Analyzing the impact of knowledge and search in Monte Carlo Tree Search in Go

by

Farhad Haqiqat

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

Master of Science

Department of Computing Science

University of Alberta

Abstract

Domain-specific knowledge plays a significant role in the success of many Monte Carlo Tree Search (MCTS) programs. The details of how knowledge affects MCTS are still not well understood. In this thesis, we focus on identifying the effects of different types of knowledge on the behaviour of the Monte Carlo Tree Search algorithm, using the game of Go as a case study. We measure the performance of each type of knowledge, and of deeper search by using two main metrics: The move prediction rate on games played by professional players, and the playing strength of an implementation in our program Fuego. We compare the result of these two evaluation methods in detail, in order to understand how effective they are in fully understanding a program's behaviour. A feature-based approach refines our analysis tools, and addresses some of the shortcomings of these two evaluation methods. This approach allows us to interpret different components of knowledge and deeper search in different phases of a game, and helps us to obtain a deeper understanding of the role of knowledge and its relation with search in the MCTS algorithm.

Acknowledgements

I would like to thank my supervisor, Prof. Martin Müller for all his guidance, patience and positive attitude throughout the journey, and for his constructive comments on my thesis.

I would like to thank my friends and family for all their support when I needed it, and giving me hope when I had none.

Finally, I would like to give my biggest gratitude to my wife Mansoureh Modarres for her encouragement throughout years of my study, and for her unfailing support.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Humans make many choices everyday which vary from trivial decisions, such as what to eat for lunch, to more important decisions, such as how to invest their money. These types of choices can be viewed as One-Shot decisions [28]. However there are many situations where we have to look at sequences of future choices in order to make a decision now. It has been shown that humans tend to make suboptimal decisions when faced with multiple choice repetitive tasks [15] they have a tendency to under experiment the optimal choice [38]. This signifies the need for decision-making algorithms to help us in making better decisions. Researchers developing such algorithms need an environment that is easy to setup and replicate, and has a sequential decision making process.

Board games provide such an environment, which is easy to implement by computers. These games have simple rules, and results obtained with algorithms can be compared to human performance as a measure of strength. Through many years of research many of these games were solved, such as Checkers [43], Hexapawn [8] and Quarto [30]; however, finding optimal solutions for other games such as Go and chess proved to be intractable using the current methods based on search algorithms. Therefore, researchers have put their focus on beating top human players as a goal to advance their algorithms' performance.

Chess programs have exceeded human level of play for the first time in 1997 [19] using the alpha-beta pruning algorithm and special hardware. Alpha-beta's success in chess was helped greatly by the development of strong evaluation functions, which examine a position and return its evaluation. Building such an evaluation function had failed in the game of Go until AlphaGo's value network [45]. To address the lack of proper evaluation functions, Monte Carlo Tree Search (MCTS) methods were developed [18].

The MCTS based program Fuego was the first program able to beat a top human professional player in Go on a 9x9 size board in 2008 [23]. Fuego achieved this level of play by using a MCTS algorithm enhanced by knowledge of features and patterns. Despite the successes that programs had on the 9x9 size board, the full 19x19 size board remained out of reach until recently, when AlphaGo [45, 46] far exceeded the human level of play. AlphaGo uses a variant of MCTS with a very strong knowledge obtained through Neural Networks (NN) in order to navigate the search in MCTS.

1.1 Research Topics

- Examine the relation between knowledge and search in Go programs and how these two impact each other.
- Examine current evaluation approaches used in Go programs, which are: move prediction and playing against another program. Understand the differences between each of these tests and how they relate to each other.
- Evaluate the impact of knowledge on the performance of a Go program.
- How does longer and deeper search improve the strength of a MCTS program, in the presence of knowledge?
- Can this increased strength be explained in terms of simple feature knowledge?

1.2 Contributions of this Thesis

In this thesis, we have done an in-depth analysis of the role of knowledge in the Monte Carlo Tree Search (MCTS) program Fuego. We studied how each component of knowledge and deeper search impacts the performance of a player in the move prediction task. We showed that the move prediction rate is correlated with the stage of the game for each of our studied players. We incorporated features as a tool to interpret a players moves and find differentiating factors that explain differences between our studied players. We studied the impact of feature knowledge on the number of simulations a move receives in MCTS.

In Chapter 2 we review the literature, and explain the terms that we will use throughout this text. In Chapter 3 we explain the tools and methods used to carry out the experiments, such as Fuego [23], which is an open source MCTS Go engine. We also reformulate the research topics of Section 1.1 in terms of this environment. In Chapter 4 we report the results obtained through our experiments, and analyze those results in-depth to answer our research questions.

Chapter 2

Literature review

In this chapter we briefly describe all the methods and terminology needed in order to explain our experiments.

2.1 Knowledge

Professional players in games such as Go or chess use their knowledge to play. Although their knowledge helps them to play at a very high level, they do not express their knowledge in a way that can be implemented easily in a program. AI researchers are interested in obtaining game-specific knowledge to improve their algorithms. One way to achieve this is by studying games played by professional players. The main approaches for obtaining knowledge in the context of games are: features, patterns and neural networks. Part of the process of obtaining knowledge is its evaluation. We first describe methods for evaluation of knowledge, and then briefly describe each method for obtaining knowledge.

Before we describe these methods we need to define knowledge. We define knowledge in the scope of games. Knowledge in this thesis is information gained by training methods that helps a program to act in an informed manner, and improves the performance of a player when applied.

2.1.1 Evaluation of Knowledge

We test the obtained knowledge in order to evaluate its strength. There are two popular approaches for testing: First, by evaluating the move prediction rate on games played by professional players, and second, by using knowledge inside a game engine and evaluating the change in strength of this engine.

Move Prediction

Move prediction is the act of predicting the next move in a game that was played before. To do a move prediction we select a position from a game and feed that position to the game-playing engine. Then we compare the response received with the next move played in the game. See Section 3.3 for a detailed description of game data used in this research.

Move Prediction Data

In any machine learning process involving data to acquire trustworthy results, available data should be divided into at least two set: *training* and *test* sets. There is no specific rule on how to choose the size of training and test set; nevertheless, it is common to have 80% of the data as training set and 20% as test set. It is important that the program that is being trained to learn the knowledge will not see the test data during the training phase.

Playing Strength

Another method for testing a certain type of knowledge is comparing playing strength with and without the use of that knowledge. In this scenario we use knowledge either as a standalone player or integrate it into an available program, and play a number of matches against other programs or another version of itself. If we know the level of strength of the opponent, then we can estimate the strength of our program from the obtained results by using the Elo rating formula [3]. If we have integrated knowledge in an existing program, then we can estimate the quality of the knowledge by measuring the increase in playing strength resulting from the added knowledge.

2.1.2 Features

Features can reveal aspects of a move in a game, which help to better understand and heuristically evaluate that move. We will define features informally using a small example for moves in the position shown in Figure 2.1, and show how those features are extracted.

First we define a list of binary features for our moves:

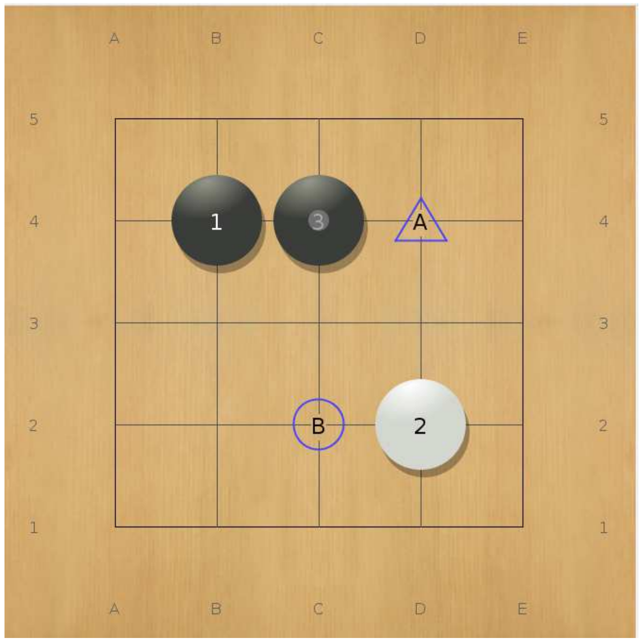


Figure 2.1: Go position.

- Manhattan Distance to previous move of our own is 1
- Manhattan Distance to previous move of our own is 2
- Manhattan Distance to previous move of our opponent is 1
- Manhattan Distance to previous move of our opponent is 2

For example, consider the two possible moves A and B for White in Figure 2.1. For move A, the first feature has a value of 0 (false), since this move does not have the distance of 1 to the previous White move. Following this process we obtain values of: 0,1,1,0 respectively for each feature for move A. Each move is represented by a vector with its corresponding values for above features. Each of the features also has a corresponding weight. Weights are represented by a vector of the same length. The evaluation of a move is a number calculated by the inner product of feature and weight vector. In order to achieve a good evaluation, defined features should be discriminative. After defining features we need to tune their weights to have a more accurate evaluation. This can be done using machine learning techniques. In the following example, we manually assigned values to the weight vector W. Move B in Figure 2.1 has a higher evaluation than move A, therefore, it is considered to be a better move by this heuristic evaluation function.

$$\begin{aligned}
&A \leftarrow < 0, 1, 1, 0 > \\
&B \leftarrow < 1, 0, 0, 1 > \\
&W \leftarrow < 0.40, 0.25, 0.15, 0.20 > \\
&Eval(A) = A.W = 0 * 0.40 + 1 * 0.25 + 1 * 0.15 + 0 * 0.20 = 0.4 \\
&Eval(B) = B.W = 1 * 0.40 + 0 * 0.25 + 0 * 0.15 + 1 * 0.20 = 0.6
\end{aligned} \tag{2.1}$$

There are many methods for learning the weights of features [44, 47, 49, 39]. We briefly describe the methods that are used in Fuego.

The Bradley-Terry Model

The Bradley-Terry (BT) model [32] predicts the outcome of a competition between individuals, when each individual is represented by a strength parameter w. In a BT model the probability of player i beating j is:

$$P(\text{i beats j}) = \frac{w_i}{w_i + w_j} \tag{2.2}$$

The BT model can be generalized to account for teams of individuals [22]. The strength of a team is defined as the product of strengths w_i of team members. Equation (2.3) shows how the model is used to predict the winning probability of team A, given manually assigned strength parameters $w_1 \ldots w_4$. If we have a feature vector for each move, then the move can be represented as a team of features in the generalized Bradley-Terry model, with the strength parameters equal to the feature weights. We can predict the quality of each move in a position by using this model.

$$\begin{aligned}
&TeamA : w_1, w_3, w_4 \\
&TeamB : w_2, w_4 \\
&TeamC : w_1, w_2, w_4 \\
&w_1 = 0.40, w_2 = 0.25, w_3 = 0.15, w_4 = 0.20 \\
&P(\text{Team A wins}) = \frac{w_1 * w_3 * w_4}{w_1 * w_3 * w_4 + w_2 * w_4 + w_1 * w_2 * w_4} \\
&P(\text{Team A wins}) \approx 0.15
\end{aligned} \tag{2.3}$$

In order to use a Bradley-Terry model we need to optimize the parameters w_i of the model. Coulom [22] has proposed using a Minorization-Maximization

(MM) method for this. In MM, the objective function is maximizing the probability of the next played move in the game, given the current position. MM uses an iterative process to optimize the objective function over a (large) test set. A simple surrogate function drives the objective function uphill. For more info on MM, please refer to [32].

Latent Factor Ranking

Another method for estimating the ranking of moves is Latent Factor Ranking (LFR) [37]. LFR ranks the next possible moves in the game from best to worst. LFR uses a Factorization Machine (FM) [40]. In Factorization Machines, feature weights and interaction between features are given by:

$$
\begin{aligned}
y(X) = w_0 + \sum_{i=1}^{m} w_i x_i + \sum_{i=1}^{m} \sum_{j=i+1}^{m} < v_i^T v_j > x_i x_j \\
< v_i^T, v_j > := \sum_{f=1}^{k} v_{i,f} . v_{j,f} \\
X = (x_1, \ldots x_n) \\
W = (w_1, \ldots w_n) \\
X \in \mathbb{R} \text{ is a feature vector (i.e. move features).} \\
W \in \mathbb{R}^n \text{ is a weight vector.} \\
V \in \mathbb{R}^{n \times k} \text{ is a pairwise interaction matrix.} \\
\text{Row } v_i \text{ within } V \text{ describes the } i\text{-th variable with } k \text{ factors.} \\
y \text{ is evaluation of a move, which determines the ranking.}
\end{aligned}
\tag{2.4}
$$

The move with highest evaluation in LFR will have the highest rank. LFR is an extension of the BT model. In BT, only individual weights of features are accounted for, while LFR in addition to features weights accounts for pairwise interaction between features. The extra information provided in this model leads to a 5% improvement in move prediction rate on Go games played by professionals [37]. Another difference between the LFR and BT models is their output LFR only produces a ranking, while BT produces a probability distribution over all legal moves.

The Factorization Bradley-Terry Model

The Factorization Bradley-Terry (FBT) model [53] combines the strong points of the LFR and BT methods. It accounts for pairwise interaction between features, and also produces a probability of each move being selected, enabling the model to be more easily applied as an evaluation function in Go programs.

$$
\begin{aligned}
y(X) &= \sum_{f \in X} w_f + \frac{1}{2} \sum_{f \in X} \sum_{g \in X, f \neq g} < v_f, v_g > \\
P(X^i wins) &= \frac{exp(y(X^i))}{\sum_{j=1}^{N} exp(y(X^j))} \\
& X^a \text{ is feature vector of move } a
\end{aligned}
\tag{2.5}
$$

FBT computes the strength of each feature group, and then uses a softmax function to calculate the winning probability of a group.

2.1.3 Patterns

If we break down a game board into smaller pieces, we find many local shapes that occur repeatedly. These shapes are called patterns. Studies have shown that a professional player knows and uses an extensive number of patterns in their play [33]. Inspired by the way professionals perceive the board, researchers incorporated patterns into their Go programs. Patterns can be seen as a special case of features and many programs use them in their feature set, selecting patterns that are used frequently in professional players games [22, 23, 37]. Figure 2.2 shows an example of 3×3 size patterns which are invariant to rotation. The square mark in the example is where the next move is going to be played. It can be of either colour. Positions marked by crosses can have any stone colour or be empty. Figure 2.2d only matches the pattern if Black moves to the square marked position. Patterns can have different shapes and sizes, and Figure 2.3 is an example of diamond shape patterns. Many Go programs have used this type of patterns [27, 23, 37].

2.1.4 Neural Networks

A neural network (NN) is combination of neurons organized in layers, and it has three types of layers, input, hidden and output. The input layer is where input data is fed to the network, the output layer produces the results, and the hidden layers organize neurons in one or more layers. Figure 2.4a shows a neural network architecture, and Figure 2.4b shows the computations involved in a single neuron of a neural network. A neuron receives an input vector and computes the inner product of the input with its weight vector. The results are then sent through a non-linear activation function, which determines the

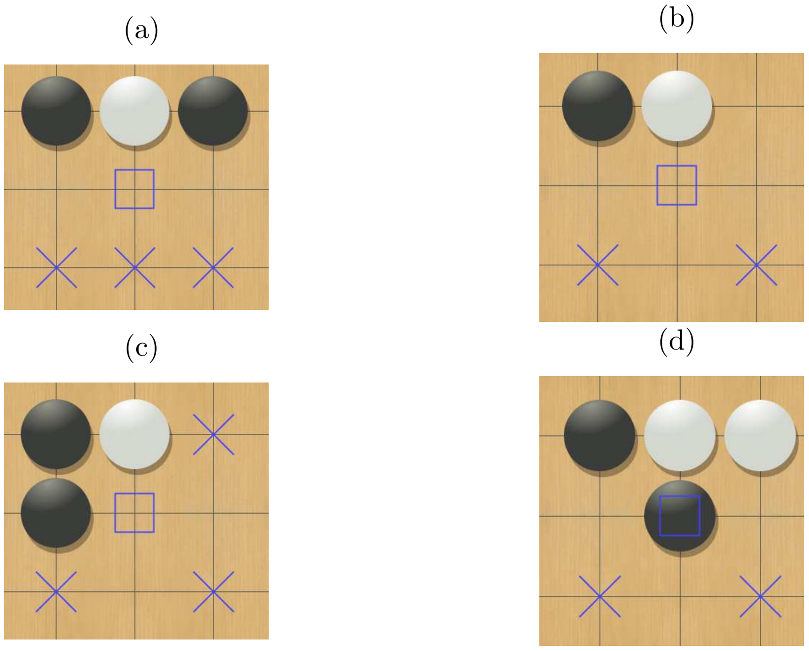


Figure 2.2: 3×3 Patterns for Hane [27].

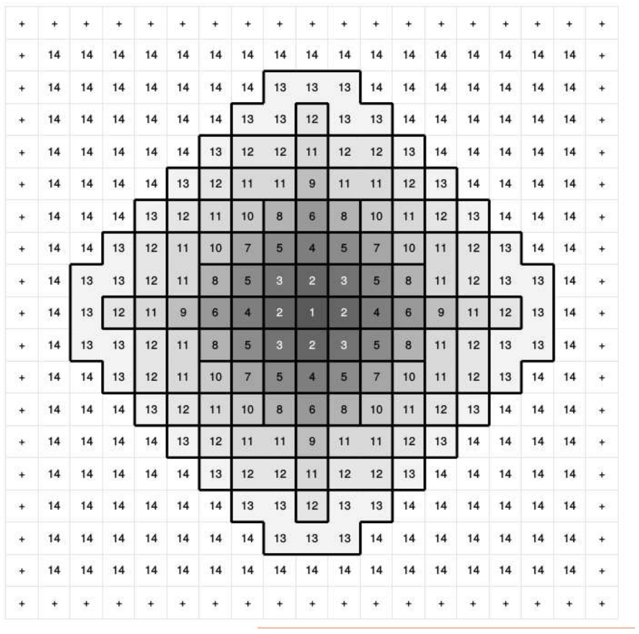
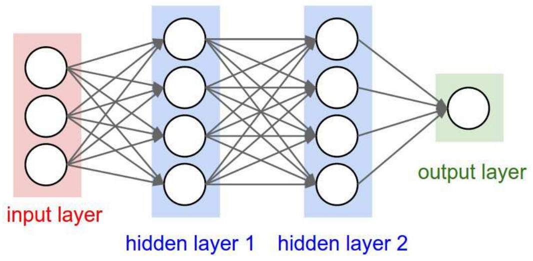
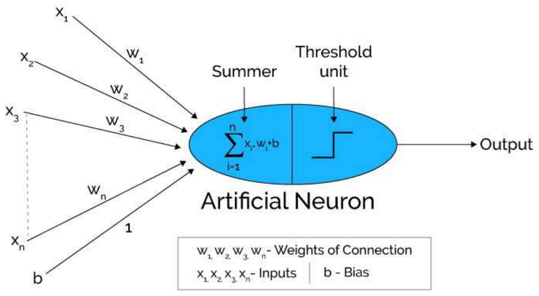


Figure 2.3: Diamond Shape Patterns [49].



(a) Schematic of Neural Network [10]


x_1
w_1
x_2
w_2
x_3
w_3
w_n
x_n
b
1
Summer
Threshold unit
$\sum_{i=1}^{n} x_i w_i + b$
Output
Artificial Neuron
w_1, w_2, w_3, w_n - Weights of Connection
x_1, x_2, x_3, x_n - Inputs
b - Bias

(b) Single Neuron in Neural Network [11]

Figure 2.4: Neural Network.

output of the neuron. With increasing number and size of hidden layers, the number of parameters of a network increases. For each layer as in Figure 2.4a, every single neuron calculates an inner product with all the outputs of the previous layer. In such a fully-connected network the number of parameters in fully-connected layers makes optimization of a network harder. Convolutional Neural Networks (CNN) mitigate this problem [35].

A CNN reduces the number of needed parameters in the model by feeding the neurons with a smaller output vector from the previous layer. This allows the construction of deeper networks. Figure 2.5 shows an example of a CNN. Many implementations of CNN were applied to Go [21, 36, 45]. With deep CNN, Maddison et al. [36] achieved a prediction rate of over 50% on Go games played by professional players. This huge increase in prediction rate was the starting point for AlphaGo [45].

2.2 Game Tree Search

In order to represent move sequences in a game, computers use a game tree. This is a directed graph with nodes representing the positions of the game, and directed edges representing moves. A game tree starts from the current position as the root of the tree. Each node in the tree has all the follow-up positions as its children. The end of game positions are leaf nodes in the tree.

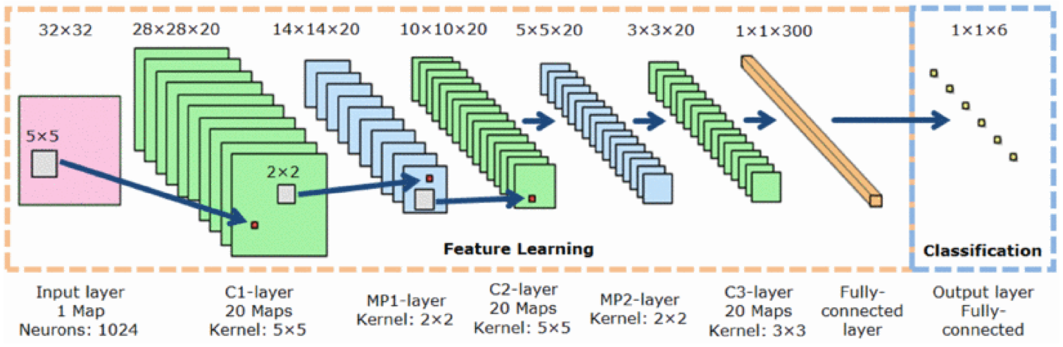


Figure 2.5: Convolutional Neural Network [17].

