A Study of Orthogonality in The Witness

by

Faisal Ur Rehma Abutarab

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science
University of Alberta

© Faisal Ur Rehma Abutarab, 2024

Abstract

In this thesis, we discuss the video game design concept of orthogonality and provide its operationalization. Our focus is on combinatorial puzzle games—puzzle games in which each puzzle, i.e. level, is composed of object types that are reused across puzzles. For example, in the puzzle game *Portal 2*, object types include light bridges, lasers and turrets. Given a new object type, orthogonality asks the following: Compared to the existing game and its levels, does the new object type add quality, novel and diverse levels in terms of their gameplay?

Our focus in this thesis is on the popular combinatorial puzzle game *The Witness*. We use our measure of orthogonality to analyze the existing game of *The Witness*, and in addition, we use the measure to generate a new type of object in the game. Furthermore, we provide a general definition of orthogonality for combinatorial puzzle games. Our discussion and operationalization of orthogonality builds upon related game design ideas and concepts developed and used in the puzzle game community.

Acknowledgements

First and foremost, I would like to thank Dr. Nathan Sturtevant for his supervision. I would also like to thank all the members of the Moving AI Lab for their support and discussions, and everyone who did their research on *The Witness* at the same time as me. Thank you Justin Stevens, Eugene Chen and Junwen Shen! I would also like to thank my committee members Dr. Matthew Guzdial and Dr. Carrie Demmans Epp. Finally, I would like to thank Antonie Bodley for providing writing support early on in my thesis writing.

Table of Contents

1 Introduction **1**

2 Background **5**

2.1 Game Design 5

2.1.1 Game Elements 5

2.1.2 Combinatorial Games 8

2.1.3 Aesthetics of Combinatorial Games 11

2.2 Procedural Content Generation 16

2.3 The Witness Domain 20

3 Defining Orthogonality **29**

3.1 Analyzing a Single Object Type by Itself 29

3.2 Analyzing Two Object Types Together 39

3.3 Complete Orthogonality Function 42

3.4 General Definition 45

4 Experimental Results **47**

4.1 Space of Constraint Types 48

4.2 Evaluating the Space of Constraint Types 49

4.3 Most Orthogonal Constraint Type 50

4.4 Expert Evaluation 52

4.5 Orthogonality Function using Form of Constraint Types 54

4.6 Understanding Existing Constraint Types 55

4.7 Understanding Gameplay Representations 56

5 Related Work **59**

5.1 Automated Game Design 59

5.2 Game Evaluation 60

6 Future Work **63**

7 Conclusion **66**

Bibliography **67**

Appendix A: Curriculum **72**

List of Tables

3.1 The total number of self-avoiding paths for different sizes of empty grids. These numbers also represent the total number of solutions for different sizes of empty grids. 34
3.2 Table of Notation for Orthogonality 43

4.1 The count of constraint types in our space of constraint types with a given orthogonality. 50
4.2 The count of constraint types in our space of constraint types with a certain orthogonality. This is for the orthogonality measure based on the tuple representation of a generalized constraint type. 55

List of Figures

2.1 Some of the object types available in *Super Mario Maker* for designers to create levels. In *Super Mario Maker*, players can create and share their own levels. 6
2.2 Four individual puzzles from *Professor Layton and the Curious Village*. Each puzzle is distinct, and would require us to specify the objects, mechanics and goals of each puzzle to have a playable puzzle.[1] 9
2.3 Three *Sokoban* levels.[2] 10
2.4 The different objects that can make up a *Sokoban* level. Note that some of these objects represent when two objects are in the same place (like the "Filled Target" and "Avatar on Target"), and the "Empty" object represents clear space. 11
2.5 In *Tombed II*, players can use their shovel to make blocks of the same color disappear (right side). Suppose the designer wants to add a switch that opens a door when the player strikes it (left side). It is possible to implement a switch and door using the existing blocks and mechanics as shown on the right side. Anthropy encourages designers in this example to not add the switch and door [27]. 13
2.6 A diagram showing how Orthogonal Unit Differentiation can be applied to the enemies in *Doom*.[3] 15
2.7 Having orthogonal object types can lead to more interesting levels if they are as different as possible. If we think of object types as colors, two objects are not that orthogonal if their colors are similar, while two objects are orthogonal if their colors are very different. 16
2.8 A matrix of object type combinations. This can help designers create levels in a systematic way. This matrix was used by game designer Mark Brown for his puzzle game *Mind Over Magnet*.[4] 17
2.9 The basic components of a SBPCG generator: possibility space, evaluator and search procedure 18

2.10 Three environmental puzzles from *The Witness* The player must draw a line in the dotted area starting from the circular part.[5] 20
2.11 Panel puzzles from an early part of the game. These panel puzzles have been solved. [6] 21
2.12 Four panel puzzles that cannot be solved by just looking at the panels only. [7] 21
2.13 A basic level with a start location and an end location. 22
2.14 A tree with nodes representing states and edges representing states connected by actions. Note that player are able to undo actions; if the player goes up, then they can undo their action by going down. . . . 22
2.15 The player cannot have the path cross itself. Thus, they have no actions available to them, except to undo their last move (go right). 23
2.16 All possible solutions to the 2×2 empty level with the start location on the bottom-left and end location on the top-right. 23
2.17 We only focus on levels like (a) that have a single start location on the bottom-left and single end location on the top-right. We do not consider levels like (b) that have multiple start and end locations. . . 24
2.18 The above level has been divided into two regions, A and B. Any self-avoiding path that goes from a start location to an end location will partition the cells of the grid. 24
2.19 An example level with only separation constraints. Figure (a) shows the initial state of the level, while (b) shows the unique solution to the level. 26
2.20 A self-avoiding path that does not solve the example level. The cells with a separation constraint and a green circle in the corner means that the constraint piece is satisfied, while a red circle means that the constraint piece is not satisfied. 26
2.21 An example level with only star constraints. Figure (a) shows the initial state of the level, while (b) shows the unique solution to the level. 27
2.22 An example level with only tetris constraints. Figure (a) shows the initial state of the level, while (b) shows the unique solution to the level, and (c) shows how the tetris shapes can be arranged so that the tetris constraints are satisfied. 28
2.23 An example level with only triangle constraints. Figure (a) shows the initial state of the level, while (b) shows the unique solution to the level. 28

3.1 All 16 levels that comprise $\mathcal{L}_{2\times 2}(\texttt{separation}, 4)$. 31

3.2 The 4 levels from $\mathcal{L}_{2\times2}(\texttt{separation}, 4)$ that have a unique solution. . 33
3.3 The two solution paths that comprise $\mathcal{D}(\mathcal{L}_{2\times2}(\texttt{separation}, 4))$.The first solution is `RURU` and the second solution is `URUR`. 33
3.4 The four levels from $\mathcal{L}_{2\times2}(\texttt{separation}, 3)$ that have a unique solution. 35
3.5 The diagram shows the sets $\mathcal{P}(\mathcal{L}_{2\times2}(\texttt{separation}, i))$ for $1 \leq i \leq 4$. . . 36
3.6 The cardinalities of $\mathcal{P}(\mathcal{L}_{3\times3}(\texttt{separation}, i))$ for $1 \leq i \leq 9$. The separation constraint can be either black or blue. 38
3.7 The cardinalities of $\mathcal{P}(\mathcal{L}_{3\times3}(\texttt{star}, i))$ for $1 \leq i \leq 9$. The star constraint can be either black or blue. 38
3.8 The cardinalities of $\mathcal{E}(\mathcal{L}_{3\times3}(\texttt{separation}, \texttt{star}, i))$ for $1 \leq i \leq 9$ 41
3.9 Level (a) only contains separation constraints, while level (b) only contains star constraints. Level (c) contains both separation and star constraints, and the unique solution to the level (the dark blue path) is not expressible by any level with only separation or star constraints alone. 42
3.10 All three levels have the same unique solution path. Level (a) contains only 5 separation constraint pieces, while level (b) only contains 8 star constraint pieces. Level (c) contains both separation and star constraints, and is able to the same unique solution path as the other two levels, but now with only four constraint pieces. 42
3.11 When calculating orthogonality, we look at how K_* interacts with each existing constraint in $\mathcal{K}$ and see if the existing object type and K_* result in new solution paths not possible with either alone. This diagram is for $\texttt{express}(\cdot)$ 44
3.12 When calculating orthogonality, we look at how K_* interacts with each existing constraint in $\mathcal{K}$ and see if the interaction is unique. This diagram is for $\texttt{diff}(\cdot)$ 44

4.1 All levels with five singleton and star constraints that have a unique solution that is solution path 56. 51
4.2 Solution path 56 is expressible by the tetris and triangle constraints with only four constraint pieces in either case. 51
4.3 Two levels with the same unique solution path. However, the reasoning is a bit different. 57

Abbreviations

AGD Automated Game Design.

ASP Answer-set Programming.

EPCG Exhaustive Procedural Content Generation.

ERA Expressive Range Analysis.

GDC Game Developers Conference.

PCG Procedural Content Generation.

PDDL Planning Domain Definition Language.

RPG Role-playing Game.

SBPCG Search-based Procedural Content Generation.

VGDL Video Game Description Language.

Glossary of Terms

Combinatorial Games Video games which consist of levels that are made from a common set of object types.

Doom A 1993 game about shooting enemies.

Flappy Bird A 2013 video game that has the player safely navigating a bird avatar through pipes.

Level A particular configuration of object types. Also sometimes referred to as a puzzle when referring to a puzzle game.

Mega Man A video game series that often includes fights against boss enemies.

Minecraft A video game, originally released in 2009, in which the game world is procedurally generated.

Myst A 1993 puzzle game.

Object A part of a level. Objects can have types, where objects of the same type behave similarly.

Portal A series of games: the original *Portal*, released in 2007, and its sequel, *Portal 2*, released in 2011.

Procedural Content Generation (PCG) The algorithmic generation of game content, including levels and mechanics.

Professor Layton and the Curious Village A 2007 game about solving different puzzles in a village.

Self-avoiding walk A path from a start location to an end location that does not intersect with itself.

Sokoban A video game first published in 1982. It is about pushing crates onto target blocks.

Soma A 2015 video game that has horror elements. In addition, it also has puzzles that connect to the narrative of the game.

Sudoku A game about filling in a grid with numbers to satisfy constraints involving the numbers in the grid.

Super Mario Maker A 2015 video game that lets players create levels similar to the ones in *Super Mario Bros.*.

Super Mario Bros. A 1985 video game about platforming.

The Witness A 2016 puzzle video game about drawing paths.

VVVVVV A 2010 video game that has puzzle elements. The game is about controlling the direction of gravity.

Chapter 1

Introduction

What makes a video game "good"—that is, what is the criteria by which the quality of a game is determined? Firstly, it should be noted that individuals have their own subjective preferences, and that game designers have their own design goals and objectives that they pursue in development. Thus, everyone has their own idea of what makes a game of high quality. In addition, every proposed criteria for the quality of a game is bound to be challenged with counter-example video games that are considered excellent by some, but that do not align with the given criteria. In short, games are diverse, and there exists no universally applicable criteria of game quality that addresses that diversity.

Instead of presupposing a universal criteria of game quality that designers must strive to meet, we recognize that there exist many different qualities and properties that designers might want in their games. For example, a designer might want their game to be difficult, balanced or fun. However, these qualities are often only intuitively understood, with different designers having dissimilar interpretations of such qualities. Recently, researchers and designers have explored how to formalize and measure certain game qualities—that is, they have created functions that return scalar values representing the degree to which a game is a certain quality. We view such formal measures as lenses [1]; each formal measure provides a certain perspective or lens on a game. Each lens asks some question about a game, and the answer

to the question is meant to provide insight into a game. This means that a formal measure of some game quality is not meant to be prescriptive—not all games need to be difficult, balanced or even fun. Additionally, a measure may not be applicable to every game; a measure can make assumptions that restrict the games that the measure can be used on, such as a game having to be single-player.

In this thesis, we introduce and formalize the game quality of **orthogonality**. This is not a new term in the game design community; instead, we are building on ideas that have been discussed in the game design community, in particular by game designers Jonathan Blow and Marc Ten Bosch [2]. Overall, orthogonality has not been heavily discussed in academic papers, and instead most of the work on orthogonality comes from game conferences, such as the Game Developers Conference (GDC), blogs and Youtube videos. This does not mean that orthogonality is not related to qualities and concepts that have been discussed by the academic community (see Related Work), but that the notion of orthogonality that this developed in this thesis largely comes from non-academic sources. These non-academic sources of knowledge are important; we use them to develop our notion of orthogonality and they will inform design decisions for our measure of orthogonality. But what is orthogonality exactly? Orthogonality, as we develop it in this thesis, comes from a game aesthetic[1] introduced by Jonathan Blow and Marc Ten Bosch in their IndieCade 2011 presentation. Their aesthetic emphasizes that designers should try to create the richest space possible and then explore it completely, where richness implies that designers can create a lot of levels that players find engaging. For example, consider the popular puzzle game *Sokoban*, which requires the player to push crates into target locations. There is an abundance of *Sokoban* levels that can be found alone, and there exists a dedicated community that creates and play those levels.

The concept of orthogonality that we introduce in this thesis is applicable when

[1]We use the term *aesthetic* to refer to some qualities or properties that a designer might want to pursue in their game.

we are changing a game. For example, in *Sokoban*, we can add a new object that the person can be on top of, but only once, and if the person is on top of the block more than once, the player loses the puzzle. Our notion of orthogonality that we formalize in this thesis asks the following: **Compared to the existing game and its space of levels, does the object type add quality, novel and diverse levels?** Essentially, we want many, different levels that are potentially enjoyable and which are different from existing levels.

Why quality, novelty and diversity? We believe that these characterize orthogonality based on existing ideas discussed in the puzzle game community. However, we note that this definition of orthogonality is a working definition, and other components could be added.

Is orthogonality not just captured by other concepts like fun [3], balance [4] or difficulty [5]? We believe that orthogonality represents something different from existing concepts that have been developed. In particular, our focus on analyzing a space of levels to evaluate a game in our measure is distinctive.

In this thesis, in order to evaluate diversity and novelty, we will examine the similarity of levels by looking at their solutions: levels are considered different if they have different solutions, i.e. different sequences of actions that complete a level. In order to calculate orthogonality, we need to be able to select—that is, generate—levels from a space of levels. This can be done using procedural content generation (PCG) methods, which focus on the algorithmic generation of game content, including levels. The concept of orthogonality developed in this thesis is for games similar to *Sokoban*, in which each individual level is assembled out of objects that are reused across levels. Such games are called combinatorial games [6].

Our focus in this thesis is not on *Sokoban*, but another popular combinatorial puzzle game, *The Witness*. In this thesis, we have the following contributions. We introduce a way to measure the orthogonality of object types in *The Witness*. We create a space of object types in *The Witness*, which we use to generate a new object

type that we analyze, including by conducting an expert evaluation. In addition, we provide an analysis of the existing object types in *The Witness* using our measure of orthogonality.

The thesis is structured as follows. In the next chapter, we cover the background material needed to understand this thesis, including game design background, procedural content generation, and we describe the game *The Witness*. In Chapter 3, we cover orthogonality in full; we describe what orthogonality is and how we concretely measure it. We then discuss experimental results. Finally, we cover related work, and end with future work and a conclusion.

Chapter 2

Background

This chapter covers the necessary background information needed to understand this thesis. In particular, we cover the following three topics: (1) game design terminology and concepts, (2) procedural content generation (PCG), and (3) the puzzle game *The Witness*.

2.1 Game Design

This section covers game design terminology and concepts. Specifically, we discuss the following topics: (1) game elements, such as levels and mechanics, (2) combinatorial games, and (3) the aesthetics of combinatorial games.

2.1.1 Game Elements

What are the various parts of a video game? Video games are multifaceted—designers must simultaneously consider the visuals, audio, narrative, levels, gameplay and other facets of a game in its creation [7]. In order to make it easier to describe and analyze video games, game designers and researchers have attempted to formalize aspects of game design [8]; approaches have looked at defining game elements [9, 10] and classifying game elements [11, 12]. Similarly, we will elaborate on the game elements that we refer to throughout this thesis to familiarize readers with such terminology. The game elements that we detail are, in order, *objects*, *levels*, *mechanics*, *goals*, and *game-*

play. Note that the game elements we discuss are not meant to be comprehensive—for example, we do not discuss narrative—instead, we discuss what we believe is needed to understand this thesis. We use *Super Mario Bros.* as a running example, since it is well-known.

Objects:

Game objects refer to the constituent pieces that are used to create levels. For example, in *Super Mario Bros.*, game objects include the Goombas and ground tiles that compose the levels in the game. Objects do not need to be singular; instead, objects can share a *type* (see Figure 2.1 for an example of object types in *Super Mario Maker*). In addition, objects can possess attributes that hold information about the object; for example, the location of objects in *Super Mario Bros.*.

