlay=ON

The ability of both agent types to grow useful game playing abilities within their members such that the overall average payoff across all the games played by all the agents in each generation has also fallen. This is shown in Figure 6.3 and it should be considered in comparison to Figure 5.3. In particular, note how the mean of the average payoff received for NC agents is now barely distinguishable from the line representing the mean of the average payoff offered and how the shape of the distributions of the average payoff from generation to generation for both agents now look identical in terms of shape. Other than a slight rise in the average payoff received in SQ populations over the generations and a slightly longer longevity, there would be no discernible difference between the performance of these two agent types at all.

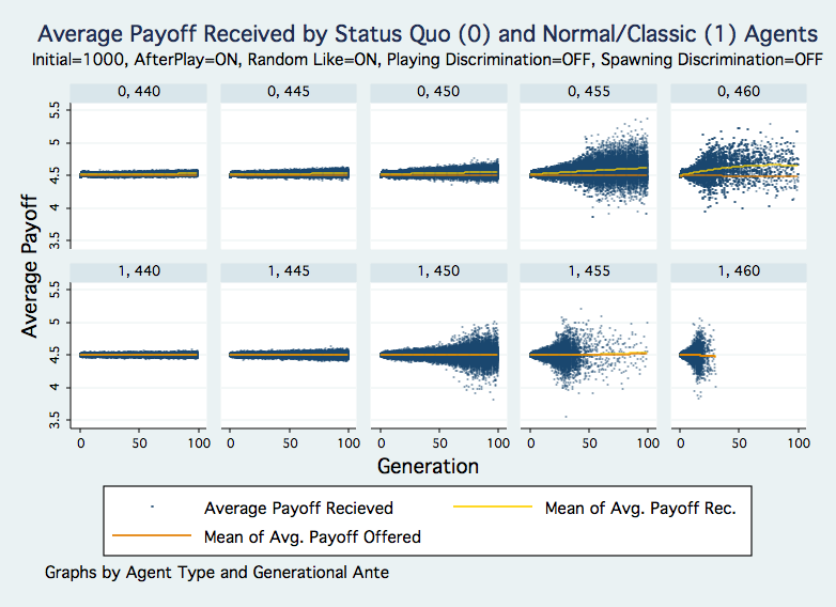


Figure 6.3: Average Payoffs Offered and Received by Agent Type and Generational Ante, AfterPlay=ON

The strongest plausible reason for this effect is the increased difficulty in coordinating actions that turning on `AfterPlay` presents to both agent types. Not only must agents be able to play games with the other agents in their population, but they are now in possession of the ability to treat different agents differently based on their assessment of their relationship with each individual agent they interact with. This relationship assessment is based on both their assessment of how they, the assessing agent, like the other agent *and* on how they, the assessing agent, have assessed how the other agent

they are interacting with have assessed the relationship. As mentioned in the conclusion to the preceding section, there is a very large number of possible `AfterPlay` matrices that an agent in either class might actually have and as a consequence coordinating between agents with different matrices becomes immediately problematic.

This is an important consequence for anyone advocating ToM because it shows that including the capacity for ToM in an agent or simulation has the potential to generate situations where the hoped for benefits are lost due to the loss of an ability to coordinate more directly.

### 6.4.1 Extended Results of Introducing ToM

The ToM like behaviour introduced here also gives agents the possibility of meaningfully discriminating during the play or spawning of a game in a way that is not obviously detrimental at the outset. `AfterPlay` is technically always active in every agent, but it is not randomized across the possible ways that an agent can assess a relationship until it is turned "on". Prior to this randomizaton `AfterPlay` is completely homogenous, making it the case that every single interaction with another agent results in the exact same relationship assessment as before the interaction took place. Allowing agents to discriminate on the basis of relationship assessments in the face of a homogenous `AfterPlay` would amount to setting up half the population for a mass suicide—since there is only one way for each agent to assess every relationship that they have with every other agent and the choice whether or not to play or spawn with these agents of a certain relationship is randomly set in each of the initial agents, this would make it the case that, on average, half of every population would simply refuse to play with anyone at all. Turning `AfterPlay` "on" prevents this result from happening by giving each agent nine different ways that they can assess the relationships that they have with other agents.

The initial reason for including the capacity for discrimination was that it would allow agents an additional means of punishing other agents that would further reflect the behaviours of actual human beings. It is often the case that people refuse to interact with other people in various ways and for various reasons, thus forcing agents to interact with which ever agent was presented to them at random is perhaps a little too contrived. It was also supposed that discrimination amongst players for the purposes of playing games or spawning

might allow agents to quickly expel poorly playing agents from the population, allowing those remaining to coordinate on an ideal set of behaviours more quickly than without. Unfortunately, as with the general introduction of a ToM-like behaviour in the first place, this did not happen.

Allowing discrimination during playing and spawning only proved to make it even harder for the agents within each population to survive, both as individuals as an entire population. This can be seen by examining Figures 6.4 and 6.5. These two charts show the effect of allowing each agent of each type 100 random matches to find a partner to play or to spawn with in the population before having the opportunity removed from them for either the remainder of the round (in the case of playing) or the generation (in the case of spawning). The populations of both agent types suffer severely under these conditions.

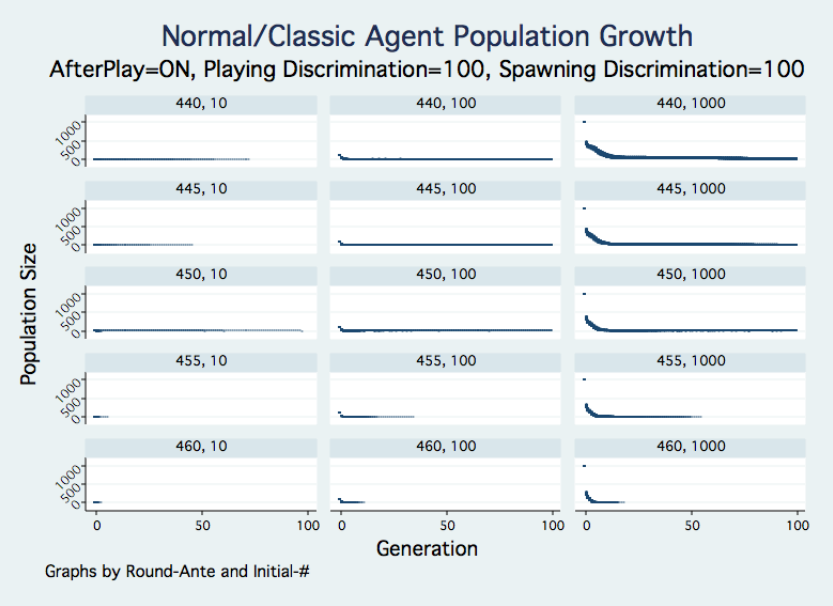


Figure 6.4: Population Growth for NC Agents by Generational Ante and Initial Population Size, AfterPlay=ON, Discrimination=ON

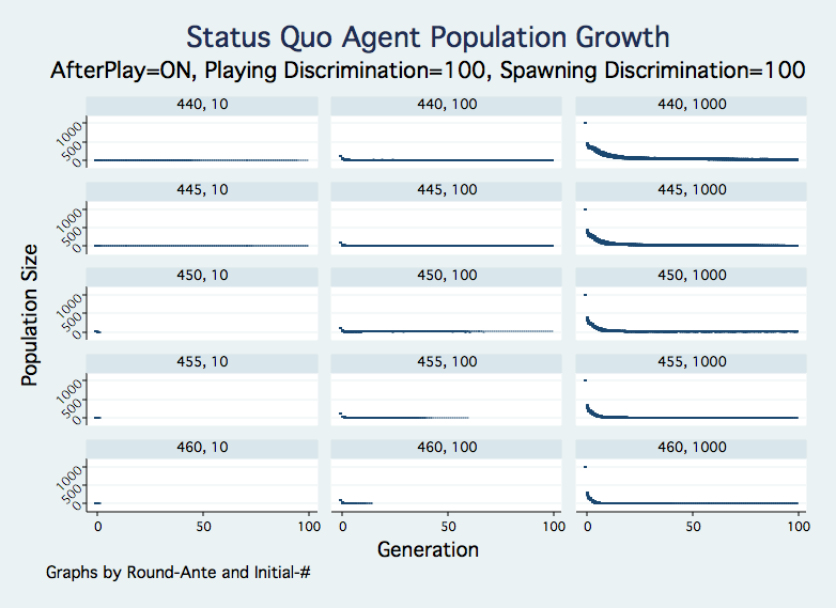


Figure 6.5: Population Growth for NC Agents by Generational Ante and Initial Population Size, AfterPlay=ON, Discrimination=ON

The consequences of these limitations are shown in the ability of either agent type to have their populations adapt to the games that they are playing every round by collecting payoffs that are above the average offered. These are displayed in Figure 6.6. While SQ agents continue to show their superiority the magnitude of the gap between them and populations composed entirely of NC agents has again been reduced. It is only at the harshest levels where the cost to live from round to round is 455 or 460 that the mean of the average payoff received rises noticeably above the mean of the average payoff offered for SQ agents. In these cases it is the loss of any population not capable of regularly performing this feat that allows this rise to occur at all. NC agents are not able to show a similar effect because they are not capable of surviving under these conditions at all, for the reasons suggested in Chapter 5.

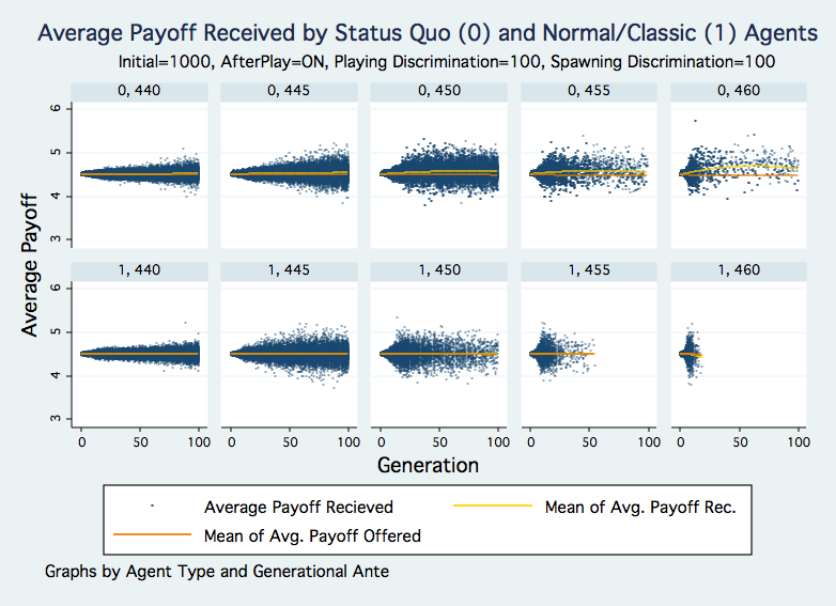


Figure 6.6: Average Payoffs Offered and Received by Agent Type and Generational Ante, AfterPlay=ON

Are these the results necessary or inevitable in any similar or generalized case? No, they are not. There exists a myriad of possible variations to the model used here, each of which may stand to reverse or mitigate the results that have been shared. This said, for every small change that might result in the salvation of these populations there are thousands of others that stand to simply interfere with the capacity of the agents in question to interact in an effective manner. That actual organisms are capable of interacting in fruitful

ways with each other at all, despite both their generally complicated natures and an environment that is far from Eden, is a testament to the refined set of capacities and powers that these organisms must possess and the power of evolution to produce adaptive agents. That the powers observed in these agents cannot simply be dropped into new agents in new contexts and expected to perform at similar levels of efficiency is a warning that any would be modeler must take heed of.

# Appendix A

# 726 2×2 Games: Case-by-Case Proof

Before proving that there are exactly 726 2×2 by dividing the total number of games in to cases and then counting the number of possibilities within each case a few properties that games possess will be outlined and terminology that will be used in the proof explained.

First, the 'structure of a game' should be understood as being entirely composed of the relationships that each of the payoffs associated with the outcomes hold to each other across each of the players. Each game has only one structure, which amounts to the blueprint or complete genetic description of that game. The game can be represented in any way whatsoever just as long as the structure is maintained across these representations. Duplications of a game across representations are not meaningful since only one representation is needed to define the game. For this reason much of the proof that there are 726 2×2 games amounts to identifying representations that duplicate game structures and removing them

References to a 'payoff structure of a player' will also be made. A player's payoff structure differs from the structure of a game by capturing only the preference relationships between the outcomes for the player being considered. It is the specific ways that the payoff structures for each of the players involved in a game intersect that defines the structure of each game, and thus the entire set of games.

The method of representation used here will be referred to as *minimal normal form*. This style of representation is characterized by presenting the game in a two-dimensional matrix format. The options available to one player

are represented by the rows, while the options available to the other player are represented by the columns. The intersection of each player selecting a row or a column generates a cell which amounts to an outcome of the game. Inside each outcome is a listing of the payoffs available to each of the players should the outcome be realized. These payoffs should be understood as representing preferences on an increasing ordinal scale, meaning that higher values are to be preferred to lower values. Since the scale is ordinal it is not possible to speak of magnitudes of preference, all that can be said is that two outcomes are either preferred, dispreferred, or equally preferred to each other. Consequently certain presentations of the payoffs available are not meaningfully different. Consider the games in Figure 2.

| | |
|---|---|
| 3, 2 | 0, 3 |
| 2, 0 | 1, 1 |

(a)

| | |
|---|---|
| 7, 2 | 0, 3 |
| 2, 0 | 1, 1 |

(b)

*Figure 2*

Games (a) and (b) are the same even though the payoff to the row player represented by the outcome in the top left cell is a 3 in (a) and a 7 in (b). Payoffs are on an ordinal scale and so both of these values carry the same information in this game, namely, the information that "the row player prefers this outcome above all others." To prevent the temptation to interpret the payoffs as providing anything beyond ordinal information all the payoffs presented will make use of the smallest integers possible with zero as a lowest value.

Ordinal payoffs are important for two reasons. First, they make it easier to recognize general structures that are present in games. Second, without the use of ordinal payoffs to define games and their structures, classifying games would become an intractable proposition, as even the most minute changes in the value of a payoff would generate a new game (This is not to say that cardinal valuations and human psychology do not affect games, see footnote).[1]

[1]Someone might ask how game theory is meant to accommodate the fact that people change their behaviour depending on the magnitudes of the payoffs they will receive, such as monetary gain or loss. Such a question should be answered by explaining that things like money are simply one factor contributing to a player's evaluation of the game. Other possible factors include, but are certainly not limited to, timeliness, risk, who else is playing, and the impact the decision will have on other people. It is only once all such factors are taken into account that the game can be defined and, so defined, represented in minimal normal form. Changing what players receive across their outcomes thus has the possibility

By convention the first payoff listed in each outcome is the payoff to the player who chooses a row with the remaining payoff going to the player who chooses a column. In the proof that follows Player A should be understood as the row player and Player B as the column player. Note that who is the row player or column player does not matter for the proof. Nor does it matter that the payoff structures assigned to the row player were not the payoff structures assigned to the column player and vice versa. All such assignments were arbitrary. What matters is the structure of the game (as defined), regardless of representational factors like player names and who gets to be the row or the column player.

| | | Player B | |
|---|---|---|---|
| | | $x_B$ | $y_B$ |
| Player A | $x_A$ | $\pi_A(x_A, x_B)$, $\pi_B(x_A, x_B)$ | $\pi_A(x_A, y_B)$, $\pi_B(x_A, y_B)$ |
| | $y_A$ | $\pi_A(y_A, x_B)$, $\pi_B(y_A, x_B)$ | $\pi_A(y_A, y_B)$, $\pi_B(y_A, y_B)$ |

Figure 3

The game represented in Figure 3 should be understood as acting as a base of reference when the payoff structures for the players are considered. On this representation $\pi_A(x_A, x_B)$ is the payoff to Player A at the outcome arrived at when Player A chooses option $x_A$ and Player B chooses option $x_B$. Note that option $x_A$ and option $x_B$ are not necessarily equivalent. For example, $x_A$ may

of changing the game they are playing, not how they play a particular game.

For example, there are numerous experiments using iterated Prisoner's Dilemmas to show that under certain conditions cooperation becomes the rational thing to do. Important to note is that the moment any game is iterated it stands to become a different game. Iterated Prisoner's Dilemmas have a different structure than non-iterated Prisoner's Dilemmas and thus are *not* Prisoner's Dilemmas. Cases that purport to show that cooperation frequently happens in one-shot Prisoner's Dilemmas are very likely the result of one of two things, and possibly both. First, it is quite possible that the people involved simply made a mistake or a series of mistakes as a result of misunderstanding the situation. Second, it is almost impossible to control each situation to ensure that it really is a Prisoner's Dilemma that is being played and not something else. Other games are entirely possible as the player's import into the trial their own past histories, expectations, senses of fairness, and personal idiosyncratic preferences regarding whatever the researchers are offering in the way of payoffs. Such imports stand to change the game being played into something other than intended.

In short, context matters and there is no way to avoid context. The realization of any game is entirely contextual and in such a way that we can never know exactly what that context is—there are no *pure* Prisoner's Dilemmas in the world that we can knowingly observe, only approximations under various qualifications. This is true for every game. This truth is not as detrimental as it may first appear. There are also no frictionless pendulums or perfect expressions of the ideal gas law, but such uses of theory are still informative.

refer to "turn left" and $x_B$ to "buy". Equivalence is not even necessary in cases where there is symmetry in distribution of the payoffs between the players.

Symmetry between the payoffs available to the players results when the game being played is such that neither player has a preference regarding which role, row or column, they are assigned when playing the game. More formally, symmetry of this sort exists when one of the following sets of conditions is satisfied:[2]

$$\pi_A(x_A, x_B) \sim \pi_B(x_A, x_B)$$
$$\pi_A(x_A, y_B) \sim \pi_B(y_A, x_B)$$
$$\pi_A(y_A, x_B) \sim \pi_B(x_A, y_B)$$
$$\pi_A(y_A, y_B) \sim \pi_B(y_A, y_B)$$

OR

$$\pi_A(x_A, y_B) \sim \pi_B(x_A, y_B)$$
$$\pi_A(x_A, x_B) \sim \pi_B(y_A, y_B)$$
$$\pi_A(y_A, y_B) \sim \pi_B(x_A, a_B)$$
$$\pi_A(y_A, x_B) \sim \pi_B(y_A, x_B)$$

Other types of symmetry within games are possible, but it is symmetry of the payoffs across the players that will be meant if "symmetry" is used without further qualification.

This proof proceeds by an exhaustive examination of fifteen cases that cover all the possible ways that two players might have preference equivalences distributed over the four possible outcomes. In each case the number of

[2]Note the use of "$\sim$" where a "$=$" might be expected. This is done because a comparison between players and their preferences is taking place. What matters for defining symmetry of this sort is that the rankings of the indicated outcomes be the same on the ordinal scale across the players. However, it is not necessary for the players to actually maintain these preferences over the outcomes were they somehow to become the other player. The sort of symmetry being described here is a property of the structure of a game in a very formalized sense, raising the possibility that many games that are formally symmetric are not practically symmetric because at least one of the players would prefer to be playing in the role of another player.

For example, it is possible that ranking the preferences of two people over the combined outcomes of a choice each is facing between two options may show that the game has the structure of a symmetric game like a Prisoner's Dilemma, Stag Hunt, or Battle of the Sexes. However, it does not follow that were the two players each offered the opportunity to exchange roles with the other that they would be indifferent to doing so. It might very well be the case that one or more players of a game, even a symmetric one, might very well prefer to be one of the other players. Games are defined by the structure of the players' preferences across the outcomes. Since this structure is a result of the players' preferences over possible states of affairs, any change to what is possible may change their preferences and thus the game being played. This is as true for non-symmetric games as symmetric ones.

meaningfully different arrangements of payoffs satisfying the structures specified for each player are counted and combined to determine the total number of games in each case. Payoff structures for a player are defined by placing conditions on the number of outcomes that the player is allowed to value equally. The conjunction of two payoff structures creates a case.

The following argument proves that there are exactly fifteen cases. In any 2×2 game there are four possible outcomes that a player must compare. The total number of comparisons that can be made is $\binom{4}{2} = 4! \div 2! \times 2!$=6 (cases where outcomes are compared to themselves are ignored). Of these six possible comparisons there are five possible forms in which equivalencies might arise when comparing outcomes, as follows:

1. All the outcomes are seen as equivalent.
2. Three outcomes are judged as being equivalent.
3. Two pairs of outcomes each have the outcomes within them judged as being equivalent, but no member of a pair is seen as being equivalent to the member of any other pair.
4. One pair and only one pair of outcomes is judged as being equivalent.
5. No outcomes are judged as being equivalent.

Since there are five ways that that equivalencies might be distributed by a single player across the outcomes the next step is to figure out how many ways that these might be distributed across two players. This is a matter of determining how many pairs can be generated from five items when repetition is allowed. The number of combinations of $k$ objects that can be generated from a total of $n$ objects when allowing repetition is $\binom{n+k-1}{k}$. This means that there are $\binom{5+2-1}{2} = \binom{6}{2} = 6! \div 2! \times 4! = 15$ total cases to be considered.

Each case is numbered. The number in the square brackets following the announcement of a case is the total number of games held within that case.

# A.1 The Proof

## A.1.1 Case 1 [1]

**Player A Payoff Structure**

$$\pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$

**Player B Payoff Structure**

$$\pi_B(x_A, x_B) = \pi_B(x_A, y_B) = \pi_B(y_A, x_B) = \pi_B(y_A, y_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 0, 0 | 0, 0 |
| 0, 0 | 0, 0 |

There is clearly only one such structure in minimal normal form.

### A.1.2 Case 2 [2]

**Player A Payoff Structure**

$$\pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$

**Player B Payoff Structure**

*One and only one of the following is true:*

$$\pi_B(x_A, x_B) \neq \pi_B(x_A, y_B) = \pi_B(y_A, x_B) = \pi_B(y_A, y_B)$$
$$\pi_B(x_A, y_B) \neq \pi_B(x_A, x_B) = \pi_B(y_A, x_B) = \pi_B(y_A, y_B)$$
$$\pi_B(y_A, x_B) \neq \pi_B(x_A, y_B) = \pi_B(x_A, x_B) = \pi_B(y_A, y_B)$$
$$\pi_B(y_A, y_B) \neq \pi_B(x_A, y_B) = \pi_B(y_A, x_B) = \pi_B(x_A, x_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 0, 1 | 0, 0 |
| 0, 0 | 0, 0 |

There is only one possible payoff structure for Player A in Minimal Normal Form. Player B has two possible structures, one where the triad of equivalent outcomes is preferred to the fourth and one where the fourth is preferred to the triad. It follows directly that there are only two games of this type.

### A.1.3 Case 3 [3]

**Player A Payoff Structure**

$$\pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$

**Player B Payoff Structure**

*One and only one of the following is true:*

$$\pi_B(x_A, x_B) = \pi_B(x_A, y_B), \pi_B(y_A, x_B) = \pi_B(y_A, y_B), \pi_B(x_A, x_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, x_B), \pi_B(x_A, y_B) = \pi_B(y_A, y_B), \pi_B(x_A, x_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, y_B), \pi_B(x_A, y_B) = \pi_B(y_A, x_B), \pi_B(x_A, x_B) \neq \pi_B(x_A, y_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 0, 1 | 0, 1 |
| 0, 0 | 0, 0 |

There is only one possible payoff structure for Player A in Minimal Normal Form. Player B has three possible payoff structures. One where they can guarantee a most preferred outcome, one where receiving a most preferred outcome is entirely in the hands of the other player and one where their receiving a most preferred outcome amounts to a 50/50 gamble. As such there are only three games of this type.

### A.1.4 Case 4 [9]

**Player A Payoff Structure**

$$\pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$

**Player B Payoff Structure**

*One and only one of the following is true:*

$$\pi_B(x_A, x_B) = \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, x_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, y_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 0, 2 | 0, 1 |
| 0, 0 | 0, 0 |

There is only one possible payoff structure for Player A in Minimal Normal Form. Player B faces a structure where they are indifferent between two and only two outcomes, the relations of this pair to the valuation of the remaining two can be anything at all. This structure demands three possible values each of which can be arranged in one of three possible ways. This makes nine possible games of this type.

### A.1.5 Case 5 [6]

**Player A Payoff Structure**

$$\pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$

**Player B Payoff Structure**

$$\pi_B(x_A, x_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B) \neq \pi_B(y_A, y_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 0, 3 | 0, 1 |
| 0, 2 | 0, 0 |

There is only one possible payoff structure for Player A in Minimal Normal Form. Player B has no equalities standing between the available payoffs making it the case that four distinct values are required to cover all the possible outcomes. There are $4! = 24$ ways that four unique items can be arranged to fill four spaces. However, many of these arrangements are not meaningfully distinct from each other, amounting only to different representations of the same game. Any game represented in normal form can have the rows swapped or the columns swapped and still represent the same game. Each possible swap reduces the number of unique game candidates by half. There are thus $24/(2 \times 2) = 6$ unique payoff structures that Player B can face and 6 possible games of this type.

### A.1.6 Case 6 [10]

**Player A Payoff Structure**

*One and only one of the following is true:*

$$\pi_A(x_A, x_B) \neq \pi_A(x_A, y_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$
$$\pi_A(x_A, y_B) \neq \pi_A(x_A, x_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$
$$\pi_A(y_A, x_B) \neq \pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, y_B)$$
$$\pi_A(y_A, y_B) \neq \pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, x_B)$$

**Player B Payoff Structure**

*One and only one of the following is true:*

$$\pi_A(x_A, x_B) \neq \pi_A(x_A, y_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$
$$\pi_A(x_A, y_B) \neq \pi_A(x_A, x_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$
$$\pi_A(y_A, x_B) \neq \pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, y_B)$$
$$\pi_A(y_A, y_B) \neq \pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, x_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 1, 1 | 0, 0 |
| 0, 0 | 0, 0 |

Player A and Player B each have two possible structures, one where the triad of equivalent outcomes is preferred to the fourth and one where the fourth is preferred to the triad. Now that Player A is no longer completely indifferent, the orientation of preferences across outcomes over the two players now matters in determining the total number of games. Each player has four possible locations that the odd payoff out might be located and two possible values that payoff can take (it can either be the largest or the smallest payoff) for eight possible variations. Combining these variations from the two players generates 64 candidate structures. There is a great deal of duplication throughout these candidates, allowing reductions to be made.

The largest reduction comes from recalling that any normal form game can be modified by swapping rows or columns without changing the game. This property allows any of the games under consideration here to be represented in at least four different ways. Consequently there are only $64/4 = 16$ candidates to be concerned with.

The next reduction comes from noting that there are only four possible payoff structures that are symmetric across players. This is true because for symmetry to exist across the players the payoff structure of each must appropriately reflect the other's such that the players would be indifferent between being assigned either role (row or column) within the game. This can only happen with this game if the odd payoff out for each player is the same and these odd payoffs are located in either the same outcome or in diagonally opposed outcomes. Since there are only two values that the odd payoff can take and two possible combinations of location there are $2 \times 2 = 4$ games with symmetry across the players. Setting these aside we are left with twelve games that are non-symmetric across the players. This means that there is duplication within this set since any non-symmetric game can undergo an additional transformation to produce a new arrangement of payoffs that is not meaningfully different from the original arrangement because it describes the same game. We can thus cut the number of non-symmetric games in half, leaving $12/2 = 6$ non-symmetric games. Taking the sum of symmetric and non-symmteric games shows that there are $4 + 6 = 10$ games of this type.

### A.1.7 Case 7 [12]

**Player A Payoff Structure**

*One and only one of the following is true:*

$$\pi_A(x_A, x_B) \neq \pi_A(x_A, y_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$
$$\pi_A(x_A, y_B) \neq \pi_A(x_A, x_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$
$$\pi_A(y_A, x_B) \neq \pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, y_B)$$
$$\pi_A(y_A, y_B) \neq \pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, x_B)$$

**Player B Payoff Structure**

*One and only one of the following is true:*

$$\pi_B(x_A, x_B) = \pi_B(x_A, y_B), \pi_B(y_A, x_B) = \pi_B(y_A, y_B), \pi_B(x_A, x_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, x_B), \pi_B(x_A, y_B) = \pi_B(y_A, y_B), \pi_B(x_A, x_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, y_B), \pi_B(x_A, y_B) = \pi_B(y_A, x_B), \pi_B(x_A, x_B) \neq \pi_B(x_A, y_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 1, 1 | 0, 1 |
| 0, 0 | 0, 0 |

Player A has two possible structures, one where the triad of equivalent outcomes is preferred to the fourth and one where the fourth is preferred to the triad. Player B has three possible payoff structures. One where they can guarantee a most preferred outcome, one where receiving a most preferred outcome is entirely in the hands of the other player and one where their receiving a most preferred outcome amounts to a 50/50 gamble. For each of the structures that Player B might be faced with there are four possible ways that each of the four payoffs available to Player A can be arranged. Since each player has a different payoff structure there can be no games that are symmetric. Consequently there are $4 \times 3 = 12$ games represented in this case by minimum normal form.

### A.1.8 Case 8 [72]

**Player A Payoff Structure**

*One and only one of the following is true:*

$$\pi_A(x_A, x_B) \neq \pi_A(x_A, y_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$
$$\pi_A(x_A, y_B) \neq \pi_A(x_A, x_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$
$$\pi_A(y_A, x_B) \neq \pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, y_B)$$
$$\pi_A(y_A, y_B) \neq \pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, x_B)$$

**Player B Payoff Structure**

*One and only one of the following is true:*

$$\pi_B(x_A, x_B) = \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, x_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, y_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 1, 2 | 0, 1 |
| 0, 0 | 0, 0 |

Player A has two possible structures, one where the triad of equivalent outcomes is preferred to the fourth and one where the fourth is preferred to the triad. Player B faces a structure where they are indifferent between two and only two outcomes, the relations of this pair to the valuation of the remaining two can be anything at all. For each payoff structure available to A there are two ways that the payoffs might be arranged making $4 \times 2 = 8$ arrangements in total. For each of the payoff structures available to B there are three ways that the payoffs might be arranged making $3 \times 3 = 9$ arrangements in total. Since each player has a different payoff structure there can be no games that are symmetric. There are thus $8 \times 9 = 72$ games that can be represented in this case by minimum normal form.

### A.1.9 Case 9 [48]

**Player A Payoff Structure**

*One and only one of the following is true:*

$$\pi_A(x_A, x_B) \neq \pi_A(x_A, y_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$
$$\pi_A(x_A, y_B) \neq \pi_A(x_A, x_B) = \pi_A(y_A, x_B) = \pi_A(y_A, y_B)$$
$$\pi_A(y_A, x_B) \neq \pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, y_B)$$
$$\pi_A(y_A, y_B) \neq \pi_A(x_A, x_B) = \pi_A(x_A, y_B) = \pi_A(y_A, x_B)$$

**Player B Payoff Structure**

$$\pi_B(x_A, x_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B) \neq \pi_B(y_A, y_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 1, 3 | 0, 1 |
| 0, 2 | 0, 0 |

Player A has two possible structures, one where the triad of equivalent outcomes is preferred to the fourth and one where the fourth is preferred to the triad. Player B is not indifferent between any of the outcomes. Player B thus has four possible values which can be arranged in $4! = 24$ possible ways. For any arrangement across the outcomes that can be made there are four possible variations that can be made by swapping rows or columns. The results of these swaps are identical in meaning to the original because they do not change the ways that Player B's payoffs across the outcomes relate to each other. Since each relationship between Player B's payoffs can be represented in four different ways there are actually only $24/4 = 6$ arrangements that need to be examined. For each of these arrangements there are eight possible ways that Player A's payoff structure can be arranged; two possible structures that can be represented with the odd payoff out belonging to each of the four possible payoffs. Since each player has a different payoff structure there can be no games that are symmetric. With eight possible ways of filling in the payoffs to Player A for each of Player B's six unique representations there are thus $8 \times 6 = 48$ games that can be represented in this case by minimum normal form.

### A.1.10 Case 10 [8]

**Player A Payoff Structure**

*One and only one of the following is true:*

$$\pi_A(x_A, x_B) = \pi_A(x_A, y_B), \pi_A(y_A, x_B) = \pi_A(y_A, y_B), \pi_A(x_A, x_B) \neq \pi_A(y_A, y_B)$$
$$\pi_A(x_A, x_B) = \pi_A(y_A, x_B), \pi_A(x_A, y_B) = \pi_A(y_A, y_B), \pi_A(x_A, x_B) \neq \pi_A(y_A, y_B)$$
$$\pi_A(x_A, x_B) = \pi_A(y_A, y_B), \pi_A(x_A, y_B) = \pi_A(y_A, x_B), \pi_A(x_A, x_B) \neq \pi_A(x_A, y_B)$$

**Player B Payoff Structure**

*One and only one of the following is true:*

$$\pi_B(x_A, x_B) = \pi_B(x_A, y_B), \pi_B(y_A, x_B) = \pi_B(y_A, y_B), \pi_B(x_A, x_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, x_B), \pi_B(x_A, y_B) = \pi_B(y_A, y_B), \pi_B(x_A, x_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, y_B), \pi_B(x_A, y_B) = \pi_B(y_A, x_B), \pi_B(x_A, x_B) \neq \pi_B(x_A, y_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 1, 1 | 0, 1 |
| 1, 0 | 0, 0 |

Player A and Player B each have three possible payoff structures, one where they can guarantee a most preferred outcome, one where receiving a most preferred outcome is entirely in the hands of the other player and one where receiving a most preferred outcome amounts to a 50/50 gamble. The payoff structures that place the equally preferred outcomes on the diagonals can be combined in two meaningfully different ways. All possible arrangements of these payoff structures between the players reduce to a case where either both players always agree which outcomes are most preferred or where no such agreement can be made. The payoff structure that places the equally preferred outcomes on the diagonal for one player can be combined in only one meaningfully different way with each of the non-diagonally oriented options from the other player, for a total of two games. Finally, the non-diagonally oriented payoff structures from each player can be combined in four possible ways. This results in there being $2 + 2 + 4 = 8$ games in this case.

### A.1.11 Case 11 [54]

**Player A Payoff Structure**

*One and only one of the following is true:*

$$\pi_A(x_A, x_B) = \pi_A(x_A, y_B), \pi_A(y_A, x_B) = \pi_A(y_A, y_B), \pi_A(x_A, x_B) \neq \pi_A(y_A, y_B)$$
$$\pi_A(x_A, x_B) = \pi_A(y_A, x_B), \pi_A(x_A, y_B) = \pi_A(y_A, y_B), \pi_A(x_A, x_B) \neq \pi_A(y_A, y_B)$$
$$\pi_A(x_A, x_B) = \pi_A(y_A, y_B), \pi_A(x_A, y_B) = \pi_A(y_A, x_B), \pi_A(x_A, x_B) \neq \pi_A(x_A, y_B)$$

**Player B Payoff Structure**

*One and only one of the following is true:*

$$\pi_B(x_A, x_B) = \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, x_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, y_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 1, 2 | 1, 0 |
| 0, 1 | 0, 0 |

Player A has three possible payoff structures, one where the most preferred outcome can be guaranteed, one where receiving a most preferred outcome is entirely in the hands of the other player and one where receiving a most preferred outcome amounts to a 50/50 gamble. Player B has three possible

structures, each of which can be filled in three ways by three values, for nine possible arrangements. With the two ways that each of Player A's payoff structures can be filled there are $2 \times 3 = 6$ possible arrangements for Player A. Since each player has a different payoff structure there can be no games that are symmetric. The total number of minimal normal forms of games of this type is $6 \times 9 = 54$.

### A.1.12 Case 12 [36]

**Player A Payoff Structure**

*One and only one of the following is true:*

$$\pi_A(x_A, x_B) = \pi_A(x_A, y_B), \pi_A(y_A, x_B) = \pi_A(y_A, y_B), \pi_A(x_A, x_B) \neq \pi_A(y_A, y_B)$$
$$\pi_A(x_A, x_B) = \pi_A(y_A, x_B), \pi_A(x_A, y_B) = \pi_A(y_A, y_B), \pi_A(x_A, x_B) \neq \pi_A(y_A, y_B)$$
$$\pi_A(x_A, x_B) = \pi_A(y_A, y_B), \pi_A(x_A, y_B) = \pi_A(y_A, x_B), \pi_A(x_A, x_B) \neq \pi_A(x_A, y_B)$$

**Player B Payoff Structure**

$$\pi_B(x_A, x_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B) \neq \pi_B(y_A, y_B)$$

**Sample Game in Minimal Normal Form**

Player A has three possible payoff structures, one where the most preferred outcome can be guaranteed, one where receiving a most preferred outcome is entirely in the hands of the other player and one where receiving a most preferred outcome amounts to a 50/50 gamble. Each of these three structures can have one of two possible payoff distributions, and the odd payoff can either be the largest or the smallest payoff for a total of $3 \times 2 = 6$ payoff distributions. Player B is indifferent between none of the possible outcomes. Since there are four outcomes, there are four different values that are required to provide payoff information to Player B. These four values can be arranged in $4! = 24$ ways. Many of these arrangements do not pick out meaningfully different arrangements given the possibility of row and column swapping, something which has inflated this number to four times the actual number of unique arrangements. There are thus $24/4 = 6$ arrangements to be concerned with for Player B. Combining the unique distributions of Player A and Player B generates $6 \times 6 = 36$ possible games.

### A.1.13 Case 13 [171]

**Player A Payoff Structure**

*One and only one of the following is true:*

$$\pi_A(x_A, x_B) = \pi_A(x_A, y_B) \neq \pi_A(y_A, x_B) \neq \pi_A(y_A, y_B)$$
$$\pi_A(x_A, x_B) = \pi_A(y_A, x_B) \neq \pi_A(x_A, y_B) \neq \pi_A(y_A, y_B)$$
$$\pi_A(x_A, x_B) = \pi_A(y_A, y_B) \neq \pi_A(x_A, y_B) \neq \pi_A(y_A, x_B)$$

**Player B Payoff Structure**

*One and only one of the following is true:*

$$\pi_B(x_A, x_B) = \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, x_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, y_B)$$
$$\pi_B(x_A, x_B) = \pi_B(y_A, y_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 2, 2 | 0, 1 |
| 1, 0 | 0, 0 |

Both players have three possible structures that each require three values. These three values can be arranged in $3! = 6$ different ways within each structure. Within each player there are thus 3x6=36 possible arrangements across the three available structures. Considering all the permutations or arrangements between players generates $36^2 = 1296$ possible arrangements. There is duplication within this set. Symmetric games are currently over represented by four times, representing the four possible presentations of each on row and column swaps. Non-symmetric games are over represented by eight times, suffering from the same row and column swap inflation plus duplication due to the lack of symmetry between the players. Symmetric games occur when a payoff structure for Player A matches a payoff structure for Player B along with a payoff distribution across the structures that would leave fully rational agents indifferent regarding which player they became. For each of these pairs of structures there are $3! = 6$ ways that payoffs might be distributed such that symmetry is maintained. There are three structures so there is a total of $3 \times 6 = 18$ symmetric games. Subtracting four times this number from the total number of permutations returns the number of permutations of non-symmetric games, of which there are $1296 - (18 \times 4) = 1224$. Knowing that there are eight times more non-symmetric games than there should be allows us to determine the exact number of non-symmetric games of this type

to be $1224/8 = 153$. Summing the number of symmetric and non-symmetric games gives $18 + 153 = 171$ games in minimal normal form.

### A.1.14 Case 14 [216]

**Player A Payoff Structure**

*One and only one of the following is true:*

$$\pi_A(x_A, x_B) = \pi_A(x_A, y_B) \neq \pi_A(y_A, x_B) \neq \pi_A(y_A, y_B)$$
$$\pi_A(x_A, x_B) = \pi_A(y_A, x_B) \neq \pi_A(x_A, y_B) \neq \pi_A(y_A, y_B)$$
$$\pi_A(x_A, x_B) = \pi_A(y_A, y_B) \neq \pi_A(x_A, y_B) \neq \pi_A(y_A, x_B)$$

**Player B Payoff Structure**

$$\pi_B(x_A, x_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B) \neq \pi_B(y_A, y_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 2, 3 | 1, 1 |
| 0, 2 | 0, 0 |

| | |
|---|---|
| 1, 3 | 1, 1 |
| 0, 2 | 0, 0 |

Player A has three possible structures that each require three values. Each structure can have the payoffs ordered within it in three different ways. There are three different structures and so $3 \times 3 = 9$ different payoff arrangements. Player B cannot have any outcomes valued in the same way. Since there are four outcomes there are $4! = 24$ possible arrangements. Since there is no symmetry between the players each of the nine arrangements for Player A can be matched to each of the twenty-four arrangements for Player B to create a unique game and so there are $9 \times 24 = 216$ Minimal Normal Form games of this type.

### A.1.15 Case 15 [78]

**Player A Payoff Structure**

$$\pi_A(x_A, x_B) \neq \pi_A(x_A, y_B) \neq \pi_A(y_A, x_B) \neq \pi_A(y_A, y_B)$$

**Player B Payoff Structure**

$$\pi_B(x_A, x_B) \neq \pi_B(x_A, y_B) \neq \pi_B(y_A, x_B) \neq \pi_B(y_A, y_B)$$

**Sample Game in Minimal Normal Form**

| | |
|---|---|
| 2, 3 | 1, 2 |
| 3, 1 | 0, 0 |

There are $4! = 24$ ways that each player may have their payoffs arranged. Combining the possible arrangements for each player produces $24^2 = 576$ arrangements that need to be considered. This set of arrangements is larger than necessary by a magnitude of four since all the possible row and column swaps are currently included and so the number of can be reduced to $576 \div 4 = 144$. This is still inflated and can be reduced further.

For a single player there are $4! = 24$ ways to arrange their preferences across the outcomes. But since this does not take into account row or column swaps there are actually $24 \div 4 = 6$ ways that uniquely arrange a player's preferences. For each of these arrangements there are two ways that the preferences of the second player may be distributed across the outcomes to generate a symmetric game. There are thus $6 \times 2 = 12$ symmetric games for this case.

Symmetric games cannot produce a new arrangement when the payoffs are inverted for the players along the axis of symmetry. Since row and column swaps have already been eliminated there are not further reductions possible with the games in this case that are symmetric. The $144 - 12 = 132$ arrangements that remain are not symmetric and so are being double counted. This means that there are $132 \div 2 = 66$ non-symmetric games that can be represented by various preference arrangements over the outcomes without creating duplicates. Combining the symmetric and non-symmetric games shows that there are exactly $12 + 66 = 78$ games in this case.

# Appendix B

# 726 2×2 Games: Brute Force Proof

As Chapter 2 makes clear, it is not easy to offer a program or simulation as a proof because it is often unclear exactly how the program or proof was constructed. One way to mitigate this issue is to provide the original code used to generate the result. This is what is done here with respect to the program used to provide a brute force proof that there are exactly 726 meaningfully different 2×2 games under the limitations provided in Chapter 3. To produce this verbatim reproduction without splitting lines, which would thereby change the formatting from the original and lead to confusion, a small font has been used to contain each line as it is in the original.

```
/* This program considers all possible permutations of ordinal payoffs in 2x2 games and from this extracts
 * a representative game to stand for all possible permutations.  As this program will show, there are 726
 * 2x2 games with purely ordinal payoffs.  If games are considered to be different depending on which
 * perspective a player is assigned to (how some games are played depends not only on the distribution of
 * the payoffs but on this distribution relative to the perspective assigned to each player) then there
 * are 1413 games (perhaps a better terminology would be to say that there are 1413 different
 * "perspectives") because there are 613 non-symmetric games.
 *
 * This program also catalogues all the status quo positions/perspectives that exist inside all the 2x2
 * games.  A sq position consists of one player taking an outcome as default and then ordinally ranking
 * all the other payoffs with respect to the status quo.  The player making this assessment does it for
 * both players.  There are 729 sq perspectives.  As with games, they are not evenly distributed.  The TeX
 * file produced lists all the games that each sq position can be found in and the frequencies that they
 * occur with when payoffs are randomly determined.
 *
 * Pieces of the terminal output can be copied into text files and used to number games in other programs.
 * The TeX file that is created will form the basis of a taxonomy of all 2x2 games.
 *
 * This work is licensed under the Creative Commons Attribution-Share Alike 2.5 Canada License.
 * This To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/2.5/ca/
 * This or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco,
 * This California, 94105, USA.
 *
 * Written by John Simpson, Department of Philosophy, University of Alberta, 2007, 2008, 2009, 2010.
 *
*/
```

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <vector>
#include <utility>
#include <algorithm>
#include <bitset>
#include <cmath>
//#include <omp.h>

using namespace std;
//using std::vector;
//using std::swap_ranges;
//using std::count_if;

void headsUp (vector<int>*);
void sqPosCounter(vector<int>*, vector<int>*, vector<vector<int> >*);
void swapDias(vector<int>*);
void swapCols(vector<int>*);
void swapRows(vector<int>*);
void swapPlayers(vector<int>*);
void printGame(vector<int>*);
void outTexGames(vector<int>*, vector<int>*, vector<int>*, vector<int>*, vector<int>*, vector<int>*,
vector<vector<int> >*, vector<vector<int> >*, ostream&);

void outSQPositions(vector<int>*, vector<int>*, vector<vector<int> >*, vector<int>*,ostream&);
int gameIndexer(vector<int>*, const bool, vector<int>*);
int complimentTest(vector<int>*);
int bit2pos(const int);
int pos2bit(const int);
void loadIndexNumbersNC(vector<int>*);

int main(void)
{
int numberCount=0;
int copyCount=0;
int tempNumber=0;
int gameTyper[5][5]={{1,2,3,4,5},{2,6,7,8,9},{3,7,10,11,12},{4,8,11,13,14},{5,9,12,14,15}};
int gameTypeCount[5][5]={{0}};
string outFileName;
string outFileName1;
string outFileName2;

outFileName="2x2texGames.tex";
ofstream outFile0(outFileName.data(), ios::out);

outFileName1="SQ-stats.txt";
ofstream outFile1(outFileName1.data(), ios::out);

outFileName2="2x2-SQPositions.tex";
ofstream outFile2(outFileName2.data(), ios::out);

vector<int> currentGame;
//holds the 8 payoffs that make-up the game currently being considered

vector<int> gameNumbers;
//holds the the lowest calculated ID number of every single game

vector<int> pureOrdinalProb;
//holds the number of permutations of the game are possible for purely ordinal generation of each separate payoff

vector<int> tenCardinalProb(726,0);
//holds the number of permutations of the game are possible for cardinal values with each payoff randomly generated

vector<int> games;
//holds the payoffs associated with each game.  Every eight values a new game starts

vector<int> gameTypes;
//Identifies which game is of which type.  Synchronized with the vector "games".

vector<int> sqPositionCount(729,0);
//holds counts of the number of times each status quo perspective appears over total number of games

vector<vector<int> > sqPositions;
```

```
//holds the exact sq numbers that appear in each particular game

vector<vector<int> > perspectives;
vector<int> indexNumbersNC;

loadIndexNumbersNC(&indexNumbersNC);

for (int i=0; i<4; i++) //0
for (int j=0; j<4; j++) //1
for (int k=0; k<4; k++) //2
for (int l=0; l<4; l++) //3
for (int m=0; m<4; m++) //4
for (int n=0; n<4; n++) //5
for (int p=0; p<4; p++) //6
for (int q=0; q<4; q++) //7
{
currentGame.push_back(i);
currentGame.push_back(j);
currentGame.push_back(k);
currentGame.push_back(l);
currentGame.push_back(m);
currentGame.push_back(n);
currentGame.push_back(p);
currentGame.push_back(q);

//cout << "Given Game" << endl;
//printGame(&currentGame);

headsUp(&currentGame);

//cout << "Post Heads Up" << endl;
//printGame(&currentGame);

//put the index "score" of the current game into a temporary variable for easy future reference
tempNumber=((currentGame.at(0)+currentGame.at(4)*4)+
(currentGame.at(2)+currentGame.at(5)*4)*16)+
((currentGame.at(1)+currentGame.at(6)*4)+
(currentGame.at(3)+currentGame.at(7)*4)*16)*256;

int rowPayType = 0;

//all the same
if (count(currentGame.begin(), currentGame.begin() + 4, currentGame.at(0))==4)
rowPayType = 1;
else
//three the same
if (count(currentGame.begin(), currentGame.begin() + 4, currentGame.at(0))==3 ||
count(currentGame.begin(), currentGame.begin() + 4, currentGame.at(1))==3)
rowPayType = 2;
else
//two sets of two the same
if (count(currentGame.begin(), currentGame.begin() + 4, currentGame.at(0))==2 &&
count(currentGame.begin(), currentGame.begin() + 4, currentGame.at(1))==2 &&
count(currentGame.begin(), currentGame.begin() + 4, currentGame.at(2))==2)
rowPayType = 3;
else
//nothing the same
if  (count(currentGame.begin(), currentGame.begin() + 4, currentGame.at(0))==1 &&
 count(currentGame.begin(), currentGame.begin() + 4, currentGame.at(1))==1 &&
 count(currentGame.begin(), currentGame.begin() + 4, currentGame.at(2))==1)
 rowPayType = 5;
else
//one payoff pair the same
rowPayType = 4;

int colPayType = 0;

if (count(currentGame.begin() + 4, currentGame.end(), currentGame.at(4))==4)
colPayType = 1;
else
if (count(currentGame.begin() + 4, currentGame.end(), currentGame.at(4))==3 ||
count(currentGame.begin() + 4, currentGame.end(), currentGame.at(5))==3)
colPayType = 2;
else
```

```
if (count(currentGame.begin() + 4, currentGame.end(), currentGame.at(4))==2 &&
count(currentGame.begin() + 4, currentGame.end(), currentGame.at(5))==2 &&
count(currentGame.begin() + 4, currentGame.end(), currentGame.at(6))==2)
colPayType = 3;
else
if  (count(currentGame.begin() + 4, currentGame.end(), currentGame.at(4))==1 &&
 count(currentGame.begin() + 4, currentGame.end(), currentGame.at(5))==1 &&
 count(currentGame.begin() + 4, currentGame.end(), currentGame.at(6))==1)
 colPayType = 5;
else
colPayType = 4;

if (count(gameNumbers.begin(), gameNumbers.end(), tempNumber)==0)
{
gameNumbers.push_back(tempNumber);
numberCount++;

pureOrdinalProb.push_back(1);

for (int r=0; r<8; r++)
games.push_back(currentGame.at(r));

gameTypes.push_back(gameTyper[rowPayType-1][colPayType-1]);

/*
//a simple printing feature to test if all the proper games are being collected and to track down discrepancies
if (rowPayType==2 && colPayType==2)
{
cout << "RPT " << rowPayType << " CPT " << colPayType << endl;
cout << gameNumbers.back() << endl;
printGame(&currentGame);
}
*/

gameTypeCount[rowPayType-1][colPayType-1]++;

//cout << tempNumber << "," << complimentTest(&currentGame) << endl;  //used to help identify symmetric games


sqPosCounter(&currentGame, &sqPositionCount, &sqPositions);
/* Error checking tool */
for (int x=0; x<(int)sqPositions.back().size(); x++)
cout << sqPositions.back().at(x) << ", ";
cout << endl << endl;

}
else
{
int r=0;
//cout << tempNumber << " " << gameNumbers.at(r) << " " << r << endl;
while (tempNumber!=gameNumbers.at(r))
{
//cout << tempNumber << " " << gameNumbers.at(r) << " " << r << endl;
r++;
}

pureOrdinalProb.at(r)++; //Adds one to the count of the number of ways this game can show up

copyCount++;
//cout << copyCount << endl;
}

tempNumber=0;

currentGame.clear();
}

/*  This next block of code is necessary to determine the frequency of each game showing up
 *  when each payoff is generated randomly with one of ten possible values.
 */
/**/
//#pragma omp parallel for firstprivate(currentGame, tempNumber)
for (int i=0; i<10; i++) //0
for (int j=0; j<10; j++) //1
```

```
{
cout << j << endl ;
for (int k=0; k<10; k++) //2
for (int l=0; l<10; l++) //3
for (int m=0; m<10; m++) //4
for (int n=0; n<10; n++) //5
for (int p=0; p<10; p++) //6
for (int q=0; q<10; q++) //7
{
currentGame.push_back(i);
currentGame.push_back(j);
currentGame.push_back(k);
currentGame.push_back(l);
currentGame.push_back(m);
currentGame.push_back(n);
currentGame.push_back(p);
currentGame.push_back(q);

//cout << "Given Game" << endl;
//printGame(&currentGame);

headsUp(&currentGame);

//cout << "Post Heads Up" << endl;
//printGame(&currentGame);

//put the index "score" of the current game into a temporary variable for easy future reference
tempNumber=((currentGame.at(0)+currentGame.at(4)*4)+
(currentGame.at(2)+currentGame.at(5)*4)*16)+
((currentGame.at(1)+currentGame.at(6)*4)+
(currentGame.at(3)+currentGame.at(7)*4)*16)*256;

int r=0;
//cout << tempNumber << " " << gameNumbers.at(r) << " " << r << endl;
while (tempNumber!=gameNumbers.at(r))
r++;

#pragma omp critical (tenCardinalProb_update)
{
tenCardinalProb.at(r)++;
}
tempNumber=0;
currentGame.clear();
}
}
/**/

cout << "There are " << numberCount << " different numbers." << endl << endl;
cout << copyCount << " were duplicates and discarded." << endl << endl;
cout << copyCount + numberCount << " total cases." << endl << endl;

outFile1 << "There are " << numberCount << " different numbers." << endl << endl;
outFile1 << copyCount << " were duplicates were discarded." << endl << endl;
outFile1 << copyCount + numberCount << " total cases." << endl << endl;

tempNumber=0;
for (int i=0; i<(int)pureOrdinalProb.size(); i++)
tempNumber=tempNumber+pureOrdinalProb.at(i);

cout << "There are " << tempNumber << " cases included in the ordinal probability." << endl << endl;
outFile1 << "There are " << tempNumber << " cases included in the ordinal probability." << endl << endl;

tempNumber=0;
for (int i=0; i<(int)tenCardinalProb.size(); i++)
tempNumber=tempNumber+tenCardinalProb.at(i);

cout << "There are " << tempNumber << " cases included in the cardinal[10] probability." << endl << endl;
outFile1 << "There are " << tempNumber << " cases included in the cardinal[10] probability." << endl << endl;

for (int i=0; i<5; i++)
for (int j=0; j<5; j++)
{
cout << "Row Type: " << i << " Column Type: " << j << " Count: " << gameTypeCount[i][j] << endl;
outFile1 << "Row Type: " << i << " Column Type: " << j << " Count: " << gameTypeCount[i][j] << endl;
```

```
}

//sort(gameNumbers.begin(), gameNumbers.end());

for (int i=0; i<int (gameNumbers.size()); i++)
{
cout << "Game Number: " << i+1 << "   Minimum Index Value: " << gameNumbers.at(i)<< endl;
outFile1 << "Game Number: " << i+1 << "   Minimum Index Value: " << gameNumbers.at(i)<< endl;
}

cout << "Games in which each SQ position is found and the total number of games holding this position" << endl;
cout << "SQ Position Number" << endl << "Games holding this SQ position (separated by commas)" << endl;
cout << "Total Number of Games with position" << endl << endl;

outFile1 << "Games in which each SQ position is found and the total number of games holding this position" << endl;
outFile1 << "SQ Position Number" << endl << "Games holding this SQ position (separated by commas)" << endl;
outFile1 << "Total Number of Games with position" << endl << endl;

for (int i=0; i<(int)sqPositionCount.size(); i++)
{
cout << i+1 << endl;
outFile1 << i+1 << endl;
int sqCount=0;
for (int j=0; j<(int)sqPositions.size(); j++)
for (int k=0; k<(int)sqPositions.at(j).size(); k++)
if (sqPositions.at(j).at(k)==i)
{
cout << j << ", ";
outFile1 << j << ", ";
sqCount++;
}
cout << endl << sqCount << endl << endl;
outFile1 << endl << sqCount << endl << endl;
}

// What follows is just a screen dump that can easily be copied and pasted.
// It is intended to be shared with the statistician so that she can adjust the analysis
// according to the different probabilities and grouping of the behaviours

vector<int> indexGame;
for (int i=0; i<(int)gameNumbers.size(); i++)
{
for (int j=0; j<8; j++)
currentGame.push_back(games.at((8*i)+j));

indexGame.push_back(gameIndexer(&currentGame, true, &indexNumbersNC));
indexGame.push_back(gameIndexer(&currentGame, false, &indexNumbersNC));

for (int j=0; j<2; j++)
for (int k=0; k<3; k++)
for (int l=0; l<3; l++)
indexGame.push_back(indexGame.at(j)+k*1413+l*4239);

sort(indexGame.begin(), indexGame.end());

//See page 111 of "C++ Standard Library" for an explanation of the next two lines
vector<int>::iterator end=unique(indexGame.begin(), indexGame.end());
indexGame.erase(end, indexGame.end());

cout << i << " ";
outFile1 << i << " ";

for (int j=0; j<(int)indexGame.size(); j++)
{
cout << indexGame.at(j) << " ";
outFile1 << indexGame.at(j) << " ";
}

cout << "=> " << (float)tenCardinalProb.at(i)/(float)(100000000) << endl;
outFile1 << "=> " << (float)tenCardinalProb.at(i)/(float)(100000000) << endl;
indexGame.clear();
currentGame.clear();
}
cout << endl << endl;
```

```
outFile1 << endl << endl;

vector< vector<int> > allEqualPerspectives;
vector<int> equalPerspectives;
for (int i=0; i<(int)gameNumbers.size(); i++)
{
for (int j=0; j<3; j++)
for (int k=0; k<3; k++)
equalPerspectives.push_back(pos2bit(i)+j*4096+k*12288);

sort(equalPerspectives.begin(), equalPerspectives.end());

allEqualPerspectives.push_back(equalPerspectives);

equalPerspectives.clear();
}

for (int i=0; i<(int)gameNumbers.size(); i++)
{
for(int j=0; j<(int)allEqualPerspectives.at(i).size(); j++)
cout << allEqualPerspectives.at(i).at(j) << " " ;

double sqCardProb=0;

for (int j=0; j<(int)sqPositions.size(); j++)
for (int k=0; k<(int)sqPositions.at(j).size(); k++)
if (sqPositions.at(j).at(k)==i)
sqCardProb=sqCardProb + (double)tenCardinalProb.at(j);

cout << " => " << sqCardProb/(double)(100000000)/8.0 << endl;
outFile1 << " => " << sqCardProb/(double)(100000000)/8.0 << endl;
}

outTexGames(&gameNumbers, &games, &gameTypes, &pureOrdinalProb, &tenCardinalProb, &sqPositionCount,
&sqPositions, &allEqualPerspectives, outFile0);
outSQPositions(&pureOrdinalProb, &tenCardinalProb, &sqPositions, &sqPositionCount, outFile2);
}

//Converts any 2x2 game to proper ordinal form
void headsUp(vector<int>* currentGamePtr)
{
vector<int> vPays(*currentGamePtr);
//cout << "Pre Ordinality" << endl;
//printGame(&vPays);

//Game is converted to its minimum ordinal form.  First for Row and then for Column.
vector<int> ordPays;

for (int h=0; h<2; h++) //h=0 ROW, h=1 COLUMN
{
for (int i=0; i<4; i++) //Looks at four given outcome associated payoffs for the player (row/column)
if (count(ordPays.begin(), ordPays.end(), vPays.at(i+(h*4)))==0)  //if ordPays doesn't have the next value already then add it
ordPays.push_back(vPays.at(i+(h*4)));

sort(ordPays.begin(), ordPays.end()); //Orders the payoffs the player is currently receiving from smallest to largest

for (int i=0; i<int(ordPays.size()); i++)
for (int j=0; j<4; j++)  //Assures all payoffs of the same magnitude receive the same rank on conversion
if (vPays.at(j+(h*4))==ordPays.at(i))
vPays.at(j+(h*4))=i;

ordPays.clear();
}

vector<int> permValues;

permValues.push_back(((vPays.at(0)+vPays.at(4)*4)+
     (vPays.at(2)+vPays.at(5)*4)*16)+
        ((vPays.at(1)+vPays.at(6)*4)+
         (vPays.at(3)+vPays.at(7)*4)*16)*256);

permValues.push_back(((vPays.at(1)+vPays.at(6)*4)+
    (vPays.at(3)+vPays.at(7)*4)*16)+
        ((vPays.at(0)+vPays.at(4)*4)+
```

```
         (vPays.at(2)+vPays.at(5)*4)*16)*256);

permValues.push_back(((vPays.at(2)+vPays.at(5)*4)+
     (vPays.at(0)+vPays.at(4)*4)*16)+
        ((vPays.at(3)+vPays.at(7)*4)+
         (vPays.at(1)+vPays.at(6)*4)*16)*256);

permValues.push_back(((vPays.at(3)+vPays.at(7)*4)+
     (vPays.at(1)+vPays.at(6)*4)*16)+
        ((vPays.at(2)+vPays.at(5)*4)+
         (vPays.at(0)+vPays.at(4)*4)*16)*256);

/*Uncommenting returns to base 726 games vs. extended perspective of non-symmetric games*/

permValues.push_back(((vPays.at(4)+vPays.at(0)*4)+
     (vPays.at(6)+vPays.at(1)*4)*16)+
        ((vPays.at(5)+vPays.at(2)*4)+
         (vPays.at(7)+vPays.at(3)*4)*16)*256);

permValues.push_back(((vPays.at(5)+vPays.at(2)*4)+
     (vPays.at(7)+vPays.at(3)*4)*16)+
        ((vPays.at(4)+vPays.at(0)*4)+
         (vPays.at(6)+vPays.at(1)*4)*16)*256);

permValues.push_back(((vPays.at(6)+vPays.at(1)*4)+
     (vPays.at(4)+vPays.at(0)*4)*16)+
        ((vPays.at(7)+vPays.at(3)*4)+
         (vPays.at(5)+vPays.at(2)*4)*16)*256);

permValues.push_back(((vPays.at(7)+vPays.at(3)*4)+
     (vPays.at(5)+vPays.at(2)*4)*16)+
        ((vPays.at(6)+vPays.at(1)*4)+
         (vPays.at(4)+vPays.at(0)*4)*16)*256);