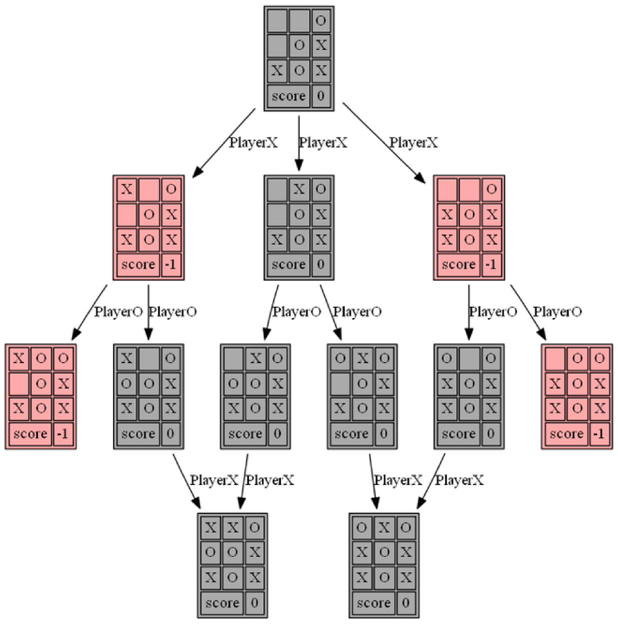


Figure 2.6: Game tree for a Tic Tac Toe Position [13].

Each leaf node has a value associated to it called reward. In the simple case it has a value of +1 for a win, −1 for a loss and 0 for a draw. Games where one player's gain equals the other player's loss are called zero-sum [42]. In this thesis, we focus on perfect information, two player, zero-sum games, and all the subsequent algorithms that we describe are designed for such games. A game is called perfect information if the environment is fully observable. Figure 2.6 shows an example of a game tree for Tic Tac Toe. The root of the tree represents the current position. Edges are possible moves for the current player. On the next level are positions where edges are moves for the opponent. The leaf nodes represent positions at the end of the game.

2.3 Minimax and Alpha-Beta

Minimax [42] is a recursive algorithm for search in two player games. A minimax game tree has two types of nodes, Min and Max. All the nodes on the same level are of the same type. From one level to the next, types alternate between Max and Min. Given the value of its children, the minimax value of a node can be computed based on its type as either the minimum or maximum of the children's values. The Minimax value is reached if both players play optimally [42]. Algorithm 1 shows the pseudo code of Minimax. The naive Minimax algorithm generates a full-width decision tree and has a computational complexity of $O(b^m)$ with branching factor b, and depth m. This is a huge computational cost for games such as Go or chess that have large effective branching factor and depth.

There are many situations where computing the minimax value does not require to generate a full-width tree. Alpha-beta pruning [42] exploits those situations and decreases the effective branching factor in minimax search. As an example, in Figure 2.7.d, State B has a value of 3. The root, which is a max node, will have at least a value of 3. State C has seen its first child and has at most a value of 2. Therefore, optimal play from the root will never choose C, because of the better option B. Therefore, alpha-beta does not need to check other children of C, because it will never be selected. Such pruning can greatly reduce the size of the generated tree from $O(b^d)$ to $O(b^{d/2})$ in the best case [42]; however, the computational complexity of the alpha-beta pruning algorithm remains exponential in the depth of the tree. In the worst case scenario, alpha-beta can not prune any node. In order to address this issue, move ordering heuristics are used. Move ordering tries to visit the best node in any given state first.

In many problems, finding the exact solution by alpha-beta pruning is not possible because of the huge size of the game tree. In those problems heuristic evaluation functions are used. An evaluation function estimates the expected value of a position (state). In order to reduce the size of the game tree, alpha-beta pruning with limited depth traverses the tree to a certain depth, and then estimates the value of the node at that depth by the evaluation function [42]. In order to apply alpha-beta pruning with limited depth on a problem, we need an evaluation function that is able to accurately estimate the expected value of a node for that problem.

Algorithm 1 Example of Minimax algorithm [42]

```
function MINIMAX-DECISION($s$) returns an action $a$
    return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULTS($s$,$a$))

function MAX-VALUE($s$) returns a reward value $R$
    if TERMINAL-TEST($s$) then return REWARD($s$)
    $Q \leftarrow -\infty$
    for each $a$ in ACTIONS($s$) do
        $Q \leftarrow$ MAX($Q$,MIN-VALUE(RESULTS($s$,$a$)))
    return $Q$

function MIN-VALUE($s$) returns a reward value $R$
    if TERMINAL-TEST($s$) then return REWARD($s$)
    $Q \leftarrow \infty$
    for each $a$ in ACTIONS($s$) do
        $Q \leftarrow$ MIN($Q$,MAX-VALUE(RESULTS($s$,$a$)))
    return $Q$
```

State s is a position of a game. ACTIONS(s) returns the list of actions in state s. RESULTS(s,a) transitions from state s by choosing action a to state s'. REWARD(s) returns the reward R for the state s. TERMINAL-TEST checks to see if the state s is terminal. Q is the value of a leaf node.

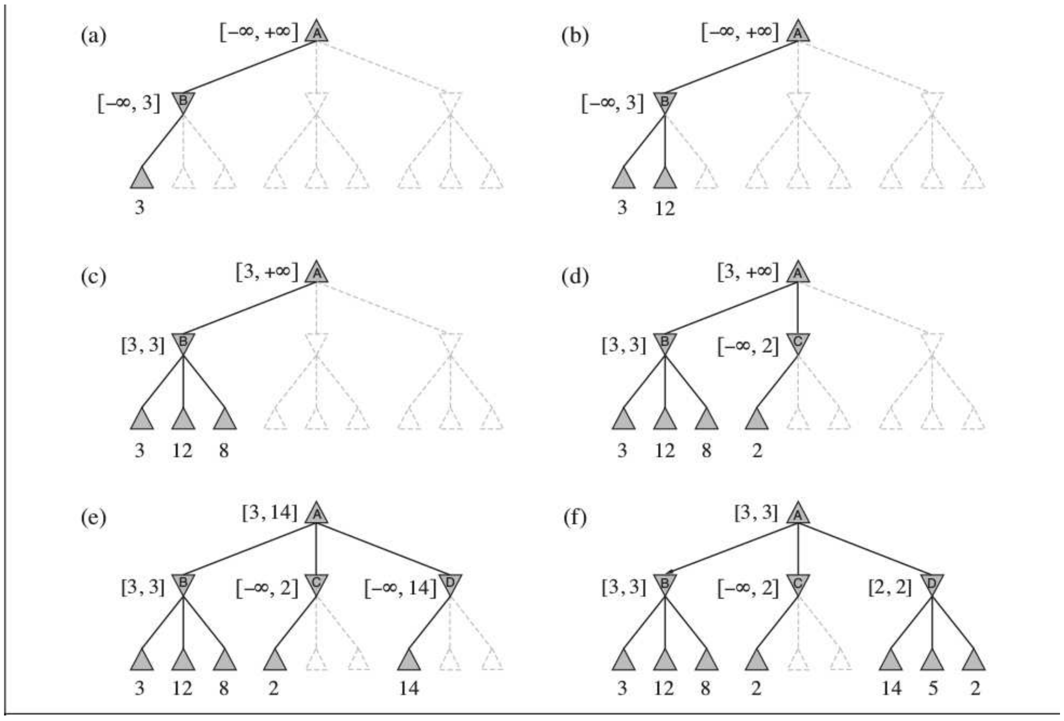


Figure 2.7: Example of alpha-beta pruning algorithm on minimax tree [42].

2.4 The Game of Go and Computer Go

Go is a two player zero-sum game with alternating play and perfect information. Chess, Tic Tac Toe and checkers are other examples of this kind of games.

Go is a strategy game which was invented in China, and later brought to other parts of the world. We explain the main rules of Go. See [1] for more explanation. The game starts with an empty board, usually 19×19. Two players Black and White take turns putting one of their stones on the board with Black going first. White can be given extra points for starting second, which is called komi. The game continues until both players pass consecutively. Stones are captured if they are surrounded by stones of the opposite colour. Captured stones must be removed from the board. When the game ends the player that has the most stones plus surrounded area plus komi wins the game.

The complexity of the game tree in Go makes it a very attractive test bed for artificial intelligence algorithms. There are 361 points on the board and each can have one of the 3 possible states of Black, White or empty. This results in 3^{361} board positions. This number is an upper bound on the number of board positions, since some states are illegal. However, the possibility of playing back into points where stones were captured greatly increases the game complexity. The computational complexity of the game makes it intractable to solve on the

Figure 2.8: Game of Go Board [5].

full size board by the alpha-beta algorithm. Until very recently, there was a lack of good heuristics for evaluating Go positions, which made the alpha-beta algorithm fail in Go.

2.5 Monte Carlo Methods

Monte Carlo (MC) methods were first popularized in physics to approximately solve intractable integrals by using sampling [18]. Later Abramson [14] applied Monte Carlo methods to games. He showed that the expected value of a move through random play from a node was better than a handcrafted evaluation function created by experts in the Game of Othello; however, he also noted that obtaining such a value needs many rounds of random play. Equation (2.6) shows how rewards obtained through rounds of play are used to compute the expected value Q of a move a:

$$\begin{aligned} &N_s \leftarrow N_s + 1 \\ &N_{s,a} \leftarrow N_{s,a} + 1 \\ &Q_{s,a} \leftarrow Q_{s,a} + \left[\frac{R_t - Q_{s,a}}{N_{s,a}}\right] \text{, formula for updating the average} \end{aligned} \tag{2.6}$$

- N_s is the overall number of visits of state s.
- $N_{s,a}$ number of times action a was selected in state s.
- $Q_{s,a}$ is the average reward obtained from move a in state s
- R_i is reward obtained at i-th game

2.5.1 Upper Confidence Bound (UCB) and the UCB1 Algorithm

When we apply Monte Carlo methods on a problem we need to repeatedly make a choice between n actions at the same state, where each action leads to a reward value. Choosing the next action is usually done by combining two ideas, greedy exploitation prefers an action that has maximum average reward up to that time, while exploration chooses the next action based on reducing the largest uncertainty. Auer et al. [16] proposed the UCB1 algorithm as a balanced solution that combines exploitation and exploration. This is achieved by computing an upper confidence bound on the reward. In UCB1, an action with large Q value still has an advantage; however, as number of visits N_s grows, actions with small number of visits $N_{s,a}$ obtain larger exploration bonus. This increases the urgency of trying moves with low visit number.

Algorithm 2 Deterministic policy UCB1 [16].

```
Initialization: play each action once
Loop:
    On trial $N_s$ in state $s$ choose action $a^*$ where:
    $a^* = \arg\max_a \left[Q_{s,a} + \sqrt{\frac{2\ln N_s}{N_{s,a}}}\right]$
```

2.5.2 UCB applied to Trees (UCT)

Although UCB1 is able to address the dilemma between exploration and exploitation, it was designed for choosing a single action in bandit problems, not sequential decision making as in games. Kocsis et al. [34] proposed the UCT algorithm based on UCB1 algorithm. UCT expands a search tree for selecting the next action.

UCT uses Monte Carlo simulations to estimate the value of each action (move). Each simulation repeats four phases: *Selection, Playout, Expansion and Update*, as Figure 2.9 shows. In this section we briefly describe each.

Selection starts at the current position of the game, represented by the root of the tree, and repeatedly selects the next node in the game tree based on the UCT formula, until it reaches a leaf node. A *Playout* or *rollout*, starts at the leaf node reached by *selection*, and repeatedly selects the next state of the game based on a simulation policy until it reaches the end of the game, and computes the result of the game. *Expansion* adds one or more children to the leaf node that we have reached through selection. *Update* propagates the simulations result back up the tree.

Kocsis et al. proved that as the number of sampled games goes to infinity, the tree produced by UCT converges to a mini-max tree. Any method that builds a tree using a Monte Carlo method to perform the search is called MCTS; nonetheless, for the remainder of this thesis, whenever we use the term MCTS, we mean a method based on UCT or an extension of the UCT formula.

Algorithm 3 UCT [34]

```
Loop:
    While $s$ is not a leaf:
        On trial $N_s$ in state $s$ choose action $a$ where:
        $a = \arg\max_a \left[Q_{s,a} + C_p\sqrt{\frac{\ln N_s}{N_{s,a}}}\right]$
        next state: $s \leftarrow s_a$
    Playout from state $s$
    Update
$C_p$ is a tunable exploration hyperparameter.
$s \leftarrow s_a$: takes action $a$ in state $s$ and results in a new state.
Playout: plays the game from given state until the end using playout policy.
Update: updates the tree statistics using the latest playout results.
```

The first difference between UCT (Algorithm 3) and UCB (Algorithm 2) is the addition of a tunable exploration parameter C_p. The other difference is that UCT is applied to all the nodes in a tree that are visited during selection.

2.5.3 PUCB

PUCB is a modification to UCT (Algorithm 3) proposed by Rosin [41]. This algorithm adds a term to UCT to incorporate feature knowledge for evaluating each action a. Algorithm 4 shows the method.

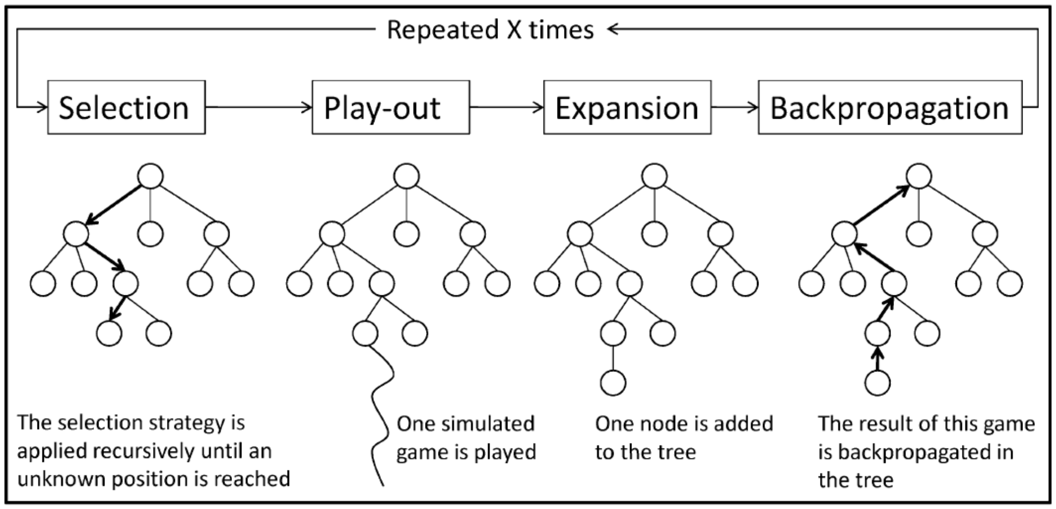

Repeated X times
Selection
Play-out
Expansion
Backpropagation
The selection strategy is applied recursively until an unknown position is reached
One simulated game is played
One node is added to the tree
The result of this game is backpropagated in the tree

Figure 2.9: Phases of MCTS [51].

Algorithm 4 PUCB [16].

```
**Initialization:** play each action once
**Loop:**
    On trial $N_s$ in state $s$ choose action $a$ where:
    $a = \arg\max_a \left[Q_{s,a} + C(N_s, N_{s,a}) - M(N_s, a)\right]$
    • $C(N_s, N_{s,a}) = \sqrt{\frac{C_p \log(N_s)}{N_s}}$ if $N_{s,a} > 0$, otherwise 0; $C_p = 3/2$.
    • $M(N_s, a) = \frac{2}{f^{eval}_{s,a}} \sqrt{\frac{\log(N_s)}{N_s}}$ if $N_s > 1$, otherwise $\frac{2}{f^{eval}_{s,a}}$.
    • $f^{eval}_{s,a}$ is the feature evaluation of move $a$ in
      state $s$.
```

2.5.4 AMAF and RAVE

Rapid Action Value Estimation (RAVE) [26] is a technique for estimating the value of a move, and it uses the All Moves As First (AMAF) [29] technique. AMAF estimation is based on the assumption that the value of a move is unaffected by when the move is played during a game. AMAF produces a rough estimation of a move's value, "as the value of an action (move) usually depends on the exact state in which it is selected [25]"

$$\hat{Q}_{s,a} = \mathbb{E}[z|s_t = s, \quad \exists\, u \geq t \quad s.t.\ a_u = a] \tag{2.7}$$

where z is the game result, t is the time step (move number), and $\hat{Q}$ is the estimated value of a move a using AMAF.

AMAF uses Monte Carlo simulations to estimate the $\hat{Q}$ value by averaging the result of every simulation, which started from state s in which action a was played after time t.

In early stages of MCTS, when we have not gathered enough information to accurately estimate a move's value, the RAVE estimate can help to improve the performance of search by biasing search towards the moves with better RAVE value. [25] observed up to 36% increase in the win-rate of their player MoGo [9] against GnuGo [6] by applying RAVE. When the number of simulations for a move increases, we have a better estimation of the move value. We decrease the weight of RAVE in move selection, by computing a weighted combination of action value obtained through Equation (2.7) and Equation (2.6), gradually fading out the $\hat{Q}_{s,a}$ term.

$$\begin{aligned} \hat{Q}^*_{s,a} &= (1 - \beta_{s,a})Q_{s,a} + \beta_{s,a}\hat{Q}_{s,a} \\ \beta_{s,a} &= \sqrt{\frac{k}{3N_s + k}} \end{aligned} \tag{2.8}$$

where $Q_{s,a}$ is obtained by Equation (2.6), $\hat{Q}$ is the RAVE estimate, $\beta_{s,a}$ is the weight parameter, k is a hyperparameter, and N_s is number of visits for state s

The UCT-RAVE algorithm [26] replaces $Q_{s,a}$ in Algorithm 3 with $Q^*_{s,a}$ from Equation (2.8).

2.5.5 Usage of Knowledge in MCTS

Domain specific knowledge in MCTS can be applied in different ways. One approach is using knowledge to initialize a new node in the tree [23] to bias the in-tree selection strategy towards or away from that node. Another application of knowledge is its usage in the playout phase to strengthen play by playout policy, and avoid some blunders. Silver and Tesauro [48] show that improving the strength of a playout policy alone does not necessarily lead to improved level of play, because it can increase bias. Reducing the bias of a playout policy plays a significant role in the strength of a MCTS player. Knowledge can also be applied as an evaluation function to cut off the search on the simulation early, and estimate the game results at that point [45]. Many program authors have reported an increase in performance of their program from adding knowledge [37, 21, 45, 48, 27, 25, 53, 41].

Chapter 3

Tools and Methods for Experiments

In this chapter we describe tools, methods and settings used in the experiments of Sections 3.3 and 3.4. We use Fuego [23] as a test-bed to conduct our experiments. In Section 3.1 we describe Fuego and its algorithm settings. We then describe the tested players in Section 3.2. We describe our experimental methods for move prediction and measuring playing strength in Sections 3.3 and 3.4. We briefly describe previous work on the analysis of game-playing programs in Section 3.5. In Section 3.6 we revisit our research questions to give more precise descriptions in terms of the definitions introduced here.

3.1 The Fuego Framework

Fuego [23] is an open-source MCTS-based search engine mostly developed by a team at the University of Alberta. Fuego is a collection of game independent libraries for two player, perfect information board games, and also contains a set of MCTS-based Go players.

We briefly describe how each of the algorithms relevant to our experiments are implemented in Fuego. SVN revision 2032, updated on $2016 - 08 - 16$, was used in our experiments.

3.1.1 UCT Move Selection

The Fuego implementation follows the standard MCTS algorithm, with in-tree move selection described in Section 2.5.2.

On trial N_s, Fuego chooses action $a^* = \arg\max_a \Big[Q_{s,a} + C(N_s, N_{s,a})\Big]$ where:

- N_s is the number of visits of current state.
- $N_{s,a}$ is the number of previous visits of move a in state s.
- $Q_{s,a}$ weighted mean of move value and RAVE value of move a in state s.
- C_p is a hyperparameter for controlling exploration with default value of 0.7.
- $C(N_s, N_{s,a}) = C_p * \sqrt{\dfrac{\log(N_s)}{N_{s,a}}}$

$$\tag{3.1}$$

3.1.2 Initialization of N and Q Values

Fuego uses feature evaluation as a prior knowledge. When a new node is added to the Monte Carlo tree, its value is initialized by prior knowledge trained by the LFR [37] method. Prior knowledge evaluation outputs a real number, and Fuego uses Equations (3.2) and (3.3) to transform that number into an initial number of visits $N_{s,a} = N_{s,a}^{prior}$ and an initial value $Q_{s,a} = Q_{s,a}^{prior}$ of move a in state s [52].

$$N_{s,a} = \begin{cases} \frac{c \times |\Gamma(s)|}{\Sigma_a f_{s,a}^{eval}} \times f_{s,a}^{eval} & \text{if } f_{s,a} \geq 0 \\ -\frac{c \times |\Gamma(s)|}{\Sigma_a f_{s,a}^{eval}} \times f_{s,a}^{eval} & \text{otherwise} \end{cases} \tag{3.2}$$

Here, c is a hyperparameter, $f_{s,a}^{eval}$ is the feature evaluation of move a in state s, and $|\Gamma(s)|$ is number of all legal moves in state s.

$$Q_{s,a} = \begin{cases} \frac{1+f_{s,a}^{eval}}{2 \times f_{max}^{eval}} & \text{if } f_{s,a} \geq 0 \\ -\frac{1+f_{s,a}^{eval}}{2 \times f_{min}^{eval}} & \text{otherwise} \end{cases} \tag{3.3}$$

where f_{min}^{eval} and f_{max}^{eval} are minimum and maximum feature evaluations of all moves in $\Gamma(s)$.

3.1.3 Additive Knowledge

In order to extend UCT with feature knowledge, Fuego also uses a variant of the PUCB method described in Section 2.5.3. However, the PUCB implementation in Fuego has some differences with the method described in Section 2.5.3. Equation (3.4) shows the details.

On trial N_s choose $a^* = \arg\max_a (Q_{s,a} - p_w * p_v)$ where:

- $p_w = \sqrt{\dfrac{5}{N_s + 5}}$
- $p_v = c_{add} * Sig(c_{sig} * f^{eval}_{s,a})$

Here, c_{add} with default value of 1 and c_{sig} with default value of 10 are hyperparameters. $Sig(t) = \frac{1}{1+e^t}$ is the sigmoid function and $f^{eval}_{s,a}$ is the feature evaluation of move a. (3.4)

One difference between Equation (3.4) and Algorithm 4 is that the exploration term C_p is set to zero in here. The other difference is that the hyperparameters used in here are different from Algorithm 4. Fuego also does not follow the initialization step in Algorithm 4, and instead uses feature knowledge for initialization.