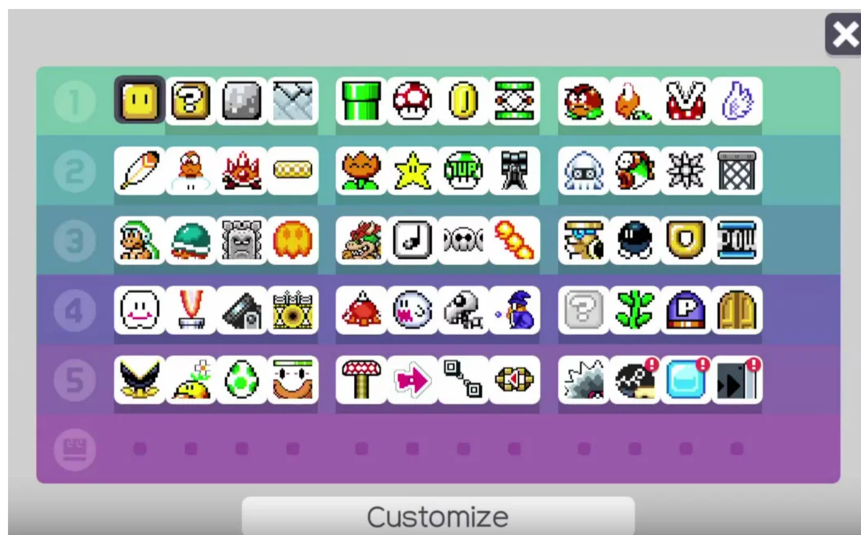

Figure 2.1: Some of the object types available in *Super Mario Maker* for designers to create levels. In *Super Mario Maker*, players can create and share their own levels.

Levels:

A level (also sometimes called a world or map) refers to a specific configuration of objects and potentially other variables that defines a particular spatial problem for players. Levels often segment a game into distinct parts [13]. For example, *Super*

Mario Bros. is split into distinct levels.

Mechanics:

There exist many definitions of mechanics in the literature, with these definitions having both overlapping and distinct parts [14]. We use the term mechanics to refer to how a level evolves according to player action. In particular, we assume that mechanics specify how objects interact and change over time, both with input by the player and when the player does not input anything. The implementation for the mechanics associated with an object can be quite complex; for example, in *Super Mario Bros.* there must be code that handles Mario's collisions, physics, how he collects coins, and the exact way his jumping works.

Goals:

Besides mechanics, designers often speak of what needs to be done in a level, i.e the goals. Although there are other types of goals, our focus in this thesis is on just completing a level [15]. In particular, we assume that the goal of a level is to satisfy a win-condition; for example, in *Super Mario Bros.*, the win-condition for a level may be to reach a flag pole. Levels may allow for additional goals [16]. In *Super Mario Bros.*, the player might try to collect coins. We do not focus on additional goals that can be pursued in a level, and instead we focus only on what is essential to completing a level.

Gameplay:

Gameplay refers to the interaction between players and the game. It may include the strategies that a player uses to complete the game, and the other things that the player has learned that helps them play the game. However, in this thesis, gameplay refers to what the player does–the actions they take—in a level in pursuit of completing the goals of a level. If a level has a unique solution that means there is only one way

for the player to complete the level (that is, there is a single sequence of actions that will result in the goals being met).

2.1.2 Combinatorial Games

In this thesis, we largely consider combinatorial single-player video games (in short, combinatorial games). In particular, we focus on combinatorial games that could also be considered puzzle games. To better understand combinatorial games, we discuss the related notion of heterogeneous games [6].

Heterogeneous:

Heterogeneous games are video games in which each problem (i.e. level) is distinct; each level can have its own mechanics, goals and objects that are never used in other levels. Examples of heterogeneous games include *Myst*, and titles in the *Professor Layton* and *WarioWare* series. In Figure 2.2, we show some example levels (puzzles) from the heterogeneous game *Professor Layton and the Curious Village.*

Combinatorial:

Combinatorial games are games in which each level is constructed from a common set of objects, and each level has the same mechanics and goals. For example, in the game *Sokoban*, each level is made up of walls, crates, target blocks and avatars (see Figure 2.4 for the objects). In each level, the avatar can push a block one space forward if there is an open space in front of it, and the goal of each level is to have a crate on every target block. In Figure 2.3, we show three *Sokoban* levels. However, these three *Sokoban* levels do not represent every single *Sokoban* level imaginable; instead, there is an infinite space of *Sokoban* levels if levels are allowed to be of any size. This space comes from arranging objects in different configurations, and from the fact the mechanics (and the goals) are defined in terms of the object types and are consistent across levels [17]. In general, a combinatorial game gives rise to a (possibly infinite)

[1] https://strategywiki.org/wiki/Professor_Layton_and_the_Curious_Village/Puzzles_1-25

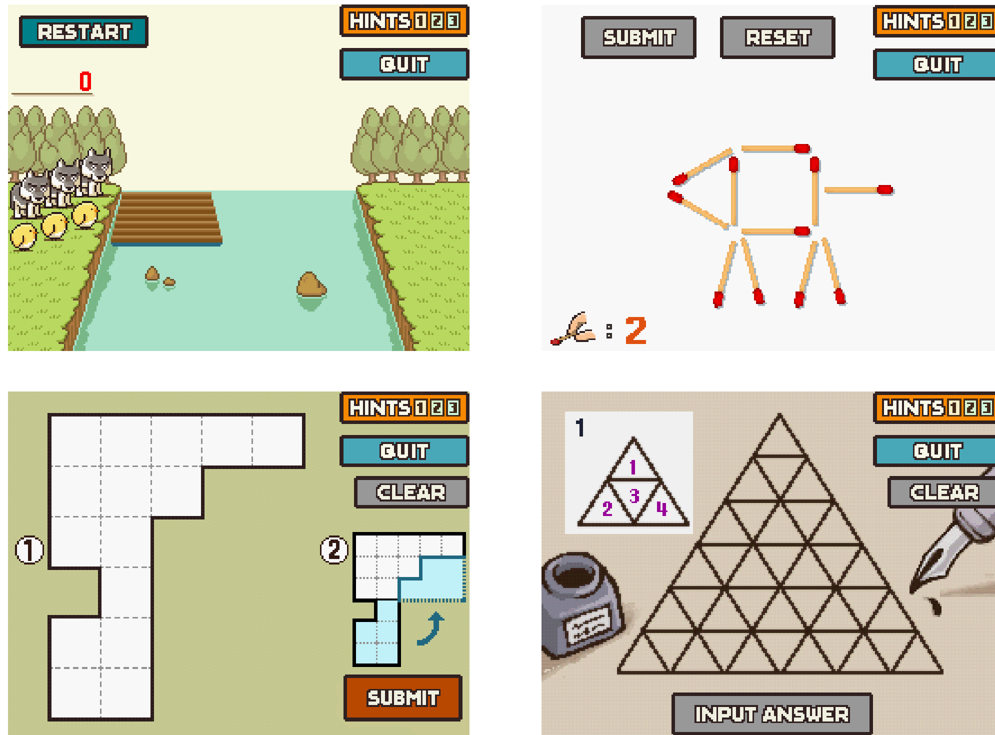

Figure 2.2: Four individual puzzles from *Professor Layton and the Curious Village*. Each puzzle is distinct, and would require us to specify the objects, mechanics and goals of each puzzle to have a playable puzzle.[1]

space of levels; Cook describes this as the possibility space of a game.[2] This space of levels represents every imaginable level that one could create for a combinatorial game, even levels that are not solvable.[3] Other examples of combinatorial games include *Super Mario Bros.*, *Portal 2*, and *The Witness*, along with Japanese logic puzzles such as *Sudoku* [18].

In addition to particular games, there exist languages and tools that can help designers and researchers create combinatorial games. PuzzleScript, a popular puzzle

[2]https://www.possibilityspace.org/tutorial-generative-possibility-space/index.html

[3]*When are two implementations of the same combinatorial game actually the same?* Hobbyists and researchers have often re-implemented combinatorial games, including *Sokoban*. However, these implementations can often differ in edge-case levels. For example, in some implementations of *Sokoban*, if there are multiple avatars in a level, then applying an action results in all the avatars being moved in that direction, while other implementations result in only a single avatar being moved. See https://www.puzzlescript.net/editor.html?hack=f9a5d9ca2ceb1c68906c47c53a4d8865 for an example of the former, and https://rlbrush.app/ for an example of the latter.

[4]https://www.sokobanonline.com/play/web-archive/david-w-skinner/microban

Figure 2.3: Three *Sokoban* levels.[4]

game engine and language, provides sections for designers to specify mechanics (rules) and goals (win conditions) that apply to each level [19], along with a section to specify which particular levels appear in a game. The distinction between the generic aspects of a game (the object types, mechanics and goals), and the specific problem instances (levels) that can created from those generic aspects is seen in other languages [20, 21], including the Planning Domain Definition Language (PDDL) [22] in which the generic aspects are specified in a domain file, and multiple problem files can be created using the same domain file.

When we are talking about a combinatorial game, we will use the term *domain* to refer to the generic aspects of a combinatorial game—its object types, mechanics and goals. Given a combinatorial game domain, designers can create levels for the game domain.

In this thesis, we assume that changes to a domain are of the following type: a new type of object and its defined attributes are added to the domain, along with adjusting the mechanics and goals (the win condition) to include the new object type.

For example, in *Sokoban*, we could add colored crates and target locations that have a color attribute, along with the usual location attributes. We could define the mechanics for the colored crates to be the same as normal crates; they can be pushed forward as long as there is an open space, while the win condition of each level is

to have each colored target location have a corresponding crate of the same color on top.

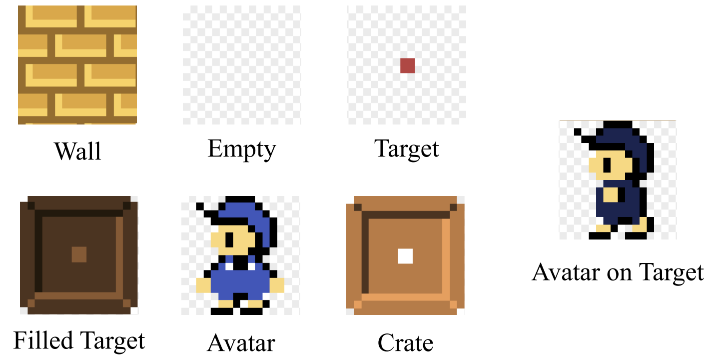

Figure 2.4: The different objects that can make up a *Sokoban* level. Note that some of these objects represent when two objects are in the same place (like the "Filled Target" and "Avatar on Target"), and the "Empty" object represents clear space.

2.1.3 Aesthetics of Combinatorial Games

There are no universal qualities that make a game "good" and instead designers and players have particular aesthetic ideals that they may want a game to satisfy [23]. For example, the creators of *Soma*, a game with heterogeneous puzzles, emphasize an aesthetic that prioritizes the narrative and contextualizing puzzles within the world [24]. We detail an aesthetic that can be applied to combinatorial games that was initially described by game designers Jonathan Blow and Marc Ten Bosch.

Before we detail their game aesthetic, we discuss some benefits of combinatorial games.[5] Since combinatorial games rely on mechanics and goals that are defined not per-level but are applied across levels, players need to learn only general rules rather a bunch of special case rules for each level. This makes planning and solving levels at least somewhat simpler since players only need to learn general rules that they can apply to new levels once the mechanics (rules) have been learned. In addition, players can learn strategies that can be applied across levels; i.e. players can generalize from

[5]See https://emshort.blog/2013/01/24/making-of-counterfeit-monkey-puzzles-and-toys/ and https://www.youtube.com/watch?v=iUi2vMZajco&t=0s for a similar discussion of the benefits of combinatorial games.

one level to another.

In their 2011 IndieCade presentation, Jonathan Blow and Marc Ten Bosch introduced a design aesthetic that can be applied to combinatorial games, although it can also be used for other types of games. Their aesthetic focuses on creating a (combinatorial) game domain that is as rich as possible and then trying to explore it as completely as possible using the levels in the game. Their design aesthetic consists of the following points [2]:

- Orthogonality
- Compatibility of Mechanics
- Generosity
- Strength of Boundary
- Richness
- Completeness
- Surprise
- Lightest Contrivance

Game designer Elyot Grant states that the puzzle designer Thomas Snyder used the term *capacity* of a combinatorial puzzle game to refer to how many puzzles (levels) a designer could create before the puzzles start becoming too similar or to how many levels a player or designer would want to solve for a particular combinatorial game domain [25]. A rich level space is one with a high capacity; designers should be able to create many different levels that players find engaging. Grant gives the following values for the capacity of different combinatorial games (these values are based on Grant's intuition):

- 2D Mazes: 1

- Portal gun (from *Portal* and *Portal 2*): 10
- *Sudoku*: 100
- *The Witness*: 1000
- Go problems: 10000+

However, creating a combinatorial game domain that is rich is not sufficient. A designer must balance the richness of the space of levels associated with the game domain (the capacity of the game) with the complexity of the domain [26]—that is, with how many different object types there are in a game domain. If a designer wanted to increase the capacity of a game domain, they could just add an additional object types (new blocks, enemies, etc.) to the game domain. If a designer wants to add something new to a game domain such as an object type, then that object type should add as many new gameplay experiences as possible compared to what already exists. Game designer Anna Anthropy notes that games should try not to have orphaned verbs, meaning that a designer should not introduce an new action and mechanic such as jumping if the player is only going to use it a few times in the game [27]. This also applies to object types (see Figure 2.5 for an example).

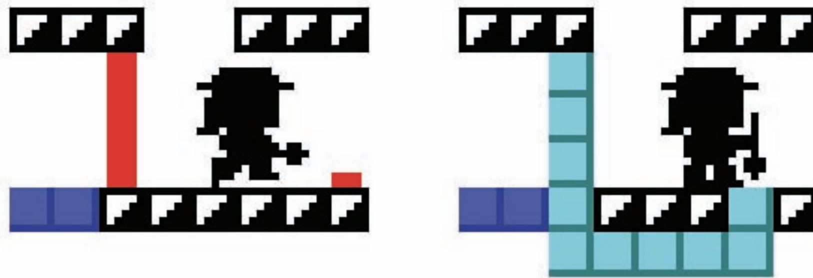

Figure 2.5: In *Tombed II*, players can use their shovel to make blocks of the same color disappear (right side). Suppose the designer wants to add a switch that opens a door when the player strikes it (left side). It is possible to implement a switch and door using the existing blocks and mechanics as shown on the right side. Anthropy encourages designers in this example to not add the switch and door [27].

Orthogonality

Blow and Ten Bosch state that orthogonality answers the following question:

> Does the potential mechanic add **enough new interesting** consequences, or are the consequences already mostly contained in the mechanics that are already there.

As stated in the introduction, we view orthogonality as the answer to the question: Compared to the existing game and its levels, does the new object type add quality, novel and diverse levels in terms of their gameplay? We can see this is similar to the definition given by Blow and Ten Bosch, but our focus is on the addition of a new object type to a game domain. But like them, we look at whether there is enough (diverse), new (novel), and interesting (quality) levels that can be created by using the new object type. We examine the different parts of the definition.

Novelty: Novelty has to do with how different some content is from existing content. We assume that we can compare levels. The notion of novelty that we develop in this thesis has to do with how different the solutions for one level are from the solutions for some other levels. Koster in his theory of fun emphasizes that having fun is predicated on learning new things [3]—with this often being operationalized as a dissimilarity function on levels [28].

Diversity: Diversity looks at the dissimilarity of content within a set. A set of content is diverse if none of the elements of the set are like each other. Again, we will look at this in terms of gameplay experiences (solutions to a level).

Quality: Quality is a measure of how *good* an individual level is by itself. For example, we may look at if the level is solvable or if it has a unique solution.

Blow and Ten Bosch's notion of orthogonality is similar to what game designer Harvey Smith calls Orthogonal Unit Differentiation—a design aesthetic that emphasizes creating game entities, such as enemies or units in strategy games, that serve different roles and have different mechanics [29]. Similarly, game designer George Fan

Figure 2.6: A diagram showing how Orthogonal Unit Differentiation can be applied to the enemies in *Doom*.[6]

emphasizes that game entities, such as enemies, should be mechanically different such that the player has to tackle the entities in different ways [30]. In Figure 2.6, we show the enemy types of the game *Doom* in terms of their behavior—notice, that the enemies are not clustered but are spread around in terms projectile versus hitscan and stay back versus charging. Designers should try to make enemies that are as different as possible, but also make it so that when enemies are combined together in combat setups that they work together nicely, i.e. they complement each other. Thus, like how cards in collectible card games or heroes in hero shooters may synergize with each other, we want object types in combinatorial games to work well together in games. This was illustrated by game designer Brett Taylor using colors as seen in Figure 2.7.

Designers have created systematic ways to create levels that deeply explore a space.[7] This has been especially true for platformers and puzzle games. The basic idea is the following: try to combine each object type with every other object type. We discuss one particular method used by game designer Brett Taylor. Taylor introduced a visualization technique to help designers think about the level possibil-

[6]https://www.youtube.com/watch?v=BEF4GVNzkUw&t=2106s
[7]https://cwpat.me/misc/puzzle-level-idea-strategies/

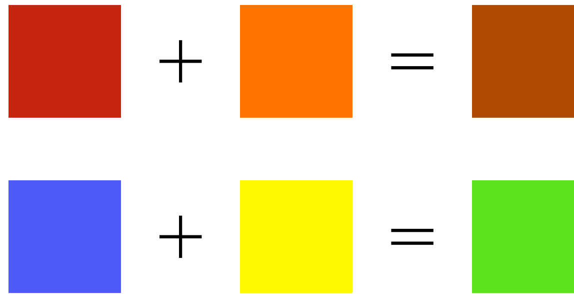

Figure 2.7: Having orthogonal object types can lead to more interesting levels if they are as different as possible. If we think of object types as colors, two objects are not that orthogonal if their colors are similar, while two objects are orthogonal if their colors are very different.

ity space associated with a game domain in terms of pairs of game object types in a level [8]. Game designers pick a cell and try to create a level (or multiple levels) that primarily uses the two object types that define that cell.