/**/
int minMut=0;
int minMutLoc=0;
minMut=*(min_element(permValues.begin(), permValues.end()));
while (minMut!=permValues.at(minMutLoc))
minMutLoc++;

switch (minMutLoc)
{
case 0:
break;
case 1:
swapRows(&vPays);
break;
case 2:
swapCols(&vPays);
break;
case 3:
swapDias(&vPays);
break;
case 4:
swapPlayers(&vPays);
break;
case 5:
swapPlayers(&vPays);
swapRows(&vPays);
break;
case 6:
swapPlayers(&vPays);
swapCols(&vPays);
break;
case 7:
swapPlayers(&vPays);
swapDias(&vPays);
break;
default:
cout << "ERROR " << minMutLoc << endl;
break;
}
```

```
currentGamePtr->clear();
*currentGamePtr = vPays;
}

void sqPosCounter(vector<int>* currentGamePtr, vector<int>* sqPositionCountPtr, vector<vector<int> >* sqPositionsPtr)
{
int pays[8];
bitset<12> bw;
int gameLoadCount=0;
vector<int> sqPositions;
for (int i=0; i<8; i++)
pays[i]=currentGamePtr->at(i);

//Determine better/worse unilateral/mutual switch values
//At the same time, collect status quo payoffs
//This information is used by agents to determine what kind of
//game they are playing when agent::play is called in the SR6 simulation series

for (int x=0; x<2; x++)
for (int y=0; y<2; y++)
{
if (x==1)  //TOP
if (y==1)  //TOP-LEFT
{
//Does row do better or worse if they switch and column stays?
if (pays[0]<pays[1])  //better
bw.set(0);
else
if (pays[0]>pays[1])  //worse
bw.set(1);

//Does row do better or worse if column unilaterally switches?
if (pays[0]<pays[2])  //better
bw.set(2);
else
if (pays[0]>pays[2])  //worse
bw.set(3);

//Does row do better or worse if they both switch?
if (pays[0]<pays[3])  //better
bw.set(4);
else
if (pays[0]>pays[3])  //worse
bw.set(5);

//Does column do better or worse if they switch and row stays?
if (pays[4]<pays[5])  //better
bw.set(6);
else
if (pays[4]>pays[5])  //worse
bw.set(7);

//Does column do better or worse if row switches and they stay?
if (pays[4]<pays[6])  //better
bw.set(8);
else
if (pays[4]>pays[6])  //worse
bw.set(9);

//Does column do better or worse if they both switch?
if (pays[4]<pays[7])  //better
bw.set(10);
else
if (pays[4]>pays[7])  //worse
bw.set(11);
}

else  //TOP-RIGHT
{
//Does row do better or worse if they switch and column stays?
if (pays[2]<pays[3])  //better
bw.set(0);
else
if (pays[2]>pays[3])  //worse
```

```
bw.set(1);

//Does row do better or worse if column unilaterally switches?
if (pays[2]<pays[0])  //better
bw.set(2);
else
if (pays[2]>pays[0])  //worse
bw.set(3);

//Does row do better or worse if they both switch?
if (pays[2]<pays[1])  //better
bw.set(4);
else
if (pays[2]>pays[1])  //worse
bw.set(5);

//Does column do better or worse if they switch and row stays?
if (pays[5]<pays[4])  //better
bw.set(6);
else
if (pays[5]>pays[4])  //worse
bw.set(7);

//Does column do better or worse if row switches and they stay?
if (pays[5]<pays[7])  //better
bw.set(8);
else
if (pays[5]>pays[7])  //worse
bw.set(9);

//Does column do better or worse if they both switch?
if (pays[5]<pays[6])  //better
bw.set(10);
else
if (pays[5]>pays[6])  //worse
bw.set(11);
}
else  //BOTTOM
if (y==1)  //BOTTOM-LEFT
{
//Does row do better or worse if they switch and column stays?
if (pays[1]<pays[0])  //better
bw.set(0);
else
if (pays[1]>pays[0])  //worse
bw.set(1);

//Does row do better or worse if column unilaterally switches?
if (pays[1]<pays[3])  //better
bw.set(2);
else
if (pays[1]>pays[3])  //worse
bw.set(3);

//Does row do better or worse if they both switch?
if (pays[1]<pays[2])  //better
bw.set(4);
else
if (pays[1]>pays[2])  //worse
bw.set(5);

//Does column do better or worse if they switch and row stays?
if (pays[6]<pays[7])  //better
bw.set(6);
else
if (pays[6]>pays[7])  //worse
bw.set(7);

//Does column do better or worse if row switches and they stay?
if (pays[6]<pays[4])  //better
bw.set(8);
else
if (pays[6]>pays[4])  //worse
bw.set(9);
```

```
//Does column do better or worse if they both switch?
if (pays[6]<pays[5])  //better
bw.set(10);
else
if (pays[6]>pays[5])  //worse
bw.set(11);
}
else  //BOTTOM-RIGHT
{
//Does row do better or worse if they switch and column stays?
if (pays[3]<pays[2])  //better
bw.set(0);
else
if (pays[3]>pays[2])  //worse
bw.set(1);

//Does row do better or worse if column unilaterally switches?
if (pays[3]<pays[1])  //better
bw.set(2);
else
if (pays[3]>pays[1])  //worse
bw.set(3);

//Does row do better or worse if they both switch?
if (pays[3]<pays[0])  //better
bw.set(4);
else
if (pays[3]>pays[0])  //worse
bw.set(5);

//Does column do better or worse if they switch and row stays?
if (pays[7]<pays[6])  //better
bw.set(6);
else
if (pays[7]>pays[6])  //worse
bw.set(7);

//Does column do better or worse if row switches and they stay?
if (pays[7]<pays[5])  //better
bw.set(8);
else
if (pays[7]>pays[5])  //worse
bw.set(9);

//Does column do better or worse if they both switch?
if (pays[7]<pays[4])  //better
bw.set(10);
else
if (pays[7]>pays[4])  //worse
bw.set(11);
}

//ASK GAME WHAT THE BITSETS ARE FOR EACH PERSPECTIVE

for (int z=0; z<2; z++)
{
int a, b, c, d, e, f, g, h, i, j, k, l=-1;

if (z==0) //when player is "row"
{
a=bw[0]; //IF COLUMN stays ROW does BETTER switching
b=bw[1]; //IF COLUMN stays ROW does WORSE switching
c=bw[2]; //IF COLUMN switches ROW does BETTER staying
d=bw[3]; //IF COLUMN switches ROW does WORSE staying
e=bw[4]; //IF BOTH switch ROW does BETTER
f=bw[5]; //IF BOTH switch ROW does WORSE
g=bw[6]; //IF ROW stays COLUMN does BETTER switching
h=bw[7]; //IF ROW stays COLUMN does WORSE switching
i=bw[8]; //IF ROW switches COLUMN does BETTER staying
j=bw[9]; //IF ROW switches COLUMN does WORSE staying
k=bw[10]; //IF BOTH switch COLUMN does BETTER
l=bw[11]; //IF BOTH switch COLUMN does WORSE
}
```

```
else  //when player is "column"
{
g=bw[0];
h=bw[1];
i=bw[2];
j=bw[3];
k=bw[4];
l=bw[5];
a=bw[6];
b=bw[7];
c=bw[8];
d=bw[9];
e=bw[10];
f=bw[11];
}

int choiceIndex=-1;

choiceIndex = l+k*2+j*4+i*8+h*16+g*32+f*64+e*128+d*256+c*512+b*1024+a*2048;
choiceIndex = bit2pos(choiceIndex);
sqPositionCountPtr->at(choiceIndex)++;
sqPositions.push_back(choiceIndex);
gameLoadCount++;
if (gameLoadCount>8)
gameLoadCount=gameLoadCount;

} //closes the z for-loop
bw.reset();
} //closes the x and y for-loops

sort(sqPositions.begin(), sqPositions.end());
sqPositionsPtr->push_back(sqPositions);

/* Error Checking Tool*/
cout << "SQ ID#s in Game Number: " << sqPositionsPtr->size() << endl;
for (int x=0; x<(int)sqPositions.size(); x++)
cout << sqPositions.at(x) << ", ";

cout << endl;

for (int x=0; x<(int)sqPositionsPtr->back().size(); x++)
cout << sqPositionsPtr->back().at(x) << ", ";

cout << endl;
/**/

sqPositions.clear();
}

void swapDias(vector<int>* vPaysPtr)
{
swap (vPaysPtr->at(0), vPaysPtr->at(3));
swap (vPaysPtr->at(1), vPaysPtr->at(2));
swap (vPaysPtr->at(4), vPaysPtr->at(7));
swap (vPaysPtr->at(5), vPaysPtr->at(6));
}

void swapCols(vector<int>* vPaysPtr)
{
swap (vPaysPtr->at(0), vPaysPtr->at(2));
swap (vPaysPtr->at(1), vPaysPtr->at(3));
swap (vPaysPtr->at(4), vPaysPtr->at(5));
swap (vPaysPtr->at(6), vPaysPtr->at(7));
}

void swapRows(vector<int>* vPaysPtr)
{
swap (vPaysPtr->at(0), vPaysPtr->at(1));
swap (vPaysPtr->at(2), vPaysPtr->at(3));
swap (vPaysPtr->at(4), vPaysPtr->at(6));
swap (vPaysPtr->at(5), vPaysPtr->at(7));
}

void swapPlayers(vector<int>* vPaysPtr)
```

```
{
swap_ranges (vPaysPtr->begin(), vPaysPtr->begin() + 4, vPaysPtr->begin() + 4);
}

void printGame(vector<int>* vPaysPtr)
{

cout << vPaysPtr->at(0) << "   " << vPaysPtr->at(4) << "   " << "|" << "   " << vPaysPtr->at(2) << "   " << vPaysPtr->at(5) << "   " << endl;
cout << "----------------" << endl;
cout << vPaysPtr->at(1) << "   " << vPaysPtr->at(6) << "   " << "|" << "   " << vPaysPtr->at(3) << "   " << vPaysPtr->at(7) << "   " << endl;

cout << endl << endl;
}

void outTexGames(vector<int>* gameNumbersPtr, vector<int>* gamesPtr, vector<int>* gameTypesPtr,
vector<int>* pureOrdinalProbPtr, vector<int>* tenCardinalProbPtr, vector<int>* sqPositionCountPtr,
vector<vector<int> >* sqPositionsPtr,  vector<vector<int> >* allEqualPerspectivesPtr, ostream& outFile0)
{
int printCount=0;
int maxGamesPerLine=3;
outFile0 << "\\documentclass[12pt, letterpaper, draft, oneside]{book}" << endl;

outFile0 << "\\usepackage[left=0.5in, right=1in, top =1in, bottom=1in]{geometry}" << endl;
outFile0 << "\\pagestyle{headings}" << endl;
outFile0 << "\\begin{document}" << endl << endl;

outFile0 << "\\appendix" << endl;
outFile0 << "\\chapter{Taxonomy of Games}" << endl;

for (int j=1; j<16; j++)
{
outFile0 << endl << "\\section{Games of Type " << j << "} " << endl << endl;

int typeCount=0;
for (int i=0; i<(int)gameNumbersPtr->size(); i++)
if (gameTypesPtr->at(i)==j)
typeCount++;

outFile0 << "Total Count of Type: " << typeCount << endl << endl ;
outFile0 << "\\vspace{5 mm}" << endl << endl;
outFile0 << "\\begin{center}" << endl;
outFile0 << "\\hspace*{\\fill}" << endl;
for (int i=0; i<(int)gameNumbersPtr->size(); i++)
{
if (gameTypesPtr->at(i)==j)
{
outFile0 << "\\parbox{.31\\linewidth}" << endl;
outFile0 << "{" << endl;
outFile0 << "Game " << i+1 << endl;
outFile0 << "\\begin{center}" << endl;
outFile0 << "\\begin{tabular}{|c|c|}" << endl;
outFile0 << "\\hline" << endl;
outFile0 << gamesPtr->at(i*8) << ", " << gamesPtr->at(i*8+4) << " & "
<< gamesPtr->at(i*8+2) << ", " << gamesPtr->at(i*8+5) << " \\\\" << endl;
outFile0 << "\\hline" << endl;
outFile0 << gamesPtr->at(i*8+1) << ", " << gamesPtr->at(i*8+6) << " & "
<< gamesPtr->at(i*8+3) << ", " << gamesPtr->at(i*8+7) << " \\\\" << endl;
outFile0 << "\\hline" << endl;
outFile0 << "\\end{tabular}" << endl;
outFile0 << "\\par Ord\\% "<< pureOrdinalProbPtr->at(i) << "$\\rightarrow$" << (float)pureOrdinalProbPtr->at(i)/(float)(65536) << endl;
outFile0 << "\\par Card\\% " << tenCardinalProbPtr->at(i) << "$\\rightarrow$" << (float)tenCardinalProbPtr->at(i)/(float)(100000000) << endl;

outFile0 << "\\vspace{3 mm}" << endl << endl;

int commaswitch = 0;
for (int k=0; k<(int)sqPositionsPtr->at(i).size(); k++)
{
if (commaswitch == 1)
outFile0 << ", ";
outFile0 << sqPositionsPtr->at(i).at(k)+1;
commaswitch=1;
}

outFile0 << endl << endl;
```

```
outFile0 << "\\end{center}" << endl;
outFile0 << "}" << endl;
outFile0 << "\\hspace*{\\fill}" << endl;

if ( (printCount+1)%maxGamesPerLine==0.0)
outFile0 << "\\vspace{5 mm}" << endl << endl;

printCount++;
}
}

if ( (printCount+1)%maxGamesPerLine!=0.0)
outFile0 << "\\vspace{5 mm}" << endl << endl;

outFile0 << "\\end{center}" << endl;

printCount=0;
}


for (int i=0; i<(int)sqPositionsPtr->size(); i++)
{
outFile0 << "\\begin{tabular}{|p{.22\\linewidth}|p{.38\\linewidth}|p{.29\\linewidth}|}" << endl;
outFile0 << "\\hline" << endl;
outFile0 << i+1 << endl;

double sqOrdProb=0;
double sqCardProb=0;

for (int j=0; j<(int)sqPositionsPtr->size(); j++)
for (int k=0; k<(int)sqPositionsPtr->at(j).size(); k++)
if (sqPositionsPtr->at(j).at(k)==i)
sqOrdProb = sqOrdProb + (double)pureOrdinalProbPtr->at(j);

for (int j=0; j<(int)sqPositionsPtr->size(); j++)
for (int k=0; k<(int)sqPositionsPtr->at(j).size(); k++)
if (sqPositionsPtr->at(j).at(k)==i)
sqCardProb = sqCardProb + (double)tenCardinalProbPtr->at(j);

outFile0 << "\\par Ord\\% $\\rightarrow$" << sqOrdProb/(double)(65536)/8.0 << endl;
outFile0 << "\\par Card\\% $\\rightarrow$" << sqCardProb/(double)(100000000)/8.0 << endl;
outFile0 << " & ";

for(int j=0; j<(int)allEqualPerspectivesPtr->at(i).size(); j++)
outFile0 << allEqualPerspectivesPtr->at(i).at(j) << " " ;

outFile0 << " & ";

for (int j=0; j<(int)sqPositionsPtr->size(); j++)
for (int k=0; k<(int)sqPositionsPtr->at(j).size(); k++)
if (sqPositionsPtr->at(j).at(k)==i)
outFile0 << j << " ";

outFile0 << " \\\\" << endl;
outFile0 << "\\hline" << endl;
outFile0 << "\\end{tabular}" << endl << endl;
}


outFile0 << endl << endl << "\\end{document}";
}


void outSQPositions(vector<int>* pureOrdinalProbPtr, vector<int>* tenCardinalProbPtr,
vector<vector<int> >* sqPositionsPtr,  vector<int>* sqPositionsCountPtr, ostream& outFile2)
{
int printCount=0;
int maxGamesPerLine=3;
int sqPos2Bit=0;
int sqPays[6]={0};
double probCountCard=0.0;
double probCountOrd=0.0;
outFile2 << "\\documentclass[12pt, letterpaper, draft, oneside]{book}" << endl;
```

```
outFile2 << "\\usepackage[left=0.5in, right=1in, top =1in, bottom=1in]{geometry}" << endl;
outFile2 << "\\pagestyle{headings}" << endl;
outFile2 << "\\begin{document}" << endl << endl;

outFile2 << "\\appendix" << endl;
outFile2 << "\\chapter{Status Quo Position Taxonomy}" << endl;

for (int i=0; i<729; i++)
{
sqPos2Bit=pos2bit(i);

for(int n=11; n>=0; n--)
{
if(sqPos2Bit>=pow(2.0,n))
{
sqPos2Bit=sqPos2Bit - pow(2.0,n);
switch (n)
{
case 0:
sqPays[5]=-1;
break;
case 1:
sqPays[5]=1;
break;
case 2:
sqPays[1]=-1;
break;
case 3:
sqPays[1]=1;
break;
case 4:
sqPays[3]=-1;
break;
case 5:
sqPays[3]=1;
break;
case 6:
sqPays[4]=-1;
break;
case 7:
sqPays[4]=1;
break;
case 8:
sqPays[2]=-1;
break;
case 9:
sqPays[2]=1;
break;
case 10:
sqPays[0]=-1;
break;
case 11:
sqPays[0]=1;
break;
default:
cout << "ERROR";
break;
}
}
}

outFile2 << "\\parbox{.31\\linewidth}" << endl;
outFile2 << "{" << endl;
outFile2 << "SQ Position " << i+1 << endl;
outFile2 << "\\begin{center}" << endl;
outFile2 << "\\begin{tabular}{|c|c|}" << endl;
outFile2 << "\\hline" << endl;
outFile2 << "\\underline{\\textbf{0}}" << ", " << "\\textbf{0}" << " & "
<< sqPays[2] << ", " << sqPays[3] << " \\\\" << endl;
outFile2 << "\\hline" << endl;
outFile2 << sqPays[0] << ", " << sqPays[1] << " & "
<< sqPays[4] << ", " << sqPays[5] << " \\\\" << endl;
outFile2 << "\\hline" << endl;
```

```
outFile2 << "\\end{tabular}" << endl << endl;
outFile2 << "\\vspace{3 mm}" << endl << endl;

double sqOrdProb=0;
double sqCardProb=0;

for (int j=0; j<(int)sqPositionsPtr->size(); j++)
for (int k=0; k<(int)sqPositionsPtr->at(j).size(); k++)
if (sqPositionsPtr->at(j).at(k)==i)
{
sqOrdProb = sqOrdProb + (double)pureOrdinalProbPtr->at(j);
probCountOrd = probCountOrd + sqOrdProb;
}

for (int j=0; j<(int)sqPositionsPtr->size(); j++)
for (int k=0; k<(int)sqPositionsPtr->at(j).size(); k++)
if (sqPositionsPtr->at(j).at(k)==i)
{
sqCardProb = sqCardProb + (double)tenCardinalProbPtr->at(j);
probCountCard = probCountCard + sqCardProb;
}

outFile2 << "\\par Ord\\% " << sqOrdProb << "$\\rightarrow$" << sqOrdProb/(double)(65536.0)/8.0 << endl;
outFile2 << "\\par Card\\% " << sqCardProb << "$\\rightarrow$" << sqCardProb/(double)(100000000.0)/8.0 << endl << endl;

outFile2 << sqPositionsCountPtr->at(i) << endl << endl << "\\vspace{3 mm}" << endl << endl;

int commaswitch = 0;
for (int j=0; j<(int)sqPositionsPtr->size(); j++)
for (int k=0; k<(int)sqPositionsPtr->at(j).size(); k++)
if (sqPositionsPtr->at(j).at(k)==i)
{
if (commaswitch == 1)
outFile2 << ", ";
outFile2 << j+1;
commaswitch=1;
}

outFile2 << endl << endl;

outFile2 << "\\end{center}" << endl;
outFile2 << "}" << endl;
outFile2 << "\\hspace*{\\fill}" << endl;

if ( (printCount+1)%maxGamesPerLine==0.0)
outFile2 << "\\vspace{5 mm}" << endl << endl;

printCount++;

for(int j=0; j<6; j++)
sqPays[j]=0;

}

cout << "Sum of Ordinal Probabilities for SQs: " << probCountOrd << endl;
cout << "Sum of Cardinal Probabilities for SQs: " << probCountCard << endl;

outFile2 << endl << endl << "\\end{document}";
}


int gameIndexer(vector<int>* currentGamePtr, const bool rowSet, vector<int>* indexNumbersNCPtr)
{
vector<int> vPays(*currentGamePtr);
int minMut=0;
int gameNum=0;

vector<int> permValues;

if (rowSet==true)
{
permValues.push_back(((vPays.at(0)+vPays.at(4)*4)+
     (vPays.at(2)+vPays.at(5)*4)*16)+
        ((vPays.at(1)+vPays.at(6)*4)+
```

```
         (vPays.at(3)+vPays.at(7)*4)*16)*256);

permValues.push_back(((vPays.at(1)+vPays.at(6)*4)+
    (vPays.at(3)+vPays.at(7)*4)*16)+
        ((vPays.at(0)+vPays.at(4)*4)+
         (vPays.at(2)+vPays.at(5)*4)*16)*256);

permValues.push_back(((vPays.at(2)+vPays.at(5)*4)+
     (vPays.at(0)+vPays.at(4)*4)*16)+
        ((vPays.at(3)+vPays.at(7)*4)+
         (vPays.at(1)+vPays.at(6)*4)*16)*256);

permValues.push_back(((vPays.at(3)+vPays.at(7)*4)+
     (vPays.at(1)+vPays.at(6)*4)*16)+
        ((vPays.at(2)+vPays.at(5)*4)+
         (vPays.at(0)+vPays.at(4)*4)*16)*256);

minMut=*(min_element(permValues.begin(), permValues.end()));
}
else
{
permValues.push_back(((vPays.at(4)+vPays.at(0)*4)+
     (vPays.at(6)+vPays.at(1)*4)*16)+
        ((vPays.at(5)+vPays.at(2)*4)+
         (vPays.at(7)+vPays.at(3)*4)*16)*256);

permValues.push_back(((vPays.at(5)+vPays.at(2)*4)+
     (vPays.at(7)+vPays.at(3)*4)*16)+
        ((vPays.at(4)+vPays.at(0)*4)+
         (vPays.at(6)+vPays.at(1)*4)*16)*256);

permValues.push_back(((vPays.at(6)+vPays.at(1)*4)+
     (vPays.at(4)+vPays.at(0)*4)*16)+
        ((vPays.at(7)+vPays.at(3)*4)+
         (vPays.at(5)+vPays.at(2)*4)*16)*256);

permValues.push_back(((vPays.at(7)+vPays.at(3)*4)+
     (vPays.at(5)+vPays.at(2)*4)*16)+
        ((vPays.at(6)+vPays.at(1)*4)+
         (vPays.at(4)+vPays.at(0)*4)*16)*256);

minMut=*(min_element(permValues.begin(), permValues.end()));
}

while (minMut!=indexNumbersNCPtr->at(gameNum))
gameNum++;;

return gameNum;
}

int complimentTest(vector<int>* currentGamePtr)
{
//take given game value and turn it back into

vector<int> vPays(*currentGamePtr);
//cout << "Pre Ordinality" << endl;
//printGame(&vPays);

//Game is converted to its minimum ordinal form.  First for Row and then for Column.
vector<int> ordPays;

vector<int> permValues;

permValues.push_back(((vPays.at(4)+vPays.at(0)*4)+
     (vPays.at(6)+vPays.at(1)*4)*16)+
        ((vPays.at(5)+vPays.at(2)*4)+
         (vPays.at(7)+vPays.at(3)*4)*16)*256);

permValues.push_back(((vPays.at(5)+vPays.at(2)*4)+
     (vPays.at(7)+vPays.at(3)*4)*16)+
        ((vPays.at(4)+vPays.at(0)*4)+
         (vPays.at(6)+vPays.at(1)*4)*16)*256);

permValues.push_back(((vPays.at(6)+vPays.at(1)*4)+
```

```
     (vPays.at(4)+vPays.at(0)*4)*16)+
        ((vPays.at(7)+vPays.at(3)*4)+
         (vPays.at(5)+vPays.at(2)*4)*16)*256);

permValues.push_back(((vPays.at(7)+vPays.at(3)*4)+
     (vPays.at(5)+vPays.at(2)*4)*16)+
        ((vPays.at(6)+vPays.at(1)*4)+
         (vPays.at(4)+vPays.at(0)*4)*16)*256);

int minMut=0;
minMut=*(min_element(permValues.begin(), permValues.end()));

return minMut;
}

int bit2pos(const int bit)
{
int q1=0, q2=0, q3=0, q4=0, q5=0, q6=0;

q1=bit/4;
q2=bit/16;
q3=bit/64;
q4=bit/256;
q5=bit/1024;
q6=bit/4096;
return bit-q1-q2*3-q3*9-q4*27-q5*81-q6*243;
}

int pos2bit (const int position)
{
int q1=0, q2=0, q3=0, q4=0, q5=0, q6=0;

q1=position/3;
q2=position/9;
q3=position/27;
q4=position/81;
q5=position/243;
q6=position/729;
return position+q1+q2*4+q3*16+q4*64+q5*256+q6*1024;
}

void loadIndexNumbersNC(vector<int>* indexNumbersNCPtr)
{
//Open input file with ifstream constructor
ifstream gameFile( "indexNumbersNC.dat", ios::in);

//exit program if input file failure
if (!gameFile)
{
cerr << "INDEXNUMBERSNC.DAT COULD NOT BE OPENED" << endl;
exit (1);
}  //end if