3.1.4 Simple Features and Patterns in Fuego

Fuego uses complex knowledge for in-tree selection and simple knowledge in the playout phase. Fuego also uses feature-based evaluation in an additive term to bias moves during in-tree selection. The features used in additive knowledge are diamond shape patterns similar to Figure 2.3 with size 4 [52].

3.1.5 Playout Policy

The playout policy in Fuego consists of a set of prioritized methods. Fuego runs these methods in a top-down manner. It stops as soon as a method returns a nonempty set S of candidate moves. Fuego chooses a move uniformly at random between moves in S, except in the case of 3×3 pattern move features, which have associated weights. In this case, the chance of a move being selected is proportional to its weight. Features in Fuego are similar to the ones in [22] using 3×3 patterns and features.

3.1.6 Move Filtering

Fuego uses a move filtering technique to reduce the branching factor of the tree. This method can filter moves that are captured in a ladder, moves on the first line, and moves inside safe groups. For more information on the implementation of the move filtering algorithm in Fuego, please refer to [4].

3.2 Fuego-Based Players Used in our Experiments

In our experiments we have used a set of players from the Fuego code base. Here, we briefly describe each player. For detailed information on the settings of each player, please refer to Appendix A.

3.2.1 Playout Policy-Only Player

This simple player uses only the playout policy of Fuego, described in Section 3.1.5, for generating the next move in the game, and it does not use search. This player helps us to understand the playout policy in Fuego better, and also helps us to measure different aspects of the playout policy, such as move prediction and playing strength.

3.2.2 Simple Features-Only Player

Here, we use the prior knowledge in Fuego as a stand-alone player. The highest evaluated move according to features is played. Having a features-only player helps us to understand how the knowledge encoded in features compares to search, and it also helps to better evaluate feature knowledge.

3.2.3 No Knowledge Player

In order to examine how knowledge helps the performance of a player, we turn off prior knowledge and move filtering in Fuego. This player uses only MCTS with the default Fuego playout policy. This player helps us better understand the impact of knowledge on a player, and specifically on move prediction and playing strength.

3.2.4 No Additive Player

Fuego by default uses additive knowledge to help its in-tree policy focus more on high-ranking moves. We turn off the additive knowledge in this player, and rollback Fuego to use MCTS with the UCT method as described in Section 3.1.1. This player helps us to better understand the role of additive knowledge in Fuego.

3.2.5 Default MCTS-Based Fuego Player

We need to be able to compare the results obtained by other players with full-strength Fuego. This player uses the full Fuego engine with all default settings.

3.2.6 Varying the Number of Simulations

For the MCTS-based players in Sections 3.2.3 to 3.2.5 we vary the number of simulations in {100, 300, 1000, 3000, 10000}. This helps us to understand the impact of more simulations on the players.

3.3 Move Prediction

One of the main experiments that we have conducted in order to evaluate our players is move prediction. In this task we have used games played by professional players. We run a player on all the positions from each game, and let it predict the next played move.

3.3.1 Move Prediction Data

For the move prediction task, we used games from Pro Game Collection [12]. In total we used 4621 games, after removing games that were played on board sizes other than 19×19. These games were played from Sep 2013 to Nov 2016. This time frame was chosen because Fuego features were trained on games prior to Sep 2013 from the same collection. Choosing this period makes sure that the test set has not been seen by our programs before.

3.4 Playing Strength

In this experiment we play matches between different players to obtain a measure of their relative strength. We used GoGui [7] to automate the process. For detailed information on the scripts, please refer to Appendix A.

3.5 Previous Work on Analysis of Go Programs

Many game programs have been used for scientific research over the years [23, 27, 20, 19]. Here, we describe studies on analyzing playout policies, simulations and strength in MCTS-based Go programs.

3.5.1 Combining Online and Offline Knowledge in UCT

Sylvain Gelly and David Silver [25] studied the impact of three different ideas on the performance of MoGo [9]:

- Impact of strength of playout policy
- Impact of RAVE
- Impact of prior knowledge

Impact of Strength of Playout Policy

For a set of playout policies P that were based on a value function Q trained by Reinforcement Learning techniques [50], the authors measured the strength of every playout policy as a stand-alone player by performing a round-robin tournament between the policies P. Then they measured the strength of versions of MoGo that use the playout policies in set P by playing games against a fixed opponent GnuGo 3.7.10 (level 0) [6]. To their surprise, MoGo's default handcrafted policy, which was weaker as a standalone player, improved the performance of MoGo compared to the other policies in these games.

Impact of RAVE

Another experiment measured the impact of RAVE on the performance of MoGo. They observed up to 36% improvement in MoGo's win-rate against GnuGo 3.7.10 (level 8) by adding RAVE.

Impact of Prior Knowledge

The last part of the study [25] focused on analyzing the impact of prior knowledge on the performance of MoGo extended with RAVE. Feature knowledge was used to assign an initial value $Q_{s,a}$ and a number of visits $N_{s,a}$ for a move a. Adding prior knowledge improved the win-rate of MoGo by 9% against GnuGo 3.7.10 (level 8).

3.5.2 Monte Carlo Simulation Balancing

David Silver and Gerald Tesauro studied the impact of *balance* in a playout policy on the performance of a player [48]. This work introduced an imbalance measure B_∞ as the expectation of the squared bias $b(s)^2$.

$$\begin{aligned} b(s) &= V^*(s) - \mathbb{E}_{\pi_\theta}[z|s] \\ g(s) &= \nabla_\theta \mathbb{E}_{\pi_\theta}[z|s] \\ B_\infty(\theta) &= \mathbb{E}_\rho[b(s)^2] \\ \nabla_\theta B_\infty(\theta) &= \nabla_\theta \mathbb{E}_\rho[b^2] = -2\mathbb{E}_\rho[b(s)g(s)] \\ & \text{where } z \text{ is the result of a game obtained through simulations, } V^*(s) \text{ is minimax value of a state } s, \pi \text{ is a playout policy, } \theta \text{ are the parameters of } \pi, \text{ and } \rho \text{ is the distribution of states.} \end{aligned} \tag{3.5}$$

In this work, first V^* is approximated by using deep Monte Carlo search for each state. Then an optimization method is applied to Equation (3.5) to balance the policy π. The resulting policies were tested against a set of machine learned policies. While the machine learned policies were stronger as a stand alone player, when used in an MCTS program, the program with balanced policy outperformed the other versions.

Simulation Balancing in Practice

Another Monte Carlo Simulation Balancing technique was analyzed by Aja Huang et al. in [31], and applied to the Go playing program Erica. The playout policy in Erica was pattern based, and trained by the MM technique [22].

After training a playout policy using the MM method, it was then balanced by simulation balancing. Erica was tested against Fuego 0.4 on the 9×9 board size by performing 1000 games with 3000 simulations per move. Simulation balancing improved the win-rate of Erica from 40.9% to 78.2%.

3.5.3 Analyzing Simulations in MCTS

Sumudu Fernando and Martin Müller conducted a study to examine playout policies in Fuego [24]. They studied three hypotheses:

- The strength of a playout policy is strongly correlated with the preservation of the game theoretic status of the game. For example, if Black is winning when the playout starts, at the end of the playout Black should be reported as the winner.
- The size of errors made during simulation matters.
- Given a playout policy, having no systematic bias is more important than having low error rate.

In order to test their hypotheses, Fuego was used to conduct the experiments on the 9×9 board size. Different variations of Fuego's playout policy were used in their study, by selecting subsets of the playout policy rules in Fuego.

Balance and Strength

To investigate the correlation between balance and strength of a policy to address their first two hypotheses, the authors measured the number of blunders each of their selected policies makes during the playout phase over 100 self-played games. In order to measure blunders, each position of the game before and after a move was evaluated by 5000 simulations using the default settings of Fuego. This results in a good estimate of the value for a position. If a move changes the estimate from over 0.75 to under 0.25, it is marked as a blunder. When a playout policy makes zero or an even number of blunders that policy

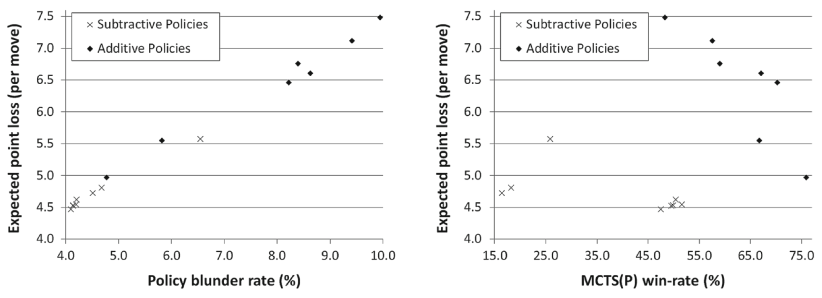


Figure 3.1: Expected point loss of policy P vs policy blunder rate (left) and vs relative strength of MCTS(P) (right) [24].

would be considered strong, since it keeps the game theoretic status of the position.

The obtained results do not show an "appreciable correlation" between balance and strength in the experiment. One reason for failed correlation were long series of blunders occurring during the playout, which makes the parity of the number of blunders random. Another reason is that non-blunder moves with smaller error add up and eventually change the result of the game, without being caught by this approach.

Error Size

To measure the policy error, the authors selected a number of positions uniformly from more that 50000 games on CGOS [2] on board size 9×9. For each position, a *fair komi* value was computed, which is an estimate of the value of a position for Black. Fuego was used to estimate the winning probability for Black by changing the komi value. The point where the winning probability becomes 50% is defined as the fair komi value. After using binary search to obtain the fair komi value for a position, all legal moves from that position were played, and the fair komi value of all the resulting positions was computed as well. The difference between original and after-move value was assigned as the move value loss. Each move that changed the winning status to losing and also lost at least 5 points was defined as a blunder.

The authors observed a strong correlation between expected point loss and policy blunder rate. They also found a "suggestive negative correlation" between expected point loss and relative strength of a policy. Figure 3.1 shows their results.

3.6 Revisiting the Research Topics

In this section we re-visit our research questions, and rephrase them using the terms that we introduced so far.

- Examine the relation between knowledge and search in Go programs and how these two impact each other.

  We want to know how knowledge impacts simulations in Fuego, and how we can express the moves with high number of simulations in terms of prominent features, and what impact the evaluation of feature knowledge has on those moves. This is discussed in Section 4.4.

- Examine current evaluation approaches used in Go programs, which are: move prediction and playing against another program or human. Understand the differences between each of these tests and how they relate to each other.

  We want to know what is the evaluation of move prediction on the players that we described in Section 3.2, and how those results compare to the results of matches that those players play against each other. This is discussed in Sections 4.1 to 4.3.

- Evaluate the impact of knowledge on the performance of a Go program.

  After understanding the difference between measures of evaluation, we want to know how usage of simple features for initialization of new nodes in the tree can impact the performance of a player in terms of both move prediction rate and playing strength. This is covered in Section 4.4.6.

- How does longer and deeper search improve the strength of a MCTS program, in the presence of knowledge?

  We want to measure the impact of varying the number of simulations in default MCTS-based Fuego. We analyze the results of move prediction and playing strength. This is discussed in Sections 4.1 to 4.3.

- Can this increased strength be explained in terms of simple feature knowledge?

  Can the impact of simple feature knowledge on the evaluation measures also be observed in the frequency of features present in the default MCTS-based Fuego player moves? This is discussed in Section 4.4.

Chapter 4

Experimental Results and Discussion

In this chapter we provide the results of our experiments. We explain those results, and use them to answer the research questions.

4.1 Move Prediction

Table 4.1 shows the results of the move prediction task described in Section 3.3 on the test set with positions from 4621 games. The players are Fuego-based engines described in Section 3.2. The move prediction rate is the fraction of positions for which the master move was predicted correctly. For the No Knowledge, No Additive, and Default Fuego players the number in the name represents the number of simulations per move used by that player. Figure 4.1 shows the prediction rate for various number of simulations.

The Playout Policy-Only and Simple Features-Only players do not use Monte Carlo simulations. Playout Policy-Only was only able to predict less than 22% of professional moves. Simple Features-Only has a much higher prediction rate of approximately 31%. Given the fact that neither of those two players uses MCTS, the gap signifies the role of the knowledge obtained through a large set of simple features trained by machine learning methods in the Simple Features-Only player, compared to the combination of fast rules and small patterns in the Playout Policy-Only player.

Removing all knowledge has a big negative impact on the prediction rate in MCTS. It drops the prediction rate to 12% in the No Knowledge player with 100 simulations. Adding more simulations compensates for the lack of

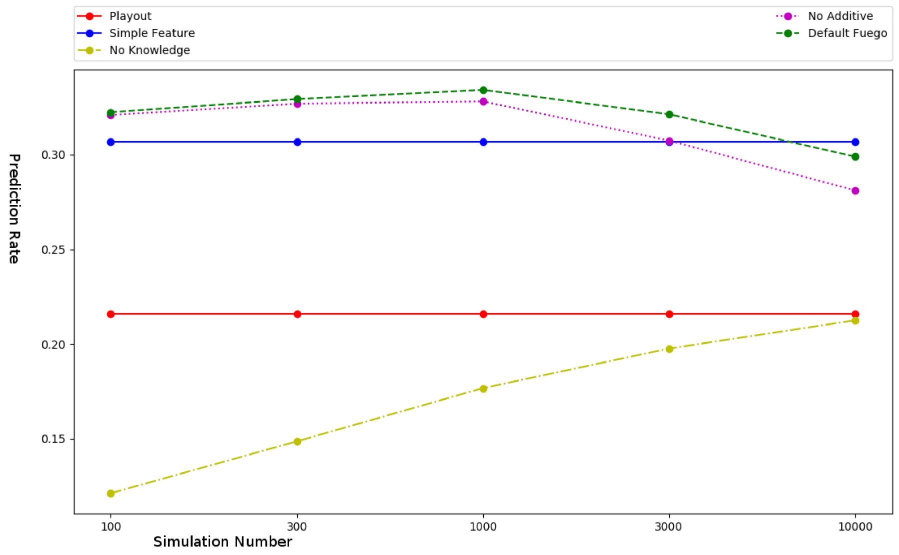


Figure 4.1: Graph of move prediction rate.

knowledge to some degree. With 10000 simulations, the prediction rate of the No Knowledge player increases to over 21%. Nonetheless, this is still far below the move prediction rate of any MCTS player utilizing knowledge. This shows the role of knowledge in giving directions to MCTS toward nodes with better outlook, when the number of simulations is limited.

The prediction rate of the No Additive player is between approximately 28% and 33%. Up to 1000 simulations increasing the number of simulations improves the prediction rate; however, after that it starts to drop. When we compare the results of a No Additive player to the Default MCTS-based player with the same number of simulations, we observe a similar pattern in change of prediction rate. The difference between prediction rates of the Default Fuego player and the No Additive player for simulations between 100 to 10000 are: 0.0015, 0.0024, 0.0061, 0.0139, 0.0178. This shows that as the number of simulations grows, additive knowledge slows down the drop of prediction rate in the Default Fuego player, and biases the selection policy in MCTS more towards professional player moves. This widening gap can also be observed in Figure 4.1.

Given the obtained results several new questions arise:

- Is there any difference in strength between players with similar prediction rate?
- What role does the number of simulations play in players strength vs prediction rate?

Experiment	Accuracy
Playout Policy-Only	0.2160
Simple Features-Only	0.3066
No Knowledge 100	0.1212
No Knowledge 300	0.1486
No Knowledge 1000	0.1767
No Knowledge 3000	0.1976
No Knowledge 10000	0.2125
No Additive 100	0.3209
No Additive 300	0.3269
No Additive 1000	0.3281
No Additive 3000	0.3074
No Additive 10000	0.2811
Default 100	0.3224
Default 300	0.3293
Default 1000	0.3342
Default 3000	0.3213
Default 10000	0.2989

Table 4.1: Result of move prediction for players based on Fuego.

- Why does the prediction rate for No Additive and Default MCTS players start to drop?

In order to start investigating these questions, we next conducted two experiments. The first experiment measures the playing strength of players against each other. The second measures the move prediction rate in different stages of the games.

4.2 Playing Strength

In order to answer the first two questions in Section 4.1, we created a round robin tournament between all the 11 players described in Section 3.2. Each round consists of 100 games between two players, with each player playing Black 50 times. All players except the Simple Features-Only player use randomization, which resulted in not having any duplicated games. We used GoGui [7] to perform the tournament. Results of the tournament are reported in Table 4.2. Reported results are from the perspective of the player in the row against the player in the column. For example the entry in the second row, second column shows that the Playout Policy-Only player has won 0 games against the No Additive player with 1000 simulations.

experiments	No Additive 1000	No Additive 300	No Additive 100
Playout Policy-Only	0	0	0
No Additive 1000	–	100	100
No Additive 300	0	–	100
No Additive 100	0	0	–
No Knowledge 1000	7	60	90
No Knowledge 300	0	17	65
No Knowledge 100	0	0	17
Default 1000	51	99	100
Default 300	1	55	97
Default 100	0	3	50
Simple Features-Only	0	0	3
experiments	Default 1000	Default 300	Default 100
Playout Policy-Only	0	0	0
No Additive 1000	49	99	100
No Additive 300	1	45	97
No Additive 100	0	3	50
No Knowledge 1000	4	65	87
No Knowledge 300	0	12	40
No Knowledge 100	0	0	7
Default 1000	–	100	100
Default 300	0	–	98
Default 100	0	2	–
Simple Features-Only	0	0	4
experiments	No Knowledge 1000	No Knowledge 300	No Knowledge 100
Playout Policy-Only	0	0	0
No Additive 1000	93	100	100
No Additive 300	40	83	100
No Additive 100	10	45	83
No Knowledge 1000	–	95	100
No Knowledge 300	5	–	98
No Knowledge 100	0	2	–
Default 1000	96	100	100
Default 300	35	88	100
Default 100	13	60	93
Simple Features-Only	2	26	67
experiments	Playout Policy-Only	Simple Features-Only	–
Playout Policy-Only	–	0	–
No Additive 1000	100	100	–
No Additive 300	100	100	–
No Additive 100	100	97	–
No Knowledge 1000	100	98	–
No Knowledge 300	100	74	–
No Knowledge 100	100	33	–
Default 1000	100	100	–
Default 300	100	100	–
Default 100	100	96	–
Simple Features-Only	100	–	–

Table 4.2: Result of 100 game matches between all pairs of players.

Table 4.2 summarizes the results of the tournament. In order to highlight some interesting trends, we have created five colour groups in the table.

4.2.1 Red Group of Experiments: Default MCTS-based Fuego vs No Additive Player

This compares the experiments with same number of simulations between the No Additive and default MCTS-based Fuego player. Increasing simulations does not change the balance of strength between these two settings, and removing additive knowledge had minimal impact on playing strength. This is consistent with what we observed in the move prediction task. It can be concluded from the result that these two players have almost the same playing strength against each other when using the same number of simulations.

4.2.2 Yellow Group of Experiments: No Knowledge vs Other MCTS-based Players

The playing strength of the No Knowledge player decreases most of the time against an opponent with the same number of simulations as the number of simulations increases. The role of knowledge becomes more important as a player's strength increases. Knowledge helps a player to avoid crucial mistakes in a game, where a stronger opponent can better exploit those mistakes. While it seems that increasing the number of simulations should compensate for lack of knowledge, there are two reasons that we do not see that effect in this group of experiments. First, the opponent also benefits from an increased number of simulations. Second, in a player that uses the knowledge, increasing the number of simulations leads to more visits of promising moves that the knowledge picks. This enables the player to examine these moves more deeply, and pick the best among them. The No Knowledge player is less focused and needs more simulations to achieve the same effect.

4.2.3 Blue Group of Experiments: Varying Number of Simulations, 300 vs 100

As expected from previous experience with MCTS-based engines, we can see that in every case, a 3x increase in number of simulations leads to a huge difference in playing strength. This is in sharp contrast to the move prediction task in Table 4.1, where the difference was small and sometimes even negative. This shows that using the move prediction rate as a measure to examine a player is not as informative as we expected it to be. There remain aspects of a player which strongly affect its comparative strength against another player, which move prediction is unable to reveal.

4.2.4 Green Group of Experiments: No Additive vs Other MCTS-based Players

This colour group compares No Additive with other MCTS-based players with the same number of simulations. In all these experiments removing the additive term has limited impact on playing strength. The biggest change in playing strength between the No Additive and Default Fuego player is in 300 simulations, where Default Fuego player won 55% of games, in 1000 simulations it is only 51% win-rate for the Default Fuego player, and in 100 simulations no difference is made. These changes in win-rate seems to be due to randomness. Removing feature knowledge decreases the playing strength by a huge margin, with win-rates of 7-17% for the No Knowledge player.

4.2.5 Gray Group of Experiments: Simple Features-Only vs No Knowledge Players

This scaling experiment shows how many simulations are needed to reach and surpass Simple Feature knowledge. With 100 simulations, the No Knowledge player is weaker than feature knowledge: it loses 67 games. With 300 simulations, the No knowledge player surpasses the strength of the Simple Features-Only player, and with 1000 simulations the No Knowledge player is much stronger, winning 98 of 100 games.

4.3 A Closer Look at Move Prediction Rate

In Section 4.1 of the previous experiment, surprisingly the move prediction rate did not show any major difference between Default Fuego and the No additive player when the number of simulations was varied between 100 to 1000, while Section 4.2 showed undeniable differences in strength between those players. We also want to understand why the prediction rate starts to drop after 3000 simulations in the Default MCTS-based and No Additive players. In this experiment, we study the effect of the game phase. We divide a game into six intervals from the opening to the endgame, and measure the prediction accuracy of each player separately for each interval. We created six intervals of 50 moves each, corresponding to move 0 to move 300. Because of the limited number of available samples after move 300 we ignored those final small endgame moves.