How is this method related to orthogonality? Suppose we were to add a new type of object to a game. As stated previously, the new object type should add try to maximize quality, novelty and diversity. When we add this new object type, we would want the following: (1) does the interaction between the new object type and each existing object type lead to quality levels; (2) when the new object type interacts with each existing object type, does it lead to novel levels or are the levels too similar to what is already possible with the existing object types; and (3) does the interaction between the new object type and old object type lead to diverse levels.

Another way to view orthogonality is as *additional capacity*. A new object type has high orthogonality if it increases the capacity of the game domain by a lot.

2.2 Procedural Content Generation

Procedural content generation (PCG) refers to the generation of game content—including mechanics and levels—using algorithmic means [31]. PCG has been used in games such as *Minecraft* to generate worlds, and games in the rogue-like genre heavily

[8] https://www.youtube.com/watch?v=B36_OL1ZXVM

[9] https://www.youtube.com/watch?v=akeVPZLZejY&t=233s

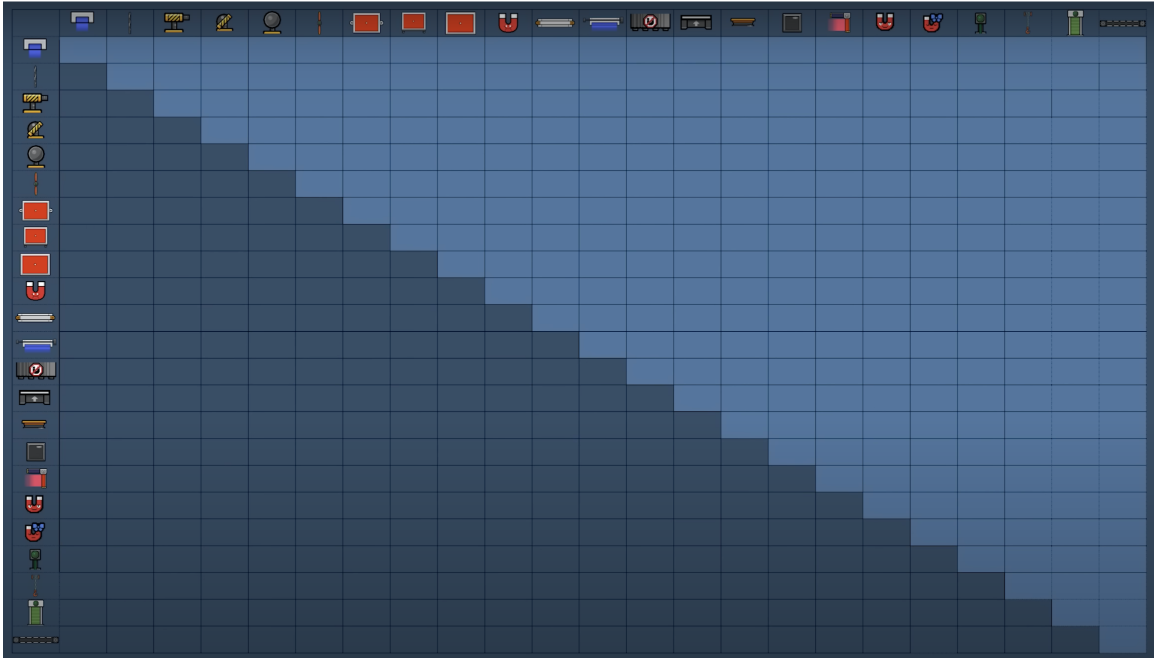

Figure 2.8: A matrix of object type combinations. This can help designers create levels in a systematic way. This matrix was used by game designer Mark Brown for his puzzle game *Mind Over Magnet*.[9]

rely on PCG as part of their game. Beyond the use of PCG in games as part of their design [32], PCG can be used during the design of a game to assist designers, even if the released game itself does not use PCG when players interact with the game. Khaled et al. suggest design metaphors—such as tool, material, designer and domain expert—for understanding the relationships between designers and PCG [33]. This thesis focuses on how PCG can be used to understand (combinatorial) game domains and to generate object types and their associated mechanics.

While there exist many PCG methods, we focus on search-based PCG (SBPCG). In their textbook, Shaker et al. provide a taxonomy of SBPCG approaches, which include the following methods [31]:

- Evolutionary search algorithms
- Exhaustive search
- Solver-based methods

We provide an overview of SBPCG approaches. In SBPCG approaches, a game content generator `Gen` consists of at least the following components:

- A possibility space P
- An evaluator E
- A search procedure S

At a high-level, the possibility space represents the content that is going to be considered by the generator, the evaluator is meant to evaluate content from the possibility space, and the search procedure is the algorithm responsible for exactly how content is generated from the possibility space and evaluated content using the evaluator.

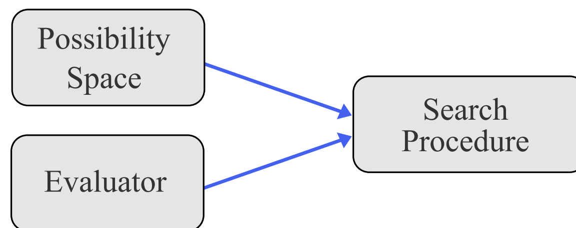

Figure 2.9: The basic components of a SBPCG generator: possibility space, evaluator and search procedure

For combinatorial games, we could imagine that the possibility space is the level possibility space associated with the game domain or a subset of that space. The evaluator could check for solvability or quantify the difficulty of a level. Finally, the search procedure could be any of the previously mentioned methods (such as evolutionary search or solver-based methods).

Exhaustive PCG

In exhaustive PCG (EPCG), all content from a finite possibility space is evaluated using the evaluator [34]. The exact search procedure does not matter too much as

long as the search procedure is able to methodically generate all content from the possibility space. Thus, EPCG can be used to enumerate and evaluate content.

SBPCG approaches often serve to filter the possibility space. For example, in solver-based methods like answer-set programming (ASP), the possibility space is constructed using choice rules and content is filtered using integrity constraints that specify the properties content should and should not have. Similarly, EPCG can be used to filter a possibility space if the evaluator is a boolean function—either it accepts the content or it rejects it. As an example, the evaluator could check if a level has a single solution.

Recently, researchers have explored generating diverse (and still high quality) sets of levels that vary in their characteristics (such as what objects are in a level) using methods such as quality-diversity search algorithms [35]. Although we do not use such quality-diversity algorithms, we use EPCG in a similar manner. Firstly, EPCG can be used to enumerate *all* content from a space, which can be used to filter that space down to a smaller set. This gives a space of content that can be examined and analyzed instead of a single optimized piece of content. Secondly, we use EPCG to generate content from different possibility spaces (different characteristics); that is, we run EPCG multiple times on different spaces.

Multiple Generators

An important aspect with such generators, although it is not specific to search-based PCG, is how to integrate multiple generators, especially generators that generate different types of content, such as mechanics and levels [7]. For example, suppose we have two generators `Gen1` and `Gen2`. *How could these interact?* It could be that they are done sequentially where the output of `Gen1` is used as input to `Gen2` or they could be structured in a nested fashion where `Gen1` uses `Gen2` as part of its evaluator. In this thesis, our mechanic generator has a nested level generator that it uses to evaluate mechanics.

2.3 The Witness Domain

The Witness is a puzzle video game released in 2016 that was developed by Thekla, Inc. and designed by Jonathan Blow. In the game, the player explores an island from a first-person perspective. The island contains a number of puzzles that the player can complete. These puzzles can be divided into two broad categories: panel puzzles and environmental puzzles.

Environmental puzzles involve the player having to view some scene from a certain perspective in order to draw a path from a starting point (see Figure 2.10).

Figure 2.10: Three environmental puzzles from *The Witness* The player must draw a line in the dotted area starting from the circular part.[10]

In this thesis, we focus on panel puzzles which are played on panels scattered around the island. Figure 2.11 shows an example of panel puzzles found in the game.

In Figure 2.12, we show more panel puzzles found in *The Witness*. Unlike the panel puzzles shown in Figure 2.11, the puzzles shown in Figure 2.12 cannot be solved by just solving looking at the individual panel puzzles. Instead, the panel puzzles must be solved by looking for visual clues in the environment or by listening to sounds present in the environment. In this thesis, we consider only panel puzzles that can be solved by only considering the panel puzzles themselves. In addition, in *The Witness*, there are panel puzzles that are played on non-rectangular grids. We focus only on rectangular grid panel puzzles in this thesis.

We now give a detailed description of panel puzzles and how they are played. Since

[10] https://www.ign.com/wikis/the-witness/River_Obelisk
[11] https://www.ign.com/wikis/the-witness/Entry_Area
[12] https://www.ign.com/wikis/the-witness/Puzzle_Solutions

Figure 2.11: Panel puzzles from an early part of the game. These panel puzzles have been solved. [11]

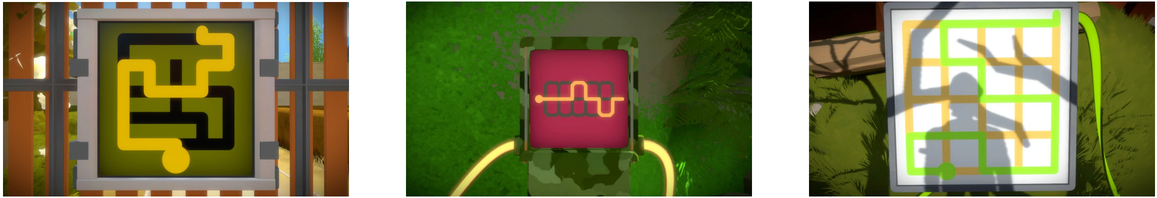

Figure 2.12: Four panel puzzles that cannot be solved by just looking at the panels only. [12]

we only consider panel puzzles in this thesis, we will refer to panel puzzles as simply puzzles or levels. In order to specify the *The Witness* domain, we need to specify the game object types, the space of levels, the mechanics and the goals. We begin by discussing the mechanics and goals.

We explain the basics of *The Witness* puzzles with a small, simple level. Figure 2.13 shows a level played on a 2×2 grid with two other parts: a start location (the circular part) and an end location (the line that protrudes out from the grid).

In *The Witness*, the goal of the player is to draw a path from the start location to the end location. From the initial state shown in Figure 2.13, there is one applicable

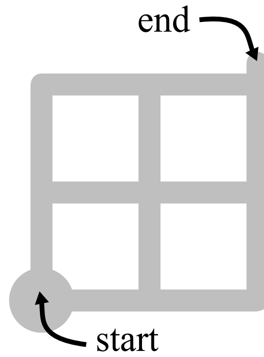

Figure 2.13: A basic level with a start location and an end location.

action: select the start location. From the state that results from selecting the start location, the player has two actions: go up or go right. Figure 2.14 visualizes the mentioned states and actions.

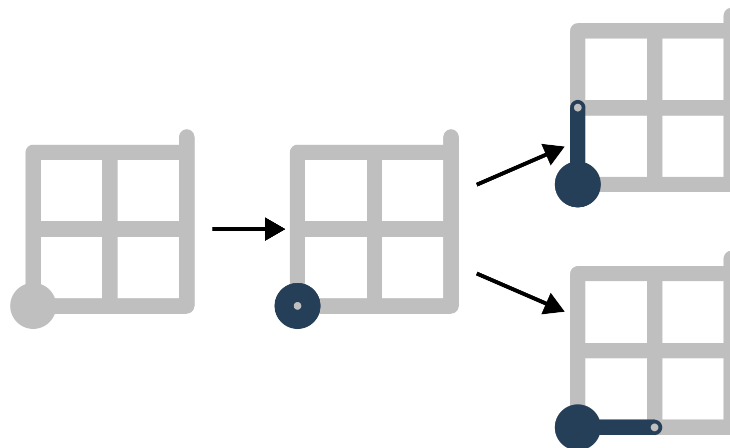

Figure 2.14: A tree with nodes representing states and edges representing states connected by actions. Note that player are able to undo actions; if the player goes up, then they can undo their action by going down.

The player can always extend their path as long as there is free space in the direction they want to extend their path. For example, in Figure 2.14 for both of the right-most states, it is possible to move up. Importantly, the player can never have the path cross itself. Figure 2.15 shows an example state where the player cannot go left, since that would result in a path that crosses itself. For the 2×2 example level from Figure 2.13, there are a total of 12 possible solutions to the level. The solutions are shown in Figure 2.16. Note that those 12 solutions are all the *self-avoiding* walks

on a 2×2 grid with the start location on the bottom-left and the end location on the top-right. [13]

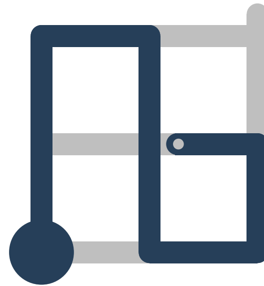

Figure 2.15: The player cannot have the path cross itself. Thus, they have no actions available to them, except to undo their last move (go right).

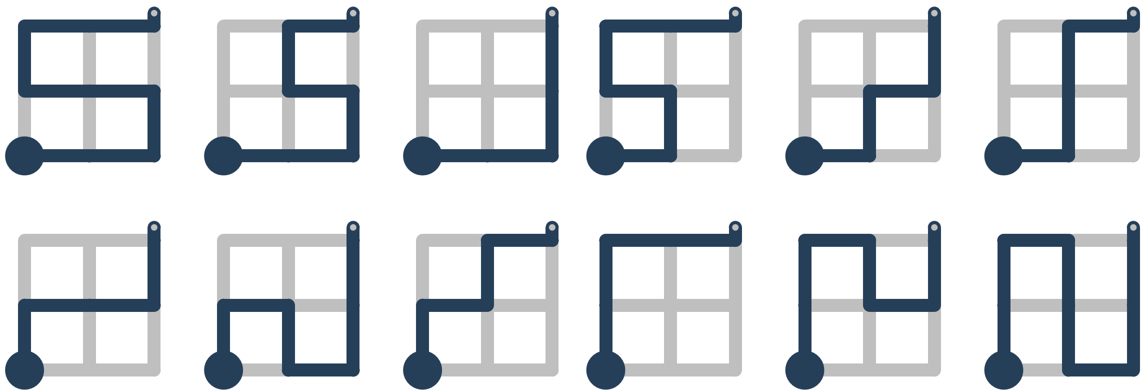

Figure 2.16: All possible solutions to the 2×2 empty level with the start location on the bottom-left and end location on the top-right.

We so far have assumed that there exists a single start location and a single end location, but in *The Witness*, it is possible to have levels with multiple start locations and end locations. The player can select any start location, and the puzzle is solved if they reach any end location. As in the case of levels with a single start and end location, the path cannot cross itself. In this thesis, we only focus on levels with a single start location and end location. In particular, the start location will be on the bottom-left and the end location will be on the top-right.

[13] See mathworld.wolfram.com/Self-AvoidingWalk.html for a brief introduction to self-avoiding walks.

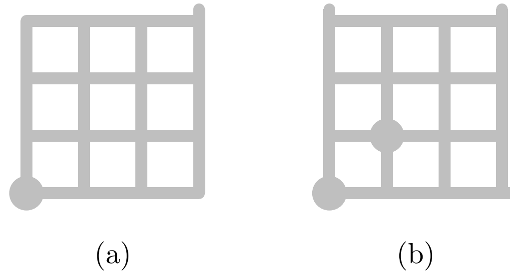

Figure 2.17: We only focus on levels like (a) that have a single start location on the bottom-left and single end location on the top-right. We do not consider levels like (b) that have multiple start and end locations.

When the player draws a path that goes from a start location to an end location that goes not self-intersect, the grid is divided into regions by the path. Each region consists of the grid cells that the region covers. For example, in Figure 2.18 there are two regions, A and B, that consist of a single cell and three cells respectively.

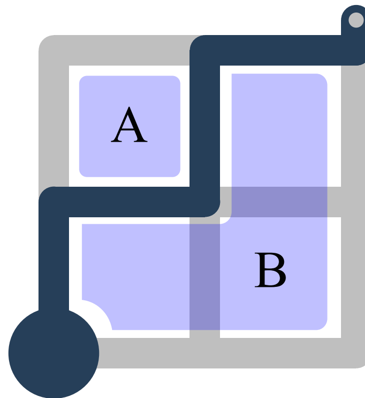

Figure 2.18: The above level has been divided into two regions, A and B. Any self-avoiding path that goes from a start location to an end location will partition the cells of the grid.

We have discussed levels from *The Witness* that have the following parts: a rectangular grid, a set of start locations, and a set of end locations. From just these parts, we have an infinite space of possible levels of different sizes, and with different starting and end locations. However, *The Witness* does not just include levels from this space of levels, it includes levels that contain other objects. In particular, levels can have region constraints. A region constraint is an object that can be placed

inside the cell of a grid. There are 4 types of region constraints that we discuss in this thesis: separation, star, tetris and triangle. [14] In this thesis, we consider the set of constraint types $\mathcal{K} = \{\texttt{separation}, \texttt{star}, \texttt{tetris}, \texttt{triangle}\}$, and all of the levels from *The Witness* we consider in this thesis can be created by using constraint types from $\mathcal{K}$. We do not consider the start and end locations as object types in $\mathcal{K}$, since we will keep these always fixed in a level (the start location in the bottom-left and the end location in the top-right). Note that constraint types have attributes such as color and an additional parameter. We refer to a constraint type with a set color and parameter as a *constraint*. When we refer to a constraint in a level, we will use the term *constraint piece*.

