Doctoral Dissertation

Computing Games and Its Implication to Measuring Playing Experience

Anggina Primanita

Supervisor: Hiroyuki Iida

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
Information Science

March, 2021

Supervised by

Prof. Dr. Hiroyuki Iida

Reviewed by

Associate Prof. Dr. Ikeda Kokolo

Associate Prof. Dr. Shirai Kiyoaki

Prof. Dr. Zulkardi

Associate Prof. Dr. Jr-Chang Chen

Abstract

Over history, games have served multiple purposes. It serves as a fun activity for players who need entertainment to become test-beds for artificial intelligence. Solving games is beneficial in providing a better understanding of how information is progressing throughout the game. Applying search algorithms to solve games divides the game-tree into two areas: "explored nodes" and "unexplored nodes", each signifies certainty and uncertainty value in the game respectively. Uncertainty in games does not only affects the way a game is solved, but also the way the game is experienced. A game typically progresses from uncertain state to certain state. Currently, a game progress can be observed using two different approaches. The first approach is by monitoring its game progress pattern using a certain indicator while the second approach is by analyzing and observing the game information progress model. Previous works have interpreted uncertainty in the game progress through various means, but there has been no clear links among those interpretations. Observing the effect of uncertainty in the game may lead to the linking between those definitions.

In this thesis, the probability-based proof number search and single conspiracy number, which derived from the idea of Conspiracy Number (CN), were used to analyze how uncertainty affects various elements of games. The probability-based proof number search is a domain-independent best-first search algorithm which main purpose is to solve games. It exploits information from both areas of a game-tree by combining certain and uncertain information to reach convergence. The single conspiracy number is an indicator used to evaluate the difficulty of the current state of the game. It is useful to evaluate progress patterns and long-term positions, as well as to evaluate game playing positions.

This thesis focuses on understanding the influence of uncertainty in the best-first search game computation along with its impact on game entertainment. To achieve it, we are guided by three purposes: (1) To find the optimal difficulty ordering procedure for game solver for different game-tree structures. To see how the uncertainty affects ordering process in best-first search, the probability-based proof number search is deployed into several games with notably distinct structures. Results from the experiment on a total of 1000 Othello positions from different stages of the game revealed that information coming from uncertainties affects the result differently in each stage. It expedites the convergence of solved positions, reaching 100% in the end-game positions. Later, the framework is expanded into a single-player game.;(2) To define the indicator for entertainment using a game-tree search framework and (3) To define the link between the game-tree search result and entertainment indicator in different game environments. Using

the single-conspiracy number as a game position evaluation indicator based on expert play demonstrates that it can show progress continuity. Furthermore, the indicator is expanded into a single-player game. The expansion result shows the game progress related to simulated players' ability. Finally, the result is used to find the momentum and potential energy of a single-player game using the motion in mind concept. The experiment demonstrates the link between the search indicator of a game-tree and the measure of entertainment. In the end, it is concluded that uncertainty plays an important part in game computation as well as game entertainment. Even more, it has become a requirement in both computation and entertainment measure, as it impacts both the quality of games' computation result and also how the game-playing experience is perceived.

Keyword: *game solver, game progress pattern, probability-based proof number search, single conspiracy number, motion in mind*

Acknowledgments

First and foremost, praises and thanks to Allah SWT, for only by His blessings in every step that I took that I am able to be who I am today.

I would like to express my deep and sincere gratitude to my supervisor, Prof. Hiroyuki Iida for giving me the opportunity to study and carry out my research here, at JAIST. His generosity and abundance, as well as patience throughout my study in JAIST, have led me to become a better researcher while still being able to enjoy my life in Japan.

I want to thank Dr. Samsuryadi and Prof. Kazuhiro Ogata as my minor research supervisors. They helped and guide me in carrying out my minor research in the field of combinatorial games.

Thank you to Associate Prof. Kokolo Ikeda and Mrs. Asakura for the kind help during my study and daily life. A big thanks to Dr. Mohd. Nor Akmal Khalid for his guidance and assistance throughout my publications. I have not been the best researcher, but he brings the best of my research. Thanks to the committee members of my Ph.D. defense: Associate Prof. Kokolo Ikeda, Associate Prof. Kiyoaki Shirai, Associate Prof. Jr-Chang Chen, and Prof. Zulkardi. Thank you to Mr. Huang Shunqi for his assistance in developing the SCN program. Thanks to all the students in the Iida lab as well as PPI-JAIST. Thank you for making my student life colorful. Thank you to all the staff of JAIST.

Next, Thank you to my parents. The knowledge about life that they imparted to me has been the biggest pillar of my life. For my brother, thank you for being my first and always brainstorming friend.

Special and loving thanks go to my husband, Dr. Hadipurnawan Satria. He had to put aside a lot of things to accompany me to JAIST as well as helping me through hard times. I am looking forward to enjoying our future together.

Finally, my thanks go to all the people who have supported me to complete the research work directly or indirectly.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Chapter Introduction

In this chapter, we first introduce the history of the game as test-beds for Artificial Intelligence (AI) and discuss how it affects the growth of AI. As the major interest of this thesis lies in analyzing the result of generalizing solvers and analyzing its results to see its impact to measure of entertainment, we overview the general game solving techniques. Finally, we summarize our contributions and outline the contents of this thesis.

1.2 Background

Game is one of the most sought after activity for human that needs entertainment. Currently, the game-playing culture has been proven as a place of comfort when human is not allowed to enjoy physical social activity [12]. However, over history, games have served multiple purposes. One of the most prominent is the use of the game as test-beds for AI. When the first concept of computing was introduced, the idea of using games as test-beds were also introduced. Some of the first games that were used as test-beds for Artificial Intelligence (AI) was Chess [60] and Checkers [53]. Up until now, games still fascinate the AI ground and still serves its purpose as an AI test-bed. The main reason for games being used as AI test-beds is because it is cheap, deterministic, easily repeatable and controllable, as well as an entertaining environment. Many problems that are encountered in games also reflect real-world problems, which made the uses of the game is much more

impactful [70].

One of the first instances of using a game as the framework to test a working AI idea was in chess. Its reference in AI, in particular, was so widespread and comprehensive that it is referred to as "The Drosophila of AI" [37]. Consisting of five stages, the history of games in AI holds many landmarks that affect the development of Computers and AI to become what we know of today. Before the discovery of the computer, the very first mention of AI is "The Turk" system. Created by Wolfgang von Kempelen, "The Turk" excites the crowd at the time as it displays the idea of machines that can operate by itself. This idea is then attracting lots of scientists and experts and led them to explore the field of automaton.

When computers were introduced, Claude Shannon proposed the Minmax algorithm as a solution to chess. In his publication, he hoped that providing a solution to chess would be a part of solving problems with greater significance. It is because, according to Shannon, computer Chess is an ideal starting point to create a general-purpose computer and it can act as a foundation for the problem of greater significance [60]. This idea is then agreed upon by Alan Turing, he stated that using chess to show machine intelligence is one of the approaches that should be tried [72]. Later, Marsland concludes that if a machine is unable to make an agreeable decision on a perfect information game such as chess, it will be hard to prove that decision taken by a computer in the imperfect or uncertain condition is a good decision [35].

The Minmax algorithm is then inspired McAllester to propose an idea called the Conspiracy Numbers (CN) [36]. It is a search procedure that is able to build a solution tree without the need of specific heuristics in a MIN/MAX Tree. The main purpose of CN is to reduce the possible node expansion by ordering the possible node expansion based on its CN. Its initial proposal entices the possibility of a more streamlined search procedure, albeit the fact that the implementation is still impractical. Ever since CN its proposal, the extension of the CN idea has been mainly expanded into two directions, namely, to solve games and to understand game progress.

The main objective of solving games is to find the game's theoretic value. The concept is usually applied to full-information games without any chance element. It is a part of achieving the general goal of creating a good game-playing program[29]. On the other

hand, solving games proved to be beneficial in providing a better understanding of how a game works [73]. This condition ultimately leads to a better understanding of how information is progressing throughout the game. Specifically, this study employs a best-first search algorithm called Probability-based Proof Number Search (PPN-Search). It is a derivative of the CN idea. PPN-Search divides the game-tree into two areas: "explored nodes" and "unexplored nodes". The "explored nodes" are nodes that are visited and have a certain value. The "unexplored nodes" are nodes that have not been visited before, which made it uncertain.

Unlike most other best-first search algorithms, such as Proof Number Search (PNS) [3], Depth-first Proof Number Search (Df-PNS) [44], or PN^2 [54, 76, 51], the PPN-Search values the uncertainty by assigning winning rate value. The first incarnation of PPN-Search on the simulated balanced game-tree was proven to be optimal. However, the impact of uncertainty values coming from "unexplored nodes" is yet to be explored.

Uncertainty in games does not only affect the way a game is solved, but also the way the game is experienced. While a game being played, it is started in a state of uncertainty where there was no information about the winner of the game [20]. As the game progresses more information is obtained. At the end of the game, all of the information is successfully obtained by its player. In this thesis, the changing states from uncertain to certain in a game is subsequently called the game progress.

Currently, the game progress can be observed using two different approaches. The first approach is by observing its game progress pattern using a certain indicator. This thesis focuses on an indicator called the Single Conspiracy Number (SCN) [65]. It is a derivation of the CN idea as well as the proof number search (PNS) [3]. Its proposed usages are to evaluate the difficulty of the current state of the game of getting the MIN/MAX value over the specified threshold point. The later usage of it, however, has shown that SCN is useful to evaluate progress patterns and long-term positions [66], as well as evaluate game playing position [7].

The second approach is by analyzing and observing the game information progress model [20]. Such a model analyzes information from the designer's viewpoint. It defines the information of the game result as the amount of solved uncertainty. Expansion of the model to different games, such as board game [21], sports game [23, 43], and video game

[78] provides sophistication measure. Later work in the game progress model allows it to expand the concept into the "Motion in Mind" model that bridges motion in physics and mind [19]. Although the two approaches seem to be dissimilar, both exploit the uncertainty in a game to measure its progress. In this thesis, the link between the two approaches are investigated.

In this thesis, the aim of adopting the variations of CN Search, namely the PPN-Search and the Single Conspiracy Number (SCN) as the main tools for the experimentation of this study involves understanding the impact of uncertainty in games. The first being the impact of uncertainty in the search-tree computation while the latter being the impact of uncertainty in the game entertainment. Based on such a discussion, the link between the effect of uncertainty in the domain of optimal play to the game entertainment measure can be established.

1.3 Statement of Research Question

This thesis focuses on understanding the influence of uncertainty in game computation as well as its impact on game entertainment. To achieve it, we have defined specific objectives that must be attained: (1) To find the optimal difficulty ordering procedure for game solver for different game-tree structures. Each game-tree structure has its own characteristics, which the PPN-Search has not previously covered. To see how the uncertainty affects the algorithm, it is deployed into games with drastically different structures and single-player game. (2) To define the indicator for entertainment using a game-tree search framework and (3) To define the link between the game-tree search result and entertainment indicator in different game environments. There are two measure of game progress which closely linked to uncertainty, but seems to be dissimilar with each other. At the end of this thesis, the link between the two will be explored through single-player game progress.

1.4 Structure of The Thesis

This thesis comprises six main chapters, given as follows:

- **Chapter 1: Introduction**

This chapter's objective is to introduce the big picture of the research, such as its definitions, how each of the keywords relates to each other in the research, as well as a brief historical overview of the domain considered. It serves to explain the main problem that the research aims to solve. The Introduction chapter also includes the statement of the research questions, as well as the goal and significance of the research. At the end of this chapter, the structure of the dissertation will be stated.

- **Chapter 2: Literature Review**

  The chapter serves as a review of the theoretical background related to this research as well as presenting the state-of-the-art research in the field. The first section of this chapter is a review of the best-first search algorithms. The main algorithms that are covered in this section are the Conspiracy Number Search (CNS), Proof Number Search (PNS), and Monte Carlo Tree Search (MCTS). In this section, the algorithms are reviewed in a historical manner, which means that the earliest idea, which is the CNS, will be reviewed first, following by the PNS and MCTS. The second section of the second chapter covers the review of state-of-the-art algorithms that aimed to combine the idea of PNS and MCTS. The search algorithms that will be covered are the Monte Carlo Proof Number Search and Product Propagation. The third section of the Literature Review covers the research related to the game-tree search. Currently, there are two distinct shapes of the game-tree. Balanced game-tree and unbalanced game-tree. It is important to present the difference between the game-tree shape as it affects the optimality of algorithms that are used to solve the game. Each of the shapes also comes with its problems, such as the see-saw problem. The last section of the Literature Review chapter covers the game refinement theory, a measure of entertainment in games based on the uncertainties. At the end of the chapter, a conclusion that leads to the justification of the research carried out in the dissertation will be presented.

- **Chapter 3: Characterizing Probability-based Proof Number Search**

  The third chapter in the dissertation covers the result of research that leads to the Characterization of PPN-Search. PPN-Search previously has only been implemented in a simulated balanced game-tree structure setting, thus, it is not enough

to get the conclusion of its optimality in solving real-world games. As an attempt to discover PPN-Search characteristics, in this chapter, the algorithm is employed to solve two games with drastically different tree structures as test-beds. The first game is a real game with an unbalanced tree structure, which is the Connect Four game, while the second game is a real game with a balanced tree structure, which is the Othello game. At the end of this chapter, the traits of PPN-Search will be presented as its conclusion, which also leads to the following chapter.

- **Chapter 4: Solving Single-Agent Game with Probability-based Proof Number Search**

  The fourth chapter of the dissertation covers research carried out related to the Application of PPN-Search in Single Agent Game. PPN-Search has previously only been used to solve games with two players. In this chapter, PPN-Search is adapted to solve a single agent game. As there are a lot of different kinds of games that require only one player, the chapter is started with the definition of single-agent game in the research context. Following that, the newly adapted Single Agent PPN-Search algorithm is presented. Following the introduction of the Single Agent PPN-Search, the game that was used as test-beds will be introduced. Its rules, as well as its single-agent game-tree structure will be presented in the chapter. Following that, the methodology of the research will be presented as well as the result and discussion. At the end of this chapter, the advantage and disadvantages of implementing Single Agent PPN-Search will be presented as a conclusion.

- **Chapter 5: Finding The Critical Aspect of Single-Agent Game Using Single Conspiracy Number Indicator**

  The fifth chapter of the dissertation covers research concerning a critical position in a single agent game. SCN is an expansion of the Conspiracy Number idea. Interestingly, although Conspiracy Number was used to show the difficulty of a position to be reached, the SCN was able to not only represent the difficulty of a position, but also showing the critical position that represents changes of player state of mind, or player strategy. In this chapter, SCN is expanded into a single agent game to find a critical position in such a game. The chapter is started with

the introduction of SCN and preliminary results of implementing SCN into a two-player board game. It is then followed by an introduction to the games used in the research. Following that, the methodology of the research will be presented as well as the result and discussion. The effectiveness of the SCN in pointing out the critical position in a single agent game will serve as the conclusion of this chapter.

- **Chapter 6: Conclusion**

  The last chapter is the conclusion of the dissertation. It concludes the whole dissertation relative to the main aim and objectives of the dissertation. Some potential future works are also outlined.

Chapter 2

Literature Review

2.1 Chapter Introduction

The chapter serves as a review of the theoretical background related to this research as well as presenting the state-of-the-art research in the field. The first section of this chapter is a review of the best-first search algorithms. The main algorithms that are covered in this section are The Conspiracy Number Search (CNS), Proof Number Search (PNS), and Monte Carlo Tree Search (MCTS). In this section, the algorithms are reviewed in a historical manner, which means that the earliest idea, which is the CNS, will be reviewed first, following by the PNS and MCTS. The idea of Conspiracy Number (CN) serves as the main inspiration behind Proof Number Search (PNS), as well as Single Conspiracy Number (SCN), both being important keywords of this thesis. Thus, it is important to review the historical importance of this search algorithm. Following the review of the CNS, the PNS algorithm will be reviewed. Currently, PNS has been implemented in different fields, however, in its inception, PNS uses the game-tree as its main test-beds. In this section, several varieties of PNS will also be reviewed. The last algorithm to be reviewed is the MCTS, also a widely used algorithm that serves as the base for the Probability-Based Proof Number Idea. In this section, several varieties of MCTS will also be reviewed. The second section of the second chapter covers the review of state-of-the-art algorithms that aimed to combine the idea of PNS and MCTS. The search algorithms that will be covered are the Monte Carlo Proof Number Search and Product Propagation. The third section of the Literature Review covers the research related to Game-tree Search. Currently, there

are two distinct shapes of game-tree. Balanced game-tree and unbalanced game-tree. It is important to present the difference between the game-tree shape as it affects the effectiveness of algorithms that are used to solve the game. Each of the shapes also comes with its problems, such as the see-saw problem. The last section of the Literature Review chapter covers the game refinement theory, a measure of entertainment in games based on the uncertainties. At the end of the chapter, a conclusion that leads to the justification of the research carried out in the dissertation will be presented.

2.2 Best-First Search Algorithms

Best-first search is an umbrella term for search algorithms that expand the next most promising node with the lowest cost [31]. One of its prominent examples, the Proof Number Search (PNS) was initially introduced as a method to find the game-theoretic value using a specific technique to progress towards a game-tree framework. In this section, notable best-first search algorithms will be reviewed.

2.2.1 Conspiracy Number Search

Conspiracy number search (CNS) is a search technique designed to be applied to the MIN/MAX tree. The algorithm is proposed by David Allen McAllester [36] based on the idea that in the larger games where the search space is very big, decisions must be able to be taken based on only a fraction of the search tree [36]. In the algorithm, the expansion of the tree is based on a set of values called the conspiracy number (CN).

The set of CN in a node represents the likelihood of the root node in a tree to take on a particular value. When a node, n is a leaf node, and its evaluation value v, then the CN of the number that is equal to $CN(v) = 0$. All other values other than v is given 1. When a node n is a terminal node, then, the value is said to required ∞ conspirators, since it cannot be changed anymore.

When n is an internal node, the set of CN are broken into two groups: numbers required to increase ($\uparrow$ *needed*) and decrease ($\downarrow$ *needed*) n's v. Values below the MIN/MAX evaluation value m in $\uparrow$ *needed* and above the MIN/MAX score in $\downarrow$ *needed* are assigned 0. $\uparrow$ *needed* and $\downarrow$ *needed* of an interior node's children can be combined to form the

parent's conspiracy numbers, CN, using the following rules [56]:

1. if $v = m$:

$$CN(v) = 0 \tag{2.1}$$

2. For a MAX node:

$$\begin{aligned} CN(v) &= \sum_{allchildreni} \downarrow needed_i(v), forallv < m \\ CN(v) &= \min_{allchildreni} \uparrow needed_i(v), forallv > m \end{aligned} \tag{2.2}$$

3. For a MIN node:

$$\begin{aligned} CN(v) &= \min_{allchildreni} \downarrow needed_i(v), forallv < m \\ CN(v) &= \sum_{allchildreni} \uparrow needed_i(v), forallv > m \end{aligned} \tag{2.3}$$

The main idea of CNS is to keep continuing the search based on the CN values until it is unlikely for the root of the tree to change. The definition of unlikely may vary from one implementation to another, thus, in the core algorithm, a conspiracy threshold, CT, is used. It represents the minimum number of conspirators required to consider the value to be unlikely to changed [56]. When the CT value is higher, the confidence in the choice of moves is greater.

It is stated that the conspiracy number is able to explore the critical line of play deeper than the noncritical lines [36]. This property, later, leads to a new ground of usage and exploration of the CN and CNS [45]. However, as a search technique, it was deemed to require heavy computation.