Figure 4.2 shows the move prediction accuracy per interval for Default Fuego with 100 and 1000 simulations, and for No Additive with 100 and 1000 simulations. While Table 4.1 showed no noticeable difference between 100 and 1000 simulations, Figure 4.2 shows that for the first 200 moves there is a major difference in both Default Fuego and No Additive players, with a higher prediction rate for 1000 simulation player. This difference fades from move 200-250 and turns to the opposite from moves 251-300.

Figure 4.3 shows the prediction accuracy for experiments where we saw the drop of prediction rate with 3000 and 10000 simulations for No Additive and Default Fuego. We added the 300 simulation players as a baseline. In the opening, the prediction rate for the Default Fuego players increases with number of simulations, and for No Additive players remains very similar for the first 50 moves. From the second interval to the last, the prediction rate of the 300 simulation players sharply increases. For the 3000 simulation players this increase is more moderate. In the 10000 simulation players we observe a drop of prediction rates for the first 250 moves, and then a slight rebound.

To explain the lower prediction rate in the late endgame in players using more simulations, we need to look at how the selection policy in MCTS works. In a game when one player's winning probability is very high, there are many moves that still result in winning for that player, while being sub-optimal in terms of score. The selection policy in Fuego maximizes winning probability, not score. After 200 moves, the winner of most of the games can be predicted with high confidence by strong players. There are many moves in those games that do not change the outcome, and Fuego chooses a "safest" move according to its noisy simulations. Professional players will not usually select such point-losing "safe" moves. Another reason lies in the impact of knowledge on players with fewer simulations. As we saw in Section 3.1.2, knowledge is used to initialize the value of a node in the Monte Carlo tree. When the number of simulations is still small, this initialization plays a major role in MCTS search. Since it is based on features learned from professional games, it biases the search toward professional moves. However, as the number of simulations grows the impact of initialization diminishes relative to the result of simulations.

4.4 Move Prediction and Feature Frequency

Since the move prediction rate alone does not explain the difference in playing strength, we try to find other differentiating factors between various players by focusing on features. Features play a major role in the success of a player.

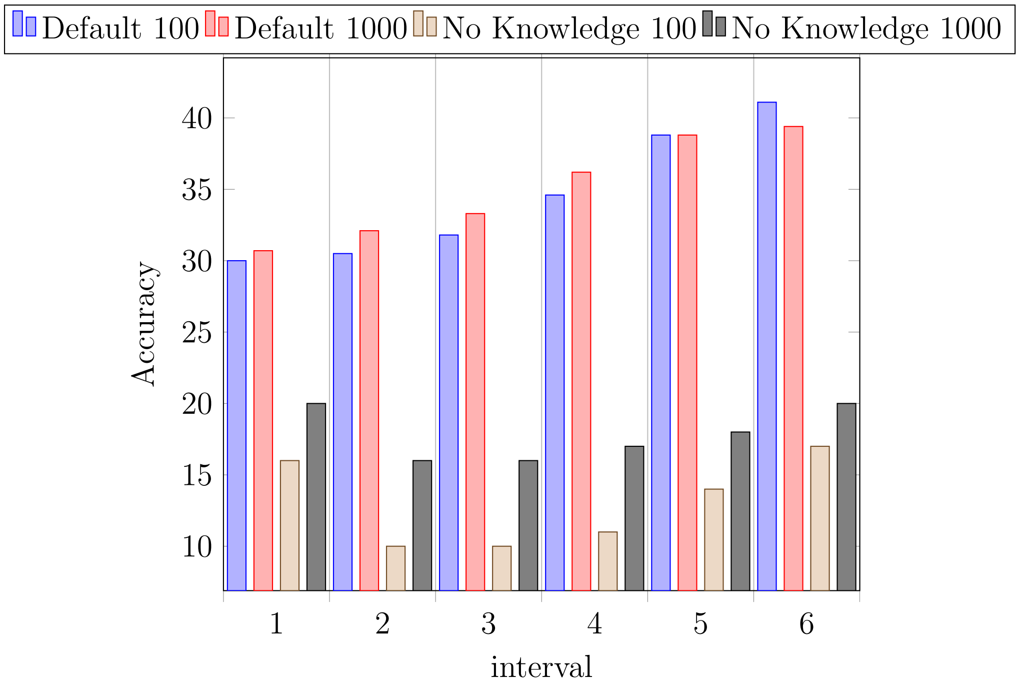


Figure 4.2: Move prediction accuracy per game phase for 100 and 1000 simulation players. Each group has 50 moves.

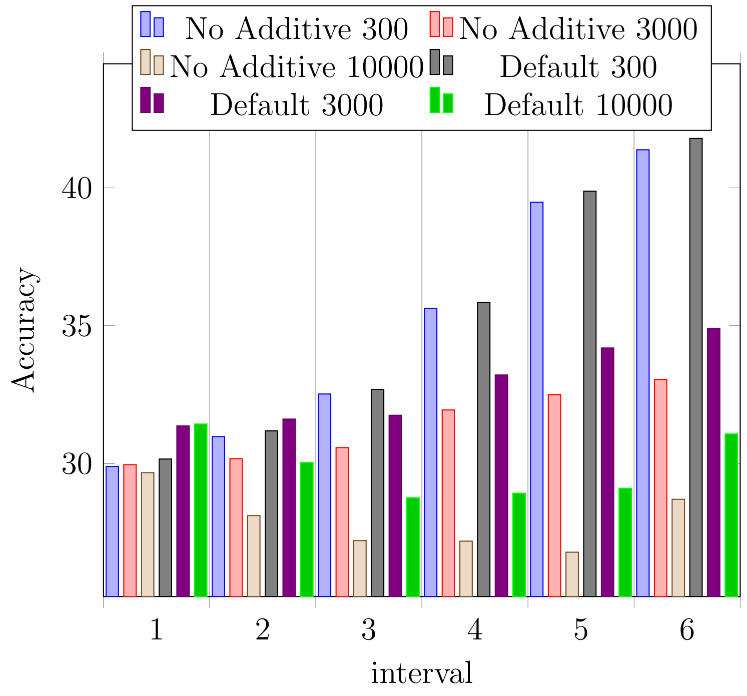
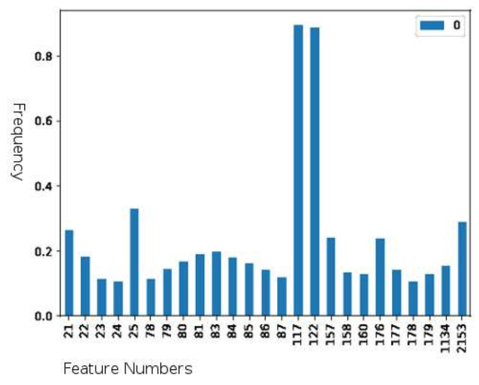
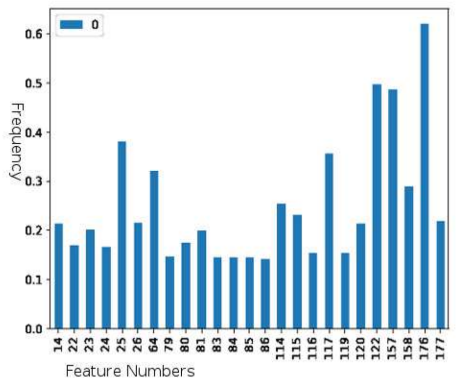


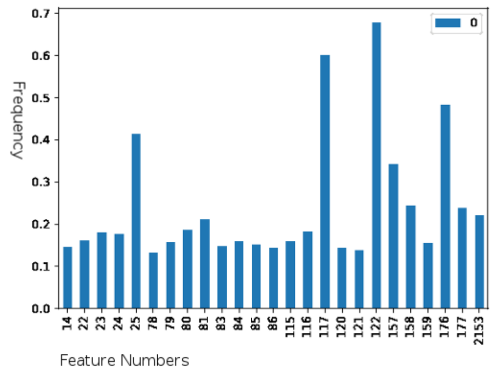
Figure 4.3: Move prediction accuracy per game phase for 300, 3000 ad 10000 simulation players. Each group has 50 moves.



(a) Feature frequencies of every legal move in master games



(b) Feature frequencies of all master moves in master games



(c) Features of master moves that have low number of simulations compared to the move played by Default Fuego using 1000 simulations.

Figure 4.4: Feature counts of baselines.

Even modern neural networks can be seen as a function that is built upon a complex set of features computed in its nodes. In order to understand the significance of different features, we use frequency of features, and we report the most frequent features for each experiment. We count the number of times each feature is present in master players' moves throughout the game records to identify frequency of features. We also record the same features over the moves generated by our computer-based players. Our goal is to gain insight on how players differ. In each experiment, we count the number of times each feature exists in the moves generated by one player. The result is a table of features with their significance for the move prediction task. For the meaning of each recurring feature number please refer to the appendix.

We need baselines to analyze the results obtained from our comparison. We have selected three such baselines. The first baseline is the frequency of features in all legal moves for every position in all the master games. The second one is the frequency of all features in master player's moves throughout all game records. The last baseline is the frequency of features in the master move which do not get any attention from our player. In order to determine these moves, we record the number of simulations allocated by the Default Fuego player for each master move in each game position. If the number of simulation for the master move is less than 1% of the move chosen by the Default Fuego, that move is marked as a low simulation master move and its features are recorded. Figure 4.4 shows the graphs for these baselines.

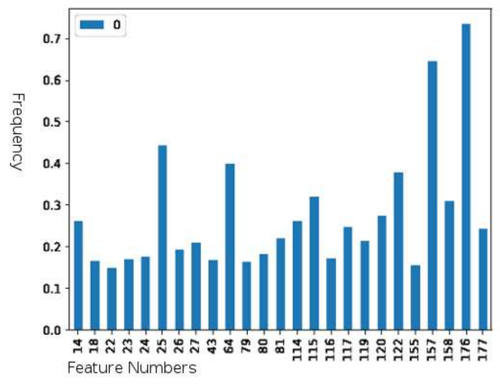
The two most prominent features in Figure 4.4a are 117 and 122. They represent a distance of 4 or more to the block of the last opponent stone and to the block of the last own stone respectively. Their frequency is more than 85% over all legal moves for each position. This is not surprising due to the size of the 19×19 board, and the distribution of legal moves in each position. The next two prominent features are 25 (moves on line 5 and upward) and 21 (moves on the first line). While moves on line 5 and upward cover 1.68 times the area of moves on the first line, they only happen 1.25 times more in the legal moves. Comparing the frequency of these two feature reveals that positions on the first line of the board remain empty longer than other points in professional games.

Figure 4.4b shows features of professional players moves. Feature 176 (distance 2 to closest opponent stone) is true for 62% of professional moves and feature 177 (distance 3 to closest opponent stone) in 22%. In total 85% of professional moves are in close proximity to opponent stones. Feature 157 and 158 (distance 2 and 3 to closest own stone) together cover almost 80% of professional moves, showing that professionals play close to their own stones as well.

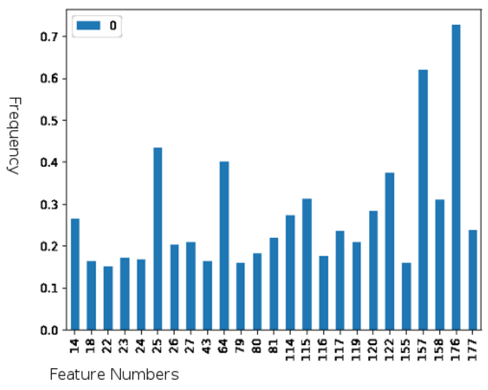
As in Figure 4.4a, in Figure 4.4c prominent features of master moves missed by Fuego are 122 and 117 with frequency of 68% and 60%. This shows that moves that usually get ignored by Fuego are non-local responses to the opponent, or "tenuki" moves that change the area of play.

4.4.1 Effect of Search and Master Move Prediction

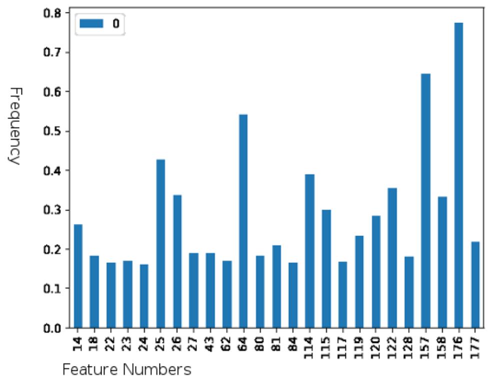
We study which master moves are a) found by search, and b) rejected by search. Figures 4.5a to 4.5c show feature frequencies for three players: Default MCTS-based Fuego, No Additive and No Knowledge. Each figure shows the frequency of features present in the moves predicted correctly by the player with 3000 simulations, while the same player with 100 simulations misses it. We also created statistics for the opposite case: moves that the player with



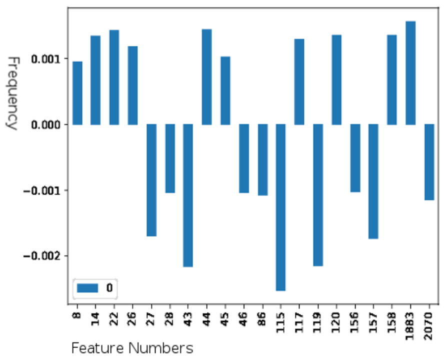
(a) Default Fuego.



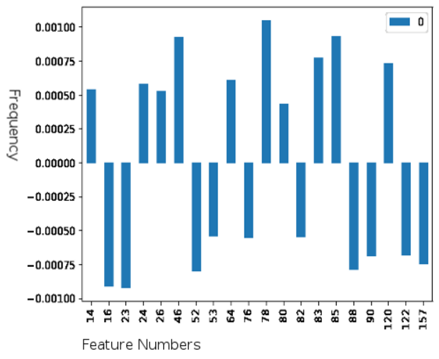
(b) No additive.



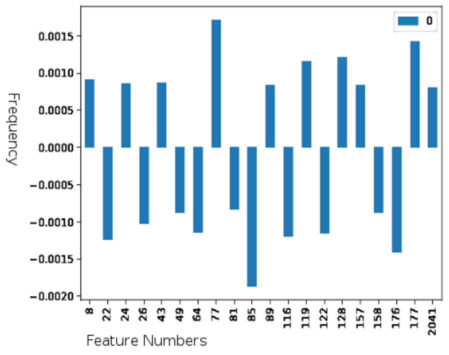
(c) No Knowledge.



(d) Difference between Figure 4.5a and when players are swapped.



(e) Difference between Figure 4.5b and when players are swapped.



(f) Difference between Figure 4.5c and when players are swapped.

Figure 4.5: Feature count for comparing the players. Features of moves predicted correctly by players with 3000 simulations but missed by players with 100 simulationis, and vice versa.

100 simulations predicts correctly but the player with 3000 simulations misses. We have not included these graphs in Section 4.4 and only show the difference graphs here. In order to understand the differences, we created graphs of difference between features frequency statistics shown in Figures 4.5a to 4.5c, and statistics of players with 100 simulations when they predict correctly and 3000 simulations fails. These results are reported in Figures 4.5d to 4.5f.

To discover if these differences in statistics of features in the players are random or consistent, we randomly selected 10 subsets of feature statistics from the feature frequency database used to make the difference graphs. We observed that frequencies in all the subsets where within 98.5% of the frequencies in the superset in top 50 features of the superset. This shows that any difference in feature frequency that is less than 1.5% can be ignored in features difference graphs due to randomness in the data.

As we can see in Figures 4.5d to 4.5f, all the differences have frequencies less than 1.5%, and they can be ignored. This shows that feature frequency for master moves predicted correctly by one player while the other missed it is not much different from the feature frequency when the other player predicts the master moves correctly and the first player misses it. This shows that we are not able to use features to find differences in mechanisms of master move prediction between the players. In order to find the differentiating factors between players, we also created graphs of difference between feature frequency on all the moves generated by each player. Those graphs are shown in Figures 4.6 to 4.8.

4.4.2 Impact of More Simulations

Figure 4.6 shows the difference in feature frequency of moves generated by default Fuego with 3000 and 100 simulations. The main difference is in features 117 and 122 which indicate changing the area of play, “tenuki”. Feature 25 (play on line number 5 and up) is another example of the impact of more simulations on the area of play. We saw that this is one of the prominent features of professional players moves. These results show that the player with more simulations can find centre and tenuki moves more often, and becomes more similar to how professional players play in these situations.

4.4.3 Impact of the Additive Term

Figures 4.7a and 4.7b show the differences between the default Fuego player and the No additive player with 3000 simulations. In Figure 4.7a, features 157 and 176 are for playing in distance of 2 to the closest own stone and opponent

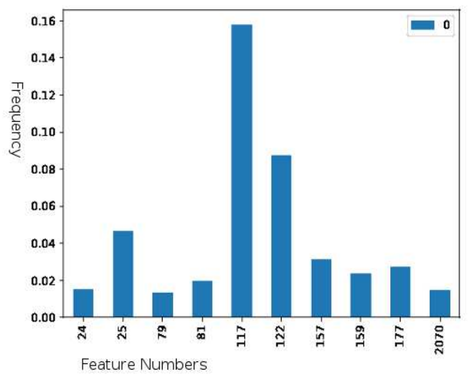
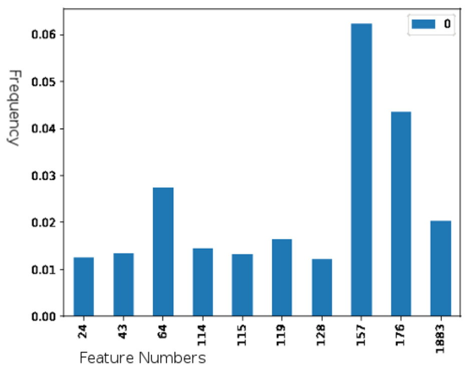
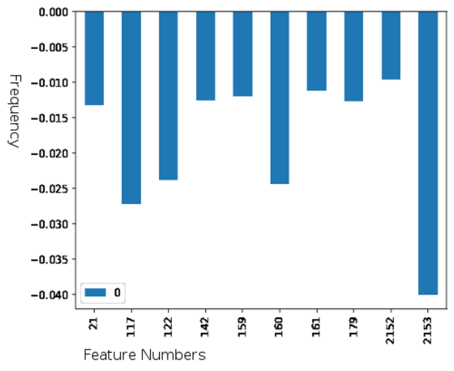


Figure 4.6: Top 10 differences between features count of default Fuego player with 3000 simulations and 100 simulations.



(a) Top 10 positive difference.



(b) Top 10 negative difference.

Figure 4.7: Difference between feature counts of default Fuego and No Additive player when both players use 3000 simulations.

colour respectively. They happen 6% and 4% more in the default Fuego player which benefits from the additive term. This shows that additive knowledge encourages playing close to previous stones. Feature 64 also happens 3% more in the player with additive term. This feature is for 3×3 patterns used in the simulations policy. This is an expected behaviour as the additive knowledge uses a diamond shape pattern to evaluate each move. Other features in Figure 4.7a have a very low frequency.

The No Additive player plays more often in empty areas of the board (feature 2153, 3×3 empty pattern), and far from all other stones, features 117, 122 and 160 (distance 5 to closest own stone).

4.4.4 Impact of Simple Feature Knowledge with Increasing Number of Simulations

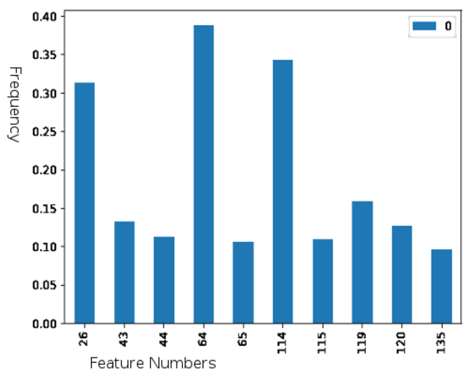
By comparing Figures 4.8a and 4.8b and Figures 4.8c and 4.8d we can understand the impact of simple feature knowledge. Features 26 (distance 2 to last opponent stone), 64 and 114 (distance 1 to block of last opponent stone) are more present in the player with knowledge, while in Figure 4.8b features 117 and 122 occur up to 42% more in the No Knowledge player. This shows that the No Knowledge player with low number of simulations plays more randomly in all areas of the board without any attention to the last own or opponent move, while the player with knowledge responds locally to those moves more often.

As the number of simulations grows, we still observe in Figures 4.8c and 4.8d the same difference in style of play from default Fuego and the No Knowledge player. This gap, however, narrows to half with consistency in relative frequency of features to each other. To some degree more simulations compensate for the lack of knowledge in the No Knowledge player, as we already observed in the move prediction task; however, more simulations are not able to completely close the gap.

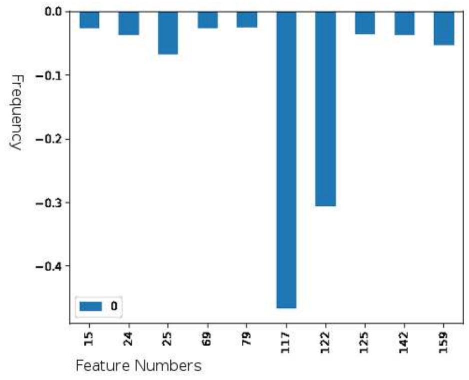
4.4.5 Features of Professional Moves

We ran another experiment to understand why some professional moves are ignored in Fuego. We compared the statistics of the default Fuego moves to the professional moves with low simulations in Figure 4.9. This helps to understand what kind of moves professional players make that Fuego does not consider, and how often those moves happen. In Figure 4.9a, features 114, 115, 119, 157, 176 are all for moves with distance of 1 or 2 to the own or opponent stones. This signifies the higher degree of locality of play in Fuego versus professional players. Also 3×3 simulation policy patterns (feature 64) occur 35% more in the default Fuego moves than in professional moves with low number of simulations, showing that many professional moves do not follow traditional 3×3 patterns as described in [27]. Looking at Figure 4.9b, features 117, 122, 159, 160, 161, 178, 179 are all for moves with distance of 4 or more to stones of either colour and feature 2153 is for the empty 3×3 square. These features happen up to 24% more in professional moves that received a very low number of simulations from Fuego. This shows that Fuego systematically likes to play locally, and moves with longer distance to the last own or opponent stone are not appealing to the program.