We start by discussing the separation constraint type, which is the first region constraint type introduced in *The Witness*.

Separation Constraint Type

Figure 2.19 shows an example level with only separation constraints. The initial state of the level is shown in (a) and the unique solution to the level is shown in (b). Note that for the solution in (b), each region partitioned by the solution path with a separation constraint piece only contains separation constraint pieces of the same color within the region. For the example level, there are three regions with two regions only having black separation constraint pieces, and a single region having blue separation constraint pieces. A separation constraint can have a color (in the example level, blue and black) and the rules of the separation constraint are the following: given a path that partitions the grid into regions, for any two separation constraint pieces in the same region, they must have the same color. If two separation constraint pieces are in the same region and have different colors, then we say that those separation constraint pieces are unsatisfied; otherwise, they are satisfied. A level with separation

[14] There is are also other region constraints in *The Witness*, such as the negative tetris and eraser/elimination constraints. We do not consider those region constraints. There are also path constraints that can be placed on the paths of a grid. We do not consider path constraints.

constraints is solvable if there exists a self-avoiding path from a start location to an end location such that all separation constraint pieces are satisfied. In Figure 2.20, we show a self-avoiding path that does not solve the level.

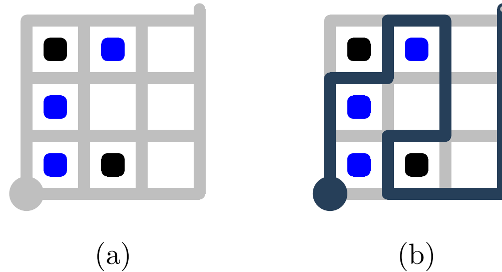

Figure 2.19: An example level with only separation constraints. Figure (a) shows the initial state of the level, while (b) shows the unique solution to the level.

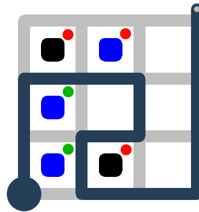

Figure 2.20: A self-avoiding path that does not solve the example level. The cells with a separation constraint and a green circle in the corner means that the constraint piece is satisfied, while a red circle means that the constraint piece is not satisfied.

We now discuss the remaining three region constraint types. They behave similarly to the separation constraint type: given a path that partitions the cells of the grid, the rules of the constraint type specify whether or not a constraint piece is satisfied in the level. A level with region constraints is solvable if there exists a self-avoiding path from a start location to an end location such that all region constraint pieces are satisfied by the path.

Star Constraint Type

A star constraint, like a separation constraint, has a set color. A star constraint piece with color c in a region is satisfied if that region contains exactly one other constraint

piece with the same color as it in the region. We show an example level and solution in Figure 2.21. Note that the other constraint piece does not have be a star constraint piece (it could, for example, be a separation constraint piece).

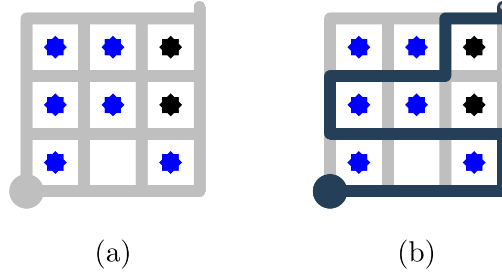

Figure 2.21: An example level with only star constraints. Figure (a) shows the initial state of the level, while (b) shows the unique solution to the level.

Tetris Constraint Type

A tetris constraint also has a color; however, we assume that a tetris constraint always has a special color that no other region constraint can have. A tetris has an additional parameter that determines its shape; the shape should be a polyomino [15]. Given a region with tetris constraint pieces inside the region, all the tetris constraint pieces in that region are satisfied if there exists a way to place the shapes of the tetris constraint pieces in such a way that the following holds: (1) every shape of a tetris constraint piece is used, (2) no shapes overlap with each other, and (3) the shapes fill the region. Note that when checking if the three conditions hold, other region constraint pieces are ignored. In addition, note that the player does not actually place any shapes; instead, this a mental check that must be done by the player. If there exists no satisfying configuration of the shapes for the tetris constraint pieces in a region then the tetris constraint pieces are unsatisfied. Figure 2.22 shows an example level with its initial state and its unique solution.

[15] https://en.wikipedia.org/wiki/Polyomino

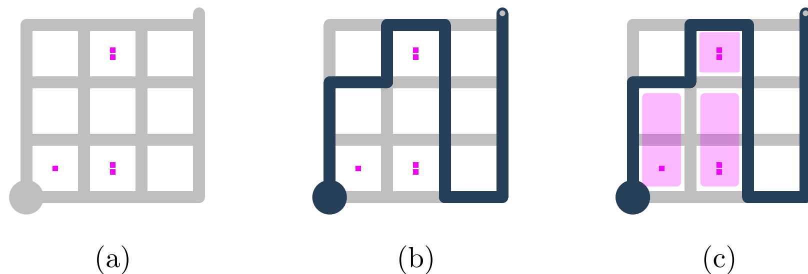

Figure 2.22: An example level with only tetris constraints. Figure (a) shows the initial state of the level, while (b) shows the unique solution to the level, and (c) shows how the tetris shapes can be arranged so that the tetris constraints are satisfied.

Triangle Constraint Type

The last constraint type from *The Witness* that we consider in this thesis is the triangle constraint type. Like a tetris constraint, a triangle constraint can have a color but we assume that a triangle constraint will always have a unique color attached to it. A triangle constraint type has an additional parameter that can take on the values 1, 2 and 3. The rules of the triangle constraint specify that it is satisfied if and only if the the number of grid edges adjacent to the cell with the triangle constraints that have a path going through them is equal to the additional parameter of the triangle constraint. Figure 2.23 shows an example level with only triangle constraints.

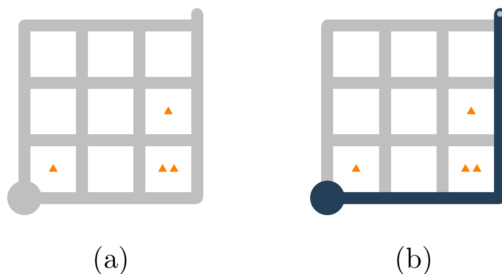

Figure 2.23: An example level with only triangle constraints. Figure (a) shows the initial state of the level, while (b) shows the unique solution to the level.

Terminology

We use the term *solution paths* to refer to self-avoiding paths/walks in a level.

Chapter 3

Defining Orthogonality

In this chapter, we define orthogonality—both specifically for *The Witness* and generally for combinatorial games. Note that our orthogonality measure for *The Witness* is just one particular implementation, and we could have made other design decisions that would have lead to a different measure. Wherever possible, we discuss why a certain design decision was made regarding our measure of orthogonality for *The Witness*.

3.1 Analyzing a Single Object Type by Itself

Terminology: A *constraint type* refers to a particular type of constraint, such as the separation constraint type. A *constraint* refers to a constraint type that has its non-location attributes set to some values. For example, we can have a black separation constraint. A *constraint piece* refers to a particular constraint in a level. For example, we could have a black separation constraint piece in the bottom-left cell of a level. We will use object type interchangeably with constraint type.

In this section, we explore finite sets of levels from *The Witness* that contain only a single constraint type. For example, we could consider sets that only contain separation constraints or only star constraints. Suppose we have some game domain D with n object types $\mathcal{K} = \{K_1, K_2, \ldots, K_n\}$, and let $\mathbb{L}$ be the possibly infinite set of levels that can be created using object types $\mathcal{K}$. For example, for *The Witness*, we

have $\mathcal{K} = \{\texttt{separation}, \texttt{star}, \texttt{tetris}, \texttt{triangle}\}$. We will use the notation $\mathbb{L}(K)$ to refer to the possibly infinite set of levels that contain only objects of type K, where $K \in \mathcal{K}$. For example, $\mathbb{L}(\texttt{separation})$ would refer to *all* levels that contain only separation constraints—along with empty constraints—and where each separation constraint in a level can be any color. However, in this thesis, we will often focus on particular object types in a level; for example, we will only focus on separation constraints that can be of two colors, black and blue. For every constraint type K, we have a set of constraints $\texttt{Con}(K)$ that we will consider. We will consider the following constraints for each constraint type:

- `separation`: $\texttt{Con}(\texttt{separation}) = \{$black and blue separation constraints$\}$
- `star`: $\texttt{Con}(\texttt{star}) = \{$black and blue star constraints$\}$
- `tetris`: $\texttt{Con}(\texttt{tetris}) = \{$tetris constraints with parameter of 1, 2 and 3$\}$
- `triangle`: $\texttt{Con}(\texttt{triangle}) = \{$triangle constraints with parameter of 1, 2 and 3$\}$

In this thesis, $\texttt{Con}(K)$ will be fixed for every constraint type. For example, we will always consider separation constraints that have a color of either black or blue. We will use the notation $\mathbb{L}(\texttt{Con}(K))$ to refer to the possibly infinite set of levels that contain only constraints from $\texttt{Con}(K)$. Since the set $\mathbb{L}(\texttt{Con}(K))$ is possibly infinite, we need to be able to consider finite subsets from this space. We use the notation $\mathcal{L}_{R\times C}(K, i)$ to refer the set of all levels that are played on a grid of size $R \times C$ that contain exactly i constraint pieces from the set of constraints $\texttt{Con}(K)$. Note that for each level in the set, there is a single start location on the bottom-left and a single end location at the top-right. In Figure 3.1, we show the set $\mathcal{L}_{2\times 2}(\texttt{separation}, 4)$—again note that $\texttt{Con}(\texttt{separation})$ will always contain only black and blue separation constraints. The set contains exactly 16 levels. Finally, we use the notation $\mathcal{L}_{R\times C}(K, i \to j)$ to refer to the following union of sets: $\cup_{t=i}^{j} \mathcal{L}_{R\times C}(K, t)$.

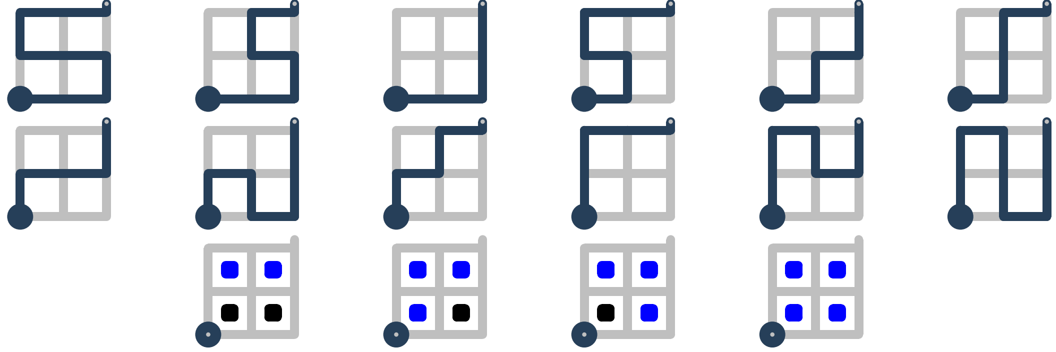

Figure 3.1: All 16 levels that comprise $\mathcal{L}_{2\times 2}(\texttt{separation}, 4)$.

We will now show how we can use a notion of orthogonality to analyze $\mathbb{L}(K)$. First, we restrict ourselves to analyzing $\mathbb{L}(\texttt{Con}(K))$ for a given, fixed set of constraints $\texttt{Con}(K)$. Previously, we stated that orthogonality is the answer to the following question:

> Compared to the existing game and its levels, does the new object type add quality, novel and diverse levels in terms of their gameplay?

Suppose that the existing game domain of *The Witness* that we consider has no constraint types defined yet, and the only levels that exist are the levels with an empty grid. The game is playable, but we consider what happens when we add a new—and only—constraint type to *The Witness* game domain. For example, what happens when we add the separation constraint type to *The Witness* when no other constraint types exist yet. We will show how we can create a measure of orthogonality for the case when we are adding a single constraint type to the game, and no other constraint types exist. Essentially, we are trying to analyze a single constraint type by itself in terms of orthogonality.

How can this be done? Remember that orthogonality consists of the following parts: quality, diversity and novelty. We begin by discussing quality.

Quality refers to any function that takes a level L from a finite set of levels $\mathcal{L}$ and returns a scalar value—that is, $\mathsf{quality} : \mathcal{L} \to \mathbb{R}$. In particular, we assume that $\mathsf{quality}(\cdot)$ is boolean; the function returns either 1 or 0. Thus, we can think of $\mathsf{quality}(\cdot)$

as a filter. For some set of levels $\mathcal{L}$, we can form the set of levels $\mathsf{quality}(\mathcal{L}) \doteq \{L \in \mathcal{L} : \mathsf{quality}(L)\} \subseteq \mathcal{L}$. In the thesis, the $\mathsf{quality}(\cdot)$ function that we use checks if a level has a unique solution. Why check if a level has a unique solution? We use this $\mathsf{quality}(\cdot)$ function for two reasons:

- A level with a unique solution is more likely to communicate a specific idea to the player. Sturtevant notes that levels with many solutions can be uninteresting since they are less likely to require the player to have a specific understanding of the level and what it is trying to teach [36].

- If a level has a unique solution, then we can map that level to a single sequence of actions instead of a set of solution paths. This simplifies comparing two levels. We just check if they have the same unique solution path.

We note that our choice of looking at solution uniqueness is nevertheless arbitrary—we could have looked at if a level is solvable and has at most two solutions, if a level has a unique solution and the solution is at least 10 actions long, or if the level is solvable and the placement of constraint pieces is symmetric.

So far, we have discussed what $\mathsf{quality}(\cdot)$ function we will use, but not how we will use it. We will use the $\mathsf{quality}(\cdot)$ function as an evaluation function in an EPCG procedure. In the EPCG procedure, we will use a generator that systemically enumerates sets of the form $\mathcal{L}_{R \times C}(K, i)$, where the generator takes as input row size R, column size C, constraint type K, and constraint piece count i. We use the EPCG generator given by Sturtevant [36] to enumerate the set. As an example, we can use EPCG to enumerate the set $\mathcal{L}_{2\times 2}(\texttt{separation}, 4)$—the set of 2×2 levels that contain exactly four separation constraints which can be either black or blue—and then filter down the set to the levels with a unique solution. In Figure 3.2, we show the four levels from $\mathcal{L}_{2\times 2}(\texttt{separation}, 4)$ that have a unique solution.

So far, we have discussed the quality part of orthogonality. We will now discuss diversity. Diversity looks at how different each level is from each other in a set. We

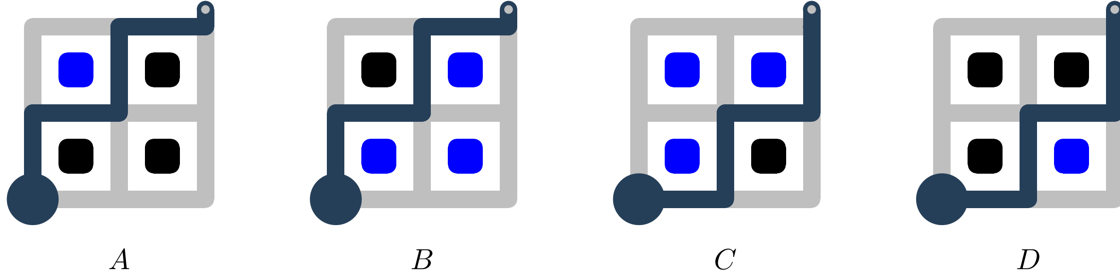

Figure 3.2: The 4 levels from $\mathcal{L}_{2\times2}(\texttt{separation}, 4)$ that have a unique solution.

will do this by comparing the gameplay of different levels, in particular, the sequences of actions needed to complete a level. Given a level L from a finite set of levels $\mathcal{L}$, let $S(L)$ be the set of solution paths to the level L. Instead of focusing on all levels, we focus only on levels that pass the $\mathsf{quality}(\cdot)$ function; in this case, if the level has a unique solution. For levels with a unique solution, $S(L)$ returns a single solution path. Given the set of levels from $\mathcal{L}$ with a unique solution, $\mathsf{quality}(\mathcal{L})$, we can construct the set $\mathcal{D}(\mathcal{L}) \doteq \{S(L) : L \in \mathsf{quality}(\mathcal{L})\}$, the distinct solution paths of levels with a unique solution. In Figure 3.3, we show the two solution paths from $\mathcal{D}(\mathcal{L}_{2\times2}(\texttt{separation}, 4))$. Looking back at Figure 3.2, for the four levels in $\mathcal{L}_{2\times2}(\texttt{separation}, 4)$ that have a unique solution, there are only two distinct solution paths: levels A and B share the same solution path, and C and D share the same solution path. Also note that levels A and B are practically the same levels except that the colors have been swapped, and similarly for levels C and D. Thus, we were able to go from four levels to only two solution paths.

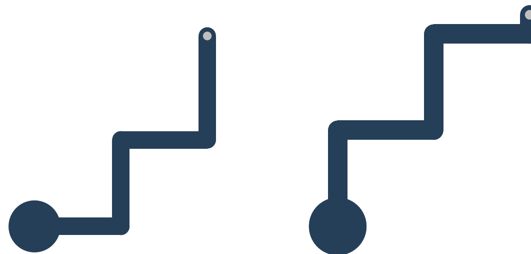

Figure 3.3: The two solution paths that comprise $\mathcal{D}(\mathcal{L}_{2\times2}(\texttt{separation}, 4))$.The first solution is `RURU` and the second solution is `URUR`.