int gn=-1;
while (gameFile >> gn)
indexNumbersNCPtr->push_back(gn);
}
```

# Appendix C

# Taxonomy: 726 2×2 Game

The games are grouped by types. These types correspond exactly to the cases that were used in proving that there are exactly 726 2×2 games in Appendix A. Accompanying the normal form representation of each game is:

1. the probability of this game being randomly generated when each payoff is randomly assigned one of four possible values (listed as "Ord%" for "ordinal probability"). The value to the left of the arrow is the number of ways the game can be generated with four values. The value to the right of the arrow is the percentage chance of producing this as calculated by taking the value on the left and dividing it by the total number of possible games that can be created this way ($4^8 = 65536$).

2. the probability of this game being randomly generated when each payoff is randomly assigned one of ten possible values (listed as "Card%" for "cardinal probability"). The value to the left of the arrow is the number of ways the game can be generated with 10 values. The value to the right of the arrow is the percentage chance of producing this as calculated by taking the value on the left and dividing it by the total number of possible games that can be created this way ($10^8 = 100000000$).

3. the list of eight status quo perspectives that are found within this game (These values correspond to the 729 status quo positions found in 2×2 games that are taxonomized in Appendix D).

## C.1 Games of Type 1

Total Count of Type: 1

Game 1

| | |
|---|---|
| 0, 0 | 0, 0 |
| 0, 0 | 0, 0 |

Ord% 16→0.000244141

Card% 100→1e-06

1, 1, 1, 1, 1, 1, 1, 1

## C.2 Games of Type 2

Total Count of Type: 2

Game 2

| | |
|---|---|
| 1, 0 | 0, 0 |
| 0, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 3600→3.6e-05

3, 7, 14, 19, 55, 163, 352, 487

Game 8

| | |
|---|---|
| 1, 0 | 1, 0 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 3600→3.6e-05

2, 4, 10, 27, 28, 82, 244, 703

## C.3 Games of Type 3

Total Count of Type: 3

Game 3

| | |
|---|---|
| 0, 1 | 0, 1 |
| 0, 0 | 0, 0 |

Ord% 96→0.00146484

Card% 1800→1.8e-05

5, 5, 9, 9, 109, 109, 217, 217

Game 5

| | |
|---|---|
| 1, 0 | 1, 0 |
| 0, 0 | 0, 0 |

Ord% 96→0.00146484

Card% 1800→1.8e-05

11, 11, 21, 21, 271, 271, 541, 541

Game 7

| | |
|---|---|
| 0, 0 | 1, 0 |
| 1, 0 | 0, 0 |

Ord% 96→0.00146484

Card% 1800→1.8e-05

13, 13, 25, 25, 325, 325, 649, 649

## C.4 Games of Type 4

Total Count of Type: 9

Game 4

| | |
|---|---|
| 0, 2 | 0, 1 |
| 0, 0 | 0, 0 |

Ord% 128→0.00195312
Card% 9600→9.6e-05

9, 9, 14, 23, 217, 217, 352, 595

Game 6

| | |
|---|---|
| 2, 0 | 1, 0 |
| 0, 0 | 0, 0 |

Ord% 128→0.00195312
Card% 9600→9.6e-05

14, 17, 21, 21, 352, 433, 541, 541

Game 9

| | |
|---|---|
| 2, 0 | 1, 0 |
| 1, 0 | 0, 0 |

Ord% 128→0.00195312
Card% 9600→9.6e-05

14, 16, 22, 27, 352, 406, 568, 703

Game 10

| | |
|---|---|
| 0, 0 | 2, 0 |
| 1, 0 | 0, 0 |

Ord% 128→0.00195312
Card% 9600→9.6e-05

14, 15, 25, 25, 352, 379, 649, 649

Game 11

| | |
|---|---|
| 1, 0 | 1, 0 |
| 2, 0 | 0, 0 |

Ord% 128→0.00195312
Card% 9600→9.6e-05

12, 14, 20, 27, 298, 352, 514, 703

Game 12

| | |
|---|---|
| 2, 0 | 1, 0 |
| 2, 0 | 0, 0 |

Ord% 128→0.00195312
Card% 9600→9.6e-05

5, 5, 18, 27, 109, 109, 460, 703

Game 15

| | |
|---|---|
| 1, 0 | 2, 0 |
| 1, 0 | 0, 0 |

Ord% 128→0.00195312
Card% 9600→9.6e-05

6, 8, 14, 27, 136, 190, 352, 703

Game 16

| | |
|---|---|
| 2, 0 | 2, 0 |
| 1, 0 | 0, 0 |

Ord% 128→0.00195312
Card% 9600→9.6e-05

11, 11, 24, 27, 271, 271, 622, 703

Game 18

| | |
|---|---|
| 1, 0 | 2, 0 |
| 2, 0 | 0, 0 |

Ord% 128→0.00195312
Card% 9600→9.6e-05

13, 13, 26, 27, 325, 325, 676, 703

## C.5 Games of Type 5

Total Count of Type: 6

Game 13

| | |
|---|---|
| 3, 0 | 1, 0 |
| 2, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 16800→0.000168

14, 18, 23, 27, 352, 460, 595, 703

Game 14

| | |
|---|---|
| 2, 0 | 1, 0 |
| 3, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 16800→0.000168

14, 18, 23, 27, 352, 460, 595, 703

Game 17

| | |
|---|---|
| 3, 0 | 2, 0 |
| 1, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 16800→0.000168

14, 17, 24, 27, 352, 433, 622, 703

Game 19

| | |
|---|---|
| 1, 0 | 2, 0 |
| 3, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 16800→0.000168

14, 15, 26, 27, 352, 379, 676, 703

Game 20

| | |
|---|---|
| 2, 0 | 3, 0 |
| 1, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 16800→0.000168

14, 17, 24, 27, 352, 433, 622, 703

Game 21

| | |
|---|---|
| 1, 0 | 3, 0 |
| 2, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 16800→0.000168

14, 15, 26, 27, 352, 379, 676, 703

## C.6 Games of Type 6

Total Count of Type: 10

Game 22

| | |
|---|---|
| 1, 1 | 0, 0 |
| 0, 0 | 0, 0 |

Ord% 144→0.00219727

Card% 8100→8.1e-05

57, 57, 181, 181, 365, 365, 493, 493

Game 23

| | |
|---|---|
| 0, 1 | 1, 0 |
| 0, 0 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

61, 73, 165, 176, 358, 370, 489, 500

Game 30

| | |
|---|---|
| 1, 1 | 1, 0 |
| 1, 0 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

41, 81, 88, 166, 262, 353, 496, 705

Game 46

| | |
|---|---|
| 0, 0 | 0, 1 |
| 1, 0 | 0, 0 |

Ord% 144→0.00219727

Card% 8100→8.1e-05

68, 68, 169, 169, 354, 354, 505, 505

Game 50

| | |
|---|---|
| 0, 1 | 1, 1 |
| 0, 1 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

34, 64, 95, 164, 246, 355, 513, 721

Game 58

| | |
|---|---|
| 1, 0 | 1, 1 |
| 1, 0 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

46, 58, 84, 189, 257, 361, 488, 709

Game 60

| | |
|---|---|
| 0, 1 | 1, 0 |
| 1, 0 | 1, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

30, 56, 100, 172, 250, 378, 490, 716

Game 321

| | |
|---|---|
| 1, 1 | 1, 1 |
| 1, 1 | 0, 0 |

Ord% 144→0.00219727

Card% 8100→8.1e-05

29, 29, 91, 91, 247, 247, 729, 729

Game 337

| | |
|---|---|
| 1, 1 | 1, 1 |
| 0, 1 | 1, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

31, 37, 83, 108, 245, 270, 706, 712

Game 344

| | |
|---|---|
| 0, 1 | 1, 1 |
| 1, 1 | 1, 0 |

Ord% 144→0.00219727

Card% 8100→8.1e-05

54, 54, 85, 85, 253, 253, 704, 704

## C.7 Games of Type 7

Total Count of Type: 12

Game 24

| | |
|---|---|
| 1, 1 | 0, 1 |
| 0, 0 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

63, 115, 122, 167, 219, 235, 356, 495

Game 27

| | |
|---|---|
| 1, 1 | 1, 0 |
| 0, 0 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

75, 183, 284, 289, 362, 497, 543, 547

Game 29

| | |
|---|---|
| 0, 1 | 1, 0 |
| 1, 0 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

79, 175, 331, 343, 376, 499, 651, 662

Game 47

| | |
|---|---|
| 0, 0 | 1, 1 |
| 1, 0 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

67, 187, 327, 338, 364, 511, 655, 667

Game 49

| | |
|---|---|
| 1, 0 | 1, 0 |
| 0, 1 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

65, 173, 273, 277, 372, 507, 554, 559

Game 57

| | |
|---|---|
| 1, 0 | 0, 1 |
| 1, 0 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

59, 111, 127, 171, 223, 230, 360, 491

Game 101

| | |
|---|---|
| 1, 1 | 1, 1 |
| 1, 0 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

32, 90, 110, 118, 220, 243, 248, 711

Game 117

| | |
|---|---|
| 1, 1 | 0, 1 |
| 1, 0 | 1, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

36, 86, 112, 135, 218, 226, 252, 707

Game 201

| | |
|---|---|
| 1, 1 | 1, 1 |
| 0, 1 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

38, 92, 264, 272, 274, 550, 567, 723

Game 217

| | |
|---|---|
| 1, 1 | 1, 0 |
| 0, 1 | 1, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

48, 102, 254, 280, 297, 542, 544, 713

Game 292

| | |
|---|---|
| 1, 0 | 1, 1 |
| 1, 1 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

52, 94, 256, 328, 334, 650, 675, 727

Game 308

| | |
|---|---|
| 1, 1 | 0, 1 |
| 0, 1 | 1, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

40, 106, 268, 326, 351, 652, 658, 715

## C.8 Games of Type 8

Total Count of Type: 72

Game 25

| | |
|---|---|
| 1, 2 | 0, 1 |
| 0, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

63, 185, 219, 235, 365, 365, 495, 601

Game 26

| | |
|---|---|
| 0, 2 | 1, 1 |
| 0, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

63, 176, 219, 235, 358, 374, 495, 608

Game 28

| | |
|---|---|
| 2, 1 | 1, 0 |
| 0, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

75, 183, 365, 365, 451, 503, 543, 547

Game 31

| | |
|---|---|
| 2, 1 | 1, 0 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

81, 184, 365, 365, 424, 502, 574, 705

Game 32

| | |
|---|---|
| 0, 1 | 1, 0 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

79, 176, 358, 376, 397, 501, 651, 662

Game 33

| | |
|---|---|
| 1, 1 | 1, 0 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

81, 176, 316, 358, 371, 498, 527, 705

Game 34

| | |
|---|---|
| 2, 1 | 1, 0 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

81, 115, 122, 167, 356, 478, 504, 705

Game 37

| | |
|---|---|
| 1, 1 | 2, 0 |
| 0, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

75, 183, 368, 370, 446, 500, 543, 547

Game 38

| | |
|---|---|
| 0, 1 | 2, 0 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

79, 177, 370, 376, 385, 500, 651, 662

Game 39

| | |
|---|---|
| 1, 1 | 2, 0 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

81, 142, 168, 203, 359, 370, 500, 705

Game 40

| | |
|---|---|
| 2, 1 | 2, 0 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

81, 186, 284, 289, 362, 497, 628, 705

Game 42

| | |
|---|---|
| 1, 1 | 2, 0 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

81, 175, 331, 343, 377, 499, 689, 705

Game 48

| | |
|---|---|
| 0, 0 | 2, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

69, 187, 365, 365, 381, 511, 655, 667

Game 51

| | |
|---|---|
| 0, 1 | 1, 2 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

66, 182, 300, 365, 365, 513, 520, 721

Game 52

| | |
|---|---|
| 2, 0 | 1, 0 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

71, 176, 358, 372, 435, 507, 554, 559

Game 53

| | |
|---|---|
| 0, 2 | 1, 1 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

70, 176, 358, 373, 408, 513, 581, 721

Game 54

| | |
|---|---|
| 0, 2 | 1, 2 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

72, 115, 122, 167, 356, 462, 513, 721

Game 59

| | |
|---|---|
| 1, 0 | 2, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

60, 138, 189, 208, 365, 365, 494, 709

Game 61

| | |
|---|---|
| 0, 1 | 1, 0 |
| 2, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

62, 154, 176, 192, 358, 378, 492, 716

Game 62

| | |
|---|---|
| 2, 0 | 0, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

77, 171, 223, 230, 360, 370, 500, 597

Game 63

| | |
|---|---|
| 2, 0 | 1, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

76, 189, 367, 370, 419, 500, 570, 709

Game 64

| | |
|---|---|
| 2, 0 | 2, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

78, 189, 284, 289, 362, 497, 624, 709

Game 66

| | |
|---|---|
| 0, 1 | 2, 0 |
| 1, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

74, 174, 304, 370, 378, 500, 516, 716

Game 67

| | |
|---|---|
| 0, 1 | 2, 0 |
| 2, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

80, 175, 331, 343, 378, 499, 678, 716

Game 71

| | |
|---|---|
| 0, 0 | 1, 1 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

68, 187, 354, 366, 392, 511, 655, 667

Game 72

| | |
|---|---|
| 1, 0 | 2, 0 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

68, 179, 354, 372, 439, 507, 554, 559

Game 73

| | |
|---|---|
| 1, 0 | 2, 0 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

68, 149, 170, 196, 354, 357, 513, 721

Game 74

| | |
|---|---|
| 1, 0 | 2, 0 |
| 2, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

67, 188, 327, 338, 364, 513, 682, 721

Game 76

| | |
|---|---|
| 2, 0 | 2, 0 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

65, 173, 273, 277, 375, 513, 635, 721

Game 78

| | |
|---|---|
| 0, 1 | 0, 2 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

68, 171, 223, 230, 354, 360, 509, 613

Game 79

| | |
|---|---|
| 1, 0 | 1, 1 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

68, 189, 311, 354, 363, 506, 532, 709

Game 80

| | |
|---|---|
| 1, 0 | 2, 1 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

67, 189, 327, 338, 364, 512, 694, 709

Game 82

| | |
|---|---|
| 1, 0 | 0, 1 |
| 2, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

68, 178, 354, 378, 412, 508, 586, 716

Game 83

| | |
|---|---|
| 2, 0 | 2, 0 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

65, 173, 273, 277, 378, 510, 640, 716

Game 85

| | |
|---|---|
| 2, 0 | 1, 1 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

59, 111, 127, 189, 369, 473, 491, 709

Game 86

| | |
|---|---|
| 2, 0 | 0, 1 |
| 2, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

59, 111, 127, 180, 378, 466, 491, 716

Game 136

| | |
|---|---|
| 1, 2 | 1, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

41, 90, 220, 243, 266, 353, 604, 711

Game 155

| | |
|---|---|
| 1, 2 | 0, 1 |
| 1, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

36, 95, 218, 226, 252, 355, 621, 725

Game 162

| | |
|---|---|
| 1, 1 | 1, 2 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

50, 90, 220, 243, 257, 361, 596, 711

Game 164

| | |
|---|---|
| 0, 2 | 1, 1 |
| 1, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

36, 104, 218, 226, 252, 378, 598, 716

Game 232

| | |
|---|---|
| 2, 1 | 1, 1 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

41, 98, 264, 353, 436, 550, 567, 723

Game 250

| | |
|---|---|
| 1, 1 | 2, 1 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

44, 95, 264, 355, 434, 550, 567, 723

Game 256

| | |
|---|---|
| 1, 2 | 1, 0 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

48, 102, 257, 361, 459, 542, 544, 719

Game 258

| | |
|---|---|
| 0, 2 | 1, 0 |
| 1, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

48, 102, 260, 378, 442, 542, 544, 716

Game 322

| | |
|---|---|
| 2, 1 | 1, 1 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

41, 97, 265, 353, 409, 577, 729, 729

Game 323

| | |
|---|---|
| 1, 0 | 1, 2 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

52, 96, 257, 361, 382, 650, 675, 727

Game 324

| | |
|---|---|
| 1, 1 | 1, 2 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

47, 93, 257, 301, 361, 515, 729, 729

Game 325

| | |
|---|---|
| 1, 2 | 1, 2 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

32, 99, 110, 118, 248, 463, 729, 729

Game 328

| | |
|---|---|
| 0, 1 | 2, 1 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

52, 95, 258, 355, 388, 650, 675, 727

Game 329

| | |
|---|---|
| 1, 1 | 2, 1 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

35, 95, 145, 191, 249, 355, 729, 729

Game 330

| | |
|---|---|
| 2, 1 | 2, 1 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

38, 92, 267, 272, 274, 631, 729, 729

Game 332

| | |
|---|---|
| 1, 1 | 2, 1 |
| 2, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

53, 94, 256, 328, 334, 677, 729, 729

Game 336

| | |
|---|---|
| 2, 1 | 0, 1 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

41, 106, 268, 353, 405, 652, 658, 717

Game 338

| | |
|---|---|
| 1, 1 | 1, 2 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

41, 108, 263, 324, 353, 523, 706, 714

Game 339

| | |
|---|---|
| 1, 2 | 1, 1 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

49, 108, 257, 361, 432, 569, 706, 718

Game 340

| | |
|---|---|
| 1, 2 | 1, 2 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

32, 108, 110, 118, 248, 486, 706, 720

Game 343

| | |
|---|---|
| 2, 1 | 1, 1 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

41, 89, 162, 193, 270, 353, 708, 712

Game 345

| | |
|---|---|
| 0, 1 | 1, 1 |
| 2, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

54, 87, 139, 216, 257, 361, 704, 710

Game 346

| | |
|---|---|
| 1, 1 | 2, 1 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

43, 95, 270, 355, 407, 594, 712, 724

Game 347

| | |
|---|---|
| 2, 1 | 2, 1 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

38, 92, 270, 272, 274, 648, 712, 726

Game 349

| | |
|---|---|
| 0, 1 | 2, 1 |
| 1, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

54, 95, 255, 307, 355, 540, 704, 722

Game 350

| | |
|---|---|
| 0, 1 | 2, 1 |
| 2, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

54, 94, 256, 328, 334, 702, 704, 728

Game 354

| | |
|---|---|
| 1, 0 | 0, 2 |
| 1, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

42, 106, 268, 378, 380, 652, 658, 716

Game 355

| | |
|---|---|
| 1, 1 | 1, 1 |
| 0, 2 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

33, 108, 137, 199, 251, 378, 706, 716

Game 356

| | |
|---|---|
| 1, 1 | 1, 2 |
| 0, 2 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

40, 108, 269, 326, 351, 685, 706, 715

Game 358

| | |
|---|---|
| 1, 2 | 1, 1 |
| 0, 2 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

51, 108, 254, 280, 297, 623, 706, 713

Game 360

| | |
|---|---|
| 1, 1 | 0, 2 |
| 1, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

39, 101, 270, 299, 378, 517, 712, 716

Game 361

| | |
|---|---|
| 1, 1 | 0, 2 |
| 1, 2 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

40, 107, 270, 326, 351, 679, 712, 715

Game 363

| | |
|---|---|
| 0, 2 | 1, 1 |
| 1, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

54, 103, 259, 378, 415, 571, 704, 716

Game 364

| | |
|---|---|
| 0, 2 | 1, 1 |
| 1, 2 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

54, 105, 254, 280, 297, 625, 704, 713

Game 366

| | |
|---|---|
| 1, 2 | 0, 2 |
| 1, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

45, 86, 112, 135, 270, 461, 707, 712

Game 367

| | |
|---|---|
| 0, 2 | 1, 2 |
| 1, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

54, 86, 112, 135, 261, 469, 704, 707

## C.9 Games of Type 9

Total Count of Type: 48

Game 35

| | |
|---|---|
| 3, 1 | 1, 0 |
| 2, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

81, 185, 365, 365, 478, 504, 601, 705

Game 36

| | |
|---|---|
| 2, 1 | 1, 0 |
| 3, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

81, 176, 358, 374, 478, 504, 608, 705

Game 41

| | |
|---|---|
| 3, 1 | 2, 0 |
| 1, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

81, 186, 365, 365, 451, 503, 628, 705

Game 43

| | |
|---|---|
| 1, 1 | 2, 0 |
| 3, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

81, 176, 358, 377, 397, 501, 689, 705

Game 44

| | |
|---|---|
| 2, 1 | 3, 0 |
| 1, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

81, 186, 368, 370, 446, 500, 628, 705

Game 45

| | |
|---|---|
| 1, 1 | 3, 0 |
| 2, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

81, 177, 370, 377, 385, 500, 689, 705

Game 55

| | |
|---|---|
| 0, 2 | 1, 3 |
| 0, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

72, 185, 365, 365, 462, 513, 601, 721

Game 56

| | |
|---|---|
| 0, 3 | 1, 2 |
| 0, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

72, 176, 358, 374, 462, 513, 608, 721

Game 65

| | |
|---|---|
| 2, 0 | 3, 1 |
| 1, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

78, 189, 365, 365, 451, 503, 624, 709

Game 68

| | |
|---|---|
| 0, 1 | 2, 0 |
| 3, 0 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

80, 176, 358, 378, 397, 501, 678, 716

Game 69

| | |
|---|---|
| 3, 0 | 2, 1 |
| 1, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

78, 189, 368, 370, 446, 500, 624, 709

Game 70

| | |
|---|---|
| 0, 1 | 3, 0 |
| 2, 0 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

80, 177, 370, 378, 385, 500, 678, 716

Game 75

| | |
|---|---|
| 1, 0 | 2, 0 |
| 3, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

69, 188, 365, 365, 381, 513, 682, 721

Game 77

| | |
|---|---|
| 3, 0 | 2, 0 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

71, 176, 358, 375, 435, 513, 635, 721

Game 81

| | |
|---|---|
| 1, 0 | 3, 1 |
| 2, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

69, 189, 365, 365, 381, 512, 694, 709

Game 84

| | |
|---|---|
| 3, 0 | 2, 0 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

71, 176, 358, 378, 435, 510, 640, 716

Game 87

| | |
|---|---|
| 3, 0 | 1, 1 |
| 2, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

77, 189, 369, 370, 473, 500, 597, 709

Game 88

| | |
|---|---|
| 3, 0 | 0, 1 |
| 2, 0 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

77, 180, 370, 378, 466, 500, 597, 716

Game 89

| | |
|---|---|
| 1, 0 | 3, 0 |
| 2, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

68, 188, 354, 366, 392, 513, 682, 721

Game 90

| | |
|---|---|
| 2, 0 | 3, 0 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

68, 179, 354, 375, 439, 513, 635, 721

Game 91

| | |
|---|---|
| 1, 0 | 2, 1 |
| 3, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

68, 189, 354, 366, 392, 512, 694, 709

Game 92

| | |
|---|---|
| 2, 0 | 3, 0 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

68, 179, 354, 378, 439, 510, 640, 716

Game 93

| | |
|---|---|
| 2, 0 | 1, 1 |
| 3, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

68, 189, 354, 369, 473, 509, 613, 709

Game 94

| | |
|---|---|
| 2, 0 | 0, 1 |
| 3, 0 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

68, 180, 354, 378, 466, 509, 613, 716

Game 326

| | |
|---|---|
| 1, 3 | 1, 2 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card%

75600→0.000756

41, 99, 266, 353, 463, 604, 729, 729

Game 327

| | |
|---|---|
| 1, 2 | 1, 3 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card%

75600→0.000756

50, 99, 257, 361, 463, 596, 729, 729

Game 331

| | |
|---|---|
| 3, 1 | 2, 1 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card%

75600→0.000756

41, 98, 267, 353, 436, 631, 729, 729

Game 333

| | |
|---|---|
| 1, 1 | 2, 1 |
| 3, 1 | 0, 0 |

Ord% 48→0.000732422

Card%

75600→0.000756

53, 96, 257, 361, 382, 677, 729, 729

Game 334

| | |
|---|---|
| 2, 1 | 3, 1 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card%

75600→0.000756

44, 95, 267, 355, 434, 631, 729, 729

Game 335

| | |
|---|---|
| 1, 1 | 3, 1 |
| 2, 1 | 0, 0 |

Ord% 48→0.000732422

Card%

75600→0.000756

53, 95, 258, 355, 388, 677, 729, 729

Game 341

| | |
|---|---|
| 1, 2 | 1, 3 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card%

75600→0.000756

41, 108, 266, 353, 486, 604, 706, 720

Game 342

| | |
|---|---|
| 1, 3 | 1, 2 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card%

75600→0.000756

50, 108, 257, 361, 486, 596, 706, 720

Game 348

| | |
|---|---|
| 3, 1 | 2, 1 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card%

75600→0.000756

41, 98, 270, 353, 436, 648, 712, 726

Game 351

| | |
|---|---|
| 0, 1 | 2, 1 |
| 3, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

54, 96, 257, 361, 382, 702, 704, 728

Game 352

| | |
|---|---|
| 2, 1 | 3, 1 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

44, 95, 270, 355, 434, 648, 712, 726

Game 353

| | |
|---|---|
| 0, 1 | 3, 1 |
| 2, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

54, 95, 258, 355, 388, 702, 704, 728

Game 357

| | |
|---|---|
| 1, 1 | 1, 3 |
| 0, 2 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

41, 108, 269, 353, 405, 685, 706, 717

Game 359

| | |
|---|---|
| 1, 3 | 1, 1 |
| 0, 2 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

51, 108, 257, 361, 459, 623, 706, 719

Game 362

| | |
|---|---|
| 1, 1 | 0, 2 |
| 1, 3 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

41, 107, 270, 353, 405, 679, 712, 717

Game 365

| | |
|---|---|
| 0, 2 | 1, 1 |
| 1, 3 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

54, 105, 257, 361, 459, 625, 704, 719

Game 368

| | |
|---|---|
| 1, 3 | 0, 2 |
| 1, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

45, 95, 270, 355, 461, 621, 712, 725

Game 369

| | |
|---|---|
| 0, 2 | 1, 3 |
| 1, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

54, 95, 261, 355, 469, 621, 704, 725

Game 370

| | |
|---|---|
| 1, 1 | 1, 2 |
| 0, 3 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

42, 108, 269, 378, 380, 685, 706, 716

Game 371

| | |
|---|---|
| 1, 2 | 1, 1 |
| 0, 3 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

51, 108, 260, 378, 442, 623, 706, 716

Game 372

| | |
|---|---|
| 1, 1 | 0, 3 |
| 1, 2 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

42, 107, 270, 378, 380, 679, 712, 716

Game 373

| | |
|---|---|
| 0, 3 | 1, 1 |
| 1, 2 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

54, 105, 260, 378, 442, 625, 704, 716

Game 374

| | |
|---|---|
| 1, 2 | 0, 3 |
| 1, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

45, 104, 270, 378, 461, 598, 712, 716

Game 375

| | |
|---|---|
| 0, 3 | 1, 2 |
| 1, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

54, 104, 261, 378, 469, 598, 704, 716

## C.10 Games of Type 10

Total Count of Type: 8

Game 95

| | |
|---|---|
| 1, 1 | 0, 1 |
| 1, 0 | 0, 0 |

Ord% 144→0.00219727

Card% 8100→8.1e-05

113, 113, 117, 117, 221, 221, 225, 225

Game 98

| | |
|---|---|
| 1, 1 | 1, 1 |
| 0, 0 | 0, 0 |

Ord% 144→0.00219727

Card% 8100→8.1e-05

119, 119, 237, 237, 275, 275, 549, 549

Game 100

| | |
|---|---|
| 0, 1 | 1, 1 |
| 1, 0 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

121, 133, 229, 241, 329, 333, 653, 657

Game 116

| | |
|---|---|
| 0, 1 | 0, 1 |
| 1, 0 | 1, 0 |

Ord% 144→0.00219727

Card% 8100→8.1e-05

129, 129, 227, 227, 279, 279, 545, 545

Game 198

| | |
|---|---|
| 1, 1 | 1, 0 |
| 0, 1 | 0, 0 |

Ord% 144→0.00219727

Card% 8100→8.1e-05

281, 281, 291, 291, 551, 551, 561, 561

Game 200

| | |
|---|---|
| 1, 0 | 1, 1 |
| 0, 1 | 0, 0 |

Ord% 288→0.00439453

Card% 16200→0.000162

283, 295, 335, 345, 553, 565, 659, 669

Game 291

| | |
|---|---|
| 0, 0 | 1, 1 |
| 1, 1 | 0, 0 |

Ord% 72→0.00109863

Card% 4050→4.05e-05

337, 337, 337, 337, 673, 673, 673, 673

Game 306

| | |
|---|---|
| 1, 0 | 0, 1 |
| 0, 1 | 1, 0 |

Ord% 72→0.00109863

Card% 4050→4.05e-05

349, 349, 349, 349, 661, 661, 661, 661

## C.11 Games of Type 11

Total Count of Type: 54

Game 96

| | |
|---|---|
| 2, 1 | 0, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

117, 122, 221, 225, 225, 239, 356, 603

Game 97

| | |
|---|---|
| 1, 1 | 0, 2 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

117, 131, 221, 225, 225, 230, 360, 599

Game 99

| | |
|---|---|
| 2, 1 | 1, 1 |
| 0, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

122, 125, 237, 237, 356, 437, 549, 549

Game 102

| | |
|---|---|
| 2, 1 | 1, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

122, 124, 238, 243, 356, 410, 576, 711

Game 103

| | |
|---|---|
| 0, 1 | 1, 1 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

123, 133, 230, 241, 360, 383, 653, 657

Game 104

| | |
|---|---|
| 1, 1 | 1, 1 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

120, 128, 230, 243, 302, 360, 518, 711

Game 105

| | |
|---|---|
| 2, 1 | 1, 1 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

113, 113, 117, 126, 221, 243, 464, 711

Game 108

| | |
|---|---|
| 0, 1 | 2, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

122, 133, 231, 241, 356, 387, 653, 657

Game 109

| | |
|---|---|
| 1, 1 | 2, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

116, 122, 144, 194, 222, 243, 356, 711

Game 110

| | |
|---|---|
| 2, 1 | 2, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

119, 119, 240, 243, 275, 275, 630, 711

Game 112

| | |
|---|---|
| 1, 1 | 2, 1 |
| 2, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

121, 134, 229, 243, 329, 333, 680, 711

Game 118

| | |
|---|---|
| 2, 1 | 0, 1 |
| 1, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

122, 135, 228, 236, 306, 356, 522, 707

Game 119

| | |
|---|---|
| 0, 1 | 0, 1 |
| 2, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

129, 129, 230, 233, 360, 441, 545, 545

Game 120

| | |
|---|---|
| 1, 1 | 0, 1 |
| 2, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

130, 135, 230, 232, 360, 414, 572, 707

Game 121

| | |
|---|---|
| 2, 1 | 0, 1 |
| 2, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

113, 113, 117, 135, 221, 234, 468, 707

Game 124

| | |
|---|---|
| 0, 1 | 1, 1 |
| 2, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

114, 135, 140, 198, 224, 230, 360, 707

Game 125

| | |
|---|---|
| 0, 1 | 2, 1 |
| 2, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

121, 135, 229, 242, 329, 333, 684, 707

Game 128

| | |
|---|---|
| 0, 2 | 1, 1 |
| 0, 2 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

132, 135, 227, 227, 279, 279, 626, 707

Game 133

| | |
|---|---|
| 1, 2 | 1, 1 |
| 0, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

237, 237, 284, 293, 362, 549, 549, 605

Game 135

| | |
|---|---|
| 0, 2 | 1, 1 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

229, 241, 333, 347, 376, 607, 657, 662

Game 152

| | |
|---|---|
| 0, 1 | 1, 2 |
| 1, 0 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

229, 241, 333, 338, 364, 619, 657, 671

Game 154

| | |
|---|---|
| 0, 2 | 0, 1 |
| 1, 0 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

227, 227, 279, 279, 372, 554, 563, 615

Game 199

| | |
|---|---|
| 2, 1 | 1, 0 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

284, 291, 362, 453, 551, 557, 561, 561

Game 202

| | |
|---|---|
| 1, 2 | 1, 1 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

284, 292, 362, 426, 556, 567, 578, 723

Game 203

| | |
|---|---|
| 1, 0 | 1, 2 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

284, 295, 362, 399, 555, 565, 659, 669

Game 204

| | |
|---|---|
| 1, 1 | 1, 2 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

284, 290, 318, 362, 524, 552, 567, 723

Game 205

| | |
|---|---|
| 1, 2 | 1, 2 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

119, 119, 275, 275, 480, 558, 567, 723

Game 208

| | |
|---|---|
| 1, 1 | 2, 0 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

287, 291, 372, 443, 551, 554, 561, 561

Game 209

| | |
|---|---|
| 0, 1 | 2, 0 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

285, 295, 372, 389, 554, 565, 659, 669

Game 210

| | |
|---|---|
| 1, 1 | 2, 0 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

146, 200, 276, 278, 372, 554, 567, 723

Game 211

| | |
|---|---|
| 2, 1 | 2, 0 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

281, 281, 291, 294, 551, 567, 632, 723

Game 213

| | |
|---|---|
| 1, 1 | 2, 0 |
| 2, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

283, 296, 335, 345, 553, 567, 686, 723

Game 218

| | |
|---|---|
| 2, 1 | 1, 0 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

156, 210, 284, 297, 362, 546, 548, 713

Game 219

| | |
|---|---|
| 1, 1 | 2, 0 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

286, 297, 372, 416, 554, 562, 588, 713

Game 220

| | |
|---|---|
| 2, 1 | 2, 0 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

281, 281, 291, 297, 551, 564, 642, 713

Game 222

| | |
|---|---|
| 0, 1 | 2, 0 |
| 1, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

282, 297, 308, 372, 534, 554, 560, 713

Game 223

| | |
|---|---|
| 0, 1 | 2, 0 |
| 2, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

283, 297, 335, 345, 553, 566, 696, 713

Game 227

| | |
|---|---|
| 0, 2 | 0, 2 |
| 1, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

129, 129, 288, 297, 470, 545, 545, 713

Game 231

| | |
|---|---|
| 2, 0 | 1, 1 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

341, 345, 376, 445, 553, 565, 662, 669

Game 248

| | |
|---|---|
| 1, 0 | 2, 1 |
| 0, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

338, 345, 364, 457, 553, 565, 665, 669

Game 293

| | |
|---|---|
| 2, 0 | 1, 1 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

340, 346, 376, 418, 580, 662, 675, 727

Game 294

| | |
|---|---|
| 0, 0 | 2, 1 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

338, 339, 364, 391, 673, 673, 673, 673

Game 295

| | |
|---|---|
| 0, 1 | 1, 2 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

310, 336, 338, 364, 538, 668, 675, 727

Game 296

| | |
|---|---|
| 0, 2 | 1, 2 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

121, 133, 329, 342, 472, 653, 675, 727

Game 299

| | |
|---|---|
| 1, 0 | 2, 1 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

148, 214, 330, 338, 364, 656, 675, 727

Game 300

| | |
|---|---|
| 2, 0 | 2, 1 |
| 1, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

283, 295, 335, 348, 634, 659, 675, 727

Game 302

| | |
|---|---|
| 1, 0 | 2, 1 |
| 2, 1 | 0, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

337, 337, 337, 337, 674, 675, 700, 727

Game 307

| | |
|---|---|
| 2, 0 | 0, 1 |
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

349, 349, 376, 403, 661, 661, 662, 663

Game 309

| 1, 1 | 0, 2 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

322, 344, 351, 376, 526, 660, 662, 715

Game 310

| 1, 2 | 0, 1 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

338, 351, 364, 430, 592, 664, 670, 715

Game 311

| 1, 2 | 0, 2 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

121, 133, 329, 351, 484, 653, 666, 715

Game 314

| 2, 0 | 1, 1 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

160, 202, 332, 351, 376, 654, 662, 715

Game 315

| 2, 0 | 2, 1 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

283, 295, 335, 351, 646, 659, 672, 715

Game 318

| 1, 1 | 0, 2 |
|---|---|
| 0, 2 | 1, 0 |

Ord% 192→0.00292969

Card% 43200→0.000432

349, 349, 350, 351, 661, 661, 688, 715

## C.12 Games of Type 12

Total Count of Type: 36

Game 106

| 3, 1 | 1, 1 |
|---|---|
| 2, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

122, 126, 239, 243, 356, 464, 603, 711

Game 107

| 2, 1 | 1, 1 |
|---|---|
| 3, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

126, 131, 230, 243, 360, 464, 599, 711

Game 111

| 3, 1 | 2, 1 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

122, 125, 240, 243, 356, 437, 630, 711

Game 113

| | |
|---|---|
| 1, 1 | 2, 1 |
| 3, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

123, 134, 230, 243, 360, 383, 680, 711

Game 114

| | |
|---|---|
| 2, 1 | 3, 1 |
| 1, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

122, 125, 240, 243, 356, 437, 630, 711

Game 115

| | |
|---|---|
| 1, 1 | 3, 1 |
| 2, 0 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

122, 134, 231, 243, 356, 387, 680, 711

Game 122

| | |
|---|---|
| 3, 1 | 0, 1 |
| 2, 0 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

122, 135, 234, 239, 356, 468, 603, 707

Game 123

| | |
|---|---|
| 2, 1 | 0, 1 |
| 3, 0 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

131, 135, 230, 234, 360, 468, 599, 707

Game 126

| | |
|---|---|
| 0, 1 | 2, 1 |
| 3, 0 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

123, 135, 230, 242, 360, 383, 684, 707

Game 127

| | |
|---|---|
| 0, 1 | 3, 1 |
| 2, 0 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

122, 135, 231, 242, 356, 387, 684, 707

Game 129

| | |
|---|---|
| 0, 3 | 1, 1 |
| 0, 2 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

132, 135, 230, 233, 360, 441, 626, 707

Game 130

| | |
|---|---|
| 0, 2 | 1, 1 |
| 0, 3 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

132, 135, 230, 233, 360, 441, 626, 707

Game 206

| | |
|---|---|
| 1, 3 | 1, 2 |
| 0, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

284, 293, 362, 480, 558, 567, 605, 723

Game 207

| | |
|---|---|
| 1, 2 | 1, 3 |
| 0, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

284, 293, 362, 480, 558, 567, 605, 723

Game 212

| | |
|---|---|
| 3, 1 | 2, 0 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

284, 294, 362, 453, 557, 567, 632, 723

Game 214

| | |
|---|---|
| 1, 1 | 2, 0 |
| 3, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

284, 296, 362, 399, 555, 567, 686, 723

Game 215

| | |
|---|---|
| 2, 1 | 3, 0 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

287, 294, 372, 443, 554, 567, 632, 723

Game 216

| | |
|---|---|
| 1, 1 | 3, 0 |
| 2, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

285, 296, 372, 389, 554, 567, 686, 723

Game 221

| | |
|---|---|
| 3, 1 | 2, 0 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

284, 297, 362, 453, 557, 564, 642, 713

Game 224

| | |
|---|---|
| 0, 1 | 2, 0 |
| 3, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

284, 297, 362, 399, 555, 566, 696, 713

Game 225

| | |
|---|---|
| 2, 1 | 3, 0 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

287, 297, 372, 443, 554, 564, 642, 713

Game 226

| | |
|---|---|
| 0, 1 | 3, 0 |
| 2, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

285, 297, 372, 389, 554, 566, 696, 713

Game 228

| | |
|---|---|
| 0, 3 | 0, 2 |
| 1, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

288, 297, 372, 470, 554, 563, 615, 713

Game 229

| | |
|---|---|
| 0, 2 | 0, 3 |
| 1, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

288, 297, 372, 470, 554, 563, 615, 713

Game 297

| | |
|---|---|
| 0, 3 | 1, 2 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

342, 347, 376, 472, 607, 662, 675, 727

Game 298

| | |
|---|---|
| 0, 2 | 1, 3 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

338, 342, 364, 472, 619, 671, 675, 727

Game 301

| | |
|---|---|
| 3, 0 | 2, 1 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

341, 348, 376, 445, 634, 662, 675, 727

Game 303

| | |
|---|---|
| 1, 0 | 2, 1 |
| 3, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

338, 339, 364, 391, 674, 675, 700, 727

Game 304

| | |
|---|---|
| 2, 0 | 3, 1 |
| 1, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

338, 348, 364, 457, 634, 665, 675, 727

Game 305

| | |
|---|---|
| 1, 0 | 3, 1 |
| 2, 1 | 0, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

338, 339, 364, 391, 674, 675, 700, 727

Game 312

| | |
|---|---|
| 1, 2 | 0, 3 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

347, 351, 376, 484, 607, 662, 666, 715

Game 313

| | |
|---|---|
| 1, 3 | 0, 2 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

338, 351, 364, 484, 619, 666, 671, 715

Game 316

| | |
|---|---|
| 3, 0 | 2, 1 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

341, 351, 376, 445, 646, 662, 672, 715

Game 317

| | |
|---|---|
| 2, 0 | 3, 1 |
| 0, 1 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

338, 351, 364, 457, 646, 665, 672, 715

Game 319

| | |
|---|---|
| 1, 1 | 0, 3 |
| 0, 2 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

350, 351, 376, 403, 662, 663, 688, 715

Game 320

| | |
|---|---|
| 1, 1 | 0, 2 |
| 0, 3 | 1, 0 |

Ord% 48→0.000732422

Card% 75600→0.000756

350, 351, 376, 403, 662, 663, 688, 715

## C.13 Games of Type 13

Total Count of Type: 171

Game 131

| | |
|---|---|
| 2, 2 | 0, 1 |
| 1, 0 | 0, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

225, 225, 239, 239, 365, 365, 603, 603

Game 132

| | |
|---|---|
| 2, 1 | 0, 2 |
| 1, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

225, 225, 230, 239, 360, 374, 603, 608

Game 134

| | |
|---|---|
| 2, 2 | 1, 1 |
| 0, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

237, 237, 365, 365, 455, 549, 549, 611

Game 137

| 2, 2 | 1, 1 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

238, 243, 365, 365, 428, 576, 610, 711

Game 138

| 0, 2 | 1, 1 |
|---|---|
| 2, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

230, 241, 360, 376, 401, 609, 657, 662

Game 139

| 1, 2 | 1, 1 |
|---|---|
| 2, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

230, 243, 320, 360, 371, 527, 606, 711

Game 140

| 2, 2 | 1, 1 |
|---|---|
| 2, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

117, 122, 221, 243, 356, 482, 612, 711

Game 143

| 1, 2 | 2, 1 |
|---|---|
| 0, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

237, 237, 368, 374, 446, 549, 549, 608

Game 144

| 0, 2 | 2, 1 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

231, 241, 374, 376, 387, 608, 657, 662

Game 145

| 1, 2 | 2, 1 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

144, 203, 222, 243, 359, 374, 608, 711

Game 146

| 2, 2 | 2, 1 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

240, 243, 284, 293, 362, 605, 630, 711

Game 148

| 1, 2 | 2, 1 |
|---|---|
| 2, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

229, 243, 333, 347, 377, 607, 689, 711

Game 153

| | |
|---|---|
| 0, 1 | 2, 2 |
| 1, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

231, 241, 365, 365, 387, 619, 657, 671

Game 156

| | |
|---|---|
| 2, 2 | 0, 1 |
| 1, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

228, 236, 306, 365, 365, 522, 621, 725

Game 157

| | |
|---|---|
| 0, 2 | 0, 1 |
| 2, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

230, 233, 360, 372, 441, 554, 563, 615

Game 158

| | |
|---|---|
| 1, 2 | 0, 1 |
| 2, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

230, 232, 360, 373, 414, 581, 621, 725

Game 159

| | |
|---|---|
| 2, 2 | 0, 1 |
| 2, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

117, 122, 221, 234, 356, 468, 621, 725

Game 163

| | |
|---|---|
| 1, 1 | 2, 2 |
| 1, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

144, 212, 222, 243, 365, 365, 602, 711

Game 165

| | |
|---|---|
| 0, 2 | 1, 1 |
| 2, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

158, 198, 224, 230, 360, 378, 600, 716

Game 166

| | |
|---|---|
| 2, 1 | 1, 2 |
| 1, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

238, 243, 367, 374, 419, 576, 608, 711

Game 167

| | |
|---|---|
| 2, 1 | 2, 2 |
| 1, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

240, 243, 284, 293, 362, 605, 630, 711

Game 169

| | |
|---|---|
| 0, 2 | 2, 1 |
| 1, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

228, 236, 306, 374, 378, 522, 608, 716

Game 170

| | |
|---|---|
| 0, 2 | 2, 1 |
| 2, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

229, 242, 333, 347, 378, 607, 684, 716

Game 174

| | |
|---|---|
| 0, 1 | 1, 2 |
| 2, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

230, 241, 360, 366, 392, 619, 657, 671

Game 175

| | |
|---|---|
| 0, 1 | 0, 2 |
| 2, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

230, 233, 360, 372, 441, 554, 563, 615

Game 176

| | |
|---|---|
| 0, 1 | 1, 2 |
| 2, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

149, 198, 224, 230, 357, 360, 621, 725

Game 177

| | |
|---|---|
| 0, 1 | 2, 2 |
| 2, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

229, 242, 333, 338, 364, 621, 684, 725

Game 179

| | |
|---|---|
| 0, 2 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

227, 227, 279, 279, 375, 621, 635, 725

Game 181

| | |
|---|---|
| 1, 1 | 0, 2 |
| 2, 0 | 0, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

225, 225, 230, 230, 360, 360, 617, 617

Game 182

| | |
|---|---|
| 1, 1 | 1, 2 |
| 2, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

230, 243, 311, 360, 363, 536, 614, 711

Game 183

| | |
|---|---|
| 1, 1 | 2, 2 |
| 2, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

229, 243, 333, 338, 364, 620, 698, 711

Game 185

| | |
|---|---|
| 1, 1 | 0, 2 |
| 2, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

230, 232, 360, 378, 414, 590, 616, 716

Game 186

| | |
|---|---|
| 0, 2 | 1, 1 |
| 2, 0 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

227, 227, 279, 279, 378, 618, 644, 716

Game 188

| | |
|---|---|
| 2, 1 | 1, 2 |
| 2, 0 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

117, 131, 221, 243, 369, 473, 599, 711

Game 189

| | |
|---|---|
| 2, 1 | 0, 2 |
| 2, 0 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

117, 131, 221, 234, 378, 468, 599, 716

Game 230

| | |
|---|---|
| 2, 2 | 1, 0 |
| 0, 1 | 0, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

365, 365, 453, 453, 557, 557, 561, 561

Game 233

| | |
|---|---|
| 2, 2 | 1, 1 |
| 0, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

365, 365, 426, 454, 556, 567, 584, 723

Game 234

| | |
|---|---|
| 2, 0 | 1, 2 |
| 0, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

368, 376, 399, 446, 555, 565, 662, 669

Game 235

| | |
|---|---|
| 2, 1 | 1, 2 |
| 0, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

318, 368, 371, 446, 527, 552, 567, 723

Game 236

| | |
|---|---|
| 2, 2 | 1, 2 |
| 0, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

122, 125, 356, 437, 480, 558, 567, 723

Game 239

| | |
|---|---|
| 1, 2 | 2, 0 |
| 0, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

368, 372, 446, 453, 554, 557, 561, 561

Game 240

| | |
|---|---|
| 0, 2 | 2, 0 |
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

372, 376, 395, 447, 554, 565, 662, 669

Game 241

| | |
|---|---|
| 1, 2 | 2, 0 |
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

152, 203, 359, 372, 438, 554, 567, 723

Game 242

| | |
|---|---|
| 2, 2 | 2, 0 |
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

284, 291, 362, 456, 551, 567, 638, 723

Game 244

| | |
|---|---|
| 1, 2 | 2, 0 |
| 2, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

341, 345, 377, 445, 553, 567, 689, 723

Game 249

| | |
|---|---|
| 1, 0 | 2, 2 |
| 0, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

365, 365, 399, 457, 555, 565, 665, 669

Game 251

| | |
|---|---|
| 1, 1 | 2, 2 |
| 0, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

318, 365, 365, 452, 530, 552, 567, 723

Game 252

| | |
|---|---|
| 1, 2 | 2, 1 |
| 0, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

368, 373, 426, 446, 556, 567, 581, 723

Game 253

| | |
|---|---|
| 1, 2 | 2, 2 |
| 0, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

122, 125, 356, 437, 480, 558, 567, 723

Game 257

| | |
|---|---|
| 2, 2 | 1, 0 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

156, 210, 365, 365, 459, 546, 548, 719

Game 259

| | |
|---|---|
| 0, 2 | 1, 0 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

156, 210, 368, 378, 446, 546, 548, 716

Game 260

| | |
|---|---|
| 1, 2 | 2, 0 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

367, 372, 419, 459, 554, 562, 588, 719

Game 261

| | |
|---|---|
| 2, 2 | 2, 0 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

284, 291, 362, 459, 551, 564, 642, 719

Game 263

| | |
|---|---|
| 0, 2 | 2, 0 |
| 1, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

314, 372, 378, 444, 534, 554, 560, 716

Game 264

| | |
|---|---|
| 0, 2 | 2, 0 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

341, 345, 378, 445, 553, 566, 696, 716

Game 268

| | |
|---|---|
| 1, 0 | 2, 1 |
| 0, 2 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

366, 372, 392, 457, 554, 565, 665, 669

Game 269

| | |
|---|---|
| 1, 1 | 2, 0 |
| 0, 2 | 0, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

372, 372, 449, 449, 554, 554, 561, 561

Game 270

| 1, 1 | 2, 1 |
|---|---|
| 0, 2 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

149, 206, 357, 372, 440, 554, 567, 723

Game 271

| 1, 1 | 2, 2 |
|---|---|
| 0, 2 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

338, 345, 364, 458, 553, 567, 692, 723

Game 273

| 1, 2 | 2, 1 |
|---|---|
| 0, 2 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

287, 291, 375, 443, 551, 567, 635, 723

Game 275

| 0, 1 | 0, 2 |
|---|---|
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

311, 363, 372, 459, 534, 554, 560, 719

Game 276

| 0, 1 | 2, 0 |
|---|---|
| 2, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

338, 345, 364, 459, 553, 566, 696, 719

Game 278

| 1, 1 | 2, 0 |
|---|---|
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

372, 378, 422, 448, 554, 562, 588, 716

Game 279

| 2, 1 | 2, 0 |
|---|---|
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

287, 291, 378, 443, 551, 564, 642, 716

Game 281

| 0, 2 | 0, 2 |
|---|---|
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

129, 129, 369, 459, 473, 545, 545, 719

Game 282

| 0, 2 | 2, 0 |
|---|---|
| 1, 1 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

129, 129, 378, 450, 476, 545, 545, 716

Game 376

| 2, 2 | 1, 1 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 64→0.000976562

Card%

57600→0.000576

365, 365, 427, 427, 583, 583, 729, 729

Game 377

| 2, 0 | 1, 2 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

367, 376, 400, 419, 582, 662, 675, 727

Game 378

| 2, 1 | 1, 2 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

319, 367, 371, 419, 527, 579, 729, 729

Game 379

| 2, 2 | 1, 2 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

122, 124, 356, 410, 481, 585, 729, 729

Game 382

| 0, 2 | 2, 1 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

373, 376, 394, 420, 581, 662, 675, 727

Game 383

| 1, 2 | 2, 1 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

151, 203, 359, 373, 411, 581, 729, 729

Game 384

| 2, 2 | 2, 1 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

284, 292, 362, 429, 578, 637, 729, 729

Game 386

| 1, 2 | 2, 1 |
|---|---|
| 2, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

340, 346, 377, 418, 580, 689, 729, 729

Game 390

| 2, 2 | 0, 1 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

365, 365, 405, 430, 592, 664, 670, 717

Game 391

| | |
|---|---|
| 1, 1 | 2, 2 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

324, 365, 365, 425, 529, 594, 714, 724

Game 392

| | |
|---|---|
| 1, 2 | 2, 1 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

367, 373, 419, 432, 581, 594, 718, 724

Game 393

| | |
|---|---|
| 1, 2 | 2, 2 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

122, 124, 356, 410, 486, 594, 720, 724

Game 396

| | |
|---|---|
| 2, 2 | 1, 1 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

162, 211, 365, 365, 432, 575, 708, 718

Game 397

| | |
|---|---|
| 0, 2 | 1, 1 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

157, 216, 367, 378, 419, 573, 710, 716

Game 398

| | |
|---|---|
| 2, 2 | 2, 1 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

284, 292, 362, 432, 578, 648, 718, 726

Game 400

| | |
|---|---|
| 0, 2 | 2, 1 |
| 1, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

313, 373, 378, 417, 540, 581, 716, 722

Game 401

| | |
|---|---|
| 0, 2 | 2, 1 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

340, 346, 378, 418, 580, 702, 716, 728

Game 405

| | |
|---|---|
| 1, 0 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

366, 378, 392, 430, 592, 664, 670, 716

Game 406

| | |
|---|---|
| 1, 1 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

149, 205, 357, 378, 413, 594, 716, 724

Game 407

| | |
|---|---|
| 1, 1 | 2, 2 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

338, 351, 364, 431, 594, 691, 715, 724

Game 409

| | |
|---|---|
| 1, 2 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

286, 297, 375, 416, 594, 635, 713, 724

Game 411

| | |
|---|---|
| 1, 1 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

311, 363, 378, 432, 535, 587, 716, 718

Game 412

| | |
|---|---|
| 1, 1 | 0, 2 |
| 2, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

338, 351, 364, 432, 593, 697, 715, 718

Game 414

| | |
|---|---|
| 0, 2 | 1, 1 |
| 1, 1 | 2, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

378, 378, 421, 421, 589, 589, 716, 716

Game 415

| | |
|---|---|
| 0, 2 | 1, 1 |
| 2, 1 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

286, 297, 378, 416, 591, 643, 713, 716

Game 417

| | |
|---|---|
| 1, 2 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

130, 135, 369, 432, 473, 572, 707, 718

Game 418

| | |
|---|---|
| 0, 2 | 2, 1 |
| 1, 1 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

130, 135, 378, 423, 475, 572, 707, 716

Game 427

| 0, 0 | 1, 2 |
|---|---|
| 2, 1 | 0, 0 |

Ord% 64→0.000976562

Card%

57600→0.000576

366, 366, 392, 392, 673, 673, 673, 673

Game 428

| 1, 0 | 1, 2 |
|---|---|
| 2, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

311, 363, 366, 392, 538, 668, 675, 727

Game 429

| 2, 0 | 1, 2 |
|---|---|
| 2, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

123, 133, 369, 383, 473, 653, 675, 727

Game 432

| 0, 0 | 2, 2 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 64→0.000976562

Card%

57600→0.000576

365, 365, 393, 393, 673, 673, 673, 673

Game 433

| 1, 0 | 2, 2 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

150, 214, 365, 365, 384, 656, 675, 727

Game 434

| 2, 0 | 2, 2 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

284, 295, 362, 402, 636, 659, 675, 727

Game 436

| 1, 0 | 2, 2 |
|---|---|
| 2, 1 | 0, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

338, 339, 364, 391, 674, 675, 700, 727

Game 440

| 2, 0 | 0, 2 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

376, 376, 403, 403, 662, 662, 663, 663

Game 441

| 2, 0 | 0, 2 |
|---|---|
| 1, 1 | 1, 0 |

Ord% 128→0.00195312

Card%

115200→0.001152

322, 376, 378, 398, 528, 660, 662, 716

Game 442

| | |
|---|---|
| 2, 0 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

123, 133, 378, 383, 484, 653, 666, 716

Game 445

| | |
|---|---|
| 2, 0 | 1, 2 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

160, 203, 359, 376, 405, 654, 662, 717

Game 446

| | |
|---|---|
| 1, 0 | 2, 1 |
| 1, 2 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

149, 214, 357, 366, 392, 656, 675, 727

Game 447

| | |
|---|---|
| 2, 0 | 2, 2 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

284, 295, 362, 405, 646, 659, 672, 717

Game 449

| | |
|---|---|
| 0, 1 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

312, 365, 365, 390, 538, 668, 675, 727

Game 450

| | |
|---|---|
| 0, 1 | 2, 2 |
| 1, 2 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

338, 339, 364, 391, 674, 675, 700, 727

Game 454

| | |
|---|---|
| 2, 0 | 1, 1 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

160, 204, 376, 378, 386, 654, 662, 716

Game 455

| | |
|---|---|
| 2, 0 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

349, 349, 378, 404, 661, 661, 690, 716

Game 457

| | |
|---|---|
| 2, 0 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

285, 295, 378, 389, 646, 659, 672, 716

Game 459

| | |
|---|---|
| 2, 1 | 0, 2 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

322, 371, 376, 405, 527, 660, 662, 717

Game 460

| | |
|---|---|
| 2, 1 | 0, 2 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

349, 349, 377, 405, 661, 661, 689, 717

Game 462

| | |
|---|---|
| 2, 0 | 2, 1 |
| 1, 2 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

285, 295, 375, 389, 635, 659, 675, 727

Game 464

| | |
|---|---|
| 2, 2 | 0, 2 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

122, 133, 356, 405, 484, 653, 666, 717

Game 465

| | |
|---|---|
| 0, 2 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

122, 133, 356, 396, 474, 653, 675, 727

Game 474

| | |
|---|---|
| 1, 1 | 1, 2 |
| 2, 1 | 0, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

311, 311, 363, 363, 533, 533, 729, 729

Game 475

| | |
|---|---|
| 2, 1 | 1, 2 |
| 2, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

120, 128, 302, 369, 473, 518, 729, 729

Game 478

| | |
|---|---|
| 1, 1 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

147, 209, 303, 365, 365, 521, 729, 729

Game 479

| | |
|---|---|
| 2, 1 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

284, 290, 321, 362, 524, 633, 729, 729

Game 481

| | |
|---|---|
| 1, 1 | 2, 2 |
| 2, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

310, 336, 338, 364, 539, 695, 729, 729

Game 485

| | |
|---|---|
| 2, 1 | 0, 2 |
| 1, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

317, 324, 371, 378, 525, 527, 714, 716

Game 486

| | |
|---|---|
| 2, 1 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

120, 128, 302, 378, 486, 518, 716, 720

Game 489

| | |
|---|---|
| 2, 1 | 1, 2 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

162, 203, 324, 359, 371, 527, 708, 714

Game 490

| | |
|---|---|
| 0, 1 | 1, 2 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

149, 216, 311, 357, 363, 540, 710, 722

Game 491

| | |
|---|---|
| 2, 1 | 2, 2 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

284, 290, 324, 362, 524, 648, 714, 726

Game 493

| | |
|---|---|
| 0, 1 | 2, 2 |
| 1, 1 | 1, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

309, 309, 365, 365, 540, 540, 722, 722

Game 494

| | |
|---|---|
| 0, 1 | 2, 2 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

310, 336, 338, 364, 540, 702, 722, 728

Game 498

| | |
|---|---|
| 1, 1 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

155, 201, 305, 378, 378, 519, 716, 716

Game 499

| | |
|---|---|
| 2, 1 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

323, 344, 351, 378, 526, 687, 715, 716

Game 501

| | |
|---|---|
| 1, 1 | 0, 2 |
| 2, 1 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

282, 297, 308, 378, 537, 641, 713, 716

Game 503

| | |
|---|---|
| 2, 1 | 0, 2 |
| 1, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

324, 344, 351, 377, 526, 689, 714, 715

Game 505

| | |
|---|---|
| 0, 2 | 2, 1 |
| 1, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

282, 297, 308, 375, 540, 635, 713, 722

Game 507

| | |
|---|---|
| 2, 2 | 0, 2 |
| 1, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

122, 135, 324, 356, 479, 531, 707, 714

Game 508

| | |
|---|---|
| 0, 2 | 2, 2 |
| 1, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

122, 135, 315, 356, 471, 540, 707, 722

Game 517

| | |
|---|---|
| 2, 2 | 1, 2 |
| 2, 1 | 0, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

113, 113, 126, 126, 464, 464, 729, 729

Game 520

| | |
|---|---|
| 1, 2 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

116, 122, 153, 194, 356, 465, 729, 729

Game 521

| | |
|---|---|
| 2, 2 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

119, 119, 275, 275, 483, 639, 729, 729

Game 523

| | |
|---|---|
| 1, 2 | 2, 2 |
| 2, 1 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

121, 134, 329, 342, 472, 680, 729, 729

Game 527

| | |
|---|---|
| 2, 2 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

113, 113, 126, 135, 464, 486, 707, 720

Game 530

| | |
|---|---|
| 2, 2 | 1, 2 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

116, 122, 162, 194, 356, 486, 708, 720

Game 531

| | |
|---|---|
| 0, 2 | 1, 2 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

114, 135, 140, 216, 369, 473, 707, 710

Game 532

| | |
|---|---|
| 2, 2 | 2, 2 |
| 0, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

119, 119, 275, 275, 486, 648, 720, 726

Game 534

| | |
|---|---|
| 0, 2 | 2, 2 |
| 2, 1 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

121, 135, 329, 342, 472, 702, 707, 728

Game 538

| | |
|---|---|
| 1, 2 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

114, 135, 140, 207, 378, 467, 707, 716

Game 539

| | |
|---|---|
| 2, 2 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

121, 135, 329, 351, 485, 693, 707, 715

Game 541

| | |
|---|---|
| 1, 2 | 0, 2 |
| 2, 1 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

132, 135, 288, 297, 470, 626, 707, 713

Game 543

| | |
|---|---|
| 2, 1 | 0, 2 |
| 2, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

121, 134, 329, 351, 486, 680, 715, 720

Game 545

| | |
|---|---|
| 0, 2 | 1, 2 |
| 2, 1 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

132, 135, 288, 297, 470, 626, 707, 713

Game 547

| | |
|---|---|
| 0, 2 | 2, 2 |
| 1, 1 | 2, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

113, 113, 135, 135, 477, 477, 707, 707

Game 622

| | |
|---|---|
| 1, 1 | 2, 1 |
| 1, 2 | 0, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

149, 149, 197, 197, 357, 357, 729, 729

Game 623

| | |
|---|---|
| 2, 1 | 2, 1 |
| 1, 2 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

146, 200, 276, 278, 375, 635, 729, 729

Game 625

| | |
|---|---|
| 1, 1 | 2, 2 |
| 1, 2 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

148, 215, 330, 338, 364, 683, 729, 729

Game 629

| | |
|---|---|
| 2, 1 | 1, 1 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

143, 162, 195, 203, 359, 378, 708, 716

Game 630

| | |
|---|---|
| 0, 1 | 1, 1 |
| 2, 2 | 1, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

141, 141, 216, 216, 365, 365, 710, 710

Game 631

| | |
|---|---|
| 2, 1 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

146, 200, 276, 278, 378, 648, 716, 726

Game 633

| | |
|---|---|
| 0, 1 | 2, 1 |
| 2, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

148, 216, 330, 338, 364, 702, 710, 728

Game 637

| | |
|---|---|
| 2, 1 | 1, 2 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

162, 202, 332, 351, 377, 689, 708, 715

Game 639

| | |
|---|---|
| 2, 2 | 1, 1 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

162, 213, 284, 297, 362, 629, 708, 713

Game 641

| | |
|---|---|
| 2, 1 | 1, 1 |
| 0, 2 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

161, 202, 332, 351, 378, 681, 715, 716

Game 643

| | |
|---|---|
| 0, 2 | 1, 1 |
| 2, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

159, 216, 284, 297, 362, 627, 710, 713

Game 649

| | |
|---|---|
| 2, 2 | 2, 1 |
| 1, 2 | 0, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

281, 281, 294, 294, 632, 632, 729, 729

Game 651

| | |
|---|---|
| 2, 1 | 2, 2 |
| 1, 2 | 0, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

283, 296, 335, 348, 634, 686, 729, 729

Game 655

| | |
|---|---|
| 2, 2 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

281, 281, 294, 297, 632, 648, 713, 726

Game 657

| | |
|---|---|
| 0, 2 | 2, 1 |
| 2, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

283, 297, 335, 348, 634, 702, 713, 728

Game 661

| 2, 1 | 2, 2 |
|---|---|
| 0, 2 | 1, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

283, 296, 335, 351, 648, 686, 715, 726

Game 664

| 2, 2 | 1, 1 |
|---|---|
| 0, 2 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

283, 297, 335, 351, 647, 699, 713, 715

Game 666

| 0, 2 | 1, 1 |
|---|---|
| 2, 2 | 2, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

281, 281, 297, 297, 645, 645, 713, 713

Game 691

| 1, 1 | 2, 2 |
|---|---|
| 2, 2 | 0, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

337, 337, 337, 337, 701, 701, 729, 729

Game 696

| 0, 1 | 2, 2 |
|---|---|
| 2, 2 | 1, 0 |

Ord% 64→0.000976562

Card% 57600→0.000576

337, 337, 337, 337, 702, 702, 728, 728

Game 699

| 2, 1 | 1, 2 |
|---|---|
| 0, 2 | 2, 0 |

Ord% 128→0.00195312

Card% 115200→0.001152

350, 350, 351, 351, 688, 688, 715, 715

## C.14 Games of Type 14

Total Count of Type: 216

Game 141

| 3, 2 | 1, 1 |
|---|---|
| 2, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

239, 243, 365, 365, 482, 603, 612, 711

Game 142

| 2, 2 | 1, 1 |
|---|---|
| 3, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 243, 360, 374, 482, 608, 612, 711

Game 147

| 3, 2 | 2, 1 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

240, 243, 365, 365, 455, 611, 630, 711

Game 149

| | |
|---|---|
| 1, 2 | 2, 1 |
| 3, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 243, 360, 377, 401, 609, 689, 711

Game 150

| | |
|---|---|
| 2, 2 | 3, 1 |
| 1, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

240, 243, 368, 374, 446, 608, 630, 711

Game 151

| | |
|---|---|
| 1, 2 | 3, 1 |
| 2, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

231, 243, 374, 377, 387, 608, 689, 711

Game 160

| | |
|---|---|
| 3, 2 | 0, 1 |
| 2, 0 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

234, 239, 365, 365, 468, 603, 621, 725

Game 161

| | |
|---|---|
| 2, 2 | 0, 1 |
| 3, 0 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 234, 360, 374, 468, 608, 621, 725

Game 168

| | |
|---|---|
| 2, 1 | 3, 2 |
| 1, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

240, 243, 365, 365, 455, 611, 630, 711

Game 171

| | |
|---|---|
| 0, 2 | 2, 1 |
| 3, 0 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 242, 360, 378, 401, 609, 684, 716

Game 172

| | |
|---|---|
| 3, 1 | 2, 2 |
| 1, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

240, 243, 368, 374, 446, 608, 630, 711

Game 173

| | |
|---|---|
| 0, 2 | 3, 1 |
| 2, 0 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

231, 242, 374, 378, 387, 608, 684, 716

Game 178

| | |
|---|---|
| 0, 1 | 3, 2 |
| 2, 0 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

231, 242, 365, 365, 387, 621, 684, 725

Game 180

| | |
|---|---|
| 0, 3 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 233, 360, 375, 441, 621, 635, 725

Game 184

| | |
|---|---|
| 1, 1 | 3, 2 |
| 2, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

231, 243, 365, 365, 387, 620, 698, 711

Game 187

| | |
|---|---|
| 0, 2 | 1, 1 |
| 3, 0 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 233, 360, 378, 441, 618, 644, 716

Game 190

| | |
|---|---|
| 3, 1 | 1, 2 |
| 2, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

239, 243, 369, 374, 473, 603, 608, 711

Game 191

| | |
|---|---|
| 3, 1 | 0, 2 |
| 2, 0 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

234, 239, 374, 378, 468, 603, 608, 716

Game 192

| | |
|---|---|
| 0, 1 | 2, 2 |
| 3, 0 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 242, 360, 366, 392, 621, 684, 725

Game 193

| | |
|---|---|
| 0, 2 | 2, 1 |
| 0, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 233, 360, 375, 441, 621, 635, 725

Game 194

| | |
|---|---|
| 1, 1 | 2, 2 |
| 3, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 243, 360, 366, 392, 620, 698, 711

Game 195

| | |
|---|---|
| 1, 1 | 0, 2 |
| 3, 0 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 233, 360, 378, 441, 618, 644, 716

Game 196

| | |
|---|---|
| 2, 1 | 1, 2 |
| 3, 0 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 243, 360, 369, 473, 617, 617, 711

Game 197

| | |
|---|---|
| 2, 1 | 0, 2 |
| 3, 0 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

230, 234, 360, 378, 468, 617, 617, 716

Game 237

| | |
|---|---|
| 2, 3 | 1, 2 |
| 0, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 455, 480, 558, 567, 611, 723

Game 238

| | |
|---|---|
| 2, 2 | 1, 3 |
| 0, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 374, 446, 480, 558, 567, 608, 723

Game 243

| | |
|---|---|
| 3, 2 | 2, 0 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 453, 456, 557, 567, 638, 723

Game 245

| | |
|---|---|
| 1, 2 | 2, 0 |
| 3, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 377, 399, 446, 555, 567, 689, 723

Game 246

| | |
|---|---|
| 2, 2 | 3, 0 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 372, 446, 456, 554, 567, 638, 723

Game 247

| | |
|---|---|
| 1, 2 | 3, 0 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

372, 377, 395, 447, 554, 567, 689, 723

Game 254

| | |
|---|---|
| 1, 2 | 2, 3 |
| 0, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 455, 480, 558, 567, 611, 723

Game 255

| | |
|---|---|
| 1, 3 | 2, 2 |
| 0, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 374, 446, 480, 558, 567, 608, 723

Game 262

| | |
|---|---|
| 3, 2 | 2, 0 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 453, 459, 557, 564, 642, 719

Game 265

| | |
|---|---|
| 0, 2 | 2, 0 |
| 3, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 378, 399, 446, 555, 566, 696, 716

Game 266

| | |
|---|---|
| 2, 2 | 3, 0 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 372, 446, 459, 554, 564, 642, 719

Game 267

| | |
|---|---|
| 0, 2 | 3, 0 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

372, 378, 395, 447, 554, 566, 696, 716

Game 272

| | |
|---|---|
| 1, 1 | 2, 3 |
| 0, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 399, 458, 555, 567, 692, 723

Game 274

| | |
|---|---|
| 1, 3 | 2, 1 |
| 0, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 375, 446, 453, 557, 567, 635, 723

Game 277

| | |
|---|---|
| 0, 1 | 2, 0 |
| 3, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 399, 459, 555, 566, 696, 719

Game 280

| | |
|---|---|
| 3, 1 | 2, 0 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 378, 446, 453, 557, 564, 642, 716

Game 283

| | |
|---|---|
| 0, 3 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

369, 372, 459, 473, 554, 563, 615, 719

Game 284

| | |
|---|---|
| 0, 2 | 3, 0 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

372, 378, 450, 476, 554, 563, 615, 716

Game 285

| | |
|---|---|
| 1, 1 | 3, 0 |
| 2, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

366, 372, 392, 458, 554, 567, 692, 723

Game 286

| | |
|---|---|
| 2, 1 | 3, 0 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

372, 375, 449, 449, 554, 567, 635, 723

Game 287

| | |
|---|---|
| 0, 1 | 3, 0 |
| 2, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

366, 372, 392, 459, 554, 566, 696, 719

Game 288

| | |
|---|---|
| 2, 1 | 3, 0 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

372, 378, 449, 449, 554, 564, 642, 716

Game 289

| | |
|---|---|
| 0, 2 | 0, 3 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

369, 372, 459, 473, 554, 563, 615, 719

Game 290

| | |
|---|---|
| 0, 3 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

372, 378, 450, 476, 554, 563, 615, 716

Game 380

| | |
|---|---|
| 2, 3 | 1, 2 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 428, 481, 585, 610, 729, 729

Game 381

| | |
|---|---|
| 2, 2 | 1, 3 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

367, 374, 419, 481, 585, 608, 729, 729

Game 385

| | |
|---|---|
| 3, 2 | 2, 1 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 429, 454, 584, 637, 729, 729

Game 387

| | |
|---|---|
| 1, 2 | 2, 1 |
| 3, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

367, 377, 400, 419, 582, 689, 729, 729

Game 388

| | |
|---|---|
| 2, 2 | 3, 1 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 373, 429, 446, 581, 637, 729, 729

Game 389

| | |
|---|---|
| 1, 2 | 3, 1 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

373, 377, 394, 420, 581, 689, 729, 729

Game 394

| | |
|---|---|
| 1, 2 | 2, 3 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 428, 486, 594, 610, 720, 724

Game 395

| | |
|---|---|
| 1, 3 | 2, 2 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

367, 374, 419, 486, 594, 608, 720, 724

Game 399

| | |
|---|---|
| 3, 2 | 2, 1 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 432, 454, 584, 648, 718, 726

Game 402

| | |
|---|---|
| 0, 2 | 2, 1 |
| 3, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

367, 378, 400, 419, 582, 702, 716, 728

Game 403

| | |
|---|---|
| 2, 2 | 3, 1 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 373, 432, 446, 581, 648, 718, 726

Game 404

| | |
|---|---|
| 0, 2 | 3, 1 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

373, 378, 394, 420, 581, 702, 716, 728

Game 408

| | |
|---|---|
| 1, 1 | 2, 3 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 405, 431, 594, 691, 717, 724

Game 410

| | |
|---|---|
| 1, 3 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

367, 375, 419, 459, 594, 635, 719, 724

Game 413

| | |
|---|---|
| 1, 1 | 0, 2 |
| 2, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 405, 432, 593, 697, 717, 718

Game 416

| | |
|---|---|
| 0, 2 | 1, 1 |
| 3, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

367, 378, 419, 459, 591, 643, 716, 719

Game 419

| | |
|---|---|
| 1, 3 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

369, 373, 432, 473, 581, 621, 718, 725

Game 420

| | |
|---|---|
| 0, 2 | 3, 1 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

373, 378, 423, 475, 581, 621, 716, 725

Game 421

| | |
|---|---|
| 1, 1 | 2, 2 |
| 0, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

366, 378, 392, 431, 594, 691, 716, 724

Game 422

| | |
|---|---|
| 1, 2 | 2, 1 |
| 0, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

375, 378, 422, 448, 594, 635, 716, 724

Game 423

| | |
|---|---|
| 1, 1 | 0, 3 |
| 2, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

366, 378, 392, 432, 593, 697, 716, 718

Game 424

| | |
|---|---|
| 0, 3 | 1, 1 |
| 1, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

378, 378, 422, 448, 591, 643, 716, 716

Game 425

| | |
|---|---|
| 1, 2 | 0, 3 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

369, 378, 432, 473, 590, 616, 716, 718

Game 426

| | |
|---|---|
| 0, 3 | 1, 2 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

378, 378, 423, 475, 590, 616, 716, 716

Game 430

| | |
|---|---|
| 3, 0 | 1, 2 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

369, 376, 401, 473, 609, 662, 675, 727

Game 431

| | |
|---|---|
| 0, 2 | 1, 3 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

366, 369, 392, 473, 619, 671, 675, 727

Game 435

| | |
|---|---|
| 3, 0 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 376, 402, 446, 636, 662, 675, 727

Game 437

| | |
|---|---|
| 1, 0 | 2, 2 |
| 3, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

366, 366, 392, 392, 674, 675, 700, 727

Game 438

| | |
|---|---|
| 2, 0 | 3, 2 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 402, 457, 636, 665, 675, 727

Game 439

| | |
|---|---|
| 1, 0 | 3, 2 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 393, 393, 674, 675, 700, 727

Game 443

| | |
|---|---|
| 3, 0 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

376, 378, 401, 484, 609, 662, 666, 716

Game 444

| | |
|---|---|
| 2, 0 | 0, 2 |
| 3, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

366, 378, 392, 484, 619, 666, 671, 716

Game 448

| | |
|---|---|
| 3, 0 | 2, 2 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

368, 376, 405, 446, 646, 662, 672, 717

Game 451

| | |
|---|---|
| 1, 0 | 3, 1 |
| 2, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

366, 366, 392, 392, 674, 675, 700, 727

Game 452

| | |
|---|---|
| 2, 0 | 3, 2 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 405, 457, 646, 665, 672, 717

Game 453

| | |
|---|---|
| 0, 1 | 2, 3 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 393, 393, 674, 675, 700, 727

Game 456

| | |
|---|---|
| 3, 0 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

376, 378, 403, 404, 662, 663, 690, 716

Game 458

| | |
|---|---|
| 2, 0 | 3, 1 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

366, 378, 392, 457, 646, 665, 672, 716

Game 461

| | |
|---|---|
| 2, 1 | 0, 2 |
| 0, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

376, 377, 403, 405, 662, 663, 689, 717

Game 463

| | |
|---|---|
| 2, 0 | 3, 1 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

366, 375, 392, 457, 635, 665, 675, 727

Game 466

| | |
|---|---|
| 2, 3 | 0, 2 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 405, 484, 619, 666, 671, 717

Game 467

| | |
|---|---|
| 0, 2 | 2, 3 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

365, 365, 396, 474, 619, 671, 675, 727

Game 468

| | |
|---|---|
| 3, 0 | 1, 1 |
| 0, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

376, 378, 403, 404, 662, 663, 690, 716

Game 469

| | |
|---|---|
| 3, 0 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

376, 378, 395, 447, 646, 662, 672, 716

Game 470

| | |
|---|---|
| 2, 1 | 0, 3 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

376, 377, 403, 405, 662, 663, 689, 717

Game 471

| | |
|---|---|
| 3, 0 | 2, 1 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

375, 376, 395, 447, 635, 662, 675, 727

Game 472

| | |
|---|---|
| 2, 2 | 0, 3 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

374, 376, 405, 484, 608, 662, 666, 717

Game 473

| | |
|---|---|
| 0, 3 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

374, 376, 396, 474, 608, 662, 675, 727

Game 476

| | |
|---|---|
| 3, 1 | 1, 2 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

320, 369, 371, 473, 527, 606, 729, 729

Game 477

| | |
|---|---|
| 1, 2 | 1, 3 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

311, 363, 369, 473, 536, 614, 729, 729

Game 480

| | |
|---|---|
| 3, 1 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

321, 368, 371, 446, 527, 633, 729, 729

Game 482

| | |
|---|---|
| 1, 1 | 2, 2 |
| 3, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

311, 363, 366, 392, 539, 695, 729, 729

Game 483

| | |
|---|---|
| 2, 1 | 3, 2 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

321, 365, 365, 452, 530, 633, 729, 729

Game 484

| | |
|---|---|
| 1, 1 | 3, 2 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

312, 365, 365, 390, 539, 695, 729, 729

Game 487

| | |
|---|---|
| 3, 1 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

320, 371, 378, 486, 527, 606, 716, 720

Game 488

| | |
|---|---|
| 2, 1 | 0, 2 |
| 3, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

311, 363, 378, 486, 536, 614, 716, 720

Game 492

| | |
|---|---|
| 3, 1 | 2, 2 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

324, 368, 371, 446, 527, 648, 714, 726

Game 495

| | |
|---|---|
| 0, 1 | 2, 2 |
| 3, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

311, 363, 366, 392, 540, 702, 722, 728

Game 496

| | |
|---|---|
| 2, 1 | 3, 2 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

324, 365, 365, 452, 530, 648, 714, 726

Game 497

| | |
|---|---|
| 0, 1 | 3, 2 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

312, 365, 365, 390, 540, 702, 722, 728

Game 500

| | |
|---|---|
| 3, 1 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

323, 371, 378, 405, 527, 687, 716, 717

Game 502

| | |
|---|---|
| 1, 1 | 0, 2 |
| 3, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

311, 363, 378, 459, 537, 641, 716, 719

Game 504

| | |
|---|---|
| 2, 1 | 0, 2 |
| 1, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

324, 371, 377, 405, 527, 689, 714, 717

Game 506

| | |
|---|---|
| 0, 2 | 2, 1 |
| 1, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

311, 363, 375, 459, 540, 635, 719, 722

Game 509

| | |
|---|---|
| 2, 3 | 0, 2 |
| 1, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

324, 365, 365, 479, 531, 621, 714, 725

Game 510

| | |
|---|---|
| 0, 2 | 2, 3 |
| 1, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

315, 365, 365, 471, 540, 621, 722, 725

Game 511

| | |
|---|---|
| 1, 1 | 1, 2 |
| 0, 3 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

323, 378, 378, 398, 528, 687, 716, 716

Game 512

| | |
|---|---|
| 1, 2 | 1, 1 |
| 0, 3 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

314, 378, 378, 444, 537, 641, 716, 716

Game 513

| | |
|---|---|
| 2, 1 | 0, 3 |
| 1, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

324, 377, 378, 398, 528, 689, 714, 716

Game 514

| | |
|---|---|
| 0, 3 | 2, 1 |
| 1, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

314, 375, 378, 444, 540, 635, 716, 722

Game 515

| | |
|---|---|
| 2, 2 | 0, 3 |
| 1, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

324, 374, 378, 479, 531, 608, 714, 716

Game 516

| | |
|---|---|
| 0, 3 | 2, 2 |
| 1, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

315, 374, 378, 471, 540, 608, 716, 722

Game 518

| | |
|---|---|
| 3, 2 | 1, 2 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 126, 356, 464, 482, 612, 729, 729

Game 519

| | |
|---|---|
| 2, 2 | 1, 3 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

126, 131, 369, 464, 473, 599, 729, 729

Game 522

| | |
|---|---|
| 3, 2 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 125, 356, 437, 483, 639, 729, 729

Game 524

| | |
|---|---|
| 1, 2 | 2, 2 |
| 3, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

123, 134, 369, 383, 473, 680, 729, 729

Game 525

| | |
|---|---|
| 2, 2 | 3, 2 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 125, 356, 437, 483, 639, 729, 729

Game 526

| | |
|---|---|
| 1, 2 | 3, 2 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 134, 356, 396, 474, 680, 729, 729

Game 528

| | |
|---|---|
| 3, 2 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 135, 356, 482, 486, 612, 707, 720

Game 529

| | |
|---|---|
| 2, 2 | 0, 2 |
| 3, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

131, 135, 369, 473, 486, 599, 707, 720

Game 533

| | |
|---|---|
| 3, 2 | 2, 2 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 125, 356, 437, 486, 648, 720, 726

Game 535

| | |
|---|---|
| 0, 2 | 2, 2 |
| 3, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

123, 135, 369, 383, 473, 702, 707, 728

Game 536

| | |
|---|---|
| 2, 2 | 3, 2 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 125, 356, 437, 486, 648, 720, 726

Game 537

| | |
|---|---|
| 0, 2 | 3, 2 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 135, 356, 396, 474, 702, 707, 728

Game 540

| | |
|---|---|
| 3, 2 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 135, 356, 405, 485, 693, 707, 717

Game 542

| | |
|---|---|
| 1, 2 | 0, 2 |
| 3, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

132, 135, 369, 459, 473, 626, 707, 719

Game 544

| | |
|---|---|
| 2, 1 | 0, 2 |
| 2, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 134, 356, 405, 486, 680, 717, 720

Game 546

| | |
|---|---|
| 0, 2 | 1, 2 |
| 3, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

132, 135, 369, 459, 473, 626, 707, 719

Game 548

| | |
|---|---|
| 2, 3 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 126, 356, 464, 486, 621, 720, 725

Game 549

| | |
|---|---|
| 0, 2 | 3, 2 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

122, 135, 356, 477, 477, 621, 707, 725

Game 550

| | |
|---|---|
| 1, 1 | 2, 2 |
| 0, 3 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

123, 135, 378, 383, 485, 693, 707, 716

Game 551

| | |
|---|---|
| 1, 2 | 2, 1 |
| 0, 3 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

132, 135, 378, 450, 476, 626, 707, 716

Game 552

| | |
|---|---|
| 2, 1 | 0, 3 |
| 2, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

123, 134, 378, 383, 486, 680, 716, 720

Game 553

| | |
|---|---|
| 0, 3 | 2, 1 |
| 1, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

132, 135, 378, 450, 476, 626, 707, 716

Game 554

| | |
|---|---|
| 2, 2 | 0, 3 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

126, 131, 378, 464, 486, 599, 716, 720

Game 555

| | |
|---|---|
| 0, 3 | 2, 2 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

131, 135, 378, 477, 477, 599, 707, 716

Game 558

| | |
|---|---|
| 1, 3 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

153, 203, 359, 374, 465, 608, 729, 729

Game 559

| | |
|---|---|
| 2, 3 | 2, 2 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 293, 362, 483, 605, 639, 729, 729

Game 561

| | |
|---|---|
| 1, 3 | 2, 2 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

342, 347, 377, 472, 607, 689, 729, 729

Game 567

| | |
|---|---|
| 2, 3 | 1, 2 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

162, 212, 365, 365, 486, 602, 708, 720

Game 568

| | |
|---|---|
| 0, 3 | 1, 2 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

158, 216, 369, 378, 473, 600, 710, 716

Game 569

| | |
|---|---|
| 2, 3 | 2, 2 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 293, 362, 486, 605, 648, 720, 726

Game 571

| | |
|---|---|
| 0, 3 | 2, 2 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

342, 347, 378, 472, 607, 702, 716, 728

Game 575

| | |
|---|---|
| 1, 3 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

149, 207, 357, 378, 467, 621, 716, 725

Game 576

| | |
|---|---|
| 2, 3 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 351, 364, 485, 621, 693, 715, 725

Game 578

| | |
|---|---|
| 1, 3 | 0, 2 |
| 2, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

288, 297, 375, 470, 621, 635, 713, 725

Game 580

| | |
|---|---|
| 2, 1 | 0, 2 |
| 3, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 351, 364, 486, 620, 698, 715, 720

Game 582

| | |
|---|---|
| 0, 3 | 1, 2 |
| 2, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

288, 297, 378, 470, 618, 644, 713, 716

Game 592

| | |
|---|---|
| 1, 2 | 2, 3 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

153, 212, 365, 365, 465, 602, 729, 729

Game 593

| | |
|---|---|
| 2, 2 | 2, 3 |
| 1, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 293, 362, 483, 605, 639, 729, 729

Game 595

| | |
|---|---|
| 1, 2 | 2, 3 |
| 2, 1 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 342, 364, 472, 620, 698, 729, 729

Game 600

| | |
|---|---|
| 2, 2 | 1, 3 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

162, 203, 359, 374, 486, 608, 708, 720

Game 601

| | |
|---|---|
| 0, 2 | 1, 3 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

149, 216, 357, 369, 473, 621, 710, 725

Game 602

| | |
|---|---|
| 2, 2 | 2, 3 |
| 0, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 293, 362, 486, 605, 648, 720, 726

Game 604

| | |
|---|---|
| 0, 2 | 2, 3 |
| 2, 1 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 342, 364, 472, 621, 702, 725, 728

Game 608

| | |
|---|---|
| 1, 2 | 0, 3 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

158, 207, 378, 378, 467, 600, 716, 716

Game 609

| | |
|---|---|
| 2, 2 | 0, 3 |
| 1, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

347, 351, 378, 485, 607, 693, 715, 716

Game 611

| | |
|---|---|
| 1, 2 | 0, 3 |
| 2, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

288, 297, 378, 470, 618, 644, 713, 716

Game 613

| | |
|---|---|
| 3, 1 | 0, 2 |
| 2, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

347, 351, 377, 486, 607, 689, 715, 720

Game 615

| | |
|---|---|
| 0, 2 | 1, 3 |
| 2, 1 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

288, 297, 375, 470, 621, 635, 713, 725

Game 624

| | |
|---|---|
| 3, 1 | 2, 1 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

152, 203, 359, 375, 438, 635, 729, 729

Game 626

| | |
|---|---|
| 1, 1 | 2, 3 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

150, 215, 365, 365, 384, 683, 729, 729

Game 627

| | |
|---|---|
| 2, 1 | 3, 1 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

149, 206, 357, 375, 440, 635, 729, 729

Game 628

| | |
|---|---|
| 1, 1 | 3, 1 |
| 2, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

149, 215, 357, 366, 392, 683, 729, 729

Game 632

| | |
|---|---|
| 3, 1 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

152, 203, 359, 378, 438, 648, 716, 726

Game 634

| | |
|---|---|
| 0, 1 | 2, 1 |
| 3, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

150, 216, 365, 365, 384, 702, 710, 728

Game 635

| | |
|---|---|
| 2, 1 | 3, 1 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

149, 206, 357, 378, 440, 648, 716, 726

Game 636

| | |
|---|---|
| 0, 1 | 3, 1 |
| 2, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

149, 216, 357, 366, 392, 702, 710, 728

Game 638

| | |
|---|---|
| 2, 1 | 1, 3 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

162, 203, 359, 377, 405, 689, 708, 717

Game 640

| | |
|---|---|
| 2, 3 | 1, 1 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

162, 213, 365, 365, 459, 629, 708, 719

Game 642

| | |
|---|---|
| 3, 1 | 1, 1 |
| 0, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

161, 203, 359, 378, 405, 681, 716, 717

Game 644

| | |
|---|---|
| 0, 2 | 1, 1 |
| 2, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

159, 216, 365, 365, 459, 627, 710, 719

Game 645

| | |
|---|---|
| 2, 1 | 1, 2 |
| 0, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

162, 204, 377, 378, 386, 689, 708, 716

Game 646

| | |
|---|---|
| 2, 2 | 1, 1 |
| 0, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

162, 213, 368, 378, 446, 629, 708, 716

Game 647

| | |
|---|---|
| 1, 1 | 0, 3 |
| 1, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

161, 204, 378, 378, 386, 681, 716, 716

Game 648

| | |
|---|---|
| 0, 3 | 1, 1 |
| 2, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

159, 216, 368, 378, 446, 627, 710, 716

Game 650

| | |
|---|---|
| 3, 2 | 2, 1 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 294, 362, 456, 632, 638, 729, 729

Game 652

| | |
|---|---|
| 2, 1 | 2, 3 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 296, 362, 402, 636, 686, 729, 729

Game 653

| | |
|---|---|
| 2, 2 | 3, 1 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

287, 294, 375, 443, 632, 635, 729, 729

Game 654

| | |
|---|---|
| 1, 2 | 3, 1 |
| 2, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

285, 296, 375, 389, 635, 686, 729, 729

Game 656

| | |
|---|---|
| 3, 2 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 297, 362, 456, 638, 648, 713, 726

Game 658

| | |
|---|---|
| 0, 2 | 2, 1 |
| 3, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 297, 362, 402, 636, 702, 713, 728

Game 659

| | |
|---|---|
| 2, 2 | 3, 1 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

287, 297, 375, 443, 635, 648, 713, 726

Game 660

| | |
|---|---|
| 0, 2 | 3, 1 |
| 2, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

285, 297, 375, 389, 635, 702, 713, 728

Game 662

| | |
|---|---|
| 2, 1 | 2, 3 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 296, 362, 405, 648, 686, 717, 726

Game 663

| | |
|---|---|
| 2, 3 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 294, 362, 459, 632, 648, 719, 726

Game 665

| | |
|---|---|
| 3, 2 | 1, 1 |
| 0, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 297, 362, 405, 647, 699, 713, 717

Game 667

| | |
|---|---|
| 0, 2 | 1, 1 |
| 3, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

284, 297, 362, 459, 645, 645, 713, 719

Game 668

| | |
|---|---|
| 2, 1 | 2, 2 |
| 0, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

285, 296, 378, 389, 648, 686, 716, 726

Game 669

| | |
|---|---|
| 2, 2 | 2, 1 |
| 0, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

287, 294, 378, 443, 632, 648, 716, 726

Game 670

| | |
|---|---|
| 1, 1 | 0, 3 |
| 2, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

285, 297, 378, 389, 647, 699, 713, 716

Game 671

| | |
|---|---|
| 0, 3 | 1, 1 |
| 2, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

287, 297, 378, 443, 645, 645, 713, 716

Game 673

| | |
|---|---|
| 3, 1 | 2, 2 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

341, 348, 377, 445, 634, 689, 729, 729

Game 678

| | |
|---|---|
| 0, 3 | 2, 1 |
| 2, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

341, 348, 378, 445, 634, 702, 716, 728

Game 682

| | |
|---|---|
| 2, 1 | 3, 2 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 351, 364, 458, 648, 692, 715, 726

Game 684

| | |
|---|---|
| 2, 3 | 1, 1 |
| 0, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 351, 364, 459, 647, 699, 715, 719

Game 692

| | |
|---|---|
| 1, 1 | 2, 3 |
| 2, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 339, 364, 391, 701, 701, 729, 729

Game 693

| | |
|---|---|
| 2, 1 | 3, 2 |
| 1, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 348, 364, 458, 634, 692, 729, 729

Game 694

| | |
|---|---|
| 1, 1 | 3, 2 |
| 2, 2 | 0, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 339, 364, 391, 701, 701, 729, 729

Game 695

| | |
|---|---|
| 3, 1 | 2, 2 |
| 0, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

341, 351, 377, 445, 648, 689, 715, 726

Game 697

| | |
|---|---|
| 0, 1 | 2, 2 |
| 3, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 339, 364, 391, 702, 702, 728, 728

Game 698

| | |
|---|---|
| 0, 1 | 3, 2 |
| 2, 2 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 339, 364, 391, 702, 702, 728, 728

Game 700

| 2, 1 | 1, 3 |
|---|---|
| 0, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

350, 351, 377, 405, 688, 689, 715, 717

Game 701

| 3, 1 | 1, 2 |
|---|---|
| 0, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

350, 351, 377, 405, 688, 689, 715, 717

Game 702

| 0, 2 | 2, 1 |
|---|---|
| 2, 3 | 1, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

338, 348, 364, 459, 634, 702, 719, 728

Game 703

| 2, 1 | 1, 2 |
|---|---|
| 0, 3 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

350, 351, 378, 404, 688, 690, 715, 716

Game 704

| 2, 2 | 1, 1 |
|---|---|
| 0, 3 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

341, 351, 378, 445, 647, 699, 715, 716

Game 705

| 2, 1 | 0, 3 |
|---|---|
| 1, 2 | 2, 0 |

Ord% 32→0.000488281

Card% 201600→0.002016

350, 351, 378, 404, 688, 690, 715, 716

## C.15 Games of Type 15

Total Count of Type: 78

Game 556

| 3, 3 | 1, 2 |
|---|---|
| 2, 1 | 0, 0 |

Ord% 4→6.10352e-05

Card% 176400→0.001764

365, 365, 482, 482, 612, 612, 729, 729

Game 557

| 3, 2 | 1, 3 |
|---|---|
| 2, 1 | 0, 0 |

Ord% 8→0.00012207

Card% 352800→0.003528

369, 374, 473, 482, 608, 612, 729, 729

Game 560

| 3, 3 | 2, 2 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 8→0.00012207

Card% 352800→0.003528

365, 365, 455, 483, 611, 639, 729, 729

Game 562

| | |
|---|---|
| 1, 3 | 2, 2 |
| 3, 1 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

369, 377, 401, 473, 609, 689, 729, 729

Game 563

| | |
|---|---|
| 2, 3 | 3, 2 |
| 1, 1 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 374, 446, 483, 608, 639, 729, 729

Game 564

| | |
|---|---|
| 1, 3 | 3, 2 |
| 2, 1 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

374, 377, 396, 474, 608, 689, 729, 729

Game 565

| | |
|---|---|
| 3, 3 | 0, 2 |
| 2, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 482, 486, 612, 621, 720, 725

Game 566

| | |
|---|---|
| 2, 3 | 0, 2 |
| 3, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

369, 374, 473, 486, 608, 621, 720, 725

Game 570

| | |
|---|---|
| 3, 3 | 2, 2 |
| 0, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 455, 486, 611, 648, 720, 726

Game 572

| | |
|---|---|
| 0, 3 | 2, 2 |
| 3, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

369, 378, 401, 473, 609, 702, 716, 728

Game 573

| | |
|---|---|
| 2, 3 | 3, 2 |
| 0, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 374, 446, 486, 608, 648, 720, 726

Game 574

| | |
|---|---|
| 0, 3 | 3, 2 |
| 2, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

374, 378, 396, 474, 608, 702, 716, 728

Game 577

| | |
|---|---|
| 3, 3 | 0, 2 |
| 1, 1 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 405, 485, 621, 693, 717, 725

Game 579

| | |
|---|---|
| 1, 3 | 0, 2 |
| 3, 1 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

369, 375, 459, 473, 621, 635, 719, 725

Game 581

| | |
|---|---|
| 2, 1 | 0, 2 |
| 3, 3 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 405, 486, 620, 698, 717, 720

Game 583

| | |
|---|---|
| 0, 3 | 1, 2 |
| 3, 1 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

369, 378, 459, 473, 618, 644, 716, 719

Game 584

| | |
|---|---|
| 0, 3 | 3, 2 |
| 1, 1 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

374, 378, 477, 477, 608, 621, 716, 725

Game 585

| | |
|---|---|
| 1, 1 | 3, 2 |
| 0, 3 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

366, 378, 392, 485, 621, 693, 716, 725

Game 586

| | |
|---|---|
| 1, 2 | 3, 1 |
| 0, 3 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

375, 378, 450, 476, 621, 635, 716, 725

Game 587

| | |
|---|---|
| 2, 1 | 0, 3 |
| 3, 2 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

366, 378, 392, 486, 620, 698, 716, 720

Game 588

| | |
|---|---|
| 0, 3 | 1, 2 |
| 2, 1 | 3, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

378, 378, 450, 476, 618, 644, 716, 716

Game 589

| | |
|---|---|
| 2, 2 | 0, 3 |
| 3, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

369, 378, 473, 486, 617, 617, 716, 720

Game 590

| | |
|---|---|
| 0, 3 | 2, 2 |
| 1, 1 | 3, 0 |

Ord% 4→6.10352e-05

Card%

176400→0.001764

378, 378, 477, 477, 617, 617, 716, 716

Game 591

| | |
|---|---|
| 2, 2 | 1, 3 |
| 3, 1 | 0, 0 |

Ord% 4→6.10352e-05

Card%

176400→0.001764

369, 369, 473, 473, 617, 617, 729, 729

Game 594

| | |
|---|---|
| 3, 2 | 2, 3 |
| 1, 1 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 374, 446, 483, 608, 639, 729, 729

Game 596

| | |
|---|---|
| 1, 2 | 2, 3 |
| 3, 1 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

366, 369, 392, 473, 620, 698, 729, 729

Game 597

| | |
|---|---|
| 2, 2 | 3, 3 |
| 1, 1 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 455, 483, 611, 639, 729, 729

Game 598

| | |
|---|---|
| 1, 2 | 3, 3 |
| 2, 1 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 396, 474, 620, 698, 729, 729

Game 599

| | |
|---|---|
| 3, 2 | 0, 3 |
| 2, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

374, 378, 482, 486, 608, 612, 716, 720

Game 603

| | |
|---|---|
| 3, 2 | 2, 3 |
| 0, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 374, 446, 486, 608, 648, 720, 726

Game 605

| | |
|---|---|
| 0, 2 | 2, 3 |
| 3, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

366, 369, 392, 473, 621, 702, 725, 728

Game 606

| | |
|---|---|
| 2, 2 | 3, 3 |
| 0, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 455, 486, 611, 648, 720, 726

Game 607

| | |
|---|---|
| 0, 2 | 3, 3 |
| 2, 1 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 396, 474, 621, 702, 725, 728

Game 610

| | |
|---|---|
| 3, 2 | 0, 3 |
| 1, 1 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

374, 378, 405, 485, 608, 693, 716, 717

Game 612

| | |
|---|---|
| 1, 2 | 0, 3 |
| 3, 1 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

369, 378, 459, 473, 618, 644, 716, 719

Game 614

| | |
|---|---|
| 3, 1 | 0, 2 |
| 2, 3 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

374, 377, 405, 486, 608, 689, 717, 720

Game 616

| | |
|---|---|
| 0, 2 | 1, 3 |
| 3, 1 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

369, 375, 459, 473, 621, 635, 719, 725

Game 617

| | |
|---|---|
| 0, 2 | 3, 3 |
| 1, 1 | 2, 0 |

Ord% 4→6.10352e-05

Card%

176400→0.001764

365, 365, 477, 477, 621, 621, 725, 725

Game 618

| | |
|---|---|
| 2, 2 | 0, 3 |
| 1, 1 | 3, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

378, 378, 401, 485, 609, 693, 716, 716

Game 619

| | |
|---|---|
| 1, 2 | 0, 3 |
| 2, 1 | 3, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

378, 378, 450, 476, 618, 644, 716, 716

Game 620

| | |
|---|---|
| 3, 1 | 0, 3 |
| 2, 2 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

377, 378, 401, 486, 609, 689, 716, 720

Game 621

| | |
|---|---|
| 0, 3 | 3, 1 |
| 1, 2 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

375, 378, 450, 476, 621, 635, 716, 725

Game 672

| | |
|---|---|
| 3, 3 | 2, 1 |
| 1, 2 | 0, 0 |

Ord% 4→6.10352e-05

Card%

176400→0.001764

365, 365, 456, 456, 638, 638, 729, 729

Game 674

| | |
|---|---|
| 3, 1 | 2, 3 |
| 1, 2 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 377, 402, 446, 636, 689, 729, 729

Game 675

| | |
|---|---|
| 2, 3 | 3, 1 |
| 1, 2 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 375, 446, 456, 635, 638, 729, 729

Game 676

| | |
|---|---|
| 1, 3 | 3, 1 |
| 2, 2 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

375, 377, 395, 447, 635, 689, 729, 729

Game 677

| | |
|---|---|
| 3, 3 | 2, 1 |
| 0, 2 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 456, 459, 638, 648, 719, 726

Game 679

| | |
|---|---|
| 0, 3 | 2, 1 |
| 3, 2 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 378, 402, 446, 636, 702, 716, 728

Game 680

| | |
|---|---|
| 2, 3 | 3, 1 |
| 0, 2 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 375, 446, 459, 635, 648, 719, 726

Game 681

| | |
|---|---|
| 0, 3 | 3, 1 |
| 2, 2 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

375, 378, 395, 447, 635, 702, 716, 728

Game 683

| | |
|---|---|
| 2, 1 | 3, 3 |
| 0, 2 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 405, 458, 648, 692, 717, 726

Game 685

| | |
|---|---|
| 3, 3 | 1, 1 |
| 0, 2 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 405, 459, 647, 699, 717, 719

Game 686

| | |
|---|---|
| 0, 3 | 1, 1 |
| 3, 2 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 378, 446, 459, 645, 645, 716, 719

Game 687

| | |
|---|---|
| 2, 1 | 3, 2 |
| 0, 3 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

366, 378, 392, 458, 648, 692, 716, 726

Game 688

| | |
|---|---|
| 2, 2 | 3, 1 |
| 0, 3 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

375, 378, 449, 449, 635, 648, 716, 726

Game 689

| | |
|---|---|
| 1, 1 | 0, 3 |
| 3, 2 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

366, 378, 392, 459, 647, 699, 716, 719

Game 690

| | |
|---|---|
| 0, 3 | 1, 1 |
| 2, 2 | 3, 0 |

Ord% 4→6.10352e-05

Card%

176400→0.001764

378, 378, 449, 449, 645, 645, 716, 716

Game 706

| | |
|---|---|
| 1, 1 | 2, 3 |
| 3, 2 | 0, 0 |

Ord% 4→6.10352e-05

Card%

176400→0.001764

366, 366, 392, 392, 701, 701, 729, 729

Game 707

| | |
|---|---|
| 2, 1 | 3, 3 |
| 1, 2 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 402, 458, 636, 692, 729, 729

Game 708

| | |
|---|---|
| 1, 1 | 3, 3 |
| 2, 2 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 393, 393, 701, 701, 729, 729

Game 709

| | |
|---|---|
| 3, 1 | 2, 3 |
| 0, 2 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 377, 405, 446, 648, 689, 717, 726

Game 710

| | |
|---|---|
| 0, 1 | 2, 3 |
| 3, 2 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

366, 366, 392, 392, 702, 702, 728, 728

Game 711

| | |
|---|---|
| 0, 1 | 3, 3 |
| 2, 2 | 1, 0 |

Ord% 4→6.10352e-05

Card%

176400→0.001764

365, 365, 393, 393, 702, 702, 728, 728

Game 712

| | |
|---|---|
| 3, 1 | 0, 3 |
| 1, 2 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

377, 378, 404, 405, 689, 690, 716, 717

Game 713

| | |
|---|---|
| 3, 1 | 1, 3 |
| 0, 2 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

377, 377, 405, 405, 689, 689, 717, 717

Game 714

| | |
|---|---|
| 0, 2 | 3, 1 |
| 2, 3 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

366, 375, 392, 459, 635, 702, 719, 728

Game 715

| | |
|---|---|
| 2, 1 | 0, 3 |
| 1, 2 | 3, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

378, 378, 404, 404, 690, 690, 716, 716

Game 716

| | |
|---|---|
| 1, 1 | 0, 3 |
| 2, 2 | 3, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

378, 378, 395, 447, 647, 699, 716, 716

Game 717

| | |
|---|---|
| 2, 2 | 3, 1 |
| 1, 3 | 0, 0 |

Ord% 4→6.10352e-05

Card%

176400→0.001764

375, 375, 449, 449, 635, 635, 729, 729

Game 718

| | |
|---|---|
| 2, 1 | 3, 2 |
| 1, 3 | 0, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

366, 375, 392, 458, 635, 692, 729, 729

Game 719

| | |
|---|---|
| 3, 2 | 2, 1 |
| 0, 3 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 378, 446, 456, 638, 648, 716, 726

Game 720

| | |
|---|---|
| 0, 2 | 2, 1 |
| 3, 3 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

365, 365, 402, 459, 636, 702, 719, 728

Game 721

| | |
|---|---|
| 3, 2 | 1, 1 |
| 0, 3 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

368, 378, 405, 446, 647, 699, 716, 717

Game 722

| | |
|---|---|
| 0, 2 | 1, 1 |
| 3, 3 | 2, 0 |

Ord% 4→6.10352e-05

Card%

176400→0.001764

365, 365, 459, 459, 645, 645, 719, 719

Game 723

| | |
|---|---|
| 3, 1 | 2, 2 |
| 0, 3 | 1, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

377, 378, 395, 447, 648, 689, 716, 726

Game 724

| | |
|---|---|
| 1, 1 | 3, 2 |
| 2, 3 | 0, 0 |

Ord% 4→6.10352e-05

Card%

176400→0.001764

366, 366, 392, 392, 701, 701, 729, 729

Game 725

| | |
|---|---|
| 0, 1 | 2, 2 |
| 3, 3 | 1, 0 |

Ord% 4→6.10352e-05

Card%

176400→0.001764

365, 365, 393, 393, 702, 702, 728, 728

Game 726

| | |
|---|---|
| 3, 1 | 1, 2 |
| 0, 3 | 2, 0 |

Ord% 8→0.00012207

Card%

352800→0.003528

377, 378, 404, 405, 689, 690, 716, 717

# Appendix D

# Taxonomy: 729 2×2 Status Quo Game Positions

What follows is a complete listing of all the possible status quo positions that may be found within all 726 possible 2×2 games. Status quo positions are represented as normal form games with the top-left cell acting as the status quo position (the payoffs in this cell are in boldface type) and the row player making the status quo position assessment (the status quo payoff to this player is underlined). Status quo payoffs are always listed as zero. The other payoffs are either listed as 1, indicating that there is a payoff available on that outcome that is better than the status quo payoff, or as -1, indicating that there is a payoff available on that outcome that is worse than the status quo payoff.

Accompanying each status quo representation are three additional pieces of information:

1. the probability of this game being randomly generated when each payoff is randomly assigned one of four possible values (listed as "Ord%" for "ordinal probability"). The value to the left of the arrow is the number of ways the game can be generated with four values. The value to the right of the arrow is the percentage chance of producing this as calculated by taking the value on the left and dividing it by the total number of possible games that can be created this way ($4^8 = 65536$).

2. the probability of this game being randomly generated when each payoff is randomly assigned one of ten possible values (listed as "Card%" for "cardinal probability"). The value to the left of the arrow is the number of ways the game can be generated with 10 values. The value to the right

of the arrow is the percentage chance of producing this as calculated by taking the value on the left and dividing it by the total number of possible games that can be created this way ($10^8 = 100000000$).

3. the list of 2×2 games that this SQ position may be found within (These values correspond to the 726 2×2 games that are taxonomized in Appendix C).

SQ Position 1

| **0**, **0** | 0, 0 |
|---|---|
| 0, 0 | 0, 0 |

Ord% 128→0.000244141
Card% 800→1e-06
8

1, 1, 1, 1, 1, 1, 1, 1

SQ Position 2

| **0**, **0** | 0, 0 |
|---|---|
| 0, 0 | 0, -1 |

Ord% 192→0.000366211
Card% 3600→4.5e-06
1

8

SQ Position 3

| **0**, **0** | 0, 0 |
|---|---|
| 0, 0 | 0, 1 |

Ord% 192→0.000366211
Card% 3600→4.5e-06
1

2

SQ Position 4

| **0**, **0** | 0, 0 |
|---|---|
| 0, -1 | 0, 0 |

Ord% 192→0.000366211
Card% 3600→4.5e-06
1

8

SQ Position 5

| **0**, **0** | 0, 0 |
|---|---|
| 0, -1 | 0, -1 |

Ord% 448→0.000854492
Card% 22800→2.85e-05
4

3, 3, 12, 12

SQ Position 6

| **0**, **0** | 0, 0 |
|---|---|
| 0, -1 | 0, 1 |

Ord% 128→0.000244141
Card% 9600→1.2e-05
1

15

SQ Position 7

| **0**, **0** | 0, 0 |
|---|---|
| 0, 1 | 0, 0 |

Ord% 192→0.000366211
Card% 3600→4.5e-06
1

2

SQ Position 8

| **0**, **0** | 0, 0 |
|---|---|
| 0, 1 | 0, -1 |

Ord% 128→0.000244141
Card% 9600→1.2e-05
1

15

SQ Position 9

| **0**, **0** | 0, 0 |
|---|---|
| 0, 1 | 0, 1 |

Ord% 448→0.000854492
Card% 22800→2.85e-05
4

3, 3, 4, 4

SQ Position 10

| **0**, **0** | 0, -1 |
|---|---|
| 0, 0 | 0, 0 |

Ord% 192→0.000366211
Card% 3600→4.5e-06
1

8

SQ Position 11

| **0**, **0** | 0, -1 |
|---|---|
| 0, 0 | 0, -1 |

Ord% 448→0.000854492
Card% 22800→2.85e-05
4

5, 5, 16, 16

SQ Position 12

| **0**, **0** | 0, -1 |
|---|---|
| 0, 0 | 0, 1 |

Ord% 128→0.000244141
Card% 9600→1.2e-05
1

11

SQ Position 13

| **0**, **0** | 0, -1 |
|---|---|
| 0, -1 | 0, 0 |

Ord% 448→0.000854492
Card% 22800→2.85e-05
4

7, 7, 18, 18

SQ Position 14

| **0**, **0** | 0, -1 |
|---|---|
| 0, -1 | 0, -1 |

Ord% 1152→0.00219727
Card% 162000→0.0002025
13

2, 4, 6, 9, 10, 11, 13, 14, 15, 17, 19, 20, 21

SQ Position 15

| **0**, **0** | 0, -1 |
|---|---|
| 0, -1 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
3

10, 19, 21

SQ Position 16

| **0**, **0** | 0, -1 |
|---|---|
| 0, 1 | 0, 0 |

Ord% 128→0.000244141
Card% 9600→1.2e-05
1

9

SQ Position 17

| **0**, **0** | 0, -1 |
|---|---|
| 0, 1 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
3

6, 17, 20

SQ Position 18

| **0**, **0** | 0, -1 |
|---|---|
| 0, 1 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
3

12, 13, 14

SQ Position 19

| **0**, **0** | 0, 1 |
|---|---|
| 0, 0 | 0, 0 |

Ord% 192→0.000366211
Card% 3600→4.5e-06
1

2

SQ Position 20

| **0**, **0** | 0, 1 |
|---|---|
| 0, 0 | 0, -1 |

Ord% 128→0.000244141
Card% 9600→1.2e-05
1

11

SQ Position 21

| **0**, **0** | 0, 1 |
|---|---|
| 0, 0 | 0, 1 |

Ord% 448→0.000854492
Card% 22800→2.85e-05
4

5, 5, 6, 6

SQ Position 22

| **0**, **0** | 0, 1 |
|---|---|
| 0, -1 | 0, 0 |

Ord% 128→0.000244141
Card% 9600→1.2e-05
1

9

SQ Position 23

| **0**, **0** | 0, 1 |
|---|---|
| 0, -1 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
3

4, 13, 14

SQ Position 24

| **0**, **0** | 0, 1 |
|---|---|
| 0, -1 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
3

16, 17, 20

SQ Position 25

| **0**, **0** | 0, 1 |
|---|---|
| 0, 1 | 0, 0 |

Ord% 448→0.000854492
Card% 22800→2.85e-05
4

7, 7, 10, 10

SQ Position 26

| **0**, **0** | 0, 1 |
|---|---|
| 0, 1 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
3

18, 19, 21

SQ Position 27

| **0**, **0** | 0, 1 |
|---|---|
| 0, 1 | 0, 1 |

Ord% 1152→0.00219727
Card% 162000→0.0002025
13

8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21

SQ Position 28

| **0**, **0** | 0, 0 |
|---|---|
| 0, 0 | -1, 0 |

Ord% 192→0.000366211
Card% 3600→4.5e-06
1

8

SQ Position 29

| **0**, **0** | 0, 0 |
|---|---|
| 0, 0 | -1, -1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
2

321, 321

SQ Position 30

| **0**, **0** | 0, 0 |
|---|---|
| 0, 0 | -1, 1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

60

SQ Position 31

| **0**, **0** | 0, 0 |
|---|---|
| 0, -1 | -1, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

337

SQ Position 32

| **0**, **0** | 0, 0 |
|---|---|
| 0, -1 | -1, -1 |

Ord% 672→0.00128174
Card% 102600→0.00012825
3

101, 325, 340

SQ Position 33

| **0**, **0** | 0, 0 |
|---|---|
| 0, -1 | -1, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

355

SQ Position 34

| **0**, **0** | 0, 0 |
|---|---|
| 0, 1 | -1, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

50

SQ Position 35

| **0**, **0** | 0, 0 |
|---|---|
| 0, 1 | -1, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

329

SQ Position 36

| **0**, **0** | 0, 0 |
|---|---|
| 0, 1 | -1, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

117, 155, 164

SQ Position 37

| **0**, **0** | 0, -1 |
|---|---|
| 0, 0 | -1, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

337

SQ Position 38

| **0**, **0** | 0, -1 |
|---|---|
| 0, 0 | -1, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

201, 330, 347

SQ Position 39

| **0**, **0** | 0, -1 |
|---|---|
| 0, 0 | -1, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

360

SQ Position 40

| **0**, **0** | 0, -1 |
|---|---|
| 0, -1 | -1, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

308, 356, 361

SQ Position 41

| **0**, **0** | 0, -1 |
|---|---|
| 0, -1 | -1, -1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

30, 136, 232, 322, 326, 331, 336, 338, 341, 343, 348, 357, 362

SQ Position 42

| **0**, **0** | 0, -1 |
|---|---|
| 0, -1 | -1, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

354, 370, 372

SQ Position 43

| **0**, **0** | 0, -1 |
|---|---|
| 0, 1 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

346

SQ Position 44

| **0**, **0** | 0, -1 |
|---|---|
| 0, 1 | -1, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

250, 334, 352

SQ Position 45

| **0**, **0** | 0, -1 |
|---|---|
| 0, 1 | -1, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

366, 368, 374

SQ Position 46

| **0**, **0** | 0, 1 |
|---|---|
| 0, 0 | -1, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

58

SQ Position 47

| **0**, **0** | 0, 1 |
|---|---|
| 0, 0 | -1, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

324

SQ Position 48

| **0**, **0** | 0, 1 |
|---|---|
| 0, 0 | -1, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

217, 256, 258

SQ Position 49

| **0**, **0** | 0, 1 |
|---|---|
| 0, -1 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

339

SQ Position 50

| **0**, **0** | 0, 1 |
|---|---|
| 0, -1 | -1, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

162, 327, 342

SQ Position 51

| **0**, **0** | 0, 1 |
|---|---|
| 0, -1 | -1, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

358, 359, 371

SQ Position 52

| **0**, **0** | 0, 1 |
|---|---|
| 0, 1 | -1, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

292, 323, 328

SQ Position 53

| **0**, **0** | 0, 1 |
|---|---|
| 0, 1 | -1, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

332, 333, 335

SQ Position 54

| **0**, **0** | 0, 1 |
|---|---|
| 0, 1 | -1, 1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
14

344, 344, 345, 349, 350,
351, 353, 363, 364, 365,
367, 369, 373, 375

SQ Position 55

| **0**, **0** | 0, 0 |
|---|---|
| 0, 0 | 1, 0 |

Ord% 192→0.000366211
Card% 3600→4.5e-06
1

2

SQ Position 56

| **0**, **0** | 0, 0 |
|---|---|
| 0, 0 | 1, -1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

60

SQ Position 57

| **0**, **0** | 0, 0 |
|---|---|
| 0, 0 | 1, 1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
2

22, 22

SQ Position 58

| **0**, **0** | 0, 0 |
|---|---|
| 0, -1 | 1, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

58

SQ Position 59

| **0**, **0** | 0, 0 |
|---|---|
| 0, -1 | 1, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

57, 85, 86

SQ Position 60

| **0**, **0** | 0, 0 |
|---|---|
| 0, -1 | 1, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

59

SQ Position 61

| **0**, **0** | 0, 0 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

23

SQ Position 62

| **0**, **0** | 0, 0 |
|---|---|
| 0, 1 | 1, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

61

SQ Position 63

| **0**, **0** | 0, 0 |
|---|---|
| 0, 1 | 1, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

24, 25, 26

SQ Position 64

| **0**, **0** | 0, -1 |
|---|---|
| 0, 0 | 1, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

50

SQ Position 65

| **0**, **0** | 0, -1 |
|---|---|
| 0, 0 | 1, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

49, 76, 83

SQ Position 66

| **0**, **0** | 0, -1 |
|---|---|
| 0, 0 | 1, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

51

SQ Position 67

| **0**, **0** | 0, -1 |
|---|---|
| 0, -1 | 1, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

47, 74, 80

SQ Position 68

| **0**, **0** | 0, -1 |
|---|---|
| 0, -1 | 1, -1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
14

46, 46, 71, 72, 73, 78, 79,
82, 89, 90, 91, 92, 93, 94

SQ Position 69

| **0**, **0** | 0, -1 |
|---|---|
| 0, -1 | 1, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

48, 75, 81

SQ Position 70

| **0**, **0** | 0, -1 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

53

SQ Position 71

| **0**, **0** | 0, -1 |
|---|---|
| 0, 1 | 1, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

52, 77, 84

SQ Position 72

| **0**, **0** | 0, -1 |
|---|---|
| 0, 1 | 1, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

54, 55, 56

SQ Position 73

| **0**, **0** | 0, 1 |
|---|---|
| 0, 0 | 1, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

23

SQ Position 74

| **0**, **0** | 0, 1 |
|---|---|
| 0, 0 | 1, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

66

SQ Position 75

| **0**, **0** | 0, 1 |
|---|---|
| 0, 0 | 1, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

27, 28, 37

SQ Position 76

| **0**, **0** | 0, 1 |
|---|---|
| 0, -1 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

63

SQ Position 77

| **0**, **0** | 0, 1 |
|---|---|
| 0, -1 | 1, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

62, 87, 88

SQ Position 78

| **0**, **0** | 0, 1 |
|---|---|
| 0, -1 | 1, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

64, 65, 69

SQ Position 79

| **0**, **0** | 0, 1 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

29, 32, 38

SQ Position 80

| **0**, **0** | 0, 1 |
|---|---|
| 0, 1 | 1, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

67, 68, 70

SQ Position 81

| **0**, **0** | 0, 1 |
|---|---|
| 0, 1 | 1, 1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

30, 31, 33, 34, 35, 36, 39, 40, 41, 42, 43, 44, 45

SQ Position 82

| **0**, **0** | -1, 0 |
|---|---|
| 0, 0 | 0, 0 |

Ord% 192→0.000366211
Card% 3600→4.5e-06
1

8

SQ Position 83

| **0**, **0** | -1, 0 |
|---|---|
| 0, 0 | 0, -1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

337

SQ Position 84

| **0**, **0** | -1, 0 |
|---|---|
| 0, 0 | 0, 1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

58

SQ Position 85

| **0**, **0** | -1, 0 |
|---|---|
| 0, -1 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
2

344, 344

SQ Position 86

| **0**, **0** | -1, 0 |
|---|---|
| 0, -1 | 0, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

117, 366, 367

SQ Position 87

| **0**, **0** | -1, 0 |
|---|---|
| 0, -1 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

345

SQ Position 88

| **0**, **0** | -1, 0 |
|---|---|
| 0, 1 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

30

SQ Position 89

| **0**, **0** | -1, 0 |
|---|---|
| 0, 1 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

343

SQ Position 90

| **0**, **0** | -1, 0 |
|---|---|
| 0, 1 | 0, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

101, 136, 162

SQ Position 91

| **0**, **0** | -1, -1 |
|---|---|
| 0, 0 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
2

321, 321

SQ Position 92

| **0**, **0** | -1, -1 |
|---|---|
| 0, 0 | 0, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

201, 330, 347

SQ Position 93

| **0**, **0** | -1, -1 |
|---|---|
| 0, 0 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

324

SQ Position 94

| **0**, **0** | -1, -1 |
|---|---|
| 0, -1 | 0, 0 |

Ord% 672→0.00128174
Card% 102600→0.00012825
3

292, 332, 350

SQ Position 95

| **0**, **0** | -1, -1 |
|---|---|
| 0, -1 | 0, -1 |

Ord% 1728→0.0032959
Card% 729000→0.00091125
13

50, 155, 250, 328, 329, 334, 335, 346, 349, 352, 353, 368, 369

SQ Position 96

| **0**, **0** | -1, -1 |
|---|---|
| 0, -1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

323, 333, 351

SQ Position 97

| **0**, **0** | -1, -1 |
|---|---|
| 0, 1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

322

SQ Position 98

| **0**, **0** | -1, -1 |
|---|---|
| 0, 1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

232, 331, 348

SQ Position 99

| **0**, **0** | -1, -1 |
|---|---|
| 0, 1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

325, 326, 327

SQ Position 100

| **0**, **0** | -1, 1 |
|---|---|
| 0, 0 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

60

SQ Position 101

| **0**, **0** | -1, 1 |
|---|---|
| 0, 0 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

360

SQ Position 102

| **0**, **0** | -1, 1 |
|---|---|
| 0, 0 | 0, 1 |

Ord% 672→0.00128174
Card% 102600→0.00012825
3

217, 256, 258

SQ Position 103

| **0**, **0** | -1, 1 |
|---|---|
| 0, -1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

363

SQ Position 104

| **0**, **0** | -1, 1 |
|---|---|
| 0, -1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

164, 374, 375

SQ Position 105

| **0**, **0** | -1, 1 |
|---|---|
| 0, -1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

364, 365, 373

SQ Position 106

| **0**, **0** | -1, 1 |
|---|---|
| 0, 1 | 0, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

308, 336, 354

SQ Position 107

| **0**, **0** | -1, 1 |
|---|---|
| 0, 1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

361, 362, 372

SQ Position 108

| **0**, **0** | -1, 1 |
|---|---|
| 0, 1 | 0, 1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

337, 338, 339, 340, 341, 342, 355, 356, 357, 358, 359, 370, 371

SQ Position 109

| **0**, **0** | -1, 0 |
|---|---|
| 0, 0 | -1, 0 |

Ord% 448→0.000854492
Card% 22800→2.85e-05
4

3, 3, 12, 12

SQ Position 110

| **0**, **0** | -1, 0 |
|---|---|
| 0, 0 | -1, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

101, 325, 340

SQ Position 111

| **0**, **0** | -1, 0 |
|---|---|
| 0, 0 | -1, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

57, 85, 86

SQ Position 112

| **0**, **0** | -1, 0 |
|---|---|
| 0, -1 | -1, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

117, 366, 367

SQ Position 113

| **0**, **0** | -1, 0 |
|---|---|
| 0, -1 | -1, -1 |

Ord% 1568→0.00299072
Card%
649800→0.00081225
12

95, 95, 105, 105, 121, 121, 517, 517, 527, 527, 547, 547

SQ Position 114

| **0**, **0** | -1, 0 |
|---|---|
| 0, -1 | -1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

124, 531, 538

SQ Position 115

| **0**, **0** | -1, 0 |
|---|---|
| 0, 1 | -1, 0 |

Ord% 672→0.00128174
Card% 102600→0.00012825
3

24, 34, 54

SQ Position 116

| **0**, **0** | -1, 0 |
|---|---|
| 0, 1 | -1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

109, 520, 530