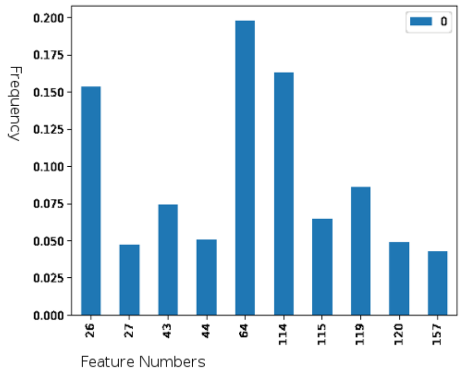
Feature differences in Figures 4.9c and 4.9d between the default Fuego moves



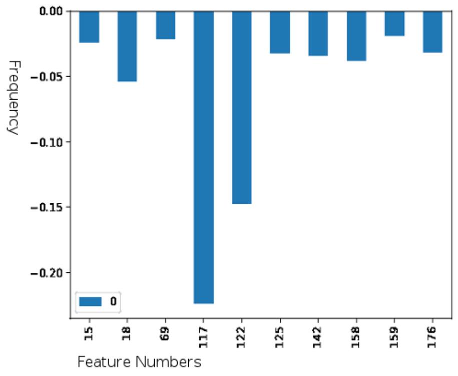
(a) Top 10 positive difference when both players use 100 simulations.



(b) Bottom 10 negative difference when both players use 100 simulations.

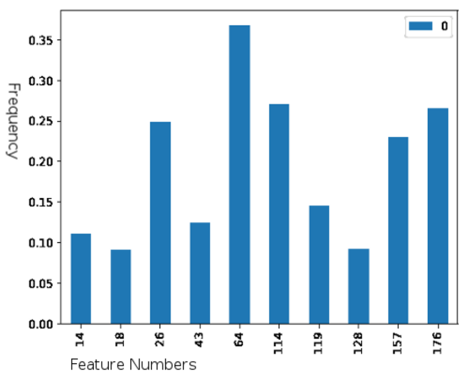


(c) Top 10 positive difference when both players use 3000 simulations.

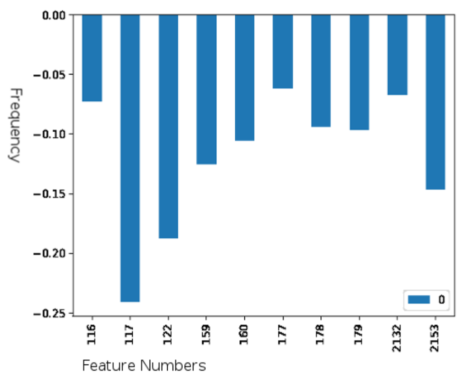


(d) Bottom 10 negative difference when both players use 3000 simulations.

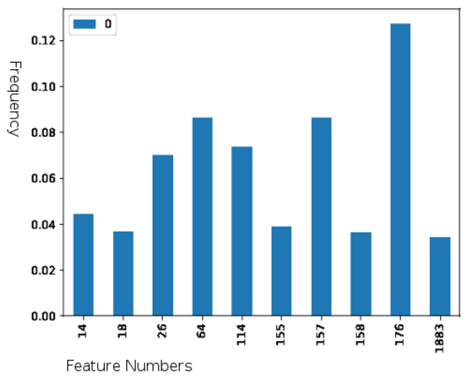
Figure 4.8: Difference between feature counts of default Fuego and No Knowledge player.



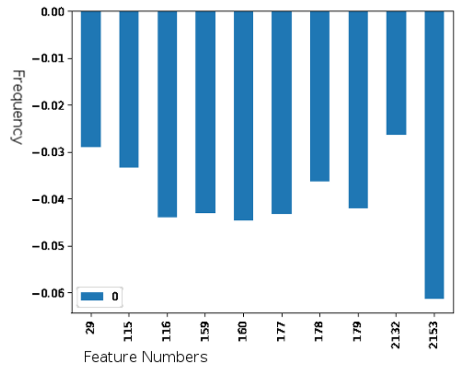
(a) Positive difference with low simulations professional moves



(b) Negative difference with low simulations professional moves



(c) Positive difference with all professional moves



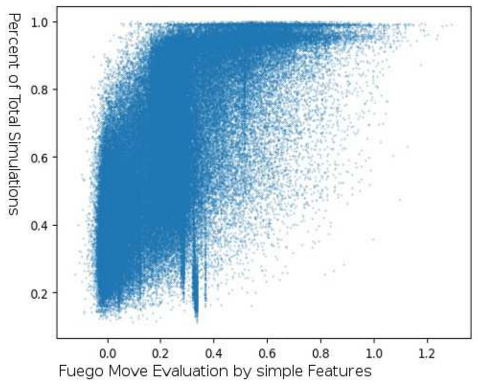
(d) Negative difference with all professional moves

Figure 4.9: Difference between feature counts of default Fuego with 3000 simulations moves and professional moves.

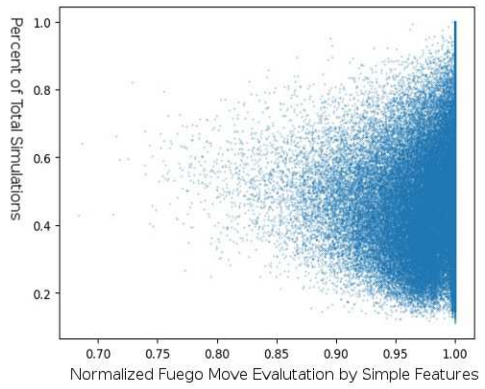
and all professional moves have similar feature differences to Figures 4.9a and 4.9b, but with different magnitude. First the magnitude of difference is much lower in Figures 4.9c and 4.9d. The other difference is that the most differentiating factor for the default Fuego player is that it plays 12% more in distance 2 of opponent stones (feature 176) than professional players. Professional moves still occure more in distance of 3 or more (features 116, 159, 160, 177, 178 and 179) to other stones, but the gap to Fuego is smaller.

4.4.6 Move Selection Analysis

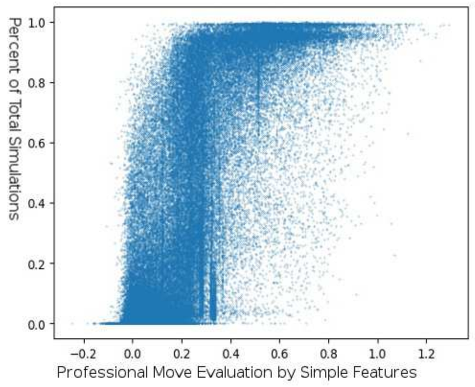
The next experiment helps us to understand under what circumstances a player can predict a professional move, while at other times it can not. We created an experiment to measure the number of simulations relative to the initial weight



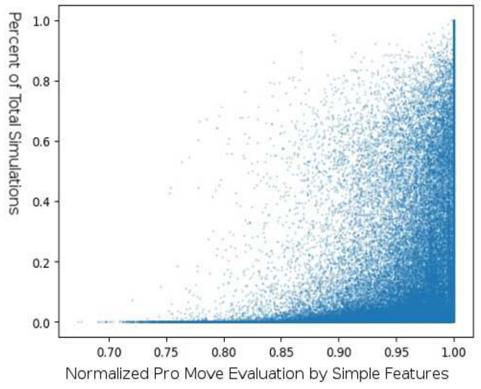
(a) Graph of selected move by default Fuego player given its initial weight



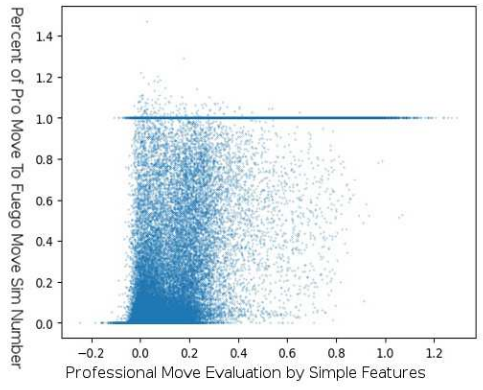
(b) Graph of selected move by default Fuego player given sigmoid of its initial weight divided by sigmoid of Max weight



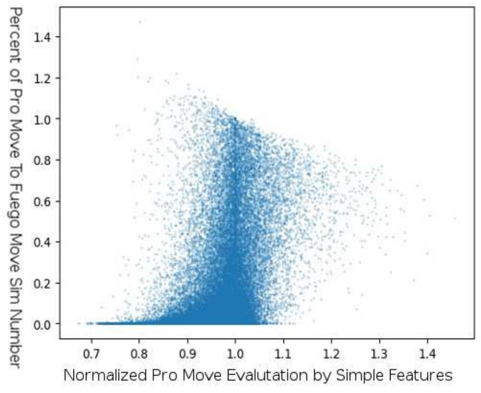
(c) Graph of played move by professional player given its initial weight



(d) Graph of selected move by professional Player given its sigmoid of its initial weight divided by sigmoid of Max weight



(e) Percent of Simulations relative to selected move divided by professional player's move given its initial weight



(f) Percent of Simulations relative to selected move divided by professional player's move given sigmoid of its initial weight divided by sigmoid of Max weight

Figure 4.10: Comparison between number of simulations for initial feature weight.

of a move. The results of this experiment are reported in Figure 4.10.

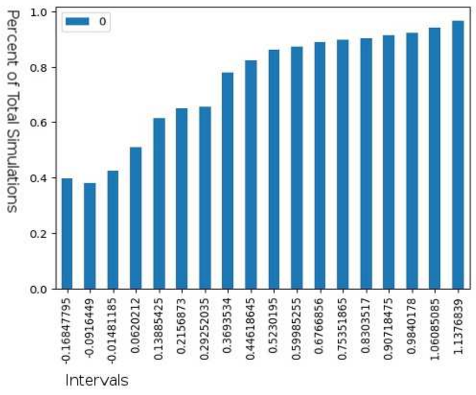
For the Y-axis of Figure 4.10 we measured two different cases. In the first case, we measured the number of simulations $sim_{s,a}$ for move a in state s relative to the total number of simulations for state s in the professional game: $\frac{sim_{s,a}}{\Sigma_i sim_{s,i}}$. For the second case, we measured the relative number of simulations $sim_{s,a}$ for move a in state s to the number of simulation $sim_{s,b}$ for move b in state s: $\frac{sim_{s,a}}{sim_{s,b}}$. The Y-axis of Figures 4.10a to 4.10d use the first case. For Figures 4.10a and 4.10b, move a is the move selected by default Fuego, and for Figures 4.10c and 4.10d it is the move selected by the professional player. The Y-axis of Figures 4.10e and 4.10f uses the second case. Move a is the move selected by the professional player and move b is the move selected by default Fuego.

The X-axis of Figure 4.10 has two different formats. In the first one, we use the initial weight $w_{s,a}$ of move a in state s of the professional game. For the second case, we compute the maximum weight $w_{s,max}$ for the state s, then compute the relative weight of move a to maximum weight $\frac{w_{s,a}}{w_{s,max}}$. Since the weight of a move can be negative, we normalize the relative value by sigmoid function $\frac{sig(w_a)}{sig(w_{max})}$. The X-axis of Figures 4.10a, 4.10c and 4.10e uses the first format. The X-axis of Figures 4.10b, 4.10d and 4.10f uses the second format. Move a is selected by default Fuego in Figures 4.10a and 4.10b, and by professional players in Figures 4.10c to 4.10f.

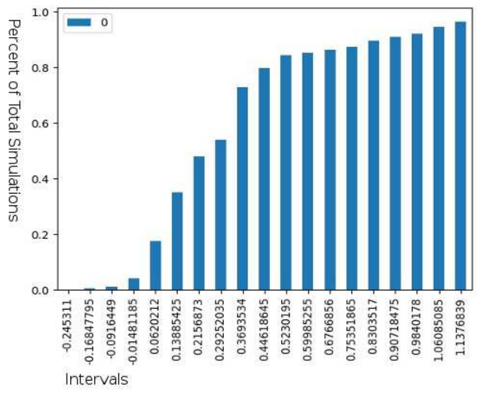
In order to understand the distribution of simulations, we created Figure 4.11a. It represents the relation between the weight of the feature for a move selected by default Fuego and the percent of simulations that move has received. Most of the moves selected by default Fuego have the majority of the simulations. Moves with higher initial weights receive almost 100% of simulations. Moves selected by Fuego have different ranges of weights from low to high. However, Figure 4.10b shows that even moves with low weights have weights close to the maximum weight of that position, and most of the times are the maximum weight.

Figure 4.10c shows that professional players moves most of the time either received the maximum number of simulations, or received close to zero. Moves that have an in-between number of simulations make up a smaller portion of professional moves. Figure 4.10d better illustrates this point. Figure 4.11b shows that for professional moves to get the attention of Fuego, they need to have higher evaluation by simple features.

We also compared the number of simulations for the professional moves and the moves selected by default Fuego. Figure 4.10e shows that very often the move played by professionals is the same as the Fuego move. However, if they differ, the chances of the professional move having a large number of simulations is low. Most of the time, it has less than 20% compared to Fuego's move.



(a) Graph of average simulations for selected move by default Fuego player given its initial weight



(b) Graph of average simulations for selected move by default professional player given its initial weight

Figure 4.11: Percent of average number of simulations for buckets of initial weights.

Figure 4.10f plots the relative number of simulations and the relative heuristic weight of the professional move to the move selected by default Fuego. The ratio of simulations drops sharply as the relative weight of the professional player's move decreases. For professional moves that have a ratio of less than 0.9, their number of simulations is near zero most of the time. There are some examples of professional moves with higher weight than the move selected by Fuego but with fewer simulations. These cases make less than 7% of total number of moves.

This experiment showed us the importance of simple feature initialization on the number of simulations a move receives. Fuego gives professional moves more simulations if they have high evaluation by simple features and on the other hand ignores them if the simple feature evaluation is low on those moves.

The move selected by Fuego does not need to have high evaluation as seen in the Figure 4.10a. It just needs to have an evaluation close to the maximum move evaluation of that position. This can be observed in Figure 4.10b. We also observed in Figures 4.10d and 4.10e how professional moves either receive close to the maximum number of simulations or close to zero.

Chapter 5

Conclusion and Future Work

In this thesis we investigated two popular evaluation methods: move prediction and playing strength, and how they relate to each other. We noticed that move prediction did not reveal important aspects of a player, and there remain many details that an aggregated move prediction percentage can not express. We showed that players with similar move prediction rate can have very different playing strengths. Sometimes, one completely overpowers the other player.

We used a playing strength experiment to understand the impact of the following concepts in MCTS: additive knowledge, simple feature knowledge, number of simulations, and playout policy. We noticed that the additive term has a very small impact on playing strength, which did not change with more simulations. Removing feature knowledge proved to have a deep negative impact on playing strength. The gap between the No Knowledge player and players with simple feature knowledge increases with more simulations.

We then dissected the move prediction rate into several intervals of a game in order to capture differences between the players at different game stages. We observed that as the number of simulations increases for a player, the move prediction rate drops as we get closer to the end of a game, due to "safe" move selection by stronger players in MCTS.

As the next step to find more differentiating factors between players, we examined feature frequencies in the move prediction task for different players. We were able to find features that differ remarkably between players, which can be used to define their behaviour. We also found relations between the evaluation of feature knowledge and the number of simulations a move receives.

For future work, we want to further the study by including neural network-based players and extending the experiments to understand the impacts of a neural network in detail. Another promising extension of this work is trying to understand neural networks in terms of both simple features and move prediction, in order to find an interpretation of their behaviour with known features of the Go game.

Bibliography

[1] Basic Rules of Go: https://senseis.xmp.net/?BasicRulesOfGo. Accessed Online on Oct 22, 2017.

[2] CGOS Server: http://cgos.boardspace.net.

[3] Elo Rating: https://senseis.xmp.net/?EloRating. Accessed Online on Aug 10, 2017.

[4] Fuego Source Code: http://fuego.sourceforge.net. SVN revision 2032, updated on Aug 16, 2016.

[5] Game of Go Picture: http://blogs.discovermagazine.com/crux/files/2016/01/shutterstock_-342026228.jpg. Accessed Online on Nov 12, 2017.

[6] Gnu Go: https://www.gnu.org/software/gnugo/devel.html. Accessed Online on Dec 7, 2017.

[7] GoGui Project: https://sourceforge.net/projects/gogui/. Accessed Online on Dec 14, 2016.

[8] HexPawn: http://www.chessvariants.com/small.dir/hexapawn.html. Accessed Online on Mar 20, 2018.

[9] MoGo Program: https://www.lri.fr/∼teytaud/mogo.html. Accessed Online on Nov 10, 2017.

[10] Neural Network Schematic: https://www.pyimagesearch.com/wp-content/uploads/2016/08/simple_neural_network_header.jpg?width=600. Accessed Online on Dec 4, 2017.

[11] Neuron Schematic: https://cdn-images-1.medium.com/max/1600/0*OHlzxsoDSEBXW0iI.jpg. Accessed Online on Dec 4, 2017.

[12] Professional Games: https://badukmovies.com/pro_games. Accessed Online on Nov 02, 2016.

[13] Using Minimax (with the full game tree) to implement the machine players to play TicTacToe in Computer with Python: https://sandipanweb.wordpress.com/2017/03/30/using-minimax-without-pruning-to-implement-the-machine-players-to-play-tictactoe-in-computer/. Accessed Online on Dec 4, 2017.

[14] Bruce Abramson. Expected-Outcome: A General Model of Static Evaluation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12(2):182–193, 1990.

[15] Daniel Acuna and Paul R Schrater. Structure learning in human sequential decision-making. In *Advances in Neural Information Processing Systems*, pages 1–8, 2009.

[16] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time Analysis of the Multi-armed Bandit Problem. *Mach. Learn.*, 47(2-3):235–256, May 2002.

[17] Pavol Bezák, Yury Rafailovich Nikitin, and Pavol Božek. Robotic Grasping System Using Convolutional Neural Networks. *American Journal of Mechanical Engineering*, 2(7):216–218, 2014.

[18] Cameron Browne, Edward Powley, Daniel Whitehouse, Simon Lucas, Peter I. Cowling, Stephen Tavener, Diego Perez, Spyridon Samothrakis, Simon Colton, and et al. A survey of Monte Carlo tree search methods. *IEEE transactions on computational intelligence and AI*, 2012.

[19] Murray Campbell, A Joseph Hoane, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[20] Guillaume M JB Chaslot, Mark HM Winands, H Jaap Van Den Herik, Jos WHM Uiterwijk, and Bruno Bouzy. Progressive strategies for Monte Carlo tree search. *New Mathematics and Natural Computation*, 4(03):343–357, 2008.

[21] Christopher Clark and Amos Storkey. Training deep convolutional neural networks to play Go. In *International Conference on Machine Learning*, pages 1766–1774, 2015.

[22] Rémi Coulom. Computing Elo Ratings of Move Patterns in the Game of Go. In H. Jaap van den Herik, Mark Winands, Jos Uiterwijk, and Maarten Schadd, editors, *Computer Games Workshop*, Amsterdam, Netherlands, June 2007.

[23] Markus Enzenberger, Martin Müller, Broderick Arneson, and Richard Segal. Fuego an open-source framework for board games and Go engine based on Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):259–270, 2010.

[24] Sumudu Fernando and Martin Müller. Analyzing Simulations in Monte Carlo Tree Search for the Game of Go. In *Computers and Games - 8th International Conference, CG 2013, Yokohama, Japan, August 13-15, 2013, Revised Selected Papers*, pages 72–83, 2013.

[25] Sylvain Gelly and David Silver. Combining online and offline knowledge in UCT. In *Proceedings of the 24th international conference on Machine learning*, pages 273–280. ACM, 2007.

[26] Sylvain Gelly and David Silver. Monte Carlo Tree Search and Rapid Action Value Estimation in Computer Go. *Artif. Intell.*, 175(11):1856–1875, July 2011.

[27] Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with patterns in Monte Carlo Go. 2006.

[28] Peijun Guo. One-shot decision theory. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 41(5):917–926, 2011.

[29] David P. Helmbold and Aleatha Parker-Wood. All-Moves-As-First Heuristics in Monte Carlo Go. In Hamid R. Arabnia, David de la Fuente, and Jos Angel Olivas, editors, *IC-AI*, pages 605–610. CSREA Press, 2009.

[30] Arthur Holshouser and Harold Reiter. *Quarto without the Twist*, 2003. http://math2.uncc.edu/∼hbreiter/Quarto.pdf.

[31] Shih-Chieh Huang, Rémi Coulom, and Shun-Shii Lin. Monte Carlo Simulation Balancing in Practice. In *Proceedings of the 7th International Conference on Computers and Games*, CG'10, pages 81–92, Berlin, Heidelberg, 2011. Springer-Verlag.

[32] David R Hunter and Kenneth Lange. A tutorial on MM algorithms. *The American Statistician*, 58(1):30–37, 2004.

[33] Yaakov Kerner. Learning strategies for explanation patterns: Basic game patterns with application to chess. *Case-Based Reasoning Research and Development*, pages 491–500, 1995.

[34] Levente Kocsis and Csaba Szepesvári. Bandit Based Monte Carlo Planning. In *Proceedings of the 17th European Conference on Machine Learning*, ECML'06, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag.

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[36] Chris J Maddison, Aja Huang, Ilya Sutskever, and David Silver. Move evaluation in Go using deep convolutional neural networks. *arXiv preprint arXiv:1412.6564*, 2014.

[37] Martin Wistuba and Lars Schmidt-Thieme. Move Prediction in Go – Modelling Feature Interactions Using Latent Factors. In *KI 2013: Advances in Artificial Intelligence*, pages 260–271. Springer Berlin Heidelberg, 2013.

[38] Robert J Meyer and Yong Shi. Sequential choice under ambiguity: Intuitive solutions to the armed-bandit problem. *Management Science*, 41(5):817–834, 1995.