Given the set of paths $\mathcal{D}(\mathcal{L})$, we can determine how diverse it is by taking its cardinality, $|\mathcal{D}(\mathcal{L})|$; in the above example, we have a value of 2. For different size grids, there are different numbers of total possible solution paths (self-avoiding walks) on an empty grid (see Table 3.1). We use $\mathtt{P}(R, C)$ to refer to the number of self-avoiding walks on an empty grid of size $R \times C$, with a single start location on the bottom-left and a single end location on the top-right.

height R	width C	count $\mathtt{P}(R, C)$
2	2	12
3	2	38
3	3	184
4	3	976
4	4	8,512

Table 3.1: The total number of self-avoiding paths for different sizes of empty grids. These numbers also represent the total number of solutions for different sizes of empty grids.

We can divide $\mathcal{D}(\mathcal{L}_{R\times C}(K, i))$ by $\mathtt{P}(R, C)$ to make sure that the number we get is between 0 and 1. In the previous example, we would have $\frac{2}{12}$ because on the 2×2 grid there are a total of 12 solution paths possible, and we were able to get two paths from $\mathcal{D}(\mathcal{L}_{2\times 2}(\mathtt{separation}, 4))$. Instead of taking the cardinality of the set, we could determine how different each solution path is from each other—that is, we could determine $\sum_{S,\, S' \in \mathcal{D}(\mathcal{L})} d_{\text{path}}(S, S')$ if we had a way to compare solution paths $d_{\text{path}}(S, S')$. Currently, we use the cardinality of the set to determine its diversity, because of the simplicity of the approach. We have shown how to determine the diversity of a set of levels by first filtering down to quality levels using a $\mathsf{quality}(\cdot)$ function, then converting that set into a set of solution paths, and taking its cardinality and normalizing the value.

So far, we have discussed quality and diversity, and in particular, how we can

use a notion of quality in our calculation for diversity. We will now discuss the final part of orthogonality, novelty. In Figure 3.2, we show the four levels from $\mathcal{L}_{2\times2}(\texttt{separation}, 3)$ that have a unique solution. Note that $\mathcal{L}_{2\times2}(\texttt{separation}, 1)$ and $\mathcal{L}_{2\times2}(\texttt{separation}, 2)$ do not contain any levels with unique solutions. From Figure 3.2 and 3.4, we can see the following:

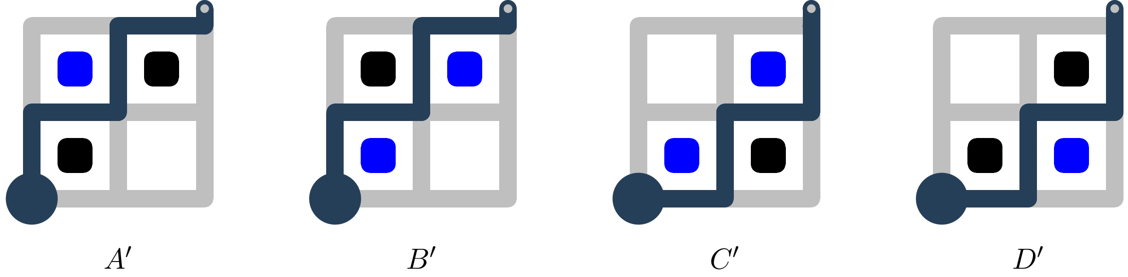

Figure 3.4: The four levels from $\mathcal{L}_{2\times2}(\texttt{separation}, 3)$ that have a unique solution.

- Levels A and A' share the same solution path, and the difference between levels A and A' is that level A contains an additional black separation constraint. We see the same relationships for the other pairs of levels.

- Since $\mathcal{L}_{2\times2}(\texttt{separation}, 1)$ and $\mathcal{L}_{2\times2}(\texttt{separation}, 2)$ do not have any levels with unique solutions, the levels in Figure 3.4 are the simplest levels (i.e levels with the fewest constraint pieces) from the set $\mathcal{L}_{2\times2}(\texttt{separation}, 1 \to 4)$ that have unique solutions.

Novelty looks at how different one set is from another set. Again, similar to what we did for diversity, we will compare levels by looking at their solution paths. Suppose we have a collection of T sets of levels $\mathcal{L}_{1:T} \doteq \{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_T\}$. Assume that these sets are ordered in some way; in particular, assume that there exists a strict partial order $<$ over $\mathcal{L}_{1:T}$. We can consider the novelty of a set $\mathcal{L} \in \mathcal{L}_{1:T}$ relative to all the sets that precede it—that is, the set $\bigcup\{\mathcal{L}' \in \mathcal{L}_{1:T} : \mathcal{L}' < \mathcal{L}\ \}$. For example, given the collection of $t_{\max}$ sets $\{\mathcal{L}_{R\times C}(K, i)\}_{i=1}^{t_{max}}$, we can order the sets by the total number of pieces in a level. Let $<_{\#}$ be the strict total order over the set $\{\mathcal{L}_{R\times C}(K, i)\}_{i=1}^{t_{max}}$ where

$\mathcal{L}_{R\times C}(K,i) <_{\#} \mathcal{L}_{R\times C}(K,j)$ if and only if $i < j$. Then we can calculate the novelty of $\mathcal{L}_{R\times C}(K,i)$ with respect to the union of all the sets that precede it in the order: $\mathcal{L}_{R\times C}(K, 1 \to i-1)$ for $i \geq 1$; define $\mathcal{L}_{R\times C}(K, 1 \to 0)$ as the empty set. Like how we incorporated quality into our calculations for diversity, we can incorporate what we did for diversity into novelty. In order to calculate novelty, we get the set of paths $\mathcal{D}(\mathcal{L}_{R\times C}(K,i))$ as we did for diversity. We also get the set of paths $\mathcal{D}(\mathcal{L}_{R\times C}(K, 1 \to i-1))$. Note that $\mathcal{D}(\mathcal{L}_{R\times C}(K, 1 \to i-1)) = \bigcup_{t=1}^{i-1} \mathcal{D}(\mathcal{L}_{R\times C}(K,t))$. Then we define the following: $\mathcal{P}(\mathcal{L}_{R\times C}(K,i)) \doteq \mathcal{D}(\mathcal{L}_{R\times C}(K,i)) - \mathcal{D}(\mathcal{L}_{R\times C}(K, 1 \to i-1))$ (we are taking the set difference of two sets). The set $\mathcal{P}(\mathcal{L}_{R\times C}(K,i))$ represents all the solution paths that can achieved in levels with a unique solution that have i constraint pieces but that are not possible with strictly fewer than i constraint pieces of only type K. In Figure 3.5, we show the sets $\mathcal{P}(\mathcal{L}_{2\times 2}(\texttt{separation}, i))$ for $1 \leq i \leq 4$ for the collection of sets $\{\mathcal{L}_{2\times 2}(\texttt{separation}, i)\}_{i=1}^{4}$

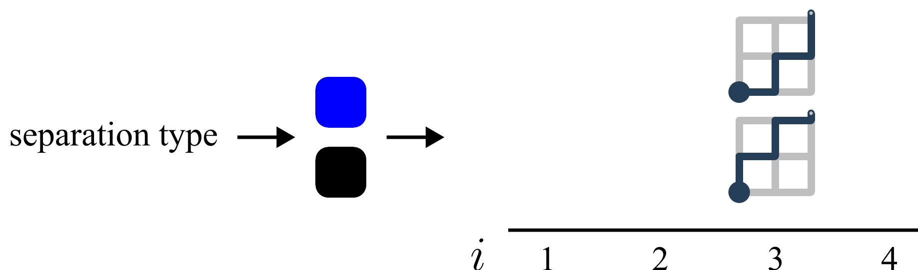

Figure 3.5: The diagram shows the sets $\mathcal{P}(\mathcal{L}_{2\times 2}(\texttt{separation}, i))$ for $1 \leq i \leq 4$.

What do we notice from Figure 3.5? First, we see that with one and two separation constraints, we are not able to get any unique solution paths. With three separation constraints, we are able to get two unique solution paths. And with four separation constraints, we are not able to get any *new* unique paths.

We note that there are several properties that hold for $\{\mathcal{P}(\mathcal{L}_{R\times C}(K,i))\}_{i=1}^{t_{max}}$:

- The path sets are disjoint. That is $\bigcap_{i=1}^{t_{max}} \mathcal{P}(\mathcal{L}_{R\times C}(K,i)) = \emptyset$. This also means that if we see a solution path $p \in \mathcal{P}(\mathcal{L}_{R\times C}(K,i))$ for some i, then there cannot exist a number of constraint pieces $i' < i$ such that $p \in \mathcal{P}(\mathcal{L}_{R\times C}(K,i'))$

- The size of the union of path sets cannot be more than $\mathtt{P}(R, C)$. That is $|\bigcup_{i=1}^{t_{max}} \mathcal{P}(\mathcal{L}_{R\times C}(K, i))| \leq \mathtt{P}(R, C)$.

We now look at grids of size 3×3. In Figure 3.6, we plot the cardinalities of $\mathcal{P}(\mathcal{L}_{R\times C}(\mathtt{separation}, i))$ for $1 \leq i \leq 9$ where the separation constraint can be either black or blue. In Figure 3.7, we show it for star constraints (again black and blue). We say a constraint type K expresses a solution path p with i constraint pieces if $p \in \mathcal{P}(\mathcal{L}_{R\times C}(K, i))$, and we say a solution path p is expressible if $p \in \mathcal{P}(\mathcal{L}_{R\times C}(K, i))$ for some $1 \leq i \leq RC$. We see the following:

- The star constraint is not very expressive by itself—meaning that it cannot express many solution paths. A level with only star constraints can only have a unique solution starting with eight pieces, and even then it is only able to express eight different paths with 8 pieces on the grid.

- The separation constraint is more expressive than the star constraint. Although is not shown in the plots because we only show the cardinalities of the path sets, every path expressible by the star constraint with eight pieces is also expressible with the separation constraint type with strictly fewer pieces.

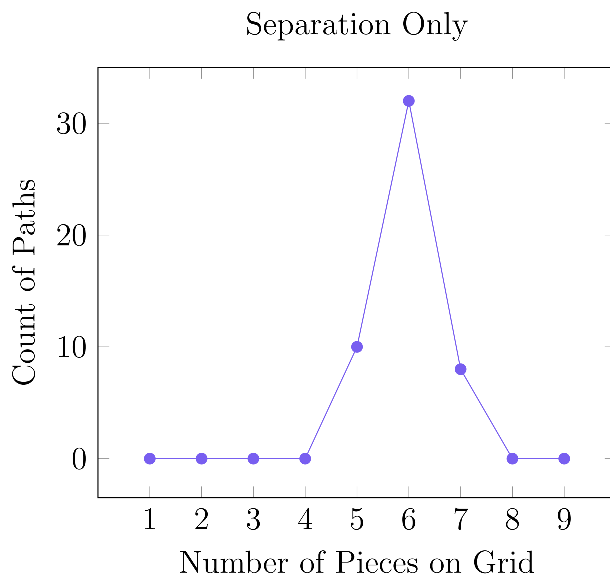

Figure 3.6: The cardinalities of $\mathcal{P}(\mathcal{L}_{3\times3}(\texttt{separation}, i))$ for $1 \le i \le 9$. The separation constraint can be either black or blue.

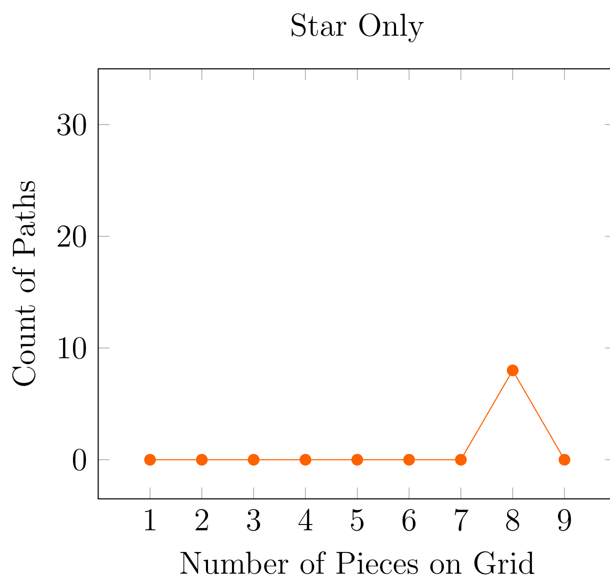

Figure 3.7: The cardinalities of $\mathcal{P}(\mathcal{L}_{3\times3}(\texttt{star}, i))$ for $1 \le i \le 9$. The star constraint can be either black or blue.

We define the orthogonality of a constraint type K by itself on a $R \times C$ grid as the following:

$$\frac{|\bigcup_{i=1}^{t_{max}} \mathcal{P}(\mathcal{L}_{R\times C}(K,i))|}{\mathtt{P}(R,C)}$$

This is equivalent to the summation of the y components in Figure 3.6 and Figure 3.7.

We have shown how we can define orthogonality by specifying quality, diversity and novelty. Novelty relies on diversity, which itself relies on quality.

Questions:

Why use EPCG to generate levels? One of the benefits of using EPCG is that we are guaranteed to not miss any level. For example, because we are using EPCG we know that if we are able to express a solution path p with five constraint pieces of a single constraint type then there is no level that has the same unique solution path as path p. This is because we exhaustively generated all levels with fewer than five pieces of the same, single constraint type. In addition, because we use EPCG, we are not going to miss a level that might contribute to the orthogonality of the constraint type.

3.2 Analyzing Two Object Types Together

In the previous section, we showed how we can analyze an object type by itself in levels in terms of orthogonality. In this section, we examine how we can analyze how two object types work together in levels in terms of orthogonality.

Suppose we have some game domain D with n object types $\mathcal{K} = \{K_1, K_2, \ldots, K_n\}$, and let $\mathbb{L}$ be the possibly infinite set of levels that can be created using object types $\mathcal{K}$. We will use the notation $\mathbb{L}(K, K')$ to refer to the possibly infinite set of levels that contain only objects of type K and K', such that $K, K' \in \mathcal{K}$. We use the notation $\mathcal{L}_{R\times C}(K, K', i)$ to refer the set of all levels that are played on a grid of size $R \times C$ that contain exactly i constraint pieces from the set of constraints $\mathtt{Con}(K) \cup$

$\texttt{Con}(K')$. Note that for each level in the set, there is a single start location on the bottom-left and a single end location at the top-right. As an example, the set $\mathcal{L}_{2\times 2}(\texttt{separation}, \texttt{star}, 4)$ would contain *all* levels that contain a total of exactly four separation and star constraint pieces. Let $\mathcal{L}_{R\times C}(K, K', i \to j)$ refer to the following union of sets: $\cup_{t=i}^{j} \mathcal{L}_{R\times C}(K, K', t)$.

We will show how we can use a notion of orthogonality to analyze $\mathbb{L}(K, K')$. We use ideas of quality and diversity that we discussed in the previous section. That means we are going to only look at levels with a unique solution, and we will examine the diversity of a set of levels by looking at what solution paths are uniquely expressible by levels in the set. More specifically, we have the following (where $S(L)$ is the set of solutions to the level L):

$$\mathcal{D}(\mathcal{L}_{R\times C}(K, K', i)) \doteq \{S(L) : L \in \mathsf{quality}(\mathcal{L}_{R\times C}(K, K', i))\}$$

We will now discuss how we can define novelty for the case when we have two types of objects in levels. Suppose we have the following three collections of sets of levels: $\{\mathcal{L}_{R\times C}(K, K', i)\}_{i=1}^{t_{max}}$, $\{\mathcal{L}_{R\times C}(K, i)\}_{i=1}^{t_{max}}$, and $\{\mathcal{L}_{R\times C}(K', i)\}_{i=1}^{t_{max}}$. Furthermore, suppose we have the following strict partial order $<$ over the union of the three collections of sets of levels: For all $i \geq 1$, we have the following: $\mathcal{L}_{R\times C}(K, i) < \mathcal{L}_{R\times C}(K, j)$, $\mathcal{L}_{R\times C}(K', i) < \mathcal{L}_{R\times C}(K', j)$, and $\mathcal{L}_{R\times C}(K, K'i) < \mathcal{L}_{R\times C}(K, K', j)$ if and only if $i < j$. In addition, we have that for all $i \geq 1$, $\mathcal{L}_{R\times C}(K, i) < \mathcal{L}_{R\times C}(K, K', i)$ and $\mathcal{L}_{R\times C}(K', i) < \mathcal{L}_{R\times C}(K, K', i)$.

Then for each set of levels $\mathcal{L}$ in the collection of sets $\{\mathcal{L}_{R\times C}(K, K', i)\}_{i=1}^{t_{max}}$, we can determine the novelty of $\mathcal{L}$ with respect to all the sets of levels that precede it in the three collections of sets of levels. We define the following:

$$\begin{aligned}\mathcal{E}(\mathcal{L}_{R\times C}(K, K', i)) \doteq \mathcal{D}(\mathcal{L}_{R\times C}(K, K', i)) - \mathcal{D}(\mathcal{L}_{R\times C}(K, K', 1 \to i-1))\\ -\mathcal{D}(\mathcal{L}_{R\times C}(K, 1 \to i)) - \mathcal{D}(\mathcal{L}_{R\times C}(K', 1 \to i))\end{aligned}$$

In Figure 3.8, we plot the cardinalities for $\mathcal{E}(\mathcal{L}_{3\times3}(\texttt{separation}, \texttt{star}, i))$ for $1 \leq i \leq 9$.