SQ Position 117

| **0**, **0** | -1, 0 |
|---|---|
| 0, 1 | -1, 1 |

Ord% 1568→0.00299072
Card% 649800→0.00081225
10

95, 95, 96, 97, 105, 121, 140, 159, 188, 189

SQ Position 118

| **0**, **0** | -1, -1 |
|---|---|
| 0, 0 | -1, 0 |

Ord% 672→0.00128174
Card% 102600→0.00012825
3

101, 325, 340

SQ Position 119

| **0**, **0** | -1, -1 |
|---|---|
| 0, 0 | -1, -1 |

Ord% 1568→0.00299072
Card% 649800→0.00081225
10

98, 98, 110, 110, 205, 205, 521, 521, 532, 532

SQ Position 120

| **0**, **0** | -1, -1 |
|---|---|
| 0, 0 | -1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

104, 475, 486

SQ Position 121

| **0**, **0** | -1, -1 |
|---|---|
| 0, -1 | -1, 0 |

Ord% 1568→0.00299072
Card% 649800→0.00081225
9

100, 112, 125, 296, 311, 523, 534, 539, 543

SQ Position 122

| **0**, **0** | -1, -1 |
|---|---|
| 0, -1 | -1, -1 |

Ord% 4032→0.00769043
Card% 4.617e+06→0.00577125
39

24, 34, 54, 96, 99, 102, 106, 108, 109, 111, 114, 115, 118, 122, 127, 140, 159, 236, 253, 379, 393, 464, 465, 507, 508, 518, 520, 522, 525, 526, 528, 530, 533, 536, 537, 540, 544, 548, 549

SQ Position 123

| **0**, **0** | -1, -1 |
|---|---|
| 0, -1 | -1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539
9

103, 113, 126, 429, 442, 524, 535, 550, 552

SQ Position 124

| | |
|---|---|
| **0, 0** | -1, -1 |
| 0, 1 | -1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

102, 379, 393

SQ Position 125

| | |
|---|---|
| **0, 0** | -1, -1 |
| 0, 1 | -1, -1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

99, 111, 114, 236, 253, 522, 525, 533, 536

SQ Position 126

| | |
|---|---|
| **0, 0** | -1, -1 |
| 0, 1 | -1, 1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
10

105, 106, 107, 517, 517, 518, 519, 527, 548, 554

SQ Position 127

| | |
|---|---|
| **0, 0** | -1, 1 |
| 0, 0 | -1, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

57, 85, 86

SQ Position 128

| | |
|---|---|
| **0, 0** | -1, 1 |
| 0, 0 | -1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

104, 475, 486

SQ Position 129

| | |
|---|---|
| **0, 0** | -1, 1 |
| 0, 0 | -1, 1 |

Ord% 1568→0.00299072
Card%
649800→0.00081225
10

116, 116, 119, 119, 227, 227, 281, 281, 282, 282

SQ Position 130

| | |
|---|---|
| **0, 0** | -1, 1 |
| 0, -1 | -1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

120, 417, 418

SQ Position 131

| | |
|---|---|
| **0, 0** | -1, 1 |
| 0, -1 | -1, -1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

97, 107, 123, 188, 189, 519, 529, 554, 555

SQ Position 132

| | |
|---|---|
| **0, 0** | -1, 1 |
| 0, -1 | -1, 1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

128, 129, 130, 541, 542, 545, 546, 551, 553

SQ Position 133

| **0**, **0** | -1, 1 |
|---|---|
| 0, 1 | -1, 0 |

Ord% 1568→0.00299072
Card% 649800→0.00081225
9

100, 103, 108, 296, 311, 429, 442, 464, 465

SQ Position 134

| **0**, **0** | -1, 1 |
|---|---|
| 0, 1 | -1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539
9

112, 113, 115, 523, 524, 526, 543, 544, 552

SQ Position 135

| **0**, **0** | -1, 1 |
|---|---|
| 0, 1 | -1, 1 |

Ord% 4032→0.00769043
Card% 4.617e+06→0.00577125
40

117, 118, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 366, 367, 417, 418, 507, 508, 527, 528, 529, 531, 534, 535, 537, 538, 539, 540, 541, 542, 545, 546, 547, 547, 549, 550, 551, 553, 555

SQ Position 136

| **0**, **0** | -1, 0 |
|---|---|
| 0, 0 | 1, 0 |

Ord% 128→0.000244141
Card% 9600→1.2e-05
1

15

SQ Position 137

| **0**, **0** | -1, 0 |
|---|---|
| 0, 0 | 1, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

355

SQ Position 138

| **0**, **0** | -1, 0 |
|---|---|
| 0, 0 | 1, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

59

SQ Position 139

| **0**, **0** | -1, 0 |
|---|---|
| 0, -1 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

345

SQ Position 140

| **0**, **0** | -1, 0 |
|---|---|
| 0, -1 | 1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

124, 531, 538

SQ Position 141

| **0**, **0** | -1, 0 |
|---|---|
| 0, -1 | 1, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
2

630, 630

SQ Position 142

| **0**, **0** | -1, 0 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

39

SQ Position 143

| **0**, **0** | -1, 0 |
|---|---|
| 0, 1 | 1, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

629

SQ Position 144

| **0**, **0** | -1, 0 |
|---|---|
| 0, 1 | 1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

109, 145, 163

SQ Position 145

| **0**, **0** | -1, -1 |
|---|---|
| 0, 0 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

329

SQ Position 146

| **0**, **0** | -1, -1 |
|---|---|
| 0, 0 | 1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

210, 623, 631

SQ Position 147

| **0**, **0** | -1, -1 |
|---|---|
| 0, 0 | 1, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

478

SQ Position 148

| **0**, **0** | -1, -1 |
|---|---|
| 0, -1 | 1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

299, 625, 633

SQ Position 149

| **0**, **0** | -1, -1 |
|---|---|
| 0, -1 | 1, -1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
14

73, 176, 270, 406, 446, 490, 575, 601, 622, 622, 627, 628, 635, 636

SQ Position 150

| **0**, **0** | -1, -1 |
|---|---|
| 0, -1 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

433, 626, 634

SQ Position 151

| **0**, **0** | -1, -1 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

383

SQ Position 152

| **0**, **0** | -1, -1 |
|---|---|
| 0, 1 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

241, 624, 632

SQ Position 153

| **0**, **0** | -1, -1 |
|---|---|
| 0, 1 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

520, 558, 592

SQ Position 154

| **0**, **0** | -1, 1 |
|---|---|
| 0, 0 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

61

SQ Position 155

| **0**, **0** | -1, 1 |
|---|---|
| 0, 0 | 1, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

498

SQ Position 156

| **0**, **0** | -1, 1 |
|---|---|
| 0, 0 | 1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

218, 257, 259

SQ Position 157

| **0**, **0** | -1, 1 |
|---|---|
| 0, -1 | 1, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

397

SQ Position 158

| **0**, **0** | -1, 1 |
|---|---|
| 0, -1 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

165, 568, 608

SQ Position 159

| **0**, **0** | -1, 1 |
|---|---|
| 0, -1 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

643, 644, 648

SQ Position 160

| **0**, **0** | -1, 1 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

314, 445, 454

SQ Position 161

| **0**, **0** | -1, 1 |
|---|---|
| 0, 1 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

641, 642, 647

SQ Position 162

| **0**, **0** | -1, 1 |
|---|---|
| 0, 1 | 1, 1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

343, 396, 489, 530, 567, 600, 629, 637, 638, 639, 640, 645, 646

SQ Position 163

| **0**, **0** | 1, 0 |
|---|---|
| 0, 0 | 0, 0 |

Ord% 192→0.000366211
Card% 3600→4.5e-06
1

2

SQ Position 164

| **0**, **0** | 1, 0 |
|---|---|
| 0, 0 | 0, -1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

50

SQ Position 165

| **0**, **0** | 1, 0 |
|---|---|
| 0, 0 | 0, 1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

23

SQ Position 166

| **0**, **0** | 1, 0 |
|---|---|
| 0, -1 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

30

SQ Position 167

| **0**, **0** | 1, 0 |
|---|---|
| 0, -1 | 0, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

24, 34, 54

SQ Position 168

| **0**, **0** | 1, 0 |
|---|---|
| 0, -1 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

39

SQ Position 169

| **0**, **0** | 1, 0 |
|---|---|
| 0, 1 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
2

46, 46

SQ Position 170

| **0**, **0** | 1, 0 |
|---|---|
| 0, 1 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

73

SQ Position 171

| **0**, **0** | 1, 0 |
|---|---|
| 0, 1 | 0, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

57, 62, 78

SQ Position 172

| **0**, **0** | 1, -1 |
|---|---|
| 0, 0 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

60

SQ Position 173

| **0**, **0** | 1, -1 |
|---|---|
| 0, 0 | 0, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

49, 76, 83

SQ Position 174

| **0**, **0** | 1, -1 |
|---|---|
| 0, 0 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

66

SQ Position 175

| **0**, **0** | 1, -1 |
|---|---|
| 0, -1 | 0, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

29, 42, 67

SQ Position 176

| **0**, **0** | 1, -1 |
|---|---|
| 0, -1 | 0, -1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

23, 26, 32, 33, 36, 43, 52, 53, 56, 61, 68, 77, 84

SQ Position 177

| **0**, **0** | 1, -1 |
|---|---|
| 0, -1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

38, 45, 70

SQ Position 178

| **0**, **0** | 1, -1 |
|---|---|
| 0, 1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

82

SQ Position 179

| **0**, **0** | 1, -1 |
|---|---|
| 0, 1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

72, 90, 92

SQ Position 180

| **0**, **0** | 1, -1 |
|---|---|
| 0, 1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

86, 88, 94

SQ Position 181

| **0**, **0** | 1, 1 |
|---|---|
| 0, 0 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
2

22, 22

SQ Position 182

| **0**, **0** | 1, 1 |
|---|---|
| 0, 0 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

51

SQ Position 183

| **0**, **0** | 1, 1 |
|---|---|
| 0, 0 | 0, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

27, 28, 37

SQ Position 184

| **0**, **0** | 1, 1 |
|---|---|
| 0, -1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

31

SQ Position 185

| **0**, **0** | 1, 1 |
|---|---|
| 0, -1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

25, 35, 55

SQ Position 186

| **0**, **0** | 1, 1 |
|---|---|
| 0, -1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

40, 41, 44

SQ Position 187

| **0**, **0** | 1, 1 |
|---|---|
| 0, 1 | 0, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

47, 48, 71

SQ Position 188

| **0**, **0** | 1, 1 |
|---|---|
| 0, 1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

74, 75, 89

SQ Position 189

| **0**, **0** | 1, 1 |
|---|---|
| 0, 1 | 0, 1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

58, 59, 63, 64, 65, 69, 79, 80, 81, 85, 87, 91, 93

SQ Position 190

| **0**, **0** | 1, 0 |
|---|---|
| 0, 0 | -1, 0 |

Ord% 128→0.000244141
Card% 9600→1.2e-05
1

15

SQ Position 191

| **0**, **0** | 1, 0 |
|---|---|
| 0, 0 | -1, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

329

SQ Position 192

| **0**, **0** | 1, 0 |
|---|---|
| 0, 0 | -1, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

61

SQ Position 193

| **0**, **0** | 1, 0 |
|---|---|
| 0, -1 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

343

SQ Position 194

| **0**, **0** | 1, 0 |
|---|---|
| 0, -1 | -1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

109, 520, 530

SQ Position 195

| **0**, **0** | 1, 0 |
|---|---|
| 0, -1 | -1, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

629

SQ Position 196

| **0**, **0** | 1, 0 |
|---|---|
| 0, 1 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

73

SQ Position 197

| **0**, **0** | 1, 0 |
|---|---|
| 0, 1 | -1, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
2

622, 622

SQ Position 198

| **0**, **0** | 1, 0 |
|---|---|
| 0, 1 | -1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

124, 165, 176

SQ Position 199

| **0**, **0** | 1, -1 |
|---|---|
| 0, 0 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

355

SQ Position 200

| **0**, **0** | 1, -1 |
|---|---|
| 0, 0 | -1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

210, 623, 631

SQ Position 201

| **0**, **0** | 1, -1 |
|---|---|
| 0, 0 | -1, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

498

SQ Position 202

| **0**, **0** | 1, -1 |
|---|---|
| 0, -1 | -1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

314, 637, 641

SQ Position 203

| **0**, **0** | 1, -1 |
|---|---|
| 0, -1 | -1, -1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

39, 145, 241, 383, 445, 489, 558, 600, 624, 629, 632, 638, 642

SQ Position 204

| **0**, **0** | 1, -1 |
|---|---|
| 0, -1 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

454, 645, 647

SQ Position 205

| **0**, **0** | 1, -1 |
|---|---|
| 0, 1 | -1, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

406

SQ Position 206

| **0**, **0** | 1, -1 |
|---|---|
| 0, 1 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

270, 627, 635

SQ Position 207

| **0**, **0** | 1, -1 |
|---|---|
| 0, 1 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

538, 575, 608

SQ Position 208

| **0**, **0** | 1, 1 |
|---|---|
| 0, 0 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

59

SQ Position 209

| **0**, **0** | 1, 1 |
|---|---|
| 0, 0 | -1, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

478

SQ Position 210

| **0**, **0** | 1, 1 |
|---|---|
| 0, 0 | -1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

218, 257, 259

SQ Position 211

| **0**, **0** | 1, 1 |
|---|---|
| 0, -1 | -1, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

396

SQ Position 212

| **0**, **0** | 1, 1 |
|---|---|
| 0, -1 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

163, 567, 592

SQ Position 213

| **0**, **0** | 1, 1 |
|---|---|
| 0, -1 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

639, 640, 646

SQ Position 214

| **0**, **0** | 1, 1 |
|---|---|
| 0, 1 | -1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

299, 433, 446

SQ Position 215

| **0**, **0** | 1, 1 |
|---|---|
| 0, 1 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

625, 626, 628

SQ Position 216

| **0**, **0** | 1, 1 |
|---|---|
| 0, 1 | -1, 1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
14

345, 397, 490, 531, 568, 601, 630, 630, 633, 634, 636, 643, 644, 648

SQ Position 217

| **0**, **0** | 1, 0 |
|---|---|
| 0, 0 | 1, 0 |

Ord% 448→0.000854492
Card% 22800→2.85e-05
4

3, 3, 4, 4

SQ Position 218

| **0**, **0** | 1, 0 |
|---|---|
| 0, 0 | 1, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

117, 155, 164

SQ Position 219

| **0**, **0** | 1, 0 |
|---|---|
| 0, 0 | 1, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

24, 25, 26

SQ Position 220

| **0**, **0** | 1, 0 |
|---|---|
| 0, -1 | 1, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

101, 136, 162

SQ Position 221

| **0**, **0** | 1, 0 |
|---|---|
| 0, -1 | 1, -1 |

Ord% 1568→0.00299072
Card%
649800→0.00081225
10

95, 95, 96, 97, 105, 121, 140, 159, 188, 189

SQ Position 222

| **0**, **0** | 1, 0 |
|---|---|
| 0, -1 | 1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

109, 145, 163

SQ Position 223

| **0**, **0** | 1, 0 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 672→0.00128174
Card% 102600→0.000128253

57, 62, 78

SQ Position 224

| **0**, **0** | 1, 0 |
|---|---|
| 0, 1 | 1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342

124, 165, 176

SQ Position 225

| **0**, **0** | 1, 0 |
|---|---|
| 0, 1 | 1, 1 |

Ord% 1568→0.00299072
Card% 649800→0.000812251

95, 95, 96, 96, 97, 97, 131, 131, 132, 132, 181, 181

SQ Position 226

| **0**, **0** | 1, -1 |
|---|---|
| 0, 0 | 1, 0 |

Ord% 672→0.00128174
Card% 102600→0.000128253

117, 155, 164

SQ Position 227

| **0**, **0** | 1, -1 |
|---|---|
| 0, 0 | 1, -1 |

Ord% 1568→0.00299072
Card% 649800→0.000812251

116, 116, 128, 128, 154, 154, 179, 179, 186, 186

SQ Position 228

| **0**, **0** | 1, -1 |
|---|---|
| 0, 0 | 1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342

118, 156, 169

SQ Position 229

| **0**, **0** | 1, -1 |
|---|---|
| 0, -1 | 1, 0 |

Ord% 1568→0.00299072
Card% 649800→0.000812251

100, 112, 125, 135, 148, 152, 170, 177, 183

SQ Position 230

| **0**, **0** | 1, -1 |
|---|---|
| 0, -1 | 1, -1 |

Ord% 4032→0.00769043
Card% 4.617e+06→0.00577125

57, 62, 78, 97, 103, 104, 107, 113, 119, 120, 123, 124, 126, 129, 130, 132, 138, 139, 142, 149, 157, 158, 161, 165, 171, 174, 175, 176, 180, 181, 181, 182, 185, 187, 192, 193, 194, 195, 196, 197

SQ Position 231

| **0**, **0** | 1, -1 |
|---|---|
| 0, -1 | 1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.00153999

108, 115, 127, 144, 151, 153, 173, 178, 184

SQ Position 232

| | |
|---|---|
| **0**, **0** | 1, -1 |
| 0, 1 | 1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

120, 158, 185

SQ Position 233

| | |
|---|---|
| **0**, **0** | 1, -1 |
| 0, 1 | 1, -1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

119, 129, 130, 157, 175, 180, 187, 193, 195

SQ Position 234

| | |
|---|---|
| **0**, **0** | 1, -1 |
| 0, 1 | 1, 1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

121, 122, 123, 159, 160, 161, 189, 191, 197

SQ Position 235

| | |
|---|---|
| **0**, **0** | 1, 1 |
| 0, 0 | 1, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

24, 25, 26

SQ Position 236

| | |
|---|---|
| **0**, **0** | 1, 1 |
| 0, 0 | 1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

118, 156, 169

SQ Position 237

| | |
|---|---|
| **0**, **0** | 1, 1 |
| 0, 0 | 1, 1 |

Ord% 1568→0.00299072
Card%
649800→0.00081225
10

98, 98, 99, 99, 133, 133, 134, 134, 143, 143

SQ Position 238

| | |
|---|---|
| **0**, **0** | 1, 1 |
| 0, -1 | 1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

102, 137, 166

SQ Position 239

| | |
|---|---|
| **0**, **0** | 1, 1 |
| 0, -1 | 1, -1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
10

96, 106, 122, 131, 131, 132, 141, 160, 190, 191

SQ Position 240

| | |
|---|---|
| **0**, **0** | 1, 1 |
| 0, -1 | 1, 1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

110, 111, 114, 146, 147, 150, 167, 168, 172

SQ Position 241

| **0**, **0** | 1, 1 |
|---|---|
| 0, 1 | 1, 0 |

Ord% 1568→0.00299072
Card%
649800→0.00081225
9

100, 103, 108, 135, 138, 144, 152, 153, 174

SQ Position 242

| **0**, **0** | 1, 1 |
|---|---|
| 0, 1 | 1, -1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

125, 126, 127, 170, 171, 173, 177, 178, 192

SQ Position 243

| **0**, **0** | 1, 1 |
|---|---|
| 0, 1 | 1, 1 |

Ord% 4032→0.00769043
Card%
4.617e+06→0.00577125
39

101, 102, 104, 105, 106, 107, 109, 110, 111, 112, 113, 114, 115, 136, 137, 139, 140, 141, 142, 145, 146, 147, 148, 149, 150, 151, 162, 163, 166, 167, 168, 172, 182, 183, 184, 188, 190, 194, 196

SQ Position 244

| **0**, **0** | 0, 0 |
|---|---|
| -1, 0 | 0, 0 |

Ord% 192→0.000366211
Card% 3600→4.5e-06
1

8

SQ Position 245

| **0**, **0** | 0, 0 |
|---|---|
| -1, 0 | 0, -1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

337

SQ Position 246

| **0**, **0** | 0, 0 |
|---|---|
| -1, 0 | 0, 1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

50

SQ Position 247

| **0**, **0** | 0, 0 |
|---|---|
| -1, -1 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
2

321, 321

SQ Position 248

| **0**, **0** | 0, 0 |
|---|---|
| -1, -1 | 0, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

101, 325, 340

SQ Position 249

| **0**, **0** | 0, 0 |
|---|---|
| -1, -1 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

329

SQ Position 250

| **0**, **0** | 0, 0 |
|---|---|
| -1, 1 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

60

SQ Position 251

| **0**, **0** | 0, 0 |
|---|---|
| -1, 1 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

355

SQ Position 252

| **0**, **0** | 0, 0 |
|---|---|
| -1, 1 | 0, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

117, 155, 164

SQ Position 253

| **0**, **0** | 0, -1 |
|---|---|
| -1, 0 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
2

344, 344

SQ Position 254

| **0**, **0** | 0, -1 |
|---|---|
| -1, 0 | 0, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

217, 358, 364

SQ Position 255

| **0**, **0** | 0, -1 |
|---|---|
| -1, 0 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

349

SQ Position 256

| **0**, **0** | 0, -1 |
|---|---|
| -1, -1 | 0, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

292, 332, 350

SQ Position 257

| **0**, **0** | 0, -1 |
|---|---|
| -1, -1 | 0, -1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

58, 162, 256, 323, 324, 327,
333, 339, 342, 345, 351,
359, 365

SQ Position 258

| **0**, **0** | 0, -1 |
|---|---|
| -1, -1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

328, 335, 353

SQ Position 259

| **0**, **0** | 0, -1 |
|---|---|
| -1, 1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

363

SQ Position 260

| **0**, **0** | 0, -1 |
|---|---|
| -1, 1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

258, 371, 373

SQ Position 261

| **0**, **0** | 0, -1 |
|---|---|
| -1, 1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

367, 369, 375

SQ Position 262

| **0**, **0** | 0, 1 |
|---|---|
| -1, 0 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

30

SQ Position 263

| **0**, **0** | 0, 1 |
|---|---|
| -1, 0 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

338

SQ Position 264

| **0**, **0** | 0, 1 |
|---|---|
| -1, 0 | 0, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

201, 232, 250

SQ Position 265

| **0**, **0** | 0, 1 |
|---|---|
| -1, -1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

322

SQ Position 266

| **0**, **0** | 0, 1 |
|---|---|
| -1, -1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

136, 326, 341

SQ Position 267

| **0**, **0** | 0, 1 |
|---|---|
| -1, -1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

330, 331, 334

SQ Position 268

| **0**, **0** | 0, 1 |
|---|---|
| -1, 1 | 0, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

308, 336, 354

SQ Position 269

| **0**, **0** | 0, 1 |
|---|---|
| -1, 1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

356, 357, 370

SQ Position 270

| **0**, **0** | 0, 1 |
|---|---|
| -1, 1 | 0, 1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

337, 343, 346, 347, 348, 352, 360, 361, 362, 366, 368, 372, 374

SQ Position 271

| **0**, **0** | 0, 0 |
|---|---|
| -1, 0 | -1, 0 |

Ord% 448→0.000854492
Card% 22800→2.85e-05
4

5, 5, 16, 16

SQ Position 272

| **0**, **0** | 0, 0 |
|---|---|
| -1, 0 | -1, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

201, 330, 347

SQ Position 273

| **0**, **0** | 0, 0 |
|---|---|
| -1, 0 | -1, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

49, 76, 83

SQ Position 274

| **0**, **0** | 0, 0 |
|---|---|
| -1, -1 | -1, 0 |

Ord% 672→0.00128174
Card% 102600→0.00012825
3

201, 330, 347

SQ Position 275

| **0**, **0** | 0, 0 |
|---|---|
| -1, -1 | -1, -1 |

Ord% 1568→0.00299072
Card% 649800→0.00081225
10

98, 98, 110, 110, 205, 205, 521, 521, 532, 532

SQ Position 276

| **0**, **0** | 0, 0 |
|---|---|
| -1, -1 | -1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

210, 623, 631

SQ Position 277

| **0**, **0** | 0, 0 |
|---|---|
| -1, 1 | -1, 0 |

Ord% 672→0.00128174
Card% 102600→0.00012825
3

49, 76, 83

SQ Position 278

| **0**, **0** | 0, 0 |
|---|---|
| -1, 1 | -1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

210, 623, 631

SQ Position 279

| **0**, **0** | 0, 0 |
|---|---|
| -1, 1 | -1, 1 |

Ord% 1568→0.00299072
Card% 649800→0.00081225
10

116, 116, 128, 128, 154, 154, 179, 179, 186, 186

SQ Position 280

| **0**, **0** | 0, -1 |
|---|---|
| -1, 0 | -1, 0 |

Ord% 672→0.00128174
Card% 102600→0.00012825
3

217, 358, 364

SQ Position 281

| **0**, **0** | 0, -1 |
|---|---|
| -1, 0 | -1, -1 |

Ord% 1568→0.00299072
Card% 649800→0.00081225
12

198, 198, 211, 211, 220, 220, 649, 649, 655, 655, 666, 666

SQ Position 282

| **0**, **0** | 0, -1 |
|---|---|
| -1, 0 | -1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

222, 501, 505

SQ Position 283

| **0**, **0** | 0, -1 |
|---|---|
| -1, -1 | -1, 0 |

Ord% 1568→0.00299072
Card% 649800→0.000812259

200, 213, 223, 300, 315, 651, 657, 661, 664

SQ Position 284

| **0**, **0** | 0, -1 |
|---|---|
| -1, -1 | -1, -1 |

Ord% 4032→0.00769043
Card% 4.617e+06→0.0057712539

27, 40, 64, 133, 146, 167, 199, 202, 203, 204, 206, 207, 212, 214, 218, 221, 224, 242, 261, 384, 398, 434, 447, 479, 491, 559, 569, 593, 602, 639, 643, 650, 652, 656, 658, 662, 663, 665, 667

SQ Position 285

| **0**, **0** | 0, -1 |
|---|---|
| -1, -1 | -1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

209, 216, 226, 457, 462, 654, 660, 668, 670

SQ Position 286

| **0**, **0** | 0, -1 |
|---|---|
| -1, 1 | -1, 0 |

Ord% 448→0.000854492
Card% 273600→0.0003423

219, 409, 415

SQ Position 287

| **0**, **0** | 0, -1 |
|---|---|
| -1, 1 | -1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

208, 215, 225, 273, 279, 653, 659, 669, 671

SQ Position 288

| **0**, **0** | 0, -1 |
|---|---|
| -1, 1 | -1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

227, 228, 229, 541, 545, 578, 582, 611, 615

SQ Position 289

| **0**, **0** | 0, 1 |
|---|---|
| -1, 0 | -1, 0 |

Ord% 672→0.00128174
Card% 102600→0.000128253

27, 40, 64

SQ Position 290

| **0**, **0** | 0, 1 |
|---|---|
| -1, 0 | -1, -1 |

Ord% 448→0.000854492
Card% 273600→0.0003423

204, 479, 491

SQ Position 291

| **0**, **0** | 0, 1 |
|---|---|
| -1, 0 | -1, 1 |

Ord% 1568→0.00299072
Card% 649800→0.0008122510

198, 198, 199, 208, 211, 220, 242, 261, 273, 279

SQ Position 292

| **0**, **0** | 0, 1 |
|---|---|
| -1, -1 | -1, 0 |

Ord% 448→0.000854492
Card% 273600→0.0003423

202, 384, 398

SQ Position 293

| **0**, **0** | 0, 1 |
|---|---|
| -1, -1 | -1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

133, 146, 167, 206, 207, 559, 569, 593, 602

SQ Position 294

| **0**, **0** | 0, 1 |
|---|---|
| -1, -1 | -1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

211, 212, 215, 649, 649, 650, 653, 655, 663, 669

SQ Position 295

| **0**, **0** | 0, 1 |
|---|---|
| -1, 1 | -1, 0 |

Ord% 1568→0.00299072
Card% 649800→0.000812259

200, 203, 209, 300, 315, 434, 447, 457, 462

SQ Position 296

| **0**, **0** | 0, 1 |
|---|---|
| -1, 1 | -1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

213, 214, 216, 651, 652, 654, 661, 662, 668

SQ Position 297

| **0**, **0** | 0, 1 |
|---|---|
| -1, 1 | -1, 1 |

Ord% 4032→0.00769043
Card% 4.617e+06→0.0057712540

217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 358, 364, 409, 415, 501, 505, 541, 545, 578, 582, 611, 615, 639, 643, 655, 656, 657, 658, 659, 660, 664, 665, 666, 666, 667, 670, 671

SQ Position 298

| **0**, **0** | 0, 0 |
|---|---|
| -1, 0 | 1, 0 |

Ord% 128→0.000244141
Card% 9600→1.2e-05

11

SQ Position 299

| **0**, **0** | 0, 0 |
|---|---|
| -1, 0 | 1, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05

360

SQ Position 300

| **0**, **0** | 0, 0 |
|---|---|
| -1, 0 | 1, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05

51

SQ Position 301

| **0**, **0** | 0, 0 |
|---|---|
| -1, -1 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

324

SQ Position 302

| **0**, **0** | 0, 0 |
|---|---|
| -1, -1 | 1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

104, 475, 486

SQ Position 303

| **0**, **0** | 0, 0 |
|---|---|
| -1, -1 | 1, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

478

SQ Position 304

| **0**, **0** | 0, 0 |
|---|---|
| -1, 1 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

66

SQ Position 305

| **0**, **0** | 0, 0 |
|---|---|
| -1, 1 | 1, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

498

SQ Position 306

| **0**, **0** | 0, 0 |
|---|---|
| -1, 1 | 1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

118, 156, 169

SQ Position 307

| **0**, **0** | 0, -1 |
|---|---|
| -1, 0 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

349

SQ Position 308

| **0**, **0** | 0, -1 |
|---|---|
| -1, 0 | 1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

222, 501, 505

SQ Position 309

| **0**, **0** | 0, -1 |
|---|---|
| -1, 0 | 1, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
2

493, 493

SQ Position 310

| **0**, **0** | 0, -1 |
|---|---|
| -1, -1 | 1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

295, 481, 494

SQ Position 311

| **0**, **0** | 0, -1 |
|---|---|
| -1, -1 | 1, -1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
14

79, 182, 275, 411, 428, 474,
474, 477, 482, 488, 490,
495, 502, 506

SQ Position 312

| **0**, **0** | 0, -1 |
|---|---|
| -1, -1 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

449, 484, 497

SQ Position 313

| **0**, **0** | 0, -1 |
|---|---|
| -1, 1 | 1, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

400

SQ Position 314

| **0**, **0** | 0, -1 |
|---|---|
| -1, 1 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

263, 512, 514

SQ Position 315

| **0**, **0** | 0, -1 |
|---|---|
| -1, 1 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

508, 510, 516

SQ Position 316

| **0**, **0** | 0, 1 |
|---|---|
| -1, 0 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

33

SQ Position 317

| **0**, **0** | 0, 1 |
|---|---|
| -1, 0 | 1, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

485

SQ Position 318

| **0**, **0** | 0, 1 |
|---|---|
| -1, 0 | 1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

204, 235, 251

SQ Position 319

| **0**, **0** | 0, 1 |
|---|---|
| -1, -1 | 1, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

378

SQ Position 320

| **0**, **0** | 0, 1 |
|---|---|
| -1, -1 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

139, 476, 487

SQ Position 321

| **0**, **0** | 0, 1 |
|---|---|
| -1, -1 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

479, 480, 483

SQ Position 322

| **0**, **0** | 0, 1 |
|---|---|
| -1, 1 | 1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

309, 441, 459

SQ Position 323

| **0**, **0** | 0, 1 |
|---|---|
| -1, 1 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

499, 500, 511

SQ Position 324

| **0**, **0** | 0, 1 |
|---|---|
| -1, 1 | 1, 1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

338, 391, 485, 489, 491, 492, 496, 503, 504, 507, 509, 513, 515

SQ Position 325

| **0**, **0** | -1, 0 |
|---|---|
| -1, 0 | 0, 0 |

Ord% 448→0.000854492
Card% 22800→2.85e-05
4

7, 7, 18, 18

SQ Position 326

| **0**, **0** | -1, 0 |
|---|---|
| -1, 0 | 0, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

308, 356, 361

SQ Position 327

| **0**, **0** | -1, 0 |
|---|---|
| -1, 0 | 0, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

47, 74, 80

SQ Position 328

| **0**, **0** | -1, 0 |
|---|---|
| -1, -1 | 0, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

292, 332, 350

SQ Position 329

| **0**, **0** | -1, 0 |
|---|---|
| -1, -1 | 0, -1 |

Ord% 1568→0.00299072
Card%
649800→0.00081225
9

100, 112, 125, 296, 311,
523, 534, 539, 543

SQ Position 330

| **0**, **0** | -1, 0 |
|---|---|
| -1, -1 | 0, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

299, 625, 633

SQ Position 331

| **0**, **0** | -1, 0 |
|---|---|
| -1, 1 | 0, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

29, 42, 67

SQ Position 332

| **0**, **0** | -1, 0 |
|---|---|
| -1, 1 | 0, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

314, 637, 641

SQ Position 333

| **0**, **0** | -1, 0 |
|---|---|
| -1, 1 | 0, 1 |

Ord% 1568→0.00299072
Card%
649800→0.00081225
9

100, 112, 125, 135, 148,
152, 170, 177, 183

SQ Position 334

| **0**, **0** | -1, -1 |
|---|---|
| -1, 0 | 0, 0 |

Ord% 672→0.00128174
Card% 102600→0.000128253

292, 332, 350

SQ Position 335

| **0**, **0** | -1, -1 |
|---|---|
| -1, 0 | 0, -1 |

Ord% 1568→0.00299072
Card% 649800→0.000812259

200, 213, 223, 300, 315, 651, 657, 661, 664

SQ Position 336

| **0**, **0** | -1, -1 |
|---|---|
| -1, 0 | 0, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342003

295, 481, 494

SQ Position 337

| **0**, **0** | -1, -1 |
|---|---|
| -1, -1 | 0, 0 |

Ord% 1568→0.00299072
Card% 649800→0.000812259

291, 291, 291, 291, 302, 302, 302, 302, 691, 691, 691, 691, 696, 696, 696, 696

SQ Position 338

| **0**, **0** | -1, -1 |
|---|---|
| -1, -1 | 0, -1 |

Ord% 4032→0.00769043
Card% 4.617e+06→0.00577125

47, 74, 80, 152, 177, 183, 248, 271, 276, 294, 295, 298, 299, 303, 304, 305, 310, 313, 317, 407, 412, 436, 450, 481, 494, 576, 580, 595, 604, 625, 633, 682, 684, 692, 693, 694, 697, 698, 702

SQ Position 339

| **0**, **0** | -1, -1 |
|---|---|
| -1, -1 | 0, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.00153949

294, 303, 305, 436, 450, 692, 694, 697, 698

SQ Position 340

| **0**, **0** | -1, -1 |
|---|---|
| -1, 1 | 0, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342003

293, 386, 401

SQ Position 341

| **0**, **0** | -1, -1 |
|---|---|
| -1, 1 | 0, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.00153949

231, 244, 264, 301, 316, 673, 678, 695, 704

SQ Position 342

| **0**, **0** | -1, -1 |
|---|---|
| -1, 1 | 0, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.00153949

296, 297, 298, 523, 534, 561, 571, 595, 604

SQ Position 343

| **0**, **0** | -1, 1 |
|---|---|
| -1, 0 | 0, 0 |

Ord% 672→0.00128174
Card% 102600→0.000128253

29, 42, 67

SQ Position 344

| **0**, **0** | -1, 1 |
|---|---|
| -1, 0 | 0, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342

309, 499, 503

SQ Position 345

| **0**, **0** | -1, 1 |
|---|---|
| -1, 0 | 0, 1 |

Ord% 1568→0.00299072
Card% 649800→0.000812259

200, 213, 223, 231, 244, 248, 264, 271, 276

SQ Position 346

| **0**, **0** | -1, 1 |
|---|---|
| -1, -1 | 0, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342

293, 386, 401

SQ Position 347

| **0**, **0** | -1, 1 |
|---|---|
| -1, -1 | 0, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

135, 148, 170, 297, 312, 561, 571, 609, 613

SQ Position 348

| **0**, **0** | -1, 1 |
|---|---|
| -1, -1 | 0, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

300, 301, 304, 651, 657, 673, 678, 693, 702

SQ Position 349

| **0**, **0** | -1, 1 |
|---|---|
| -1, 1 | 0, 0 |

Ord% 1568→0.00299072
Card% 649800→0.000812251

306, 306, 306, 306, 307, 307, 318, 318, 455, 455, 460, 460

SQ Position 350

| **0**, **0** | -1, 1 |
|---|---|
| -1, 1 | 0, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

318, 319, 320, 699, 699, 700, 701, 703, 705

SQ Position 351

| **0**, **0** | -1, 1 |
|---|---|
| -1, 1 | 0, 1 |

Ord% 4032→0.00769043
Card% 4.617e+06→0.005771253

308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 356, 361, 407, 412, 499, 503, 539, 543, 576, 580, 609, 613, 637, 641, 661, 664, 682, 684, 695, 699, 699, 700, 701, 703, 704, 705

SQ Position 352

| **0**, **0** | -1, 0 |
|---|---|
| -1, 0 | -1, 0 |

Ord% 1152→0.00219727
Card% 162000→0.0002025
13

2, 4, 6, 9, 10, 11, 13, 14, 15, 17, 19, 20, 21

SQ Position 353

| **0**, **0** | -1, 0 |
|---|---|
| -1, 0 | -1, -1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

30, 136, 232, 322, 326, 331, 336, 338, 341, 343, 348, 357, 362

SQ Position 354

| **0**, **0** | -1, 0 |
|---|---|
| -1, 0 | -1, 1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
14

46, 46, 71, 72, 73, 78, 79, 82, 89, 90, 91, 92, 93, 94

SQ Position 355

| **0**, **0** | -1, 0 |
|---|---|
| -1, -1 | -1, 0 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

50, 155, 250, 328, 329, 334, 335, 346, 349, 352, 353, 368, 369

SQ Position 356

| **0**, **0** | -1, 0 |
|---|---|
| -1, -1 | -1, -1 |

Ord% 4032→0.00769043
Card%
4.617e+06→0.00577125
39

24, 34, 54, 96, 99, 102, 106, 108, 109, 111, 114, 115, 118, 122, 127, 140, 159, 236, 253, 379, 393, 464, 465, 507, 508, 518, 520, 522, 525, 526, 528, 530, 533, 536, 537, 540, 544, 548, 549

SQ Position 357

| **0**, **0** | -1, 0 |
|---|---|
| -1, -1 | -1, 1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
14

73, 176, 270, 406, 446, 490, 575, 601, 622, 622, 627, 628, 635, 636

SQ Position 358

| **0**, **0** | -1, 0 |
|---|---|
| -1, 1 | -1, 0 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

23, 26, 32, 33, 36, 43, 52, 53, 56, 61, 68, 77, 84

SQ Position 359

| **0**, **0** | -1, 0 |
|---|---|
| -1, 1 | -1, -1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

39, 145, 241, 383, 445, 489, 558, 600, 624, 629, 632, 638, 642

SQ Position 360

| **0**, **0** | -1, 0 |
|---|---|
| -1, 1 | -1, 1 |

Ord% 4032→0.00769043
Card%
4.617e+06→0.00577125
40

57, 62, 78, 97, 103, 104, 107, 113, 119, 120, 123, 124, 126, 129, 130, 132, 138, 139, 142, 149, 157, 158, 161, 165, 171, 174, 175, 176, 180, 181, 181, 182, 185, 187, 192, 193, 194, 195, 196, 197

SQ Position 361

| **0**, **0** | -1, -1 |
|---|---|
| -1, 0 | -1, 0 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

58, 162, 256, 323, 324, 327, 333, 339, 342, 345, 351, 359, 365

SQ Position 362

| **0**, **0** | -1, -1 |
|---|---|
| -1, 0 | -1, -1 |

Ord% 4032→0.00769043
Card%
4.617e+06→0.00577125
39

27, 40, 64, 133, 146, 167, 199, 202, 203, 204, 206, 207, 212, 214, 218, 221, 224, 242, 261, 384, 398, 434, 447, 479, 491, 559, 569, 593, 602, 639, 643, 650, 652, 656, 658, 662, 663, 665, 667

SQ Position 363

| **0**, **0** | -1, -1 |
|---|---|
| -1, 0 | -1, 1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
14

79, 182, 275, 411, 428, 474, 474, 477, 482, 488, 490, 495, 502, 506

SQ Position 365

| **0**, **0** | -1, -1 |
|---|---|
| -1, -1 | -1, -1 |

Ord% 10368→0.0197754

Card% 3.2805e+07→0.0410063

182

22, 22, 25, 25, 28, 28, 31, 31, 35, 35, 41, 41, 48, 48, 51, 51, 55, 55, 59, 59, 65, 65, 75, 75, 81, 81, 131, 131, 134, 134, 137, 137, 141, 141, 147, 147, 153, 153, 156, 156, 160, 160, 163, 163, 168, 168, 178, 178, 184, 184, 230, 230, 233, 233, 237, 237, 243, 243, 249, 249, 251, 251, 254, 254, 257, 257, 262, 262, 272, 272, 277, 277, 376, 376, 380, 380, 385, 385, 390, 390, 391, 391, 394, 394, 396, 396, 399, 399, 408, 408, 413, 413, 432, 432, 433, 433, 438, 438, 439, 439, 449, 449, 452, 452, 453, 453, 466, 466, 467, 467, 478, 478, 483, 483, 484, 484, 493, 493, 496, 496, 497, 497, 509, 509, 510, 510, 556, 556, 560, 560, 565, 565, 567, 567, 570, 570, 577, 577, 581, 581, 592, 592, 597, 597, 598, 598, 606, 606, 607, 607, 617, 617, 626, 626, 630, 630, 634, 634, 640, 640, 644, 644, 672, 672, 677, 677, 683, 683, 685, 685, 707, 707, 708, 708, 711, 711, 720, 720, 722, 722, 725, 725

SQ Position 364

| **0**, **0** | -1, -1 |
|---|---|
| -1, -1 | -1, 0 |

Ord% 4032→0.00769043

Card% 4.617e+06→0.00577125

39

47, 74, 80, 152, 177, 183, 248, 271, 276, 294, 295, 298, 299, 303, 304, 305, 310, 313, 317, 407, 412, 436, 450, 481, 494, 576, 580, 595, 604, 625, 633, 682, 684, 692, 693, 694, 697, 698, 702

SQ Position 366

| **0**, **0** | -1, -1 |
|---|---|
| -1, -1 | -1, 1 |

Ord% 1728→0.0032959

Card% 8.748e+06→0.010935

42

71, 89, 91, 174, 192, 194, 268, 285, 287, 405, 421, 423, 427, 427, 428, 431, 437, 437, 444, 446, 451, 451, 458, 463, 482, 495, 585, 587, 596, 605, 628, 636, 687, 689, 706, 706, 710, 710, 714, 718, 724, 724

SQ Position 367

| **0**, **0** | -1, -1 |
|---|---|
| -1, 1 | -1, 0 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

63, 166, 260, 377, 378, 381, 387, 392, 395, 397, 402, 410, 416

SQ Position 368

| **0**, **0** | -1, -1 |
|---|---|
| -1, 1 | -1, -1 |

Ord% 1728→0.0032959
Card%
8.748e+06→0.010935
39

37, 44, 69, 143, 150, 172, 234, 235, 238, 239, 245, 246, 252, 255, 259, 265, 266, 274, 280, 388, 403, 435, 448, 480, 492, 563, 573, 594, 603, 646, 648, 674, 675, 679, 680, 686, 709, 719, 721

SQ Position 369

| **0**, **0** | -1, -1 |
|---|---|
| -1, 1 | -1, 1 |

Ord% 1728→0.0032959
Card%
8.748e+06→0.010935
40

85, 87, 93, 188, 190, 196, 281, 283, 289, 417, 419, 425, 429, 430, 431, 475, 476, 477, 519, 524, 529, 531, 535, 542, 546, 557, 562, 566, 568, 572, 579, 583, 589, 591, 591, 596, 601, 605, 612, 616

SQ Position 370

| **0**, **0** | -1, 1 |
|---|---|
| -1, 0 | -1, 0 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

23, 37, 38, 39, 44, 45, 62, 63, 66, 69, 70, 87, 88

SQ Position 371

| **0**, **0** | -1, 1 |
|---|---|
| -1, 0 | -1, -1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

33, 139, 235, 378, 459, 476, 480, 485, 487, 489, 492, 500, 504

SQ Position 372

| **0**, **0** | -1, 1 |
|---|---|
| -1, 0 | -1, 1 |

Ord% 4032→0.00769043
Card%
4.617e+06→0.00577125
40

49, 52, 72, 154, 157, 175, 208, 209, 210, 215, 216, 219, 222, 225, 226, 228, 229, 239, 240, 241, 246, 247, 260, 263, 266, 267, 268, 269, 269, 270, 275, 278, 283, 284, 285, 286, 287, 288, 289, 290

SQ Position 373

| **0**, **0** | -1, 1 |
|---|---|
| -1, -1 | -1, 0 |

Ord% 1152→0.00219727

Card% 1.944e+06→0.00243 13

53, 158, 252, 382, 383, 388, 389, 392, 400, 403, 404, 419, 420

SQ Position 374

| **0**, **0** | -1, 1 |
|---|---|
| -1, -1 | -1, -1 |

Ord% 1728→0.0032959

Card% 8.748e+06→0.010935 39

26, 36, 56, 132, 142, 143, 144, 145, 150, 151, 161, 166, 169, 172, 173, 190, 191, 238, 255, 381, 395, 472, 473, 515, 516, 557, 558, 563, 564, 566, 573, 574, 584, 594, 599, 600, 603, 610, 614

SQ Position 375

| **0**, **0** | -1, 1 |
|---|---|
| -1, -1 | -1, 1 |

Ord% 1728→0.0032959

Card% 8.748e+06→0.010935 40

76, 77, 90, 179, 180, 193, 273, 274, 286, 409, 410, 422, 462, 463, 471, 505, 506, 514, 578, 579, 586, 615, 616, 621, 623, 624, 627, 653, 654, 659, 660, 675, 676, 680, 681, 688, 714, 717, 717, 718

SQ Position 376

| | |
|---|---|
| **0**, **0** | -1, 1 |
| -1, 1 | -1, 0 |

Ord% 4032→0.00769043

Card% 4.617e+06→0.0057712539

29, 32, 38, 135, 138, 144, 231, 234, 240, 293, 297, 301, 307, 309, 312, 314, 316, 319, 320, 377, 382, 430, 435, 440, 440, 441, 443, 445, 448, 454, 456, 459, 461, 468, 469, 470, 471, 472, 473

SQ Position 377

| | |
|---|---|
| **0**, **0** | -1, 1 |
| -1, 1 | -1, -1 |

Ord% 1728→0.0032959

Card% 8.748e+06→0.01093539

42, 43, 45, 148, 149, 151, 244, 245, 247, 386, 387, 389, 460, 461, 470, 503, 504, 513, 561, 562, 564, 613, 614, 620, 637, 638, 645, 673, 674, 676, 695, 700, 701, 709, 712, 713, 713, 723, 726

SQ Position 378

| | |
|---|---|
| **0**, **0** | -1, 1 |
| -1, 1 | -1, 1 |

Ord% 10368→0.0197754

Card% 3.2805e+07→0.041006372

60, 61, 66, 67, 68, 70, 82, 83, 84, 86, 88, 92, 94, 164, 165, 169, 170, 171, 173, 185, 186, 187, 189, 191, 195, 197, 258, 259, 263, 264, 265, 267, 278, 279, 280, 282, 284, 288, 290, 354, 355, 360, 363, 370, 371, 372, 373, 374, 375, 397, 400, 401, 402, 404, 405, 406, 411, 414, 414, 415, 416, 418, 420, 421, 422, 423, 424, 424, 425, 426, 426, 441, 442, 443, 444, 454, 455, 456, 457, 458, 468, 469, 485, 486, 487, 488, 498, 498, 499, 500, 501, 502, 511, 511, 512, 512, 513, 514, 515, 516, 538, 550, 551, 552, 553, 554, 555, 568, 571, 572, 574, 575, 582, 583, 584, 585, 586, 587, 588, 588, 589, 590, 590, 599, 608, 608, 609, 610, 611, 612, 618, 618, 619, 619, 620, 621, 629, 631, 632, 635, 641, 642, 645, 646, 647, 647, 648, 668, 669, 670, 671, 678, 679, 681, 686, 687, 688, 689, 690, 690, 703, 704, 705, 712, 715, 715, 716, 716, 719, 721, 723, 726

SQ Position 379

| **0**, **0** | -1, 0 |
|---|---|
| -1, 0 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
3

10, 19, 21

SQ Position 380

| **0**, **0** | -1, 0 |
|---|---|
| -1, 0 | 1, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

354, 370, 372

SQ Position 381

| **0**, **0** | -1, 0 |
|---|---|
| -1, 0 | 1, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

48, 75, 81

SQ Position 382

| **0**, **0** | -1, 0 |
|---|---|
| -1, -1 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

323, 333, 351

SQ Position 383

| **0**, **0** | -1, 0 |
|---|---|
| -1, -1 | 1, -1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

103, 113, 126, 429, 442, 524, 535, 550, 552

SQ Position 384

| **0**, **0** | -1, 0 |
|---|---|
| -1, -1 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

433, 626, 634

SQ Position 385

| **0**, **0** | -1, 0 |
|---|---|
| -1, 1 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

38, 45, 70

SQ Position 386

| **0**, **0** | -1, 0 |
|---|---|
| -1, 1 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

454, 645, 647

SQ Position 387

| **0**, **0** | -1, 0 |
|---|---|
| -1, 1 | 1, 1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

108, 115, 127, 144, 151, 153, 173, 178, 184

SQ Position 388

| **0**, **0** | -1, -1 |
|---|---|
| -1, 0 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

328, 335, 353

SQ Position 389

| **0**, **0** | -1, -1 |
|---|---|
| -1, 0 | 1, -1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

209, 216, 226, 457, 462, 654, 660, 668, 670

SQ Position 390

| **0**, **0** | -1, -1 |
|---|---|
| -1, 0 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

449, 484, 497

SQ Position 391

| **0**, **0** | -1, -1 |
|---|---|
| -1, -1 | 1, 0 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

294, 303, 305, 436, 450, 692, 694, 697, 698

SQ Position 392

| **0**, **0** | -1, -1 |
|---|---|
| -1, -1 | 1, -1 |

Ord% 1728→0.0032959
Card%
8.748e+06→0.010935
42

71, 89, 91, 174, 192, 194, 268, 285, 287, 405, 421, 423, 427, 427, 428, 431, 437, 437, 444, 446, 451, 451, 458, 463, 482, 495, 585, 587, 596, 605, 628, 636, 687, 689, 706, 706, 710, 710, 714, 718, 724, 724

SQ Position 393

| **0**, **0** | -1, -1 |
|---|---|
| -1, -1 | 1, 1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
12

432, 432, 439, 439, 453, 453, 708, 708, 711, 711, 725, 725

SQ Position 394

| **0**, **0** | -1, -1 |
|---|---|
| -1, 1 | 1, 0 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

382, 389, 404

SQ Position 395

| **0**, **0** | -1, -1 |
|---|---|
| -1, 1 | 1, -1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
9

240, 247, 267, 469, 471, 676, 681, 716, 723

SQ Position 396

| **0**, **0** | -1, -1 |
|---|---|
| -1, 1 | 1, 1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
9

465, 467, 473, 526, 537, 564, 574, 598, 607

SQ Position 397

| **0**, **0** | -1, 1 |
|---|---|
| -1, 0 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

32, 43, 68

SQ Position 398

| **0**, **0** | -1, 1 |
|---|---|
| -1, 0 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

441, 511, 513

SQ Position 399

| **0**, **0** | -1, 1 |
|---|---|
| -1, 0 | 1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

203, 214, 224, 234, 245, 249, 265, 272, 277

SQ Position 400

| **0**, **0** | -1, 1 |
|---|---|
| -1, -1 | 1, 0 |

Ord% 192→0.000366211
Card% 518400→0.0006483

377, 387, 402

SQ Position 401

| **0**, **0** | -1, 1 |
|---|---|
| -1, -1 | 1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

138, 149, 171, 430, 443, 562, 572, 618, 620

SQ Position 402

| **0**, **0** | -1, 1 |
|---|---|
| -1, -1 | 1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

434, 435, 438, 652, 658, 674, 679, 707, 720

SQ Position 403

| **0**, **0** | -1, 1 |
|---|---|
| -1, 1 | 1, 0 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

307, 319, 320, 440, 440, 456, 461, 468, 470

SQ Position 404

| **0**, **0** | -1, 1 |
|---|---|
| -1, 1 | 1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

455, 456, 468, 703, 705, 712, 715, 715, 726

SQ Position 405

| **0**, **0** | -1, 1 |
|---|---|
| -1, 1 | 1, 1 |

Ord% 1728→0.0032959
Card% 8.748e+06→0.0109353

336, 357, 362, 390, 408, 413, 445, 447, 448, 452, 459, 460, 461, 464, 466, 470, 472, 500, 504, 540, 544, 577, 581, 610, 614, 638, 642, 662, 665, 683, 685, 700, 701, 709, 712, 713, 713, 721, 726

SQ Position 406

| **0**, **0** | 1, 0 |
|---|---|
| -1, 0 | 0, 0 |

Ord% 128→0.000244141
Card% 9600→1.2e-05
1

9

SQ Position 407

| **0**, **0** | 1, 0 |
|---|---|
| -1, 0 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

346

SQ Position 408

| **0**, **0** | 1, 0 |
|---|---|
| -1, 0 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

53

SQ Position 409

| **0**, **0** | 1, 0 |
|---|---|
| -1, -1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

322

SQ Position 410

| **0**, **0** | 1, 0 |
|---|---|
| -1, -1 | 0, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

102, 379, 393

SQ Position 411

| **0**, **0** | 1, 0 |
|---|---|
| -1, -1 | 0, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

383

SQ Position 412

| **0**, **0** | 1, 0 |
|---|---|
| -1, 1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

82

SQ Position 413

| **0**, **0** | 1, 0 |
|---|---|
| -1, 1 | 0, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

406

SQ Position 414

| **0**, **0** | 1, 0 |
|---|---|
| -1, 1 | 0, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

120, 158, 185

SQ Position 415

| **0**, **0** | 1, -1 |
|---|---|
| -1, 0 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

363

SQ Position 416

| **0**, **0** | 1, -1 |
|---|---|
| -1, 0 | 0, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

219, 409, 415

SQ Position 417

| **0**, **0** | 1, -1 |
|---|---|
| -1, 0 | 0, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

400

SQ Position 418

| **0**, **0** | 1, -1 |
|---|---|
| -1, -1 | 0, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

293, 386, 401

SQ Position 419

| **0**, **0** | 1, -1 |
|---|---|
| -1, -1 | 0, -1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

63, 166, 260, 377, 378, 381, 387, 392, 395, 397, 402, 410, 416

SQ Position 420

| **0**, **0** | 1, -1 |
|---|---|
| -1, -1 | 0, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

382, 389, 404

SQ Position 421

| **0**, **0** | 1, -1 |
|---|---|
| -1, 1 | 0, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
2

414, 414

SQ Position 422

| **0**, **0** | 1, -1 |
|---|---|
| -1, 1 | 0, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

278, 422, 424

SQ Position 423

| **0**, **0** | 1, -1 |
|---|---|
| -1, 1 | 0, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

418, 420, 426

SQ Position 424

| **0**, **0** | 1, 1 |
|---|---|
| -1, 0 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

31

SQ Position 425

| **0**, **0** | 1, 1 |
|---|---|
| -1, 0 | 0, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

391

SQ Position 426

| **0**, **0** | 1, 1 |
|---|---|
| -1, 0 | 0, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

202, 233, 252

SQ Position 427

| **0**, **0** | 1, 1 |
|---|---|
| -1, -1 | 0, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
2

376, 376

SQ Position 428

| **0**, **0** | 1, 1 |
|---|---|
| -1, -1 | 0, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

137, 380, 394

SQ Position 429

| **0**, **0** | 1, 1 |
|---|---|
| -1, -1 | 0, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

384, 385, 388

SQ Position 430

| **0**, **0** | 1, 1 |
|---|---|
| -1, 1 | 0, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

310, 390, 405

SQ Position 431

| **0**, **0** | 1, 1 |
|---|---|
| -1, 1 | 0, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

407, 408, 421

SQ Position 432

| **0**, **0** | 1, 1 |
|---|---|
| -1, 1 | 0, 1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

339, 392, 396, 398, 399, 403, 411, 412, 413, 417, 419, 423, 425

SQ Position 433

| **0**, **0** | 1, 0 |
|---|---|
| -1, 0 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
3

6, 17, 20

SQ Position 434

| **0**, **0** | 1, 0 |
|---|---|
| -1, 0 | -1, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

250, 334, 352

SQ Position 435

| **0**, **0** | 1, 0 |
|---|---|
| -1, 0 | -1, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

52, 77, 84

SQ Position 436

| **0**, **0** | 1, 0 |
|---|---|
| -1, -1 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

232, 331, 348

SQ Position 437

| **0**, **0** | 1, 0 |
|---|---|
| -1, -1 | -1, -1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

99, 111, 114, 236, 253, 522, 525, 533, 536

SQ Position 438

| **0**, **0** | 1, 0 |
|---|---|
| -1, -1 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

241, 624, 632

SQ Position 439

| **0**, **0** | 1, 0 |
|---|---|
| -1, 1 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

72, 90, 92

SQ Position 440

| **0**, **0** | 1, 0 |
|---|---|
| -1, 1 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

270, 627, 635

SQ Position 441

| **0**, **0** | 1, 0 |
|---|---|
| -1, 1 | -1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

119, 129, 130, 157, 175, 180, 187, 193, 195

SQ Position 442

| **0**, **0** | 1, -1 |
|---|---|
| -1, 0 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

258, 371, 373

SQ Position 443

| **0**, **0** | 1, -1 |
|---|---|
| -1, 0 | -1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

208, 215, 225, 273, 279, 653, 659, 669, 671

SQ Position 444

| **0**, **0** | 1, -1 |
|---|---|
| -1, 0 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

263, 512, 514

SQ Position 445

| **0**, **0** | 1, -1 |
|---|---|
| -1, -1 | -1, 0 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

231, 244, 264, 301, 316, 673, 678, 695, 704

SQ Position 446

| **0**, **0** | 1, -1 |
|---|---|
| -1, -1 | -1, -1 |

Ord% 1728→0.0032959
Card% 8.748e+06→0.01093539

37, 44, 69, 143, 150, 172, 234, 235, 238, 239, 245, 246, 252, 255, 259, 265, 266, 274, 280, 388, 403, 435, 448, 480, 492, 563, 573, 594, 603, 646, 648, 674, 675, 679, 680, 686, 709, 719, 721

SQ Position 447

| **0**, **0** | 1, -1 |
|---|---|
| -1, -1 | -1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

240, 247, 267, 469, 471, 676, 681, 716, 723

SQ Position 448

| **0**, **0** | 1, -1 |
|---|---|
| -1, 1 | -1, 0 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

278, 422, 424

SQ Position 449

| **0**, **0** | 1, -1 |
|---|---|
| -1, 1 | -1, -1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
12

269, 269, 286, 286, 288, 288, 688, 688, 690, 690, 717, 717

SQ Position 450

| **0**, **0** | 1, -1 |
|---|---|
| -1, 1 | -1, 1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
9

282, 284, 290, 551, 553, 586, 588, 619, 621

SQ Position 451

| **0**, **0** | 1, 1 |
|---|---|
| -1, 0 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

28, 41, 65

SQ Position 452

| **0**, **0** | 1, 1 |
|---|---|
| -1, 0 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

251, 483, 496

SQ Position 453

| **0**, **0** | 1, 1 |
|---|---|
| -1, 0 | -1, 1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
10

199, 212, 221, 230, 230, 239, 243, 262, 274, 280

SQ Position 454

| **0**, **0** | 1, 1 |
|---|---|
| -1, -1 | -1, 0 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

233, 385, 399

SQ Position 455

| **0**, **0** | 1, 1 |
|---|---|
| -1, -1 | -1, -1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
9

134, 147, 168, 237, 254, 560, 570, 597, 606

SQ Position 456

| **0**, **0** | 1, 1 |
|---|---|
| -1, -1 | -1, 1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
10

242, 243, 246, 650, 656, 672, 672, 675, 677, 719

SQ Position 457

| **0**, **0** | 1, 1 |
|---|---|
| -1, 1 | -1, 0 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539 9

248, 249, 268, 304, 317, 438, 452, 458, 463

SQ Position 458

| **0**, **0** | 1, 1 |
|---|---|
| -1, 1 | -1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.002916 9

271, 272, 285, 682, 683, 687, 693, 707, 718

SQ Position 459

| **0**, **0** | 1, 1 |
|---|---|
| -1, 1 | -1, 1 |

Ord% 1728→0.0032959
Card% 8.748e+06→0.010935 40

256, 257, 260, 261, 262, 266, 275, 276, 277, 281, 283, 287, 289, 359, 365, 410, 416, 502, 506, 542, 546, 579, 583, 612, 616, 640, 644, 663, 667, 677, 680, 684, 685, 686, 689, 702, 714, 720, 722, 722

SQ Position 460

| **0**, **0** | 1, 0 |
|---|---|
| -1, 0 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05 3

12, 13, 14

SQ Position 461

| **0**, **0** | 1, 0 |
|---|---|
| -1, 0 | 1, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243 3

366, 368, 374

SQ Position 462

| **0**, **0** | 1, 0 |
|---|---|
| -1, 0 | 1, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243 3

54, 55, 56

SQ Position 463

| **0**, **0** | 1, 0 |
|---|---|
| -1, -1 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.000243 3

325, 326, 327

SQ Position 464

| **0**, **0** | 1, 0 |
|---|---|
| -1, -1 | 1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539 10

105, 106, 107, 517, 517, 518, 519, 527, 548, 554

SQ Position 465

| **0**, **0** | 1, 0 |
|---|---|
| -1, -1 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648 3

520, 558, 592

SQ Position 466

| **0**, **0** | 1, 0 |
|---|---|
| -1, 1 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

86, 88, 94

SQ Position 467

| **0**, **0** | 1, 0 |
|---|---|
| -1, 1 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

538, 575, 608

SQ Position 468

| **0**, **0** | 1, 0 |
|---|---|
| -1, 1 | 1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

121, 122, 123, 159, 160, 161, 189, 191, 197

SQ Position 469

| **0**, **0** | 1, -1 |
|---|---|
| -1, 0 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

367, 369, 375

SQ Position 470

| **0**, **0** | 1, -1 |
|---|---|
| -1, 0 | 1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

227, 228, 229, 541, 545, 578, 582, 611, 615

SQ Position 471

| **0**, **0** | 1, -1 |
|---|---|
| -1, 0 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

508, 510, 516

SQ Position 472

| **0**, **0** | 1, -1 |
|---|---|
| -1, -1 | 1, 0 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

296, 297, 298, 523, 534, 561, 571, 595, 604

SQ Position 473

| **0**, **0** | 1, -1 |
|---|---|
| -1, -1 | 1, -1 |

Ord% 1728→0.0032959
Card% 8.748e+06→0.010935

85, 87, 93, 188, 190, 196, 281, 283, 289, 417, 419, 425, 429, 430, 431, 475, 476, 477, 519, 524, 529, 531, 535, 542, 546, 557, 562, 566, 568, 572, 579, 583, 589, 591, 591, 596, 601, 605, 612, 616

SQ Position 474

| **0**, **0** | 1, -1 |
|---|---|
| -1, -1 | 1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

465, 467, 473, 526, 537, 564, 574, 598, 607

SQ Position 475

| **0**, **0** | 1, -1 |
|---|---|
| -1, 1 | 1, 0 |

Ord% 192→0.000366211
Card% 518400→0.000648 3

418, 420, 426

SQ Position 476

| **0**, **0** | 1, -1 |
|---|---|
| -1, 1 | 1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.002916 9

282, 284, 290, 551, 553, 586, 588, 619, 621

SQ Position 477

| **0**, **0** | 1, -1 |
|---|---|
| -1, 1 | 1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.002916 12

547, 547, 549, 549, 555, 555, 584, 584, 590, 590, 617, 617

SQ Position 478

| **0**, **0** | 1, 1 |
|---|---|
| -1, 0 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.000243 3

34, 35, 36

SQ Position 479

| **0**, **0** | 1, 1 |
|---|---|
| -1, 0 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648 3

507, 509, 515

SQ Position 480

| **0**, **0** | 1, 1 |
|---|---|
| -1, 0 | 1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539 9

205, 206, 207, 236, 237, 238, 253, 254, 255

SQ Position 481

| **0**, **0** | 1, 1 |
|---|---|
| -1, -1 | 1, 0 |

Ord% 192→0.000366211
Card% 518400→0.000648 3

379, 380, 381

SQ Position 482

| **0**, **0** | 1, 1 |
|---|---|
| -1, -1 | 1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.002916 10

140, 141, 142, 518, 528, 556, 556, 557, 565, 599

SQ Position 483

| **0**, **0** | 1, 1 |
|---|---|
| -1, -1 | 1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.002916 9

521, 522, 525, 559, 560, 563, 593, 594, 597

SQ Position 484

| | |
|---|---|
| **0**, **0** | 1, 1 |
| -1, 1 | 1, 0 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

311, 312, 313, 442, 443, 444, 464, 466, 472

SQ Position 485

| | |
|---|---|
| **0**, **0** | 1, 1 |
| -1, 1 | 1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

539, 540, 550, 576, 577, 585, 609, 610, 618

SQ Position 486

| | |
|---|---|
| **0**, **0** | 1, 1 |
| -1, 1 | 1, 1 |

Ord% 1728→0.0032959
Card% 8.748e+06→0.01093539

340, 341, 342, 393, 394, 395, 486, 487, 488, 527, 528, 529, 530, 532, 533, 536, 543, 544, 548, 552, 554, 565, 566, 567, 569, 570, 573, 580, 581, 587, 589, 599, 600, 602, 603, 606, 613, 614, 620

SQ Position 487

| | |
|---|---|
| **0**, **0** | 0, 0 |
| 1, 0 | 0, 0 |

Ord% 192→0.000366211
Card% 3600→4.5e-06

2

SQ Position 488

| | |
|---|---|
| **0**, **0** | 0, 0 |
| 1, 0 | 0, -1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05

58

SQ Position 489

| | |
|---|---|
| **0**, **0** | 0, 0 |
| 1, 0 | 0, 1 |

Ord% 288→0.000549316
Card% 16200→2.025e-05

23

SQ Position 490

| | |
|---|---|
| **0**, **0** | 0, 0 |
| 1, -1 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05

60

SQ Position 491

| | |
|---|---|
| **0**, **0** | 0, 0 |
| 1, -1 | 0, -1 |

Ord% 672→0.00128174
Card% 102600→0.00012825

57, 85, 86

SQ Position 492

| | |
|---|---|
| **0**, **0** | 0, 0 |
| 1, -1 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05

61

SQ Position 493

| | |
|---|---|
| **0**, **0** | 0, 0 |
| 1, 1 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
2

22, 22

SQ Position 494

| | |
|---|---|
| **0**, **0** | 0, 0 |
| 1, 1 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

59

SQ Position 495

| | |
|---|---|
| **0**, **0** | 0, 0 |
| 1, 1 | 0, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

24, 25, 26

SQ Position 496

| | |
|---|---|
| **0**, **0** | 0, -1 |
| 1, 0 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
1

30

SQ Position 497

| | |
|---|---|
| **0**, **0** | 0, -1 |
| 1, 0 | 0, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

27, 40, 64

SQ Position 498

| | |
|---|---|
| **0**, **0** | 0, -1 |
| 1, 0 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

33

SQ Position 499

| | |
|---|---|
| **0**, **0** | 0, -1 |
| 1, -1 | 0, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

29, 42, 67

SQ Position 500

| | |
|---|---|
| **0**, **0** | 0, -1 |
| 1, -1 | 0, -1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

23, 37, 38, 39, 44, 45, 62, 63, 66, 69, 70, 87, 88

SQ Position 501

| | |
|---|---|
| **0**, **0** | 0, -1 |
| 1, -1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

32, 43, 68

SQ Position 502

| | |
|---|---|
| **0**, **0** | 0, -1 |
| 1, 1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

31

SQ Position 503

| | |
|---|---|
| **0**, **0** | 0, -1 |
| 1, 1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

28, 41, 65

SQ Position 504

| | |
|---|---|
| **0**, **0** | 0, -1 |
| 1, 1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

34, 35, 36

SQ Position 505

| **0**, **0** | 0, 1 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 288→0.000549316
Card% 16200→2.025e-05
2

46, 46

SQ Position 506

| **0**, **0** | 0, 1 |
|---|---|
| 1, 0 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

79

SQ Position 507

| **0**, **0** | 0, 1 |
|---|---|
| 1, 0 | 0, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

49, 52, 72

SQ Position 508

| **0**, **0** | 0, 1 |
|---|---|
| 1, -1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

82

SQ Position 509

| **0**, **0** | 0, 1 |
|---|---|
| 1, -1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

78, 93, 94

SQ Position 510

| **0**, **0** | 0, 1 |
|---|---|
| 1, -1 | 0, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

83, 84, 92

SQ Position 511

| **0**, **0** | 0, 1 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

47, 48, 71

SQ Position 512

| **0**, **0** | 0, 1 |
|---|---|
| 1, 1 | 0, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

80, 81, 91

SQ Position 513

| **0**, **0** | 0, 1 |
|---|---|
| 1, 1 | 0, 1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

50, 51, 53, 54, 55, 56, 73, 74, 75, 76, 77, 89, 90

SQ Position 514

| **0**, **0** | 0, 0 |
|---|---|
| 1, 0 | -1, 0 |

Ord% 128→0.000244141
Card% 9600→1.2e-05
1

11

SQ Position 515

| **0**, **0** | 0, 0 |
|---|---|
| 1, 0 | -1, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

324

SQ Position 516

| **0**, **0** | 0, 0 |
|---|---|
| 1, 0 | -1, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

66

SQ Position 517

| **0**, **0** | 0, 0 |
|---|---|
| 1, -1 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

360

SQ Position 518

| **0**, **0** | 0, 0 |
|---|---|
| 1, -1 | -1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

104, 475, 486

SQ Position 519

| **0**, **0** | 0, 0 |
|---|---|
| 1, -1 | -1, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

498

SQ Position 520

| **0**, **0** | 0, 0 |
|---|---|
| 1, 1 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

51

SQ Position 521

| **0**, **0** | 0, 0 |
|---|---|
| 1, 1 | -1, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

478

SQ Position 522

| **0**, **0** | 0, 0 |
|---|---|
| 1, 1 | -1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

118, 156, 169

SQ Position 523

| **0**, **0** | 0, -1 |
|---|---|
| 1, 0 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

338

SQ Position 524

| **0**, **0** | 0, -1 |
|---|---|
| 1, 0 | -1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

204, 479, 491

SQ Position 525

| **0**, **0** | 0, -1 |
|---|---|
| 1, 0 | -1, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

485

SQ Position 526

| **0**, **0** | 0, -1 |
|---|---|
| 1, -1 | -1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

309, 499, 503

SQ Position 527

| **0**, **0** | 0, -1 |
|---|---|
| 1, -1 | -1, -1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

33, 139, 235, 378, 459, 476, 480, 485, 487, 489, 492, 500, 504

SQ Position 528

| **0**, **0** | 0, -1 |
|---|---|
| 1, -1 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

441, 511, 513

SQ Position 529

| **0**, **0** | 0, -1 |
|---|---|
| 1, 1 | -1, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

391

SQ Position 530

| **0**, **0** | 0, -1 |
|---|---|
| 1, 1 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

251, 483, 496

SQ Position 531

| **0**, **0** | 0, -1 |
|---|---|
| 1, 1 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

507, 509, 515

SQ Position 532

| **0**, **0** | 0, 1 |
|---|---|
| 1, 0 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

79

SQ Position 533

| **0**, **0** | 0, 1 |
|---|---|
| 1, 0 | -1, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
2

474, 474

SQ Position 534

| **0**, **0** | 0, 1 |
|---|---|
| 1, 0 | -1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

222, 263, 275

SQ Position 535

| **0**, **0** | 0, 1 |
|---|---|
| 1, -1 | -1, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

411

SQ Position 536

| **0**, **0** | 0, 1 |
|---|---|
| 1, -1 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

182, 477, 488

SQ Position 537

| **0**, **0** | 0, 1 |
|---|---|
| 1, -1 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

501, 502, 512

SQ Position 538

| **0**, **0** | 0, 1 |
|---|---|
| 1, 1 | -1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

295, 428, 449

SQ Position 539

| **0**, **0** | 0, 1 |
|---|---|
| 1, 1 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

481, 482, 484

SQ Position 540

| **0**, **0** | 0, 1 |
|---|---|
| 1, 1 | -1, 1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
14

349, 400, 490, 493, 493, 494, 495, 497, 505, 506, 508, 510, 514, 516

SQ Position 541

| **0**, **0** | 0, 0 |
|---|---|
| 1, 0 | 1, 0 |

Ord% 448→0.000854492
Card% 22800→2.85e-05
4

5, 5, 6, 6

SQ Position 542

| **0**, **0** | 0, 0 |
|---|---|
| 1, 0 | 1, -1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

217, 256, 258

SQ Position 543

| **0**, **0** | 0, 0 |
|---|---|
| 1, 0 | 1, 1 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

27, 28, 37

SQ Position 544

| **0**, **0** | 0, 0 |
|---|---|
| 1, -1 | 1, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

217, 256, 258

SQ Position 545

| **0**, **0** | 0, 0 |
|---|---|
| 1, -1 | 1, -1 |

Ord% 1568→0.00299072
Card%
649800→0.00081225
10

116, 116, 119, 119, 227, 227, 281, 281, 282, 282

SQ Position 546

| **0**, **0** | 0, 0 |
|---|---|
| 1, -1 | 1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

218, 257, 259

SQ Position 547

| **0**, **0** | 0, 0 |
|---|---|
| 1, 1 | 1, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

27, 28, 37

SQ Position 548

| **0**, **0** | 0, 0 |
|---|---|
| 1, 1 | 1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

218, 257, 259

SQ Position 549

| **0**, **0** | 0, 0 |
|---|---|
| 1, 1 | 1, 1 |

Ord% 1568→0.00299072
Card%
649800→0.00081225
10

98, 98, 99, 99, 133, 133, 134, 134, 143, 143

SQ Position 550

| **0**, **0** | 0, -1 |
|---|---|
| 1, 0 | 1, 0 |

Ord% 672→0.00128174
Card% 102600→0.00012825
3

201, 232, 250

SQ Position 551

| **0**, **0** | 0, -1 |
|---|---|
| 1, 0 | 1, -1 |

Ord% 1568→0.00299072
Card% 649800→0.00081225
10

198, 198, 199, 208, 211, 220, 242, 261, 273, 279

SQ Position 552

| **0**, **0** | 0, -1 |
|---|---|
| 1, 0 | 1, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

204, 235, 251

SQ Position 553

| **0**, **0** | 0, -1 |
|---|---|
| 1, -1 | 1, 0 |

Ord% 1568→0.00299072
Card% 649800→0.00081225
9

200, 213, 223, 231, 244, 248, 264, 271, 276

SQ Position 554

| **0**, **0** | 0, -1 |
|---|---|
| 1, -1 | 1, -1 |

Ord% 4032→0.00769043
Card% 4.617e+06→0.00577125
40

49, 52, 72, 154, 157, 175, 208, 209, 210, 215, 216, 219, 222, 225, 226, 228, 229, 239, 240, 241, 246, 247, 260, 263, 266, 267, 268, 269, 269, 270, 275, 278, 283, 284, 285, 286, 287, 288, 289, 290

SQ Position 555

| **0**, **0** | 0, -1 |
|---|---|
| 1, -1 | 1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539
9

203, 214, 224, 234, 245, 249, 265, 272, 277

SQ Position 556

| **0**, **0** | 0, -1 |
|---|---|
| 1, 1 | 1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

202, 233, 252

SQ Position 557

| **0**, **0** | 0, -1 |
|---|---|
| 1, 1 | 1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539
10

199, 212, 221, 230, 230, 239, 243, 262, 274, 280

SQ Position 558

| **0**, **0** | 0, -1 |
|---|---|
| 1, 1 | 1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539
9

205, 206, 207, 236, 237, 238, 253, 254, 255

SQ Position 559

| **0**, **0** | 0, 1 |
|---|---|
| 1, 0 | 1, 0 |

Ord% 672→0.00128174
Card%
102600→0.00012825
3

49, 52, 72

SQ Position 560

| **0**, **0** | 0, 1 |
|---|---|
| 1, 0 | 1, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

222, 263, 275

SQ Position 561

| **0**, **0** | 0, 1 |
|---|---|
| 1, 0 | 1, 1 |

Ord% 1568→0.00299072
Card%
649800→0.00081225
12

198, 198, 199, 199, 208, 208, 230, 230, 239, 239, 269, 269

SQ Position 562

| **0**, **0** | 0, 1 |
|---|---|
| 1, -1 | 1, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

219, 260, 278

SQ Position 563

| **0**, **0** | 0, 1 |
|---|---|
| 1, -1 | 1, -1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

154, 157, 175, 228, 229, 283, 284, 289, 290

SQ Position 564

| **0**, **0** | 0, 1 |
|---|---|
| 1, -1 | 1, 1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

220, 221, 225, 261, 262, 266, 279, 280, 288

SQ Position 565

| **0**, **0** | 0, 1 |
|---|---|
| 1, 1 | 1, 0 |

Ord% 1568→0.00299072
Card%
649800→0.00081225
9

200, 203, 209, 231, 234, 240, 248, 249, 268

SQ Position 566

| **0**, **0** | 0, 1 |
|---|---|
| 1, 1 | 1, -1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

223, 224, 226, 264, 265, 267, 276, 277, 287

SQ Position 567

| **0**, **0** | 0, 1 |
|---|---|
| 1, 1 | 1, 1 |

Ord% 4032→0.00769043
Card%
4.617e+06→0.00577125
39

201, 202, 204, 205, 206, 207, 210, 211, 212, 213, 214, 215, 216, 232, 233, 235, 236, 237, 238, 241, 242, 243, 244, 245, 246, 247, 250, 251, 252, 253, 254, 255, 270, 271, 272, 273, 274, 285, 286

SQ Position 568

| **0**, **0** | -1, 0 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 128→0.000244141
Card% 9600→1.2e-05
1

9

SQ Position 569

| **0**, **0** | -1, 0 |
|---|---|
| 1, 0 | 0, -1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

339

SQ Position 570

| **0**, **0** | -1, 0 |
|---|---|
| 1, 0 | 0, 1 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

63

SQ Position 571

| **0**, **0** | -1, 0 |
|---|---|
| 1, -1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

363

SQ Position 572

| **0**, **0** | -1, 0 |
|---|---|
| 1, -1 | 0, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

120, 417, 418

SQ Position 573

| **0**, **0** | -1, 0 |
|---|---|
| 1, -1 | 0, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

397

SQ Position 574

| **0**, **0** | -1, 0 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

31

SQ Position 575

| **0**, **0** | -1, 0 |
|---|---|
| 1, 1 | 0, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

396

SQ Position 576

| **0**, **0** | -1, 0 |
|---|---|
| 1, 1 | 0, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

102, 137, 166

SQ Position 577

| **0**, **0** | -1, -1 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

322

SQ Position 578

| **0**, **0** | -1, -1 |
|---|---|
| 1, 0 | 0, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

202, 384, 398

SQ Position 579

| **0**, **0** | -1, -1 |
|---|---|
| 1, 0 | 0, 1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

378

SQ Position 580

| **0**, **0** | -1, -1 |
|---|---|
| 1, -1 | 0, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

293, 386, 401

SQ Position 581

| **0**, **0** | -1, -1 |
|---|---|
| 1, -1 | 0, -1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

53, 158, 252, 382, 383, 388, 389, 392, 400, 403, 404, 419, 420

SQ Position 582

| **0**, **0** | -1, -1 |
|---|---|
| 1, -1 | 0, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

377, 387, 402

SQ Position 583

| **0**, **0** | -1, -1 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
2

376, 376

SQ Position 584

| **0**, **0** | -1, -1 |
|---|---|
| 1, 1 | 0, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

233, 385, 399

SQ Position 585

| **0**, **0** | -1, -1 |
|---|---|
| 1, 1 | 0, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

379, 380, 381

SQ Position 586

| **0**, **0** | -1, 1 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
1

82

SQ Position 587

| **0**, **0** | -1, 1 |
|---|---|
| 1, 0 | 0, -1 |

Ord% 128→0.000244141
Card% 115200→0.000144
1

411

SQ Position 588

| **0**, **0** | -1, 1 |
|---|---|
| 1, 0 | 0, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

219, 260, 278

SQ Position 589

| **0**, **0** | -1, 1 |
|---|---|
| 1, -1 | 0, 0 |

Ord% 128→0.000244141
Card% 115200→0.000144
2

414, 414

SQ Position 590

| **0**, **0** | -1, 1 |
|---|---|
| 1, -1 | 0, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

185, 425, 426

SQ Position 591

| **0**, **0** | -1, 1 |
|---|---|
| 1, -1 | 0, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

415, 416, 424

SQ Position 592

| **0**, **0** | -1, 1 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342
3

310, 390, 405

SQ Position 593

| **0**, **0** | -1, 1 |
|---|---|
| 1, 1 | 0, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

412, 413, 423

SQ Position 594

| **0**, **0** | -1, 1 |
|---|---|
| 1, 1 | 0, 1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

346, 391, 392, 393, 394, 395, 406, 407, 408, 409, 410, 421, 422

SQ Position 595

| **0**, **0** | -1, 0 |
|---|---|
| 1, 0 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-05
3

4, 13, 14

SQ Position 596

| **0**, **0** | -1, 0 |
|---|---|
| 1, 0 | -1, -1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

162, 327, 342

SQ Position 597

| **0**, **0** | -1, 0 |
|---|---|
| 1, 0 | -1, 1 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

62, 87, 88

SQ Position 598

| **0**, **0** | -1, 0 |
|---|---|
| 1, -1 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

164, 374, 375

SQ Position 599

| **0**, **0** | -1, 0 |
|---|---|
| 1, -1 | -1, -1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

97, 107, 123, 188, 189, 519, 529, 554, 555

SQ Position 600

| **0**, **0** | -1, 0 |
|---|---|
| 1, -1 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

165, 568, 608

SQ Position 601

| | |
|---|---|
| **0**, **0** | -1, 0 |
| 1, 1 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

25, 35, 55

SQ Position 602

| | |
|---|---|
| **0**, **0** | -1, 0 |
| 1, 1 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

163, 567, 592

SQ Position 603

| | |
|---|---|
| **0**, **0** | -1, 0 |
| 1, 1 | -1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.00153910

96, 106, 122, 131, 131, 132, 141, 160, 190, 191

SQ Position 604

| | |
|---|---|
| **0**, **0** | -1, -1 |
| 1, 0 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

136, 326, 341

SQ Position 605

| | |
|---|---|
| **0**, **0** | -1, -1 |
| 1, 0 | -1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.00153909

133, 146, 167, 206, 207, 559, 569, 593, 602

SQ Position 606

| | |
|---|---|
| **0**, **0** | -1, -1 |
| 1, 0 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

139, 476, 487

SQ Position 607

| | |
|---|---|
| **0**, **0** | -1, -1 |
| 1, -1 | -1, 0 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.00153909

135, 148, 170, 297, 312, 561, 571, 609, 613

SQ Position 608

| | |
|---|---|
| **0**, **0** | -1, -1 |
| 1, -1 | -1, -1 |

Ord% 1728→0.0032959
Card% 8.748e+06→0.01093539

26, 36, 56, 132, 142, 143, 144, 145, 150, 151, 161, 166, 169, 172, 173, 190, 191, 238, 255, 381, 395, 472, 473, 515, 516, 557, 558, 563, 564, 566, 573, 574, 584, 594, 599, 600, 603, 610, 614

SQ Position 609

| | |
|---|---|
| **0**, **0** | -1, -1 |
| 1, -1 | -1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.00291609

138, 149, 171, 430, 443, 562, 572, 618, 620

SQ Position 610

| | |
|---|---|
| **0**, **0** | -1, -1 |
| 1, 1 | -1, 0 |

Ord% 192→0.000366211
Card% 518400→0.000648 3

137, 380, 394

SQ Position 611

| | |
|---|---|
| **0**, **0** | -1, -1 |
| 1, 1 | -1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.002916 9

134, 147, 168, 237, 254, 560, 570, 597, 606

SQ Position 612

| | |
|---|---|
| **0**, **0** | -1, -1 |
| 1, 1 | -1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.002916 10

140, 141, 142, 518, 528, 556, 556, 557, 565, 599

SQ Position 613

| | |
|---|---|
| **0**, **0** | -1, 1 |
| 1, 0 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.000243 3

78, 93, 94

SQ Position 614

| | |
|---|---|
| **0**, **0** | -1, 1 |
| 1, 0 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648 3

182, 477, 488

SQ Position 615

| | |
|---|---|
| **0**, **0** | -1, 1 |
| 1, 0 | -1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539 9

154, 157, 175, 228, 229, 283, 284, 289, 290

SQ Position 616

| | |
|---|---|
| **0**, **0** | -1, 1 |
| 1, -1 | -1, 0 |

Ord% 192→0.000366211
Card% 518400→0.000648 3

185, 425, 426

SQ Position 617

| | |
|---|---|
| **0**, **0** | -1, 1 |
| 1, -1 | -1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.002916 12

181, 181, 196, 196, 197, 197, 589, 589, 590, 590, 591, 591

SQ Position 618

| | |
|---|---|
| **0**, **0** | -1, 1 |
| 1, -1 | -1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.002916 9

186, 187, 195, 582, 583, 588, 611, 612, 619

SQ Position 619

| **0**, **0** | -1, 1 |
|---|---|
| 1, 1 | -1, 0 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

152, 153, 174, 298, 313, 431, 444, 466, 467

SQ Position 620

| **0**, **0** | -1, 1 |
|---|---|
| 1, 1 | -1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

183, 184, 194, 580, 581, 587, 595, 596, 598

SQ Position 621

| **0**, **0** | -1, 1 |
|---|---|
| 1, 1 | -1, 1 |

Ord% 1728→0.0032959
Card% 8.748e+06→0.01093540

155, 156, 158, 159, 160, 161, 176, 177, 178, 179, 180, 192, 193, 368, 369, 419, 420, 509, 510, 548, 549, 565, 566, 575, 576, 577, 578, 579, 584, 585, 586, 601, 604, 605, 607, 615, 616, 617, 617, 621

SQ Position 622

| **0**, **0** | -1, 0 |
|---|---|
| 1, 0 | 1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-053

16, 17, 20

SQ Position 623

| **0**, **0** | -1, 0 |
|---|---|
| 1, 0 | 1, -1 |

Ord% 288→0.000549316
Card% 194400→0.0002433

358, 359, 371

SQ Position 624

| **0**, **0** | -1, 0 |
|---|---|
| 1, 0 | 1, 1 |

Ord% 288→0.000549316
Card% 194400→0.0002433

64, 65, 69

SQ Position 625

| **0**, **0** | -1, 0 |
|---|---|
| 1, -1 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

364, 365, 373

SQ Position 626

| **0**, **0** | -1, 0 |
|---|---|
| 1, -1 | 1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

128, 129, 130, 541, 542, 545, 546, 551, 553

SQ Position 627

| **0**, **0** | -1, 0 |
|---|---|
| 1, -1 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

643, 644, 648

SQ Position 628

| **0**, **0** | -1, 0 |
|---|---|
| 1, 1 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

40, 41, 44

SQ Position 629

| **0**, **0** | -1, 0 |
|---|---|
| 1, 1 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

639, 640, 646

SQ Position 630

| **0**, **0** | -1, 0 |
|---|---|
| 1, 1 | 1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

110, 111, 114, 146, 147, 150, 167, 168, 172

SQ Position 631

| **0**, **0** | -1, -1 |
|---|---|
| 1, 0 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

330, 331, 334

SQ Position 632

| **0**, **0** | -1, -1 |
|---|---|
| 1, 0 | 1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.00153910

211, 212, 215, 649, 649, 650, 653, 655, 663, 669

SQ Position 633

| **0**, **0** | -1, -1 |
|---|---|
| 1, 0 | 1, 1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

479, 480, 483

SQ Position 634

| **0**, **0** | -1, -1 |
|---|---|
| 1, -1 | 1, 0 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

300, 301, 304, 651, 657, 673, 678, 693, 702

SQ Position 635

| **0**, **0** | -1, -1 |
|---|---|
| 1, -1 | 1, -1 |

Ord% 1728→0.0032959
Card% 8.748e+06→0.01093540

76, 77, 90, 179, 180, 193, 273, 274, 286, 409, 410, 422, 462, 463, 471, 505, 506, 514, 578, 579, 586, 615, 616, 621, 623, 624, 627, 653, 654, 659, 660, 675, 676, 680, 681, 688, 714, 717, 717, 718

SQ Position 636

| **0**, **0** | -1, -1 |
|---|---|
| 1, -1 | 1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

434, 435, 438, 652, 658, 674, 679, 707, 720

SQ Position 637

| **0**, **0** | -1, -1 |
|---|---|
| 1, 1 | 1, 0 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

384, 385, 388

SQ Position 638

| **0**, **0** | -1, -1 |
|---|---|
| 1, 1 | 1, -1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
10

242, 243, 246, 650, 656, 672, 672, 675, 677, 719

SQ Position 639

| **0**, **0** | -1, -1 |
|---|---|
| 1, 1 | 1, 1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
9

521, 522, 525, 559, 560, 563, 593, 594, 597

SQ Position 640

| **0**, **0** | -1, 1 |
|---|---|
| 1, 0 | 1, 0 |

Ord% 288→0.000549316
Card% 194400→0.000243
3

83, 84, 92

SQ Position 641

| **0**, **0** | -1, 1 |
|---|---|
| 1, 0 | 1, -1 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

501, 502, 512

SQ Position 642

| **0**, **0** | -1, 1 |
|---|---|
| 1, 0 | 1, 1 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

220, 221, 225, 261, 262, 266, 279, 280, 288

SQ Position 643

| **0**, **0** | -1, 1 |
|---|---|
| 1, -1 | 1, 0 |

Ord% 192→0.000366211
Card% 518400→0.000648
3

415, 416, 424

SQ Position 644

| **0**, **0** | -1, 1 |
|---|---|
| 1, -1 | 1, -1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
9

186, 187, 195, 582, 583, 588, 611, 612, 619

SQ Position 645

| **0**, **0** | -1, 1 |
|---|---|
| 1, -1 | 1, 1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
12

666, 666, 667, 667, 671, 671, 686, 686, 690, 690, 722, 722

SQ Position 646

| **0**, **0** | -1, 1 |
|---|---|
| 1, 1 | 1, 0 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

315, 316, 317, 447, 448, 452, 457, 458, 469

SQ Position 647

| **0**, **0** | -1, 1 |
|---|---|
| 1, 1 | 1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

664, 665, 670, 684, 685, 689, 704, 716, 721

SQ Position 648

| **0**, **0** | -1, 1 |
|---|---|
| 1, 1 | 1, 1 |

Ord% 1728→0.0032959
Card% 8.748e+06→0.01093539

347, 348, 352, 398, 399, 403, 491, 492, 496, 532, 533, 536, 569, 570, 573, 602, 603, 606, 631, 632, 635, 655, 656, 659, 661, 662, 663, 668, 669, 677, 680, 682, 683, 687, 688, 695, 709, 719, 723

SQ Position 649

| **0**, **0** | 1, 0 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 448→0.000854492
Card% 22800→2.85e-054

7, 7, 10, 10

SQ Position 650

| **0**, **0** | 1, 0 |
|---|---|
| 1, 0 | 0, -1 |

Ord% 672→0.00128174
Card% 102600→0.000128253

292, 323, 328

SQ Position 651

| **0**, **0** | 1, 0 |
|---|---|
| 1, 0 | 0, 1 |

Ord% 672→0.00128174
Card% 102600→0.000128253

29, 32, 38

SQ Position 652

| **0**, **0** | 1, 0 |
|---|---|
| 1, -1 | 0, 0 |

Ord% 672→0.00128174
Card% 102600→0.000128253

308, 336, 354

SQ Position 653

| **0**, **0** | 1, 0 |
|---|---|
| 1, -1 | 0, -1 |

Ord% 1568→0.00299072
Card% 649800→0.000812259

100, 103, 108, 296, 311, 429, 442, 464, 465

SQ Position 654

| **0**, **0** | 1, 0 |
|---|---|
| 1, -1 | 0, 1 |

Ord% 448→0.000854492
Card% 273600→0.0003423

314, 445, 454

SQ Position 655

| **0**, **0** | 1, 0 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 672→0.00128174
Card% 102600→0.000128253

47, 48, 71

SQ Position 656

| **0**, **0** | 1, 0 |
|---|---|
| 1, 1 | 0, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342

299, 433, 446

SQ Position 657

| **0**, **0** | 1, 0 |
|---|---|
| 1, 1 | 0, 1 |

Ord% 1568→0.00299072
Card% 649800→0.000812259

100, 103, 108, 135, 138, 144, 152, 153, 174

SQ Position 658

| **0**, **0** | 1, -1 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 672→0.00128174
Card% 102600→0.000128253

308, 336, 354

SQ Position 659

| **0**, **0** | 1, -1 |
|---|---|
| 1, 0 | 0, -1 |

Ord% 1568→0.00299072
Card% 649800→0.000812259

200, 203, 209, 300, 315, 434, 447, 457, 462

SQ Position 660

| **0**, **0** | 1, -1 |
|---|---|
| 1, 0 | 0, 1 |

Ord% 448→0.000854492
Card% 273600→0.000342

309, 441, 459

SQ Position 661

| **0**, **0** | 1, -1 |
|---|---|
| 1, -1 | 0, 0 |

Ord% 1568→0.00299072
Card% 649800→0.000812259

306, 306, 306, 306, 307, 307, 318, 318, 455, 455, 460, 460

SQ Position 662

| **0**, **0** | 1, -1 |
|---|---|
| 1, -1 | 0, -1 |

Ord% 4032→0.00769043
Card% 4.617e+06→0.00577125

29, 32, 38, 135, 138, 144, 231, 234, 240, 293, 297, 301, 307, 309, 312, 314, 316, 319, 320, 377, 382, 430, 435, 440, 440, 441, 443, 445, 448, 454, 456, 459, 461, 468, 469, 470, 471, 472, 473

SQ Position 663

| **0**, **0** | 1, -1 |
|---|---|
| 1, -1 | 0, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.00153

307, 319, 320, 440, 440, 456, 461, 468, 470

SQ Position 664

| **0**, **0** | 1, -1 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342 3

310, 390, 405

SQ Position 665

| **0**, **0** | 1, -1 |
|---|---|
| 1, 1 | 0, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539 9

248, 249, 268, 304, 317, 438, 452, 458, 463

SQ Position 666

| **0**, **0** | 1, -1 |
|---|---|
| 1, 1 | 0, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539 9

311, 312, 313, 442, 443, 444, 464, 466, 472

SQ Position 667

| **0**, **0** | 1, 1 |
|---|---|
| 1, 0 | 0, 0 |

Ord% 672→0.00128174
Card% 102600→0.00012825 3

47, 48, 71

SQ Position 668

| **0**, **0** | 1, 1 |
|---|---|
| 1, 0 | 0, -1 |

Ord% 448→0.000854492
Card% 273600→0.000342 3

295, 428, 449

SQ Position 669

| **0**, **0** | 1, 1 |
|---|---|
| 1, 0 | 0, 1 |

Ord% 1568→0.00299072
Card% 649800→0.00081225 9

200, 203, 209, 231, 234, 240, 248, 249, 268

SQ Position 670

| **0**, **0** | 1, 1 |
|---|---|
| 1, -1 | 0, 0 |

Ord% 448→0.000854492
Card% 273600→0.000342 3

310, 390, 405

SQ Position 671

| **0**, **0** | 1, 1 |
|---|---|
| 1, -1 | 0, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539 9

152, 153, 174, 298, 313, 431, 444, 466, 467

SQ Position 672

| **0**, **0** | 1, 1 |
|---|---|
| 1, -1 | 0, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.001539 9

315, 316, 317, 447, 448, 452, 457, 458, 469

SQ Position 673

| **0**, **0** | 1, 1 |
|---|---|
| 1, 1 | 0, 0 |

Ord% 1568→0.00299072
Card% 649800→0.0008122516

291, 291, 291, 291, 294, 294, 294, 294, 427, 427, 427, 427, 432, 432, 432, 432

SQ Position 674

| **0**, **0** | 1, 1 |
|---|---|
| 1, 1 | 0, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

302, 303, 305, 436, 437, 439, 450, 451, 453

SQ Position 675

| **0**, **0** | 1, 1 |
|---|---|
| 1, 1 | 0, 1 |

Ord% 4032→0.00769043
Card% 4.617e+06→0.0057712539

292, 293, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 323, 328, 377, 382, 428, 429, 430, 431, 433, 434, 435, 436, 437, 438, 439, 446, 449, 450, 451, 453, 462, 463, 465, 467, 471, 473

SQ Position 676

| **0**, **0** | 1, 0 |
|---|---|
| 1, 0 | -1, 0 |

Ord% 192→0.000366211
Card% 43200→5.4e-053

18, 19, 21

SQ Position 677

| **0**, **0** | 1, 0 |
|---|---|
| 1, 0 | -1, -1 |

Ord% 288→0.000549316
Card% 194400→0.0002433

332, 333, 335

SQ Position 678

| **0**, **0** | 1, 0 |
|---|---|
| 1, 0 | -1, 1 |

Ord% 288→0.000549316
Card% 194400→0.0002433

67, 68, 70

SQ Position 679

| **0**, **0** | 1, 0 |
|---|---|
| 1, -1 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

361, 362, 372

SQ Position 680

| **0**, **0** | 1, 0 |
|---|---|
| 1, -1 | -1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

112, 113, 115, 523, 524, 526, 543, 544, 552

SQ Position 681

| **0**, **0** | 1, 0 |
|---|---|
| 1, -1 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

641, 642, 647

SQ Position 682

| **0**, **0** | 1, 0 |
|---|---|
| 1, 1 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

74, 75, 89

SQ Position 683

| **0**, **0** | 1, 0 |
|---|---|
| 1, 1 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

625, 626, 628

SQ Position 684

| **0**, **0** | 1, 0 |
|---|---|
| 1, 1 | -1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

125, 126, 127, 170, 171, 173, 177, 178, 192

SQ Position 685

| **0**, **0** | 1, -1 |
|---|---|
| 1, 0 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

356, 357, 370

SQ Position 686

| **0**, **0** | 1, -1 |
|---|---|
| 1, 0 | -1, -1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

213, 214, 216, 651, 652, 654, 661, 662, 668

SQ Position 687

| **0**, **0** | 1, -1 |
|---|---|
| 1, 0 | -1, 1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

499, 500, 511

SQ Position 688

| **0**, **0** | 1, -1 |
|---|---|
| 1, -1 | -1, 0 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

318, 319, 320, 699, 699, 700, 701, 703, 705

SQ Position 689

| **0**, **0** | 1, -1 |
|---|---|
| 1, -1 | -1, -1 |

Ord% 1728→0.0032959
Card% 8.748e+06→0.0109353

42, 43, 45, 148, 149, 151, 244, 245, 247, 386, 387, 389, 460, 461, 470, 503, 504, 513, 561, 562, 564, 613, 614, 620, 637, 638, 645, 673, 674, 676, 695, 700, 701, 709, 712, 713, 713, 723, 726

SQ Position 690

| **0**, **0** | 1, -1 |
|---|---|
| 1, -1 | -1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

455, 456, 468, 703, 705, 712, 715, 715, 726

SQ Position 691

| **0**, **0** | 1, -1 |
|---|---|
| 1, 1 | -1, 0 |

Ord% 192→0.000366211
Card% 518400→0.0006483

407, 408, 421

SQ Position 692

| **0**, **0** | 1, -1 |
|---|---|
| 1, 1 | -1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

271, 272, 285, 682, 683, 687, 693, 707, 718

SQ Position 693

| **0**, **0** | 1, -1 |
|---|---|
| 1, 1 | -1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

539, 540, 550, 576, 577, 585, 609, 610, 618

SQ Position 694

| **0**, **0** | 1, 1 |
|---|---|
| 1, 0 | -1, 0 |

Ord% 288→0.000549316
Card% 194400→0.0002433

80, 81, 91

SQ Position 695

| **0**, **0** | 1, 1 |
|---|---|
| 1, 0 | -1, -1 |

Ord% 192→0.000366211
Card% 518400→0.0006483

481, 482, 484

SQ Position 696

| **0**, **0** | 1, 1 |
|---|---|
| 1, 0 | -1, 1 |

Ord% 672→0.00128174
Card% 1.2312e+06→0.0015399

223, 224, 226, 264, 265, 267, 276, 277, 287

SQ Position 697

| **0**, **0** | 1, 1 |
|---|---|
| 1, -1 | -1, 0 |

Ord% 192→0.000366211
Card% 518400→0.0006483

412, 413, 423

SQ Position 698

| **0**, **0** | 1, 1 |
|---|---|
| 1, -1 | -1, -1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

183, 184, 194, 580, 581, 587, 595, 596, 598

SQ Position 699

| **0**, **0** | 1, 1 |
|---|---|
| 1, -1 | -1, 1 |

Ord% 288→0.000549316
Card% 2.3328e+06→0.0029169

664, 665, 670, 684, 685, 689, 704, 716, 721

SQ Position 700

| **0**, **0** | 1, 1 |
|---|---|
| 1, 1 | -1, 0 |

Ord% 672→0.00128174
Card%
1.2312e+06→0.001539
9

302, 303, 305, 436, 437, 439, 450, 451, 453

SQ Position 701

| **0**, **0** | 1, 1 |
|---|---|
| 1, 1 | -1, -1 |

Ord% 288→0.000549316
Card%
2.3328e+06→0.002916
12

691, 691, 692, 692, 694, 694, 706, 706, 708, 708, 724, 724

SQ Position 702

| **0**, **0** | 1, 1 |
|---|---|
| 1, 1 | -1, 1 |

Ord% 1728→0.0032959
Card%
8.748e+06→0.010935
42

350, 351, 353, 401, 402, 404, 494, 495, 497, 534, 535, 537, 571, 572, 574, 604, 605, 607, 633, 634, 636, 657, 658, 660, 678, 679, 681, 696, 696, 697, 697, 698, 698, 702, 710, 710, 711, 711, 714, 720, 725, 725

SQ Position 703

| **0**, **0** | 1, 0 |
|---|---|
| 1, 0 | 1, 0 |

Ord% 1152→0.00219727
Card% 162000→0.0002025
13

8, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21

SQ Position 704

| **0**, **0** | 1, 0 |
|---|---|
| 1, 0 | 1, -1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
14

344, 344, 345, 349, 350, 351, 353, 363, 364, 365, 367, 369, 373, 375

SQ Position 705

| **0**, **0** | 1, 0 |
|---|---|
| 1, 0 | 1, 1 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

30, 31, 33, 34, 35, 36, 39, 40, 41, 42, 43, 44, 45

SQ Position 706

| **0**, **0** | 1, 0 |
|---|---|
| 1, -1 | 1, 0 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

337, 338, 339, 340, 341, 342, 355, 356, 357, 358, 359, 370, 371

SQ Position 707

| **0**, **0** | 1, 0 |
|---|---|
| 1, -1 | 1, -1 |

Ord% 4032→0.00769043
Card%
4.617e+06→0.00577125
40

117, 118, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 366, 367, 417, 418, 507, 508, 527, 528, 529, 531, 534, 535, 537, 538, 539, 540, 541, 542, 545, 546, 547, 547, 549, 550, 551, 553, 555

SQ Position 708

| **0**, **0** | 1, 0 |
|---|---|
| 1, -1 | 1, 1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

343, 396, 489, 530, 567, 600, 629, 637, 638, 639, 640, 645, 646

SQ Position 709

| **0**, **0** | 1, 0 |
|---|---|
| 1, 1 | 1, 0 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

58, 59, 63, 64, 65, 69, 79, 80, 81, 85, 87, 91, 93

SQ Position 710

| **0**, **0** | 1, 0 |
|---|---|
| 1, 1 | 1, -1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
14

345, 397, 490, 531, 568, 601, 630, 630, 633, 634, 636, 643, 644, 648

SQ Position 711

| **0**, **0** | 1, 0 |
|---|---|
| 1, 1 | 1, 1 |

Ord% 4032→0.00769043
Card%
4.617e+06→0.00577125
39

101, 102, 104, 105, 106, 107, 109, 110, 111, 112, 113, 114, 115, 136, 137, 139, 140, 141, 142, 145, 146, 147, 148, 149, 150, 151, 162, 163, 166, 167, 168, 172, 182, 183, 184, 188, 190, 194, 196

SQ Position 712

| **0**, **0** | 1, -1 |
|---|---|
| 1, 0 | 1, 0 |

Ord% 1728→0.0032959

Card% 729000→0.00091125

13

337, 343, 346, 347, 348, 352, 360, 361, 362, 366, 368, 372, 374

SQ Position 713

| **0**, **0** | 1, -1 |
|---|---|
| 1, 0 | 1, -1 |

Ord% 4032→0.00769043

Card% 4.617e+06→0.00577125

40

217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 358, 364, 409, 415, 501, 505, 541, 545, 578, 582, 611, 615, 639, 643, 655, 656, 657, 658, 659, 660, 664, 665, 666, 666, 667, 670, 671

SQ Position 714

| **0**, **0** | 1, -1 |
|---|---|
| 1, 0 | 1, 1 |

Ord% 1152→0.00219727

Card% 1.944e+06→0.00243

13

338, 391, 485, 489, 491, 492, 496, 503, 504, 507, 509, 513, 515

SQ Position 716

| **0**, **0** | 1, -1 |
|---|---|
| 1, -1 | 1, -1 |

Ord% 10368→0.0197754

Card%

3.2805e+07→0.0410063

172

60, 61, 66, 67, 68, 70, 82, 83, 84, 86, 88, 92, 94, 164, 165, 169, 170, 171, 173, 185, 186, 187, 189, 191, 195, 197, 258, 259, 263, 264, 265, 267, 278, 279, 280, 282, 284, 288, 290, 354, 355, 360, 363, 370, 371, 372, 373, 374, 375, 397, 400, 401, 402, 404, 405, 406, 411, 414, 414, 415, 416, 418, 420, 421, 422, 423, 424, 424, 425, 426, 426, 441, 442, 443, 444, 454, 455, 456, 457, 458, 468, 469, 485, 486, 487, 488, 498, 498, 499, 500, 501, 502, 511, 511, 512, 512, 513, 514, 515, 516, 538, 550, 551, 552, 553, 554, 555, 568, 571, 572, 574, 575, 582, 583, 584, 585, 586, 587, 588, 588, 589, 590, 590, 599, 608, 608, 609, 610, 611, 612, 618, 618, 619, 619, 620, 621, 629, 631, 632, 635, 641, 642, 645, 646, 647, 647, 648, 668, 669, 670, 671, 678, 679, 681, 686, 687, 688, 689, 690, 690, 703, 704, 705, 712, 715, 715, 716, 716, 719, 721, 723, 726

SQ Position 715

| **0**, **0** | 1, -1 |
|---|---|
| 1, -1 | 1, 0 |

Ord% 4032→0.00769043

Card%

4.617e+06→0.00577125

39

308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 356, 361, 407, 412, 499, 503, 539, 543, 576, 580, 609, 613, 637, 641, 661, 664, 682, 684, 695, 699, 699, 700, 701, 703, 704, 705

SQ Position 717

| **0**, **0** | 1, -1 |
|---|---|
| 1, -1 | 1, 1 |

Ord% 1728→0.0032959

Card%

8.748e+06→0.010935

39

336, 357, 362, 390, 408, 413, 445, 447, 448, 452, 459, 460, 461, 464, 466, 470, 472, 500, 504, 540, 544, 577, 581, 610, 614, 638, 642, 662, 665, 683, 685, 700, 701, 709, 712, 713, 713, 721, 726

SQ Position 718

| **0**, **0** | 1, -1 |
|---|---|
| 1, 1 | 1, 0 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
13

339, 392, 396, 398, 399, 403, 411, 412, 413, 417, 419, 423, 425

SQ Position 719

| **0**, **0** | 1, -1 |
|---|---|
| 1, 1 | 1, -1 |

Ord% 1728→0.0032959
Card%
8.748e+06→0.010935
40

256, 257, 260, 261, 262, 266, 275, 276, 277, 281, 283, 287, 289, 359, 365, 410, 416, 502, 506, 542, 546, 579, 583, 612, 616, 640, 644, 663, 667, 677, 680, 684, 685, 686, 689, 702, 714, 720, 722, 722

SQ Position 720

| **0**, **0** | 1, -1 |
|---|---|
| 1, 1 | 1, 1 |

Ord% 1728→0.0032959
Card%
8.748e+06→0.010935
39

340, 341, 342, 393, 394, 395, 486, 487, 488, 527, 528, 529, 530, 532, 533, 536, 543, 544, 548, 552, 554, 565, 566, 567, 569, 570, 573, 580, 581, 587, 589, 599, 600, 602, 603, 606, 613, 614, 620

SQ Position 721

| **0**, **0** | 1, 1 |
|---|---|
| 1, 0 | 1, 0 |

Ord% 1728→0.0032959
Card%
729000→0.00091125
13

50, 51, 53, 54, 55, 56, 73, 74, 75, 76, 77, 89, 90

SQ Position 722

| **0**, **0** | 1, 1 |
|---|---|
| 1, 0 | 1, -1 |

Ord% 1152→0.00219727
Card%
1.944e+06→0.00243
14

349, 400, 490, 493, 493, 494, 495, 497, 505, 506, 508, 510, 514, 516

SQ Position 723

| **0**, **0** | 1, 1 |
|---|---|
| 1, 0 | 1, 1 |

Ord% 4032→0.00769043
Card%
4.617e+06→0.00577125
39

201, 202, 204, 205, 206, 207, 210, 211, 212, 213, 214, 215, 216, 232, 233, 235, 236, 237, 238, 241, 242, 243, 244, 245, 246, 247, 250, 251, 252, 253, 254, 255, 270, 271, 272, 273, 274, 285, 286

SQ Position 724

| **0**, **0** | 1, 1 |
|---|---|
| 1, -1 | 1, 0 |

Ord% 1152→0.00219727

Card% 1.944e+06→0.00243

13

346, 391, 392, 393, 394, 395, 406, 407, 408, 409, 410, 421, 422

SQ Position 725

| **0**, **0** | 1, 1 |
|---|---|
| 1, -1 | 1, -1 |

Ord% 1728→0.0032959

Card% 8.748e+06→0.010935

40

155, 156, 158, 159, 160, 161, 176, 177, 178, 179, 180, 192, 193, 368, 369, 419, 420, 509, 510, 548, 549, 565, 566, 575, 576, 577, 578, 579, 584, 585, 586, 601, 604, 605, 607, 615, 616, 617, 617, 621

SQ Position 726

| **0**, **0** | 1, 1 |
|---|---|
| 1, -1 | 1, 1 |

Ord% 1728→0.0032959

Card% 8.748e+06→0.010935

39

347, 348, 352, 398, 399, 403, 491, 492, 496, 532, 533, 536, 569, 570, 573, 602, 603, 606, 631, 632, 635, 655, 656, 659, 661, 662, 663, 668, 669, 677, 680, 682, 683, 687, 688, 695, 709, 719, 723

SQ Position 727

| **0**, **0** | 1, 1 |
|---|---|
| 1, 1 | 1, 0 |

Ord% 4032→0.00769043
Card%
4.617e+06→0.00577125
39

292, 293, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 323, 328, 377, 382, 428, 429, 430, 431, 433, 434, 435, 436, 437, 438, 439, 446, 449, 450, 451, 453, 462, 463, 465, 467, 471, 473

SQ Position 728

| **0**, **0** | 1, 1 |
|---|---|
| 1, 1 | 1, -1 |

Ord% 1728→0.0032959
Card%
8.748e+06→0.010935
42

350, 351, 353, 401, 402, 404, 494, 495, 497, 534, 535, 537, 571, 572, 574, 604, 605, 607, 633, 634, 636, 657, 658, 660, 678, 679, 681, 696, 696, 697, 697, 698, 698, 702, 710, 710, 711, 711, 714, 720, 725, 725

SQ Position 729

| **0**, **0** | 1, 1 |
|---|---|
| 1, 1 | 1, 1 |

Ord% 10368→0.0197754
Card%
3.2805e+07→0.0410063
182

321, 321, 322, 322, 324, 324, 325, 325, 326, 326, 327, 327, 329, 329, 330, 330, 331, 331, 332, 332, 333, 333, 334, 334, 335, 335, 376, 376, 378, 378, 379, 379, 380, 380, 381, 381, 383, 383, 384, 384, 385, 385, 386, 386, 387, 387, 388, 388, 389, 389, 474, 474, 475, 475, 476, 476, 477, 477, 478, 478, 479, 479, 480, 480, 481, 481, 482, 482, 483, 483, 484, 484, 517, 517, 518, 518, 519, 519, 520, 520, 521, 521, 522, 522, 523, 523, 524, 524, 525, 525, 526, 526, 556, 556, 557, 557, 558, 558, 559, 559, 560, 560, 561, 561, 562, 562, 563, 563, 564, 564, 591, 591, 592, 592, 593, 593, 594, 594, 595, 595, 596, 596, 597, 597, 598, 598, 622, 622, 623, 623, 624, 624, 625, 625, 626, 626, 627, 627, 628, 628, 649, 649, 650, 650, 651, 651, 652, 652, 653, 653, 654, 654, 672, 672, 673, 673, 674, 674, 675, 675, 676, 676, 691, 691, 692, 692, 693, 693, 694, 694, 706, 706, 707, 707, 708, 708, 717, 717, 718, 718, 724, 724

# Appendix E

# SQ and NC Simulation Program Code

## E.1 Introduction

Chapter 4 contained a very high-level description of the underlying program that drives the simulations used in this project. This appendix is devoted to a very low level description of this code. It amounts to a walk-through of the code with highlights of pieces that have particular philosophical significance. The process for this will be to present a block of code and then a short synopsis of what it does and how. Code blocks will be presented in approximately the order that they are encountered when the simulation is actually run. Before this granular presentation takes place a mid-level review of the entire simulation is presented so that the detailed presentation that forms the bulk of this chapter might be more intelligible.[1]

## E.2 Mid-Level Agent Description

As mentioned earlier, there are two agent classes, a class of agent that makes choices in the way that can be considered rational from the perspective of classical economic theory (NC agents) and a class of agent that makes decisions

[1]Please note that since creating the extensive annotations in this appendix that some additional small changes have been made. The full original code used to produce all the simulations used in this thesis and for much of the accompanying analysis is available directly from the University of Alberta Libraries electronic thesis collection, on the CD accompanying the hard copies of this thesis, and by contacting the author, John Simpson, directly via either john.simpson@ualberta.ca or jsympson@gmail.com.

with a procedure more akin to a heuristic based on the framing of a status quo position (SQ agents). For the most part these agent classes contain the same code, but there are cases where there are differences, both subtle and significant. In order to keep this section as compact as possible I am going to adopt the following convention for reporting information about the construction of these agent classes: when reference is made to the composition of an agent and a single number is reported that number is to be understood as applying to both classes; in cases where numbers are presented with a vertical bar dividing them, as in 4|5, the number before the bar is meant to apply to the NC agent class and the number after to the SQ agent class.

It will also be helpful to note that names referring to functions and variables used in running the simulation series will be referred to with italics and camel-case capitalization.[2] This choice has been made so that the `choice` might be easily distinguished from the act of making a choice and to preserve the exact capitalization used in the naming to reflect the case sensitivity of C++. This convention will be maintained even in section headings and when the name is the first word of a sentence.

Agents are made up of 16 different data members which are accessed and manipulated through 28|27 member functions. Some of these data items are quite tiny and relatively insignificant to the behaviour of any agent and are used simply to track changes within the composition of an agent for data reporting or for debugging purposes. One example of this is the variable bitsSet, which simply tracks how many bits are set to 1 or "on" in the bitset choice, which controls an agent's decision making behaviour when playing a game with another agent. Most data items have a direct impact on agent behaviour though. Three of these are of particularly high importance and may be considered to be the core of the agent class. In order of descending importance these important structures with each agent are the bitset choice, the array afterPlay, and the map know.

[2]The camel case convention for naming removes the spaces between multiple words that form the name, if any, and capitalizes the first letter of every word except the first. So, a name like "University of Alberta" would become universityOfAlberta in camel case. Camel case is often used in programming to strike a compromise across the need for meaningful variable names, the need for readability, and the inability to have spaces in names.

### choice

choice is a bitset coding for an agent's behaviour in any possible scenario that they could possibly face within the simulation environment. The total number of bits within choice is 12717|6561.[3] In SQ agents these bits code for whether or not the agent will play the game with the choice that has the possibility of bringing about the status quo outcome or change to the other option. If a bit is set to 1 or "on" then an SQ agent stays with the action currently associated with the status quo, otherwise they switch to the alternative action. In NC agents the bits within choice code for which choice an agent will make when presented with a game that can be thought of in normal form. The choices available to an NC class agent are to play 'top' or 'bottom' if they are assigned to be the row player or 'left' or 'right' if assigned to be the column player. If the bit corresponding to the combination of the position the agent has been assigned to play within the game and the relationship status the agent assigns to the other player is set to 1 or "on" then the agent will make the choice that corresponds to 'top' in the standard presentation of the position they have been assigned to play in that game.[4] The behaviour of an agent when given a game and another agent to play with is entirely determined by the composition of choice.

Implementing the behaviour of agents within a bitset coding for all the possible variations in conditions that an agent might face makes it very easy to allow for discontinuities in the behaviour of agents. All that needs to happen for this to occur is for a single bit to be inverted within a set of bits related by whatever features that are of interest and suddenly the agent is no longer exhibiting continuity in their behaviour. The entire procedure of making a choice is also relatively simple to understand and track, the entire process being analogous to looking up a location on a map after being given the full set of co-ordinates. This simplicity also adds speed to the program. Bitsets are relatively small compared to other data types and are composed of the most fundamental information type within modern digital computers, the bit.

[3] See previous section for a description of this numbering convention.

[4] I needed a way to simplify determining how an NC class agent is to play games. Since NC class agents are immune to framing effects, they are "fully" rational, that is, they see all possible presentations of a game in the same way. Exactly which presentation is seen is irrelevant, all that matters being that they are able to recognize a game and their assigned position within that game. By "the standard presentation of the position" I simply mean to refer to the presentation that I arbitrarily choose to act as the standard within this series of simulations.

This makes operations on the bitset, including searches and reading/writing, extremely fast in comparison to maintaining a mathematical expression.

One possible criticism with this approach is that it is too rigidly deterministic or pure-strategy focused insofar as the agents always make the same choices under the same conditions, a departure from actual human behaviour. It is commonly the case that people make different choices under the same conditions. In principle this is an issue that can be easily overcome by swapping the bitset for a list of floating point values, each of which would correspond to a described scenario and list the probability of one of two choices being made. Unfortunately this option was not feasible given the machines on which the simulation was to be run, since floating point values are much larger and more complicated than bits. Making this change would have increased the memory footprint left by each agent by least eight times and significantly slowed the simulation, more than doubling computation time.

### `afterPlay`

Agent updating is done via the data member known as `afterPlay`, a $3\times10\times10|3\times3\times3$ array used to determine how an agent adjusts their perceptions of the relationship they have with another agent after playing a game with that agent. `afterPlay` is used twice by an agent after a game is played, once to update the way in which the agent frames their relationship with the agent just played with as it extends from the updating agent to the other agent, and once to update the way in which the agent sees the other agent as extending the relationship to them. By dividing the relationship into two parts and using the same procedure to update both halves of this relationship agents "simulate" being the other agent as best they can from their limited perspective.

It is important to note that `afterPlay` only updates the way that agents see themselves as relating to other agents and not how they will behave if given the opportunity to choose again under identical circumstances. Relationships are thus flexible across the existence of an agent within a simulation while their contextual based behaviours are not. While this is a standard approach used in updating agents (Skyrms (2005), Axelrod (1984), Epstein (2006), Epstein and Axtell (1996)) it can be criticized for not adequately reflecting the full spectrum of possible ways that human beings learn. It is my hope that this criticism can be defused by noting that learning of the sort that seems to be

| | |
|---|---|
| 7, 3 | 5, 3 |
| 3, 4 | 1, 1 |

Figure E.1: Example of NC Type Agent Decision Procedure

lacking is actually taking place through the genetic algorithm that governs the life of each population.[5]

The updating procedure that uses `afterPlay` requires three inputs, the way in which the relationship to be updated was framed before playing the game, an assessment of how the agent which that relationship is extending from did in the game, and an assessment of how the agent which the relationship is extending to did.[6] Differences in how the assessment of how well each agent did in the game are the source of the the differences in the dimension sizes between the two agent classes.

NC class agents assess how well an agent did by checking to see how many payoffs available to that player within the game were equivalent to each other and then assessing how many of the payoffs available to that player were greater than what they received. Setting aside the detailed mechanics of the choice procedure until the second part of this appendix, consider the following example. A, the row player, and B, the column player, play the game described in Figure E.1 with the final outcome being <Top, Right> with payoffs of 5 to A and 3 to B.

The updating procedure of A might happen as follows. First, A evaluates and makes changes to the relationship that it assesses as standing between itself and B as it extends from itself to B. Suppose that this relationship was assessed to be of type '2' before playing the game. Next, A considers how it did with respect to what else was available. A received a payoff of 2 against a payoff distribution with no equivalencies and a total of one payoff greater than what they received. This state of affairs is indexed as type '2'. Finally, A considers how B did with respect to what was available to B. B received a payoff of 3 against a payoff distribution with one pair of equivalent payoffs and only one available payoff that was better. This state of affairs is indexed as type '5'. With this information A updates how they extend the relationship

[5]Going forward, an investigation of the differences in agent-based models making use of various combinations of updating procedures would be a valuable contribution to the community.

[6]If Jill likes Jack then the relationship of liking extends from Jill to Jack.

from themselves to B by looking in afterPlay at location 2×1×5 and assigning the relationship type found at this location to be the new way that they extend their relationship to B.

To update how A perceives B extending the relationship to them A uses the same afterPlay matrix but changes the inputs to reflect B's perspective. The first input thus represents the way that A perceived B as extending the relationship to them prior to playing the game—perhaps this was a '1'. The second coordinate input now becomes how B fared in the game, which was found to be of type '5', and the final input becomes how A fared in the game, which was found to be of type '1'. A thus updates how they perceive B extending the relationship to them by looking at location 1×5×1 within afterPlay and assigning the relationship type found there to the extension of the relationship from B to them.

Relationship updating within SQ agents follows the same framework with the exception that assessments about how either agent fared in the game are made with respect to how the outcome they actually received compares to the status quo: greater than the status quo, equivalent to the status quo, or less than the status quo. The results of each comparison is assigned a number $< 0, 1, 2 >$, which maps directly onto <'<', '=', '>'>, to be used as an index for afterPlay. The agent's liking of the other agent prior to playing the game is also assigned a number [0, 2] and this serves as the third index value for afterPlay.

For example, if A and B are SQ class agents playing the game in figure 1 with outcome <T, L> and status quo <T, R> then A's updating procedure would proceed as follows. A extends the relationship to B with type '2'as before. A received a payoff of 5 which is worse than the status quo outcome of 7 and so the second index to afterPlay is '0'. B has performed just as well as they would have at the status quo and so their performance is indexed as a '1'. A thus updates how they extend the relationship to B by looking at location 2×0×1 within afterPlay and assigning the relationship type found there.

Despite its apparent complexity `afterPlay` does provide a mechanism for updating agents that is simple to understand and read. Despite the simplicity of the method for updating agents after they play a game, the sheer size of the matrix in NC class agents does make it more difficult to examine the structure of `afterPlay` both for individual agents and across the entire population. This is unfortunate because it makes it more difficult to glance at the matrix and

discover patterns. Still, the updating procedure of agents using `afterPlay` amounts to looking up values in a table which gives it a speed similar to that produced by using a bitset. As a matrix `afterPlay` also brings with it the benefit of being able to modify discontinuities in the updating procedures of individual agents.

### `know`

`know` is an instance of the C++ data item called a map; it returns the data item with the key or index value matching a submitted value, if any. It is in this map that each agent holds a record of the agents that they have interacted with in the past, consisting of: the ID of an agent (the key), the number of rounds since that agent has been seen, the way that the agent owning `know` likes that agent, and the way that owning agents perceives the other agent in question as liking them. From an implementation perspective `know` has been the most problematic. Not only do maps have problematic idiosyncracies not shared by other members of C++ Standard Library (e.g. automatically creating entries for inputs that are looked up and do not currently exist), but they also represent the majority of an agent's memory requirements in any population with long-living agents. The more an agent interacts with other agents the more agents they know and thus the more information that is stored in `know`. Over the course of the lifetime of a population it is entirely possible for there to have been hundreds of thousands of agents that have come and gone, each of which will have interacted with many other agents. If there are 10,000 agents currently alive and each has interacted with 10,000 different agents on average over the course of the simulation then it is entirely possible that upwards of 4.8 Gb of RAM will be needed to store just the `know` components of all the agents. These memory limitations can be overcome in a number of ways, but each amounts to taking a stance regarding an epistemology of agent-knowledge; of course, so does not doing anything at all.

As with many computer models, the final decision on how to overcome the possibility of ever expanding memory needs was influenced heavily by pragmatic considerations. All simulations are given a maximum population size of 15,000 members and each agent within that population has the same capacity to forget other agents. In addition to pragmatic necessity, cutting the short simulations with large populations is easily defensible on the argument that

the populations that typically reach such a size typically exhibit exponential growth with no sign of slowing down and so, from the perspective of having a set of behaviours that can sustain the agents within the population, can be considered "stable". Forgetting other agents is done in two ways. First, all agents who are removed from the population at the end of a generation are removed from the memories of all remaining agents. Second, if an agent does not interact with another agent for 100 generations then the entry corresponding to the unseen agent is removed from `know`; however, since populations were limited in length to a maximum of 100 generations this limit is of no consequence in the results reported here.[7]

`know` is both simple to understand and implement and, providing that there are constraints placed on its growth or final size, capable of producing a model with reasonable execution speed. It is also an important component of creating an agent-based model that adequately represents actual human decision makers since people clearly do remember who they have interacted with.

## E.3 Program Framework

Despite being made up of over ten thousand lines of code, the program is really relatively simple.[8] The program is summarized as follows:

C++ Library Inclusions
Function Declarations
Main Function
{
    Major Variable Declarations
    Set-up Output Files
    Trials Loop

[7]The predicted consequence of constricting the threshold for forgetting is that as the size of the population increases more and more emphasis will be placed on how agents interact with "new" agents. This is not necessarily a bad thing since it seems reasonable to assume that actual human beings behave in a similar way. One future extension of the investigations carried out with this model will be to vary the threshold of forgetfulness and observe how this affects the dynamics of the relationships within the population.

[8]10,000+ plus lines may seem like a lot, but it is actually rather small for a project of this nature, due in part to efficient programming techniques provided by the C++ programming language and also to the lack of graphical output. The Sugarscape model of Epstein and Axtell Epstein and Axtell (1996) comes in at over 20,000 lines.