[39] Liva Ralaivola, Lin Wu, and Pierre Baldi. SVM and pattern-enriched common fate graphs for the game of Go. In *ESANN*, volume 2005, pages 27–29, 2005.

[40] Steffen Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology*, 3(3):57, 2012.

[41] Christopher D. Rosin. Multi-armed Bandits with Episode Context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, March 2011.

[42] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[43] Jonathan Schaeffer, Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, and Steve Sutphen. Checkers is solved. *Science*, 317(5844):1518–1522, 2007.

[44] Nicol N Schraudolph, Peter Dayan, and Terrence J Sejnowski. Temporal difference learning of position evaluation in the game of Go. In *Advances in Neural Information Processing Systems*, pages 817–824, 1994.

[45] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[46] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[47] David Silver, Richard Sutton, and Martin Müller. Reinforcement learning of local shape in the game of Go. In *IJCAI*, 2007.

[48] David Silver and Gerald Tesauro. Monte Carlo Simulation Balancing. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 945–952, New York, NY, USA, 2009. ACM.

[49] David Stern, Ralf Herbrich, and Thore Graepel. Bayesian Pattern Ranking for Move Prediction in the Game of Go. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 873–880, New York, NY, USA, 2006. ACM.

[50] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA, 1st edition, 1998.

[51] Mark HM Winands, Yngvi Bjornsson, and Jahn-Takeshi Saito. Monte Carlo tree search in lines of action. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(4):239–250, 2010.

[52] Chenjun Xiao. Factorization Ranking Model for Fast Move Prediction in the Game of Go. Master's thesis, University of Alberta, 2016.

[53] Chenjun Xiao and Martin Müller. Factorization Ranking Model for Move Prediction in the Game of Go. In *AAAI*, pages 1359–1365, 2016.