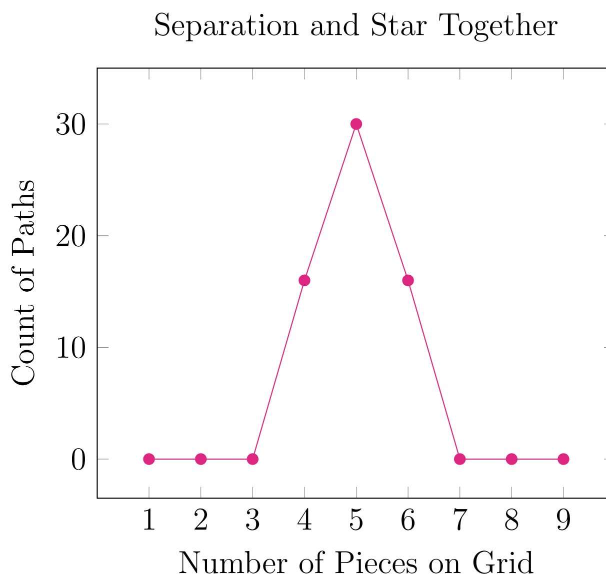

Figure 3.8: The cardinalities of $\mathcal{E}(\mathcal{L}_{3\times3}(\texttt{separation}, \texttt{star}, i))$ for $1 \leq i \leq 9$

.

What does $\mathcal{E}(\mathcal{L}_{R\times C}(K, K', i))$ intuitively measure? Essentially, a path p lies in $\mathcal{E}(\mathcal{L}_{R\times C}(K, K', i))$ for two reasons:

- The path p does not exist in $\mathcal{D}(\mathcal{L}_{R\times C}(K, i))$ or $\mathcal{D}(\mathcal{L}_{R\times C}(K', i))$ for any $1 \leq i \leq t_{max}$—that is, the path p is only uniquely expressible using both constraint types K and K' together in levels, and not any number of constraint pieces of type K and K' alone. See Figure 3.9 for example levels and paths.

- The path p does exist in $\mathcal{D}(\mathcal{L}_{R\times C}(K, i'))$ or $\mathcal{D}(\mathcal{L}_{R\times C}(K', i'))$ for some $1 \leq i' \leq t_{max}$ but $i' > i$—that is, the path p can be more efficiently expressed (i.e. fewer constraint pieces to get unique solution) using both constraint types K and K' together in levels compared only using constraint pieces of type K and K' alone in levels. See Figure 3.10 for example levels and paths.

We define the following for constraint types K and K' on a $R \times C$ grid:

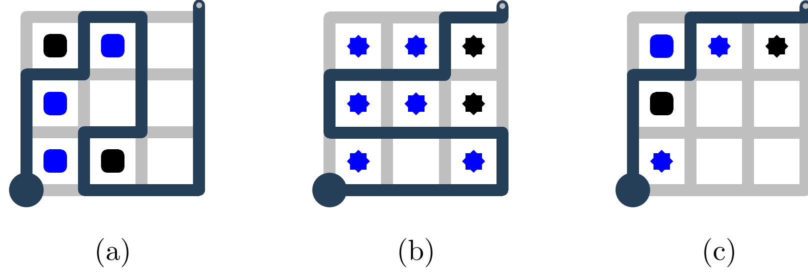

Figure 3.9: Level (a) only contains separation constraints, while level (b) only contains star constraints. Level (c) contains both separation and star constraints, and the unique solution to the level (the dark blue path) is not expressible by any level with only separation or star constraints alone.

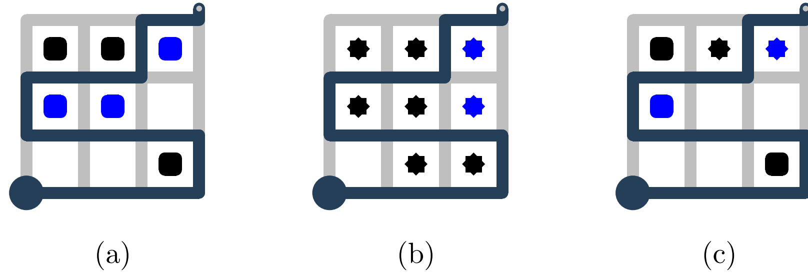

Figure 3.10: All three levels have the same unique solution path. Level (a) contains only 5 separation constraint pieces, while level (b) only contains 8 star constraint pieces. Level (c) contains both separation and star constraints, and is able to the same unique solution path as the other two levels, but now with only four constraint pieces.

$$\texttt{express}(K, K') \doteq \frac{|\bigcup_{i=1}^{t_{max}} \mathcal{E}(\mathcal{L}_{R\times C}(K, K', i))|}{\texttt{P}(R, C)}$$

It represents how well the two constraint types work together—either entirely new solutions are expressible or they can be expressed with fewer pieces. When the existing game has only a single constraint type K, and we add K', then we define the orthogonality to be the same as $\texttt{express}(K, K')$.

3.3 Complete Orthogonality Function

In the previous sections, we defined what orthogonality is when there are no constraint types and a single, new constraint type is added, and we defined what orthogonality

$\mathcal{K}$	A set of object types
$K \in \mathcal{K}$	An object type
$\texttt{express}(K, K')$	How well object types K and K' work together compared to by themselves
$\mathcal{D}(\cdot)$	A set of paths

Table 3.2: Table of Notation for Orthogonality

is when there is a single constraint type and a single new object type is added. We now examine the more general case where there are n existing constraint types, and a single new object type is added to the game. Suppose we have some game domain D with n object types $\mathcal{K} = \{K_1, K_2, \ldots, K_n\}$, and let $\mathbb{L}$ be the possibly infinite set of levels that can be created using object types $\mathcal{K}$. Suppose we add a new object type K_* to the game domain D.

We consider how three object types interact together in levels. Suppose we have three object types K, K' and K'', we can want to understand if K' and K interact differently from K' and K''. We define the following:

$$d((K', K), (K', K'')) \doteq \Big| \bigcup_{i=1}^{t_{max}} \mathcal{D}(\mathcal{L}_{R\times C}(K', K, i)) - \mathcal{D}(\mathcal{L}_{R\times C}(K', K, 1 \to i-1)) - \mathcal{D}(\mathcal{L}_{R\times C}(K', K'', 1 \to i))\Big| / \texttt{P}(R, C)$$

Notice that this very similar to how $\texttt{express}(K, K')$ is defined; it is high if either new paths can be expressed or if paths can be expressed more efficiently.

We define the orthogonality of object types K_* and K' as, where $\mathcal{K}$ is a set of other object types:

$$o(K_*, K', \mathcal{K}) = \texttt{express}(K_*, K') * \texttt{diff}(K_*, K', \mathcal{K})$$

where $\texttt{diff}(K, K', \mathcal{K}) = \min_{K'' \in \mathcal{K} - K'} d((K', K), (K', K''))$

The orthogonality of a new object type K_* and a set of existing object types $\mathcal{K}$ is the average of the orthogonality between K_* and each object type in $\mathcal{K}$:

$$o(K_*, \mathcal{K}) = \frac{\sum_{K' \in \mathcal{K}} o(K_*, K', \mathcal{K})}{|\mathcal{K}|}$$

We visualize what `express`$(\cdot)$ and `diff`$(\cdot)$ are in the below figures. The light blue cells refer to sets of levels that we are comparing against (all these cells represent sets of levels with only old object types), and the dark cells are the sets of levels with the new object type K_*.

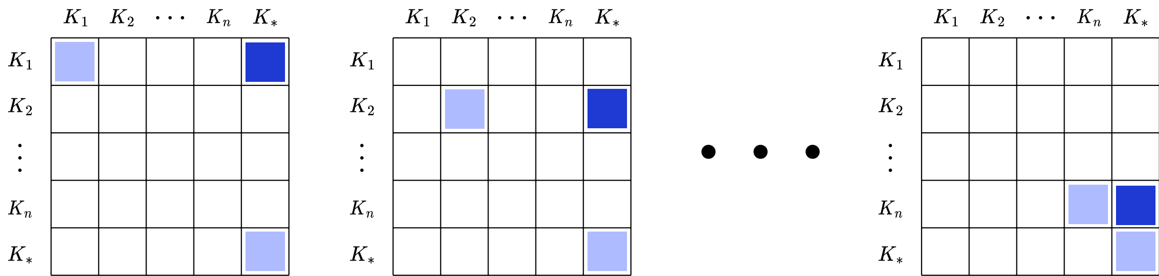

Figure 3.11: When calculating orthogonality, we look at how K_* interacts with each existing constraint in $\mathcal{K}$ and see if the existing object type and K_* result in new solution paths not possible with either alone. This diagram is for `express`$(\cdot)$

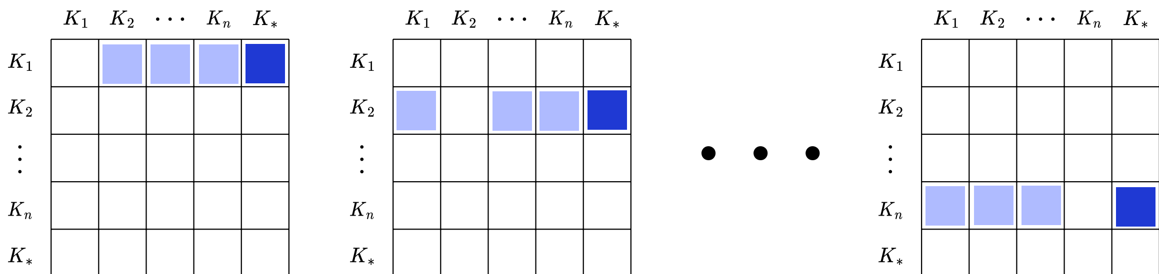

Figure 3.12: When calculating orthogonality, we look at how K_* interacts with each existing constraint in $\mathcal{K}$ and see if the interaction is unique. This diagram is for `diff`$(\cdot)$

Questions:

Why only look at pairs of object types together in levels? The first reason is that many levels in *The Witness* only contain constraint pieces of two types. The second

reason is that puzzle designers often create levels which contain only two types of objects.[1] The last reason is that it is a simple approach that could be extended further; we could look at a trio of object types in levels or we could look at multi-sets of constraints in levels.

3.4 General Definition

In the previous sections, we developed a specific measure of orthogonality for *The Witness*. In this section, we discuss a general definition of orthogonality.

Previously, we stated that orthogonality is the answer to the following question:

> Compared to the existing game and its levels, does the new object type add quality, novel and diverse levels in terms of their gameplay?

We restrict our discussion of orthogonality to (single-player) combinatorial games, in which we examine what happens when we add a new object type to a game domain. More formally, suppose we have a game domain D with n object types $\mathcal{K} = \{K_1, \ldots, K_n\}$ and set of levels $\mathbb{L}$ that can be constructed from $\mathcal{K}$ and another game domain D' with $n+1$ object types $\mathcal{K}' = \{K_1, \ldots, K_n, K_*\}$ and set of levels $\mathbb{L}'$ such that $\mathcal{K} \subseteq \mathcal{K}'$ and $\mathbb{L} \subseteq \mathbb{L}'$. Thus, game domains D and D' are the same except D' now has levels that contain the new object type K_*. Note that $\mathbb{L}$ and $\mathbb{L}'$ are both possibly an infinite set of levels—they represent *all* possible levels that could be created for game domains D and D' respectively. In order to calculate orthogonality, we need to have finite sets of levels. Suppose that we have the finite sets of levels $\mathcal{L} \subseteq \mathbb{L}$ and $\mathcal{L}' \subseteq \mathbb{L}'$, then orthogonality is a function that takes in those two finite sets of levels and returns a scalar value; the higher the number the higher the orthogonality. So far, we have not discussed the exact details of what a measure of orthogonality must calculate.

In general, an orthogonality measure must look at the following:

[1] https://cwpat.me/misc/puzzle-level-idea-strategies/

- Quality: Are the levels in each set high quality? Here, quality refers to any property that can be calculated by just looking at each level by itself, and not other levels in a set. We could see if the level is solvable, has a unique solution, has symmetric placement of objects or any other property we might want to consider. Thus, one design choice that must be made is what levels are considered quality in a given orthogonality measure.
- Diversity: How different are the levels in each set from each other? Again, this a design choice that must be made in terms of how exactly levels with be differentiated from each other.
- Novelty: How different are the sets from each other? Similar to diversity, but across sets instead of within a set.

Chapter 4

Experimental Results

In this chapter, we answer a series of questions about the orthogonality measure for *The Witness* that we detailed in the previous chapter. Broadly, we examine the orthogonality measure in the following ways:

- We create a space of constraint types that includes the separation and star constraint types.
- We examine this space in two ways: (1) we exhaustively evaluate the space of constraint types in order to understand the space as a whole, and (2) we examine the constraint type with the highest orthogonality.
- We manually create a curriculum for the constraint type with the highest orthogonality, and then perform an expert evaluation with Patrick Traynor, a puzzle game designer. We also discuss the limitations of our expert evaluation.
- We look at alternative ways to define orthogonality. In particular, we show the importance of defining orthogonality on the gameplay afforded by constraint types, rather than the form of a constraint type.
- We examine how we can use the orthogonality measure to understand the existing constraint types in *The Witness*.
- Finally, we discuss alternatives to looking at solution paths.

4.1 Space of Constraint Types

The space of constraint types that we specify in this section generalizes the separation and star constraint types. Thus, all constraint types in this space have to do with color and counting. We call a constraint type from this space a **generalized constraint type**. Each constraint type in the space is represented as a tuple $\langle$ `X`, `Comp`, `Num` $\rangle$ where `X` is a non-empty subset of the set $\{00, 01, 10, 11\}$, `Comp` is from the set $\{\texttt{all}, \texttt{None}, ==, \geq\}$, and `Num` is from the set $\{1, 2, \texttt{None}\}$. `Num` is `None` if and only if `Comp` is `all` or `None`. Each generalized constraint type can have a color, just like separation and star constraints.

The interpretation of a generalized constraint type $\langle$ `X`, `Comp`, `Num` $\rangle$ is as follows. Suppose we have a level and a self-avoiding walk that segments the grid of the level into regions. We will detail what determines whether or not a generalized constraint piece in a region R is satisfied. Remember that the player has solved a level if all the constraint pieces in a level are satisfied by the self-avoiding walk. For each generalized constraint piece c_1 in a level within the region R, we look at its relationship to all other non-empty constraint pieces in the same region R—including itself and other generalized constraint pieces. In particular, for a generalized constraint piece c_1 and a non-empty constraint piece c_2—both from the region R—we see if they have the same type and the same color. For example, if constraint pieces c_1 and c_2 have the same type and color, we associate the pair of constraint pieces with the string 11 and if they have the same type, but not the same color, then we associate them with the string 10. We then check if this string is in the set `X`. We repeat this for all the other non-empty constraint pieces in the region R. We count the number of times this occurs within the region R for the generalized constraint piece c_1. We denote this quantity by the term $\text{count}_{\texttt{X}}(c_1, R)$. We compare $\text{count}_{\texttt{X}}(c_1, R)$ to `Comp` and `Num`. For example, if `Comp` is `all`, then $\text{count}_{\texttt{X}}(c_1, R)$ must equal the number of non-empty constraint in R for c_1 to be satisfied, and if `Comp` is $==$ and `Num` is 2, then $\text{count}_{\texttt{X}}(c_1, R)$

must be equal to 2 for c_1 to be satisfied.

The separation constraint type has the tuple $\langle\{11, 01, 00\}, \texttt{all}, \texttt{None}\rangle$ and the star constraint type can be represented as the tuple $\langle\{01, 11\}, ==, 2\}\rangle$. We exhaustively enumerate all possible legal combinations of $\texttt{X}, \texttt{Comp}$ and $\texttt{Num}$ to get the space of constraint types. The total size of this space of constraint types is 96.

4.2 Evaluating the Space of Constraint Types

Using the enumerated space of constraint types and our orthogonality measure, we calculated the following orthogonality value for each generalized constraint type K_*: $o(K_*, \{\texttt{separation}, \texttt{star})$ where every generalized, separation and star constraint can be of two colors, black and blue, the size of each level is 3×3, and t_{max} is 5. It took roughly two hours to evaluate the orthogonality value for all constraint types in our space of 96 constraint types using a 2017 Macbook Pro (2.8 GHz Quad-Core Intel Core i7 with 16GB RAM). We do not believe that it is that important that it took two hours, because it could have also taken all day, but that is fine since speed of evaluation is not our main concern when we are using EPCG (although obviously it should still finish in a reasonable time).

It is useful to analyze the entirety of our space of constraint types. In Table 4.1, we count the number of different orthogonality numbers we see in our space of 96 constraint types. For example, in the table we can see that almost 60 of the 96 constraint types had an orthogonality of 0. The two highest scoring constraint type in Table 4.1 both have == 1 in their definition. This would suggest that future iterations of our system should focus on creating a space of constraint types that all have to do with the constraint being "alone." It also suggests that our space of constraint types is limited, since most constraint types have zero orthogonality.