```
{
    Population Loop and beginning of multithread processing
    {
        Environment Loop
        {
            Build Initial population for environment
            Collect Data on Initial Population
            Generation Loop
            {
                Round Loop
                {
                    Build Games
                    Match Agents and Games
                    Play Games
                    Update Agents
                }
                Remove Poorly Performing Agents
                Spawning of Remaining Agents
            }
            Collect Generation Data
        }
    }
}
```

With the above outline providing a general guide to the programs making up this simulation the actual code used to produce the simulation will now be presented and described. Throughout this description it is important to recognize that there are two distinct classes of agents. While there is a great deal of similarity between these two classes there are also important differences. As the code used to produce the simulations is presented the following presentations of conventions should be understood. When a particular block of code is identical for both agent classes then the code will be presented on the left with a description/explanation on the right. In those cases where a block of code is different across the two agent classes the code for the two agent classes will be presented side by side with the description placed immediately below. Code for NC class agents will always be presented against a light grey

background and be placed to the left of SQ class agent code. In those cases where one agent class has a piece of code for which there is no analogous code in the other agent class then the side-by-side approach will still be used with code being filled in for only the relevant agent class. A Monaco font will be used on code to help distinguish it from the accompanying descriptions. Code will also be coloured using the conventions that are customary in many C++ editors.

### C++ Library Inclusions

The following libraries are used to run the simulation:

```
#include<iostream>
#include<iomanip>
#include<fstream>
```

These three libraries allow information that the simulation produces to be printed either on the screen or written to files on the disk

```
#include<map>
#include<bitset>
#include<vector>
```

These three libraries allow for various types of data containers to be used. Maps are collections of data that can be accessed via a key. Bitsets are just lists of bits that allow for compact storage and fast access of binary styled data. Vectors are lists that allow for complicated pieces of information to be inserted and removed. The vectors are used to store the agents and games used in the simulations. The maps and bitsets are used to store information inside the agents that make up a population.

```
#include<algorithm>
```

The algorithm library allows the use of the find function on vector libraries. This is a search algorithm that will return the first instance of a specific member of the vector. This is used to know at what location certain agents are within the vector that stores all the agent information.

```
#include<omp.h>
#include<gsl/gsl_rng.h>
```

These two libraries give the simulation a set of advanced features. omp.h gives access to the OpenMP library which allows for multi-threaded execution (more than one processor or core can contribute to the running of the simulation). gsl_rng.h gives access to the Gnu Scientific Library (GSL) random number generation (RNG) routines. The GSL RNG is important as it is thread safe (each thread gets its own rng variables so that they are not over writing each other, preventing crashes and allowing for perfect re-runs of the simulation from initial seed values).

```
#include"agent.h"
#include"game.h"
#include"match.h"
```

These files describe the objects outside the standard C++ library that are used in the simulation and the ways in which these objects interact with the program and other objects. `agent.h` lists all the properties that an object of type agent has, such as how much memory it needs and how it will respond when asked something akin to "Do you know agent #123". `game.h` contains similar information for games. `match.h` was needed to help streamline the pairing of agents before they play games since the pair type in the standard C library was too weak (it only allowed for pairs and I needed triples, two agents and one game).

### Function Declarations

```
void prepOutfiles(ostream&, ostream&, ostream&,
ostream&, ostream&, const int , const int , const
int );
```

```
void prepOutfiles(ostream&, ostream&, ostream&,
const int , const int , const int );
```

`prepOutfiles` creates and formats the files that will hold the output of the simulation. The formatting involved is limited to simply creating column labels for csv formatted spreadsheets. SQ simulations require three files to hold output and NC simulations require five. Multiple files are required because ascii files have an upper limit on the number of characters that are allowed per line. NC agents are more complicated than SQ agents and so more files are required to hold population summary information. The function takes the names of three files (the "ostream&" items) and four constant integers as inputs and returns nothing (the"void").

| | |
|---|---|
| | `int bit2pos(const int );`<br>`int pos2bit(const int );` |

The way that each agent will respond to all possible situations is stored in a bitset within that agent. The situation that each bit codes for can be represented by a number that results in treating the possible conditions as a set of switches which can then be read as a binary number. The set of conditions coded by this switch representation has gaps due to impossible scenarios that are ignored inorder to compact the bitset (No agent can face a scenario where the answer to "Would you do worse in X?" and "Would you do better in X?" is "Yes" in both cases). Consequently a pair of conversion algorithms is needed to allow for rapid reading and writing to the bitset on the basis of the conditions an agent is facing. These functions provide these algorithms. These functions each take a constant integer as input and return an integer.

| | |
|---|---|
| | `int PDbitBase(const int );`<br>`int SHbitBase(const int );`<br>`int CKbitBase(const int );`<br>`int BSbitBase(const int );` |

When only a certain game type is possible for agents to face, it would be a waste to have within each agent a complete description of how they would behave in ALL possible 2×2 games. In such cases a more limited bitset is used and these functions allow the simulation to continue as normal by readdressing calls to positions that exist in the full bitset to positions in the smaller bitset. These functions each take a constant integer as input and return an integer.

```
void cull(vector<agent*>*,
int &, const int ,
const float, vector<int
>*, map<unsignedint ,
pair<unsignedshort,
unsignedshort> >*, const
bool, gsl_rng*);
```

At the end of a generation those agents that have not collected enough points to survive into the next round are removed from the population. Those that pass this first test must also pass a random culling of the population. The inputs into this function are as follows:

- **vector<agent*>*** a pointer to the vector that holds all pointers to the agents within this population.
- **int &** a reference to an integer that may be changed. This is used to count how many agents are removed.
- **const int** an integer that remains constant. Here it is the price or ante to live into the next round.
- **const float** a floating point number that remains constant. Here it is the random death rate.
- **vector<int>*** a pointer to a vector of integers. This is used to hold the IDs of all those agents that fail to survive the cull.
- **map<unsigned int, pair<unsigned short, unsigned short> >*** a pointer to a map that takes an unsigned integer as the key and returns a pair made up of unsigned short integers. This is the map of social knowledge regarding who knows who that may be used in future revisions of this program that investigate social networking on a deeper level.
- **const bool** a boolean value that is constant. This is a switch that says whether social networking data needs to be collected and updated as a result of the cull.
- **gsl_rng *** a pointer to the Gnu Scientific Library random number generator object. This is used to pass the random number generator into the cull function.

```
void breed(vector<agent*>*,
int &, const int , const
int , const int , const
bool, const bool, const int
, const int , gsl_rng*, int
&);
```

Following the removal of "unfit" and "unlucky" agents those remaining are allowed to attempt spawning. Agents are allowed to spawn when they find a partner who is willing to spawn with them and both partners can pay the cost of spawning. The inputs into this function are as follows:

- **vector<agent*>*** a pointer to the vector that holds all pointers to the agents within this population.
- **int &** a reference to an integer that may be changed. This is used to count how many agents are generated.
- Three variables of type **const int** which are integers that remain constant throughout the function. These are the cost to spawn and the number of times that agents are allowed to try and find partners to play with and to spawn with.
- Two variables of type **const bool** a boolean value that is constant. These set whether new agents have randomized or uniform behaviours when they meet new agents and after they play a game with another agent.
- Two variables of type **const int** which are integers that remain constant throughout the function. These set the generation in which the new agent are generated so that their "age" may be tracked and the number of rounds that an agent may go without seeing another agent before they "forget" them.
- **gsl_rng *** a pointer to Gnu Scientific Library random number generator object. This is used to pass the random number generator into the cull function.
- **int &** a reference to an integer that may be changed. This is used to keep track of the total number of agents generated throughout the simulation and to assign each new agent a unique ID number based on this.

```
void statsSummary(const int , vector<agent*>*,
const int , const int , const int , const int ,
ostream&, ostream&, ostream&, ostream&, ostream&,
ostream&, const bool);
```

```
void statsSummary(const int , vector<agent*>*,
const int , const int , const int , const int
, ostream&, ostream&, ostream&, ostream&, const
bool);
```

This is the function that collects and outputs the results of the simulation at the end of a specifiec number of generations. The inputs into this function are as follows:

- **vector<agent*>*** a pointer to the vector that holds all pointers to the agents within this population.
- Four variables of type **const int** which are integers that remain constant throughout the function. These are the number of agents that were removed at the end of the past generation, the number of agents created at the end of the past generation, the size of the bitset that codes for the complete behvaiour set of each agent, and number that codes for the game type that the agents within the generation faced.
- Four objects of type **ostream&** which are references to the files to which output is streamed.
- **const bool** a boolean value that is constant. This is used to flag whether social networking data is to be collected or not.



### E.3.1 Main Function

The main function is the set of commands that executes when the program runs, creating variables, performing calculations and calling other functions as required.

#### Major Variable Declarations

| | |
|---|---|
| `int ante=34;` | number of resource points necessary to survive at the end of a generation |

| | |
|---|---|
| `floatrndDeathRate=0.0;` | Probability that any given agent will die,regardless of resources held |
| `int spawnCost=9;` | Per Agent Cost to Spawn |
| `int g=0;` | Sets game type for ALL rounds. 0: Random, 1: PD, 2:Stag Hunt, 3: Chicken, 4: Battle of the Sexes |
| `int runType=0;` | Controls what data is collected. 0: Pull only population stats for fast population assessments 1: Pull Full stats summaries. |
| `int numAgents=100;` | Number of Agents to create at the start of trial. |
| `int numPops=12;` | Number of populations to test across different environmental seeds. |
| `int numEnvs=12;` | Number of times an the same initial population should be tested across the same environmental conditions but with a different seed to the random number generator. |
| `int numGens=100;` | Number of generations within a population. |
| `int numRounds=10;` | Number of rounds per generation. |
| `int spawnDisc=0;` | If 0 then indiscriminate spawning. If >0 then indicates number of spawning attempts allowed per generation. |

| | |
|---|---|
| `int playDisc=0;` | If 0 then indiscriminate game playing. If >0 then indicates the number of attempts allowed per round. |
| `boolrndAft=false;` | Sets randomization in agent `afterPlay` behaviour. False sets N for all outcomes. |
| `boolrndLike=false;` | Sets randomization across agents of how unknown agents are initially liked. |
| `int maxAgents=10000;` | Number of living agents at which a population ceases to run. |
| `int maxAbsent=numRounds;` | Number of generations not encountering a particular agent after which that agent is forgotten. |
| `int deadGen=0;` | Tracks number of dead at end of each generation. |
| `int bornGen=0;` | Tracks the number of agents born into each new generation |
| `int paySumGen=0;` | Sum of all possible payoffs over a generation |
| `int paidSumGen=0;` | Total of payoffs actually received |
| `int gamesGen=0;` | Total number of games played over a generation |
| `stringoutFileName;` | holds the name of the output file |
| `boolsocNet=false;` | When TRUE the simulation collects social networking information in additional outFiles. The socNet files are always created, but only filled is socNet is on. |

| | |
|---|---|
| `int trialLineNum=0;` | Says which line of In.dat the master data was pulled from. Also used to differentiate between different runs. |
| `int nthreads=1;` | Tracks the total number of threads used. |
| `int tid=0;` | Stores the number of the thread running a particular process. |
| `int npt=0;` | npt is used to control rng resetting. Declaring this variable outside the introduction of parallel processing allows it to be universal across all threads. |

| | |
|---|---|
| `int choiceSize=12717;` | `int choiceSize=6561;` |



Initializes the size of the choice bitset for each agent class. There are less interventionist ways to do this, but this made the programming simpler.

### E.3.2 Prepare Input and Output Files

| | |
|---|---|
| `game::loadGameNumbers();` | |

NC class agents need to be able to recognize what game they are playing in order to determine how to play. This process of recognition is not particularly easy. The function *loadGameNumbers* is part of the NC game class. It loads a series of numbers that can be used as an index for identifying games.

| | |
|---|---|
| `ifstreaminFile( "In.dat", ios::in);` | `ifstreaminFile( "In.dat", ios::in);` |

Open a the file named "In.dat" as an input only file with the C++ ifstream constructor. In.dat holds all the information regarding the environment and populations that will be generated during a run of the simulation. Sample content from an In.dat file is as follows:

1 0 10 5000 100 0 350 400 0.00 0 100 1 1 12 12 100 100
1 0 10 5000 100 0 350 400 0.00 100 0 1 1 12 12 100 100
1 0 10 5000 100 0 350 400 0.00 100 100 1 1 12 12 100 100

```
if(!inFile)
{
  cerr << "IN.DAT COULD NOT BE OPENED"<< endl;
  exit(1);
}
```

This is an error checking routine. If In.dat could not be opened for some reason then an error message is displayed and the simulation exits.

```
outFileName="RT-#0.csv";
ofstreamoutFile0(outFileName.data(), ios::out);
outFileName="RT-#1.csv";
ofstreamoutFile1(outFileName.data(), ios::out);
outFileName="RT-#2.csv";
ofstreamoutFile2(outFileName.data(), ios::out);
outFileName="RT-#3.csv";
ofstreamoutFile3(outFileName.data(), ios::out);
outFileName="RT-#4.csv";
ofstreamoutFile4(outFileName.data(), ios::out);
```

```
outFileName="RT-#0.csv";
ofstreamoutFile0(outFileName.data(), ios::out);
outFileName="RT-#1.csv";
ofstreamoutFile1(outFileName.data(), ios::out);
outFileName="RT-#2.csv";
ofstreamoutFile2(outFileName.data(), ios::out);
```

Creates a series of files that will hold the results of the simulation. RT-#0.csv holds all the basic summary information of the population that is generated during the simulation. Files RT-#1.csv and higher hold population averages concerning the number of options that are turned off in each agent across the entire population. These outfiles are formatted as comma delimited values files (.csv) and are basically ASCII files. ASCII files have a limitation on the number of characters per line

so it was necessary to break the files up. SQ agents require a total of three outfiles. NC agents require a total of five outfiles because of the larger range of behaviours that need to be encoded and tracked within this agent class.

```
if( !outFile0 || !outFile1 || !outFile2 ||
!outFile3 || !outFile4)
{
cerr << "AN OUTPUT FILE COULD NOT BE OPENED"<<
endl;
exit(1);
}
```

```
if( !outFile0 || !outFile1 || !outFile2)
{
cerr << "AN OUTPUT FILE COULD NOT BE OPENED"<<
endl;
exit(1);
}
```

This is an error checking routine. If any of the output files could not be created for some reason then an error message is displayed and the simulation exits.



```
inFile >> runType >> socNet >> numAgents >>
maxAgents >> maxAbsent >> g;
```

This reads the first entries from In.dat so that the outfiles may be appropriately formatted

```
inFile >> runType >> socNet >> numAgents >>
maxAgents >> maxAbsent >> g;
```

This reads the first entries from In.dat so that the outfiles may be appropriately formatted

```
prepOutfiles(outFile0, outFile1, outFile2,
outFile3, outFile4, choiceSize, g, runType);
```

```
prepOutfiles(outFile0, outFile1, outFile2,
choiceSize, g, runType);
```

As mentioned above prepOutfiles creates and formats the files that will hold the output of the simulation. This is where this function is called for the first and only time.

```
inFile.seekg(0);
```

Resets reading of In.dat to the beginning.

### E.3.3 Trials Loop

```
while(inFile >> runType >> socNet >> numAgents >>
maxAgents >> maxAbsent >> g >> ante >> spawnCost
>> rndDeathRate >> spawnDisc >> playDisc >>
rndLike >> rndAft >> numPops >> numEnvs >>
numGens >> numRounds) {
```

This is a while-loop. It continues until there is no more data in In.dat to read and thus no more population trials to run.

```
(void) omp_set_num_threads(4);
#pragma omp parallel
```

### E.3.4 Population Loop and Beginning of Multi-Thread Processing

```
for firstprivate(spawnDisc, playDisc, rndLike,
rndAft, runType, socNet, numAgents, maxAgents,
maxAbsent, g, ante, spawnCost, rndDeathRate,
numPops, numEnvs, numGens, numRounds, tid, npt)
```

Each line read from In.dat at the beginning of the while-loop says how many populations are to be tested at the given conditions and how many environments each population is to be tested in. Multithreaded processing takes over after the while-loop so that more than one of the specified number of populations can be tested at a time. omp_set_num_threads specifies how many threads are to be created. The pragma designates that the for-loop that follows it (the loop that controls how many populations are run) is to be run in parallel via OpenMP and that each thread is to be given its own copy of the variables listed in parentheses following the firstprivate statement.

```
for(int np=0; np<numPops; np++) {
npt=np;
  tid = omp_get_thread_num();

#pragmaomp critical (output_data) {
if(tid == 0){
  nthreads = omp_get_num_threads();
  printf("Number of threads = %d\n", nthreads);}}
```

This block of code is used to help track mutli-threaded execution. It starts by ensuring that each thread makes sure it has a copy of the population that it is running that no other thread can update (npt=np) and gets its own thread ID number (useful for debugging). Thread number 0 then outputs on screen the total number of threads divided up to work on the process of running the simulation through various populations.

```
gsl_rng* r = gsl_rng_alloc(gsl_rng_mt19937);
```

Each thread initializes its own random number generator using the Mersene-Twister Algorithm.

### E.3.5 Environment Loop

```
for(int ne=0; ne<numEnvs; ne++) {
```

Start the loop that will cycle through the number of environments that the population the thread has been assigned must be run through

### E.3.6 Build Initial Population for Environment

```
int agentCount=0;
```

agentCount is used to count the number of agents created during the lifetime of a population as well as assign unique ID numbers to agents as they are generated. Here it is initialized to zero.

```
vector<agent*> allAgents;
```

A vector called allAgents is created that can hold pointers to objects that are of type agent. Pointers amount to shortcuts to where an actual object is really stored. They are used here because the vector holds what amounts to an ordered queue of data. As agents are added and removed to the population the vector must be reorganized and/or resized. Since pointers are smaller than the objects they reference it is easier and faster to reorganize and resize a vector of pointers than it is to do so on a vector of complicated and large objects.

```
gsl_rng_set(r, npt);
```

The random number generator used by the thread is then seeded with the population number to ensure that the created population is the same as it enters every simulation environment.

```
#pragmaomp critical (output_data) {
cout << "Thread "<< tid << " is beginning Trial
"<< trialLineNum << ", Population "<< npt << ",
Environment "<< ne << endl;
}
```

Forces each thread to output on screen what population and environment that it is about to begin simulating. The critical pragma is necessary to prevent two threads from trying to output to the screen at the same time.

```
for(int i=0; i<numAgents; i++) {
  agentCount++;
  allAgents.push_back(newagent(spawnDisc,
playDisc, rndLike, rndAft, 1, maxAbsent, r,
agentCount)); }
```

This loop does two things: it creates an object of type agent and stores it in the memory of the computer and it creates a pointer to that agent and stores it in the allAgents vector.

```
gsl_rng_set(r, ne);
```

Once the population has been created with the random numbers generated using npt as the seed the random number generator must be reseeded to ensure that the environment is reproducible. Here the random number generator is reseeded with the environment number.

### E.3.7 Collect Data on Initial Population

```
#pragmaomp critical (output_data) {
  outFile0 << trialLineNum << ","<< npt << ","<<
ne << ","<< g << ","<< ante << ","<< spawnCost <<
","<< rndDeathRate << ",";
  outFile0 << spawnDisc << ","<< playDisc <<
","<< rndLike << ","<< rndAft << ","<< numAgents
<< ","<< numGens << ",";
  outFile0 << numRounds << ","<< -1 << ","<<
","<< ",";

  switch(runType) {
    case0:  {
      popStatsSummary(-1, &allAgents, 0, 0,
choiceSize, g, outFile0);
      break;}
    case1:  {
      statsSummary(-1, &allAgents, 0, 0,
choiceSize, g, outFile0, outFile1, outFile2,
arcFile, socNet);
      break;}
}}
```

This section of code outputs basic information to the output files based on the population that was initially created. The critical pragma locks this section of code (and all other places where file output occurs with a similar pragma around it) so that only one thread can write to the output files at a time.

### E.3.8 Generation Loop

```
for(int gen=0; gen<numGens; gen++) {
```

Start the for-loop that controls the number of generations

```
paySumGen =0;
paidSumGen = 0;
gamesGen = 0;
```

Ensure the generation specific tracking variables are reset to zero. paySumGen tracks the total amount agents could have received during all the games they played during the generation. paidSumGen tracks how much they actually did receive. gamesGen tracks how many games were played. Together these values can be used to track how close the population is to obtaining an ideal form of play.

### E.3.9 Round Loop

```
for(int round=0; round<numRounds; round++) {
```

### E.3.10 Build Games

```
int numGames=allAgents.size()/2;
```

Initialize a variable called numGames that holds integer values and set it equal to half the size of the current population. Because numGames is an integer all values returned from the calculation are rounded down to the nearest whole number. numGames sets the number of games that will be generated for play in this round.

```
vector<game*> allGames;
for(int i=0; i<numGames; i++)
  allGames.push_back(newgame(g, r));
```

Initialize a vector called allGames that holds pointers to objects of type game. Game objects hold the eight payoffs assoicated with the four outcomes of a 2x2 game and provide all the functionality assocaited with playing the game described by these payoffs (e.g. Once each of the playing agents tells the game what their move is the game will them tell these agents the outcome) The following for-loop loads allGames with as many games/pointers as indicated inside numGames.

### E.3.11 Match Agents and Games

```
vector<match> allMatches;
```

Initialize a vector called allMatches that holds objects of type match. Match objects are integer pairs, with each integer corresponding to the ID of an agent within the population currently being simulalted by a thread.

```
vector<pair<unsignedlong, unsignedshort> >
agentIndex;
```

Initialize a vector called agentIndex that holds pairs made up of a long integer and a short integer. The long integer will hold the position of an agent within the allAgents vector. The short integer is used to count how many times that agent has attempted to match with another agent to play a game.

```
pair<unsignedlong, unsignedshort> tempPair;
```

A pair called tempPair is then created to allow for the agentIndex vector to be filled in a straightforward manned.

```
for(int i=0; i<(int )(allAgents.size()); i++) {
  tempPair.first=i;
  tempPair.second=0;
  agentIndex.push_back(tempPair); }
```

```
match tempMatch;
```

```
int index1=0;
int index2=0;
bool matchFlag=false;
```

Two integer values are created and initialized to zero. These will be used to track which agent positions collected inside agentIndex and which refer to agent positions inside allAgents are currently being tested.
A bool is also created. It will be used to track whether or not a match between two agents has been formed.

```
while(agentIndex.size()>=2) {
```

Begin a loop that will complete the task of matching agents. This is a while-loop that continues as long as there are two or more agents referred to inside agentIndex. References to agents are removed from agentIndex as agents are matched or if the surpass the limits on matching attempts.

```
index1 = gsl_rng_uniform_int(r,agentIndex.size());
index2 = gsl_rng_uniform_int(r,agentIndex.size());
matchFlag=false;
```

Fill the index values with a random integer between zero and one less than the size of agentIndex. Since zero base counting is used for vectors as well as other C++ objects this index value can be used to pick out any agent referenced in agentIndex. matchFlag is also reset to "false" in case it had been switch to "true" during a previous iteration of the loop.

```
if(index1!=index2) {
agentIndex.at(index1).second++;
agentIndex.at(index2).second++;
```

Check to see that an agent has not been matched with themselves. If an agent has been matched with itself then the rest of the matching process is skipped and the loop returns to the beginning where two more numbers within the range specified earlier are drawn. If index1 and index2 refer to different agents then these agents immediately have the number of spawning attempts they have made incremented by one.

```
if(playDisc<=0
  || (playDisc>0
  && ((allAgents.at(agentIndex.at(index1).first))->
playAgree(allAgents.at(agentIndex.at(index2).first)->
agentID())==true
  && (allAgents.at(agentIndex.at(index2).first))->
playAgree(allAgents.at(agentIndex.at(index1).first)->
agentID())==true))) {
  tempMatch.assignment(1, agentIndex.at(index1).first);
  tempMatch.assignment(0, agentIndex.at(index2).first);
  allMatches.push_back(tempMatch);
  matchFlag=true; }
```

Perform a check to see if a match can be made out of the two agents referred to by index1 and index2 inside agentIndex.

A match can be made under the following conditions:

1. The variable playDisc is less than or equal to zero. If this is true then it means that agents are not allowed to have meaningful preferences about who they are matched with. All random pairings are accepted under this condition.

2. The variable playDisc is greater than zero, allowing each agent the number of matching opportunities equal to playDisc, and both agents agree to be matched with the other for the purposes of playing a game that is currently unknown to them.

If either condition is met then the pair is pushed into the vector allMatches and match flag is set to "true".

```
if(index1 > index2) {

if(matchFlag==true|| agentIndex.at(index1).second>=
playDisc) {
  agentIndex.erase( agentIndex.begin() + index1); }

if(matchFlag==true|| agentIndex.at(index2).second>=
playDisc) {
  agentIndex.erase( agentIndex.begin() + index2); }
}
```

Remove matched agents from agentIndex so that they cannot be matched a second time this round.

The order in which agents are removed matters as the vector contracts as objects are removed from it. For example, suppose that index2=4 and index1=3. Removing the agent referred to at index1 first would cause the vector to shift such that the agent previously occupying the fourth position in agentIndex would now occupy the empty third position, the agent previously occupying the fifth postion would now occupy the fourth and so on. Removing the agent referred to by index2 next would thus result in pulling the agent who now occupies the fourth position in agentIndex rather than the agent the index value originally referred to.

This series of conditionals ensures that the larger index value is always removed first so that pulling the wrong agent due to contraction of the vector can be prevented.

```
else {

if(matchFlag==true|| agentIndex.at(index2).second>=
playDisc) {
  agentIndex.erase( agentIndex.begin() + index2); }

if(matchFlag==true|| agentIndex.at(index1).second>=
playDisc) {
  agentIndex.erase( agentIndex.begin() + index1); }
}
```

### E.3.12 Play Games

```
int rowID=0;
int colID=0;
boolrowPlay=-1;
boolcolPlay=-1;
int rowPay=-1;
int colPay=-1;
```

Declare the variables that will be used to play games with the matched pairs of agents. rowID is the ID number of the agent assigned to be the row player, similarly for colID. What rowPlay and colPlay designate depends on the agent type (SQ or NC) involved. For SQ agents these designated whether the row or column players will stay on or switch from the status quo position presented to them. For NC agents these designate whether the top row/left column will be played versus the alternative. rowPay and colPay designate the final payout received by the row and column players from the game.

```
for(int i=0; i<(int )allMatches.size(); i++) {
```

The for-loop ensures that all matched agents are given a game to play.

```
rowID=allAgents.at(allMatches.at(i).firstInPair())->
agentID();
colID=allAgents.at(allMatches.at(i).secondInPair())->
agentID();
```

rowID and colD are updated with the actual ID#s of the agents playing the game, as opposed to their position of the pointer to them as held in the allAgents vector.

```
rowPlay=allAgents.at(allMatches.at(i).firstInPair())->
play(allGames.at(i), colID, true);
colPlay=allAgents.at(allMatches.at(i).secondInPair())->
play(allGames.at(i), rowID, false);
```

The row player takes the ID of the column player and the available information about the the game being played and makes a decision how to play.
The column player takes the ID of the row player and the available information about the the game being played and makes a decision how to play.

```
rowPay=allGames.at(i)->outcome(true, rowPlay,
colPlay);
colPay=allGames.at(i)->outcome(false, rowPlay,
colPlay);
```

The game then assigns payoffs based on the plays made by each player.

```
gamesGen++;
```

The number of games played this generation is updated.

```
paySumGen = paySumGen + allGames.at(i)->getPaySum();
paidSumGen = paidSumGen + rowPay + colPay;
```

The total amount available to be won and the total amount won by all agents over the generation is updated

### E.3.13 Update Agents

```
allAgents.at(allMatches.at(i).firstInPair())->
updateAgent(allGames.at(i), colID, rowPay,
colPay, true);
allAgents.at(allMatches.at(i).secondInPair())->
updateAgent(allGames.at(i), rowID, colPay,
rowPay, false);
```

Finally, each agent updates both how it believes the other agent relates to them and how they relate to the other agent on the basis of the outcomes of the game.

```
agentIndex.clear();
allMatches.clear();

for(int i=0; i<(int )allGames.size(); i++)
  deleteallGames.at(i);
allGames.clear();
```

Once all the games are created, agents matched into pairs and games played all that is left for the round is to clean-up the elements that took up memory to avoid accidentally reusing these elements in a future round.

For agentIndex and allMatches this is done simply by issuing a "clear" command. This cannot be done for allGames since this command would only remove the pointers held in the vecotr and not the elements themselves. Each element pointed to by a pointer inside allGames is thus deleted first and then the vector is cleared.

### E.3.14 Remove Poorly Performing Agents

```
vector<int > deadIDs;
```

Initialize a vector on integers called deadIDs. This vector will hold the ID numbers of agents that do not survive into the next generation. These IDs will be used to update the agents who remain by removing all reference of these now dead agents from them as a way of saving space.

```
cull(&allAgents, deadGen, ante, rndDeathRate,
&deadIDs, &vertTimes, socNet, r);
```

Call the cull function. This is the function that removes all agents from the population who failed to acquire enough points while playing games this generation to pay the ante into the next generation. This function also randomly removes agents from the population at the rate specified by rndDeathRate.

```
if(numGens-gen>1)
  for(int i=0; i<(int )(allAgents.size()); i++)
    allAgents.at(i)->agentAgePlus();
```

All agents who survive longer past the cull function have their age incremented by one.

### E.3.15 Spawning of Remaining Agents

```
breed(&allAgents, bornGen, spawnCost, spawnDisc,
playDisc, rndLike, rndAft, gen+1, maxAbsent, r,
agentCount);
```

Call the breed function. This function matches willing agents who have enough points left over after paying the ante into the next generation to pay the breeding cost to attempt to find a spawning match. Succesful matching results in a new agent being created that receives 50% of its make-up from each parent.



### E.3.16 Collect Generation Data

```
#pragmaomp critical (output_data){
if((gen % 5==0) || gen==1 || (int
)allAgents.size()>maxAgents ||
allAgents.size()<=1){
```

Every fifth generation collect information about the population. This procedure for collecting this information is wrapped in a critical pragma to keep multiple threads from writing to the outfiles at the same time.

Information that is immediately available is put directly into the outfiles here. Information that must be extracted from the population is collected via the statsSummary function.