Appendix A

Appendix

List of Features appeared in the moves selected by Fuego or professional players.

```
  "2": "FE_CAPTURE_ADJ_ATARI",
  "3": "FE_CAPTURE_RECAPTURE",
  "4": "FE_CAPTURE_PREVENT_CONNECTION",
  "5": "FE_CAPTURE_NOT_LADDER",
  "6": "FE_CAPTURE_LADDER",
  "7": "FE_CAPTURE_MULTIPLE",
  "8": "FE_EXTENSION_NOT_LADDER",
  "9": "FE_EXTENSION_LADDER",
  "10": "FE_TWO_LIB_SAVE_LADDER",
  "11": "FE_TWO_LIB_STILL_LADDER",
  "12": "FE_TWO_LIB_SELF_LADDER",
  "13": "FE_THREE_LIB_REDUCE_OWN_LIB",
  "14": "FE_THREE_LIB_REDUCE_OPP_LIB",
  "15": "FE_SELFATARI",
  "16": "FE_ATARI_LADDER",
  "17": "FE_ATARI_KO",
  "18": "FE_ATARI_OTHER",
  "19": "FE_DOUBLE_ATARI",
  "20": "FE_DOUBLE_ATARI_DEFEND",
  "21": "FE_LINE_1",
  "22": "FE_LINE_2",
  "23": "FE_LINE_3",
  "24": "FE_LINE_4",
  "25": "FE_LINE_5+",
  "26": "FE_DIST_PREV_2",
  "27": "FE_DIST_PREV_3",
  "28": "FE_DIST_PREV_4",
  "29": "FE_DIST_PREV_5",
  "30": "FE_DIST_PREV_6",
  "31": "FE_DIST_PREV_7",
  "32": "FE_DIST_PREV_8",
  "33": "FE_DIST_PREV_9",
  "34": "FE_DIST_PREV_10",
  "35": "FE_DIST_PREV_11",
  "36": "FE_DIST_PREV_12",
  "37": "FE_DIST_PREV_13",
  "38": "FE_DIST_PREV_14",
  "39": "FE_DIST_PREV_15",
  "40": "FE_DIST_PREV_16",
  "41": "FE_DIST_PREV_17",
  "42": "FE_DIST_PREV_OWN_0",
  "43": "FE_DIST_PREV_OWN_2",
  "44": "FE_DIST_PREV_OWN_3",
  "45": "FE_DIST_PREV_OWN_4",
  "46": "FE_DIST_PREV_OWN_5",
  "47": "FE_DIST_PREV_OWN_6",
  "48": "FE_DIST_PREV_OWN_7",
  "49": "FE_DIST_PREV_OWN_8",
  "50": "FE_DIST_PREV_OWN_9",
  "51": "FE_DIST_PREV_OWN_10",
  "52": "FE_DIST_PREV_OWN_11",
  "53": "FE_DIST_PREV_OWN_12",
  "54": "FE_DIST_PREV_OWN_13",
  "55": "FE_DIST_PREV_OWN_14",
  "56": "FE_DIST_PREV_OWN_15",
  "57": "FE_DIST_PREV_OWN_16",
  "58": "FE_DIST_PREV_OWN_17",
  "60": "FE_GOUCT_NAKADE",
  "61": "FE_GOUCT_ATARI_CAPTURE",
  "62": "FE_GOUCT_ATARI_DEFEND",
  "63": "FE_GOUCT_LOWLIB",
  "64": "FE_GOUCT_PATTERN",
  "65": "FE_GOUCT_CAPTURE",
  "66": "FE_GOUCT_RANDOM_PRUNED",
  "67": "FE_GOUCT_REPLACE_CAPTURE_FROM",
  "68": "FE_GOUCT_REPLACE_CAPTURE_TO",
  "69": "FE_GOUCT_SELFATARI_CORRECTION_FROM",
  "70": "FE_GOUCT_SELFATARI_CORRECTION_TO",
  "72": "FE_GOUCT_CLUMP_CORRECTION_TO",
  "73": "FE_POS_1",
  "74": "FE_POS_2",
  "75": "FE_POS_3",
  "76": "FE_POS_4",
  "77": "FE_POS_5",
  "78": "FE_POS_6",
  "79": "FE_POS_7",
  "80": "FE_POS_8",
  "81": "FE_POS_9",
  "82": "FE_POS_10",
  "83": "FE_GAME_PHASE_1",
  "84": "FE_GAME_PHASE_2",
  "85": "FE_GAME_PHASE_3",
  "86": "FE_GAME_PHASE_4",
  "87": "FE_GAME_PHASE_5",
  "88": "FE_GAME_PHASE_6",
  "89": "FE_GAME_PHASE_7",
  "90": "FE_GAME_PHASE_8",
  "91": "FE_GAME_PHASE_9",
  "92": "FE_GAME_PHASE_10",
  "93": "FE_GAME_PHASE_11",
  "94": "FE_GAME_PHASE_12",
  "95": "FE_SIDE_EXTENSION_3",
  "96": "FE_SIDE_EXTENSION_4",
  "97": "FE_SIDE_EXTENSION_5",
  "98": "FE_SIDE_EXTENSION_6",
  "99": "FE_SIDE_EXTENSION_7",
  "100": "FE_SIDE_EXTENSION_8",
  "101": "FE_SIDE_EXTENSION_9",
  "102": "FE_SIDE_EXTENSION_10",
  "103": "FE_SIDE_EXTENSION_11",
  "104": "FE_SIDE_EXTENSION_12",
  "105": "FE_SIDE_EXTENSION_13",
  "106": "FE_SIDE_EXTENSION_14",
  "107": "FE_SIDE_EXTENSION_15",
  "110": "FE_SIDE_EXTENSION_18",
  "113": "FE_CORNER_OPENING_MOVE",
  "114": "FE_CFG_DISTANCE_LAST_1",
  "115": "FE_CFG_DISTANCE_LAST_2",
  "116": "FE_CFG_DISTANCE_LAST_3",
  "117": "FE_CFG_DISTANCE_LAST_4_OR_MORE",
  "118": "FE_CFG_DISTANCE_LAST_OWN_0",
  "119": "FE_CFG_DISTANCE_LAST_OWN_1",
  "120": "FE_CFG_DISTANCE_LAST_OWN_2",
  "121": "FE_CFG_DISTANCE_LAST_OWN_3",
  "122": "FE_CFG_DISTANCE_LAST_OWN_4_OR_MORE",
  "123": "FE_TWO_LIB_NEW_SELF_LADDER",
  "124": "FE_OUR_PROTECTED_LIBERTY",
  "125": "FE_OPP_PROTECTED_LIBERTY",
  "126": "FE_OUR_CUT_WITH_KO",
  "127": "FE_OPP_CUT_WITH_KO",
  "128": "FE_SAVE_STONES_1",
  "129": "FE_SAVE_STONES_2",
  "130": "FE_SAVE_STONES_3",
  "131": "FE_SAVE_STONES_4_6",
  "132": "FE_SAVE_STONES_7_10",
  "133": "FE_SAVE_STONES_11_20",
  "134": "FE_SAVE_STONES_21_OR_MORE",
  "135": "FE_KILL_STONES_1",
  "136": "FE_KILL_STONES_2",
  "137": "FE_KILL_STONES_3",
  "138": "FE_KILL_STONES_4_6",
  "139": "FE_KILL_STONES_7_10",
  "140": "FE_KILL_STONES_11_20",
  "141": "FE_KILL_STONES_21_OR_MORE",
  "142": "FE_KILL_OWN_STONES_1",
  "143": "FE_KILL_OWN_STONES_2",
  "144": "FE_KILL_OWN_STONES_3",
  "145": "FE_KILL_OWN_STONES_4_6",
  "146": "FE_KILL_OWN_STONES_7_10",
  "147": "FE_KILL_OWN_STONES_11_20",
  "149": "FE_SNAPBACK",
  "155": "FE_CUT",
  "156": "FE_CONNECT",
  "157": "FE_DIST_CLOSEST_OWN_STONE_2",
  "158": "FE_DIST_CLOSEST_OWN_STONE_3",
  "159": "FE_DIST_CLOSEST_OWN_STONE_4",
  "160": "FE_DIST_CLOSEST_OWN_STONE_5",
  "161": "FE_DIST_CLOSEST_OWN_STONE_6",
  "162": "FE_DIST_CLOSEST_OWN_STONE_7",
  "163": "FE_DIST_CLOSEST_OWN_STONE_8",
  "164": "FE_DIST_CLOSEST_OWN_STONE_9",
  "165": "FE_DIST_CLOSEST_OWN_STONE_10",
  "166": "FE_DIST_CLOSEST_OWN_STONE_11",
  "167": "FE_DIST_CLOSEST_OWN_STONE_12",
  "168": "FE_DIST_CLOSEST_OWN_STONE_13",
  "169": "FE_DIST_CLOSEST_OWN_STONE_14",
  "170": "FE_DIST_CLOSEST_OWN_STONE_15",
  "171": "FE_DIST_CLOSEST_OWN_STONE_16",
  "172": "FE_DIST_CLOSEST_OWN_STONE_17",
  "173": "FE_DIST_CLOSEST_OWN_STONE_18",
  "174": "FE_DIST_CLOSEST_OWN_STONE_19",
  "175": "FE_DIST_CLOSEST_OWN_STONE_20_OR_MORE",
  "176": "FE_DIST_CLOSEST_OPP_STONE_2",
  "177": "FE_DIST_CLOSEST_OPP_STONE_3",
  "178": "FE_DIST_CLOSEST_OPP_STONE_4",
  "179": "FE_DIST_CLOSEST_OPP_STONE_5",
  "180": "FE_DIST_CLOSEST_OPP_STONE_6",
  "181": "FE_DIST_CLOSEST_OPP_STONE_7",
  "182": "FE_DIST_CLOSEST_OPP_STONE_8",
  "183": "FE_DIST_CLOSEST_OPP_STONE_9",
  "184": "FE_DIST_CLOSEST_OPP_STONE_10",
  "185": "FE_DIST_CLOSEST_OPP_STONE_11",
  "186": "FE_DIST_CLOSEST_OPP_STONE_12",
  "187": "FE_DIST_CLOSEST_OPP_STONE_13",
  "188": "FE_DIST_CLOSEST_OPP_STONE_14",
  "189": "FE_DIST_CLOSEST_OPP_STONE_15",
  "190": "FE_DIST_CLOSEST_OPP_STONE_16",
  "191": "FE_DIST_CLOSEST_OPP_STONE_17",
  "192": "FE_DIST_CLOSEST_OPP_STONE_18",
  "193": "FE_DIST_CLOSEST_OPP_STONE_19",
  "194": "FE_DIST_CLOSEST_OPP_STONE_20_OR_MORE",
  "1001": "\nWEB\nBBB\n",
  "1003": "\nBEB\nWBB\n",
  "1004": "\nWEB\nWBB\n",
  "1005": "\nEEB\nWBB\n",
  "1009": "\nBEB\nBWB\n",
  "1010": "\nWEB\nBWB\n",
  "1011": "\nEEB\nBWB\n",
  "1012": "\nBEB\nWWB\n",
  "1013": "\nWEB\nWWB\n",
  "1014": "\nEEB\nWWB\n",
  "1015": "\nBEB\nEWB\n",
  "1016": "\nWEB\nEWB\n",
  "1017": "\nEEB\nEWB\n",
  "1019": "\nWEB\nBEB\n",
  "1020": "\nEEB\nBEB\n",
  "1021": "\nBEB\nWEB\n",
  "1027": "\nWEB\nBBW\n",
  "1028": "\nEEB\nBBW\n",
  "1029": "\nBEB\nWBW\n",
  "1030": "\nWEB\nWBW\n",
  "1031": "\nEEB\nWBW\n",
  "1032": "\nBEB\nEBW\n",
  "1033": "\nWEB\nEBW\n",
  "1034": "\nEEB\nEBW\n",
  "1035": "\nWEB\nBWW\n",
  "1036": "\nEEB\nBWW\n",
  "1037": "\nBEB\nWWW\n",
  "1038": "\nWEB\nWWW\n",
  "1039": "\nEEB\nWWW\n",
  "1040": "\nBEB\nEWW\n",
  "1041": "\nWEB\nEWW\n",
  "1042": "\nEEB\nEWW\n",
  "1044": "\nEEB\nBEW\n",
  "1045": "\nBEB\nWEW\n",
  "1046": "\nWEB\nWEW\n",
  "1047": "\nEEB\nWEW\n",
  "1048": "\nBEB\nEEW\n",
  "1049": "\nWEB\nEEW\n",
  "1050": "\nEEB\nEEW\n",
  "1051": "\nWEB\nBBE\n",
  "1053": "\nWEB\nWBE\n",
  "1054": "\nEEB\nWBE\n",
  "1056": "\nWEB\nEBE\n",
  "1058": "\nWEB\nBWE\n",
  "1059": "\nEEB\nBWE\n",
  "1060": "\nWEB\nWWE\n",
  "1061": "\nEEB\nWWE\n",
  "1062": "\nBEB\nEWE\n",
  "1063": "\nWEB\nEWE\n",
  "1064": "\nEEB\nEWE\n",
  "1065": "\nWEB\nBEE\n",
  "1066": "\nEEB\nBEE\n",
  "1070": "\nWEB\nEEE\n",
  "1071": "\nEEB\nEEE\n",
  "1072": "\nWEW\nBBB\n",
  "1073": "\nEEW\nBBB\n",
  "1074": "\nWEW\nWBB\n",
  "1075": "\nEEW\nWBB\n",
  "1076": "\nWEW\nEBB\n",
  "1077": "\nEEW\nEBB\n",
  "1078": "\nWEW\nBWB\n",
  "1079": "\nEEW\nBWB\n",
  "1080": "\nWEW\nWWB\n",
  "1081": "\nEEW\nWWB\n",
  "1082": "\nWEW\nEWB\n",
  "1083": "\nEEW\nEWB\n",
  "1085": "\nEEW\nBEB\n",
  "1086": "\nWEW\nWEB\n",
  "1087": "\nEEW\nWEB\n",
  "1088": "\nWEW\nEEB\n",
  "1089": "\nEEW\nEEB\n",
  "1090": "\nEEW\nBBW\n",
  "1091": "\nWEW\nWBW\n",
  "1092": "\nEEW\nWBW\n",
  "1093": "\nWEW\nEBW\n",
  "1094": "\nEEW\nEBW\n",
  "1095": "\nEEW\nBWW\n",
  "1100": "\nEEW\nBEW\n",
  "1101": "\nWEW\nWEW\n",
  "1102": "\nEEW\nWEW\n",
  "1103": "\nWEW\nEEW\n",
  "1105": "\nEEW\nBBE\n",
  "1106": "\nEEW\nWBE\n",
  "1108": "\nEEW\nEBE\n",
  "1109": "\nEEW\nBWE\n",
  "1112": "\nEEW\nEWE\n",
  "1113": "\nEEW\nBEE\n",
  "1114": "\nEEW\nWEE\n",
  "1115": "\nWEW\nEEE\n",
  "1116": "\nEEW\nEEE\n",
  "1117": "\nEEE\nBBB\n",
  "1118": "\nEEE\nWBB\n",
  "1119": "\nEEE\nEBB\n",
  "1120": "\nEEE\nBWB\n",
  "1121": "\nEEE\nWWB\n",
  "1122": "\nEEE\nEWB\n",
  "1123": "\nEEE\nBEB\n",
  "1124": "\nEEE\nWEB\n",
  "1125": "\nEEE\nEEB\n",
  "1126": "\nEEE\nWBW\n",
  "1127": "\nEEE\nEBW\n",
  "1128": "\nEEE\nWWW\n",
  "1129": "\nEEE\nEWW\n",
  "1130": "\nEEE\nWEW\n",
  "1131": "\nEEE\nEEW\n",
  "1133": "\nEEE\nEWE\n",
  "1134": "\nEEE\nEEE\n",
  "1203": "\nBWB\nBEB\nBBB\n",
  "1204": "\nBWW\nBEB\nBBB\n",
  "1209": "\nWBW\nBEB\nBBB\n",
  "1211": "\nWWW\nBEB\nBBB\n",
  "1213": "\nWEW\nBEB\nBBB\n",
  "1218": "\nBWB\nBEW\nBBB\n",
  "1219": "\nBWW\nBEW\nBBB\n",
  "1220": "\nBWE\nBEW\nBBB\n",
  "1221": "\nBEB\nBEW\nBBB\n",
  "1222": "\nBEW\nBEW\nBBB\n",
  "1224": "\nWBB\nBEW\nBBB\n",
  "1225": "\nWBW\nBEW\nBBB\n",
  "1226": "\nWBE\nBEW\nBBB\n",
  "1227": "\nWWB\nBEW\nBBB\n",
  "1228": "\nWWW\nBEW\nBBB\n",
  "1229": "\nWWE\nBEW\nBBB\n",
  "1230": "\nWEB\nBEW\nBBB\n",
  "1231": "\nWEW\nBEW\nBBB\n",
  "1232": "\nWEE\nBEW\nBBB\n",
  "1233": "\nEBB\nBEW\nBBB\n",
  "1234": "\nEBW\nBEW\nBBB\n",
  "1235": "\nEBE\nBEW\nBBB\n",
  "1236": "\nEWB\nBEW\nBBB\n",
  "1238": "\nEWE\nBEW\nBBB\n",
  "1239": "\nEEB\nBEW\nBBB\n",
  "1241": "\nEEE\nBEW\nBBB\n",
  "1242": "\nBEB\nBEE\nBBB\n",
  "1246": "\nWBW\nBEE\nBBB\n",
  "1248": "\nWWB\nBEE\nBBB\n",
  "1249": "\nWWW\nBEE\nBBB\n",
  "1250": "\nWWE\nBEE\nBBB\n",
  "1252": "\nWEW\nBEE\nBBB\n",
  "1257": "\nEWB\nBEE\nBBB\n",
  "1263": "\nBBB\nWEW\nBBB\n",
  "1264": "\nBBW\nWEW\nBBB\n",
  "1265": "\nBBE\nWEW\nBBB\n",
  "1267": "\nBWW\nWEW\nBBB\n",
  "1268": "\nBWE\nWEW\nBBB\n",
  "1269": "\nBEB\nWEW\nBBB\n",
  "1270": "\nBEW\nWEW\nBBB\n",
  "1271": "\nBEE\nWEW\nBBB\n",
  "1272": "\nWBW\nWEW\nBBB\n",
  "1273": "\nWBE\nWEW\nBBB\n",
  "1274": "\nWWW\nWEW\nBBB\n",
  "1275": "\nWWE\nWEW\nBBB\n",
  "1276": "\nWEW\nWEW\nBBB\n",
  "1277": "\nWEE\nWEW\nBBB\n",
  "1278": "\nEBE\nWEW\nBBB\n",
  "1279": "\nEWE\nWEW\nBBB\n",
  "1280": "\nEEE\nWEW\nBBB\n",
  "1281": "\nBBB\nWEE\nBBB\n",
  "1282": "\nBBW\nWEE\nBBB\n",
  "1283": "\nBBE\nWEE\nBBB\n",
  "1284": "\nBWB\nWEE\nBBB\n",
  "1285": "\nBWW\nWEE\nBBB\n",
  "1286": "\nBWE\nWEE\nBBB\n",
  "1287": "\nBEB\nWEE\nBBB\n",
  "1288": "\nBEW\nWEE\nBBB\n",
  "1289": "\nBEE\nWEE\nBBB\n",
  "1290": "\nWBB\nWEE\nBBB\n",
  "1291": "\nWBW\nWEE\nBBB\n",
  "1292": "\nWBE\nWEE\nBBB\n",
  "1293": "\nWWB\nWEE\nBBB\n",
  "1294": "\nWWW\nWEE\nBBB\n",
  "1296": "\nWEB\nWEE\nBBB\n",
  "1297": "\nWEW\nWEE\nBBB\n",
  "1298": "\nWEE\nWEE\nBBB\n",
  "1299": "\nEBB\nWEE\nBBB\n",
  "1300": "\nEBW\nWEE\nBBB\n",
  "1301": "\nEBE\nWEE\nBBB\n",
  "1302": "\nEWB\nWEE\nBBB\n",
  "1303": "\nEWW\nWEE\nBBB\n",
  "1304": "\nEWE\nWEE\nBBB\n",
  "1305": "\nEEB\nWEE\nBBB\n",
  "1306": "\nEEW\nWEE\nBBB\n",
  "1307": "\nEEE\nWEE\nBBB\n",
  "1308": "\nBBB\nEEE\nBBB\n",
  "1311": "\nBWB\nEEE\nBBB\n",
  "1312": "\nBWW\nEEE\nBBB\n",
  "1313": "\nBWE\nEEE\nBBB\n",
  "1314": "\nBEB\nEEE\nBBB\n",
  "1315": "\nBEW\nEEE\nBBB\n",
  "1316": "\nBEE\nEEE\nBBB\n",
  "1317": "\nWBW\nEEE\nBBB\n",
  "1318": "\nWBE\nEEE\nBBB\n",
  "1321": "\nWEW\nEEE\nBBB\n",
  "1322": "\nWEE\nEEE\nBBB\n",
  "1324": "\nEWE\nEEE\nBBB\n",
  "1325": "\nEEE\nEEE\nBBB\n",
  "1326": "\nWBB\nBEB\nBBW\n",
  "1327": "\nWBW\nBEB\nBBW\n",
  "1328": "\nWBE\nBEB\nBBW\n",
  "1329": "\nWWB\nBEB\nBBW\n",
  "1330": "\nWWW\nBEB\nBBW\n",
  "1331": "\nWWE\nBEB\nBBW\n",
  "1332": "\nWEB\nBEB\nBBW\n",
  "1333": "\nWEW\nBEB\nBBW\n",
  "1334": "\nWEE\nBEB\nBBW\n",
  "1336": "\nEBW\nBEB\nBBW\n",
  "1338": "\nEWB\nBEB\nBBW\n",
  "1339": "\nEWW\nBEB\nBBW\n",
  "1340": "\nEWE\nBEB\nBBW\n",
  "1341": "\nEEB\nBEB\nBBW\n",
  "1342": "\nEEW\nBEB\nBBW\n",
  "1344": "\nWWB\nBEW\nBBW\n",
  "1345": "\nWWW\nBEW\nBBW\n",
  "1346": "\nWWE\nBEW\nBBW\n",
  "1347": "\nWEB\nBEW\nBBW\n",
  "1348": "\nWEW\nBEW\nBBW\n",
  "1349": "\nWEE\nBEW\nBBW\n",
  "1350": "\nEBB\nBEW\nBBW\n",
  "1351": "\nEBW\nBEW\nBBW\n",
  "1352": "\nEBE\nBEW\nBBW\n",
  "1353": "\nEWB\nBEW\nBBW\n",
  "1355": "\nEWE\nBEW\nBBW\n",
  "1356": "\nEEB\nBEW\nBBW\n",
  "1358": "\nEEE\nBEW\nBBW\n",
  "1360": "\nWEW\nBEE\nBBW\n",
  "1361": "\nWEE\nBEE\nBBW\n",
  "1363": "\nEBW\nBEE\nBBW\n",
  "1365": "\nEWB\nBEE\nBBW\n",
  "1366": "\nEWW\nBEE\nBBW\n",
  "1367": "\nEWE\nBEE\nBBW\n",
  "1369": "\nEEW\nBEE\nBBW\n",
  "1371": "\nBBW\nWEB\nBBW\n",
  "1372": "\nBBE\nWEB\nBBW\n",
  "1373": "\nBWB\nWEB\nBBW\n",
  "1374": "\nBWW\nWEB\nBBW\n",
  "1375": "\nBWE\nWEB\nBBW\n",
  "1376": "\nBEB\nWEB\nBBW\n",
  "1377": "\nBEW\nWEB\nBBW\n",
  "1378": "\nBEE\nWEB\nBBW\n",
  "1379": "\nWBW\nWEB\nBBW\n",
  "1380": "\nWBE\nWEB\nBBW\n",
  "1381": "\nWWB\nWEB\nBBW\n",
  "1382": "\nWWW\nWEB\nBBW\n",
  "1383": "\nWWE\nWEB\nBBW\n",
  "1384": "\nWEB\nWEB\nBBW\n",
  "1385": "\nWEW\nWEB\nBBW\n",
  "1386": "\nWEE\nWEB\nBBW\n",
  "1387": "\nEBW\nWEB\nBBW\n",
  "1388": "\nEBE\nWEB\nBBW\n",
  "1389": "\nEWB\nWEB\nBBW\n",
  "1390": "\nEWW\nWEB\nBBW\n",
  "1391": "\nEWE\nWEB\nBBW\n",
  "1392": "\nEEB\nWEB\nBBW\n",
  "1393": "\nEEW\nWEB\nBBW\n",
  "1394": "\nEEE\nWEB\nBBW\n",
  "1395": "\nBBW\nWEW\nBBW\n",
  "1396": "\nBBE\nWEW\nBBW\n",
  "1397": "\nBWB\nWEW\nBBW\n",
  "1398": "\nBWW\nWEW\nBBW\n",
  "1399": "\nBWE\nWEW\nBBW\n",
  "1400": "\nBEB\nWEW\nBBW\n",
  "1401": "\nBEW\nWEW\nBBW\n",
  "1402": "\nBEE\nWEW\nBBW\n",
  "1403": "\nWBB\nWEW\nBBW\n",
  "1404": "\nWBW\nWEW\nBBW\n",
  "1405": "\nWBE\nWEW\nBBW\n",
  "1406": "\nWWB\nWEW\nBBW\n",
  "1407": "\nWWW\nWEW\nBBW\n",
  "1408": "\nWWE\nWEW\nBBW\n",
  "1409": "\nWEB\nWEW\nBBW\n",
  "1410": "\nWEW\nWEW\nBBW\n",
  "1411": "\nWEE\nWEW\nBBW\n",
  "1412": "\nEBB\nWEW\nBBW\n",
  "1413": "\nEBW\nWEW\nBBW\n",
  "1414": "\nEBE\nWEW\nBBW\n",
  "1415": "\nEWB\nWEW\nBBW\n",
  "1416": "\nEWW\nWEW\nBBW\n",
  "1417": "\nEWE\nWEW\nBBW\n",
  "1418": "\nEEB\nWEW\nBBW\n",
  "1419": "\nEEW\nWEW\nBBW\n",
  "1420": "\nEEE\nWEW\nBBW\n",
  "1422": "\nBBE\nWEE\nBBW\n",
  "1423": "\nBWB\nWEE\nBBW\n",
  "1424": "\nBWW\nWEE\nBBW\n",
  "1425": "\nBWE\nWEE\nBBW\n",
  "1426": "\nBEB\nWEE\nBBW\n",
  "1427": "\nBEW\nWEE\nBBW\n",
  "1428": "\nBEE\nWEE\nBBW\n",
  "1429": "\nWBB\nWEE\nBBW\n",
  "1430": "\nWBW\nWEE\nBBW\n",
  "1431": "\nWBE\nWEE\nBBW\n",
  "1432": "\nWWB\nWEE\nBBW\n",
  "1433": "\nWWW\nWEE\nBBW\n",
  "1434": "\nWWE\nWEE\nBBW\n",
  "1435": "\nWEB\nWEE\nBBW\n",
  "1436": "\nWEW\nWEE\nBBW\n",
  "1437": "\nWEE\nWEE\nBBW\n",
  "1438": "\nEBB\nWEE\nBBW\n",
  "1439": "\nEBW\nWEE\nBBW\n",
  "1440": "\nEBE\nWEE\nBBW\n",
  "1441": "\nEWB\nWEE\nBBW\n",
  "1442": "\nEWW\nWEE\nBBW\n",
  "1443": "\nEWE\nWEE\nBBW\n",
  "1444": "\nEEB\nWEE\nBBW\n",
  "1445": "\nEEW\nWEE\nBBW\n",
  "1446": "\nEEE\nWEE\nBBW\n",
  "1447": "\nBBW\nEEB\nBBW\n",
  "1449": "\nBWW\nEEB\nBBW\n",
  "1450": "\nBWE\nEEB\nBBW\n",
  "1451": "\nBEB\nEEB\nBBW\n",
  "1452": "\nBEW\nEEB\nBBW\n",
  "1453": "\nBEE\nEEB\nBBW\n",
  "1454": "\nWBW\nEEB\nBBW\n",
  "1455": "\nWBE\nEEB\nBBW\n",
  "1456": "\nWWW\nEEB\nBBW\n",
  "1457": "\nWWE\nEEB\nBBW\n",
  "1458": "\nWEB\nEEB\nBBW\n",
  "1459": "\nWEW\nEEB\nBBW\n",
  "1460": "\nWEE\nEEB\nBBW\n",
  "1461": "\nEBW\nEEB\nBBW\n",
  "1463": "\nEWW\nEEB\nBBW\n",
  "1464": "\nEWE\nEEB\nBBW\n",
  "1465": "\nEEB\nEEB\nBBW\n",
  "1466": "\nEEW\nEEB\nBBW\n",
  "1467": "\nEEE\nEEB\nBBW\n",
  "1468": "\nBBW\nEEW\nBBW\n",
  "1469": "\nBBE\nEEW\nBBW\n",
  "1470": "\nBWB\nEEW\nBBW\n",
  "1471": "\nBWW\nEEW\nBBW\n",
  "1472": "\nBWE\nEEW\nBBW\n",
  "1473": "\nBEB\nEEW\nBBW\n",
  "1474": "\nBEW\nEEW\nBBW\n",
  "1475": "\nBEE\nEEW\nBBW\n",
  "1476": "\nWBW\nEEW\nBBW\n",
  "1477": "\nWBE\nEEW\nBBW\n",
  "1478": "\nWWB\nEEW\nBBW\n",
  "1480": "\nWWE\nEEW\nBBW\n",
  "1481": "\nWEB\nEEW\nBBW\n",
  "1482": "\nWEW\nEEW\nBBW\n",
  "1483": "\nWEE\nEEW\nBBW\n",
  "1484": "\nEBB\nEEW\nBBW\n",
  "1485": "\nEBW\nEEW\nBBW\n",
  "1486": "\nEBE\nEEW\nBBW\n",
  "1487": "\nEWB\nEEW\nBBW\n",
  "1489": "\nEWE\nEEW\nBBW\n",
  "1490": "\nEEB\nEEW\nBBW\n",
  "1491": "\nEEW\nEEW\nBBW\n",
  "1492": "\nEEE\nEEW\nBBW\n",
  "1493": "\nBBW\nEEE\nBBW\n",
  "1494": "\nBBE\nEEE\nBBW\n",
  "1495": "\nBWB\nEEE\nBBW\n",
  "1496": "\nBWW\nEEE\nBBW\n",
  "1497": "\nBWE\nEEE\nBBW\n",
  "1499": "\nBEW\nEEE\nBBW\n",
  "1500": "\nBEE\nEEE\nBBW\n",
  "1501": "\nWBB\nEEE\nBBW\n",
  "1502": "\nWBW\nEEE\nBBW\n",
  "1503": "\nWBE\nEEE\nBBW\n",
  "1504": "\nWWB\nEEE\nBBW\n",
  "1505": "\nWWW\nEEE\nBBW\n",
  "1506": "\nWWE\nEEE\nBBW\n",
  "1507": "\nWEB\nEEE\nBBW\n",
  "1508": "\nWEW\nEEE\nBBW\n",
  "1509": "\nWEE\nEEE\nBBW\n",
  "1510": "\nEBB\nEEE\nBBW\n",
  "1511": "\nEBW\nEEE\nBBW\n",
  "1512": "\nEBE\nEEE\nBBW\n",
  "1513": "\nEWB\nEEE\nBBW\n",
  "1514": "\nEWW\nEEE\nBBW\n",
  "1515": "\nEWE\nEEE\nBBW\n",
  "1516": "\nEEB\nEEE\nBBW\n",
  "1517": "\nEEW\nEEE\nBBW\n",
  "1518": "\nEEE\nEEE\nBBW\n",
  "1522": "\nEWB\nBEB\nBBE\n",
  "1523": "\nEWW\nBEB\nBBE\n",
  "1524": "\nEWE\nBEB\nBBE\n",
  "1528": "\nEWB\nBEW\nBBE\n",
  "1531": "\nEEB\nBEW\nBBE\n",
  "1537": "\nBBE\nWEB\nBBE\n",
  "1538": "\nBWB\nWEB\nBBE\n",
  "1539": "\nBWW\nWEB\nBBE\n",
  "1540": "\nBWE\nWEB\nBBE\n",
  "1541": "\nBEB\nWEB\nBBE\n",
  "1542": "\nBEW\nWEB\nBBE\n",
  "1543": "\nBEE\nWEB\nBBE\n",
  "1544": "\nWBW\nWEB\nBBE\n",
  "1545": "\nWBE\nWEB\nBBE\n",
  "1546": "\nWWB\nWEB\nBBE\n",
  "1547": "\nWWW\nWEB\nBBE\n",
  "1548": "\nWWE\nWEB\nBBE\n",
  "1549": "\nWEB\nWEB\nBBE\n",
  "1550": "\nWEW\nWEB\nBBE\n",
  "1551": "\nWEE\nWEB\nBBE\n",
  "1552": "\nEBW\nWEB\nBBE\n",
  "1553": "\nEBE\nWEB\nBBE\n",
  "1554": "\nEWB\nWEB\nBBE\n",
  "1555": "\nEWW\nWEB\nBBE\n",
  "1556": "\nEWE\nWEB\nBBE\n",
  "1557": "\nEEB\nWEB\nBBE\n",
  "1558": "\nEEW\nWEB\nBBE\n",
  "1559": "\nEEE\nWEB\nBBE\n",
  "1560": "\nBBE\nWEW\nBBE\n",
  "1561": "\nBWB\nWEW\nBBE\n",
  "1562": "\nBWW\nWEW\nBBE\n",
  "1563": "\nBWE\nWEW\nBBE\n",
  "1564": "\nBEB\nWEW\nBBE\n",
  "1565": "\nBEW\nWEW\nBBE\n",
  "1566": "\nBEE\nWEW\nBBE\n",
  "1567": "\nWBW\nWEW\nBBE\n",
  "1568": "\nWBE\nWEW\nBBE\n",
  "1569": "\nWWB\nWEW\nBBE\n",
  "1571": "\nWWE\nWEW\nBBE\n",
  "1572": "\nWEB\nWEW\nBBE\n",
  "1573": "\nWEW\nWEW\nBBE\n",
  "1574": "\nWEE\nWEW\nBBE\n",
  "1575": "\nEBB\nWEW\nBBE\n",
  "1576": "\nEBW\nWEW\nBBE\n",
  "1577": "\nEBE\nWEW\nBBE\n",
  "1578": "\nEWB\nWEW\nBBE\n",
  "1579": "\nEWW\nWEW\nBBE\n",
  "1580": "\nEWE\nWEW\nBBE\n",
  "1581": "\nEEB\nWEW\nBBE\n",
  "1582": "\nEEW\nWEW\nBBE\n",
  "1583": "\nEEE\nWEW\nBBE\n",
  "1584": "\nBBE\nWEE\nBBE\n",
  "1585": "\nBWB\nWEE\nBBE\n",
  "1586": "\nBWW\nWEE\nBBE\n",
  "1587": "\nBWE\nWEE\nBBE\n",
  "1588": "\nBEB\nWEE\nBBE\n",
  "1589": "\nBEW\nWEE\nBBE\n",
  "1590": "\nBEE\nWEE\nBBE\n",
  "1591": "\nWBW\nWEE\nBBE\n",
  "1592": "\nWBE\nWEE\nBBE\n",
  "1593": "\nWWB\nWEE\nBBE\n",
  "1594": "\nWWW\nWEE\nBBE\n",
  "1596": "\nWEB\nWEE\nBBE\n",
  "1597": "\nWEW\nWEE\nBBE\n",
  "1598": "\nWEE\nWEE\nBBE\n",
  "1599": "\nEBB\nWEE\nBBE\n",
  "1600": "\nEBW\nWEE\nBBE\n",
  "1601": "\nEBE\nWEE\nBBE\n",
  "1602": "\nEWB\nWEE\nBBE\n",
  "1603": "\nEWW\nWEE\nBBE\n",
  "1604": "\nEWE\nWEE\nBBE\n",
  "1605": "\nEEB\nWEE\nBBE\n",
  "1606": "\nEEW\nWEE\nBBE\n",
  "1607": "\nEEE\nWEE\nBBE\n",
  "1609": "\nBWW\nEEB\nBBE\n",
  "1610": "\nBWE\nEEB\nBBE\n",
  "1611": "\nBEB\nEEB\nBBE\n",
  "1614": "\nWBW\nEEB\nBBE\n",
  "1616": "\nWWW\nEEB\nBBE\n",
  "1617": "\nWWE\nEEB\nBBE\n",
  "1619": "\nWEW\nEEB\nBBE\n",
  "1623": "\nEWW\nEEB\nBBE\n",
  "1624": "\nEWE\nEEB\nBBE\n",
  "1628": "\nBBE\nEEW\nBBE\n",
  "1629": "\nBWB\nEEW\nBBE\n",
  "1630": "\nBWW\nEEW\nBBE\n",
  "1631": "\nBWE\nEEW\nBBE\n",
  "1632": "\nBEB\nEEW\nBBE\n",
  "1633": "\nBEW\nEEW\nBBE\n",
  "1634": "\nBEE\nEEW\nBBE\n",
  "1635": "\nWBW\nEEW\nBBE\n",
  "1636": "\nWBE\nEEW\nBBE\n",
  "1637": "\nWWB\nEEW\nBBE\n",
  "1639": "\nWWE\nEEW\nBBE\n",
  "1640": "\nWEB\nEEW\nBBE\n",
  "1641": "\nWEW\nEEW\nBBE\n",
  "1642": "\nWEE\nEEW\nBBE\n",
  "1643": "\nEBW\nEEW\nBBE\n",
  "1644": "\nEBE\nEEW\nBBE\n",
  "1645": "\nEWB\nEEW\nBBE\n",
  "1647": "\nEWE\nEEW\nBBE\n",
  "1648": "\nEEB\nEEW\nBBE\n",
  "1649": "\nEEW\nEEW\nBBE\n",
  "1650": "\nEEE\nEEW\nBBE\n",
  "1652": "\nBWB\nEEE\nBBE\n",
  "1653": "\nBWW\nEEE\nBBE\n",
  "1654": "\nBWE\nEEE\nBBE\n",
  "1655": "\nBEB\nEEE\nBBE\n",
  "1656": "\nBEW\nEEE\nBBE\n",
  "1657": "\nBEE\nEEE\nBBE\n",
  "1658": "\nWBW\nEEE\nBBE\n",
  "1659": "\nWBE\nEEE\nBBE\n",
  "1660": "\nWWB\nEEE\nBBE\n",
  "1664": "\nWEW\nEEE\nBBE\n",
  "1665": "\nWEE\nEEE\nBBE\n",
  "1667": "\nEBW\nEEE\nBBE\n",
  "1669": "\nEWB\nEEE\nBBE\n",
  "1671": "\nEWE\nEEE\nBBE\n",
  "1672": "\nEEB\nEEE\nBBE\n",
  "1674": "\nEEE\nEEE\nBBE\n",
  "1675": "\nBWB\nWEW\nBWB\n",
  "1676": "\nBWW\nWEW\nBWB\n",
  "1677": "\nBWE\nWEW\nBWB\n",
  "1679": "\nBEW\nWEW\nBWB\n",
  "1680": "\nBEE\nWEW\nBWB\n",
  "1681": "\nWBW\nWEW\nBWB\n",
  "1682": "\nWBE\nWEW\nBWB\n",
  "1683": "\nWWW\nWEW\nBWB\n",
  "1684": "\nWWE\nWEW\nBWB\n",
  "1685": "\nWEW\nWEW\nBWB\n",
  "1686": "\nWEE\nWEW\nBWB\n",
  "1687": "\nEBE\nWEW\nBWB\n",
  "1688": "\nEWE\nWEW\nBWB\n",
  "1689": "\nEEE\nWEW\nBWB\n",
  "1690": "\nBEB\nWEE\nBWB\n",
  "1691": "\nBEW\nWEE\nBWB\n",
  "1692": "\nBEE\nWEE\nBWB\n",
  "1693": "\nWBW\nWEE\nBWB\n",
  "1694": "\nWBE\nWEE\nBWB\n",
  "1695": "\nWWB\nWEE\nBWB\n",
  "1696": "\nWWW\nWEE\nBWB\n",
  "1697": "\nWWE\nWEE\nBWB\n",
  "1698": "\nWEB\nWEE\nBWB\n",
  "1699": "\nWEW\nWEE\nBWB\n",
  "1700": "\nWEE\nWEE\nBWB\n",
  "1701": "\nEBW\nWEE\nBWB\n",
  "1702": "\nEBE\nWEE\nBWB\n",
  "1703": "\nEWB\nWEE\nBWB\n",
  "1704": "\nEWW\nWEE\nBWB\n",
  "1705": "\nEWE\nWEE\nBWB\n",
  "1706": "\nEEB\nWEE\nBWB\n",
  "1707": "\nEEW\nWEE\nBWB\n",
  "1708": "\nEEE\nWEE\nBWB\n",
  "1710": "\nBWW\nEEE\nBWB\n",
  "1711": "\nBWE\nEEE\nBWB\n",
  "1712": "\nBEB\nEEE\nBWB\n",
  "1713": "\nBEW\nEEE\nBWB\n",
  "1714": "\nBEE\nEEE\nBWB\n",
  "1715": "\nWBW\nEEE\nBWB\n",
  "1716": "\nWBE\nEEE\nBWB\n",
  "1717": "\nWWW\nEEE\nBWB\n",
  "1718": "\nWWE\nEEE\nBWB\n",
  "1719": "\nWEW\nEEE\nBWB\n",
  "1720": "\nWEE\nEEE\nBWB\n",
  "1721": "\nEBE\nEEE\nBWB\n",
  "1722": "\nEWE\nEEE\nBWB\n",
  "1723": "\nEEE\nEEE\nBWB\n",
  "1724": "\nWBW\nWEB\nBWW\n",
  "1725": "\nWBE\nWEB\nBWW\n",
  "1726": "\nWWW\nWEB\nBWW\n",
  "1727": "\nWWE\nWEB\nBWW\n",
  "1728": "\nWEW\nWEB\nBWW\n",
  "1729": "\nWEE\nWEB\nBWW\n",
  "1730": "\nEBW\nWEB\nBWW\n",
  "1731": "\nEBE\nWEB\nBWW\n",
  "1732": "\nEWW\nWEB\nBWW\n",
  "1733": "\nEWE\nWEB\nBWW\n",
  "1734": "\nEEW\nWEB\nBWW\n",
  "1735": "\nEEE\nWEB\nBWW\n",
  "1736": "\nWWB\nWEW\nBWW\n",
  "1737": "\nWWW\nWEW\nBWW\n",
  "1739": "\nWEB\nWEW\nBWW\n",
  "1740": "\nWEW\nWEW\nBWW\n",
  "1741": "\nWEE\nWEW\nBWW\n",
  "1742": "\nEBW\nWEW\nBWW\n",
  "1743": "\nEBE\nWEW\nBWW\n",
  "1744": "\nEWB\nWEW\nBWW\n",
  "1747": "\nEEB\nWEW\nBWW\n",
  "1748": "\nEEW\nWEW\nBWW\n",
  "1750": "\nWEB\nWEE\nBWW\n",
  "1751": "\nWEW\nWEE\nBWW\n",
  "1752": "\nWEE\nWEE\nBWW\n",
  "1753": "\nEBW\nWEE\nBWW\n",
  "1754": "\nEBE\nWEE\nBWW\n",
  "1755": "\nEWB\nWEE\nBWW\n",
  "1756": "\nEWW\nWEE\nBWW\n",
  "1757": "\nEWE\nWEE\nBWW\n",
  "1758": "\nEEB\nWEE\nBWW\n",
  "1759": "\nEEW\nWEE\nBWW\n",
  "1760": "\nEEE\nWEE\nBWW\n",
  "1761": "\nBWW\nEEB\nBWW\n",
  "1762": "\nBWE\nEEB\nBWW\n",
  "1763": "\nBEW\nEEB\nBWW\n",
  "1764": "\nBEE\nEEB\nBWW\n",
  "1765": "\nWBW\nEEB\nBWW\n",
  "1766": "\nWBE\nEEB\nBWW\n",
  "1767": "\nWWW\nEEB\nBWW\n",
  "1768": "\nWWE\nEEB\nBWW\n",
  "1769": "\nWEW\nEEB\nBWW\n",
  "1770": "\nWEE\nEEB\nBWW\n",
  "1771": "\nEBW\nEEB\nBWW\n",
  "1772": "\nEBE\nEEB\nBWW\n",
  "1773": "\nEWW\nEEB\nBWW\n",
  "1774": "\nEWE\nEEB\nBWW\n",
  "1775": "\nEEW\nEEB\nBWW\n",
  "1776": "\nEEE\nEEB\nBWW\n",
  "1777": "\nBWW\nEEW\nBWW\n",
  "1778": "\nBWE\nEEW\nBWW\n",
  "1779": "\nBEB\nEEW\nBWW\n",
  "1782": "\nWBW\nEEW\nBWW\n",
  "1786": "\nWEB\nEEW\nBWW\n",
  "1788": "\nWEE\nEEW\nBWW\n",
  "1789": "\nEBW\nEEW\nBWW\n",
  "1793": "\nEEB\nEEW\nBWW\n",
  "1795": "\nEEE\nEEW\nBWW\n",
  "1796": "\nBWW\nEEE\nBWW\n",
  "1797": "\nBWE\nEEE\nBWW\n",
  "1798": "\nBEB\nEEE\nBWW\n",
  "1799": "\nBEW\nEEE\nBWW\n",
  "1800": "\nBEE\nEEE\nBWW\n",
  "1801": "\nWBW\nEEE\nBWW\n",
  "1802": "\nWBE\nEEE\nBWW\n",
  "1803": "\nWWB\nEEE\nBWW\n",
  "1804": "\nWWW\nEEE\nBWW\n",
  "1805": "\nWWE\nEEE\nBWW\n",
  "1806": "\nWEB\nEEE\nBWW\n",
  "1807": "\nWEW\nEEE\nBWW\n",
  "1808": "\nWEE\nEEE\nBWW\n",
  "1809": "\nEBW\nEEE\nBWW\n",
  "1810": "\nEBE\nEEE\nBWW\n",
  "1811": "\nEWB\nEEE\nBWW\n",
  "1812": "\nEWW\nEEE\nBWW\n",
  "1813": "\nEWE\nEEE\nBWW\n",
  "1814": "\nEEB\nEEE\nBWW\n",
  "1815": "\nEEW\nEEE\nBWW\n",
  "1816": "\nEEE\nEEE\nBWW\n",
  "1817": "\nEBW\nWEB\nBWE\n",
  "1818": "\nEBE\nWEB\nBWE\n",
  "1819": "\nEWW\nWEB\nBWE\n",
  "1820": "\nEWE\nWEB\nBWE\n",
  "1821": "\nEEW\nWEB\nBWE\n",
  "1822": "\nEEE\nWEB\nBWE\n",
  "1826": "\nEEB\nWEW\nBWE\n",
  "1827": "\nEEW\nWEW\nBWE\n",
  "1828": "\nEEE\nWEW\nBWE\n",
  "1829": "\nEEB\nWEE\nBWE\n",
  "1830": "\nEEW\nWEE\nBWE\n",
  "1831": "\nEEE\nWEE\nBWE\n",
  "1832": "\nBWE\nEEB\nBWE\n",
  "1833": "\nBEW\nEEB\nBWE\n",
  "1834": "\nBEE\nEEB\nBWE\n",
  "1835": "\nWBW\nEEB\nBWE\n",
  "1836": "\nWBE\nEEB\nBWE\n",
  "1837": "\nWWW\nEEB\nBWE\n",
  "1838": "\nWWE\nEEB\nBWE\n",
  "1839": "\nWEW\nEEB\nBWE\n",
  "1840": "\nWEE\nEEB\nBWE\n",
  "1841": "\nEBW\nEEB\nBWE\n",
  "1842": "\nEBE\nEEB\nBWE\n",
  "1843": "\nEWW\nEEB\nBWE\n",
  "1844": "\nEWE\nEEB\nBWE\n",
  "1845": "\nEEW\nEEB\nBWE\n",
  "1846": "\nEEE\nEEB\nBWE\n",
  "1848": "\nBEB\nEEW\nBWE\n",
  "1849": "\nBEW\nEEW\nBWE\n",
  "1850": "\nBEE\nEEW\nBWE\n",
  "1851": "\nWBW\nEEW\nBWE\n",
  "1852": "\nWBE\nEEW\nBWE\n",
  "1855": "\nWEB\nEEW\nBWE\n",
  "1856": "\nWEW\nEEW\nBWE\n",
  "1857": "\nWEE\nEEW\nBWE\n",
  "1858": "\nEBW\nEEW\nBWE\n",
  "1859": "\nEBE\nEEW\nBWE\n",
  "1862": "\nEEB\nEEW\nBWE\n",
  "1863": "\nEEW\nEEW\nBWE\n",
  "1864": "\nEEE\nEEW\nBWE\n",
  "1865": "\nBWE\nEEE\nBWE\n",
  "1866": "\nBEB\nEEE\nBWE\n",
  "1867": "\nBEW\nEEE\nBWE\n",
  "1868": "\nBEE\nEEE\nBWE\n",
  "1869": "\nWBW\nEEE\nBWE\n",
  "1870": "\nWBE\nEEE\nBWE\n",
  "1871": "\nWWW\nEEE\nBWE\n",
  "1872": "\nWWE\nEEE\nBWE\n",
  "1873": "\nWEB\nEEE\nBWE\n",
  "1874": "\nWEW\nEEE\nBWE\n",
  "1875": "\nWEE\nEEE\nBWE\n",
  "1876": "\nEBW\nEEE\nBWE\n",
  "1877": "\nEBE\nEEE\nBWE\n",
  "1878": "\nEWB\nEEE\nBWE\n",
  "1879": "\nEWW\nEEE\nBWE\n",
  "1880": "\nEWE\nEEE\nBWE\n",
  "1881": "\nEEB\nEEE\nBWE\n",
  "1882": "\nEEW\nEEE\nBWE\n",
  "1883": "\nEEE\nEEE\nBWE\n",
  "1887": "\nWBW\nEEE\nBEB\n",
  "1888": "\nWBE\nEEE\nBEB\n",
  "1889": "\nWWW\nEEE\nBEB\n",
  "1891": "\nWEW\nEEE\nBEB\n",
  "1892": "\nWEE\nEEE\nBEB\n",
  "1894": "\nEWE\nEEE\nBEB\n",
  "1895": "\nEEE\nEEE\nBEB\n",
  "1896": "\nWBW\nEEB\nBEW\n",
  "1897": "\nWBE\nEEB\nBEW\n",
  "1898": "\nWWW\nEEB\nBEW\n",
  "1899": "\nWWE\nEEB\nBEW\n",
  "1900": "\nWEW\nEEB\nBEW\n",
  "1901": "\nWEE\nEEB\nBEW\n",
  "1902": "\nEBW\nEEB\nBEW\n",
  "1904": "\nEWW\nEEB\nBEW\n",
  "1905": "\nEWE\nEEB\nBEW\n",
  "1906": "\nEEW\nEEB\nBEW\n",
  "1907": "\nEEE\nEEB\nBEW\n",
  "1910": "\nWEW\nEEW\nBEW\n",
  "1911": "\nWEE\nEEW\nBEW\n",
  "1912": "\nEBW\nEEW\nBEW\n",
  "1913": "\nEBE\nEEW\nBEW\n",
  "1915": "\nEWE\nEEW\nBEW\n",
  "1917": "\nEEE\nEEW\nBEW\n",
  "1918": "\nWEB\nEEE\nBEW\n",
  "1919": "\nWEW\nEEE\nBEW\n",
  "1920": "\nWEE\nEEE\nBEW\n",
  "1921": "\nEBW\nEEE\nBEW\n",
  "1922": "\nEBE\nEEE\nBEW\n",
  "1923": "\nEWW\nEEE\nBEW\n",
  "1924": "\nEWE\nEEE\nBEW\n",
  "1925": "\nEEB\nEEE\nBEW\n",
  "1926": "\nEEW\nEEE\nBEW\n",
  "1927": "\nEEE\nEEE\nBEW\n",
  "1928": "\nEBW\nEEB\nBEE\n",
  "1930": "\nEWW\nEEB\nBEE\n",
  "1931": "\nEWE\nEEB\nBEE\n",
  "1932": "\nEEW\nEEB\nBEE\n",
  "1933": "\nEEE\nEEB\nBEE\n",
  "1935": "\nEWE\nEEW\nBEE\n",
  "1936": "\nEEW\nEEW\nBEE\n",
  "1937": "\nEEE\nEEW\nBEE\n",
  "1938": "\nEEB\nEEE\nBEE\n",
  "1939": "\nEEW\nEEE\nBEE\n",
  "1940": "\nEEE\nEEE\nBEE\n",
  "1941": "\nWBW\nBEB\nWBW\n",
  "1942": "\nWBE\nBEB\nWBW\n",
  "1943": "\nWWW\nBEB\nWBW\n",
  "1944": "\nWWE\nBEB\nWBW\n",
  "1945": "\nWEW\nBEB\nWBW\n",
  "1946": "\nWEE\nBEB\nWBW\n",
  "1947": "\nEBE\nBEB\nWBW\n",
  "1948": "\nEWE\nBEB\nWBW\n",
  "1949": "\nEEE\nBEB\nWBW\n",
  "1950": "\nWWW\nBEW\nWBW\n",
  "1951": "\nWWE\nBEW\nWBW\n",
  "1952": "\nWEW\nBEW\nWBW\n",
  "1953": "\nWEE\nBEW\nWBW\n",
  "1954": "\nEBW\nBEW\nWBW\n",
  "1955": "\nEBE\nBEW\nWBW\n",
  "1956": "\nEWW\nBEW\nWBW\n",
  "1957": "\nEWE\nBEW\nWBW\n",
  "1958": "\nEEW\nBEW\nWBW\n",
  "1959": "\nEEE\nBEW\nWBW\n",
  "1960": "\nWEW\nBEE\nWBW\n",
  "1961": "\nWEE\nBEE\nWBW\n",
  "1962": "\nEBW\nBEE\nWBW\n",
  "1963": "\nEBE\nBEE\nWBW\n",
  "1964": "\nEWW\nBEE\nWBW\n",
  "1965": "\nEWE\nBEE\nWBW\n",
  "1966": "\nEEW\nBEE\nWBW\n",
  "1967": "\nEEE\nBEE\nWBW\n",
  "1969": "\nWBE\nWEW\nWBW\n",
  "1970": "\nWWW\nWEW\nWBW\n",
  "1971": "\nWWE\nWEW\nWBW\n",
  "1972": "\nWEW\nWEW\nWBW\n",
  "1973": "\nWEE\nWEW\nWBW\n",
  "1974": "\nEBE\nWEW\nWBW\n",
  "1975": "\nEWE\nWEW\nWBW\n",
  "1976": "\nEEE\nWEW\nWBW\n",
  "1978": "\nWBE\nWEE\nWBW\n",
  "1979": "\nWWW\nWEE\nWBW\n",
  "1980": "\nWWE\nWEE\nWBW\n",
  "1981": "\nWEW\nWEE\nWBW\n",
  "1982": "\nWEE\nWEE\nWBW\n",
  "1983": "\nEBW\nWEE\nWBW\n",
  "1984": "\nEBE\nWEE\nWBW\n",
  "1985": "\nEWW\nWEE\nWBW\n",
  "1986": "\nEWE\nWEE\nWBW\n",
  "1987": "\nEEW\nWEE\nWBW\n",
  "1988": "\nEEE\nWEE\nWBW\n",
  "1990": "\nWBE\nEEE\nWBW\n",
  "1991": "\nWWW\nEEE\nWBW\n",
  "1992": "\nWWE\nEEE\nWBW\n",
  "1993": "\nWEW\nEEE\nWBW\n",
  "1994": "\nWEE\nEEE\nWBW\n",
  "1995": "\nEBE\nEEE\nWBW\n",
  "1996": "\nEWE\nEEE\nWBW\n",
  "1997": "\nEEE\nEEE\nWBW\n",
  "2000": "\nEWW\nBEB\nWBE\n",
  "2001": "\nEWE\nBEB\nWBE\n",
  "2002": "\nEEW\nBEB\nWBE\n",
  "2004": "\nEWW\nBEW\nWBE\n",
  "2005": "\nEWE\nBEW\nWBE\n",
  "2006": "\nEEW\nBEW\nWBE\n",
  "2007": "\nEEE\nBEW\nWBE\n",
  "2008": "\nEEW\nBEE\nWBE\n",
  "2009": "\nEEE\nBEE\nWBE\n",
  "2010": "\nWBE\nWEB\nWBE\n",
  "2011": "\nWWW\nWEB\nWBE\n",
  "2012": "\nWWE\nWEB\nWBE\n",
  "2013": "\nWEW\nWEB\nWBE\n",
  "2014": "\nWEE\nWEB\nWBE\n",
  "2015": "\nEBE\nWEB\nWBE\n",
  "2016": "\nEWW\nWEB\nWBE\n",
  "2017": "\nEWE\nWEB\nWBE\n",
  "2018": "\nEEW\nWEB\nWBE\n",
  "2019": "\nEEE\nWEB\nWBE\n",
  "2020": "\nWBE\nWEW\nWBE\n",
  "2022": "\nWWE\nWEW\nWBE\n",
  "2023": "\nWEW\nWEW\nWBE\n",
  "2024": "\nWEE\nWEW\nWBE\n",
  "2025": "\nEBW\nWEW\nWBE\n",
  "2026": "\nEBE\nWEW\nWBE\n",
  "2027": "\nEWW\nWEW\nWBE\n",
  "2028": "\nEWE\nWEW\nWBE\n",
  "2029": "\nEEW\nWEW\nWBE\n",
  "2030": "\nEEE\nWEW\nWBE\n",
  "2031": "\nWBE\nWEE\nWBE\n",
  "2032": "\nWWW\nWEE\nWBE\n",
  "2033": "\nWWE\nWEE\nWBE\n",
  "2034": "\nWEW\nWEE\nWBE\n",
  "2035": "\nWEE\nWEE\nWBE\n",
  "2036": "\nEBW\nWEE\nWBE\n",
  "2037": "\nEBE\nWEE\nWBE\n",
  "2038": "\nEWW\nWEE\nWBE\n",
  "2039": "\nEWE\nWEE\nWBE\n",
  "2040": "\nEEW\nWEE\nWBE\n",
  "2041": "\nEEE\nWEE\nWBE\n",
  "2042": "\nWBE\nEEB\nWBE\n",
  "2043": "\nWWE\nEEB\nWBE\n",
  "2044": "\nWEW\nEEB\nWBE\n",
  "2045": "\nWEE\nEEB\nWBE\n",
  "2046": "\nEBE\nEEB\nWBE\n",
  "2047": "\nEWE\nEEB\nWBE\n",
  "2048": "\nEEW\nEEB\nWBE\n",
  "2050": "\nWBE\nEEW\nWBE\n",
  "2051": "\nWWW\nEEW\nWBE\n",
  "2052": "\nWWE\nEEW\nWBE\n",
  "2053": "\nWEW\nEEW\nWBE\n",
  "2054": "\nWEE\nEEW\nWBE\n",
  "2055": "\nEBE\nEEW\nWBE\n",
  "2056": "\nEWW\nEEW\nWBE\n",
  "2057": "\nEWE\nEEW\nWBE\n",
  "2058": "\nEEW\nEEW\nWBE\n",
  "2059": "\nEEE\nEEW\nWBE\n",
  "2060": "\nWBE\nEEE\nWBE\n",
    "2061": "\nWWW\nEEE\nWBE\n",
    "2062": "\nWWE\nEEE\nWBE\n",
    "2063": "\nWEW\nEEE\nWBE\n",
    "2064": "\nWEE\nEEE\nWBE\n",
    "2065": "\nEBW\nEEE\nWBE\n",
    "2066": "\nEBE\nEEE\nWBE\n",
    "2067": "\nEWW\nEEE\nWBE\n",
    "2068": "\nEWE\nEEE\nWBE\n",
    "2069": "\nEEW\nEEE\nWBE\n",
    "2070": "\nEEE\nEEE\nWBE\n",
    "2073": "\nWEW\nWEW\nWWW\n",
    "2078": "\nWEW\nWEE\nWWW\n",
    "2081": "\nEWW\nWEE\nWWW\n",
    "2085": "\nWWW\nEEE\nWWW\n",
    "2088": "\nWEE\nEEE\nWWW\n",
    "2089": "\nEBE\nEEE\nWWW\n",
    "2101": "\nWWE\nEEB\nWWE\n",
    "2102": "\nWEE\nEEB\nWWE\n",
    "2103": "\nEBE\nEEB\nWWE\n",
    "2104": "\nEWE\nEEB\nWWE\n",
    "2105": "\nEEE\nEEB\nWWE\n",
    "2107": "\nWEW\nEEW\nWWE\n",
    "2109": "\nEBE\nEEW\nWWE\n",
    "2112": "\nEEE\nEEW\nWWE\n",
    "2114": "\nWEW\nEEE\nWWE\n",
    "2115": "\nWEE\nEEE\nWWE\n",
    "2116": "\nEBE\nEEE\nWWE\n",
    "2118": "\nEWE\nEEE\nWWE\n",
    "2119": "\nEEW\nEEE\nWWE\n",
    "2120": "\nEEE\nEEE\nWWE\n",
    "2122": "\nWEE\nEEE\nWEW\n",
    "2123": "\nEBE\nEEE\nWEW\n",
    "2124": "\nEWE\nEEE\nWEW\n",
    "2125": "\nEEE\nEEE\nWEW\n",
    "2127": "\nEWE\nEEB\nWEE\n",
    "2128": "\nEEE\nEEB\nWEE\n",
    "2130": "\nEEE\nEEW\nWEE\n",
    "2131": "\nEEW\nEEE\nWEE\n",
    "2132": "\nEEE\nEEE\nWEE\n",
    "2134": "\nEWE\nBEB\nEBE\n",
    "2137": "\nEEE\nBEW\nEBE\n",
    "2139": "\nEBE\nWEW\nEBE\n",
    "2140": "\nEWE\nWEW\nEBE\n",
    "2141": "\nEEE\nWEW\nEBE\n",
    "2142": "\nEBE\nWEE\nEBE\n",
  "2143": "\nEWE\nWEE\nEBE\n",
  "2144": "\nEEE\nWEE\nEBE\n",
  "2146": "\nEWE\nEEE\nEBE\n",
  "2147": "\nEEE\nEEE\nEBE\n",
  "2151": "\nEWE\nEEE\nEWE\n",
  "2152": "\nEEE\nEEE\nEWE\n",
  "2153": "\nEEE\nEEE\nEEE\n"
```