Orthogonality	Count
0.0	58
0.0006	2
0.0017	1
0.0026	5
0.0031	1
0.0035	8
0.0043	10
0.0046	4
0.0064	4
0.0078	1
0.0099	1
0.013	1

Table 4.1: The count of constraint types in our space of constraint types with a given orthogonality.

4.3 Most Orthogonal Constraint Type

The optimal new constraint type from our space of constraint types is the following: a new constraint piece in a region must be unique, meaning that no other constraint piece in the same region can share its color or type. The tuple representation of the new constraint type is $\langle\{01, 10, 11\}, ==, 1\rangle$. We call this constraint type the singleton constraint type, and represent it visually as a hexagon. The singleton constraint type is able to express solution paths that are not possible with the star or separation constraint, either alone or together. Note however that this does not mean that these solution paths are entirely new; instead, they come from being more efficient—that is, it requires fewer pieces in a level to get the solution path as a unique solution to a level. Specifically, the singleton and separation constraints with five pieces on the grid can express 8 new solution paths more efficiently. The singleton and star constraints

with five pieces on the grid can express 16 new solutions paths more efficiently. The new solution paths look similar to solution path 56 shown in Figure 4.1: there is one large region with smaller regions of size 1 or 2.

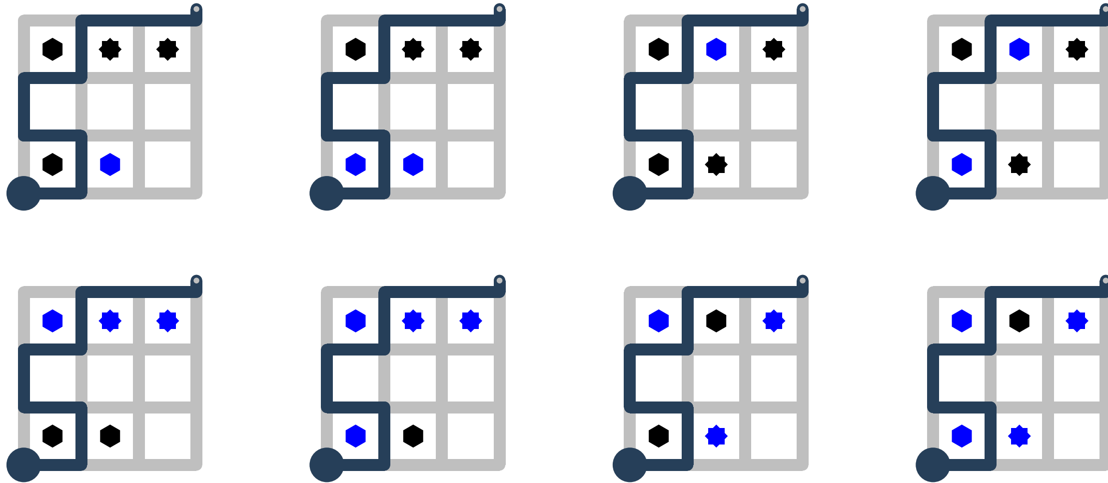

Figure 4.1: All levels with five singleton and star constraints that have a unique solution that is solution path 56.

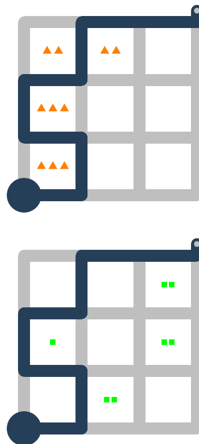

Figure 4.2: Solution path 56 is expressible by the tetris and triangle constraints with only four constraint pieces in either case.

We note that the singleton constraint type combines in a similar way with both the star and separation constraint types, with the new solutions expressible with the separation constraint type being a subset of those expressible with the star constraint

type. This may suggest that in levels with the singleton constraint type, that the other constraint types are not equally contributing to the unique solution path, and that instead the solution path is mostly determined by the singleton constraint types. We can see this in Figure 4.1, where for each level, there is only two star constraints and both of them have the same color. This means that we know they must be in the same region. For the separation and singleton constraint types, we often see the same thing: new solution paths can be expressed with levels where each separation constraint is a single color. However, we later show in our expert evaluation that we can create interesting levels with combinations of object types for hand-designed levels.

In addition, we find that the triangle and tetris constraint types can represent the same unique solution paths as the singleton constraint type can with the separation and star constraint types, but more efficiently—that is, with less constraint pieces in a level (see Figure 4.2).

4.4 Expert Evaluation

In the previous section, we discussed the object type with the highest orthogonality, the single constraint type. We want to understand if this constraint type could be added to *The Witness*, and if the singleton constraint type is interesting. There is no reason to believe that just because the singleton constraint type had the highest orthogonality that it should be interesting. In order to answer these questions, we performed an expert evaluation with Patrick Traynor, a game designer who created the puzzle game *Patrick's Parabox*, which released on March 29, 2022 and as of February 2024 has over 2,000 reviews with 99% being positive. We manually designed a curriculum of levels that introduced the singleton constraint, along with how it combines with the separation and star constraints, and presented it to Traynor to play (see the appendix for the curriculum). A curriculum of levels in our case is a finite chain of levels where upon the completion of a level the player moves onto the

single next level. The curriculum was designed informally based on our intuitions about what makes a good curriculum, and before we sent the puzzles to Traynor, we tested the levels ourselves. In general, we started with what we thought were easy levels and increased the perceived difficulty as the curriculum progressed, with the hardest levels being at the end. We created a website that allowed Traynor to play the levels one by one. There was no way to skip levels, and Traynor was required to finish the current level to get the next level. Traynor was not told how the singleton constraint type works; but he was able to figure this out by himself. Traynor recorded himself playing the curriculum which he sent to us. It took him roughly 20 minutes to complete the curriculum, and he verbalized his thinking while he played. We asked the question: "Do you think the new constraint adds something new to the game?" He also gave a written response.

Traynor enjoyed the puzzles, and found that they would fit well in *The Witness*. Specifically, he stated that "the combination puzzles with hexagon+squares and hexagon+stars are both good, and promising!"

However, he believed that the singleton constraint was too close to the star constraint, since they both have to do with group counting (2 for star, and 1 for singleton), and they both interact with the color of constraints of different types. Traynor states that "if we were actually making *The Witness* here, I think there would be a consideration to maybe cut either the star or the hexagon." We believe the issue is because of our limited space of constraint types, which we derived by generalizing the star and separation constraints. Future work could look at how to support designers in creating large object type spaces that cover many object types.

Limitations of Expert Evaluation

We discuss limitations of our expert evaluation. Our expert evaluation had Patrick Traynor, a game designer, play a hand-crafted curriculum of levels. However, there are some problems with this form of evaluation. The supervisor of this thesis knows

Traynor, which means that his feedback may not be fully truthful. In addition, we only had a single expert evaluate our singleton constraint. It would be beneficial to have more people play our curriculum, especially puzzle game designers or people who play a lot of puzzle games. Also, it is not clear whether or not the issues Traynor had with the singleton constraint could be solved by giving a different and perhaps improved curriculum of levels. Also, it would be beneficial to try to create a curriculum of levels for a constraint type with low, but non-zero, orthogonality so we could compare the experiences of the low orthogonality constraint type and the singleton constraint type.

4.5 Orthogonality Function using Form of Constraint Types

An important question that could be asked is why we do not define orthogonality on the form of the constraint types; that is, in terms of the representation of the constraint types as a tuple $\langle \cdot, \cdot, \cdot \rangle$. We could do this for the space of generalized constraint types we created for *The Witness*. Let $o_{\text{form}}(K_*, \mathcal{K}) = min_{K' \in \mathcal{K}} d(K_*, K')$ where d is defined on the representation of K as a tuple $\langle \texttt{X}, \texttt{Comp}, \texttt{Num} \rangle$ (we calculate the fraction of elements that two generalized constraint type tuples share). For the following examples and results, we assume $\mathcal{K}$ consists of the separation and star constraints.

Defining orthogonality in terms of the form of constraint types is problematic for two reasons. Firstly, there may be multiple way to write down a generalized constraint type as a tuple. For example, the separation constraint can be written as $\langle \{11, 01, 00\}, \texttt{all}, \texttt{None} \rangle$, but it can also be written as $\langle \{10\}, \texttt{None}, \texttt{None} \rangle$. Even though the constraint type tuples are the same in terms of its how they would work in a level, there is non-zero orthogonality. Secondly, there may be constraint types that are different from the existing constraint types in terms of their form, but the constraint type itself is undesirable. For example, the constraint $\langle \{11, 01, 00, 10\}, \texttt{None}, \texttt{None} \rangle$

can never be satisfied, and thus is an undesirable object type, but defining orthogonality in terms of the form of a mechanic would give it a non-zero orthogonality. In Table 4.2 , we show how many constraint types have a specific orthogonality. For example, 2 constraint types had an orthogonality of 0 (the separation and star constraint, defined as $\langle\{11, 01, 00\}, \texttt{all}, \texttt{None}\rangle$ and $\langle\{01, 11\}, ==, 2\rangle$ respectively). From the graph, we can see that using this form measure of orthogonality results in all object types, except the two object types representing the star and separation, having non-zero orthogonality. However, as we previously stated, there should be more object types with an orthogonality of 0. The constraint type $\langle\{11, 01, 00, 10\}, >=, 1\rangle$ has an orthogonality of 1, but it represents a constraint type which is always satisfied.

Orthogonality	0	1/3	2/3	1
Count	2	33	47	14

Table 4.2: The count of constraint types in our space of constraint types with a certain orthogonality. This is for the orthogonality measure based on the tuple representation of a generalized constraint type.

4.6 Understanding Existing Constraint Types

We explore how orthogonal the existing constraint types are in *The Witness* are, in particular, we focus on the separation, star, triangle and tetris constraints. We are going back to our normal notion of orthogonality, and not the notion of orthogonality defined in terms of the form of a constraint type.

We examine what the best constraint is by itself—that is, we want to determine what constraint by itself can express the most unique solution paths. We use the following expressivity (orthogonality when there is only one constraint type) evaluation function for object type K:

$$e(K) = \frac{|\bigcup_{i=1}^{t_{max}} \mathcal{P}(\mathcal{L}_{R\times C}(K, i))|}{\texttt{P}(R, C)}$$

We evaluated the existing constraints on a grid size $R = 3$ and $C = 2$, with $t_{max} = 6$. Separation and star constraints are allowed two colors, while triangle is allowed all parameters, and tetris is allowed to be the first three shapes (1x1 block, 1x2 block, and 2x1 block). We have the following results: e(`separation`) = 0.21 (it can express 8/38 solution paths, e(`star`) = 0, e(`triangle`) = 0.97 (it can express all but one solution path uniquely), and e(`tetris`) = 0.26.

The triangle constraint is very expressive by itself, with all of its unique solution paths coming from levels with only 2 and 3 constraints in a level. This makes it difficult to extend triangle constraints with other constraints (that is, $o(K, \mathcal{K} = \{$`triangle`$\}$ will be low). We see this in *The Witness* itself, where the triangle constraint does not get its own area in the game world, and where its interactions with other constraint types are not exhaustively explored. The star constraint has 0 expressivity by itself, thus it is a good constraint to combine with other constraints (it combines well with the separation constraint to give new unique solution paths). Again, we see this in the game where the star constraint area has levels with the star and separation constraints, along with the star and tetris constraints. The separation and tetris constraints have similar expressivity, but the separation constraint has all of its unique solution paths being expressed with four and five constraints in a level, while for the tetris constraint, it is instead 1, 2 and 3 constraints in a level.

4.7 Understanding Gameplay Representations

Finally, we show a few circumstances where the unique solution path obscures differences in the underlying reasoning done by players. In Figure 4.3, we show two similar levels with the same unique solution path. Level X is deduced by the fact that there should be a line between two adjacent separation constraints of different colors. In Level Y, the unique solution occurs because the star constraints must be paired together (pairing with the separation constraint will leave a star constraint by itself), and that you cannot have all three constraints in the same region (since the

separation constraint is the same color as the star constraints in this level).

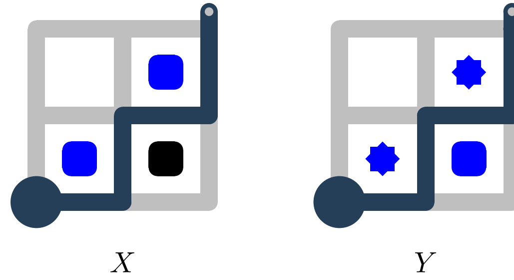

Figure 4.3: Two levels with the same unique solution path. However, the reasoning is a bit different.

We note that for a 2×2 grid, the path sets $\mathcal{P}_{2\times2}(\texttt{separation}, i)$ and $\mathcal{P}_{2\times2}(\texttt{separation}, \texttt{star}, i)$, where the separation and star constraint types can be black and blue, are the same for all $1 \leq i \leq 4$. That is, when we consider both the separation and star constraint types together in levels, we get the same plot as in Figure 3.5. Put differently, the star and separation constraint types are not more expressive together in 2×2 levels. However, as we showed in previous examples, when the grid size increases to 3×3, we are able to a see substantial difference between the path sets when we consider only the separation constraint type and when we consider both separation and star constraint types. That is, the star and separation constraint types are more expressive together in 3×3 levels, while this is not true for 2×2 levels. Thus, while looking at only paths may ignore the reasoning required to solve a level, by looking at larger boards, we are able to see differences between constraint types that we were not able to make on smaller grids.

This brings up a bigger question: what exactly does it mean for two levels to be different. Clearly, comparing sets of solutions does not consider *how* players solve levels. In addition, there can be trivial changes to a level that change the solution set but do not make a *meaningful* change to actual gameplay. For example, we could rotate the level X by 90 degrees and the set of solutions would change, but clearly the level is practically the same.

Currently, we attribute the entirety of a level to the set of solutions, but clearly some actions are made based on only a subset of objects in a level. For example, in level X in Figure 4.3, the solution starts with right and then up. Conditional on having moved right, the player must move up because of the two separation constraints. We note that methods to explain why certain actions were taken and attribute the action to a subset of the objects in the level may help researchers understand levels in a more nuanced manner [37].

An important property that future work can look at is trying to define a way to compare levels that is cognitively grounded. Farrell has examined this for stories using a cognitive model, but in general there is a need for methods that allow for detailed comparisons between puzzle levels [38]. Another important thing to look at is how integral an object is to a level. In levels with only two object types, we want both object types to contribute equally to creating the gameplay experience, and not one type of object largely contributing to the gameplay experience while the other object types do not do much to contribute to the gameplay experience.

Chapter 5

Related Work

In this chapter, we cover related work. In particular, we focus on automated game design and game evaluation.

5.1 Automated Game Design

Automated game design (AGD) refers to the autonomous generation of games, with most systems focusing on rule (mechanic) generation [39]. Many AGD systems rely on search using a fitness function to quantify the quality of a game. Browne's Ludi system uses evolutionary search to generate two-player combinatorial games [40]. For single–player games, Nielsen et al. generate games for the Video Game Description Language (VGDL) [41], but evaluate a game using a randomly generated non-crashing level [42]. Lim and Harrel generate games in PuzzleScript, a puzzle game engine, but evaluate the game using pre-authored levels [43]. The Mechanic Miner system generates mechanics for a puzzle platform game and levels that utilize the generated mechanic, but the process is sequential: a mechanic is generated using a sample level, and then once a mechanic is found, levels that use the mechanic are generated [44]. Our work is related to the previous work in that it is also an example of AGD, but the previous work apply it to other types of games (two-player games) or when they apply it to single-player games they do not consider multiple levels in their evaluation of a single mechanic or object type. However, note that work by Guzdial and Riedl does

consider multiple levels in their evaluation, but not using a notion of orthogonality [45].

All the previously mentioned AGD systems rely on a fitness function to guide the search process, but not all AGD systems work in this manner. Some systems rely on designers sculpting a space of games using constraints. Smith and Mateas generate rules in their game Variations Forever using answer-set programming (ASP) to constrain a space of mini-games [46]. Zook and Riedl created a system designers can use to prototype game mechanics from a variety of genres that meet design requirements and playability requirements [47]. Their system could be used to evaluate a game mechanic on a set of pre-authored levels, but their system itself does not integrate level generation as our orthogonality measure does.

Some systems do not generate games or game mechanics, but instead focus on refining a game or generating variations on a game. Isaksen et al. generate variants of the game *Flappy Bird* that play differently as possible [48]. Although we generate a new mechanic, it is really a variation on a theme of mechanics.

Finally, we note that previous work has looked at generating game objects. Butler et al. generate Mega-man-like bosses that meet design requirements [49], while Sorochan and Guzdial use search to generate game units in a real-time strategy game [50]. Unlike the previous work, we generate object types and their mechanics for a combinatorial game.

Cicero is a system built to help designers create games in VGDL [51], with the system providing a variety of tools, including Pitako, a recommendation system for game mechanics [52].

5.2 Game Evaluation

In the PCG community, there has been a need to quantify the quality of games, for example, for use in AGD systems. Many of these evaluations look at how deep a game is. Lantz et al. introduce their strategy ladder model, designed in particular

for two-player games [53]. In their model, they look at how the performance of optimal strategies change as computational resources increase—that is, they look at the shape of the resulting graph as a way to determine a game's depth. A game with high strategic depth should be easy to learn, but hard to master.