```
outFile0 << trialLineNum << ","<< np << ","<< ne
<< ","<< g << ","<< ante << ","<< spawnCost <<
","<< rndDeathRate << ",";
outFile0 << spawnDisc << ","<< playDisc << ","<<
rndLike << ","<< rndAft << ","<< numAgents <<
","<< numGens << ",";
outFile0 << numRounds << ","<< gen << ",";
```

Each of the tracking variables has it's current value sent to the outfile to be saved for future analysis. The outfiles are comma separated ASCII files and so they may be read by any spreadsheet program.

```
if(paySumGen>0 && gamesGen>0)
  outFile0 << (float)paySumGen/((float)gamesGen*8.0);
outFile0 << ",";

if(paidSumGen>0 && gamesGen>0)
  outFile0 << (float)paidSumGen/((float)gamesGen*2.0);
  outFile0 << ",";
```

Certain pieces of the data collected in the outfiles are the result of division. In these cases it is important to guard against division by zero. These conditionals check to make sure that division by zero will not occur and either allow the calculation to take place and be included in the output or simply insert a comma adjacent to the previous one. Two commas side by side with nothing between represents an empty cell in a comma delimited spreadsheet.

```
switch(runType){
  case0:{
    popStatsSummary(gen, &allAgents, deadGen,
bornGen, choiceSize, g, outFile0);
    break;}
  case1:{
    statsSummary(gen, &allAgents, deadGen,
bornGen, choiceSize, g, outFile0, outFile1,
outFile2, outFile3, outFile4, arcFile, socNet);
    break;}}}}
```

For testing purposes two different ways of running the simulation were created. The first, case zero, collects only very basic information from the population so that the simulation can run a great deal faster by not having to sift through the make-up of each individual agent in the population. This was a useful feature when it was necessary to test whether or not various environmental conditions would lead to population crashes, explosions, or long term survivable populations. Case 1 calls the full statistics summary and was used in all production simulations.

```
for(int i=0; i<(int )allAgents.size(); i++)
      (allAgents).at(i)->cleanKnow(&deadIDs,
agentCount);
```

Agents are "cleaned". This amounts to having all reference to agents who have been removed from the population removed from the agents that remain in the population. This is a way to conserve memory.

```
deadIDs.clear();
```

The vector deadIDs is cleared to free memory.

```
if(numGens-gen<=1  || (int
)allAgents.size()>maxAgents ||
(int )allAgents.size()<=1 || (int
)allAgents.size()==0){
  break;}
```

A check is performed to see if more agents are alive then the maximum allowed to start a new generation. If so the simulation ceases with this environment/population and moves to a new one. This is an important, although unfortunate, check. Without it populations can easily explode, filling all the available RAM and forcing the simulation to put the overflow on the hard disk swap space when desktop machines are being used. Using swap space effectively grinds execution of the simulation to a halt. On the univeristy cluster stepping outside the memory limits declared at the start of the program results in the complete termination of the program. Neither scenario os desireable.

**Finish the Environment trial for current population**

```
for(int i=0; i<(int )allAgents.size();i++)
    deleteallAgents.at(i);
  allAgents.clear();
```

Once a population simulation hits the maximum number of generations allowed, surpasses the maximum number of agents allowed or has one or fewer agents left the run of that population through the given environment ceases. When this happens all the agents remaining in the population are deleted and then the allAgents vector is cleared of the pointers to where these agents used to be.

**Finish running trials on this population**

```
gsl_rng_free(r);
```

This releases the memory associated with the GSL random number generator.

**Finish running the simulation program**

```
cout<< "done"<< endl;

return0;
```

Output an onscreen message to indicate that the program has finished.

Return ``0'' as the termination status of the simulation, meaning that execution was successful

### E.3.17 Additional Functions

These functions are called during the main body of the program that run the simulation. They support the running of the program by providing specific functionality that it makes sense to group together as a single action and to separate from the main body of the program to make it easier to understand.

`prepOutfiles`

This function prepares the files use to hold the output data. In summary, it adds descriptive headers to the files that will hold the output from the simulation so that information loaded in during the running of the simulation can be more easily identified.

```
void prepOutfiles(ostream&outFile0, os-
tream&outFile1, ostream&outFile2, os-
tream&outFile3, ostream&outFile4, const int
choiceSize, const int g, const int runType) {
```

This is the declaration of the prepOutfiles function. The "void" at the beginning means that this function does not return a value in the way that a function like y=2x4 would. The items in brackets are the arguments the function needs to operate. The first three are references to the files the function will operate with. A constant integer called choiceSize is passed so that the function knows where to split the files that hold the population summaries of how the choice bitset is arranged across all the agents in the population. g is also a constant integer and indicates what type of game the agents are all playing. runType in a constant integer that codes for different ways that the outfiles can be formatted. It was used to turn on and off various methods of tracking infomation during testing.

```
outFile0 << setprecision( 5 ) <<
"\"TrialLine#\",\"Population#\",\"Environment#\",
\"Game Type\",\"Round-Ante\",\"Spawn
Cost\",\"Random Death\",\"Spawn Disc?\",\"Play
Disc?\",\"Randomize Like?\",\"Randomize
AfterPlay?\","; outFile0 << "\"Initial-#\",\"Total
Generations\",\"Rounds/Gen\",\"Generation\",
\"Avg.  Payoff Off.\",\"Avg.  Payoff
Rec.\",\"#Agents-Dead\",";
outFile0 << "\"#Agents-Born\",\"Births:Deaths\",
\"Avg.  Age\",\"Max Age\",\"#Max Age\",\"Avg.
Bits\",\"#Agents-End\",";
```

outFile0 is the file that holds all the basic summary information on each population. This block of code sets the decimal precision that the file can hold to five and the inserts the first group of headers.

```
switch(runType) {
  case0:
    break;
```

This switch is controlled by runType. If runType is set to "0" then this entire function ends. This was a testing feature.

```
Case1:  {
```

If runType is set to "1" then the entire set of headers in inserted into all the outFiles.

```
outFile0 << "\"Avg.# Known\", \"Avg.
#Ns\",\"Avg.  #Es\",\"Avg.  #Fs\",\"Avg.
#~Ns\",\"Avg.  #~Es\",\"Avg.  #~Fs\",";
```

Insert the headers that indicate the columns for various relationship averages. These relationship averages are how many agents are know by each agent, how many agents each agent assigns to relationship type ``N'', how many agents each agent assigns to relationship type ``E'', how many agents each agent assigns to relationship type ``F'', how many agents each agent believes assigns them to relationship type ``N'', how many agents each agent believes assigns them to relationship type ``E'', how many agents each agent believes assigns them to relationship type ``F''.

```
for(int i=0;i<3;i++)
for(int j=0;j<10;j++)
for(int k=0;k<10;k++)
for(int l=0;l<3;l++){
```

```
for(int i=0;i<3;i++)
for(int j=0;j<3;j++)
for(int k=0;k<3;k++)
for(int l=0;l<3;l++){
```

The size of afterPlay differs between the NC and SQ agent classes. The matrix takes the way the agent framed the relationship with the other playing agent, the way the agent believed that the other playing agent framed the relationship with them and how well either the agent or the agent they were playing against did relative to a bench mark and returns a relationship framing value to use on the next interaction with that agent. This is reflected in the differences between the sizes of the loops used to create headers in the output file. The NC agent class needs 900 header labels where the SQ agent class only needs 81.

```
switch(i){
  case0:
    outFile0 << "\"N\""<< j << k << l << ",";
    break;
  case1:
    outFile0 << "\"E\""<< j << k << l << ",";
    break;
  case2:
    outFile0 << "\"F\""<< j << k << l << ",";
    break;}}
```

This switch, in conjunction with the three consecution for-loops, controls which description is put into the header for each afterPlay description. Header labels are in the format Xjkl. The X represents the new relationship assignment that results from the combination of factors j, k, and l. The j represents how the agent for whom the assessment is being made fared over the game. For NC agents this means a comparison to the best available option. For SQ agents this means a comparison to the original status quo option. K is the same assessment but made for the other player. The l represents the way the relationship had been assessed prior to playing the game.

```
outFile0 << "\"SpEE\",\"SpEN\",\"SpEF\",\"SpNE\",
\"SpNN\",\"SpNF\",\"SpFE\",\"SpFN\",\"SpFF\",";
outFile0 << "\"PpEE\",\"PpEN\",\"PpEF\",\"PpNE\",
\"PpNN\",\"PpNF\",\"PpFE\",\"PpFN\",\"PpFF\",";
```

outFile0 is the file that holds all the basic summary information on each population. This block of code inserts descriptive headers for summaries of the average number of players exhibiting various types of preferences for interacting with other agents. Sp designates a spawning preference and Pp designates a playing preference. The capital letters following the designation of the preference type indicate the relationship framing conditions under which the interaction is allowed by an agent. XY would indicate that when a given agent frames their relationship with another agent as being of type X and believes that the other agent sees the relationship between them of being of type Y then the action the preference is related to is allowed to occur.

```
switch(g) {
```

Depending on the type of games that agents within the simulation will play the outfiles will be setup differently. This is to conserve space. If only one type of game is going to be played then only information inside each agent that is related to that game type needs to be collected.

```
case0:  {
  for(int i=0; i<10; i++)
    outFile0 << "\"S\""<< i+1 << ",\"S\""<< i+1
<< "%,";

  for(int i=0; i<10; i++)
    outFile0 << "\"NS\""<< i+1 << ",\"NS\""<< i+1
<< "%,";}
  break;
```

A game type of "0" means that agents are playing random games and thus that all data on the shape of the the bitst choice within each agent needs to be collected across the population. The loops given here add descriptive headers that will indicate which bits within the bitset choice are set (i.e. they have a value of "1") most frequently, designated by "S", and what that frequency is, as well as those that are set with the smallest frequencey, designated by "NS".

```
case1:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++)
    outFile0<< "\"PD"<< 13740+j*4096+i*12288 <<
"\",";}
  break;
```

```
case1:  {
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++)
    outFile0<< "\"PD"<< PDbitBase(k)+j*4096+
i*12288 << "\",";}
  break;
```

A game of type "1" means that the only game that agent's will ever play is a prisoner's dilemma. For status quo agents this means that there are only 36 bits in the entire choice bitset that matters. This set of nested looped and the PDbitBase function combine together to list all these bits as descriptive headers in outFile0. They other outfiles will not be needed since so few bits matter.

```
case2:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++)
    outFile0<< "\"SH"<< 9720+j*4096+i*12288 <<
"\",";}
  break;
```

```
case2:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++)
    outFile0<< "\"SH"<< SHbitBase(k)+j*4096+
i*12288 << "\",";}
  break;
```

A game of type "2" means that the only game that agent's will ever play is a stag hunt. For status quo agents this means that there are only 36 bits in the entire choice bitset that matters. This set of nested looped and the SHbitBase function combine together to list all these bits as descriptive headers in outFile0. They other outfiles will not be needed since so few bits matter.

```
case3:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++)
    outFile0<< "\"CK"<< 2010+j*4096+i*12288 <<
"\",";}
  break;
```

```
case3:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++)
    outFile0<< "\"CK"<< CKbitBase(k)+j*4096+
i*12288 << "\",";}
  break;
```

A game of type "3" means that the only game that agent's will ever play is chicken. For status quo agents this means that there are only 36 bits in the entire choice bitset that matters. This set of nested looped and the CKbitBase function combine together to list all these bits as descriptive headers in outFile0. They other outfiles will not be needed since so few bits matter.

```
case4:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++)
    outFile0<< "\"BS"<< 3765+j*4096+i*12288 <<
"\",";}
  break;
```

```
case4:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++)
    outFile0<< "\"BS"<< BSbitBase(k)+j*4096+
i*12288 << "\",";}
  break;
```

A game of type "4" means that the only game that agent's will ever play is battle of the sexes. For status quo agents this means that there are only 36 bits in the entire choice bitset that matters. This set of nested looped and the BSbitBase function combine together to list all these bits as descriptive headers in outFile0. They other outfiles will not be needed since so few bits matter.

```
outFile1 << setprecision( 5 );
outFile2 << setprecision( 5 );
outFile3 << setprecision( 5 );
outFile4 << setprecision( 5 );
```

```
outFile1 << setprecision( 5 );
outFile2 << setprecision( 5 );
```

Set the decimal precision of the outFiles to five digits of accuracy.

```
for(int i=0; i<choiceSize; i++)
  if(i <= 3180)
    outFile1 << "X"<< i << ",";
  else
    if(i <= 6359)
      outFile2 << "X"<< i << ",";
    else
      if(i<= 9538)
        outFile3 << "X"<< i << ",";
      else
        outFile4 << "X"<< i << ",";
  outFile1 << endl;
  outFile2 << endl;
  outFile3 << endl;
  outFile4 << endl;}}
```

```
for(int i=0; i<choiceSize; i++)
  if(i <= 3280)
    outFile1 << "X"<< pos2bit(i) << ",";
  else
    outFile2 << "X"<< pos2bit(i) << ",";

  outFile1 << endl;
  outFile2 << endl;}}
```



In SQ type agents the first 3280 headers that will designate population averages for bits set within the bitset choice are sent to to outFile1. The function pos2bit converts the position in the bitset to the value that that position corresponds to. The remaining headers are sent to outFile2. Insert an end of line in each file so that data is entered on a new line and not as a header.

```
outFile0 << endl;}
```

Insert an end of line in outFile0 so that data is entered on a new line and not as a header.

```
void cull(vector<agent*> *allAgentsPtr,
int &deadGenRef, const int anteRef, const
floatrndDeathRateRef, vector<int >* deadIDsPtr,
map<unsignedint , pair<unsignedshort,
unsignedshort> > *vertMapPtr, const boolsocNet,
gsl_rng* r){
```

Declares the function the removes agents from the population either because they have not collected enough points in the current generation to survive into the next or because of the random death rate. The void at the start of this function indicates that it does not return a value. As inputs the functions takes a pointer to the vector that stores information on the whereabouts in memory of the current agents in the populations. A reference to an integer which stores a count about the number of agents who are culled. A constant integer that holds the ante that each agent must pay to survive into the next round. A constant floating point value that sets the random death rate. A pointer to a vector of integers that holds the IDs of all the agents who are culled. A pointer to a map used in tracking social network information. A constant boolean that controls whether social networking information is collected or not. A pointer to the random number generator.

```
deadGenRef=0;
```

Ensures that counter that tracks the number of agents who are culled from the population is set to zero before the cull begins.

```
for(int i=0; i<(int )allAgentsPtr->size(); i++){
```

Initializes the loop that iterate through all the agents in the population.

```
if((allAgentsPtr->at(i))->agentRes() < anteRef ||
gsl_rng_uniform(r) <= rndDeathRateRef){

  deadIDsPtr->push_back((allAgentsPtr->at(i))->
agentID());

  if(socNet==true){
    tempPair.first=(allAgentsPtr->at(i))->
agentBirthday();
    tempPair.second=(allAgentsPtr->at(i))->
agentBirthday()+(allAgentsPtr->at(i))->
agentAge();
    vertMapPtr->insert(make_pair((allAgentsPtr->
at(i))->agentID(), make_pair(tempPair.first,
tempPair.second)));}

  deleteallAgentsPtr->at(i);
  allAgentsPtr->erase((allAgentsPtr->begin())+i);

  i = i - 1;

  deadGenRef++;}
```

Tests to see if the agent currently being inspected does not have enough resource to pay the ante or if they receive a random number from the generator that is smaller than the random death rate.

If either of these conditions is true then the agent is removed from the population. First the agent's ID is added to the collection of dead agent IDs. Then the agent is deleted. Then the reference to the agent in the vector allAgents is deleted.

Removing the agent from the vector shrinks the size of the vector and shifts the position of all agent references that followed the deleted agent. The loop counter must be decremented to account for this.

Increment the counter that tracks the number of agents removed from the population.

```
else
  (allAgentsPtr->at(i))->
setRes((allAgentsPtr->at(i))->agentRes() -
anteRef);}}
```

If the agent is not removed from the population then the cost of living into the next round is subtracted from their current resource total.

```
void breed(vector<agent*> *allAgentsPtr, int
&bornGenRef, const int spawnCostRef, const int
spawnDisc, const int playDisc, const boolrndLike,
const boolrndAft, const int birthPop, const int
maxAbsent, gsl_rng* r, int &agentCountRef ){
```

Declares the function that allows agents the opportunity to spawn with each other, thereby adding new agents to the population. The void at the start of this function indicates that it does not return a value. As inputs the functions takes a pointer to the vector that stores information on the whereabouts in memory of the current agents in the populations. A reference to an integer that tracks the number of agents created during the spawn. A constant integer that indicates whether agents are allowed to have preferences regarding who they spawn with and, if so, how many chances to find an acceptable spawning partner they receive. A constant integer that indicates whether agents are allowed to have preferences regarding who they play with and, if so, how many chances to find an acceptable playing partner they receive. A constant boolean that indicates whether the initial relationship frame that agents assign to each other on meeting for the first time is randomized or not in this population. A constant boolean that indicates whether agents are capable of assigning new relationship frames to other agents after they have played with them or not. A constant integer that indicates what generation the agent was created in. A constant integer that controls how many rounds an agent can go without interacting with another agent before they forget about them. A pointer to the random number generator. A reference to an integer that tracks how many agents have been created in the population throughout the simulation.

```
vector<pair<unsignedlong, unsignedshort> >
agentIndex;
```

Create a vector that will hold pairs consisting of the position of an agent in the vector allAgents and the number of times that spawning has been attempted with that agent.

```
pair<unsignedlong, unsignedshort> tempPair;
```

Declare a pair that will be used to hold values before being loaded into the vector of pairs just created.

```
bornGenRef = 0;
```

Set the number of agents created during spawning to zero before spawning begins.

```
for(int i=0; i<(int )allAgentsPtr->size(); i++){
```

Begin the loop that will iterate through all the agents in the population for spawning.

```
if((allAgentsPtr->at(i))->agentRes() >=
spawnCostRef){
  tempPair.first=i;
  tempPair.second=0;
  agentIndex.push_back(tempPair);}}
```

Load all agents who are able to pay the spawning cost into the vector that holds pairs of information for spawning.

```
unsignedint index1=0;
unsignedint index2=0;
```

Declare two unsigned integers to hold positions of agents in the vector agentIndex. These integers are "unsigned", meaning that negative values are not allowed. This increases the available size of positive integers.

```
while(agentIndex.size()>=2){
```

Declare the loop that will allow eligible agents to attempt to spawn. This loop will continue as long as there are two or more agents left in the pool.

```
index1 = gsl_rng_uniform_int(r,agentIndex.size());
index2 = gsl_rng_uniform_int(r,agentIndex.size());
```

Assign random numbers within the range zero to the one

```
if(index1!=index2){
  agentIndex.at(index1).second++;
  agentIndex.at(index2).second++;
```

Check to see if the randomly selected agents turned out to be the same. If they are then the remainder of the loop is skipped and two new random numbers are generated for use in picking agents from agentIndex. If they are not the same then the first thing done is to increment the second value in the agentIndex pair that corresponds to the number of spawning attempts made by each agent.

```
if(spawnDisc<=0
  || (spawnDisc>0
  && ((allAgentsPtr->at(agentIndex.at(index1).
first))->spawnAgree(allAgentsPtr->at(agentIndex.
at(index2).first)->agentID())==true
  && (allAgentsPtr->at(agentIndex.at(index2).
first))->spawnAgree(allAgentsPtr->at(agentIndex.
at(index1).first)->agentID())==true))){
```

Check to see if spawning between these two agents is possible. This occurs when one of two conditions is satisfied. Either spawnDisc is less than or equal to zero which means that all agents in the population can spawn with whoever they would like or spawnDisc is greater than zero and so agents within the population may reject presented spawning partners up to the number of times equal to the value of spawnDisc.

```
(allAgentsPtr->at(agentIndex.at(index1).first))->
setRes((allAgentsPtr->at(agentIndex.at(index1).
first))->agentRes()-spawnCostRef);
(allAgentsPtr->at(agentIndex.at(index2).first))->
setRes((allAgentsPtr->at(agentIndex.at(index2).
first))->agentRes()-spawnCostRef);
```

If the conditions for spawning have been satisfied then each agent has the cost of spawning subtracted from their point resources.

```
allAgentsPtr->push_back(newagent(spawnDisc,
playDisc, rndLike, rndAft, birthPop, maxAbsent,
r, agentCountRef));
(allAgentsPtr->back())->agentSpawn((allAgentsPtr->
at(agentIndex.at(index1).first)),(allAgentsPtr->
at(agentIndex.at(index2).first)),r);
agentCountRef++;
```

Create a new agent with the same constraints and abilities as all the other agents in the population. This simply creates a new agent with randomized the same as when the initial seed population was created. This needs to be corrected to reflect the influence of the agents who spawned this new agent so a function called agentSpawn is called which steps through every value that can be influenced by the parents and determines for each case whether to copy the information from one parent with 50% randomization.

```
bornGenRef++;}
```

Add one to the total number of agents created through the spawning process.

```
if(index1 > index2){
  if((allAgentsPtr->at(agentIndex.at(index1).
first))->agentRes() < spawnCostRef ||
agentIndex.at(index1).second>= spawnDisc){
  agentIndex.erase( agentIndex.begin() +
index1);}
    if((allAgentsPtr->at(agentIndex.at(index2).
first))->agentRes() < spawnCostRef ||
agentIndex.at(index2).second>= spawnDisc){
  agentIndex.erase( agentIndex.begin() +
index2);}}
else{
  if((allAgentsPtr->at(agentIndex.at(index2).
first))->agentRes() < spawnCostRef ||
agentIndex.at(index2).second>= spawnDisc){
  agentIndex.erase( agentIndex.begin() +
index2);}
if((allAgentsPtr->at(agentIndex.at(index1).
first))->agentRes() < spawnCostRef ||
agentIndex.at(index1).second>= spawnDisc){
  agentIndex.erase( agentIndex.begin() +
index1);}}}}
```

This section of code removes agents from the collection of agents available for spawning if they have either reached the maximum number of spawning attempts available to them or if they no longer have enough resources to be eligible for spawning again at the end of this generation.

There is so much code here for two reasons. First, each agent must be tested against each set of conditions. Second, the order that agents are removed from the pool of eligible spawners matters because of the way that they are referred to in agentIndex. Removing an agent from agentIndex lowers the index number of all agents referred to in the vector if they have an index number higher than the one removed. This can create a tracking error and result in the removal of the wrong agent.

```
agentIndex.clear();}
```

agentIndex is cleared to ensure that the memory it was using is freed and to remove the possibility of future conflicts.

| | |
|---|---|
| `void statsSummary(const int gen, vector<agent*> *allAgentsPtr, const int deadGen, const int bornGen, const int choiceSize, const int g, ostream&outFile0, ostream&outFile1, ostream&outFile2, ostream&outFile3, ostream&outFile4, ostream&arcFile, const boolsocNet){` | `void statsSummary(const int gen, vector<agent*> *allAgentsPtr, const int deadGen, const int bornGen, const int choiceSize, const int g, ostream&outFile0, ostream&outFile1, ostream&outFile2, ostream&arcFile, const boolsocNet){` |

This function collects summary information from the population and places it in the various outfiles for storage until it can be analyzed. It is called when a population is first created, when a population has gone through its last generation and every five generations in between these two events. The void at the beginning of the function declaration indicates that this function does not return a value. For inputs this function takes a constant integer holding the generation in which the function was called from, a pointer to the vector which holds the pointers to all the agents currently alive in the population. A constant integer holding the number of agents that died in the current generation. A constant integer holding the number of agents that were spawned in the current generation. A constant integer holding the size of the choice bitset inside of each agent. A constant integer indicating the type of game(s) that agents within this population are playing. References to various files which will hold the results of the information summary about to take place. A constant boolean that indicates whether social network tracking is turned on or not.

| | |
|---|---|
| `int maxAge=0;` | An integer that will hold the age of oldest still living agent in generations survived. |
| `int maxAgeCount=0;` | An integer that will hold a count of the number of agents that count as being "oldest". |
| `int bitsSetGen=0;` | An integer that will hold the sum of all the bits within the bitset choice that are set to "1" or "on" within the population. |

| | |
|---|---|
| `floatavBitsSetGen=0;` | A floating point value that will hold the average number of bits set to "1" or "on" across the population. |
| `int choiceTotals[choiceSize];` | A matrix container on integer values that will hold the exact number of agents who have each bit set to "1" or "on" within the bitset choice. |
| `floatavAgeGen;` | A floating point value that will hold the average age of all agents in the population. |

| | |
|---|---|
| `floatafterPlayStats[3][10][10][3] = {{{{0}}}};` | `floatafterPlayStats[3][3][3][3] = {{{{0}}}};` |

A matrix container of floating point values that will hold the average number of agents that have various afterPlay distributions set.

| | |
|---|---|
| `floatspawnPrefStats[9] = {0};` | A matrix container of floating point values that that will hold the average number of agents with each spawning preference. |
| `floatplayPrefStats[9] = {0};` | A matrix container of floating point values that that will hold the average number of agents with each playing preference. |
| `int likingCounts[6] = {0};` | A matrix container integer values that will hold counts of the number of agents with various initial ways of liking other agents. |
| `int ageGen=0;` | An integer value that will hold the sum of the ages of all living agents (use to calculate the average age). |

```
int agentsKnown=0;
```

An integer value that will hold the sum of all the agents known by each agent.

```
for(int i=0; i<choiceSize; i++)
  choiceTotals[i]=0;
```

The container choiceTotals cannot have all its values initialized to zero on creation because the size of this container depends on a variable (choiceSize). This loop ensures that all the values in choiceTotals are initialized to zero by iterating through the entire container and setting each value to zero.

```
for(int i=0; i<(int )allAgentsPtr->size(); i++){
```

Data must be collected from each agent. This declares a loop that will cycle through each agent using "i" as the index value for making reference to agents within the vector allAgents.

```
switch(g) {
```

Which summary procedure is called will depend on which game(s) agents in the population are playing. This declares a switch that will ensure that the proper information is colelcted based on game type.

```
case0:{
  for(int j=0; j<choiceSize; j++)
    if((*allAgentsPtr).at(i)->
choiceBitTest(j)==true)
      choiceTotals[j]++;
  break;}
```

When the game type (g) is zero the agents are playing random games. This means that the entire choice bitset is relevant and so every bit within this bitset is tested within every agent.

```
case1:{
  for(int j=0;j<3;j++)
  for(int k=0;k<3;k++)
    if((*allAgentsPtr).at(i)->
choiceBitTest(1357+k*4096+j*12288)==true)
      choiceTotals[1357+k*4096+j*12288]++;
  break;}
```

```
case1:{
  for(int j=0;j<3;j++)
  for(int k=0;k<3;k++)
  for(int l=0;l<4;l++)
    if((*allAgentsPtr).at(i)->
choiceBitTest(bit2pos(PDbitBase(l)+k*4096+
j*12288))==true)
      choiceTotals[bit2pos(PDbitBase(l)+k*4096+
j*12288)]++;
  break;}
```

When the game type (g) is one the agents are playing prisoner's dilemmas. This means that only bits within the bitset choice that refer to PDs are relevant and so only these bits are tested within every agent.



```
case2:{
  for(int j=0;j<3;j++)
  for(int k=0;k<3;k++)
    if((*allAgentsPtr).at(i)->
choiceBitTest(1181+k*4096+ j*12288)==true)
        choiceTotals[1181+k*4096+j*12288]++;
  break;}
```

```
case2:{
  for(int j=0;j<3;j++)
  for(int k=0;k<3;k++)
  for(int l=0;l<4;l++)
    if((*allAgentsPtr).at(i)->
choiceBitTest(bit2pos(SHbitBase(l)+k*4096+
j*12288))==true)
      choiceTotals[bit2pos(SHbitBase(l)+k*4096+
j*12288)]++;
  break;}
```

When the game type (g) is two the agents are playing stag hunts. This means that only bits within the bitset choice that refer to SHs are relevant and so only these bits are tested within every agent.

```
case3:{
  for(int j=0;j<3;j++)
  for(int k=0;k<3;k++)
    if((*allAgentsPtr).at(i)->
choiceBitTest(408+k*4096+j*12288)==true)
        choiceTotals[408+k*4096+j*12288]++;
  break;}
```

```
case3:{
  for(int j=0;j<3;j++)
  for(int k=0;k<3;k++)
  for(int l=0;l<4;l++)
    if((*allAgentsPtr).at(i)->
choiceBitTest(bit2pos(CKbitBase(l)+k*4096+j*12288)==true))
      choiceTotals[bit2pos(CKbitBase(l)+k*4096+
j*12288)]++;
  break;}
```

When the game type (g) is three the agents are playing games of chicken. This means that only bits within the bitset choice that refer to CKs are relevant and so only these bits are tested within every agent.



```
case4:{
  for(int j=0;j<3;j++)
  for(int k=0;k<3;k++)
    if((*allAgentsPtr).at(i)->
choiceBitTest(627+k*4096+j*12288)==true)
      choiceTotals[627+k*4096+j*12288]++;
  break;}
```

```
case4:{
  for(int j=0;j<3;j++)
  for(int k=0;k<3;k++)
  for(int l=0;l<4;l++)
    if((*allAgentsPtr).at(i)->
choiceBitTest(bit2pos(BSbitBase(l)+k*4096+
j*12288)==true))
      choiceTotals[bit2pos(BSbitBase(l)+k*4096+
j*12288)]++;
  break;}}
```

When the game type (g) is four the agents are playing Battle of the Sexes dilemmas. This means that only bits within the bitset choice that refer to BSs are relevant and so only these bits are tested within every agent.

```
bitsSetGen = bitsSetGen +
(*allAgentsPtr).at(i)->agentBitsSet();
ageGen = ageGen + (*allAgentsPtr).at(i)->
agentAge();
```

Add to the variable that tracks the number of bits set within the bitset choice across all the agents in the population the number of bits set within the current agent being examined.

```
if((  (*allAgentsPtr).at(i)->agentAge()) >=
maxAge){
  if((  (*allAgentsPtr).at(i)->agentAge()) >
maxAge){
    maxAge = (*allAgentsPtr).at(i)->agentAge();
    maxAgeCount = 1;}
  else
    maxAgeCount++;}
```

Check to see how old the agent is and compare that with the age of the previously discovered oldest agent. If the current agent is younger than the oldest agent then do nothing. If they are the same age then increment the variable that tracks the number of agents at the maximum age by one. If they are older then the previously thought to be the oldest age then set the age of the current agent being examined to be the oldest age and then reset the counter for the number of agents at that age to one.

```
for(int j=0; j<10; j++)
for(int k=0; k<10; k++)
for(int l=0; l<3; l++)
  switch((*allAgentsPtr).at(i)->
afterPlayChar(j,k,l)){
    case'N':
      afterPlayStats[0][j][k][l]++;
      break;
    case'E':
      afterPlayStats[1][j][k][l]++;
      break;
    case'F':
      afterPlayStats[2][j][k][l]++;
      break;
    default:
      break;}
```

```
for(int j=0; j<3; j++)
for(int k=0; k<3; k++)
for(int l=0; l<3; l++)
  switch((*allAgentsPtr).at(i)->
afterPlayChar(j,k,l)){
    case'N':
      afterPlayStats[0][j][k][l]++;
      break;
    case'E':
      afterPlayStats[1][j][k][l]++;
      break;
    case'F':
      afterPlayStats[2][j][k][l]++;
      break;
    default:
      break;}
```

This set of nested loops collects information about the structure of the afterPlay matrix within the agent currently being examined and adds this to the appropriate value within the container that collects this information for the entire population.

```
for(int j=0; j<9; j++)
  if((*allAgentsPtr).at(i)->spawnPrefSet(j)==true)
    spawnPrefStats[j]++;
```

This for loop tests to see which spawning conditions are turned on within the agent currently being examined and adds any that are to the appropriate variable in the container that tracks this for the entire population.

```
for(int j=0; j<9; j++)
  if((*allAgentsPtr).at(i)->playPrefSet(j)==true)
    playPrefStats[j]++;
```

This for loop tests to see which playing conditions are turned on within the agent currently being examined and adds any that are to the appropriate variable in the container that tracks this for the entire population.

```
agentsKnown = agentsKnown+(*allAgentsPtr).at(i)->
agentsKnown();
```

Adds the number of agents know by the agent currently being examined to the global total.

```
for(int j=0; j<6; j++)
      likingCounts[j] += (*allAgentsPtr).at(i)->
likingStatsCount(j);
```

Adds the number of agents that the current agent being examined likes to the global totals.

```
if(socNet==true)
  for(int j=0; j<(int )allAgentsPtr->size();
j++){
    if((*allAgentsPtr).at(i)->agentKnow((
*allAgentsPtr).at(j)->agentID())==true){
      arcFile << (*allAgentsPtr).at(i)->agentID()
<< " ";
      arcFile << (*allAgentsPtr).at(j)->agentID()
<< " ";
      arcFile << (*allAgentsPtr).at(i)->
agentLike((*allAgentsPtr).at(j)->agentID(),
true);
      arcFile << " ["<< gen+1 << "]\r"<< endl;

      arcFile << (*allAgentsPtr).at(i)->agentID()
<< " ";
      arcFile << (*allAgentsPtr).at(j)->agentID()
<< " ";
      arcFile << (*allAgentsPtr).at(i)->
agentLike((*allAgentsPtr).at(j)->agentID(),
false)+1;
      arcFile << " ["<< gen+1 << "]\r"<< endl;}}}
```

In cases where social networking is turned on this block of code adds all the social information about this agent to the social networking file. The file produced can then be used by the social network visualization program Pajek (Slovenian for spider) to analyze the social environment within the population. This is not a feature that was used in the final version of this program due to the complexity this kind of analysis at the level of individuals within the very large populations being examined. It is, however, a feature that promises to be of great value in future iterations of the program.

This concludes the loop that examines each agent.

```
avAgeGen = (float)ageGen/(float)allAgentsPtr->
size();
avBitsSetGen = (float)bitsSetGen/(
(float)allAgentsPtr->size()*choiceSize);
```

Calculate the average age of the agents surviving into the next generation and the average number of bits set within these agents.

```
outFile0 << deadGen << ","<< bornGen << ",";
if(deadGen>0 && bornGen>0)
  outFile0 << (float)bornGen/(float)deadGen;
outFile0 << ",";
```

Send to the main summary file the number of agents who died in the current generation, the number of agents created in the current generation and, if both these values are greater than zero, the birth rate.

```
if((float)allAgentsPtr->size() > 0)
  outFile0 << avAgeGen;
```

If there are agents living within the population then output the average age of those agents.

```
outFile0 << ","<< maxAge << ",";
outFile0 << maxAgeCount << ",";
```

Output both the age of the oldest agents and the number of agents at this age.

```
if((float)allAgentsPtr->size() > 0)
  outFile0 << avBitsSetGen;
```

If there are agents living within the population then output the average number of bits set across the population.

```
outFile0 << ","<< allAgentsPtr->size() << ",";
```

Output the number of agents living in the population.

```
if((float)allAgentsPtr->size() > 0)
  outFile0 << (float)agentsKnown/
(float)allAgentsPtr->size();
outFile0 << ",";
```

If there are agents living within the population then output the average number of agents known within the population.

```
for(int j=0; j<6; j++){
  if((float)allAgentsPtr->size() > 0)
    outFile0 << (float)likingCounts[j]/
(float)allAgentsPtr->size();
  outFile0 << ",";}
```

Outputs the average number of ways that agents initially approach a relationship with a new agent if there are agents within the population to make such a calculation meaningful.

```
for(int h=0; h<3; h++){
  for(int j=0; j<10; j++)
  for(int k=0; k<10; k++)
  for(int l=0; l<3; l++){
    if((float)allAgentsPtr->size() > 0)
      outFile0 << afterPlayStats[h][j][k][l]/
(float)allAgentsPtr->size();
    outFile0 << ",";}}
```

```
for(int h=0; h<3; h++){
  for(int j=0; j<3; j++)
  for(int k=0; k<3; k++)
  for(int l=0; l<3; l++){
    if((float)allAgentsPtr->size() > 0)
      outFile0 << afterPlayStats[h][j][k][l]/
(float)allAgentsPtr->size();
    outFile0 << ",";}}
```

This block of code outputs the average number of agents expressing each possible afterPlay result if there are agents within the population to make this relevant.

```
for(int j=0; j<9; j++){
  if((float)allAgentsPtr->size() > 0)
    outFile0 << spawnPrefStats[j]/
(float)allAgentsPtr->size();
  outFile0 << ",";}
```

Output the average number of agents with each spawning preference if there are agents in the population to make this worth while.

```
for(int j=0; j<9; j++){
  if((float)allAgentsPtr->size() > 0)
    outFile0 << playPrefStats[j]/
(float)allAgentsPtr->size();
  outFile0 << ",";}
```

Output the average number of agents with each playing preference if there are agents in the population to make this worth while.

Create additional variables that will be used in assessing the bits with the highest and lowest frequencies so that this information can be collected in the outfile that holds the high level summary information.

```
int junkCounter=0;
```

junkCounter is an integer that will just hold a value briefly before it is used elsewhere.

```
floatjunkValue=0.0;
```

junkValue is a floating point value that will just hold a value briefly before it is used elsewhere.

```
vector<float> topBitProportions(10, 0.0);
```

topBitProportions is a vector of floating point values that will hold the population averages for bits with the top/bottom ten frequencies. It is initialized with 10 values each set to 0.0.

```
vector<int > topBitReferences(10, 0);
```

topBitReferences is a is a vector of integers that will hold the locations in the choice bitset of the bits with the top/bottom ten frequencies. It is initialized with 10 values, each of which is set to 0.

```
switch(g){
```

Again, which game agents have played plays a role in determining how much information to collect and output. Here a switch is called with game type (g) as the controlling variable.

```
case0:{
```

Opens the case where the game type is zero (all randomly generated games)

```
for(int j=0; j<choiceSize; j++){
  junkValue=(float)choiceTotals[j]/
(float)allAgentsPtr->size();

  junkCounter=0;
  while(junkCounter<10 && junkValue <
topBitProportions.at(junkCounter))
    junkCounter++;

  if(junkCounter<10){
    topBitProportions.insert(find
(topBitProportions.begin(),
topBitProportions.end(),topBitProportions.at
(junkCounter)),junkValue);
    topBitProportions.pop_back();
    topBitReferences.insert(find
(topBitReferences.begin(),topBitReferences.end(),
topBitReferences.at(junkCounter)),j);
    topBitReferences.pop_back();}}
```

This block of code loops through choiceTotals to find the largest values. It does this by comparing each value to those currently stored in topBitProportions. When it finds a larger value than any of the ten values currently held it inserts the new value into the vector in such a way that order is maintained. The largest element is then removed. topBitReferences is similarly updated.

```
for(int j=0; j<(int )topBitProportions.size();
j++){
  outFile0 << pos2bit(topBitReferences.at(j)) <<
","<< topBitProportions.at(j) << ",";}
```

Outputs the top ten bits and the percentage of the population expressing them.

```
for(int j=0; j<10; j++){
  topBitProportions.at(j)=0.0;
  topBitReferences.at(j)=0;}
```

Resets all the referencing and proportion information to zero in preparation for reusing these containers when collecting the behaviours that have come to be expressed with the lowest frequencies.

```
for(int j=0; j<choiceSize; j++){
  junkValue=1.0-((float)choiceTotals[j]/
(float)allAgentsPtr->size());
  junkCounter=0;
  while(junkCounter<10 && junkValue <
topBitProportions.at(junkCounter))
    junkCounter++;
  if(junkCounter<10){
    topBitProportions.insert(find(
topBitProportions.begin(),topBitProportions.end(),
topBitProportions.at(junkCounter)),junkValue);
    topBitProportions.pop_back();
    topBitReferences.insert(find(
topBitReferences.begin(),topBitReferences.end(),
topBitReferences.at(junkCounter)),j);
    topBitReferences.pop_back();}}
```

This block of code loops through choiceTotals to find the smallest values. It does this by comparing each value to those currently stored in topBitProportions. When it finds a smaller value than any of the ten values currently held it inserts the new value into the vector in such a way that order is maintained. The largest element is then removed. topBitReferences is similarly updated.

```
for(int j=0; j<(int )topBitProportions.size();
j++)
  outFile0 << pos2bit(topBitReferences.at(j)) <<
","<< 1.0-topBitProportions.at(j) << ",";
```

Outputs the top ten bits and the percentage of the population expressing them.

```
for(int j=0; j<choiceSize; j++){
  if(j <= 3180){
    if((float)allAgentsPtr->size() > 0)
      outFile1 << (float)choiceTotals[j]/
(float)allAgentsPtr->size();
    outFile1 << ",";}
  else
    if(j <= 6359){
      if((float)allAgentsPtr->size() > 0)
        outFile2 << (float)choiceTotals[j]/
(float)allAgentsPtr->size();
      outFile2 << ",";}
    else
      if(j <= 9538){
        if((float)allAgentsPtr->size() > 0)
        outFile3 << (float)choiceTotals[j]/
(float)allAgentsPtr->size();
        outFile3 << ",";}
      else{
        if((float)allAgentsPtr->size() > 0)
          outFile4 << (float)choiceTotals[j]/
(float)allAgentsPtr->size();
        outFile4 << ",";}}
```

```
for(int j=0; j<choiceSize; j++){
  if(j <= 3280){
    if((float)allAgentsPtr->size() > 0)
      outFile1 << (float)choiceTotals[j]/
(float)allAgentsPtr->size();
    outFile1 << ",";}
  else{
    if((float)allAgentsPtr->size() > 0)
      outFile2 << (float)choiceTotals[j]/
(float)allAgentsPtr->size();
    outFile2 << ",";}}
```

These sets of loops send the total proportion of the population expressing each bit to the outfiles for storage. NC agents have more outfiles, and thus more nested for-loops, because they have more cases that must be included in their behaviour set as a direct result of how they play games.

```
outFile1 << endl;
outFile2 << endl;
outFile3 << endl;
outFile4 << endl;
```

```
outFile1 << endl;
outFile2 << endl;
```

Finished the line in each of the outfiles and sets the insertion point to the beginning of the next line in each file. This prepares the file to receive input from the next generation that is to be recorded.

```
topBitReferences.clear();
topBitProportions.clear();}
break;
```

Clears the population proportion and bit reference trackers and then breaks the switch for the case of games of type 0.

```
case1:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++){
    if((float)allAgentsPtr->size() > 0)
      outFile0<< (float)choiceTotals[1357+j*1413+
i*4239]/(float)allAgentsPtr->size() << ",";
    outFile0 << ",";}}
  break;
```

```
case1:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++){
    if((float)allAgentsPtr->size() > 0)
      outFile0<< (float)choiceTotals[bit2pos(
PDbitBase(k)+j*4096+i*12288)]/
(float)allAgentsPtr->size();
    outFile0 << ",";}}
  break;
```

When only games of type 1 (Prisoner's Dilemmas) are played it is not necessary to collect information about all the behaviours that an agent is capable of expressing. In this case all behaviours except for those directly related to the Prisoner's Dilemma have no effect or influence on an agents ability to survive. Without a causal role the distribution of these behaviours across the population will never be anything but random and so can be ignored. These blocks of code ensure that only PD relevant behaviours are collected and sent to the outfiles.

The SQ and NC blocks differ for two reasons. First, the locations of the PD relevant behaviours are in a different location with the bitsets of each of these agent classes. Second, SQ class agents have a compression algorithm that allows for more efficient storage of the bits that code for various behaviours and special functions are required to unpack these locations.

```
case2:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++){
    if((float)allAgentsPtr->size() > 0)
      outFile0<< (float)choiceTotals[1181+j*1413+
i*4239]/(float)allAgentsPtr->size() << ",";
    outFile0 << ",";}}
  break;
```

```
case2:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++){
    if((float)allAgentsPtr->size() > 0)
      outFile0<< (float)choiceTotals[bit2pos(
SHbitBase(k)+j*4096+i*12288)]/
(float)allAgentsPtr->size();
    outFile0 << ",";}}
  break;
```



When only games of type 2 (Stag Hunts) are played it is not necessary to collect information about all the behaviours that an agent is capable of expressing. In this case all behaviours except for those directly related to the Stag Hunt have no effect or influence on an agents ability to survive. Without a causal role the distribution of these behaviours across the population will never be anything but random and so can be ignored. These blocks of code ensure that only SH relevant behaviours are collected and sent to the outfiles.

The SQ and NC blocks differ for two reasons. First, the locations of the SH relevant behaviours are in a different location with the bitsets of each of these agent classes. Second, SQ class agents have a compression algorithm that allows for more efficient storage of the bits that code for various behaviours and special functions are required to unpack these locations.

```
case3:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++){
    if((float)allAgentsPtr->size() > 0)
      outFile0<< (float)choiceTotals[408+j*1413+
i*4239]/(float)allAgentsPtr->size() << ",";
    outFile0 << ",";}}
  break;
```

```
case3:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++){
    if((float)allAgentsPtr->size() > 0)
      outFile0<< (float)choiceTotals[bit2pos(
CKbitBase(k)+j*4096+i*12288)]/
(float)allAgentsPtr->size();
    outFile0 << ",";}}
  break;
```

When only games of type 3 (Chicken) are played it is not necessary to collect information about all the behaviours that an agent is capable of expressing. In this case all behaviours except for those directly related to Chicken have no effect or influence on an agents ability to survive. Without a causal role the distribution of these behaviours across the population will never be anything but random and so can be ignored. These blocks of code ensure that only CK relevant behaviours are collected and sent to the outfiles.

The SQ and NC blocks differ for two reasons. First, the locations of the CK relevant behaviours are in a different location with the bitsets of each of these agent classes. Second, SQ class agents have a compression algorithm that allows for more efficient storage of the bits that code for various behaviours and special functions are required to unpack these locations.

```
case4:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++){
    if((float)allAgentsPtr->size() > 0)
      outFile0<< (float)choiceTotals[627+j*1413+
i*4239]/(float)allAgentsPtr->size() << ",";
    outFile0 << ",";}}
  break;}
```

```
case4:{
  for(int i=0;i<3;i++)
  for(int j=0;j<3;j++)
  for(int k=0;k<4;k++){
    if((float)allAgentsPtr->size() > 0)
      outFile0<< (float)choiceTotals[bit2pos(
BSbitBase(k)+j*4096+i*12288)]/
(float)allAgentsPtr->size();
    outFile0 << ",";}}
  break;}
```

When only games of type 4 (Battles of the Sexes) are played it is not necessary to collect information about all the behaviours that an agent is capable of expressing. In this case all behaviours except for those directly related to the Battle of the Sexes have no effect or influence on an agents ability to survive. Without a causal role the distribution of these behaviours across the population will never be anything but random and so can be ignored. These blocks of code ensure that only BS relevant behaviours are collected and sent to the outfiles.

The SQ and NC blocks differ for two reasons. First, the locations of the BS relevant behaviours are in a different location with the bitsets of each of these agent classes. Second, SQ class agents have a compression algorithm that allows for more efficient storage of the bits that code for various behaviours and special functions are required to unpack these locations.

```
outFile0 << endl;}
```

This ends the line in outFile 0 and resets the insertion point to the beginning of the next line. Once there is nothing more to do regarding collecting statistics for the current generation and the summary collection function closes.

### E.3.18 agent.h

Note that any appearance of SQ agent being more complicated than NC agents is a result of much of the complexity of NC agents being loaded into the game objects that they interact with.

| | |
|---|---|
| `#include<map>` | Includes the map library when compiling the code related to agents. Maps are collections of data that can be accessed via a key. |
| `#include<bitset>` | Includes the bitset library when compiling the code related to agents. Bitsets are just lists of bits that allow for compact storage and fast access of binary styled data. |
| `#include<gsl/gsl_rng.h>` | Includes the gnu scientific library when compiling the code related to agents. This is necessary to make use of the random number generators available within this library. |
| `#include<vector>` | Includes the vector library when compiling the code related to agents. Vectors are lists that allow for complicated pieces of information to be inserted and removed. |
| `#include"game.h"` | Includes the header information for game object when compiling the code related to agents. This allows functions within agent objects to interact with agent objects. |
| `usingnamespacestd;` | Simplifies coding by preventing the need to specify which namespace various elements of the code are being drawn from. Everything used here comes from the standard namespace and this line declares that. |
| `classagent{` | Begins the description of the agent class. |

| | |
|---|---|
| `public:` | Designates that all the functions that follow are public in the sense that they may be called by any other object or function. |
| `agent(const int , const int , const bool, const bool, const int ,const int , gsl_rng*, const int );`<br>`~agent();` | The constructor and destructor for the agent class. The constructor takes eight inputs. The destructor, designated by the '~' takes no inputs. |
| `boolplay(game*const , const int , const bool);` | This sets up the play function which says what an agent would do in a specific situation. It takes as inputs the details of the game and the other player and returns a boolean that corresponds to a play of either Top/Left or Bottom/Right. |
| `int agentID();` | Returns the ID number of the agent. |
| `void updateAgent(game*const , const int , int , int , const bool);` | Updates the agent after a game has been played. The details of the outcome are taken as inputs. Nothing is returned since only internal changes to the agent result. |
| `void printAgent();` | Prints the details of the agent on screen. No inputs are required and no output value is returned given. |
| `int agentRes();` | Returns the total number of resource points that the agent has collected. |
| `void setRes(int );` | Modifies the number of resource points an agent has to the value passed to the function when it is called. Nothing is returned. |

| | |
|---|---|
| `void agentSpawn(agent*, agent*, gsl_rng*);` | Modifies a newly created agent so that they reflect the make-up of the two agents that collaborated to bring the new agent into existence. Pointers to the parent agents and to the random number generator are passed to the function. Nothing is returned since only changes to the agent result. |
| `boolchoiceBitTest(int );` | Tests whether a bit within the bitset choice as designated by integer value passed to the function is set to 1 or zero. An appropriate boolean value is returned. |
| `unsignedcharafterPlayChar(const int , const int , const int );` | Returns the agent's response to outcomes of the sort described by the three input integers. The output is in the form of a character that codes for the new relationship status that will be assessed after playing with another agent under the given condition. |
| `int agentAge();` | Returns the age of the agent in generations. |
| `void agentAgePlus();` | Increments the current age of the agent. |
| `int agentBirthday();` | Returns the generation in which the age was created. |
| `int agentBitsSet();` | Returns the total number of bits set to "on" or 1 within the bitset choice. |
| `int afterPlayValues(const int , const int , const int );` | |

| | |
|---|---|
| `int choiceSize();` | Returns the size of the bitset choice. Used in debugging. |
| `boolspawnAgree(const int );` | Returns a bool corresponding to whether the agent will spawn with the agent whose ID is passed or not. |
| `boolspawnPrefSet(const int );` | Returns a bool corresponding to whether or not a specific bit is set within spawnPref. |
| `boolplayAgree(const int );` | Returns a bool corresponding to whether the agent will play with the agent whose ID is passed or not. |
| `boolplayPrefSet(const int );` | Returns a bool corresponding to whether or not a specific bit is set within playPref. |
| `unsignedchargetDefaultMLU();` | Returns the liking category which the agent assigns agents that they have never met before to. |
| `unsignedchargetDefaultULM();` | Returns the liking category which the agent believes that other agents that assign them to when they have never met before to. |
| `boolagentKnow(const int );` | Returns a bool indicating whether the agent knows the agent indicated by the integer passed to the function. |
| `shortagentLike(const int , const bool);` | Allows specific likings for agents in know to be discovered for arc creation. This is used for social network mapping and is turned off for all the trials run in this project. |

| | |
|---|---|
| `int agentsKnown();` | Returns the number of other agents that the agent has recorded inside the map know. |
| `int likingStatsCount(const int );` | Returns the number of likings of a the type indicated by the integer passed to the function that an agent has. |
| `void cleanKnow(vector<int >*, const int );` | Updates know to reflect shifts to a new generation by removing references to dead agents or those who haven't been seen for a long time. |
| `private:` | Designates that all the functions that follow are private in the sense that they may only be called or interacted with directly by the agent itself. |
| `int bitsSet;` | An integer designating the number of bits set within the bitset choice. |
| `int id;` | An integer indicating the ID number of the agent. |
| `shortbirthday;` | A short integer indicating the generation in which the agent was created. |
| `shortage;` | A short integer indicating the number of generations that the agent has been alive. |
| `unsignedcharafterPlay[10][10][3];` | `unsignedcharafterPlay[3][3][3];` |

This matrix captures how an agent will take various outcomes following the playing of a game and the assignment of payoffs. The agent updates how much they like the other agent based on the combination of whether or not they do better, worse, or the same from the status quo in combination with how the other player does wrt the status quo as well. Unsigned chars are used to save space over ints. The first index value says how the agent fared against the status quo. The second index value says how the other agent fared against the status quo. The third index value says what the agent thought of the other agent prior to playing There are three possible values that can appear in the array:

F - make opponent a "friend"

N - make opponent a "neutral party" (Note that any non F or E value is counted as an N)

E - make opponenet an "enemy"

```
map<unsignedint , pair<pair<unsignedchar,
unsignedchar>, unsignedshort> > know;
```

If agents know anything about their opponent it is represented in a map. This map takes the opponent's ID as the key and returns a pair of chars and an unsigned short integer. The chars correspond to the extension of the relationship from each agent to the other and the short integer is the number of generations since the agent was last seen.

```
unsignedshortmaxAbsent;
```

An unsigned short indicating how long another agent may remain in the agent's memory (the map know) before being forgotten (removed).

```
bitset<12717> choice;
```

```
bitset<6561> choice;
```

Creates a bitset called choice that holds the responses of the agent to all the possible situations that they might be found in. A scenario is defined by a perspective on a game and the two projections of the relationship that the agent assesses as existing between them and the other agent. NC class agents have 12717 bits because there 1413 different perspectives that they can have on a game and nine ways of combining the relationship assessments. SQ class agents have 6561 bits because there are 729 different perspectives that they can have and nine ways of combining the relationships.

```
int resource;
```

Creates an integer variable that will store the total number of points held by the agent that may be used as resources for paying the ante or for spawning.

```
bitset<9> spawnPref;
```

Controls whether or not an agent will spawn with other agents depending on how those agents are liked. The first group of three bits sets 'E', the second 'N' and the third 'F' for the projection of the relationship from the agent to the other agent. Within these sets the first bit sets for 'E', the second for 'N', and the third for 'F'. A set bit means an agent will spawn with agents with that property.

```
bitset<9> playPref;
```

Controls whether or not an agent will spawn with other agents depending on how those agents are liked. The first group of three bits sets 'E', the second 'N' and the third 'F' for the projection of the relationship from the agent to the other agent. Within these sets the first bit sets for 'E', the second for 'N', and the third for 'F'. A set bit means an agent will play with agents with that property.

```
unsignedchardmLu, duLm;
```

Creates a set of unsigned chars that will hold the default values for how the agent assess their relationship with mew agents.
dmLu = default me likes you
duLm = default you likes me

```
int bit2pos(const int );
int pos2bit(const int );
```

Creates a set of functions to handle indexing within the bitset choice to accommodate the compression that has been built into it. Without these functions there would be many wasted bits.

```
int likingStats[6];
```

Creates an array of integers that track how many agents the agent likes in each of the three possible ways and how many agents the agent think likes them in each of the three possible ways.

agent.cpp

```
#include<iostream>
```

Includes the library that allows data to input from the keyboard and to be output on the screen. Only the output capabilities are used.

```
#include"agent.h"
```

Includes the agent header file where all the functions included here are defined.

```
#include<gsl/gsl_rng.h>
```

Includes the Gnu Scientific Library random number generator library.

```
#include<algorithm>
```

Includes the algorithm library which holds all the search functions that can be used with the C++ standard data types.

```
usingnamespacestd;
```

Simplifies coding by preventing the need to specify which namespace various elements of the code are being drawn from. Everything used here comes from the standard namespace and this line declares that.

```
agent::agent(const int spawnDisc, const int
playDisc, const boolrndLike, const boolrndAft,
const int genNum, const int mA,  gsl_rng* r,
const int agentCount)
{
```

Opens the agent constructor.

```
id= agentCount;
```

Sets the ID of the agent to the current count of the number of agents.

```
birthday=genNum;
```

Sets the birthday of the agent to the current generation number.

```
age= 0;
```

Sets the age of the agent to zero.

```
resource= 0;
```

Sets the number of resources of the agent to zero.

```
maxAbsent= mA;
```

Sets the number of generations that another agent can go unseen before being removed from the agent's memory to the global limit for this, mA.

```
bitsSet=0;
```

Sets the total number of bits set within the bitset choice to zero.

```
for(int i=0; i<(int )choice.size(); i++)
  if(gsl_rng_uniform(r) <= 0.5){
    choice.set(i);
    bitsSet++;}
```

Randomizes the setting of all the bits within the bitset choice.

```
for(int i=0; i<10; i++)
for(int j=0; j<10; j++)
for(int k=0; k<3; k++){
  if(rndAft==true){
    if(gsl_rng_uniform(r) < 1.0/3.0)
      afterPlay[i][j][k]='E';
    else{
      if(gsl_rng_uniform(r) > 0.5)
        afterPlay[i][j][k]='N';
      else
        afterPlay[i][j][k]='F';}}
  else
    afterPlay[i][j][k]='N';}
```

```
for(int i=0; i<3; i++)
for(int j=0; j<3; j++)
for(int k=0; k<3; k++){
  if(rndAft==true){
    if(gsl_rng_uniform(r) < 1.0/3.0)
      afterPlay[i][j][k]='E';
    else{
      if(gsl_rng_uniform(r) > 0.5)
        afterPlay[i][j][k]='N';
      else
        afterPlay[i][j][k]='F';}}
  else
    afterPlay[i][j][k]='N';}
```

Sets up the afterPlay matrix of the agent. If the variable rndAft is set to true then the matrix is randomized with each possibility receiving either an E, N, or F with equal probability. If rndAft is set to false then every possibility receives the same value, N.

```
if(spawnDisc>0){
  for(int i=0; i<int (spawnPref.size()); i++)
    if(gsl_rng_uniform(r) <= 0.5)
      spawnPref.set(i);}
else{
  for(int i=0; i<int (spawnPref.size()); i++)
    spawnPref.set(i);}
```

Sets up how an agent responds to requests to spawn. If the variable spawnDisc is greater that zero then the bits within the bitset are randomly set to be on or off. If spawnDisc is not greater than zero then all the bits within the bitset spawnPref are turned on (the agent will spawn with anyone under any conditions).

```
if(playDisc>0){
  for(int i=0; i<int (playPref.size()); i++)
    if(gsl_rng_uniform(r) <= 0.5)
      playPref.set(i);}
else{
  for(int i=0; i<int (playPref.size()); i++)
    playPref.set(i);}
```

Sets up how an agent responds to requests to play. If the variable playDisc is greater that zero then the bits within the bitset are randomly set to be on or off. If playDisc is not greater than zero then all the bits within the bitset playPref are turned on (the agent will spawn with anyone under any conditions).

```
if(rndLike==true){
  if(gsl_rng_uniform(r) < 1.0/3.0)
    dmLu='E';
  else
    if(gsl_rng_uniform(r) > 0.5)
      dmLu='N';
    else
      dmLu='F';

  if(gsl_rng_uniform(r) < 1.0/3.0)
    duLm='E';
  else
    if(gsl_rng_uniform(r) > 0.5)
      duLm='N';
    else
      duLm='F';}
else{
  dmLu='N';
  duLm='N';}
```

If the variable rndLike is set to true then the default likings of the agent are randomized with equal probability between E, F, and N. If rndLike is set to false then both dmLu and duLm are set to N.

```
for(int i=0; i<6; i++)
  likingStats[i]=0;
}
```

Ensures that the counts for all the cases where an agent likes another agent in a certain way or thinks that they are liked in turn in a certain way are zero (the agent hasn't met anyone yet).
This is the last thing to be done in the agent constructor

```
boolagent::play(game*const gamePtr, const int
opID, const boolrow){
```

Opens the play function which directs an agent's play of a game.

```
int mLu=-1;
int uLm=-1;
```

Initializes a pair of variables that will track exactly what the relationship the agent perceives as being the relationship between themselves and the other agent that they have been matched with.

```
if(know.count(opID)==0){
```

Opens a conditional that tests to see if the agent has meet the agent they have been matched with before. If the agent has no record of the other agent then the following actions are carried out.

```
know.insert(make_pair(opID,make_pair(
make_pair(dmLu,duLm),0)));
```

A place for the new agent is created inside the map know.

```
switch(dmLu){
  case'E':
    likingStats[1]++;
    break;
  case'F':
    likingStats[2]++;
    break;
  case'N':
    likingStats[0]++;
    break;}
```

The default value for extending the relationship from the agent to the new agent is given for the relationship with the new agent.

```
switch(duLm){
  case'E':
    likingStats[4]++;
    break;
  case'F':
    likingStats[5]++;
    break;
  case'N':
    likingStats[3]++;
    break;}}
```

The default value for extending the relationship from the new agent to the agent is given for the relationship with the new agent.

```
if(know.at(opID).first.first=='E')
  mLu=1;
else{
  if(know.at(opID).first.first=='F')
    mLu=2;
  else
    mLu=0;}
```

Finds out how the agent extends the relationship from themselves to the other agent and stores a corresponding value in the variable mLu.

```
if(know.at(opID).first.second=='E')
  uLm=1;
else{
  if(know.at(opID).first.second=='F')
    uLm=2;
  else
    uLm=0;}
```

Finds out how the agent extends the relationship from the other agent to themselves and stores a corresponding value in the variable uLm.

```
know.at(opID).second=maxAbsent;
```

Resets the counter that tracks how long it has been since the agent has been seen.