2.2.2 Proof Number Search

One of the prominent best-first search algorithms in the context of solving games is the proof number search (PNS) [3]. PNS utilizes two variables, proof and disproof numbers (pn and dn for short, respectively), as the search indicators to find the best possible options. The proof number (PN) of a node n represents the smallest number of leaf nodes that have to be proven in order to prove that it is a win, while the disproof number

represents the smallest number of leaf nodes that have to be disproved in order to prove that it is a loss, thus representing the difficulty in proving a node [24]. A node is then chosen to provide the best possible choice (called the most proving node (MPN)) and subsequently expanded for further search [27, 22, 46]. The formalism of PN calculation of a node is as follows:

1. When n is a terminal node:

   (a) if n is a win for the attacker:

$$\begin{aligned} n.pn &= 0 \\ n.dn &= \infty \end{aligned} \tag{2.4}$$

   (b) if n is a loss for the attacker:

$$\begin{aligned} n.pn &= \infty \\ n.dn &= 0 \end{aligned} \tag{2.5}$$

2. When n is an internal node:

   (a) if n is an OR node:

$$\begin{aligned} n.pn &= \min_{n_c \epsilon childrenofn} n_c.pn \\ n.dn &= \sum_{n_c \epsilon childrenofn} n_c.pn \end{aligned} \tag{2.6}$$

   (b) if n is an AND node:

$$\begin{aligned} n.pn &= \sum_{n_c \epsilon childrenofn} n_c.pn \\ n.dn &= \min_{n_c \epsilon childrenofn} n_c.pn \end{aligned} \tag{2.7}$$

An improvement of PNS that aims to reduce the usage of computing resources is the depth-first proof number (df-pn) search [41], which turns PNS from a best-first search algorithm into a depth-first search by introducing iterative deepening. The df-pn expands

less on the interior node and reduces the number of proof numbers and disproof numbers that have to be recomputed.

The most recent improvements to the df-pn are the DFPN-E. It added an edge cost initialization to the df-pn. Its application to the chemical synthesis problem with an unbalanced search space, as the tree is identified to be lopsided, is proven to be successful with the least number of nodes traversed[25].

2.2.3 Monte-Carlo Tree Solver

The Monte-Carlo tree search (MCTS) is another prevalent best-first search algorithm that has been employed to solve games. First proposed by [10], the MCTS framework consists of four steps: (1) selection, (2) expansion, (3) simulation, and (4) backpropagation. The algorithm starts with selecting the next action based on a stored value (selection). When it encounters a state that cannot be found in the tree, it expands the node (expansion). The node expansion is based on multiple, randomly simulated games (simulation or playout). The value is then stored and backpropagated to the root of the tree, where the algorithm continues to repeat the steps until the desired outcome is reached (backpropagation). MCTS utilizes simulation to gain information from unexplored nodes, and this was expanded by the introduction of upper confidence bound applied to trees (UCT) by [30]. A new best-first search algorithm derived from MCTS and UCT was then introduced by [8]. The algorithm's goal was to overcome the difficulty found in building heuristic knowledge for a non-terminal game state by employing stochastic simulations. Determining a game-playing strategy is effective when using multiple simulations. Its usage has become well known, especially in the field of Go, and in part led to the AlphaGo's wins against top grandmasters [63].

MCTS or UCT solver includes four strategic steps:

1. **Selection**: Selection picks a child to be searched based on the information gained from the root to a leaf node. In this step, the child with the largest simulation value is selected. For UCT, it selects the child with the largest UCT value, obtained from:

$$v_i + \sqrt{\frac{C \times lnn_p}{n_i}} \tag{2.8}$$

where v_i is the simulation value of the node i, n_i is the visit count of the child i and n_p is the visit count of current node p. C in the equation is the parameter that can be tuned. A transformed UCT/MCTS solver assumes that a root node having the value ∞ as a proved win, while a root node with $-\infty$ is a proved not to win.

2. **Play-Out**: If the currently visited nodes do not have any value. It means that it is yet to be part of the tree. When this occurs, a certain number of self-play simulations are carried. Normally, each simulation consists of the two players carrying out random moves until the end of the game. The number of wins is recorded to be part of the node.

3. **Expansion**: In the expansion step, it is decided whether a node will be added to the tree.

4. **Backpropagation**: In this step, the value coming from the Play-out steps is back-propagated from the leaf nodes to the whole game-tree, up to the root node. In the UCT and MCTS solver, the values being propagated are not limited to the 1,0,-1. The game-theoretical value of ∞ and $-\infty$ are also backpropagated in a Negamax-like way.

2.3 Combining Certainties with Uncertainties

In this section, two algorithms that aim to combine all information available in a search tree are introduced. Both of the algorithms use the information from explored nodes, as well as unexplored nodes.

2.3.1 Product Propagation

Product Propagation (PP) proposed as an algorithm to solve perfect information two-player two-outcome games by combining the idea of PNS and probabilistic reasoning [50]. It is designed to work with the MAX/MIN game-tree. Each node n is associated with a number $PPN(n) \in [0, 1]$.

When n is a terminal node, if the state it represents is equal to win for the MAX node, the $PPN(n)$ is equal to 1, other wise, if MAX loses, the value is equal to 0. When n is

a leaf node, it is given a $PPN(n)$ value of $\frac{1}{2}$. The formalism of $PPN(n)$ calculation for each nodes is as follows:

1. If a node n is a terminal node:

   (a) if (n) is a winning node,

   $$n.ppn = 1 \tag{2.9}$$

   (b) if (n) is not a winning node,

   $$n.ppn = 0 \tag{2.10}$$

2. If a node n is a leaf node:

   $$n.ppn = 0.5 \tag{2.11}$$

3. if a node n is an internal node, then

   (a) if n is an OR node,

   $$n.ppn = 1 - \prod_{n_c \in childrenofn} 1 - n_c.ppn \tag{2.12}$$

   (b) if n is an AND node,

   $$n.ppn = \prod_{n_c \in childrenofn} n_c.ppn \tag{2.13}$$

As PP is a best-first search algorithm, it needs to find the most promising nodes to expand. In this case, if the parent is a MAX node, then the most promising node is the child with the highest $PPN(n)$, similarly, if the parent is a MIN node, the most promising node is the child with the lowest $PPN(n)$. The search continues until the root node $PPN(n)$ reach 1 or 0.

In three perfect information game test-beds, namely Y, Domineering, and NoGo, PP, as well as its enhancements PP^2 were able to outperform other PNS variants[50].

The product propagation combines two information in a game-tree, information coming from certainty (terminal nodes) and uncertainty (leaf nodes). In the case of PP, information coming from the uncertainty is given a certain value, which is 0.5 to represent the probability of winning and losing.

2.3.2 Monte-carlo Proof Number Search

Although it has been proven that MCTS converges to minimax when evaluating the available moves [28], MCTS converges only in so-called "Monte Carlo Perfect" games [?]. By leveraging both strengths of PNS and MCTS, an improved game solver's quality is expected. An early example of such a combination is the Monte-Carlo proof number search (MCPNS) [52]. The MCPNS is a best-first search designed to work in an AND/OR game-tree. The work proposed the best-first search algorithm by employing its MIN/SUM rules to backpropagate information coming from the simulation, while gaining access to node information based on its Monte-Carlo infused proof number, called the Monte-Carlo proof number (*pmc*) as well as its counterpart, the Monte-Carlo disproof number (*dmc*). The formalism of *pmc* and *dmc* calculation is as follows:

1. When n is a terminal node:

   (a) if n is a winning node:

   $$\begin{aligned} n.pmc &= 0 \\ n.dmc &= \infty \end{aligned} \tag{2.14}$$

   (b) if n is not a winning node:

   $$\begin{aligned} n.pmc &= \infty \\ n.dmc &= 0 \end{aligned} \tag{2.15}$$

2. When n is a leaf node, and R is the winning rate of play-out steps. Take θ as a small positive number smaller than 1 and close to 0:

   (a) if $R \in (0, 1)$:

   $$\begin{aligned} n.pmc &= 1 - R \\ n.dmc &= R \end{aligned} \tag{2.16}$$

(b) if $R = 1$:

$$\begin{aligned} n.pmc &= \theta \\ n.dmc &= R - \theta \end{aligned} \tag{2.17}$$

(c) if $R = 0$:

$$\begin{aligned} n.pmc &= R - \theta \\ n.dmc &= \theta \end{aligned} \tag{2.18}$$

3. When n is an internal node:

(a) if n is an OR node:

$$\begin{aligned} n.pmc &= \min_{n_c \epsilon children of n} n_c.pmc \\ n.dmc &= \sum_{n_c \epsilon children of n} n_c.pmc \end{aligned} \tag{2.19}$$

(b) if n is an AND node:

$$\begin{aligned} n.pmc &= \sum_{n_c \epsilon children of n} n_c.pmc \\ n.dmc &= \min_{n_c \epsilon children of n} n_c.pmc \end{aligned} \tag{2.20}$$

To find the most promising nodes to expand the MCPNS is similar to that of PNS. If the parent is an OR node, then the most promising node is the child with the highest *pmc*, similarly, if the parent is a AND node, the most promising node is the child with the highest *dmc*.

Experiments with the Life-and-Death sub-problem of Go shows that with the right set of parameters, MCPNS is able to outperform PNS[52]. Following the experiment, there has been no further implementation of the algorithm that we know of although The MCPNS provides a better degree of flexibility while retaining its reliability. However, we see that there could be a major drawback in the calculation as its MIN/SUM rules compute results from integer numbers, while the statistical results from the MCTS counterparts produce a real number, thus leading to a loss of information from the simulation.

2.4 Game-tree Structures

Game-tree is heavily used as a test-bed for best-first search algorithms. In this section two tree structures that are popular to be used to represent games are introduced.

2.4.1 MIN/MAX tree

MIN/MAX game-tree stems from the two-person zero-sum perfect information game. The players are thought to be MIN and MAX player. In which the first player to move is the MAX player[48]. Each player takes their turn interleaving.

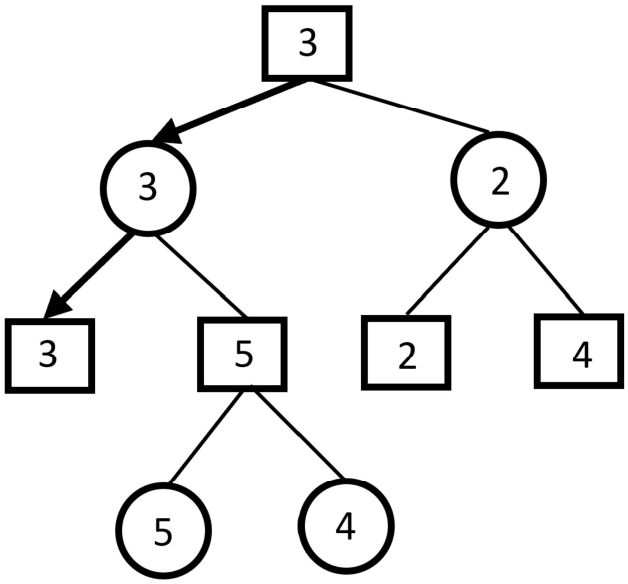


Figure 2-1: Illustration of a MIN/MAX game-tree. The MAX nodes are represented by square, while the MIN nodes are represented by circles.

A MIN/MAX tree illustration is shown in Figure 2-1. The squares represents MAX nodes, while the circles represents the MIN nodes. The edge represents to a transition between players' move. When a node represents a state where no more player can take a move, it is called the "terminal" nodes, which forms the outermost leaves of the tree. On the other hand, when a leaf nodes represent a state where there is still a possibility for it to be expanded, then it is thought to be the "internal" or "internal leaf" nodes[48]. The value v of internal nodes n are computed based on the following formulation:

$$
\begin{aligned}
MAXnode : v(n) &= \max_{c \in ch(n)} v(c) \\
MINnode : v(n) &= \min_{c \in ch(n)} v(c)
\end{aligned}
\tag{2.21}
$$

2.4.2 AND/OR Tree

AND/OR game-tree is a tree representation that consists of the AND and OR nodes. In the game-tree the OR nodes are usually representing a state resultant of the attacking player, while the AND nodes represent the opponent, although, it can also be represented the other way around [68].

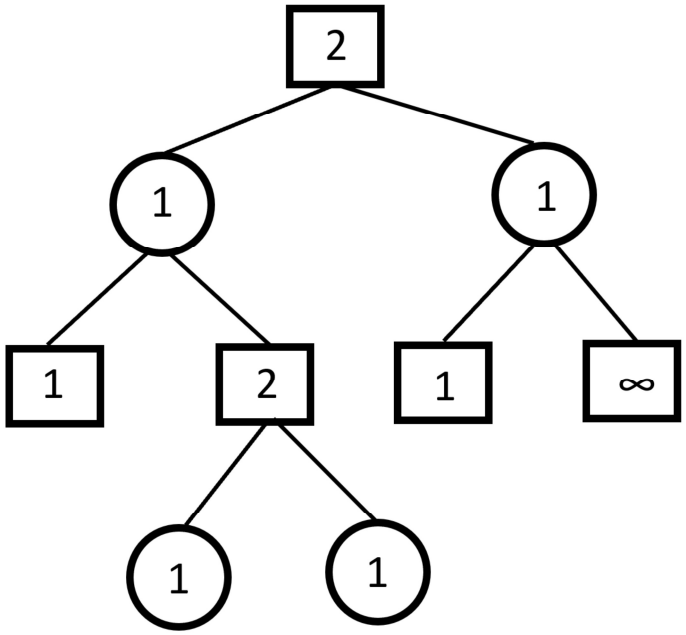


Figure 2-2: Illustration of an AND/OR game-tree. The OR nodes are represented by square, while the AND nodes are represented by circles.

The depiction of and AND/OR game-tree is shown in Figure 2-2. Each edge represents a transition between states, which usually is a move or a ply in-game. When a new node is added to the tree, regardless of it being AND or OR nodes, it is called an "internal" node or "internal leaf" node. When a leaf node represents the end state of a game, it is called "terminal" nodes [68, 49].

For a given game-tree with values assigned to the leaf nodes, the value v of internal

nodes n that is not a leaf node is determined by the following formula:

$$\begin{aligned} AND\ node : v(n) &= \bigwedge_{c \in ch(n)} v(c) \\ OR\ node : v(n) &= \bigvee_{c \in ch(n)} v(c) \end{aligned} \tag{2.22}$$

If the root node value is equal to TRUE or FALSE then the tree is considered to be "solved" and the value of the tree is the value of its root. If the root node value is TRUE, it is said to be "proved", otherwise, it is "disproved".

2.4.3 Balanced vs Unbalanced Game-tree

As the game is represented into a game-tree, it will retain the characteristics of a tree. One of them is the shape of the tree. The rules and the length of the game affect the shape of its game-tree. In algorithm and data structure, a balanced tree is defined to be a tree where the distance between the root and all of its leaves is not varied while an unbalanced tree is otherwise [6]. These definitions are then extended to the game-tree and allow it to be categorized into a balanced and unbalanced game-tree.

In the game whose rule allows the game to end early, in which the player experienced a sudden-death situation. Because of this, terminal nodes can occur in a varied depth of the tree, making the game-tree to be unbalanced. On the other hand, when a game rule does not allow to do so, the terminal nodes will appear in a similar depth, making the game-tree balanced.

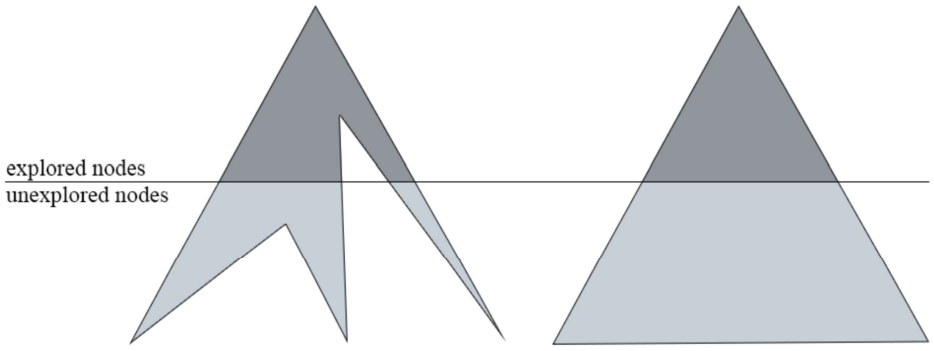


Figure 2-3: Illustration of information in unbalanced (left) versus balanced (right) game-tree.

An unbalanced game-tree provides more information from terminal leaves in the earlier stage of the game compared to a balanced game-tree (Figure 2-3). It is easier to find positions that lead to a win or a lose with high certainty at earlier game states than balanced game-tree.

2.5 Uncertainty in Entertainment: Game Refinement Theory

Based on the concepts of game progress and game information progress, a general model of game improvement is proposed [20]. It fills the gap between board games and sports [69, 79, 23] to the measure of its entertainment value. Game information progress presents the degree of certainty of a game's results in time or in steps. Let G be the winning player's score and T the total score of the game. Game progress $x(t)$ will be given as a linear function of time t with $0 \leq t \leq T$ and $0 \leq x(t) \leq G$, as shown in (2.23).

$$x(t) = \frac{G}{T}\, t \tag{2.23}$$

However, the game information progress given by (2.23) is usually unknown during the in-game period. Hence, the game information progress is reasonably assumed to be exponential. This is because the game outcome is uncertain until the very end of the game in many games. Hence, a realistic model of game information progress is given by (2.24).

$$x(t) = G(\frac{t}{T})^n \tag{2.24}$$

Here n stands for a constant parameter which is given based on the perspective of an observer in the game considered. Then acceleration of game information progress is obtained by deriving (2.24) twice. Solving it at $t = T$, the equation becomes (2.25).

$$x^{''}(T) = \frac{Gn(n-1)}{T^n} t^{n-2} = \frac{G}{T^2} n(n-1) \tag{2.25}$$

Hence, it is reasonably expected that the larger the value of (2.25) is, the more the game becomes exciting due to the uncertainty of the game outcome. Thus, we use its

root square, given by (2.26) as a game refinement measure for the game considered.

$$GR = \frac{\sqrt{G}}{T} \tag{2.26}$$

2.5.1 Motion in Mind

The game refinement theory plays an essential role in quantifying game sophistication by determining the rate of solved uncertainty along the game length where fairness, excitement, and thrills were identified [21, 20]. When a game is perceived as fair to the player, the player's experience is considered a sense of entertainment. Such a concept is explored further via the "motion in mind", which analogously defines the mind's subjective law of motions to the natural law of the physics [19]. Such an analogical link between motion in mind and the motion in natural physics is given in Table 5.3.

Table 2.1: Analogical Link Between Motion in Mind and Motion in Physics.

Notation	Game	Notation	Physics
y	solved uncertainty	x	displacement
t	total score or game length	t	time
v	winning rate	v	velocity
m	winning hardness	M	Mass
a	acceleration in mind	g	gravitational acceleration
$\vec{p}$	momentum of game	$\vec{p}$	momentum
E_p	potential energy of game	U	potential energy

The definition for each analogy is as follows [19]:

- **Mass:** In the concept of motion in mind, mass is defined as the level of challenge experienced by the player during the gameplay. It is related heavily to the frequency of risk in the game.

- **Velocity:** Defined to be the rate of solving uncertainty. It has an opposing relationship to m ($m = 1 - v$).

- **Acceleration:** Defined as the "gravitational acceleration in mind". It is an indicator of the gamified feeling occurring in the player's mind. It is determined that if the acceleration, GR, is located between 0.07 and 0.08, the player will feel gamified.

- **Momentum:** Defined as the mass of the object times the velocity. In the game context, refers to the balance between effort and ability given by the player.
- **Potential Energy:** Defined as the required amount of information needed by the player while progressing in the game, following the analogy definition of gravitational potential energy [19].

These definitions have been applied to calculate players' engagements in board games as well as scoring sports games. Comparison between three board games, Go, Chess, and Shogi has shown that the difference in the games' Motion in Minds units is closely related to the cultural aspects of the games' origin. On the other hand, comparing motion in mind values of Table Tennis, Basketball, and Soccer resulting in the ability to measure the said sports' engagements and its effect on the sports' popularity [19]. The definitions have also been used to establish the relationship between game-playing and rewarding experience based on its reward frequency variable [77].