For single-player games, many level evaluations function have been explored—including difficulty evaluations [54] and game-specific evaluations [55]. In addition, there exist measures of strategic depth for single-player games [56, 57], but these measures equate the depth of a game with the depth of a single level from a game. There exist attempts to quantify the quality of a space of levels for a single-player game. Expressive range analysis (ERA) can be used to understand a space of content [58], in particular, for levels. In ERA, metrics must be defined to evaluate the quality of each level, which can be visualized using 2D histograms, where the axes are defined by the different values of the metrics. There has been work on analyzing phase transitions [59], and deriving maximal values for a space of content. For example, for 9×9 Sudoku it is known that it takes a minimum of 17 clues in a level to have a unique solution [60]. Orthogonality provides a different way to look at a space of content that could complement ERA and these methods. Note that our notion of orthogonality that incorporates novelty, diversity and quality is similar to how researchers evaluate PCG systems, especially ones that train on data (see plagiarism, self-similarity, and playability) [61].

Beyond trying to understand single-player combinatorial game domains, there has been work on trying to understand emergent narratives [62], cellular automata [63], and spaces of two-player games [64]. We share similar concerns as these researchers in that we are trying to understand and analyze complex systems where it is not obvious to designers what things are possible in that space.

Finally, we note that the ideas of orthogonality and the aesthetics of Jonathan Blow and Marc Ten Bosch have similarities to ideas such as open-endedness [65] and emergence [66]. In particular, like the other ideas, orthogonality has a focus on

the ability for a system (a combinatorial game domain) to produce many novel and interesting things (in our case, levels).

Chapter 6

Future Work

Our system was made for the *The Witness*, and the orthogonality evaluation function is geared towards it. However, future work could look at generalizing our work to other games. In particular, future work could look at Japanese logic puzzles, for example, Sudoku. We could look at generating variants of Sudoku. This has been previously explored by Browne, who generated Ludoku, a simpler variant of Sudoku [67]. Popular variants of Sudoku often allow initial board configurations that previously did not have a unique solution to now have a unique solution. For example, a video solving The Miracle Sudoku by Mitchell Lee has over 3 millions views where the initial board configuration only has two numbers on it.[1] Our current approach looks only at which object types are in a level, the unique solution path, and the number of non-empty constraints. For Sudoku, we would need to look at levels in more detail—that is, we would need to look at the location of object types and the initial board configuration (the level itself). We could also extend Sudoku by coming up with new variants that place new entities in the initial level configuration (for example, with the thermometers constraint).

We could also extend the work to games like *Sokoban* (and in general to games written in PuzzleScript). However, unlike in Japanese logic puzzles and *The Witness*, most *Sokoban* levels do not have a unique solution. Looking at the movement of the boxes only and not the movement of the person gives fewer solutions, but most levels

[1] https://www.youtube.com/watch?v=yKf9aUIxdb4&t=400s

still permit multiple solutions. For *Sokoban*, we need a way to determine if there are multiple, qualitatively different solutions to a single level. That is, we want to characterize a level by the main idea it is trying to get across to players. Approaches could look at defining a distance function on solutions [68], clustering player solutions [69], or requiring that certain concepts be thought and that there exist no shortcuts [70]. Online versions of *Sokoban* extend the game with new game objects, such as ice blocks, and extensions of our work could look at generating new types of entities.[2] *Sokoban* is an important game to extend our work on orthogonality, because it is popular and there are many games similar to it (games in which you are pushing blocks or other items).

Finally, we note that our notion of orthogonality is based on game objects that can be placed in levels. However, many mechanics are actions that the player can take (verbs). We could look at whether or not action mechanics when combined together interfere with each other (for example, in the game *VVVVVV* there is a gravity inversion mechanic, but what would happen to levels if normal jumping was also allowed).

In this thesis, we have largely discussed puzzle games (while also sometimes discussing other types of games like platformers). We believe that we could extend the notion of orthogonality introduced in this thesis by looking towards other types of games. For example, in genres like role-playing games (RPGs) and immersive sims, mechanics allow the player to be complete a level or problem in multiple different ways; the addition of new mechanics opens up new avenues of play. In an RPG, stealth mechanics encourage the player to take new paths through a level. In this thesis, we focused entirely on solving the win-condition of a level. But games often have other objectives that players can choose to optimize for. For example, in *Super Mario Bros.*, collecting coins is not necessary to completing a level, but it does affect

[2]See https://www.sokobanonline.com/play/tutorials for an online version of *Sokoban* that contains new object types not seen in classic *Sokoban*.

how the player experiences the game. Further development of orthogonality could incorporate goals that do not need to be met to solve a level.

We note that orthogonality has to do fundamentally with adding content to a game so that new gameplay experiences arise. This process of adding novel content that changes up the gameplay experience of a game is central to live-service games that continually get updated in order to be fresh. It would be interesting to look at orthogonality in terms of other things, like characters in a hero shooter or cards in a digital collectible card game.

Finally, future work could look at building mixed-initiatives systems in which a computational system and a human designer work together to create orthogonal mechanics or explore the orthogonality of some new mechanic.

Chapter 7

Conclusion

In this thesis, we explored the concept of orthogonality. We showed how it could be applied to combinatorial puzzle games, in particular *The Witness*. Using *The Witness*, we showed how a measure of orthogonality can be used to analyze a game and also generate game content. We detailed limitations of our approach and possible improvements and future work.

Bibliography

[1] J. Schell, *The Art of Game Design: A book of lenses*. CRC press, 2008.

[2] J. Blow and M. Ten Bosch, *Designing to reveal the nature of the universe*, https://www.youtube.com/watch?v=OGSeLSmOALU&t=1585s, IndieCade, 2011.

[3] R. Koster, *Theory of fun for game design*. " O'Reilly Media, Inc.", 2013.

[4] A. B. Jaffe, "Understanding game balance with quantitative methods," Ph.D. dissertation, 2013.

[5] E. Y. C. Chen, A. White, and N. R. Sturtevant, "Entropy as a measure of puzzle difficulty," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 19, 2023, pp. 34–42.

[6] H. Tulleken, *How are puzzle games designed? (introduction)*, http://devmag.org.za/2011/04/16/how-are-puzzle-games-designed-introduction/.

[7] A. Liapis, G. N. Yannakakis, M. J. Nelson, M. Preuss, and R. Bidarra, "Orchestrating game generation," *IEEE Transactions on Games*, vol. 11, no. 1, pp. 48–68, 2018.

[8] M. S. O. Almeida and F. S. C. da Silva, "A systematic review of game design methods and tools," in *Entertainment Computing–ICEC 2013: 12th International Conference, ICEC 2013, São Paulo, Brazil, October 16-18, 2013. Proceedings 12*, Springer, 2013, pp. 17–29.

[9] R. Hunicke, M. LeBlanc, R. Zubek, *et al.*, "Mda: A formal approach to game design and game research," in *Proceedings of the AAAI Workshop on Challenges in Game AI*, San Jose, CA, vol. 4, 2004, p. 1722.

[10] E. Aarseth, "Playing research: Methodological approaches to game analysis," in *Proceedings of the digital arts and culture conference*, RMIT University Melbourne, VIC, 2003, pp. 28–29.

[11] R. Zubek, *Elements of game design*. MIT Press, 2020.

[12] M. S. Debus, *Unifying game ontology: a faceted classification of game elements*. IT-Universitetet i København, 2019.

[13] J. P. Zagal, C. Fernández-Vara, and M. Mateas, "Rounds, levels, and waves: The early evolution of gameplay segmentation," *Games and Culture*, vol. 3, no. 2, pp. 175–198, 2008.

[14] P. Lo, D. Thue, and E. Carstensdottir, "What is a game mechanic?" In *Entertainment Computing–ICEC 2021: 20th IFIP TC 14 International Conference, ICEC 2021, Coimbra, Portugal, November 2–5, 2021, Proceedings 20*, Springer, 2021, pp. 336–347.

[15] M. S. Debus, J. P. Zagal, and R. E. Cardona-Rivera, "A typology of imperative game goals," *Game Studies*, vol. 20, no. 3, 2020.

[16] M. C. Green, A. Khalifa, R. Canaan, P. Bontrager, and J. Togelius, "Game mechanic alignment theory," in *Proceedings of the 16th International Conference on the Foundations of Digital Games*, 2021, pp. 1–11.

[17] S. Harvey, *Systemic level design*, https://slideplayer.com/slide/1461389/.

[18] L. V. Hufkens and C. Browne, "A functional taxonomy of logic puzzles," in *2019 IEEE Conference on Games (CoG)*, IEEE, 2019, pp. 1–4.

[19] S. Lavelle, *Puzzlescript*, https://github.com/increpare/PuzzleScript, 2015.

[20] T. Schaul, "A video game description language for model-based or interactive learning," in *Proceedings of the IEEE Conference on Computational Intelligence in Games*, Niagara Falls: IEEE Press, 2013.

[21] N. Nethercote, P. J. Stuckey, R. Becket, S. Brand, G. J. Duck, and G. Tack, "Minizinc: Towards a standard cp modelling language," in *International Conference on Principles and Practice of Constraint Programming*, Springer, 2007, pp. 529–543.

[22] D. M. McDermott, "The 1998 ai planning systems competition," *AI magazine*, vol. 21, no. 2, pp. 35–35, 2000.

[23] S. Lundgren, K. Bergström, and S. Björk, "Exploring aesthetic ideals of gameplay.," in *DiGRA Conference*, vol. 10, 2009.

[24] T. Grip, *4-layers, a narrative design approach*, 2014. [Online]. Available: https://www.gamedeveloper.com/design/4-layers-a-narrative-design-approach.

[25] E. Grant, *Elyot grant - 30 puzzle design lessons, extended director's cut (part 1 of 3)*, 2021. [Online]. Available: https://www.youtube.com/watch?v=oCHciE9CYfA.

[26] A. Nealen, A. Saltsman, and E. Boxerman, "Towards minimalist game design," in *Proceedings of the 6th international conference on foundations of digital games*, 2011, pp. 38–45.

[27] A. Anthropy and N. Clark, *A game design vocabulary: Exploring the foundational principles behind good game design*. Pearson Education, 2014.

[28] Z. Wang, J. Liu, and G. N. Yannakakis, "The fun facets of mario: Multifaceted experience-driven pcg via reinforcement learning," in *Proceedings of the 17th International Conference on the Foundations of Digital Games*, 2022, pp. 1–8.

[29] H. Smith, "Orthogonal unit design," GDC, 2003. [Online]. Available: https://www.gdcvault.com/play/1022697/Orthogonal-Unit.

[30] G. Fan, "Rules of the game: Five more techniques from quite inventive designers," GDC, 2016. [Online]. Available: https://www.gdcvault.com/play/1023210/Rules-of-the-Game-Five.

[31] N. Shaker, J. Togelius, and M. J. Nelson, "Procedural content generation in games," 2016.

[32] G. Smith, "Understanding procedural content generation: A design-centric analysis of the role of pcg in games," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2014, pp. 917–926.

[33] R. Khaled, M. J. Nelson, and P. Barr, "Design metaphors for procedural content generation in games," in *Proceedings of the SIGCHI conference on human factors in computing systems*, 2013, pp. 1509–1518.

[34] N. Sturtevant and M. Ota, "Exhaustive and semi-exhaustive procedural content generation," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 14, 2018, pp. 109–115.

[35] D. Gravina, A. Khalifa, A. Liapis, J. Togelius, and G. N. Yannakakis, "Procedural content generation through quality diversity," in *2019 IEEE Conference on Games (CoG)*, IEEE, 2019, pp. 1–8.

[36] N. R. Sturtevant, "Exploring epcg in the witness.," in *KEG@ AAAI*, 2019, pp. 58–63.

[37] J. Espasa *et al.*, "Using small muses to explain how to solve pen and paper puzzles," *arXiv preprint arXiv:2104.15040*, 2021.

[38] R. Farrell, M. Fisher, and S. G. Ware, "Salience vectors for measuring distance between stories," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 18, 2022, pp. 95–104.

[39] M. Cook, "Formalizing non-formalism: Breaking the rules of automated game design," 2015.

[40] C. Browne and F. Maire, "Evolutionary game design," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 2, no. 1, pp. 1–16, 2010.

[41] T. Schaul, "A video game description language for model-based or interactive learning," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, IEEE, 2013, pp. 1–8.

[42] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "Towards generating arcade game rules with vgdl," in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE, 2015, pp. 185–192.

[43] C.-U. Lim and D. F. Harrell, "An approach to general videogame evaluation and automatic generation using a description language," in *2014 IEEE Conference on Computational Intelligence and Games*, IEEE, 2014, pp. 1–8.

[44] M. Cook, S. Colton, A. Raad, and J. Gow, "Mechanic miner: Reflection-driven game mechanic discovery and level design," in *Applications of Evolutionary Computation: 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3-5, 2013. Proceedings 16*, Springer, 2013, pp. 284–293.

[45] M. Guzdial and M. O. Riedl, "Conceptual game expansion," *IEEE Transactions on Games*, vol. 14, no. 1, pp. 93–106, 2021.

[46] A. M. Smith and M. Mateas, "Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, IEEE, 2010, pp. 273–280.

[47] A. Zook and M. Riedl, "Automatic game design via mechanic generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, 2014.

[48] A. Isaksen, D. Gopstein, J. Togelius, and A. Nealen, "Discovering unique game variants," in *Computational Creativity and Games Workshop at the 2015 International Conference on Computational Creativity*, 2015.

[49] E. Butler, K. Siu, and A. Zook, "Program synthesis as a generative method," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, 2017, pp. 1–10.

[50] K. Sorochan and M. Guzdial, "Generating real-time strategy game units using search-based procedural content generation and monte carlo tree search," *arXiv preprint arXiv:2212.03387*, 2022.

[51] T. L. de Araujo Machado, "Cicero-an ai-assisted game design system," Ph.D. dissertation, New York University Tandon School of Engineering, 2019.

[52] T. Machado, D. Gopstein, A. Nealen, and J. Togelius, "Pitako-recommending game design elements in cicero," in *2019 IEEE Conference on Games (CoG)*, IEEE, 2019, pp. 1–8.

[53] F. Lantz, A. Isaksen, A. Jaffe, A. Nealen, and J. Togelius, "Depth in strategic games," in *31st AAAI Conference on Artificial Intelligence, AAAI 2017*, AI Access Foundation, 2017, pp. 967–974.

[54] P. Jarušek and R. Pelánek, "Difficulty rating of sokoban puzzle," in *STAIRS 2010*, IOS Press, 2010, pp. 140–150.

[55] A. Summerville, J. R. Mariño, S. Snodgrass, S. Ontañón, and L. H. Lelis, "Understanding mario: An evaluation of design metrics for platformers," in *Proceedings of the 12th international conference on the foundations of digital games*, 2017, pp. 1–10.

[56] T. S. Nielsen, G. A. Barros, J. Togelius, and M. J. Nelson, "General video game evaluation using relative algorithm performance profiles," in *European Conference on the Applications of Evolutionary Computation*, Springer, 2015, pp. 369–380.

[57] D. Apeldoorn and V. Volz, "Measuring strategic depth in games using hierarchical knowledge bases," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, IEEE, 2017, pp. 9–16.

[58] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 workshop on procedural content generation in games*, 2010, pp. 1–7.

[59] I. P. Gent and T. Walsh, "The tsp phase transition," *Artificial Intelligence*, vol. 88, no. 1-2, pp. 349–358, 1996.

[60] G. McGuire, B. Tugemann, and G. Civario, "There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem via hitting set enumeration," *Experimental Mathematics*, vol. 23, no. 2, pp. 190–217, 2014.

[61] E. Halina and M. Guzdial, "Tree-based reconstructive partitioning: A novel low-data level generation approach," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 19, 2023, pp. 244–254.

[62] Q. Kybartas, "Quantitative analysis of emergent narratives," in *The Authoring Problem: Challenges in Supporting Authoring for Interactive Digital Narratives*, Springer, 2023, pp. 321–334.

[63] M. Vispoel, A. J. Daly, and J. M. Baetens, "Progress, gaps and obstacles in the classification of cellular automata," *Physica D: Nonlinear Phenomena*, vol. 432, p. 133 074, 2022.

[64] S. Omidshafiei *et al.*, "Navigating the landscape of multiplayer games," *Nature communications*, vol. 11, no. 1, p. 5603, 2020.

[65] A. Song, "A little taxonomy of open-endedness," in *ICLR Workshop on Agent Learning in Open-Endedness*, 2022.

[66] N. Guttenberg and L. Soros, "Designing emergence in games," in *ALIFE 2023: Ghost in the Machine: Proceedings of the 2023 Artificial Life Conference*, MIT Press, 2023.

[67] C. Browne, "Ludoku: A game design experiment," *Game & Puzzle Design*, vol. 3, no. 2, pp. 35–46, 2017.

[68] J. Osborn, B. Samuel, J. McCoy, and M. Mateas, "Evaluating play trace (dis) similarity metrics," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 10, 2014, pp. 139–145.

[69] J. Campbell, J. Tremblay, and C. Verbrugge, "Clustering player paths.," in *FDG*, 2015.

[70] E. Butler, E. Andersen, A. M. Smith, S. Gulwani, and Z. Popović, "Automatic game progression design through analysis of solution features," in *Proceedings of the 33rd annual ACM conference on human factors in computing systems*, 2015, pp. 2407–2416.

Appendix A: Curriculum

The curriculum of levels presented to Patrick Traynor. The curriculum is ordered from top left to bottom right. Traynor was not told the rules of the singleton constraint, and had to learn it from the curriculum. Note that not all levels in the curriculum have a unique solution.