Default MCTS-based Fuego player settings

```
uct_param_search number_threads 1
uct_param_player ignore_clock 1
uct_param_player max_games NUSIM
uct_param_player reuse_subtree 0
uct_param_player ponder 0
uct_param_player forced_opening_moves 0
go_rules cgos
```

No Additive player settings

```
uct_param_globalsearch use_additive_predictor 0
uct_param_search number_threads 1
uct_param_player ignore_clock 1
uct_param_player max_games NUSIM
uct_param_player reuse_subtree 0
uct_param_player ponder 0
uct_param_player forced_opening_moves 0
go_rules cgos
```

No Knowledge player setting

```
uct_param_feature_knowledge prior_knowledge_type none
uct_param_globalsearch use_default_prior_knowledge 0
uct_param_globalsearch use_tree_filter 0
uct_param_search number_threads 1
uct_param_player ignore_clock 1
uct_param_player max_games NUSIM
uct_param_player reuse_subtree 0
uct_param_player ponder 0
uct_param_player forced_opening_moves 0
go_rules cgos
```

Playout Policy-Only player setting

```
uct_param_player search_mode playout_policy
uct_param_search number_threads 1
uct_param_player ignore_clock 1
uct_param_player max_games NUSIM
uct_param_player reuse_subtree 0
uct_param_player ponder 0
uct_param_player forced_opening_moves 0
go_rules cgos
```