As observed from the variable definitions of motion in mind, the central premises were made based on the uncertainties and the game's hardness, both related to the sense of entertainment in the game.

2.6 Chapter Summary

In this chapter, related works prior to the current thesis were introduced. Works related to the important keywords, namely the game-tree shapes, best-first search algorithms, and algorithms that aimed to exploit information from uncertainties were reviewed. In relation to the entertainment aspects, a measure of the entertainment aspect of the game that highly depends on the uncertainty in the game, the Game Refinement theory, is introduced. These studies are significant as it serves as the base to the research carried out in this thesis regarding the impact of uncertainty as well as the linking between search indicator and the measure of entertainment.

Chapter 3

Characterizing the Probability-based Proof Number Search

This chapter is an updated and abridged version of the following publications:

- A. Primanita and H. Iida. (2019). Nature of Probability-based Proof number Search. Sriwijaya International Conference on Information Technology and Its Applications (SICONIAN 2019), pp. 485-491, 2019.
- A. Primanita, M. N. A. Khalid and H. Iida. (2020). Characterizing the Nature of Probability-Based Proof Number Search: A Case Study in the Othello and Connect Four Games, Information Journal, vol. 5, no. 11, p. 264-279, 2020.

3.1 Chapter Introduction

In this chapter, we cover the result of experiments that leads to the Characterization of PPN-Search. PPN-Search previously has only been implemented in a simulated balanced game tree structure setting, thus, it is not enough to get the conclusion of its optimality in solving real-world games. As an attempt to discover PPN-Search characteristics, in this chapter, the algorithm is employed to solve two games with drastically different tree structures as test-beds. The first game is a real game with an unbalanced tree structure, which is the Connect Four game, while the second game is a real game with a balanced tree structure, which is the Othello game. This chapter will start with the overview of PPN-Search, followed by a brief introduction to both games, its rules, as well as its game

tree structure. Following that, the methodology of the research will be presented as well as the result and discussion. During the research, a problem was defined and was able to be alleviated with an added parameter. The introduction of this parameter will be covered in the discussion section. At the end of this chapter, the traits of PPN-Search will be presented as its conclusion, which also leads to the following chapter.

3.2 Probability-based Proof Number Search Algorithm

Probability-based proof number search (PPN-Search) is a game solver algorithm that aims to improve the idea of PNS. It uses an indicator called "Probability-based Proof Number"(PPN) to indicate the probability of proving a node [67]. The PPN of a root node is calculated based on the PPN of the current terminal and leaf nodes that is backpropagated to the internal nodes. Details of PPN-Search are specified in the following two subsections.

3.2.1 Probability-based Proof Number

A probability-based proof number (PPN) is a number that specifies the probability of a node to be proven in an AND/OR tree. There are three types of nodes in such a tree, viz. terminal node, leaf node, and internal node. All of these nodes have its own PPN ($n.ppn$) that is calculated based on the following formula:

1. If a node n is a terminal node:

   (a) if (n) is a winning node,

   $$n.ppn = 1 \tag{3.1}$$

   (b) if (n) is not a winning node,

   $$n.ppn = 0 \tag{3.2}$$

2. If a node n is a leaf node, then, let R be the winning rate as a result of game play-out, and θ is a small positive number close to 0:

   (a) if $R \in (0, 1)$

   $$n.ppn = R \tag{3.3}$$

(b) if $R = 1$,

$$n.ppn = R - \theta \tag{3.4}$$

(c) if $R = 0$

$$n.ppn = R + \theta \tag{3.5}$$

3. if a node n is an internal node, then

(a) if n is an OR node,

$$n.ppn = 1 - \prod_{n_c \in childrenofn} 1 - n_c.ppn \tag{3.6}$$

(b) if n is an AND node,

$$n.ppn = \prod_{n_c \in childrenofn} n_c.ppn \tag{3.7}$$

As seen on the formula, PPN that is assigned to an OR and AND internal node is the product of its child values. Figure 3-1 illustrates these calculations.

Because of the formulation, naturally, a node's PPN contains two based information derived from the current game tree. The first one is acquired from the terminal nodes (Eq. 1), in which the value is already certain and known, while the second is from the leaf nodes (Eq. 1). The second value is a value that represents uncertainties as it is calculated from the probability of winning or losing from the current position, thus, providing more information from part of the tree that is yet to be expanded.

While obtaining PPN of a node, every statistical value produced by a simulated game is accounted (Figure 3-1). This is different from the MIN/SUM rules used by MCPNS (Figure 3-2). The usage of MIN/SUM rules may disregard some of these values due to its rule.

To avoid confusion, in this thesis the first player is always an OR node. At any given node, PPN signifies the possibility of the node to be proved; a higher value means that it is more likely to be proved, which means that from the root position, it is more likely for the first player to win the game. PPN of the left side AND node in Figure 3-1 is the result of applying Eq. (3.7). Its PPN is the product of all its children. PPN of the OR

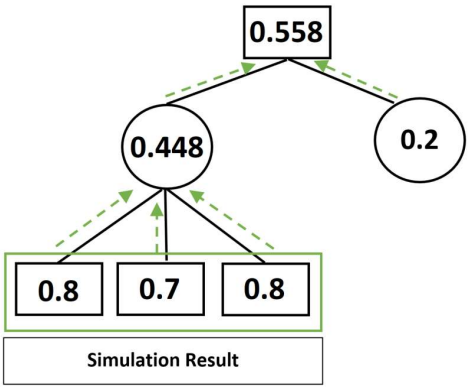


Figure 3-1: Illustration of PPN calculation in PPN-Search. OR nodes are displayed as square, while AND nodes are displayed as circle.

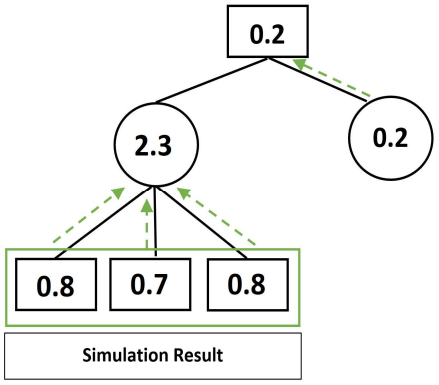


Figure 3-2: Illustration of *pmc* calculation in MCPNS. OR nodes are displayed as square, while AND nodes are displayed as circle.

node at the top of the subtree is the result of applying Eq. (3.6). As seen in the figure, children of all nodes affect the way PPN is calculated. The distinction of the probability rule application can be observed as the game-tree of MCPNS is displayed in Figure 3-2. MCPNS applies the AND/OR rule in updating its *pmc* value. The left side AND node's *pmc* is the sum of all its children, and the top OR node's *pmc* is the minimum *pmc* of its children. In this case, the *pmc* that has been previously calculated does not affect the value of the top OR node.

3.2.2 Algorithm of Probability-based Proof Number Search

The PPN-Search consists of four following steps:

1. **Selection**: For all nodes, select the child with maximum PPN at OR nodes, and child with minimum PPN at AND nodes. Regard these nodes as the most proving node (MPN) for expansion. The process of the selection step is given in Algorithm 1

2. **Expansion**: Expand the most proving node (Algorithm 2). In this phase, all available next moves from the MPN position is regarded as its child.

3. **Play-out**: For positions that are not already in the tree. Simulate the move in a random self-play mode until the end of the game. After several play-outs, the PPN-Search of expanded nodes is derived from Monte-Carlo evaluations. In this

step, if the result of simulations (R) is equal to 1 or 0, it is either reduced or added with a small value (θ) to differentiate it from the Leaf node (Algorithm 3).

4. **Backpropagation**: Update the PPN-Search from extended nodes back to the root, while following the AND/OR probability rules given in section 3.2.1 (Algorithm 4).

Algorithm 1 Selection algorithm

```
1:  procedure SELECTION(node)
2:      CreateNode(MPN)
3:      if node is OR node then
4:          MPN.ppn ← −∞
5:          for all child of node do                ▷ Find the child with highest MPN
6:              if child.ppn ≥ MPN.ppn then
7:                  MPN ← child
8:      else                                        ▷ node is AND node
9:          MPN.ppn ← ∞
10:         for all child of node do                ▷ Find the child with lowest MPN
11:             if child.ppn ≤ MPN.ppn then
12:                 MPN ← child
13:     return MPN
```

Algorithm 2 Expansion algorithm

```
1:  procedure EXPANSION(node)
2:      if node is an Internal node then
3:          Expand node
4:          PlayOut(node,player)                    ▷ Determine node's PPN
5:      else if node is a Leaf node then
6:          if First player wins then
7:              root.ppn ← 1
8:          else
9:              root.ppn ← 0
```

All of the steps are repeated until PPN of the root node reaches 1 or 0 (see Algorithm 5). If the PPN is equal to 1, the game is defined as solved.

Algorithm 3 Playout algorithm

```
1: procedure PLAYOUT(node, player)
2:     win ← 0
3:     for count ← 1 to simulation_num do
4:         Simulate game randomly until the End
5:         if Current player wins then
6:             win++
7:     R = win/simulation_num
8:
9:     if R is 1 then
10:        R ← R − θ
11:    else if R is 0 then
12:        R ← R + θ
13:    node.ppn ← R
```

Algorithm 4 Backpropagation algorithm

```
1: procedure BACKPROPAGATION(node)
2:     if node is an AND node then
3:         ppn ← 1
4:         for each child of node do
5:             ppn ← ppn ∗ child.ppn
6:     else if node is an OR node then
7:         ppn ← 1
8:         for each child of node do
9:             ppn ← ppn ∗ (1 − child.ppn)
10:        ppn ← 1 − ppn
11:    node.ppn ← ppn
```

Algorithm 5 Probability-based Proof Number Search

```
1: function PPN-SEARCH(root)
2:     Create root node
3:     while root.ppn ≠ 0 and root.ppn ≠ 1 do
4:         MPN = Selection(root)
5:         Expansion(MPN)
6:         BackPropagation(root)
7:     if root.ppn = 1 then
8:         return true                          ▷ Root is solved
9:     else if root.ppn = 0 then
10:        return false                         ▷ Root is unsolved
```

3.3 Game Test-Beds

To understand the presence of uncertainty and its effect on the PPN-Search algorithm, we conduct two experiments with two different game-tree structures. One unbalanced while the other is balanced.

3.3.1 Connect Four

Connect Four is a two-player perfect information board game and belongs to the broader class of k-in-a-row games such as tic-tac-toe or Gomoku [13]. The two players (red and yellow) take turns placing their pieces on the board, and the main goal is to line up (at least) four chips in an either horizontal, vertical, or diagonal manner on a standard 7×6 board (Figure 3-3).

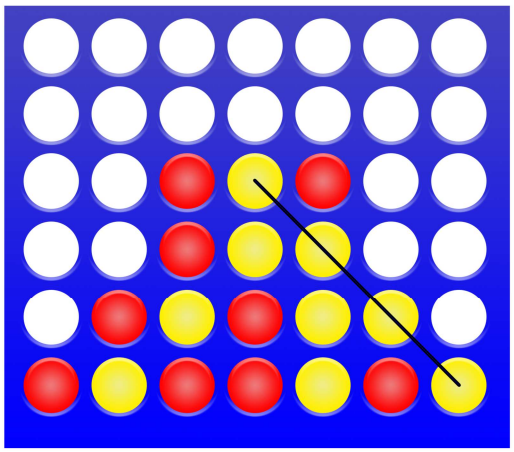


Figure 3-3: Illustration of Connect Four with a 7×6 board.

In Connect Four, gravity is essential so that the pieces fall as far to the bottom as possible. Thus, in each state, at most w moves are possible (a piece placed in one of the columns, if it is not yet filled). The default board size of Connect Four has been solved by Allen[1], and was then independently solved 15 days later in [1]. Its reachable states were roughly estimated in 7×10^{13} states.

The game can be won between 13-ply to 42-ply (the board is filled), making the depth of the search tree highly varied. There are also possibilities for sudden death moves to occur during the game, which made its game-tree structure unbalanced. In terms of

[1]reported in rec.games.programmer on Oct 1, 1988

complexity, Connect Four has a reasonably good game-tree due to its move limitation and board size.

3.3.2 Othello

Othello is a board game that requires two players to play, of which one player is playing black disks while the other plays white disks. Othello game has several characteristics:

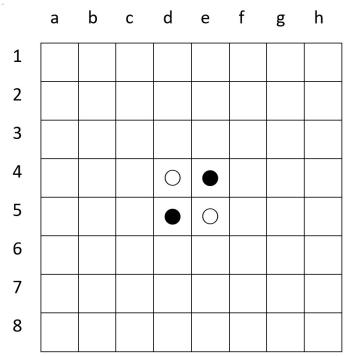
1. It does not have random elements in its rule.
2. Every gained advantage of a player is equal to the gained disadvantage of their opponent.
3. All of the players are perfectly informed of every previous event in the game, including the initialization stage. All of these characteristics define Othello as a deterministic, zero-sum, perfect information game.

Othello is generally played on an 8×8 board. At the start of the game, two disks of each color are placed diagonally in the center of the board (Figure 3-4(a)). The first to play is the player who plays a black disk, followed by a white player. Each player takes turns and makes a legal move on the board until there is no empty square in the board or the opponent is no longer able to make a legal move. A legal move in Othello is defined as placing a disk adjacent to an opponent disk so that the opponent's disk or a row of the opponent's disks is in between the new disk and another disk of the player's color. All of the opponent's disks between these two disks are considered as "captured" and turned over to match the player's color.

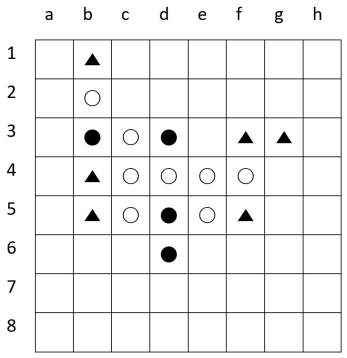
The state-space size of Othello is approximately 10^{28} [74], with maximum moves of 60 (30 for each player). It is a balanced game in which the player does not encounter "sudden death" while playing the game. Most of the time, the player is required to play the game until the board is full. Othello can be divided up into three phases: the opening, middle game, and end game. The opening game is defined as the first moves up until the 20^{th} to 26^{th} move. The end game is defined as the last 10-16, and the middle game is all states between the opening and the end game [75].

In playing Othello, some terms are used to explain the movements of the player. These terms are essential to Othello, as they affect how the game progresses [33]. The player's

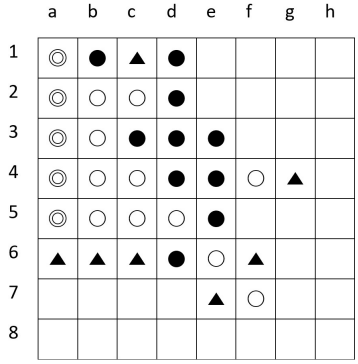
degree of mobility, or "mobility" in short, is used to describe the number of options that are available to a player in a particular stage of the game. In Figure 3-4(b), the available options for the black player are marked with a triangle. In this case, the degree of mobility of the black player is 6. Another relevant term in Othello is "stability," which defines a state where there are disks that cannot be captured by the opponent, no matter what moves they have made. Figure 3-4(c) is an illustration of an 8×8 Othello board after 22 moves, with the black player to move. In this stage, white disks marked with an inner circle are the stable disks of the white player. It is a definite disk that cannot be turned into black no matter what sequence is taken by the black player in the subsequent game.



(a) Initial position



(b) After eight moves, black is to play. The triangle marked the black player's mobility of six (six move options)



(c) After 22 moves, black is to play. The triangle marked the black player's mobility of seven (seven position options), and the white disks with marked inner circles are the stable disks (cannot be captured by the black player)

Figure 3-4: Illustration of Othello with an 8×8 board.

3.4 Experimental Setup

The experiments were performed by a computer with an Intel i5-8400 processor (2.81 GHz) using 8 GB of RAM, running Windows 10, on a 64-bit machine. The program for both Connect Four and Othello are built in C++. The experiments were performed in isolation from each other to ensure correctness and independent measurements. This means that only one position is solved by one algorithm at a time.

Three different algorithms, PNS, MCPNS, and PPN-Search, were given the task to solve each position independently. The parameters for each algorithm are given as follows: For MCPNS and PNS, the number of simulated moves is 60, while the $\theta = 0.01$ was set for the PPN-Search.

The experimental procedures were twofold (due to the nature of gameplay between the two games, Connect Four and Othello). For the first part of the experiment, 200 Connect Four positions were generated, where each position contains 12-ply of randomly generated moves. All algorithms are terminated when it expands into 35,000,000 nodes or when the time elapsed reaches 420 seconds, whichever comes first. The number of iterations performed, the number of nodes and visited nodes, and the time used to solve the position were measured.

For the second part of the experiment, 200 Othello positions were generated for three different stages. Each position contains either 18, 26, or 32 randomly generated moves (a total of 600 positions). In Othello's case, one move is defined as a position taken by one player or a half-ply. These positions are selected, which represent the opening, mid-opening, and middle positioning of the Othello game stage. All algorithms are stopped once it either expands into 8,000,000 nodes, or the time elapsed reaches 600 seconds. The number of completions for each position was measured. As for the second part of the experiment, the *pr* for PPN-Search is 0.0001.

3.5 Result and Discussion

3.5.1 Experimental Result on Connect 4

The result of the experiment is shown in Figure 4-2. All of the positions that successfully converged under the set limit displays the same conclusion. In the limited configuration, PNS performs the best, with 122 positions (solved and unsolved), followed by PPN-Search with 102 positions, and MCPNS with 82 positions (out of 200 positions in total). Note that PNS is bounded by the amount of memory (it stopped its search due to an excessive number of nodes visited), while PPN-Search and MCPNS are bounded by time (it stopped its search due to time limit set).

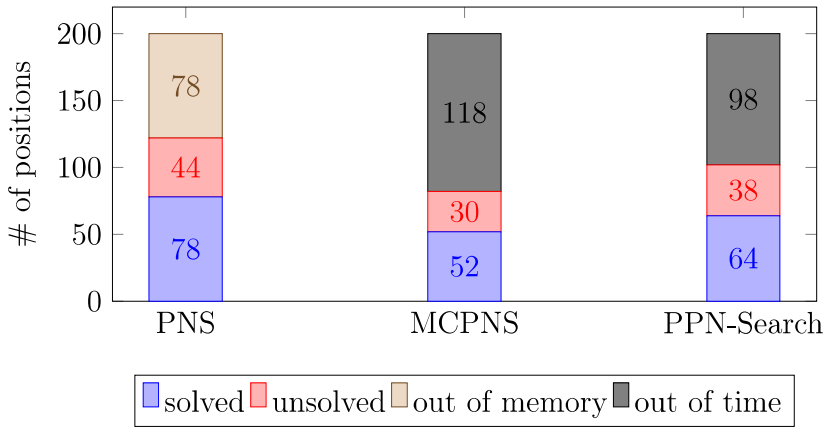
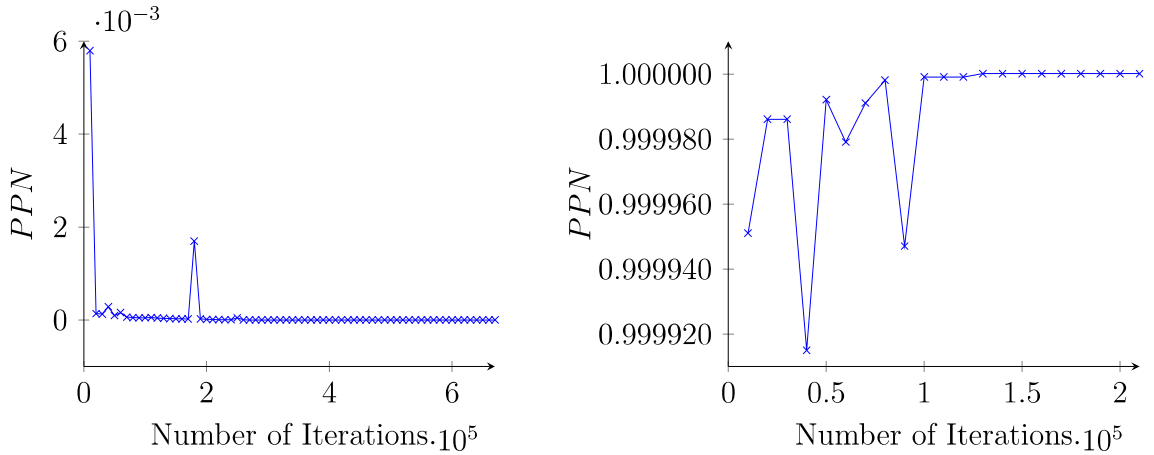


Figure 3-5: Number of Connect Four positions solved, unsolved, and out of bounds by PNS, MCPNS, and PPN-Search.