```
int gamePer=-1;
if(row==true)
  gamePer=gamePtr->getGameIndex(true);
else
  gamePer=gamePtr->getGameIndex(false);
int choiceIndex=-1;
choiceIndex = gamePer+mLu*1413+uLm*4239;
```

```
int a, b, c, d, e, f, g, h, i, j, k, l=-1;
if(row==true){
  a=gamePtr->payBits(0);
  b=gamePtr->payBits(1);
  c=gamePtr->payBits(2);
  d=gamePtr->payBits(3);
  e=gamePtr->payBits(4);
  f=gamePtr->payBits(5);
  g=gamePtr->payBits(6);
  h=gamePtr->payBits(7);
  i=gamePtr->payBits(8);
  j=gamePtr->payBits(9);
  k=gamePtr->payBits(10);
  l=gamePtr->payBits(11);}
else{
  g=gamePtr->payBits(0);
  h=gamePtr->payBits(1);
  i=gamePtr->payBits(2);
  j=gamePtr->payBits(3);
  k=gamePtr->payBits(4);
  l=gamePtr->payBits(5);
  a=gamePtr->payBits(6);
  b=gamePtr->payBits(7);
  c=gamePtr->payBits(8);
  d=gamePtr->payBits(9);
  e=gamePtr->payBits(10);
  f=gamePtr->payBits(11);}
int choiceIndex=-1;
choiceIndex = l+k*2+j*4+i*8+h*16+g*32+f*64+e*128+
d*256+c*512+b*1024+a*2048+mLu*4096+ uLm*12288;
choiceIndex = bit2pos(choiceIndex);
```

Determines which bit within the bitset `choice` codes for the behaviour that the agent expresses under the current scenario. The differences in the code between the two agent types is a result of both how each type makes a decision and the storage efficiencies that have been introduced into SQ agents.

```
if(choice.test(choiceIndex))
  returntrue;
else
  returnfalse;}
```

This conditional tests the bit within the bitset choice that corresponds to scenario currently being faced. The conditonal returns true if the appropriate bit is set to 1 or 'on' and false if it is set to 0 or 'off'.

```
int agent::agentID(){
  returnid;}
```

Returns the ID number of the agent.

```
int agent::agentRes(){
  returnresource;}
```

Returns the total number of resources held by the agent.

```
void agent::setRes(int newResource){
  resource= newResource;}
```

Sets the resources of the agent to the value passed to the function.

```
void agent::printAgent(){
```

Opens the function that will output agent information on the screen.

```
cout << "Agent ID#"<< id<< endl;
cout << "Age:  "<< age<< endl;
cout << "Choice Matrix"<< endl;
```

Outputs the ID number and age of the agent with appropriate titles. Also outputs the title for the choice matrix.

```
for(int i=0; i<(int )choice.size(); i++){
  cout << choice[i];}
```

This for-loop outputs the complete bitset choice as a set of ones and zeroes.

```
cout << endl << endl << "AfterplayMatrix"<< endl;
for(int i=0; i<10; i++){
  for(int j=0; j<10; j++)
  for(int k=0; k<3; k++){
    cout << afterPlay[i][j][k] << "  ";}

  cout << endl;}
cout << endl << endl;}
```

```
cout << endl << endl << "AfterplayMatrix"<< endl;
for(int i=0; i<3; i++){
  for(int j=0; j<3; j++)
  for(int k=0; k<3; k++){
    cout << afterPlay[i][j][k] << "  ";}

  cout << endl;}
cout << endl << endl;}
```

Outputs the `afterPlay` matrix for each agent. This is the last thing printed.

```
void agent::updateAgent(game*const gamePtr, const
int opID, int myPay, int otherPay, const boolrow)
{
```

Opens the function that updates agents after playing a game with another agent.

```
resource= resource+ myPay;
```

Adds whatever payoff the agent received from playing the game to their resource total.

```
int mBWS=0;
int uBWS=0;
int PL=0;
```

Creates three integer variables and initializes them to zero. The first two will be used to track how each agent did in comparison to some benchmark. For SQ agents it is the status quo payoff. For NC agents the comparison is made against the best value being offered.
BWS=Better Worse Same, u = you, m = me.
The third will hold a value corresponding to the way that they agent like the other agent *before* the game was played.
PL=Previous Liking.

```
if(row==true){
  mBWS = gamePtr->outRanks[0];
  uBWS = gamePtr->outRanks[1];}
else{
  mBWS = gamePtr->outRanks[1];
  uBWS = gamePtr->outRanks[0];}
```

```
if(row==true){
  mBWS = gamePtr->sqPays[0]-myPay;
  uBWS = gamePtr->sqPays[1]-otherPay;}
else{
  mBWS = gamePtr->sqPays[1]-myPay;
  uBWS = gamePtr->sqPays[0]-otherPay;}
```

Sets the values for mBWS and uBWS. The conditional is needed because these values swap depending on whether the assessment is being made for the row or the the column player. The differences between the agent types is a result of the different types of comparisons that they are making.

```
if(mBWS > 0)
  mBWS = 0;
else
  if(mBWS < 0)
    mBWS = 2;
  else
    mBWS = 1;

if(uBWS > 0)
  uBWS = 0;
else
  if(uBWS < 0)
    uBWS = 2;
  else
    uBWS = 1;
```

SQ type agents need a little more work to get the BWS variables adjusted to properly reflect the state of affairs set. NC type agents have this done for them already within the games that they play.

```
if(know.at(opID).first.first=='E')
  PL=1;
else{
  if(know.at(opID).first.first=='F')
    PL=2;
  else
    PL=0;}
```

Finds the previous liking of the agent for the other agent and assigns a numerical value to PL in response to this.

```
if(know.at(opID).first.first!=afterPlay[mBWS]
[uBWS][PL]){
  if(know.at(opID).first.first=='E')
    likingStats[1]--;
  else
    if(know.at(opID).first.first=='F')
      likingStats[2]--;
    else
      likingStats[0]--;
  if(afterPlay[mBWS][uBWS][PL]=='E')
    likingStats[1]++;
  else
    if(afterPlay[mBWS][uBWS][PL]=='F')
      likingStats[2]++;
    else
      likingStats[0]++;
  know.at(opID).first.first=afterPlay[mBWS]
[uBWS][PL];}
```

If the way the agent liked the other agent before playing is not the same as after playing then some updating occurs. During the update a new value is assigned to how the agent likes the other agent and the counts of the total number of ways that an agent likes another are updated accordingly.

```
if(know.at(opID).first.second=='E')
  PL=1;
else{
  if(know.at(opID).first.second=='F')
    PL=2;
  else
    PL=0;}
```

Finds the previous liking of that the agent believes that the other agent assigns to them and assigns a numerical value to PL in response to this.

```
if(know.at(opID).first.second!=afterPlay[uBWS]
[mBWS][PL]){
  if(know.at(opID).first.second=='E')
    likingStats[4]--;
  else
    if(know.at(opID).first.second=='F')
      likingStats[5]--;
    else
      likingStats[3]--;
  if(afterPlay[uBWS][mBWS][PL]=='E')
    likingStats[4]++;
  else
    if(afterPlay[mBWS][uBWS][PL]=='F')
      likingStats[5]++;
    else
      likingStats[3]++;
  know.at(opID).first.second=afterPlay[uBWS]
[mBWS][PL];}}
```

If the way the agent thinks the other agent liked them before playing is not the same as after playing then some updating occurs. During the update a new value is assigned to how the agent thinks that the other agent likes them and the counts of the total number of ways that an agent likes another are updated accordingly.

This closes the update function.

```
void agent::agentSpawn(agent* mom, agent* dad,
gsl_rng* r){
```

Opens the function for agent spawning. This function takes a newly created agent and modifies it to reflect parental influences.

```
for(int i=0; i<(int )choice.size(); i++){
  if(gsl_rng_uniform(r)>0.5){
    if(mom->choiceBitTest(i))
      choice.set(i);
    else
      choice.reset(i);}
  else{
    if(dad->choiceBitTest(i))
      choice.set(i);
    else
      choice.reset(i);}}
```

Runs through the bitset choice. Each bit is taken from one parent or the other with equal probability.



```
for(int i=0; i<10; i++)
for(int j=0; j<10; j++)
for(int k=0; k<3; k++){
  if(gsl_rng_uniform(r)>0.5)
    afterPlay[i][j][k]= mom->
afterPlayChar(i,j,k);
  else
    afterPlay[i][j][k]= dad->
afterPlayChar(i,j,k);}
```

```
for(int i=0; i<3; i++)
for(int j=0; j<3; j++)
for(int k=0; k<3; k++){
  if(gsl_rng_uniform(r)>0.5)
    afterPlay[i][j][k]= mom->
afterPlayChar(i,j,k);
  else
    afterPlay[i][j][k]= dad->
afterPlayChar(i,j,k);}
```

Runs through the afterPlay matrix. Each value is taken from one parent or the other with equal probability.

```
for(int i=0; i<int (spawnPref.size()); i++){
  if(gsl_rng_uniform(r) <= 0.5){
    if(mom->spawnPrefSet(i))
      spawnPref.set(i);
    else
      spawnPref.reset();}
  else{
    if(dad->spawnPrefSet(i))
      spawnPref.set(i);
    else
      spawnPref.reset();}}
```

Runs through the spawnPref bitset. Each value is taken from one parent or the other with equal probability.

```
for(int i=0; i<int (playPref.size()); i++){
  if(gsl_rng_uniform(r) <= 0.5){
    if(mom->playPrefSet(i))
      playPref.set(i);
    else
      playPref.reset();}
  else{
    if(dad->playPrefSet(i))
      playPref.set(i);
    else
      playPref.reset();}}
```

Runs through the playPref bitset. Each value is taken from one parent or the other with equal probability.

```
if(gsl_rng_uniform(r) <= 0.5)
  dmLu=mom->getDefaultMLU();
else
  dmLu=dad->getDefaultMLU();

if(gsl_rng_uniform(r) <= 0.5)
  duLm=mom->getDefaultULM();
else
  duLm=dad->getDefaultULM();}
```

Takes the values for dmLu and duLm with from each parent with equal probability.

```
boolagent::choiceBitTest(int bit){
  if(choice.test(bit))
    returntrue;
  else
    returnfalse;}
```

Returns whether or not a bit is set within the bitset choice.

```
unsignedcharagent::afterPlayChar(const int i,
const int j, const int k){
  returnafterPlay[i][j][k];}
```

Returns the value within afterPlay matrix corresponding to the values passed.

```
int agent::afterPlayValues(const int j, const int
k, const int l){
  int apv=0;
  switch(afterPlay[j][k][l]){
    case'F':
      apv=1;
      break;
    case'E':
      apv=-1;
      break;
    default:
      break;}
  returnapv;}
```

Returns a value from the afterPlay matrix instead of the character corresponding to the values passed to the function.

```
int agent::agentAge(){
  returnage;}
```

Returns the age of the agent.

```
void agent::agentAgePlus(){
  age++;}
```

Increments the age of the agent.

```
int agent::agentBirthday(){
  returnbirthday;}
```

Returns the birthday of the agent.

```
int agent::agentBitsSet(){
  returnbitsSet;}
```

Returns a count of the total number of bits set to one inside the bitset choice.

```
int agent::choiceSize(){
  returnint (choice.size());}
```

Returns the size of the bitset choice.

```
boolagent::spawnAgree(const int opID){
```

Opens the function that determines if an agent will spawn with another.

```
if(know.count(opID)==0)
    know.insert(make_pair(opID,make_pair
(make_pair(dmLu,duLm),0)));
```

Tests whether or not the agent to be spawned with is already stored inside the map know. If they are not then a new record is generated and inserted into the map with the default values.

```
switch(know.at(opID).first.first){
  case'E':
    switch(know.at(opID).first.second){
      case'E':
        if(spawnPref.test(0)==true)
          return true;
        break;
      case'F':
        if(spawnPref.test(2)==true)
          return true;
        break;
      case'N':
        if(spawnPref.test(1)==true)
          return true;
        break;}
```

This switch is the core of the `spawnAgree` function. It amounts to a test of the `spawnPref` bitset to see if the relationship assessed by the agent as existing between them and the possible spawning partner is such that they will allow spawning to take place.
For clean formatting it has been split over this block of code plus two more.

```
  case'F':
    switch(know.at(opID).first.second){
      case'E':
        if(spawnPref.test(6)==true)
          return true;
        break;
      case'F':
        if(spawnPref.test(8)==true)
          return true;
        break;
      case'N':
        if(spawnPref.test(7)==true)
          return true;
        break;}
```

Block two of the switch.

```
  case'N':
    switch(know.at(opID).first.second){
      case'E':
        if(spawnPref.test(3)==true)
          return true;
        break;
      case'F':
        if(spawnPref.test(5)==true)
          return true;
        break;
      case'N':
        if(spawnPref.test(4)==true)
          return true;
        break;}}
return false;}
```

Block three of the switch.

```
boolagent::spawnPrefSet(const int bit){
  if(spawnPref.test(bit)==true)
    returntrue;
  else
    returnfalse;}
```

Tests to see whether or not a given bit within the bitset spawnPref is set or not.

```
boolagent::playAgree(const int opID){
```

Opens the function that determines if an agent will play with another.

```
if(know.count(opID)==0)
  know.insert(make_pair(opID,make_pair(make_pair(
dmLu,duLm),0)));
```

Tests whether or not the agent to be played with is already stored inside the map know. If they are not then a new record is generated and inserted into the map with the default values.

```
switch(know.at(opID).first.first){
  case'E':
    switch(know.at(opID).first.second){
      case'E':
        if(playPref.test(0)==true)
          return true;
        break;
      case'F':
        if(playPref.test(2)==true)
          return true;
        break;
      case'N':
        if(playPref.test(1)==true)
          return true;
        break;}
```

This switch is the core of the `playAgree` function. It amounts to a test of the `playPref` bitset to see if the relationship assessed by the agent as existing between them and the possible spawning partner is such that they will allow spawning to take place.
For cleaner presentation this block of code is split between this block and the next two.

```
  case'F':
    switch(know.at(opID).first.second){
      case'E':
        if(playPref.test(6)==true)
          return true;
        break;
      case'F':
        if(playPref.test(8)==true)
          return true;
        break;
      case'N':
        if(playPref.test(7)==true)
          return true;
        break;}
```

This is block two of the switch.

```
  case'N':
    switch(know.at(opID).first.second){
      case'E':
        if(playPref.test(3)==true)
          return true;
        break;
      case'F':
        if(playPref.test(5)==true)
          return true;
        break;
      case'N':
        if(playPref.test(4)==true)
          return true;
        break;}}
return false;}
```

This is block three of the switch.

```
boolagent::playPrefSet(const int bit){
  if(playPref.test(bit)==true)
    returntrue;
  else
    returnfalse;}
```

Tests to see whether or not a given bit within the bitset playPref is set or not.

```
unsignedcharagent::getDefaultMLU(){
  returndmLu;}
```

Returns an agent's default value for the way they extend a relationship from themselves to another agent.

```
unsignedcharagent::getDefaultULM(){
  returnduLm;}
```

Returns an agent's default value for the way they extend a relationship from another agent to themselves.

```
boolagent::agentKnow(const int testAgent){
  if(know.count(testAgent)>=1)
    return1;
  else
    return0;}
```

Returns a bool corresponding to whether or not the agent has a record for the agent whose ID was passed to the function inside the map know.

```
shortagent::agentLike(const int knowPos, const
boolfs){
  if(fs==true)
    if(know.at(knowPos).first.first=='E')
      return1;
    else
      if(know.at(knowPos).first.first=='F')
        return5;
      else
        return3;
  else
    if(know.at(knowPos).first.second=='E')
      return2;
    else
      if(know.at(knowPos).first.second=='F')
        return6;
      else
        return4;}
```

Returns a value corresponding to how the agent sees the relationship between themselves and another agent.

```
int agent::agentsKnown(){
  return(int )know.size();}
```

```
int agent::bit2pos(const int bit){
  int q1=0, q2=0, q3=0, q4=0, q5=0, q6=0;
  q1=bit/4;
  q2=bit/16;
  q3=bit/64;
  q4=bit/256;
  q5=bit/1024;
  q6=bit/4096;
  returnbit-q1-q2*3-q3*9-q4*27-q5*81-q6*243;}
```

Coverts a binary value into a position within the bitset choice.

```
int agent::pos2bit(const int position){
  int q1=0, q2=0, q3=0, q4=0, q5=0, q6=0;
  q1=position/3;
  q2=position/9;
  q3=position/27;
  q4=position/81;
  q5=position/243;
  q6=position/729;
  return position+q1+q2*4+q3*16+q4*64+q5*256+
q6*1024;}
```

Converts a position within the bitset choice into a binary value.

```
int agent::likingStatsCount(const int index){
  returnlikingStats[index];}
```

Returns the value stored in the array likingStats corresponding to the number passed to the function.

```
void agent::cleanKnow(vector<int >* deadIDsPtr,
const int agentCount){
  for(int i=0; i<agentCount; i++){
    if(know.count(i)>0){
      if( (int ) count(deadIDsPtr->begin(),
deadIDsPtr->end(), i) > 0 ||
know.at(i).second==0)
        know.erase(i);
      else
        know.at(i).second--;}}}
```

Removes from the agent's know map the agent that is passed to the function.

```
agent::~agent(){}
```

The destructor for an agent.



### E.3.19 game.h

```
#include<bitset>
```

Includes the library that holds all the information for working with the C++ standard library data type known as a bitset.

```
#include<utility>
```

Includes the library that holds the utility functions.

```
#include<gsl/gsl_rng.h>
```

Includes the library for the Gnu Scientific Library random number generators.

```
#include<vector>
```

Includes the library that allows for the the handling of the C++ standard library container known as a vector. Only NC type agents need this library because only NC type agents need vectors when handling game objects.

| | |
|---|---|
| `usingnamespacestd;` | Simplifies coding by preventing the need to specify which namespace various elements of the code are being drawn from. Everything used here comes from the standard namespace and this line declares that. |
| `classgame{` | Begins the description of the game class. |
| `friendclassagent;` | Declares the class agent to be a friend. This means that agent objects can have more direct access to the functions and variables within game objects. |
| `public:` | Declares the functions and variables that follow to be public in the sense that they can be accessed by any other object. |
| `game(const int , gsl_rng*);`<br>`~game();` | The game constructor and destructor. |
| `int outcome(const bool, const bool, const bool);` | Returns the outcome payoff of a game to a player given the moves of the two players and whether or not the payoff to be returned is for the Row or the Column player. |
| `int  payoff(bool, bool, bool);` | |

Returns the payoff corresponding to the set of boolean values passed to the function. The booleans designate the row play, the column play, and the player for whom the payoff to be returned is for.

| | |
|---|---|
| `void printGame();` | Prints the game and associated information on the screen. |

| | |
|---|---|
| | `int payBits(const int );` |

Returns information on whether or not a payoff is better or worse than another value.

| | |
|---|---|
| `int getPaySum();` | Returns the sum of all the payoffs offered by the game. Used to ensure that the random number generator is working correctly. |

| | |
|---|---|
| `int getGameIndex(const bool);` | |

Returns the index position of the game within GAMENUMBERS.DAT.

| | |
|---|---|
| `static void loadGameNumbers();` | |

Loads all the game numbers from the file GAMENUMBERS.DAT into memory for quick reference.

| | |
|---|---|
| `private:` | Declares that all functions and variables that follow are private such that they can only be accessed directly by the game object that owns them. |
| `int gameType;` | Declares an integer that will hold a value corresponding to the type of game that the game object is. |

| | |
|---|---|
| `static vector<int > gameNumbers;` | |

Declares a vector that will be accessible by all game objects. This vector will hold all the values from the file GAMENUMBERS.DAT.

| | |
|---|---|
| `void RNDgame(gsl_rng*);` | Declares the function that will generate random games. |

| | |
|---|---|
| `int gameIndexer(vector<int >*, const bool);` | |

Declares the function that will match the ID numbers of the games produced to their position in the file GAMENUMBERS.DAT.

| | |
|---|---|
| `void swapDias(vector<int >*);`<br>`void swapCols(vector<int >*);`<br>`void swapRows(vector<int >*);`<br>`void swapPlayers(vector<int >*);` | Declares the functions that will manipulate the games produced so that the same game will always be put into the same presentation, no matter how it was originally generated. |
| `int pays[8];` | Declares the array that will hold the eight payoff values that define a game. |
| `int paySum;` | Declares the integer that will hold the sum of all the payoffs available in the game. |
| | `bitset<2> sq;` |

Declares the bitset that will hold the status quo designation.

| | |
|---|---|
| | `bitset<12> bw;` |

Declares the bitset that will hold comparison information for status quo payoffs against all the other payoffs in the game..

| | |
|---|---|
| | `int sqPays[2];` |

Declares an array that will hold the status quo payoffs for easy reference.

| | |
|---|---|
| `bitset<2> out;` | Declares a bitset that will designate the outcome of the game. |

```
int gameIndexRow;
int gameIndexCol;
```

Declares the integer variables that will hold the game index values for the Row and Column players.

```
boolcolShift;
boolrowShift;
```

Declares the boolean variables that will hold information regarding whether or not row or column shifts are necessary to make the game conform to the presentation standards.

### E.3.20 game.cpp

```
#include<iostream>
```

Includes the C++ library that allows data to be accepted from the keyboard and to be written to the screen.

```
#include<fstream>
#include<algorithm>
```

Includes the library that allows data to be written and read to files and the library that allows for searches to be carried out on C++ Standard Library containers.

```
#include"game.h"
```

Includes the header file the declares all the functions and variables that belong to game objects.

```
usingnamespacestd;
```

Simplifies coding by preventing the need to specify which namespace various elements of the code are being drawn from. Everything used here comes from the standard namespace and this line declares that.

```
vector<int > game::gameNumbers;
```

Declares a global vector called gameNumbers that will hold all the game ID numbers from the file GAMENUMBERS.DAT

```
game::game(const int type, gsl_rng* r){
```

Declares the game object constructor.

```
gameType= type;

switch(gameType){
  case0:
    RNDgame(r);
    break;
  case1:
    PDgame(r);
    break;
  case2:
    HNTgame(r);
    break;
  case3:
    CHKgame(r);
    break;
  case4:
    BSgame(r);
    break;}
```

Sets the variable gameType to the type of game that the function was asked to create.

Based on gameType one of five functions is called, each of which produces games of a different type. For this simulation series only games of type zero are looked at so the other have been ignored.

```
paySum=0;
  for(int i=0; i<8; i++)
    paySum=paySum+pays[i];
```

Determines the sum of all the payoffs available in the game and stores them in the variable paySum.

| | |
|---|---|
| `vector<int > vgi(pays, pays+ 8);` | |

Declares a vector that will hold game information during the indexing procedure that is necessary for NC agents to play the game (vgi=Vector Game Indexer).

| | |
|---|---|
| `colShift=false;`<br>`rowShift=false;` | |

Sets the shift tracking variables to a default value.

| | |
|---|---|
| `gameIndexRow= gameIndexer(&vgi, true);`<br>`gameIndexCol= gameIndexer(&vgi, false);` | |

Determines the game index values for both the Row and Column perspectives.



| | |
|---|---|
| `outRanks[0]=-1;`<br>`outRanks[1]=-1;` | |

Sets the default outcome rankings to a value that will make little sense if they are not updated correctly in the future.

| | |
|---|---|
| | `if(gsl_rng_uniform(r) > 0.5)`<br>`  sq.flip(0);`<br>`if(gsl_rng_uniform(r) > 0.5)`<br>`  sq.flip(1);` |

Sets a random status quo outcome for the game.

```
if(sq.test(0))  //TOP
  if(sq.test(1)){
    sqPays[0]=pays[0];
    sqPays[1]=pays[4];
      if(pays[0]<pays[1])  //better
        bw.set(0);
      else
        if(pays[0]>pays[1])
          bw.set(1);
      if(pays[0]<pays[2])  //better
        bw.set(2);
      else
        if(pays[0]>pays[2])
          bw.set(3);
      if(pays[0]<pays[3])
        bw.set(4);
      else
        if(pays[0]>pays[3])
          bw.set(5);
      if(pays[4]<pays[5])
        bw.set(6);
      else
        if(pays[4]>pays[5])
          bw.set(7);
      if(pays[4]<pays[6])
        bw.set(8);
      else
```

This block of code is incomplete. It is just too long to include. It is just a long list of conditionals that set the bits within the bitset bw, which tracks how the non status quo payoffs compare to the status quo payoffs.

```
void game::printGame(){
  cout << "Game type:"<< gameType<< endl;
  cout << pays[0] << "   "<< pays[4] << "   "<<
"|"<< "   "<< pays[2] << "   "<< pays[5] << "
"<< endl;
  cout << "----------------"<< endl;
  cout << pays[1] << "   "<< pays[6] << "   "<<
"|"<< "   "<< pays[3] << "   "<< pays[7] << "
"<< endl;
  cout << endl;
  cout << "With Status Quo Outcome:"<< endl;
  if(sq.test(0))
    if(sq.test(1))
      cout << "TOP-LEFT";
    else
      cout << "TOP-RIGHT";
  else
    if(sq.test(1))
      cout << "BOTTOM-LEFT";
    else
      cout << "BOTTOM-RIGHT";
  cout << endl << endl;}
```

Prints the game and associated information on the screen.

| | |
|---|---|
| | `int game::payBits(const int bitTest){`<br>`  int bit=0;`<br>`  if(bw.test(bitTest))`<br>`  bit=1;`<br>`  returnbit;}` |

Returns the value of the bit within the bitset bw corresponding to the index value passed to the function.

| | |
|---|---|
| `int game::getPaySum(){`<br>`  returnpaySum;}` | Returns the sum of all the payoffs in the game. |
| `int game::outcome(const boolrow, const boolrowPlay, const boolcolPlay){` | Opens the function that determines the outcome of a game. |

| | |
|---|---|
| `int outcomePay=-1;`<br>`int payLoc=-1;` | |

Used by NC game objects to store information about the game. outcomePay will hold the payoff for the agent under consideration and payLoc will hold an integer that corresponds to one of the four possible outcomes.

```
if(row==true)
  if(rowShift==false)
    if(colShift==false)
      if(rowPlay==true)
        if(colPlay==true){
          outcomePay=pays[0];
          payLoc=0;}
        else{
          outcomePay=pays[2];
          payLoc=2;}
      else
        if(colPlay==true){
          outcomePay=pays[1];
          payLoc=1;}
        else{
          outcomePay=pays[3];
          payLoc=3;}
```

```
if(row==true)
  if(sq.test(0))
    if(sq.test(1))
      if(rowPlay==true)
        if(colPlay==true)
          returnpays[0];
        else
          returnpays[2];
      else
        if(colPlay==true)
          returnpays[1];
        else
          returnpays[3];
```

The first block of code used to actually determine the outcome of the game.

```
  else
    if(rowPlay==true)
      if(colPlay==true){
        outcomePay=pays[2];
        payLoc=2;}
      else{
        outcomePay=pays[0];
        payLoc=0;}
    else
      if(colPlay==true){
        outcomePay=pays[3];
        payLoc=3;}
      else{
        outcomePay=pays[1];
        payLoc=1;}
```

```
  else
    if(rowPlay==true)
      if(colPlay==true)
        returnpays[2];
      else
        returnpays[0];
    else
      if(colPlay==true)
        returnpays[3];
      else
        returnpays[1];
```

The second block of code used to actually determine the outcome of the game.

```
else
  if(colShift==false)
    if(rowPlay==true)
      if(colPlay==true){
        outcomePay=pays[1];
        payLoc=1;}
      else{
        outcomePay=pays[3];
        payLoc=3;}
    else
      if(colPlay==true){
        outcomePay=pays[0];
        payLoc=0;}
      else{
        outcomePay=pays[2];
        payLoc=2;}
```

```
else
  if(sq.test(1))  //BOTTOM-LEFT
    if(rowPlay==true)
      if(colPlay==true)
        returnpays[1];
      else
        returnpays[3];
    else
      if(colPlay==true)
        returnpays[0];
      else
        returnpays[2];
```

The third block of code used to actually determine the outcome of the game.

```
else
  if(rowPlay==true)
    if(colPlay==true){
      outcomePay=pays[3];
      payLoc=3;}
    else{
      outcomePay=pays[1];
      payLoc=1;}
  else
    if(colPlay==true){
      outcomePay=pays[2];
      payLoc=2;}
    else{
      outcomePay=pays[0];
      payLoc=0;}
```

```
else
  if(rowPlay==true)
    if(colPlay==true)
      returnpays[3];
    else
      returnpays[1];
  else
    if(colPlay==true)
      returnpays[2];
    else
      returnpays[0];
```

The fourth block of code used to actually determine the outcome of the game.

```
else
  if(rowShift==false)  //TOP
    if(colShift==false)  //TOP-LEFT
      if(rowPlay==true)
        if(colPlay==true){
          outcomePay=pays[4];
          payLoc=4;}
        else{
          outcomePay=pays[5];
          payLoc=5;}
      else
        if(colPlay==true){
          outcomePay=pays[6];
          payLoc=6;}
        else{
          outcomePay=pays[7];
          payLoc=7;}
```

```
else
  if(sq.test(0))  //TOP
    if(sq.test(1))  //TOP-LEFT
      if(rowPlay==true)
        if(colPlay==true)
          returnpays[4];
        else
          returnpays[5];
      else
        if(colPlay==true)
          returnpays[6];
        else
          returnpays[7];
```

The fifth block of code used to actually determine the outcome of the game.

```
  else
    if(rowPlay==true)
      if(colPlay==true){
        outcomePay=pays[5];
        payLoc=5;}
      else{
        outcomePay=pays[4];
        payLoc=4;}
    else
      if(colPlay==true){
        outcomePay=pays[7];
        payLoc=7;}
      else{
        outcomePay=pays[6];
        payLoc=6;}
```

```
  else
    if(rowPlay==true)
      if(colPlay==true)
        returnpays[5];
        else
          returnpays[4];
      else
        if(colPlay==true)
          returnpays[7];
        else
          returnpays[6];
```

The sixth block of code used to actually determine the outcome of the game.

```
else
  if(colShift==false)
    if(rowPlay==true)
      if(colPlay==true){
        outcomePay=pays[6];
        payLoc=6;}
      else{
        outcomePay=pays[7];
        payLoc=7;}
    else
      if(colPlay==true){
        outcomePay=pays[4];
        payLoc=4;}
      else{
        outcomePay=pays[5];
        payLoc=5;}
```

```
else
  if(sq.test(1))  //BOTTOM-LEFT
    if(rowPlay==true)
      if(colPlay==true)
        returnpays[6];
      else
        returnpays[7];
    else
      if(colPlay==true)
        returnpays[4];
      else
        returnpays[5];
```

The seventh block of code used to actually determine the outcome of the game.

```
else
  if(rowPlay==true)
    if(colPlay==true){
      outcomePay=pays[7];
      payLoc=7;}
    else{
      outcomePay=pays[6];
      payLoc=6;}
  else
    if(colPlay==true){
      outcomePay=pays[5];
      payLoc=5;}
    else{
      outcomePay=pays[4];
      payLoc=4;}
```

```
else
  if(rowPlay==true)
    if(colPlay==true)
      returnpays[7];
    else
      returnpays[6];
  else
    if(colPlay==true)
      returnpays[5];
    else
      returnpays[4];}
```

The eighth block of code used to actually determine the outcome of the game.

```
int numEq=-1;
int numGr=0;
int basePay=-1;
int maxPay=-1;
int outRank=-1;
```

NC agents evaluate outcomes based on how the outcome ranks against the other possible outcomes that were available. This block of code declares a set of integer values that are used in making this determination.

```
if(row==true)
  basePay=0;
else
  basePay=4;
```

This conditional tests to see if the outcome determination is being made for the row player of the column player. Based on the assessment basePay is either set to zero or four. basePay is used to designate the starting payoff out of the eight possible payoffs to begin the comparison of payoffs for. If the payoffs for both players were listed in a line such that all the row players payoffs preceded the column players payoffs then, using zero-base counting, the row player's payoffs would start at position zero and the column players payoffs would start at position four.

```
maxPay=basePay+4;
```

If the payoffs for both players were listed in a line such that all the row players payoffs preceded the column players payoffs then, using zero-base counting, the row player's payoffs would start at position zero and the column players payoffs would start at position four. The row player's payoffs would end at position three and the column player's payoffs would end at position seven. maxPay can thus be understood to be the first location beyond the last index position relevant to the player under consideration.

```
for(int i=basePay; i<maxPay; i++){
  if(pays[payLoc]==pays[i])
    numEq++;
  if(pays[payLoc]>pays[i])
    numGr++;}
```

This for-loop determines how many payoffs available to the agent were greater than what they actually received and how many were equivalent. These will be used by the player when ranking the outcome.

```
switch(numEq){
```

Opens a switch based on the number of equalities that will be part of a system of switches that will determine the outcome rank.

```
case0:
  switch(numGr){
    case0:
      outRank=0;
      break;
    case1:
      outRank=1;
      break;
    case2:
      outRank=2;
      break;
    case3:
      outRank=3;
      break;}
    break;
```

The first switch inside the number of equalities switch that is based on the number of outcomes where a greater payoff that what was received was offered. Ranks are assigned.

```
case1:
  switch(numGr){
    case0:
      outRank=4;
      break;
    case1:
      outRank=5;
      break;
    case2:
      outRank=6;
      break;}
    break;
```



The second switch inside the number of equalities switch that is based on the number of outcomes where a greater payoff that what was received was offered. Ranks are assigned.

```
case2:
  switch(numGr){
    case0:
      outRank=7;
      break;
    case1:
      outRank=8;
      break;}
    break;
```

The third switch inside the number of equalities switch that is based on the number of outcomes where a greater payoff that what was received was offered. Ranks are assigned.

```
case3:
  outRank=9;
  break;}
```

The final case of the initial switch. Here all the alternatives are equal to the payoff received. Consequently there are no payoffs that are greater and so an embedded switch is not needed. Only one rank is assigned because there is only one case.

```
if(row==true)
  outRanks[0]=outRank;
  else
  outRanks[1]=outRank;
```

outRanks is a two-place array that holds the outcome ranking values for both the row and the column player. This conditional assigns the appropriate rank to the appropriate player.



```
returnoutcomePay;}
```

Returns the outcome payoff for the agent for whom the function was called.

```
void game::RNDgame(gsl_rng* r){
  for(int i=0; i<8; i++)
    pays[i]=(int )(gsl_rng_uniform_int(r,10));}
```

Generates a 2$\times$2 game with payoffs ranging from zero to nine.

```
int game::gameIndexer(vector<int >*
currentGamePtr, const boolrowSet){
```

Opens the game indexing function. This function is used in the case where NC agents are the ones playing games. It determines what game is being played and, more importantly, which perspective belongs to each player.

```
vector<int > vPays(*currentGamePtr);
int minMut=0;
int minMutLoc=0;
int gameNum=0;
vector<int > ordPays;
vector<int > permValues;
```

Declares the major variables involved in carrying out the function.

```
for(int h=0; h<2; h++){
  for(int i=0; i<4; i++)
    if(count(ordPays.begin(), ordPays.end(),
vPays.at(i+(h*4)))==0)
      ordPays.push_back(vPays.at(i+(h*4)));

    sort(ordPays.begin(), ordPays.end());
    for(int i=0; i<int (ordPays.size()); i++)
    for(int j=0; j<4; j++)
      if(vPays.at(j+(h*4))==ordPays.at(i))
        vPays.at(j+(h*4))=i;

    ordPays.clear();}
```

This set of nested for-loops and conditionals takes the original game and recasts it in minimal normal form (See Appendix: 2×2 Game Taxonomy).

```
if(rowSet==true){
permValues.push_back(((vPays.at(0)+vPays.at(4)*4)+
  (vPays.at(2)+vPays.at(5)*4)*16)+
  ((vPays.at(1)+vPays.at(6)*4)+
  (vPays.at(3)+vPays.at(7)*4)*16)*256);
permValues.push_back(((vPays.at(1)+vPays.at(6)*4)+
  (vPays.at(3)+vPays.at(7)*4)*16)+
  ((vPays.at(0)+vPays.at(4)*4)+
  (vPays.at(2)+vPays.at(5)*4)*16)*256);

permValues.push_back(((vPays.at(2)+vPays.at(5)*4)+
  (vPays.at(0)+vPays.at(4)*4)*16)+
  ((vPays.at(3)+vPays.at(7)*4)+
  (vPays.at(1)+vPays.at(6)*4)*16)*256);

permValues.push_back(((vPays.at(3)+vPays.at(7)*4)+
  (vPays.at(1)+vPays.at(6)*4)*16)+
  ((vPays.at(2)+vPays.at(5)*4)+
  (vPays.at(0)+vPays.at(4)*4)*16)*256);
minMut=*(min_element(permValues.begin(),
permValues.end()));
while(minMut!=permValues.at(minMutLoc))
  minMutLoc++;
if(minMutLoc==(1 || 3))
  rowShift=true;}
```

If the game is being analyzed from the perspective of the row player then the game is indexed from this perspective.

```
else{
permValues.push_back(((vPays.at(4)+vPays.at(0)*4)+
  (vPays.at(6)+vPays.at(1)*4)*16)+
  ((vPays.at(5)+vPays.at(2)*4)+
  (vPays.at(7)+vPays.at(3)*4)*16)*256);

permValues.push_back(((vPays.at(5)+vPays.at(2)*4)+
  (vPays.at(7)+vPays.at(3)*4)*16)+
  ((vPays.at(4)+vPays.at(0)*4)+
  (vPays.at(6)+vPays.at(1)*4)*16)*256);

permValues.push_back(((vPays.at(6)+vPays.at(1)*4)+
  (vPays.at(4)+vPays.at(0)*4)*16)+
  ((vPays.at(7)+vPays.at(3)*4)+
  (vPays.at(5)+vPays.at(2)*4)*16)*256);

permValues.push_back(((vPays.at(7)+vPays.at(3)*4)+
  (vPays.at(5)+vPays.at(2)*4)*16)+
  ((vPays.at(6)+vPays.at(1)*4)+
  (vPays.at(4)+vPays.at(0)*4)*16)*256);
minMut=*(min_element(permValues.begin(),
permValues.end()));
while(minMut!=permValues.at(minMutLoc))
  minMutLoc++;
if(minMutLoc==(1 || 3))
  colShift=true;}
```

If the game is being analyzed from the perspective of the column player then the game is indexed from this perspective.

```
while(minMut!=gameNumbers.at(gameNum))
  gameNum++;;
```

The value of the game perspective taken from the preceding conditionals has been stored in the variable minMut. This while-loop counts through the perspective values stored in gameNumbers until it finds a match.

```
returngameNum;}
```

The index position of the perspective is returned.

```
int game::getGameIndex(const boolrowOn){
  if(rowOn==true)
    returngameIndexRow;
  else
    returngameIndexCol;}
```



Returns the game index for either the row or column player as determined by a boolean value passed to the function.

```
void game::swapDias(vector<int >* vPaysPtr){
  swap (vPaysPtr->at(0), vPaysPtr->at(3));
  swap (vPaysPtr->at(1), vPaysPtr->at(2));
  swap (vPaysPtr->at(4), vPaysPtr->at(7));
  swap (vPaysPtr->at(5), vPaysPtr->at(6));}
```

Reorganizes the payoffs in a game in a way that is equivalent to a row and column swap performed on a normal form game.

```
void game::swapCols(vector<int >* vPaysPtr){
  swap (vPaysPtr->at(0), vPaysPtr->at(2));
  swap (vPaysPtr->at(1), vPaysPtr->at(3));
  swap (vPaysPtr->at(4), vPaysPtr->at(5));
  swap (vPaysPtr->at(6), vPaysPtr->at(7));}
```

Reorganizes the payoffs in a game in a way that is equivalent to a column swap performed on a normal form game.

```
void game::swapRows(vector<int >* vPaysPtr){
  swap (vPaysPtr->at(0), vPaysPtr->at(1));
  swap (vPaysPtr->at(2), vPaysPtr->at(3));
  swap (vPaysPtr->at(4), vPaysPtr->at(6));
  swap (vPaysPtr->at(5), vPaysPtr->at(7));}
```

Reorganizes the payoffs in a game in a way that is equivalent to a row swap performed on a normal form game.

```
void game::swapPlayers(vector<int >* vPaysPtr){
  swap_ranges (vPaysPtr->begin(),
vPaysPtr->begin() + 4, vPaysPtr->begin() + 4);}
```

Reorganizes the payoffs in a game in a way that is equivalent to a player swap performed on a normal form game.

```
void game::loadGameNumbers(){
  ifstreamgameFile( "gameNumbers.dat", ios::in);

  if(!gameFile){
    cerr << "GAMENUMBERS.DAT COULD NOT BE
OPENED"<< endl;
    exit(1);}

  int gn=-1;
  while(gameFile >> gn)
    gameNumbers.push_back(gn);}
```

Loads the game index numbers into the vector gameNumbers for easy future reference.

```
game::~game(){}
```

The game destructor.

### E.3.21 match.h

```
#include<utility>
#include"agent.h"
```

The pair container is part of the utility library and since pairs are needed in match objects this library is included. The agent header is included because match objects are made up of pairs

```
using std::pair;
```

Only one particular item from the C++ standard library can benefit from simplified reference so a using declaration is applied only to it.

```
classmatch{
```

Opens the declaration of the match class.

```
match(void );
~match();
```

The match constructor and destructor.

```
void assignment(bool, int );
```

Declares the assignment function which will accept the ID of the agent as an integer and then place the ID as either the first or second member of the pair based on the boolean passed to the function.

```
void printMatch(const int , const int );
```

Declares the function that outputs the contents of a match object.

```
int firstInPair();
```

Declares the function that will return the first member of a match object.

```
int secondInPair();
```

Declares the function that will return the second member of a match object.

```
private:
```

Declares that all following items are private in the sense that only the object itself has direct access to them.

```
pair<int , int > players;}
```

Declares a pair of integers that will be called players.

### E.3.22 match.cpp

```
#include <iostream>
#include"match.h"
```

Includes the iostream library so that information can be output on the screen and the match header, which holds all the function declarations.

```
usingnamespacestd;
```

Simplifies coding by preventing the need to specify which namespace various elements of the code are being drawn from. Everything used here comes from the standard namespace and this line declares that.

```
match::match(void ){
  players.first= -1;
  players.second= -1;}
```

The match constructor. Creates a new match object and fills it with nonsense values that can be used for error checking elsewhere.

```
void match::assignment(boolpairPos, int
indexPos){
  if(pairPos==true)
    players.first=indexPos;
  else
    players.second=indexPos;}
```

Assigns new information to the items within the pair that composed a match object.

```
void match::printMatch(const int rowID, const int
colID){
  cout << "The 1st/ROW player in pair has index
number "<< players.first<< " and ID# "<< rowID <<
endl;
  cout << "The 2nd/COLUMN player in pair has
index number "<< players.second<< " and ID# "<<
colID << endl << endl;  }
```

Prints on the screen the index number of each agent in the pair and their ID number.

```
int match::firstInPair(){
  returnplayers.first;}
```

Return the index number of first player stored in the pair.

```
int match::secondInPair(){
  returnplayers.second;}
```

Return the index numbersecond player in the pair.

```
match::~match(){}
```

The destructor for match objects.

# Bibliography


Appel, K. and W. Haken (1977). Every planar map is four colorable: Part i discharging. *Illinois Journal of Mathematics 21*, 429–490.

Appel, K., W. Haken, and J. Koch (1977). Every planar map is four colorable: Part ii reducibility. *Illinois Journal of Mathematics 21*, 491–567.

Axelrod, R. (1984). *The Evolution of Cooperation*. New York: Basic Books.

Axelrod, R. (1997). *The Complexity of Cooperation*. Princeton University Press.

Baltagi, B. H. (2008). *Econometric Analysis of Panel Data*. Etobicoke, Canada: John Wiley and Sons.

Barkow, J. H., L. Cosmides, and J. Tooby (1992). *The Adapted Mind*. Toronto: Oxford University Press.

Barnes, B. (1985). *About Science*. Oxford University Press.

Baum, C. F. (2006). *An Introduction to Modern Econometrics Using Stata*. College Station, Texas: Stata Press.

Bazerman, M. H. (1994). *Judgement in Managerial Decision Making* (Third ed.). Toronto: John Wiley and Sons.

Bicchieri, C. (2006). *The Grammar of Society*. New York: Cambridge University Press.

Billari, F. C., T. Fent, A. Prskawetz, and J. Scheffran (Eds.) (2006). *Agent-Based Computational Modelling: Applications in Demography, Social, Economic and Environmental Sciences*. New York: Physica-Verlag.

Binmore, K. (1994). *Game Theory and the Social Contract*, Volume 1: Playing Fair. Cambridge, MA: MIT Press.

Binmore, K. (1998). *Game Theory and the Social Contract*, Volume 2: Just Playing. Cambridge, MA: MIT Press.

Binmore, K. (2005). *Natural Justice*. Toronto: Oxford University Press.

Buuren, S. V., H. C. Boshuizen, and D. L. Knook (1999). Multiple imputation of missing blood pressure covariates in survival analysis. *Statistics in Medicine 18*, 681–694.

Byrne, R. W. and A. Whiten (Eds.) (1988). *Machiavellian intelligence : social expertise and the evolution of intellect in monkeys, apes, and humans*. New York: Oxford University Press.

Cartwright, N. (1980). Do the laws of physics state the facts. *Pacific Philosophical Quarterly 61*, 75–84.

Cartwright, N. (1983). *How the Laws of Physics Lie*. New York: Oxford University Press.

Chapman, B., G. Jost, and R. van der Pas (2008). *Using OpenMP: Portable Shared Memory Parallel Programming*. London: MIT Press.

Cohen, L. J. (1989). *An Introduction to the Philosophy of Induction and Probability*. Toronto: Clarendon Press.

Collins, H. and T. Pinch (1998). *The Golem: What You Should Know about Science* (Second ed.). New York: Cambridge University Press.

Conesa, J. C. and C. Garriga (2003). Status quo problem in social security reforms. *Macroeconomic Dynamics 7*(5), 691–710.

Crookall, D., R. Oxford, and D. Saunders (1987). Towards a reconceptualization of simulation: From representation to reality. *Simulation/Games for Learning: The Journal of SAGSET 17*(4), 147–171.

Cummins, D. D. (2000). How social environment shaped the evolution of mind. *Synthese 122*, 3–28.

Davies, M. and T. Stone (1995a). *Folk Psychology*. Cambridge, Massachusetts: Blackwell Publishing.

Davies, M. and T. Stone (1995b). *Mental Simulation: Evaluations and Applications*. Cambridge, Massachusetts: Blackwell Publishing.

Dawson, E., T. Gilovich, and D. T. Regan. Motivated reasoning and susceptibility to the "cell a" bias.

Deitel, H. M. and P. J. Deitel (2005). *C++ How to Program* (Fifth ed.). Upper Saddle River, New Jersey: Prentice Hall.

Dunbar, R. (1996). *Grooming, Gossip, and the Evoloution of Language*. Cambridge, Massachusetts: Harvard University Press.

Eckel, F. A. and C. F. Mass (2005). Aspects of effective mesoscale, shrot-range ensemble forecasting. *Weather and Forecasting 20*, 328–350.

Epstein, J. M. (2006). *Generative Social Science*. Princeton, New Jersey: Princeton University Press.

Epstein, J. M. and R. Axtell (1996). *Growing Artificial Societies: Social Science from the Bottom Up*. Washington, D.C.: Brookings Institution Press.

Fetzer, J. H. (1987). The role of models in computer science. *Monist 82*(1), 20–37.

Ficici, S. G. and A. Pfeffer (2008a, May 12-16). Modeling how humans reason about others with partial information. In Padgham, Parkes, Müller, and Parsons (Eds.), *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Estoril, Portugal, pp. 315–322. AAMAS 2008.

Ficici, S. G. and A. Pfeffer (2008b, May 12-16). Simultaneously modeling humans' preferences and their beliefs about others' preferences. In Padgham, Parkes, Müller, and Parsons (Eds.), *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Estoril, Portugal, pp. 323–330.

Fokianos, K. (2004). Truncated poisson regression for time series of counts. *Scandanavian Journal of Statistics 28*(4), 645–659.

Fudenberg, D. and J. Tirole (1991). *Game Theory*. MIT Press.

Gauthier, D. (1988). *Morals by Agreement*. Toronto: Clarendon Press.

Giere, R. N. (1988). *Explaining Science: A Cognitive Approach*. Chicago: University of Chicago Press.

Gigerenzer, G., P. M. Todd, and the ABC Research Group (Eds.) (1999). *Simple Hueristics that Make Us Smart*. New York: Oxford University Press.

Gilbert, N. (2008). *Agent-Based Models*. Los Angeles: SAGE Publications.

Gilovich, T. (1991). *How We Know What Isn't So: The Fallibility of Human Reason in Everyday Life*. Toronto: Maxwell Macmillan Canada.

Gilovich, T., D. Griffin, and D. Kahneman (Eds.) (2002). *Heuristics and Biases: The Psychology of Intuitive Judgement*. New York: Cambridge University Press.

Gneiting, T. and A. E. Raferty (2005, October). Weather forecasting with ensemble methods. *Science*, 248–249.

Gray, T. (2009). *Theo Gray's Mad Science: Experiments You Can Do at Home–But Probably Shouldn't*. New York: Black Dog & Leventhal Publishers.

Hanson, N. R. (1971). *Observation and Explanation: A Guide to Philosophy of Science*. Harper & Row.

Harsanyi, J. C. (1977). *Rational Behaviour and Bargaining Equilibrium in Games and Social Situations*. New York: Cambridge University Press.

Hartmann, S. (1996). The world as a process. In R. Hegselmann, U. Mueller, and K. G. Troitzsch (Eds.), *Modelling and Simulation in the Social Sciences from the Philosophy of Science Point of View*. Dordrecht: Kluwer Academic Publishers.

Heath, O. V. S. (1970). *Investigation by Experiment*. Edward Arnold Publishers Ltd.

Hegselmann, R., U. Mueller, and K. G. Troitzsch (Eds.) (1996). *Modelling and Simulation in the Social Sciences from the Philosophy of Science Point of View*. London: Kluwer Academic Publishers.

Hesse, M. B. (1966). *Models and Analogies in Science*. Notre Dame, Ind.: University of Notre Dame Press.

Hickam, H. H. (1998). *Rocket Boys*. Delacore.

Horkheimer, M. (1982). *Critical Theory: Selected Essays*. New York: Continuum.

Horkheimer, M. and T. W. Adorno (1972). *Dialectic of Enlightenment*. New York: Continuum.

Houtekamer, P. L., L. Lefaivre, J. Derome, H. Ritchie, and H. L. Mitchell (1996). A system simulation approach to ensemble prediction. *Monthly Weather Review 124*, 1225–1242.

Huberman, B. and N. Glance (1993). Evolutionary games and computer simulations. *Proceedings of the Natural Accademy of Sciences 90*, 7715–18.

Huizinga, J. (1950). *Homo Ludens: A Study of the Play Element in Culture*. Boston: Beacon Press.

Humphreys, P. (2004). *Extending Ourselves*. New York: Oxford University Press.

Hurley, S. and N. Chater (Eds.) (2005). *Perspectives on Imitation: From Neuroscience to Social Science*, Volume 2: Imitation, Human Development, and Culture. Cambridge, Massachusetts: MIT Press.

Josuttis, N. M. (1999). *The C++ Standard Library: A Tutorial and Reference* (Toronto ed.). Addison Wesley.

Kahneman, D., J. L. Knetsch, and R. H. Thaler (1991). Anomalies: The endowment effect, loss aversion, and status quo bias. *The Journal of Economic Perspectives 5*(1), 193–206.

Kahneman, D. and A. Tversky (Eds.) (2000). *Choices, Values, and Frames*. New York: Cambridge University Press.

Kaplan, M. (1996). *Decision Theory as Philosophy*. New York: Cambridge University Press.

Kelly, T. (2004). Sunk costs, rationality, and acting for the sake of the past. *NOÛS 38*(1), 60–85.

Kim, K.-J. and H. Lipson (2009, July 8-12). Towards a "theory of mind" in simulated robots. In *GECCO'09*, Montréal, Québec, Canada.

Kirk, R. E. (2008). *Statistics: An Introduction* (Fifth ed.). Belmont, California: Thompson-Wadsworth.

Knuth, D. E. (1986). *The TeXbook*. Don Mills, Ontario: Addison Wesley.

Kuhn, T. S. (1970). *The Structure of Scientific Revolutions*. Chicago: University of Chicago Press.

Kuhn, T. S. (1977). *The Essential Tension: Selected Studies in Scientific Tradition and Change*. University of Chicago Press.

Latour, B. and S. Woolgar (1986). *Laboratory life : the construction of scientific facts*. Princeton, N.J.: Princeton University Press.

Liebrand, W. B., A. Nowak, and R. Hegselmann (Eds.) (1998). *Computer Modeling of Social Processes*. London: SAGE.

Lilla, M. and CBC Radio (2008). Ideas: The stillborn god.

Luce, R. D. and H. Raiffa (1989). *Games and Decisions: Introduction and Critical Survey*. New York: Dover Publications, Inc.

Marsell, S. C., D. V. Pynadath, and S. J. Read (2004). Psychsim: Agent-based modeling of social interactions and influence. *ICCM 2004*.

Mérő, L. (1998). *Moral Calculations: Game Theory, Logic, and Human Frailty*. New York: Copernicus, Springer-Verlag.

Moore, D. S. and W. I. Notz (2006). *Statistics: Concepts and Controversies*. New York: W. H. Freeman and Company.

Morgan, M. S. and M. Morrison (Eds.) (1999). *Models as Mediators: Perspectives on Natural and Social Science*. New York: Cambridge University Press.

Morgan, W. J. (2008). Some futher words on suits on play. *Journal of the Philosophy of Sport 35*, 120–141.

Moulin, H. (1988). *Axioms of Cooperative Decision Making*. New York: Cambridge University Press.

Myerson, R. B. (1991). *Game Theory: Analysis of Conflict*. Cambridge, Massachusetts: Harvard University Press.

Nagel, E. (1961). *The Structure of Science: Problems in the Logic of Scientific Explanation*. London: Routledge.

Namatame, A., T. Terano, and K. Kurumatani (Eds.) (2002). *Agent Based Approaches in Economic and Social Complex Systems*. Washington, D.C.: IOS Press.

Nowak, M. A. and R. M. May (1992). Evolutionary games and spatial chaos. *Nature 359*, 826–29.

Ouellette, J. (2005). *Black Bodies and Quantum Cats: Tales from the Annals of Physics*. Toronto: Penguin Group.

Pears, D. F. (1984). *Motivated Irrationality*. Toronto: Clarendon Press.

Premack, D. G. and G. Woodruff (1978). Does the chimpanzee have a theory of mind? *Behavioral and Brain Sciences 1*, 515–526.

Rabe-Hesketh, S. and A. Skrondal (2008). *Multilevel and Longitudinal Modeling Using Stata* (Second ed.). College Station, Texas: Stata Press.

Rapoport, A. (1966). *Two-Person Game Theory*. Ann Arbor, Michigan: University of Michigan Press.

Rapoport, A. (1970). *N-Person Game Theory: Concepts and Applications*. Ann Arbor, Michigan: University of Michigan Press.

Rapoport, A. (1983). *Mathematical Models in the Social and Behavioural Sciences*. Toronto: John Wiley and Sons.

Rapoport, A. (1989). *Decision Theory and Decision Behaviour*. Boston: Kluwer Academic Publishers.

Rapoport, A. and M. Guyer (1966). A taxonomy of 2x2 games games. *General Systems Yearbook 11*, 203–214.

Rapoport, A., M. J. Guyer, and D. G. Gordon (1976). *The 2x2 Game*. Ann Arbor, Michigan: University of Michigan Press.

Robertson, N., D. Sanders, P. Seymour, and R. Thomas (1997). The four color theorem. *Journal of Combinatorial Theory, Series B 70*, 2–44.

Robinson, D. and D. Goforth (2005). *The Topology of 2x2 Games: A New Periodic Table*. Routledge Advances in Game Theory. New York: Routledge.

Royston, P. (2007). Multiple imputation of missing values: further update of ice, with an emphasis on interval censoring. *The Stata Journal 7*(4), 445–464.

Russell, B. (1951). *The Impact of Science on Society*. Columbia University Press.

Samuelson, W. and R. Zeckhauser (1988). Status quo bias in decision making. *Journal of Risk and Uncertainty 1*, 7–59.

Sawyer, R. K. (2003). The mechanics of emergence. *Philosophy of the Social Sciences 34*(2), 260–282.

Schaeffer, J., N. Burch, Y. Björnsson, A. Kishimoto, M. Müller, R. Lake, P. Lu, and S. Sutphen (2007). Checkers is solved. *Science 317*(5844), 1518–1522.

Schelling, T. C. (1971). Dynamic models of segregation. *Journal of Mathematical Sociology 1*, 143–186.

Searle, J. R. (1992). *The Rediscovery of the Mind*. Cambridge, M.A.: MIT Press.

Sen, A. (2002). *Rationality and Freedom*. Cambridge, Massachusetts: The Belknap Press of Harvard University Press.

Sharpanskykh, A. and J. Treur (2008, July 15-19). An ambient agent model for automated mindreading by identifying and monitoring representation relations. Athens, Greece. PETRA'08.

Simons, D. J. and C. F. Chabris (1999). Gorillas in our midst: sustained inattentional blindness for dynamic events. *Perception 28*, 1059–1074.

Skyrms, B. (1996). *The Evolution of the Social Contract*. New York: Cambridge University Press.

Skyrms, B. (2005). *The Stag Hunt and the Evolution of Social Structure*. New York: Cambridge University Press.

Smith, L. (2007). *Chaos: A Very Short Introduction*. New York: Oxford University Press.

Smith, P. (1998). *Explaining Chaos*. New York: Cambridge University Press.

Sober, E. and D. S. Wilson (1996). *Unto Others: The Evolution and Psychology of Unselfish Behaviour*. Cambridge, M.A.: Harvard University Press.

Staff (2005). Proof and beauty. *Economist 375*, 73–74.

Stanovich, K. E. (1999). *Who is Rational?* Mahwah, New Jersey: Lawrence Erlbaum Associates, Publishers.

StataCorp (2007a). *Stata Graphics Reference Manual* (Release 10 ed.). College Station, Texas: StataCorp LP.

StataCorp (2007b). *Stata Longitudinal/Panel-Data Reference Manual* (Release 10 ed.). College Station, Texas: StataCorp LP.

StataCorp (2007c). *Stata User Reference Manual* (Release 10 ed.). College Station, Texas: StataCorp LP.

Steele, D. R. (1996). Nozick on sunk costs. *Ethics 106*, 509–524.

Suits, B. (1977, Fall). Words on play. *Journal of the Philosophy of Sport 4*, 117–131.

Suits, B. (1978). *The Grasshopper: Games, Life and Utopia*. Toronto: University of Toronto Press.

Suits, B. (2005). *The Grasshopper: Games, Life and Utopia*. Broadview Press Ltd.

Sunstein, C. R. (2003). *Why Societies Need Dissent*. New York: Harvard University Press.

Suppe, F. (1977). *The Structure of Scientific Theories*. Urbana: University of Illinois Press.

Surowiecki, J. (2004). *The Wisdom of Crowds*. New York: Doubleday.

Taleb, N. N. (2007). *The Black Swan: The Impact of the Highly Improbable*. New York: Random House.

Terano, T., H. Deguchi, and K. Takadama (Eds.) (2003). *Meeting the challenge of social problems via agent-based simulation*. New York: Springer-Verlag.

Terza, J. V. (1985). A tobit-type estimator for the censored poisson regression model. *Economics Letters 18*, 361–365.

Thompson, R. B. (2008). *Illustrated Guide to Home Chemistry Experiments: All Lab, No Lecture*. Sebastopol, CA: O'Reilly Media, Inc.

Vallentyne, P. (Ed.) (1991). *Contractarianism and Rational Choice: Essays on David Gauthier's Morals by Agreement*. New York: Cambridge University Press.

van der Vaart, E. and R. Verbrugge (2008, May, 12-16., 2008). Agent-based models of animal cognition: A proposal and prototype. In P. Padgham, Müller, and Parsons (Eds.), *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Estoril, Portugal, pp. 1145–1152.

von Neumann, J. and O. Morgenstern (1967). *Theory of Games and Economic Behaviour* (Third ed.). New York: John Wiley and Sons.

Wang, X. H. and B. Z. Yang (2003). Classification of 2x2 games and strategic business behaviour. *American Economist 47*(2), 78–85.

Whiten, A. and R. W. Byrne (Eds.) (1997). *Machiavellian intelligence II : extensions and evaluations*. New York: Cambridge University Press.

Wilson, R. A. (2004). *Boundaries of the Mind: The Individual and the Fragile Sciences*, Volume Cognition. New York: Cambridge University Press.

Wimmer, H. and J. Perner (1983). Beliefs about beliefs: Representation and constraining function of wrong beliefs in young children's understanding of deception. *Cognition 13*, 103–128.

Winsberg, E. (2001). Simulations, models, and theories: Complex physical systems and their representations. *Philosophy of Science 68*, S442–S454.

Winsberg, E. (2003). Simulated experiments: Methodology for a virtual world. *Philosophy of Science 70*, 105–125.

Wittgenstein, L. (2001). *Philosophical Investigations: The German Text with a Revised English Translation* (Third ed.). Blackwell Publishing.

Wooldridge, J. M. (2002). *Econometric Analysis of Cross Section and Panel Datq*. Cambridge, Massachusetts: MIT Press.

Young, H. P. (1998). *Individual Strategy and Social Structure*. Princeton, New Jersey: Princeton University Press.

Zhang, Y., P. Coleman, M. Pellon, and J. Leezer (2008). A multi-agnet simulation for social agents. *SpringSim 2008*.