PPN number combines two kinds of information viz. information that is calculated with certainty (from the explored part of the tree) and information that is calculated without 100% certainty (from the unexplored part of the tree). As the combined value is utilized, it expedites the convergence of information. The PPN-Search visited a total of 2,295,856 nodes, while PNS and MCPNS visited 1,921,730 and 1,114,283 nodes, respectively, which corresponds to the position with the highest time difference between PNS and PPN-Search (322.345 s). This observation implies that PPN-Search was unable to thoroughly combine such information from both parts of the tree (sub-optimal performance), which can be demonstrated from the PPN value of the root for such a position (Figure 3-6(a)).

In the first 10,000 iterations, the root $PPN = 0.005794$ which also its peak value for the entire solving proces. The root PPN decreases and stabilizes (between the 70000^{th} iteration and the 170000^{th} iteration), while a sudden increase appears (180000^{th} iteration) with $PPN = 0.001699$ at the root, which subsequently stayed under 0.001 for the rest of the process. While $PPN \approx 0.000000$ by the 340000^{th} iteration, the algorithm keeps iterating until it reaches the 677765^{th} iteration (twice the expected iterations). This also affects the number of nodes needed to solve the position. By the 340000^{th} iteration, the total number of nodes explored is 1,205,010, while at the 677765^{th} iteration, the total number of nodes explored is also almost doubled (2,274,949 nodes).

Another solved position with a major time difference of 77.802 s between PNS and PPN-Search was also observed. For this position, PPN-Search visited a total of 661,783 nodes, while PNS and MCPNS visited 11,055 and 1,567 nodes, respectively. In this case, there is a large difference between total nodes visited by PNS and MCPNS. Observation of the PPN value of the root of this position (Figure 3-6(b)) shows a similar trend. The $PPN = 0.99995$ at the root for the first 10,000 iterations, which then fluctuates until the 100000^{th} iteration, where the $PPN = 1.00000$. Nevertheless, the algorithm makes 214,110 iterations.



(a) An example position with a time difference of 322.345 s

(b) An example position with a time difference of 77.802 s

Figure 3-6: PPN value of the root of two different positions for PPN-Search.

From these two observations, it is concluded that the prolonged search problem has occurred in the PPN-Search when it is employed to an unbalanced tree. This problem may occur due to the precision of the floating-point number, which contains a trailing

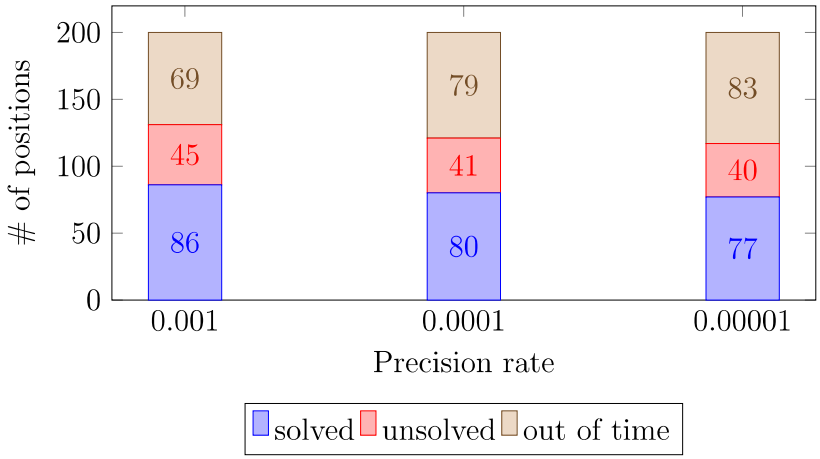
value. In other words, when the root PPN is shown to be equal to 0.000000, it might still contain a minimal trailing number. When this occurs, the implementation of PPN-Search does not recognize that it has been converged. In the program implementation, the root PPN value is compared against an integer (1 represents solved and 0 represents unsolved). The trailing numbers in the floating-point which are caused in nature by how the computer stores the floating points lead to an incessant loop since it is hard to reach such a precise integer number. As such, a precision rate (pr) value was added to the core PPN-search algorithm to ensure that the search can be timely optimized while not affecting the other parameters (Algorithm 6).

Algorithm 6 Probability-based Proof Number Search (with precision rate parameter)

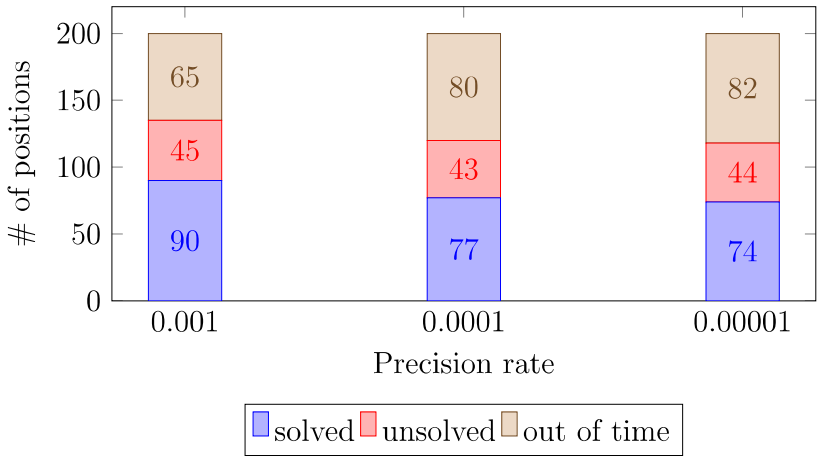
```
1:  function PPN-SearchEarch(root)
2:      Create root node
3:      while |root.ppn − 0| ≤ pr and |root.ppn − 1| ≤ pr do
4:          MPN ← Selection(root)
5:          Expansion(MPN)
6:          BackPropagation(root)
7:      if root.ppn is 1 then
8:          return true                                  ▷ Root is solved
9:      else if root.ppn is 0 then
10:         return false                                 ▷ Root is unsolved
```

A further experiment was conducted with two different values of θ ($\theta = 0.01, 0.001$) and three different pr values ($pr = 0.001, 0.0001, 0.00001$) were used, resulting into six different configurations (Figure 3-7(a) and Figure 3-7(b)). These configurations were applied to 200 generated Connect Four positions similar to those of the original experiment in Section 3.4.

The introduction of the pr parameter into PPN-Search implementation is proven to lead to an increase in the total converging positions in PPN-Search. It is found that for the case of Connect 4, the optimal configuration is when $\theta = 0.001$ and $pr = 0.001$ with a total of 135 converged positions. The optimal configuration is higher in quality compared to PNS (122 positions) and MCPNS (82 positions) from the earlier experiment. The average elapsed time and nodes explored are given in Table 3.1. The change of θ value does not significantly affect the average resources needed to converge, but an increase in the pr value results in a longer elapsed time and a higher number of traversed nodes. A small number of pr value leads to an increase in the total converged position. However, lowering the pr value to a rate where it is less precise than that of θ led to false conclusions.



(a) $\theta = 0.01$



(b) $\theta = 0.001$

Figure 3-7: Number of positions solved, unsolved, and out of bounds by PPN-Search.

In this case, the best pr value would be of the same with θ, or one rate above it (smaller pr).

Table 3.1: Average time and node for each PPN-Search configurations.

θ	pr	Average Time(s)$*$	Average Node$**$
0.01	no precision	251.180925	2,004,052.895
0.01	0.001	183.256315	1,323,718.975
0.01	0.0001	214.63364	1,582,187.985
0.01	0.00001	226.707205	1,667,750.875
0.001	0.001	174.90736	1,269,587.96
0.001	0.0001	217.987025	1,614,710.08
0.001	0.00001	229.07937	1,638,641.2

$*$less is better; $**$ more is better, relative to $*$

3.5.2 Experiment Results on Othello

The experiment results are given in Table 3.2. The experiment was conducted with consideration of the precision rate ($pr = 0.00001$) parameter, identified from the Connect Four experiment. All of the positions converged to the same conclusion unless it reached the termination condition (positions without any conclusion are designated as "out"). Based on the results, PPN-Search performs the best by solving most positions in every stage of the game (72% positions on average), followed by PNS (21.09% positions on average) and MCPNS (12.14% position on average).

Table 3.2: Experimental result of different algorithms applied to Othello positions.

Moves	Algorithm	Solved	Unsolved	Out	Completion* (%)
18	PPN-Search	52	56	92	54.00
	PNS	12	1	187	6.50
	MCPNS	5	4	191	4.50
26	PPN-Search	63	88	49	75.50
	PNS	19	16	165	17.50
	MCPNS	10	10	180	10.00
32	PPN-Search	82	91	27	86.50
	PNS	39	39	122	39.00
	MCPNS	6	37	157	21.50

*Completion = (Solved + Unsolved) / 200; even if the result concludes that it is unsolved, the algorithm is still able to conclude.

The PPN-Search performance is aligned with the findings from the use of PPN-Search on the P-game tree [67]. This further justifies that PPN-Search is indeed the optimal solver for games with a balanced tree structure. Comparing the performance of each best-first search algorithms in different stages of Othello shows an interesting result (Figure 3-8). Each algorithm displays different growth patterns. PNS and MCPNS start in almost the same position, however, toward the end stage of the game, although both exhibit an increase, MCPNS shows a sharper increase than PNS.

PPN-Search completion growth is the only one that displays a different trend. Compared to the sharp increase that is exhibited by MCPNS in the later stage of the game, the PPN-Search displays a sharp increase earlier, which is between the opening and middle game stage. While the increase in completion percentage in PNS and MCPNS is expected (more information revealed throughout the gameplay), the growth pattern of PPN-Search is a curious situation. Hence, a further experiment is conducted by adding more positions to solve.

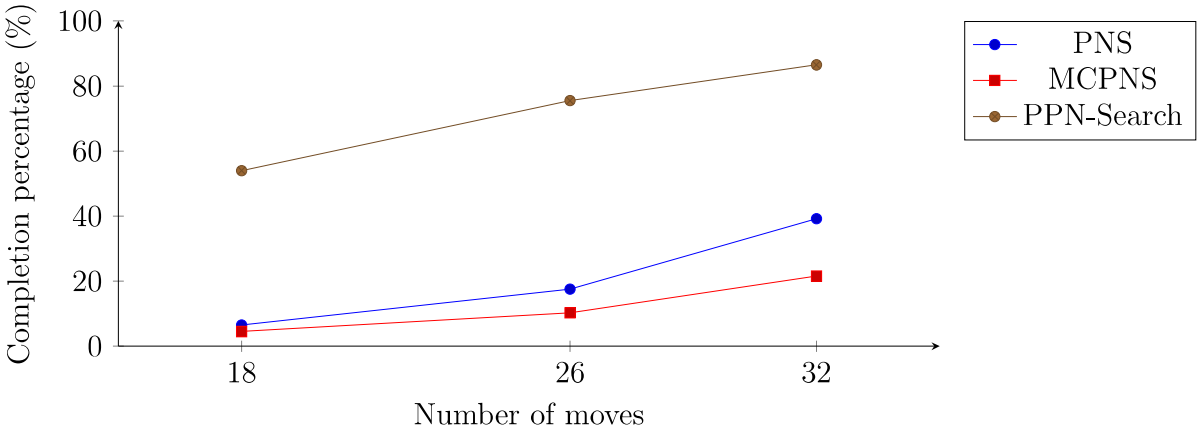


Figure 3-8: The completion percentage of PNS, MCPNS, and PPN-Search for different numbers of moves.

Similar to the original experiment setup (Section 3.4), while retaining similar experiment variables to ensure that the result is valid, another 200 Othello positions were generated for three additional stages (a total of six stages). The new stages considered 22 (opening), 36 (middle), and 40 (end) randomly generated positions; thus, illustrating the performance of PPN-Search throughout all stages of Othello (see results in Table 3.3). The reduction of the number of positions that are marked as "Out" shows the increase in

the completion percentage.

Table 3.3: Additional experiment result of applying PPN-Search to different stages of Othello positions.

Moves	Solved	Unsolved	Out	Completion* (%)
18	52	56	92	54.00
22	60	57	83	58.50
26	63	88	49	75.50
32	82	91	27	86.50
36	97	94	8	95.50
40	83	117	0	100.00

*Completion = (Solved + Unsolved) / 200; even if the result concludes that it is unsolved, the algorithm is still able to conclude.

The output of the game in the opening stage (move 18 and 22) is highly varied. This is caused by the many spaces available in the Othello board, leading to a large search space (Figure 3-9). However, the degree of mobility of each player can be small, as there are not as many disks on the board, which makes the PPN-Search convergence increase slowly. By the mid-opening stage of the game (between opening and middle stages) that is represented by move 26, one-third of the board has been filled with players' disk. The focus has likely shifted from the center position, where a strong Othello player increases their mobility by making a high number of varying moves. Thus, PPN-Search completion experienced sharp increases.

The middle stage (move 32 and 36) follows the player that competed to gain more stable disks that cannot be captured by their opponent. As more parts of the board are filled with Othello disks, the mobility is again reduced, which makes the completion of PPN-Search increase slowly. Finally, coming to the border of the middle and end stages (move 40), the number of positions available is minimal, and the sequence of moves taken becomes crucial. The PPN-Search degree of completion is increased from the previous stage and reaches 100% completion. At this stage, PPN-Search is said to solve most positions.

3.5.3 Discussion

In this chapter, we conducted two experiments with PPN-Search in order to see the effect of uncertainty on the best-first search algorithm. Two games were chosen due to its

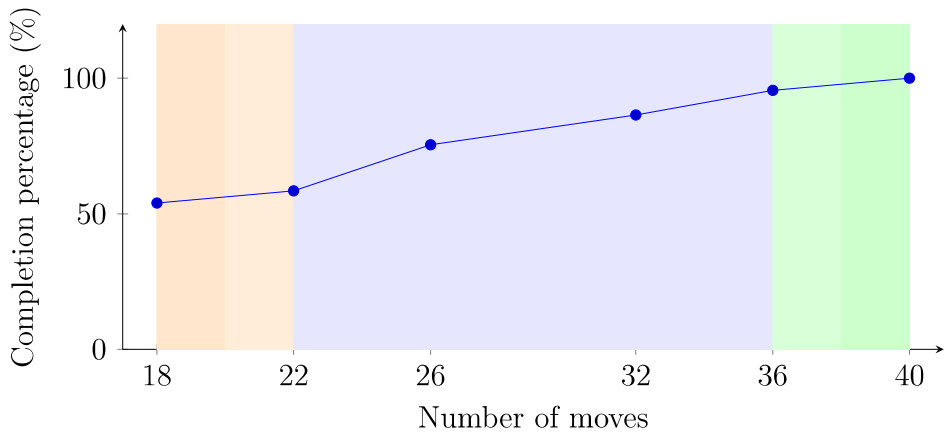


Figure 3-9: The completion percentage of PPN-Search in different stages of Othello. The left-most area colored in orange represents the opening stage of Othello, the middle area (blue) represents the middle game of Othello, and the right-most green area represents the end game of Othello.

distinctive game-tree structure. Connect Four, which represents a real case of a game with an unbalanced tree structure, and Othello, that represents a real case of a game with a balanced tree structure. Also, two related algorithms, PNS and MCPNS, were similarly applied to these games for comparison purposes. Based on such a setting, the experiments were conducted twofold.

Observation on the results yields from the first experiment with Connect Four showed that PPN-Search encountered a problem that leads to it performs sub-optimally. This behavior contradicted the notion that PPN-Search combined values that were calculated with certainty and uncertainty to obtain as much information as possible. It is seen as if PPN-Search needed more information instead. Upon further analysis of the progression of root PPN nodes in Connect Four games, the problem is able to be identified as the prolonged search problem. Both best-first search algorithms that PPN-Search was compared with the use of the integer-based backpropagation technique, while PPN-Search uses real numbers. The application of real number-based backpropagation calculation to floating-point causes the algorithm to underperforms.

A precision rate (pr) parameter is introduced to the PPN-Search algorithm to negate the precision problem. Experiments with various configurations show that the addition of the pr value increases the performance of PPN-Search without affecting its result accuracy. It was found that the closer the pr value is to θ, the better the PPN-Search performance.

However, a pr value lower than θ will worsen the PPN-Search performance. The results from the experiments demonstrate that PPN-Search with a pr value reduces the amount of explored nodes needed to solve a position by up to 57%. This situation implies that even a small amount of explored nodes allowed information from an unexplored area to be exploited and combined to reach the desired goal.

Observation on the results yields from the second experiment with different stages of the Othello game (opening, mid-opening, and middle stages) shows that an increasing amount of information, whether calculated with certainty or not allowed for more positions to be converged. It shows that PPN-Search combines both information results in a drastically better performance compared to the other two algorithms, and highlighting the importance of information from the unexplored nodes. Two algorithms in the experiment, the PPN-Search, and MCPNS, that uses information calculated from the uncertain area (unexplored nodes) experience sharp increases, albeit in a different stage of the game.

From only three distinct stages of Othello, the experiment was expanded to a total of six stages of Othello. The result from additional stages of Othello's positions serves as an indication of the PPN-Search behavior. In the first three position stages (opening to mid-opening) the player mobility was limited by the number of disks available on the board, however, the possible search space is enormous. In this case, the information coming from uncertainties becomes crucial for selecting the MPN. The selection of MPN in this stage is vital, as it leads to better mobility or thinning out the number of choices (a sharp increase in the PPN-Search completion percentage).

Compared to PNS and MCPNS, such an increase appears in the later stage (middle) when more information becomes available. This situation showed that PPN-Search was able to obtain such information earlier. For the middle to end game stages, the PPN-Search degree of completion slowly increases to 100% completion when approaching the end game. In this stage, players' mobility is again limited because the board is getting filled with player disks and the search space becomes small. In this stage, information from the explored nodes becomes more prominent since its information is more readily available. As such, information from unexplored nodes becomes prominent and highly critical for PPN-Search and lies between the opening and middle stages of the Othello game.

Table 3.4: Best performance search algorithms in different tree structure.

Tree structure	**Complexity**	**Best performance algorithm**
Balanced tree	Small(easy)	PPN-Search on P-game tree [67]
Unbalanced tree	Big(hard)	Df-pn in Go [26]
Unbalanced tree	Small(easy)	PPN-Search on Connect Four

Current insight suggested from the two experiments leads to a hypothesis that PPN-Search is suitable for a game that requires a long look-ahead strategy. The current state of the best performing algorithm on different tree structures relative to the result of current research is shown in Table 3.4. For a game with a hard problem, the current best performing algorithm is a depth-first proof number search (Df-pn). It is an expansion of PNS that is aimed to tackle larger problems.

3.6 Chapter Summary

PPN-Search has been introduced to solve two different tree structures (an unbalanced tree in Connect Four and a balanced tree in Othello), where PPN-Search yields better performance than the other related algorithms. PPN-Search application to Connect Four showed that the quality of the available information is critical compared to the quantity of the information, where a small amount of explored nodes, combined with the appropriate statistical information (or rather the precision of said information) of the unexplored nodes, can vastly improve the performance of the solver. Furthermore, the PPN-Search application to Othello demonstrates the importance of considering the appropriate "moment" to take advantage of the information from both the explored and unexplored nodes to solve more positions faster and earlier.

From the two experiments and results from the previous research, it can be concluded that PPN-Search is a best-first search algorithm used to solve game positions. It uses only one variable, PPN, as opposed to the two variables to show the state of proof and disproof in the game. PPN-Search is the most suitable to solve games with a balanced game-tree but also a good quality solver for a game with an unbalanced game-tree, especially when the computation resource is limited. PPN-Search regards uncertainties as information and able to utilize it using its backpropagation mechanism.

Chapter 4

Solving Single-Agent Game with Probability-based Proof Number Search

4.1 Chapter Introduction

The fourth chapter of the dissertation covers research carried out related to the Application of PPN-Search in Single Agent Game. PPN-Search has previously only been used to solve games with two players. In this chapter, PPN-Search is adapted to solve a single agent game. As there is a lot of different kind of game that requires only one player, the chapter is started with the definition of single-agent game in the research context. Following that, the newly adapted Single Agent PPN-Search algorithm is presented. Several adaptations have to be done to the representation of the game tree structure and PPN-Search framework as there is essentially only one agent playing the game, rather than two. These adaptations will be presented in a detailed manner. Following the introduction of the Single Agent PPN-Search, The game that was used as test-beds will be introduced. Its rules, as well as its single-agent game tree structure will be presented in the chapter. Following that, the methodology of the research will be presented as well as the result and discussion. At the end of this chapter, the advantage and disadvantages of implementing Single Agent PPN-Search will be presented as a conclusion.

4.2 Single Agent Game

A single-agent or single-player game is defined as a game that can be played by one player at a time [14]. Compared to the previously explored two-player games, in which the main purpose of each player is to defeat the opponent, a single-player game's main purpose is to achieve a certain objective, usually, in the form of a score, points, [71] or a desirable ending position. Currently, there are two main directions of study regarding single-player games. The first is to create an optimization algorithm that mainly focuses on achieving the highest score possible [71, 55, 59, 15]. The second direction is to analyze the differences in play experience between multiplayer and single-player games. Studies in the latter direction have focused on the different gaming experiences on different platforms [17, 11]. Single-player games, however, exhibit a multitude of usage modes beyond earning points or having a certain experience. Because of their nature of only having one player, they have been proven to be an effective platform for teaching [16] or real-world problem modeling [5].

Previous works in the field of heuristic search divided it into two sub-fields: the single-agent and two-agent search. Previously, both directions were studied separately from each other, however, Jonathan Schaeffer in his publication concludes that both areas are actually converging[58]. There is no essential difference in search-space properties as well as enhancements that occur in between the single-agent and two-agent search. Both are recognized as part of the many properties in the search space. Understanding the search algorithm through its application in both sub-fields benefits the improvement of search techniques as ideas that were proposed for a two-agent search does not have to be rediscovered for one-agent search and vice versa. Knowing this benefit, it is a logical direction to expand previously known search algorithms to the single-agent game.

4.3 Single Agent Probability-based Proof Number Search Algorithm

The PPN-Search has previously been applied to multiple games. However, it has not been applied to single-agent games. In order to expand the framework to single-agent games, we have to first look back at the general framework of PPN-Search.

Table 4.1: Mechanics In Single Player Games.

Game	**Mechanics**
SameGame	gravity between blocks[55]
8puzzle	limited move direction[47]
Sokoban	shape of the map[61]
2048	randomly spawn number tiles[38]

There are two main nodes in the PPN-Search, the OR nodes and AND nodes. In the two-agent search framework, the OR nodes are supposed to represent the maximizing agent, in which for each iteration, it will select the child with maximum PPN. The AND nodes, on the other hand, are supposed to represent the minimizing agent, in which for each iteration, it will select the child with minimum PPN as its MPN before back-propagating its result to the root[67].

In the realm of the single-agent games, the maximizing agent is clear, the sole player of the game tries to gain the most upper hand and reach their objective. However, the minimizing agent can be elusive as there is no clear opposing player that tries to gain upper hand in the game. If we look at the objective of the game, it is to reach an objective in form of a score, point, or an end position. Single-agent games usually possess a mechanic which used to hindered the player to reach their objective.

The inherent inhibitor mechanics in single-agent games (Table 4.1) are built so that the player would not be able to proceed further in the game, which equals to the point of the minimizing agent. Thus, in the case of Single-agent PPN-Search, the feature of these inhibitor mechanics are used in place of the second player.

4.4 Game Test-Bed: 2x2 2048

2048 is a popular game that was first published in 2014 by Gabrielle Cirulli[9]. It is a quite recent game but already found its popularity ground[38], not only because it is considered to be fun, but also, because it is thought to have a pedagogical property[42, 16].

The first iteration of the game is inspired by the game of Three![32] and played in a 4×4 board. Its initial position consists of 2 numbered 2 tiles positioned arbitrarily inside the grid. Players then can select from 4 directions to move the grid, namely, up, down, left, and right. Each move, all tiles will be shifted to the direction chosen and when there

are two same numbered tiles are adjacent in the direction, it will be combined, creating a higher numbered tile that is equal to the addition of those two tiles. After the shift, a new 2 or 4 numbered tiles will be spawned arbitrarily on the board. The mechanics continue until there are no more moves that can be taken.

The 2×2 2048 is a variant of the original 2048 games. In this iteration of the game, rather than a 4×4 grid, the grid size is smaller. In total, there are only 4 grids in the game. This means that the player mobilization is very limited compared to the bigger board. The rule of the initial position of the game remains the same, in which there are 2 number 2 tiles placed arbitrarily on the board. As the board size is smaller, the initial positions of 2x2 2048 can be enumerated into 6 positions (Figure 4-1).

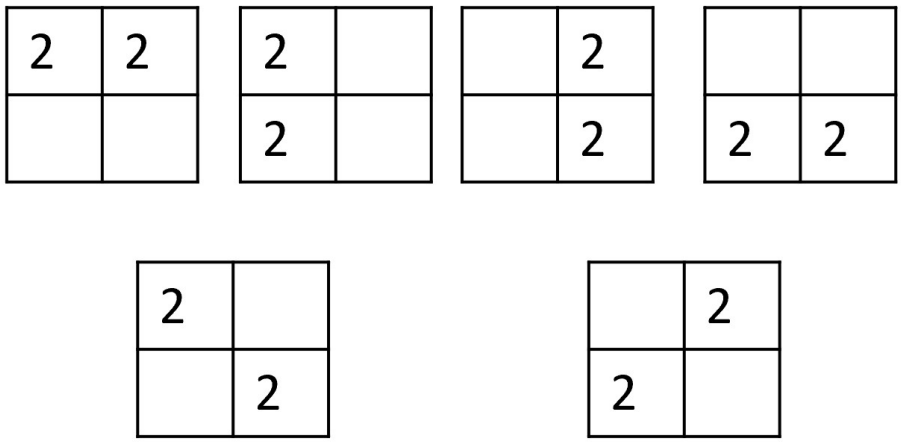


Figure 4-1: Illustration of initial positions ins 2x2 2048.

Using brute force enumeration, it is calculated that the highest reachable tile in 2x2 2048 is 32 tile[40].

4.5 Research Methodology

4.5.1 Application of PPN-Search to 2x2 2048

PPN-Search was developed with the idea of applying it into two-player games, however, its expansion to single-agent games can initiate its generalization. In this thesis, the single-agent game 2x2 2048 is chosen to be the game test-bed.

As the game nature is different from the previous implementations, modifications are

done to the algorithm that was introduced in section 3.2.1. The modification is done in the playout step, in which it is modified into algorithm 7.

Algorithm 7 Playout algorithm

```
1:  procedure PLAYOUT(node, player)
2:      win ← 0
3:      for count ← 1 to simulation_num do
4:          Simulate game randomly until End
5:          if node is an OR node then
6:              if objective reached then
7:                  win++
8:          else if node is an AND node then
9:              if no moves available then
10:                 win++
11:     R = win/simulation_num
12:
13:     if R is 1 then
14:         R ← R − θ
15:     else if R is 0 then
16:         R ← R + θ
17:     node.ppn ← R
```

The rest of the algorithms were implemented in the same way. Note that, even after the modification, the PPN-Search stays non-heuristic dependent.

Regarding the board states of the game, the OR nodes consist of at most 4 states, which represents the 4 move directions. The AND nodes consist of at most 6 states. At most, there can be 3 empty tiles, but, as there are two possible tiles, numbered 2 and 4, the states are doubled.

4.5.2 Experimental Setup

The experiment is configured to check the algorithms' performance in the single-player game framework. As 2x2 2048 only has 6 opening positions. All positions are used in the experiment, disregarding the mirroring positions. Two win objectives were set, the first one being the highest number tiled equal to 16 and the second being the highest number tiled equal to 32. Three different algorithms, PNS, MCPNS, and PPN-Search were given

the task to solve each position independently. For the experiment, parameters for each algorithm are as follows; For MCPNS and PPN-Search, the number of simulated moves is 60, and for PPN-Search, $\theta = 0.01$ and pr=0.0001. The number of iterations performed, of nodes, visited, was measured and recorded.

The experiments were performed by a computer with an Intel i5-8400 processor running at 2.81 GHz using 8 GB of RAM, running Windows 10, on a 64-bit machine. To ensure correctness and independent measurement, the experiments are performed sequentially. It means that only one position is solved by one algorithm at a time.

4.6 Result and Discussion

The result of the experiment is shown in Figure 4-2. All of the positions converged to the same conclusions, and all algorithms were able to solve the initial positions.

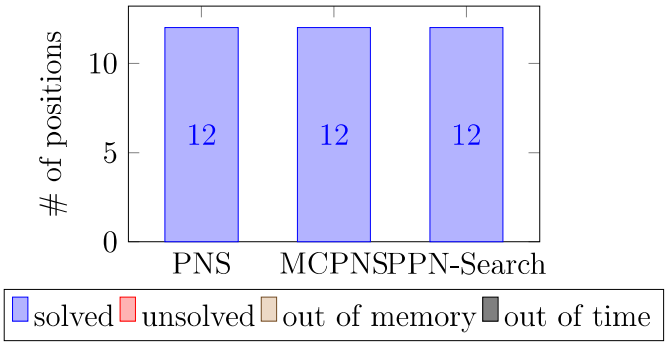


Figure 4-2: Number of 2x2 2048 initial positions solved, unsolved, and out of bounds by PNS, MCPNS, and PPN-Search.

As all of the algorithms were able to solve all the initial positions of 2x2 2048, it can be categorized to be ultra-weakly solved with the game-theoretical value being a win for the player. This means that given no change in rules and handicaps, a player that employs a perfect play strategy would be able two achieve the two objectives set for the game. The highest numbered tile equal to 16 and 32. This aligns with the combinatorial enumeration of the game[40].

To compare the effectiveness of the three algorithms, the number of visited nodes and iterations are compared and displayed in Table 4.2.

From the table comparison, the result of the quality measure between the algorithms

Table 4.2: Average iterations and nodes visited of PNS, MCPNS, and PPN-Search.

Algorithm	PNS		MCPNS		PPN-Search	
Win Objective	16	32	16	32	16	32
Iteration	6	11	4	4.67	2	2
Nodes	15.67	32.33	12.33	15.67	7.5	8.67

is aligned to that of the previous chapter. The PNS took the most resources to reach convergence, following by MCPNS. The algorithm that requires the least resources is PPN-Search in both scenarios.

Nodes traversed by each of the search algorithms represent the state of the board at one time. The combinatoric bounds of game states in 2x2 2048 are calculated to be 537 states[40]including the 6 initial positions. When mirroring positions as well as repeated positions are removed, it is calculated to be 59 possible states with 2 initial positions[39]. Compared to the nodes traversed to reach convergence, the best-first search algorithms all traversed fewer nodes than the available nodes. This shows the efficacy of the generalization.

The generalization of two-player PPN-Search to single-agent PPN-Search is considered as "vanilla" as it disregards the possibility of enhancements, such as transposition table or counting mirroring/rotating positions as the same state. Thus, it is compared to the combinatoric bound of the game. The two lesser performance best-first search algorithms were also modified to meet the same objectives. All algorithms were able to convergence while traversing less than 1% of all states, with PPN-Search showing to be the highest quality solver.

As the 2048 game is translated into a game-tree, it exhibits the behavior of an unbalanced-tree. The end of the game can be reached as early as the highest available tile equal to 8 to the highest quality tile equal to 32. thus making its game tree highly varied, and its structure unbalanced. The single-agent PPN-Search requires modification to its core algorithm. However, the modification does not eliminate its merit. In that, the algorithm remains to be non-dependent to the game heuristics as it only. In this case, the single-agent PPN-Search becomes the highest quality solver for a single-player game with an unbalanced tree.

The PPN of a node stores information from both its predecessors and likely successors,

relieving the need to explore unnecessary nodes. A human player should make an educated guess to win a single-agent game with random mechanics. Such a condition is reflected by the PPN, considering certain information from its predecessors and uncertain information from its likely successors similar to a human player using their prior knowledge to make an educated guess. The way the algorithm explores a game tree is similar to that of human intuition. This insight is further emphasized with the implementation of PPN-Search in a single-agent game. The 2048 game requires creating a higher numbered tile piece and considering the placements of the tile. In the 2x2 board, the player's mobility becomes limited, with only four grids available. It needs a clear look-ahead tile placement strategy for the player to reach the desirable highest tile. PPN-Search requiring the least resource shows that it is able to choose the most possible difficult node, making it the most suitable difficulty-ordering procedure.

4.7 Chapter Summary

In this chapter, the generalization from the two-player to single-agent best-first search algorithm was done towards the 2x2 2048 games. An experiment that compares the efficacy of three best-first search algorithms, PPN, MCPNS, and PPN-Search ultra-weakly solved the game. The game's theoretical value of 2x2 2048 is a win to the player. All three best-first search algorithms were able to reach convergence while traversing least than 1% states of the game without any enhancements, with PPN-Search using the least nodes. Making it the best fit solver in the experiment.

Employing the generalization method to 2048 also categorized the game-tree structure of 2x2 2048. It is a game with an unbalanced tree. The result of this chapter updates the table from the previous chapter relative to the result of current research into Table **??**. As the framework generalized, future enhancements to two-player PPN-Search should be able to be applied as well to the single-agent game.

Chapter 5

Finding the Critical Aspect of Single-Agent Game Using Single Conspiracy Number Indicator

This chapter is an updated and abridged version of the following publications:

- Y. An, A. Primanita, M. Khalid and H. Iida. (2020). Ascertaining the Play Outcome using the Single Conspiracy Number in GoBang. 2020 IEEE Conference on Games (CoG), pp. 658-661), 2020.

5.1 Chapter Introduction

This chapter covers research concerning a critical position in a single agent game. SCN is an expansion of the Conspiracy Number idea. Interestingly, although Conspiracy Number was used to show the difficulty of a position to be reached, the SCN was able to not only represent the difficulty of a position, but also showing the critical position that represents changes of player state of mind, or player strategy. In this chapter, SCN is expanded into a single agent game to find a critical position in such a game. The chapter is started with the introduction of SCN and preliminary results of implementing SCN into a two-player board game. It is then followed by an introduction to the games used in the research. Following that, the methodology of the research will be presented as well as the result and discussion. The effectiveness of the SCN in pointing out the critical position in a single

agent game will serve as the conclusion of this chapter.

5.2 Single Conspiracy Number

Proof Number Search (PNS) is successful in practical use for reducing the size of conspiracy numbers into two factors: proof number and disproof number. Inspired by this idea, a notion of SCN is proposed as a factor combining the features of proof number and conspiracy number [45, 65, 66].

SCN shows the difficulty of a node getting a value not less than T, where T is a threshold of legal MIN/MAX values. When T equals the maximum legal MIN/MAX value, the SCN is equivalent to the proof number. When T equals the minimum legal MIN/MAX value, the SCN is 0 because there is no difficulty for a node to get a value not less than the minimum. When T is between the maximum and the minimum legal MIN/MAX value, the SCN indicates the difficulty of a position getting a score not less than T.

Compared with the conventional evaluation way using MIN/MAX values, SCN is somehow more game-independent and expected to obtain more information on game progress patterns while showing the potential change of the MIN/MAX values. Therefore, SCN would be a good supplement to evaluation function values for analyzing game progress patterns.

Let $n.scn$ be the SCN of a node n, m be the MIN/MAX value of node n, and T is a threshold of legal MIN/MAX values. Then, the formalism of the SCN is given as follows:

- When n is a terminal node:
  - If $m \geq T$, $n.scn = 0$
  - If $m < T$, $n.scn = \infty$
- When n is a leaf node (not terminal):
  - If $m \geq T$, $n.scn = 0$
  - If $m < T$, $n.scn = 1$
- When n is an internal node:

– If n is MAX node, then

$$n.scn = \min_{n_c \in \text{child of } n} n_c.scn$$

– If n is MIN node, then

$$n.scn = \sum_{n_c \in \text{child of } n} (n_c.scn)$$

5.2.1 Application of SCN to Two-player Games

Xiang Qi

The foremost application of SCN is to track the game information progress pattern in Xiang Qi (or Chinese Chess). It is a two-player (BLACK and RED) without hidden information as well as non-deterministic elements, which made it part of abstract strategy games. Besides the differences of the rules, its piece behavior, and cultural reflections in Xiang Qi compared to Chess [34]. Xiang Qi is an extremely tactical game that requires to evaluate positions far ahead into the future. Additionally, Xiang Qi does not allow perpetual checks and resignation, stressing the importance of clear win (or lose) in the game. As such, Xiang Qi is the perfect test-bed to determine the effectiveness of SCN in estimating long-term outcomes based on the reflected game progress pattern of SCN [66].

As SCN is a lightweight indicator, it can be embedded in a different MIN/MAX application framework easily. In the case of Xiang Qi, it is embedded in the ElephantEye (Light) program, one of the most powerful AI engines of Chinese Chess. In the implementation, only the original alpha-beta search was used to compute the SCN. It is to ensure that all the information progress is obtainable. In the case of Xiang Qi, three types of positions, the tactical positions, drawn positions, and opening positions were analyzed [66].

Tactical positions in Xiang Qi are positions that are composed as a puzzle and not usually obtainable in real games. Applying SCN to these positions shows that the RED player (usually plays second) has a significant advantage. When compared to the evaluation function values, SCN was able to show that RED player positions stay stable amidst a threat by the BLACK player, ascertaining the advantage for RED player. The drawn

positions are positions in which both sides have an approximately equal chance to overtake one another. In this case, the value of the relative SCN (RSCN) frequencies plays a more important role as it shows the balance of advantage between both players. RSCN is a more rigorous evaluation of complex positions in the game progress patterns using SCN along each move positions made by both sides of players [66].

The last set of positions are the opening position. In the experiment, the positions are taken from handicap games as its outcome is said to be clearer than the other positions. In these positions, the Threshold value plays the most important role, as SCN can evaluate the positions on different levels where the thresholds correspond to relative look-ahead advantage. This process traverses the game tree repeatedly to obtain more information rather than depending on the static evaluation of current board pieces. The experimental results of SCN in Xiang Qi shows that SCN is more accurate in showing information in the game progress pattern rather than the value of evaluation function produced by the game's MIN/MAX tree [65, 66].

Checkers

Checkers is a two-player abstract strategy board that involves diagonal moves of usually, two contrasting color game pieces [64]. In this thesis, the two pieces are RED and WHITE. It involves diagonal moves by each player and captures by jumping over opponent pieces. A player is considered to win when all the opponent's pieces are captured or blocked, or draw when both sides cannot force a victory or it becomes repetitive [7]. Checkers is known to have high decision complexity, similar to Xiang Qi [57].

Similar to its application in Xiang Qi, SCN is implemented on a popular open-source checkers program, namely, Samuel AI. The Samuel AI is using $\alpha\beta$ search algorithm, and SCN was applied to the application while excluding the use of the transposition table. Again, to avoid information loss [7].

The experiment concluded that in an unbalanced game-tree structured game like Checkers, SCN of the root node is able to show that for winning position, the SCN value is low, while it is high for losing position[7]. Moreover, when a constant intermediate SCN value is found, it suggests a "loop" or repetitive play (which may lead to a draw).

GoBang

Gobang is a k-in-a-row game and a part of 15,15,5-game with added rules to maintain fairness. Solving a 5-in-a-row game with various board sizes shows that the larger the board is, the more advantageous it is for the first player. This situation is likely because as the board size increases, the first player's strategies also increase [18].

In GoBang, the players play their pieces (black or white) one by one to advance the game. The condition for victory is by making one's pieces to form the "five-in-a-row" either horizontally, vertically, or diagonally[1]. So, when the black player tries to connect his pieces into a line, his opponent (white) needs to attack by linking his white pieces by cutting down the potential threat of the black pieces. It is worth noting that, compared to a similarly played game (e.g., Gomoku), both GoBang and Gomoku try to form pieces of "five-in-a-row," and black starts first. However, pieces placed by the players in GoBang can be removed by surrounding two neighboring opponent's pieces with their pieces.

Compared to the previous implementations of SCN, GoBang is considered unique. It is because, albeit its simple rules, it has a huge state space. It is estimated that GoBang state space is 10^{105} [2]. By understanding the information progress in GoBang, it is hoped that it can be used for game analysis, as well as benefiting the expansion of SCN into a single-agent game.

The experiment in this case is unique as it uses data from level-3 players instead. The decision taken by the level-3 players was mapped into a MIN/MAX tree and has the SCN of the root node calculated [4].

The SCN value was observed to indicate both the game conditions (favorable or unfavorable) and progress continuity (ending or condition changes) [4]. Additionally, opposing and similar SCN trends imply "defensive" and "neutral" game situations, respectively. The result of the experiment also concludes that the Threshold value plays an important role in the result of observable game progress. A higher threshold relative to the maximum attainable MIN/MAX value was proven to be more effective in showing the game progress.

[1] `https://www.yucata.de/en/Rules/Gomoku`

5.3 Research Methodology

5.3.1 2048 as The Test-Beds of Single-Agent Game

With previous knowledge regarding the utilization of SCN in two-player games. The SCN is expanded into a one-player game. It is to find a broader, generalized view of what affects the critical positions identified by SCN as well as how uncertainty in game progress affects the way players progressing in the game.

The 2048 game was first published by its creator, Gabriele Cirulli, in 2014. The main objective of 2048 is to join the same tile currently available on the board to get a 2048 tile [9]. The original version of the game is played in a 4×4 board. The original game's starting position consists of 2 numbered 2 tiles (Figure 5-1 (left)). Each turn, the player has at most four options: to move the tiles to the right, left, up, and down.

Moving the tiles in a particular direction pushes the tiles in the opposite direction according to its direction. If there are two tiles with the same numbers next to each other, it will be merged, creating a number equal to the sum of the two tiles. A turn is only valid if it causes a change in the board, whether it is moving tiles or merging tiles. If a move that the player chooses does not change the board's state, it is not considered a turn. After a valid turn, a new tile will pop up at a random location on the board (Figure 5-1 (right)). The new tile can be a number 2 or number 4 tile.

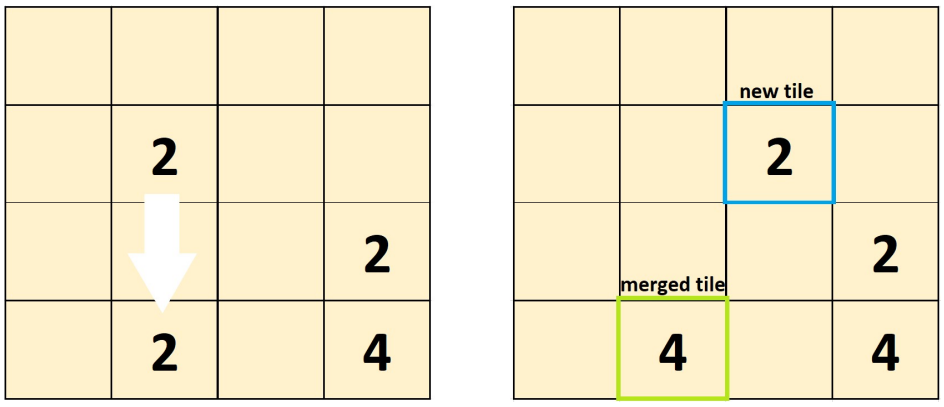


Figure 5-1: Illustration of tile merging from an initial state of 2048 game, where the player choose to move "down" (left). Then, a random tile appeared in a single turn of the 2048 game right after the merged tile (right).

The 2048 game is a stochastic with complete information game. The player can observe the entire state of the board at any point in the game. However, the player cannot predict

the entire game progress through the board's current state as, in every turn, a random tile will occur [62]. The appearance of the random tile gives a unique character to the game, as for each turn, there is only at most four options, but it has a large number of possible board states.

5.3.2 SCN and Expectimax Algorithm

Expectimax or expectiminimax algorithm is a variation of the MIN/MAX algorithm. It is developed specifically for games that depend not only on player skill but also on random chances [49]. Random chances in the game are represented as special nodes, namely, the chance nodes. The chance nodes appear alternately with the MIN/MAX nodes.

Compared to the previous implementation of SCN in the MIN/MAX framework, the treatment to get the m value is different. In this experiment, the m value is calculated based on the expectimax value of the nodes. However, the definition still holds as the m value is the node's inherent minimax value. For every leaf and terminal node in a tree, the nodes with expectimax value are calculated based on (5.1). For all other nodes other than leaf and terminal, the value of the Expectimax is calculated based on the Algorithm 8.

$$eval(state) = \sum_{i=0}^{3} \sum_{j=0}^{3} weight[i][j] \times board[i][j] \tag{5.1}$$

Algorithm 8 Expectimax

```
1: function EXPECTIMAX(state)
2:     if state is a MAX node then
3:         return highest eval value of child node;
4:     if state is a MIN node then
5:         return lowest eval value of child node;
6:     if state is a Chance node then
7:         return average eval value of child node;
```

The SCN value in a single-player game with random chances like the 2048 game is calculated only in the MAX and MIN nodes. In this case, the chance nodes are treated the same way as MIN nodes. The SCN calculation in the 2048 game is illustrated in Figure 5-2.

In 2048, the board's weight will keep increasing as the game progresses. It is because the weight is calculated based on the number of tiles and their position. The longer the

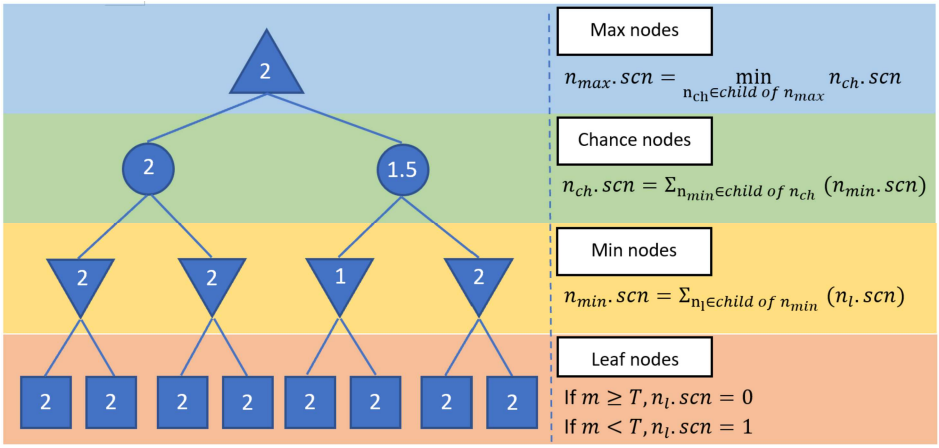


Figure 5-2: Illustration of SCN calculation in the Expectimax framework.

game plays, the higher the available numbered tiles will be. In the previous implementations, it is observed that the T value is vital in observing the game progress and was given a value that represents a middle stage between stability and instability of a state. In the 2048 game, the T value cannot be defined with only one single value. As the game progresses, the m value will keep increasing, although the state difficulty is the same; thus, a new way to calculate T is proposed.

In this thesis, the value of the new T is calculated following formula 5.2. The value n is an integer starting from 1. It is incremented by 1 every time a new highest number tile appears.

$$T_{new} = T_{old} * 2^n \tag{5.2}$$

For the 2048 games, every appearance of a new tile is thought to be a game information progress milestone. Thus, every step taken by the player is a step that aims to achieve the milestone. Because of this reason, the T value in the 2048 case is changed every time a milestone is achieved.

5.3.3 Experimental Setup

An experiment was conducted to test the effectiveness of SCN in representing the 2048 game progress. For the experiment, a specialized program written in Python was developed. The experiment was performed by a computer with an Intel i5-8400 processor running at 2.81 GHz using 8 GB of RAM, running Windows 10, on a 64-bit machine.

In this experiment, the algorithm is set to play 2048 game 20 times in 3 different configurations based on the Expectimax search depth (d). The 3 different configurations represent different player capabilities, from level-1 player ($d = 2$), level-2 player ($d = 3$), and level-3 player ($d = 4$). For each configuration, the Single Agent SCN for every turn is recorded.

The T value is set to two different setups to observe how the milestone system affects the game progress observation. The first setup ($T1$) is when the T changes every time a new higher number tile appears. The second setup ($T2$) is a benchmark when the T value is constant.

5.4 Result and Discussion

5.4.1 Result

Implementing the Expectimax algorithm with three different configurations yields results displayed in Table 5.1. It can be observed that the deeper the search depth (higher d), the higher the chance of getting to the highest tiles (2048 or greater). This situation also affects the average score and average steps taken to reach the result. Therefore, it can be said that the three configurations that represent level-1, level-2, and level-3 players have performed as expected.

Table 5.1: Result of Expectimax implementation on 2048 Game.

	Highest Tile				Average	
d	512	1024	2048	4096	Score	Steps taken
2	5	7	7	1	20212.00	1031.00
3	0	9	10	1	25163.00	1244.00
4	2	4	10	4	32866.25	1537.05

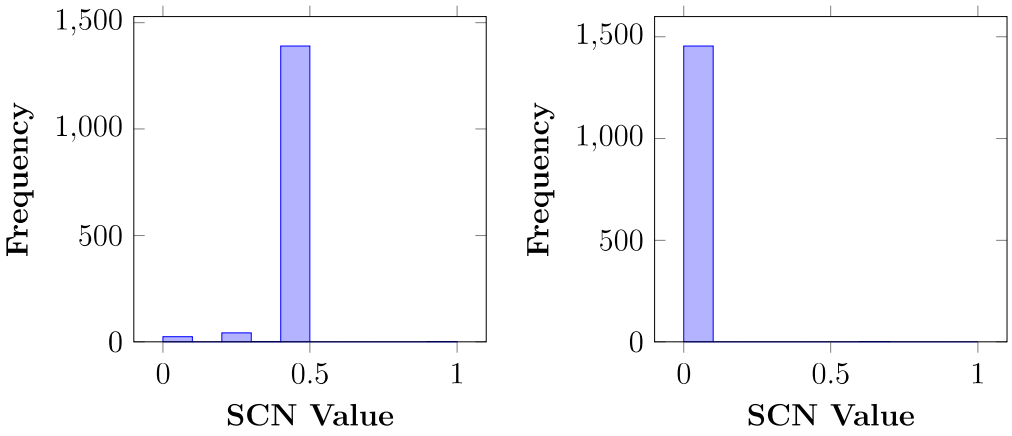
A comparison between the incremental threshold setup ($T1$) and constant threshold ($T2$) shows that the $T1$ setup is a better fit for a single-player game like 2048. It can be seen on the result (Figure 5-3) that the $T2$ only shows the early game progress. Once the m value reaches the Threshold, it never goes down again, resulting in the SCN value staying the same until the end of the game. Based on this result, the following discussion and experiments will focus on the $T1$ setup. For every game performed, the SCN of each

step were taken and recorded. This situation allowed the game progression to be observed for a different type of player.

The SCN for a simulated level-1 player ($d = 2$) is shown in Figure 5-4. As the T value is changed every time a new milestone is met, the SCN will drop to a low number when the transition occurred. Based on the previous observations of game progress in SCN[65, 7, 4], low SCN implies a stable position, in which the result of the game can be ascertained. In the case of 2048, it shows that level-1 player stays mostly in stable positions. This implies that the game could be short-run and ended early. However, due to the nature of 2048, a higher numbered tile can be obtained using the randomly spawned tile. This allows the level-1 player to continue the game, although they are in a constant near-ending stable position. In this case, the level-1 player is thought to not using their skill to achieve the milestone in the game, rather, they took a constant wager, which may end the game prematurely.

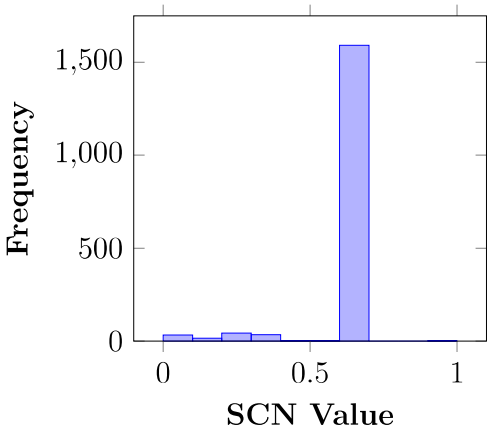
For a game with a simulated level-2 player ($d = 3$), the game's result for each category is shown in Figure 5-5. In these games, the SCN value fluctuates among several ranges of values. Indicating that the players are not taking the constant gamble, but applying a certain look ahead skill and strategy. For example, the player may employ a strategy such as arranging their higher pieces into one specific area on the board to have more flexibility in their movements. This condition corresponds to the high SCN value's frequency, indicating that the game was unstable, and the game became speculative. The game ended when the frequency of the lower SCN values is greater than the frequency of the higher SCN values.

For a game with a simulated level-3 player ($d = 4$), the result is shown in Figure 5-6. For these games, the fluctuation of the SCN is the highest (the SCN value has a greater variety). This condition shows that in every step, the player chooses a more stable position. It is justified based on the frequency of lower SCN values. However, a similar indication was also observed compared to the simulated level-2 player, where the greater frequency of high SCN values occurred (the game state mostly in unstable positions). The main difference with the level-2 player is that the level-3 player can find a way to overcome the unstable positions, which leads them to more prolonged game playing and possibly, better scores.

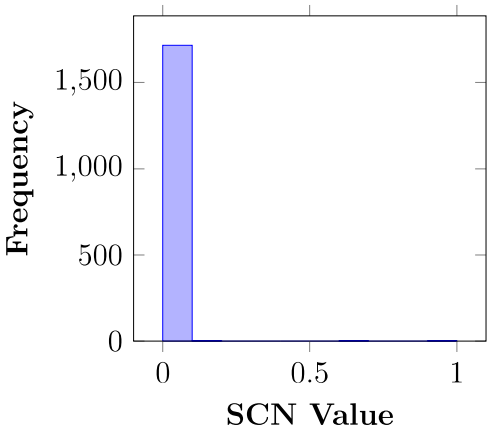


(a) $T1$ with $d = 2$.

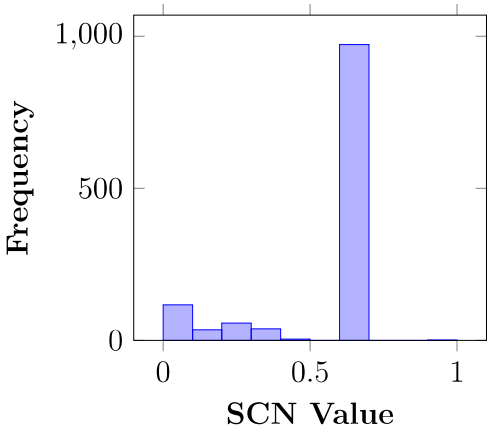
(b) $T2$ with $d = 2$.



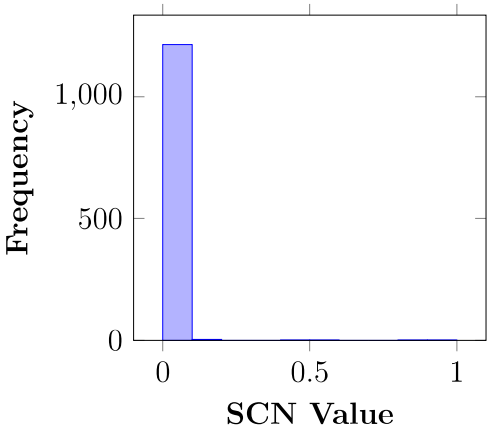
(c) $T1$ with $d = 3$.



(d) $T2$ with $d = 3$.

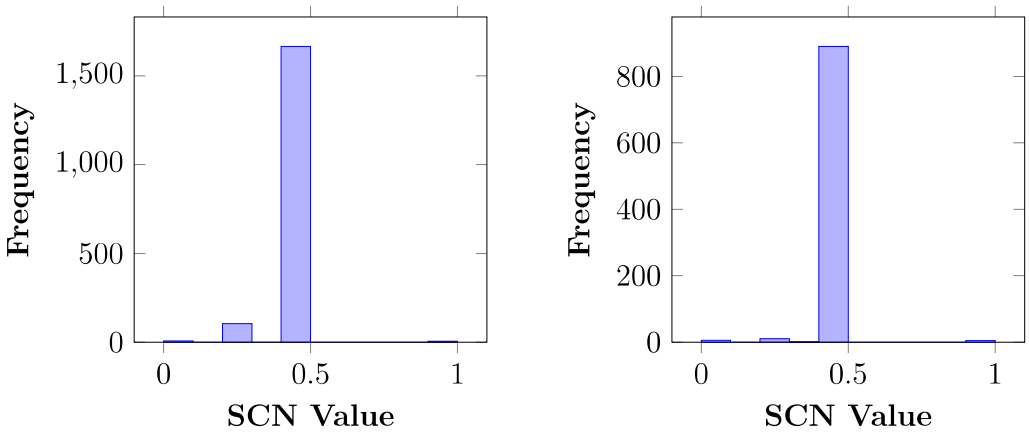


(e) $T1$ with $d = 4$.



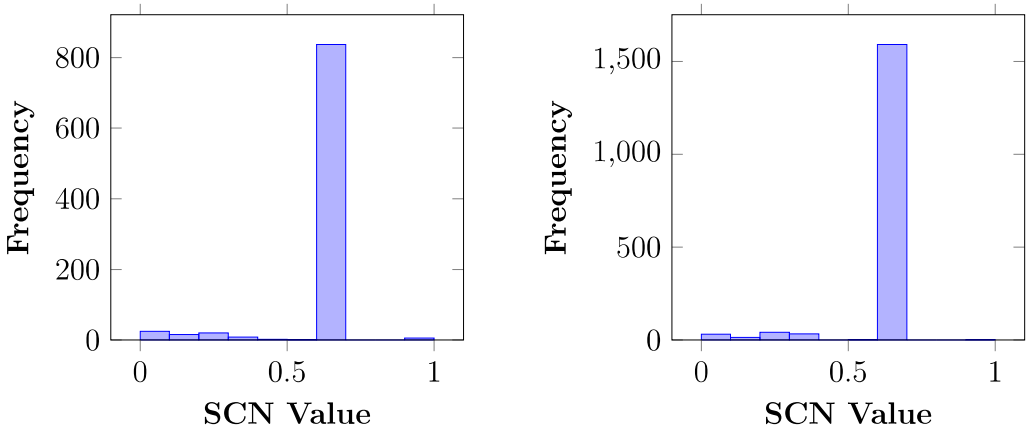
(f) $T2$ with $d = 4$.

Figure 5-3: Comparison of SCN of 2048 games calculated using threshold configuration of $T1$ (change with new high number of tile) and $T2$ (constant). In all of the results, the highest number on the board at the end of the game is 2048. Note that the SCN value is normalized between 0 and 1.



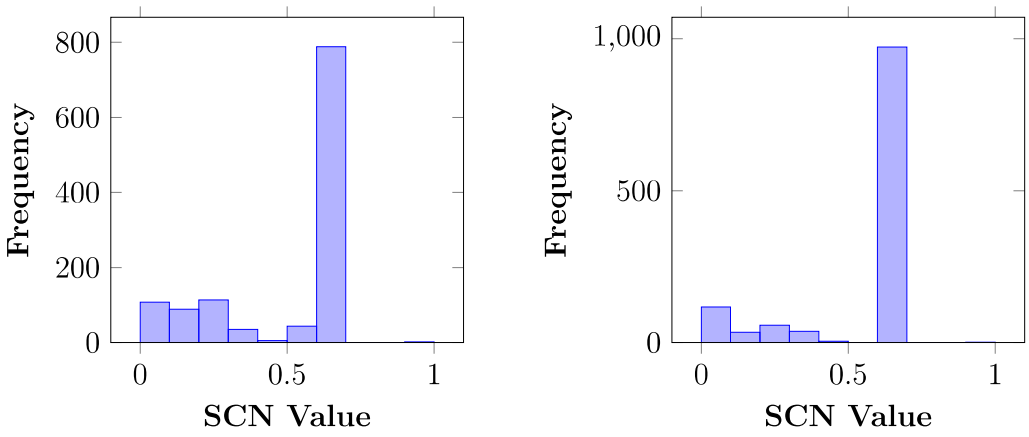
(a) 1024 is the maximum tile value reached (b) 2048 is the maximum tile value reached

Figure 5-4: Illustration of 2048 game with $d = 2$ (SCN value normalized between 0 and 1)



(a) 1024 is the maximum tile value reached (b) 2048 is the maximum tile value reached

Figure 5-5: Illustration of 2048 game with $d = 3$ (SCN value normalized between 0 and 1)



(a) 1024 is the maximum tile value reached (b) 2048 is the maximum tile value reached

Figure 5-6: Illustration of 2048 game with $d = 4$

5.4.2 Discussion

The 2048 game is a unique, single-agent, stochastic game with complete information because the current state is observable, but the entire game progress cannot be predicted. In this experiment, SCN was employed to observe the game progress for the 2048 game. By employing a MIN/MAX variant, the Expectimax algorithm, the SCN for 2048 games can be calculated.

In general, SCN cannot be applied directly in the 2048 game case. It is because 2048 has a different structure from the previous board games where SCN was implemented. The previous board games have limited turns and maximum score that the player can gain; however, 2048 is the opposite. Although it is virtually limited in the number of steps taken, the difference between low and high evaluation value is enormous. Thus, only having one threshold as the resolution for dividing what is considered a stable and unstable position is unrealistic in such a setting.

In this chapter, a new way to calculate the T value is proposed. It is calculated by dividing the game into several manageable milestones. Every time the milestones are reached, the T value is updated, making the game progress as a whole is observable. Additionally, the game-playing data collection was divided into two distinct objectives (reaching the maximum of 1024 and 2048 tiles) to analyze the game progress better and manage the possible SCN values.

5.4.3 Game Play Implications to level-3ise

The experiment was carried out while emulating different player ability levels to give different insights into how a game progresses. This situation is justified by the Expectimax result (Table 5.1). Results of the Expectimax playing 2048 games with different d shows that the lower the d value, the lower the average score, and average steps are taken. This condition is akin to a different player's ability level of playing the game. A simulated level-1 player has the lowest average scores and steps. They also have the lowest chance to reach 2048 with only 40% accuracy. The simulated level-2 player is right in the middle with 55% accuracy, and the simulated level-3 player is on the top with 70% accuracy to reach 2048 and more. The simulated level-3 player also shows a better result than the simulated level-2 player by four times higher possibility to reach number tile 4096.

Observing each of the simulated player SCN shows an interesting result. Each simulated player shows a different kind of playing style to reach the aimed value of 2048. For a simulated level-1 player (see Figure 5-4), most of the time, its SCN value is constantly high (relative to all possible values). Based on the previous research with SCN [65, 66, 7, 4], high SCN value typically indicates unstable position since the root may "conspire" to change much more frequently. Since level-1 player does not have enough capability to play the game effectively, their game's progression showed little intuition. In this case, their lack of skill is shown in the SCN frequency as they are continually choosing an unstable position. It is such as they took a constant gamble to reach the game objective.

For a simulated level-2 player (see Figure 5-5), some knowledge and level-3ise with the 2048 game were emulated. This notion is shown based on their SCN value having high fluctuations. There are times when they choose a better, more stable position than a level-1 player did. One interesting finding is that having a low SCN value most of the time indicates the game should end. A stable position is found when the SCN value is low, which means more child nodes in the game search tree with a high evaluation value. Having constant low SCN occurred when there are multiple high valued tiles; thus, merging low valued tiles will keep the game state stable. As such, it is expected that a player with enough knowledge of the 2048 games would gain the knowledge to achieve such a position. However, their lack of level-3ise leads them to a deadlock, which forces

them to end the game earlier.

A simulated level-3 player is shown the most fluctuations of SCN values compared to other players. This situation shows that an experienced player progresses through their game while changing their stance frequently from stable to unstable positions and vice versa. As previously stated, a player having a constant low SCN value occurs when there are more high-valued tiles in the current board (Figure 5-7(a)). The main difference from the level-2 player is that the level-3 player would match these high valued numbers, which results in more empty board positions, which resulted in high SCN value (Figure 5-7(b)). An level-3 player gradually chooses a more stable position, and as they hit the lowest, they would start to choose a less stable position, coming back to the more unstable state. However, the frequency of high SCN values for the level-3 player is highest, indicating that the game stayed in such a turnover position until the game cannot continue (forced end due to the board filled up to the point where no further move can be made).

512	256	128	64
16	8	8	32
8	16		2
2			4

(a) Board with low SCN value

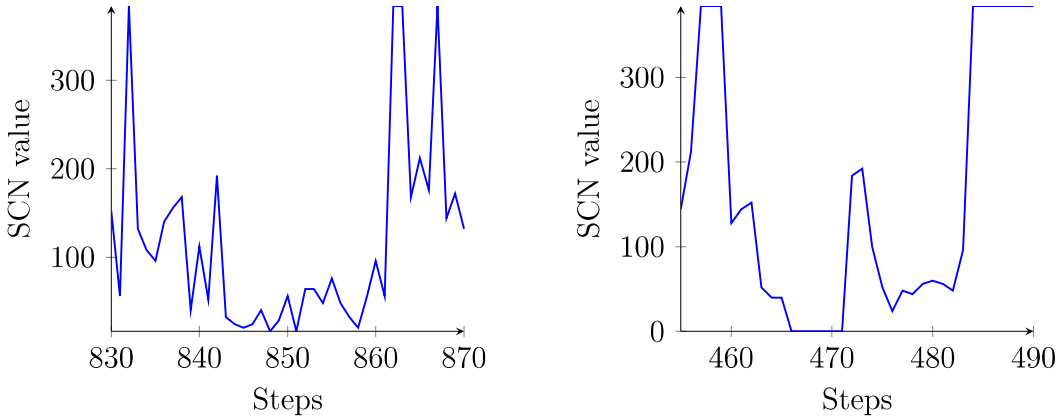
1024	8	4	2
4	16	2	
2	8		
4	4		

(b) Board with high SCN value

Figure 5-7: Examples of boards with different SCN values

Comparing the result of simulated level-2 and level-3 players shows the difference in their intuition and decision-making ability. The simulated level-3 player could escape over multiple "pitfalls" in their games, while the level-2 player would succumb to it. This situation leads to the "stability trap" in the 2048 game. In the current context, "stability trap" refers to a position on the board that is considered stable (promising), but do not favor the player (i.e., causing the game to end early). This trap can be demonstrated when a player chooses a more stable position on the board, constantly creating multiple tiles

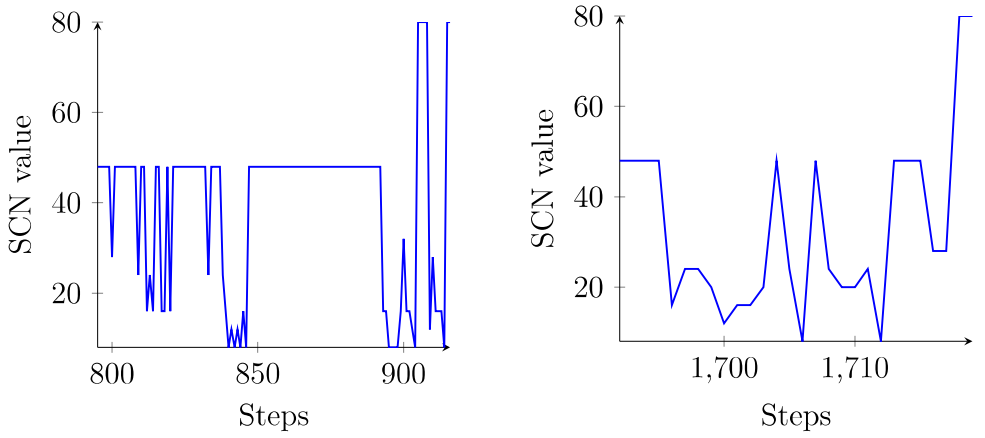
(with high numbers) and filling the board with untouchable tiles. An level-3 player would be able to escape this position (Figure 5-8). However, the level-2 and level-1 players have less chance of getting out of the trap, which lessens their chance to progress (Figure 5-9).



(a) 1024 is the maximum tile value reached

(b) 2048 is the maximum tile value reached

Figure 5-8: Depiction of the occurrence of the "stability trap" in the simulated level-3 player game ($d = 4$). level-3 players would be able to escape the position and continue their game, leading to a higher score and longer game



(a) 1024 is the maximum tile value reached

(b) 2048 is the maximum tile value reached

Figure 5-9: Depiction of the occurrence of the "stability trap" in the simulated level-2 player game ($d = 3$). level-2 players have a higher chance to be trapped in the position, leading to an early end of the game

5.5 Implications of Player Experience to Player Entertainment

As observed in the experiment with 2048, a player with different level-3ise levels may differently experience the game. A level-1-level player exploits the game's available uncertainty (chance), while an level-3 player exploits both their ability and the game's available uncertainty (skill and chance). The concept of entertainment measure that employs skill and chance as a measure of entertainment has previously been introduced in Section 2.5, known as the game refinement theory. In this section, the game refinement theory expansion into the Motion in Mind is introduced, where the link between SCN and Motion in Mind is established.

5.5.1 Applying Result of Different Player level-3ise to Motion in Mind concept

SCN is an indicator that shows the difficulty of a node getting a value not less than its threshold (T). The result of its application in a complete information single-player game, 2048, has shown that SCN was able to show the scale of challenge imposed towards the player with different levels of level-3ise. A player with a lower skill level depends on the game's uncertainties, shown by recurring low SCN value. Based on the notion of motion in mind in entertainment aspects, the speed of uncertainty solved (v_M) should correspond to the normalized SCN values.

Based on the hypothesis, the value of normalized SCN ($norm.SCN$) over steps k obtained from the previous experiments (Section 5.3.3) is used to calculate the value of motion in mind in 2048. To broader the observation, a new $d = 5$ value is calculated. It is to simulate the motion in mind value for level-4 player. The mass (m_M), not to be confused with SCN m value), velocity (v_M), momentum ($\vec{p}$), and potential energy (E_p) of the game 2048 is calculated based on Formula 5.3–5.7 [19].

$$norm.scn = \frac{1}{k}\sum_{n=1}^{k}\frac{n.scn_i - \min n.scn}{\max n.scn - \min n.scn} \tag{5.3}$$

$$v_M = norm.scn \tag{5.4}$$

$$m_M = 1 - v_M \tag{5.5}$$

$$\vec{p} = m_M - m_M^2 \tag{5.6}$$

$$E_p = 2 * m_M * v_M^2 \tag{5.7}$$

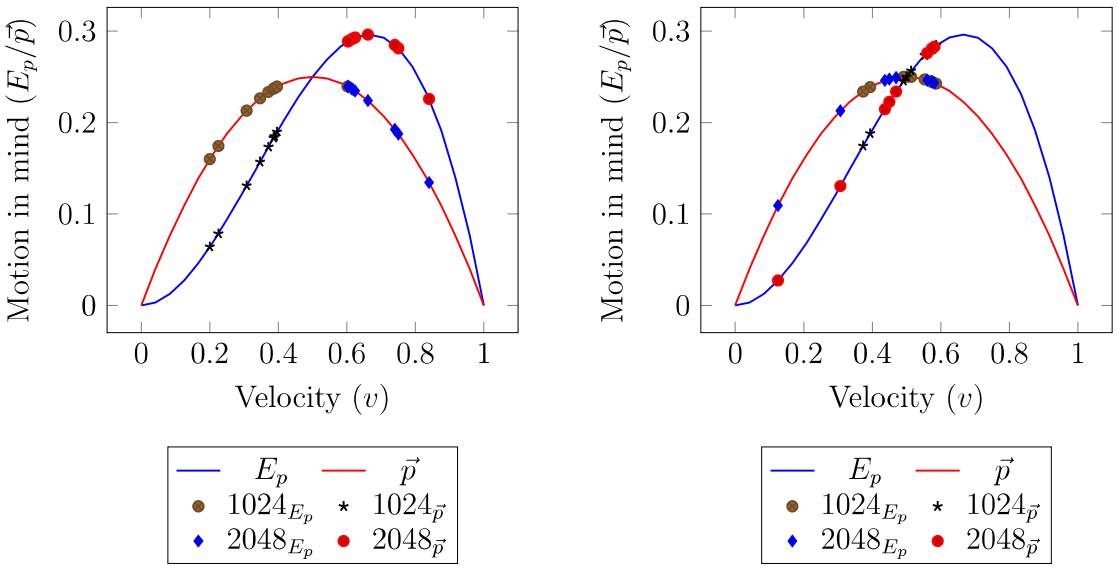
Table 5.2: Results of the Application of Motion in Mind to 2048.

d	Highest Tile	v	M	$\vec{p}$	E_p
2	1024	0.32782	0.67218	0.21517	0.14447
2	2048	0.32727	0.67273	0.21383	0.14411
3	1024	0.49763	0.50237	0.24464	0.24881
3	2048	0.45281	0.54719	0.22709	0.22439
4	1024	0.36248	0.63752	0.18039	0.16753
4	2048	0.44654	0.55346	0.21766	0.22072
5	1024	0.38209	0.61791	0.17264	0.18042
5	2048	0.28728	0.71272	0.18343	0.11764

Results displayed in Table 5.2 is based on k=steps taken from the initial game to the end. The motion in mind values of 2048 games, albeit from calculated from different depth, is similar. In average, $v = 0.38549$ while $M = 0.61451$, giving the average $\vec{p} = 0.20686$ and $E_p = 0.18101$. Looking at the value of its velocity, momentum, and potential energy, the value shows that the game possesses a high degree of fairness and equality [19]. In addition, the game is considered sophisticated enough where its fairly challenging to the level of effort made by player (high $\vec{p}$) except for the level-1 player with $d = 2$ by having $M > \frac{1}{2}$. It explains the popularity of 2048, as players with different degrees of skills can play the game with equal levels of fairness. Equal level of fairness in this case does not mean equal level of challenge. It means that that the level of challenge felt by player is suitable for their skill. Making the game equally engaging for different skill level players.

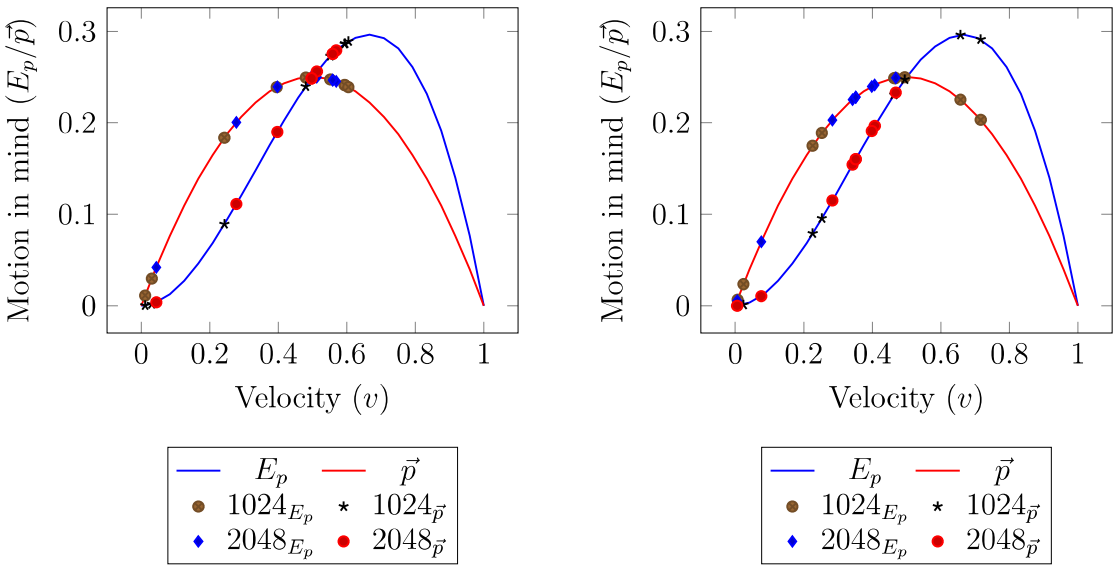
The result, however, does not show the different experiences between the skill level that the SCN is suggesting. In this case, the difference in skill level is observable because of the changing threshold. Thus, the *norm.scn* data were then observed separately, with k=steps taken to reach the next threshold. Results from the data segregation are displayed in Figure 5-10.

As seen in the results, although averaging the mass over the whole game resulting in similar value, the momentum and potential energy for the player in different skill levels suggests different game progress. For a level-1 player, the value is concentrated at a certain



(a) Simulated level-1 Player

(b) Simulated level-2 Player



(c) Simulated level-3 Player

(d) Simulated level-4 Player

Figure 5-10: SCN based Motion in Mind values for: (a) d=2, (b) d=3, (c) d=4, and (d) d=5.

point. The game is considered high-tension ($v \simeq 0.2$ with fair E_p and small $\vec{p}$). During the game, it reached the peak of potential energy, but not the peak of momentum. This situation means that the player reached the peak of certainty; however, it is very close to the game's end. A level-1 player never reaches the peak momentum shows that they do not reach the competitive balance in the game, which means that their game-playing relies on the board's random tiles.

A level-2 player starts their game in a similar position with a level-1 player. However, they later reach the peak of the game's potential energy and momentum. This situation implies that a level-2 player's game is engaging; however, towards the end of the game, they reach the peak of certainty and strike the game's competitive balance. Such a condition showed that their progress does not rely only on chance but also on their skill.

Game-playing progress for level-3 and level-4 player has a distinctive similarity, in which both start at very low momentum and potential energy values. This result is interesting because it shows that the initial game state for level-3 and level-4 players is considered challenging. The game started boring for both of the players; however, once they overcome several milestones, the game becomes more interesting. However, the time they took to reach the peak of momentum and potential energy is a bit later than the level-1 and level-2 player. The difference between game playing progress in the level-3 and level-4 player is towards the game's end. An level-3 player's momentum and potential energy are concentrated at the cross point between momentum and potential energy. This situation shows that they strike the competitive balance in most of their game-plays. Making the game more interesting. The level-3 player game ended while they are in the peak of the momentum, which means that their game-play is more balanced, while for a level-4 player, the game ended later, of which it comes down to the area where the game is considered challenging.

The notion that the game started at a boring position for the level-3 and level-4 simulated player is interesting as described by uncertainty. At the start of the game, it is deemed deterministic, of which it is inevitable for the player to reach the next milestone. For both of the players, the first milestone is considered deterministic, which means that it is undoubtedly reachable. Because of that, both players did not deem such part of the game as game-playing; instead, it is merely steps required to be taken to reach the real

"game". For the level-4 player, this condition continues until the second threshold, while for the level-3 player, they only exhibit it in the first threshold.

5.6 Chapter Summary

SCN indicates a position's difficulty of a game state getting to a specific future state (given by the threshold value). Since SCN displays the game progress related to the players' ability, observing the simulated player's SCN trend highlights the moment where skills and knowledge affect the game's result. This condition shows that regardless of the number of players in the game, SCN can be a powerful indicator for analyzing the game progress patterns.

Table 5.3: Analogical Link Between Motion in Mind and SCN in Games

Notation	Game	Notation	SCN
v	winning rate	v_M	average stability over time
m	winning hardness	m_M	average instability over time
$\vec{p}$	momentum of game	$\vec{p}$	momentum of normalized SCN
E_p	potential energy of game	E_p	potential energy of normalized SCN

Justifying such a claim leads to the application of the SCN to a stochastic single-agent game with complete information, 2048. Integrating the Expectimax algorithm as the bridge between two-player and single-player games, the game progress of the 2048 game can be analyzed based on different simulated players' perspective. Different patterns of SCN provided insights to identify and understand different effects of game-playing positions (corresponds with the case of the two-player game, such as the speculative play). As such, SCN provided a visual interpretation of the player progression in finding a good strategy according to their ability.

The principal understanding of SCN is that it indicates the difficulty of reaching a certain threshold, which aligns with the definition of mass in the concept of Motion in Mind. Applying normalized value of SCN throughout the whole games shows the overall game-playing progress of 2048 while observing it in a certain period based on certain

milestones allows for the observation of game playing of different player skill levels. From the discussion, it can be concluded that although not for all the Notation, the link between game-tree search indicator to the concept of entertainment in Motion in mind can be established (Table 5.3).

Chapter 6

Conclusion

Employing a best-first search algorithm to solve games leads to a better understanding of how information progresses throughout the game. In this thesis, we are specifically focusing on the expansion of Probability-based Proof Number Search to different game-tree structures. The experiments done on a two-player game with an unbalanced tree, namely Connect Four, shows that PPN-Search is a better performing algorithm in terms of convergence. This result is attained by introducing a precision rate (pr) parameter to negate the risk from the prolonged search problems, which hinder the convergence of the result. In order to generalize PPN-Search, we expanded the game into a one-player game framework. The expansion was done by modifying the game-tree to fit the framework of a single-player game. Experiment with a single-player game shows that PPN-Search is also the better performing algorithm in a single-player game with an unbalanced-tree structure. The single-agent PPN-Search required modification to its core algorithm so that it can combine uncertainty coming from the game mechanics itself. The modification, however, does not eliminate PPN-Search's main characteristic, which is the algorithm being a domain-independent algorithm.

Observation towards the effect of uncertainty value gained from "unexplored nodes" was done using a two-player game, namely Othello. It is concluded that information coming from uncertainties affects the ability of PPN-Search to solve positions for all stages of the Othello game. It is critical in the opening stages, as it allows more positions to reach convergence and prominent in the latter position as it leads to the rapid quality increase of the solver towards the later game stages. This shows that uncertainty plays

a crucial role in the best first-search algorithm that combines information both from explored and unexplored nodes.

Uncertainty affects the way a game is solved. On the other hand, it affects the way the game is experienced. The two terms seem to diverge; however, this thesis shows that in the end, the two can converge by establishing the link between the Single Conspiracy Number (SCN) indicator and the Motion in Mind concept. SCN is used as the leading indicator for a single-player game, namely, 2048. The proposed usage of SCN was to assess the difficulty of the game's current state to reach a particular MIN/MAX value over the specified threshold point. It is also useful to evaluate progress patterns, long term positions, and game-playing position. Its application to a single-player game, as opposed to two-player games, shows the game progress related to the simulated players' ability. It explicitly displays how the lower-skilled simulated player depends highly on the uncertainty that the game itself possesses. For the higher-skilled players, SCN indicates that these players could play the game by harnessing the game's uncertainty to benefit them, which leads to extended play and higher scores. This result shows that SCN is a powerful indicator in both the two-player and single-player game domain. As the framework has been expanded, it also allows for the application to problems related to skill level observation, for example, in the gamification framework.

The definition of SCN is aligned with the definition of velocity in the concept of Motion in Mind. Applying the normalized value of SCN over time in the game as the velocity value (difficulty rate), corresponding to the Motion in Mind concept, allows for the simulated player gaming experience to be mapped. Although the game's general experience is relatively uniform for all simulated players, a thorough observation towards the milestones in 2048, signaled by the changing of the threshold value in SCN, shows that different player skills experience the game differently. Lower level players perceived the game as a challenge and engaged in the game right from the start, while higher-level players experience a certain level of boredom, signaled by a very low momentum and potential energy at the start of the game. Higher-level players perceived the early game as a certainty, and it is a mundane step to be taken for them to reach the real game. Results from these experiments enable the link between game-tree search indicator and motion in mind.

In this thesis, both PPN-Search and SCN were expanded and first experimented with the single-player game. The generalization of the framework is beneficial to both of the algorithms. It is because when a framework is generalized, future enhancements for the two-player framework should be able to be applied as well to the single-agent game and vice versa.

Research carried out in the scope of this thesis is guided by the following objectives: (1) To find the optimal difficulty ordering procedure for game solver for different game-tree structures (2) To define the indicator for entertainment using a game-tree search framework and (3) To define the link between the game-tree search result and entertainment indicator in different game environments. The answer to the first question gained by using games as test-beds for PPN-Search, the algorithm combines all the available information in the tree by backpropagating it in a certain rule so that it can exploit the information. It is observed that the combination results in better quality solver that allows for a more efficient difficulty ordering both in balanced and unbalanced game-tree, as well as in the single-player framework. The second and third objectives is closely related to one another. Results from our research show that in the domain of the two-player game, SCN ascertains the incoming risks, while in the single-player game, it indicates the players' progress relative to their ability. The result from the application allows for the linkage between game-tree search and Motion in Mind theory. In the end, we can conclude that uncertainty plays an important part in games computation as well as game entertainment, even more, it has become a requirement in both computation and entertainment measure, as it impacts both the quality of games' computation result as well as how the game-playing experience is perceived.

Future works in this research direction are, but not limited to: (1) enhancement to the generalized best-first search algorithm to solve (very) hard games, (2) application to single-agent general game playing, (3) further investigation of the established link between game-tree search and Motion in Mind concept.

Bibliography

[1] L Victor Allis. A knowledge-based approach of Connect-Four-the game is solved: White wins. 1988.

[2] L Victor Allis. *Searching for solutions in games and artificial intelligence.* PhD thesis, Rijksuniversiteit Limburg, Maastricht, 1994.

[3] L Victor Allis, Maarten van der Meulen, and H Jaap Van Den Herik. Proof-number search. *Artificial Intelligence*, 66(1):91–124, 1994.

[4] Yuan An, Anggina Primanita, Mohd Nor Akmal Khalid, and Hiroyuki Iida. Ascertaining the Play Outcome using the Single Conspiracy Number in GoBang. In *IEEE Conference on Games (CoG2020)*. IEEE, 2020.

[5] Oliver Anteros Linnarsson and Sebastian Lorenzo. *Strategy Synthesis for Real-world Problems Modeled as Single-player Games of Imperfect Information.* PhD thesis, KTH ROYAL INSTITUTE OF TECHNOLOGY SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE, 2020.

[6] Paul E Black. Dictionary of algorithms and data structures. 1998.

[7] Sarot Busala, Zhang Song, Hiroyuki Iida, Mohd Nor Akmal Khalid, and Umi Kalsom Yusof. Single Conspiracy Number Analysis in Checkers. In *ICAART (2)*, pages 539–546, 2019.

[8] Guillaume M. J-B. Chaslot, Mark H. M. Winands, H. Jaap Van Den Herik, Jos W. H. M. Uiterwijk, and Bruno Bouzy. Progressive Strategies for Monte-Carlo Tree Search. *New Mathematics and Natural Computation*, 04(03):343–357, 2008.

[9] Gabriele Cirulli. 2048 Game Official Website, 2014. `http://https://play2048.co/`, Last accessed on 2020-08-24.

[10] Rémi Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*, pages 72–83. Springer, 2006.

[11] Alena Denisova and Paul Cairns. Player experience and deceptive expectations of difficulty adaptation in digital games. *Entertainment Computing*, 29:56–68, 2019.

[12] Nick Dyer-Witheford and Greig de Peuter. Postscript: Gaming while empire burns. *Games and culture*, page 1555412020954998, 2020.

[13] Stefan Edelkamp and Peter Kissmann. On the complexity of BDDs for state space search: A case study in Connect Four. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[14] Tracy Fullerton. *Game design workshop: a playcentric approach to creating innovative games*. CRC press, 2014.

[15] Konrad Godlewski and Bartosz Sawicki. MCTS Based Agents for Multistage Single-Player Card Game. In *2020 IEEE 21st International Conference on Computational Problems of Electrical Engineering (CPEE)*, pages 1–4. IEEE, 2020.

[16] Hung Guei, Ting-Han Wei, I Wu, et al. 2048-like games for teaching reinforcement learning. *ICGA Journal*, (Preprint):1–24, 2020.

[17] Anders Hansen, Kirstine Bundgaard Larsen, Helene Høgh Nielsen, Miroslav Kalinov Sokolov, and Martin Kraus. Asymmetrical Multiplayer Versus Single Player: Effects on Game Experience in a Virtual Reality Edutainment Game. In *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*, pages 22–33. Springer, 2020.

[18] Wei-Yuan Hsu, Chu-Ling Ko, Jr-Chang Chen, Ting-Han Wei, Chu-Hsuan Hsueh, and I-Chen Wu. On solving the 7, 7, 5-game and the 8, 8, 5-game. *Theoretical Computer Science*, 815:79–94, 2020.

[19] Hiroyuki Iida and Mohd Nor Akmal Khalid. Using games to study law of motions in mind. *IEEE Access*, 8:138701–138709, 2020.

[20] Hiroyuki Iida, Kazutoshi Takahara, Jun Nagashima, Yoichiro Kajihara, and Tsuyoshi Hashimoto. An application of game-refinement theory to Mah Jong. In *International Conference on Entertainment Computing*, pages 333–338. Springer, 2004.

[21] Hiroyuki Iida, Nobuo Takeshita, and Jin Yoshimura. A metric for entertainment of boardgames: its implication for evolution of chess variants. In *Entertainment Computing*, pages 65–72. Springer, 2003.

[22] Taichi Ishitobi, Aske Plaat, Hiroyuki Iida, and Jaap van den Herik. Reducing the Seesaw Effect with Deep Proof-Number Search. In Aske Plaat, Jaap van den Herik, and Walter Kosters, editors, *Advances in Computer Games*, pages 185–197, Cham, 2015. Springer International Publishing.

[23] Li Jiangzhou, Anggina Primanita, Mohd Nor Akmal Khalid, and IIDA Hiroyuki. Analyzing The Improvement Process of Table Tennis Using The Game Refinement Theory. In *Sriwijaya International Conference on Information Technology and Its Applications (SICONIAN 2019)*, pages 437–442. Atlantis Press, 2020.

[24] Mohd Nor Akmal Khalid, Umi Kalsom Yusof, Hiroyuki Iida, and Taichi Ishitobi. Critical Position Identification in Games and Its Application to Speculative Play. In *Proceedings of the International Conference on Agents and Artificial Intelligence-Volume 2*, pages 38–45. SCITEPRESS-Science and Technology Publications, Lda, 2015.

[25] Akihiro Kishimoto, Beat Buesser, Bei Chen, and Adi Botea. Depth-First Proof-Number Search with Heuristic Edge Cost and Application to Chemical Synthesis Planning. In *Advances in Neural Information Processing Systems*, pages 7226–7236, 2019.

[26] Akihiro Kishimoto and Martin Müller. Df-pn in Go: An application to the one-eye problem. In *Advances in computer games*, pages 125–141. Springer, 2004.

[27] Akihiro Kishimoto, Mark HM Winands, Martin Müller, and Jahn-Takeshi Saito. Game-tree search using proof numbers: The first twenty years. *Icga Journal*, 35(3):131–156, 2012.

[28] Julien Kloetzer, Hiroyuki Iida, and Bruno Bouzy. The monte-carlo approach in amazons. In *Proceedings of the Computer Games Workshop*, pages 185–192, 2007.

[29] Julien Kloetzer, Hiroyuki Iida, and Bruno Bouzy. A comparative study of solvers in Amazons endgames. In *2008 IEEE Symposium On Computational Intelligence and Games*, pages 378–384. IEEE, 2008.

[30] Levente Kocsis and Csaba Szepesvári. Bandit based Monte-Carlo Planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

[31] Richard E Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.

[32] B Kuchera. Why it took a year to make, and then break down, an amazing puzzle game.[Online; posted 06-February-2013], 2015.

[33] Immanuel Lazard. Othello Strategy Guide. `http://radagast.se/othello/Help/strategy.html`, 1993. 18.10.2019.

[34] Li Ma et al. Xiangqi vs Chess—The Cultural Differences Reflected in Chinese and Western Games. *Open Journal of Social Sciences*, 8(03):52, 2020.

[35] TA Marsland. A short history of computer chess. In *Computers, Chess, and Cognition*, pages 3–7. Springer, 1990.

[36] David Allen McAllester. Conspiracy numbers for min-max search. *Artificial Intelligence*, 35(3):287–310, 1988.

[37] John McCarthy. Chess as the Drosophila of AI. In *Computers, chess, and cognition*, pages 227–237. Springer, 1990.

[38] Rahul Mehta. 2048 is (PSPACE) hard, but sometimes easy. *arXiv preprint arXiv:1408.6315*, 2014.

[39] John Lees Miller. The Mathematics of 2048: Counting States by Exhaustive Enumeration.[Online;posted 12-December-2017, 2017.

[40] John Lees Miller. The Mathematics of 2048: Counting States with Combinatorics.[Online;posted 17-September-2017, 2017.

[41] Ayumu Nagai. Df-pn algorithm for searching AND/OR trees and its applications. *PhD thesis, Department of Information Science, University of Tokyo*, 2002.

[42] Todd W Neller. Pedagogical possibilities for the 2048 puzzle game. *Journal of Computing Sciences in Colleges*, 30(3), 2015.

[43] Nathan Nossal and Hiroyuki Iida. Game refinement theory and its application to score limit games. In *2014 IEEE Games Media Entertainment*, pages 1–3. IEEE, 2014.

[44] Jakub Pawlewicz and Ryan B Hayward. Scalable parallel DFPN search. In *International Conference on Computers and Games*, pages 138–150. Springer, 2013.

[45] Jakub Pawlewicz and Ryan B Hayward. Conspiracy number search with relative sibling scores. *Theoretical Computer Science*, 644:127–142, 2016.

[46] Anggina Primanita and IIDA Hiroyuki. Nature of Probability-based Proof Number Search. In *Sriwijaya International Conference on Information Technology and Its Applications (SICONIAN 2019)*, pages 485–491. Atlantis Press, 2020.

[47] Alexander Reinefeld. Complete Solution of the Eight-Puzzle and the Bene t of Node Ordering in IDA*. In *International Joint Conference on Artificial Intelligence*, pages 248–253, 1993.

[48] Ronald L Rivest. Game tree searching by min/max approximation. *Artificial Intelligence*, 34(1):77–96, 1987.

[49] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 2002.

[50] Abdallah Saffidine and Tristan Cazenave. Developments on product propagation. In *International Conference on Computers and Games*, pages 100–109. Springer, 2013.

[51] Abdallah Saffidine, Nicolas Jouandeau, and Tristan Cazenave. Solving breakthrough with race patterns and job-level proof number search. In *Advances in Computer Games*, pages 196–207. Springer, 2011.

[52] Jahn-Takeshi Saito, Guillaume Chaslot, Jos WHM Uiterwijk, and H Jaap Van Den Herik. Monte-Carlo proof-number search for computer Go. In *International Conference on Computers and Games*, pages 50–61. Springer, 2006.

[53] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[54] Maarten PD Schadd, Mark HM Winands, Jos WHM Uiterwijk, H JAAP VAN DEN HERIK, and Maurice HJ Bergsma. Best play in Fanorona leads to draw. *New Mathematics and Natural Computation*, 4(03):369–387, 2008.

[55] Maarten PD Schadd, Mark HM Winands, H Jaap Van Den Herik, Guillaume MJ-B Chaslot, and Jos WHM Uiterwijk. Single-player monte-carlo tree search. In *International Conference on Computers and Games*, pages 1–12. Springer, 2008.

[56] Jonathan Schaeffer. Conspiracy numbers. *Artificial Intelligence*, 43(1):67–84, 1990.

[57] 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.

[58] Jonathan Schaeffer, Aske Plaat, and Andreas Junghanns. Unifying single-agent and two-player search. *Information Sciences*, 135(3-4):151–175, 2001.

[59] Arta Seify and Michael Buro. Single-Agent Optimization Through Policy Iteration Using Monte-Carlo Tree Search, 2020.

[60] Claude E Shannon. XXII. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

[61] Yaron Shoham and Jonathan Schaeffer. The FESS Algorithm: A Feature Based Approach to Single-Agent Search. In *2020 IEEE Conference on Games (CoG)*, pages 96–103. IEEE, 2020.

[62] Johan Sijtsma, Nils H Jansen, and Frits W Vaandrager. Creating a Formal Model of the Game 2048. 2020.

[63] 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, 2016.

[64] Amarjeet Singh and Kusum Deep. Use of evolutionary algorithms to play the game of checkers: Historical developments, challenges and future prospects. In *Proceedings of the Third International Conference on Soft Computing for Problem Solving*, pages 55–62. Springer, 2014.

[65] Zhang Song and Hiroyuki Iida. Using single Conspiracy Number to analyze game progress patterns. In *2017 International Conference on Computer, Information and Telecommunication Systems (CITS)*, pages 219–222. IEEE, 2017.

[66] Zhang Song and Hiroyuki Iida. Using single conspiracy number for long term position evaluation. *ICGA Journal*, 40(3):269–280, 2018.

[67] Zhang Song, Hiroyuki Iida, and H Jaap Van Den Herik. Probability based Proof Number Search. In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence, ICAART 2019*, 2019.

[68] David Stern, Ralf Herbrich, and Thore Graepel. Learning to solve game trees. In *Proceedings of the 24th international conference on Machine learning*, pages 839–846. ACM, 2007.

[69] Arie Pratama Sutiono, Ayu Purwarianti, and Hiroyuki Iida. A mathematical model of game refinement. In *International Conference on Intelligent Technologies for Interactive Entertainment*, pages 148–151. Springer, 2014.

[70] Maciej Świechowski. Game AI Competitions: Motivation for the Imitation Game-Playing Competition. In *2020 15th Conference on Computer Science and Information Systems (FedCSIS)*, pages 155–160. IEEE, 2020.

[71] Jun Tao, Gui Wu, Zhentong Yi, and Peng Zeng. Optimization and Improvement for the Game 2048 Based on the MCTS Algorithm. In *2020 Chinese Control And Decision Conference (CCDC)*, pages 235–239. IEEE, 2020.

[72] Alan M Turing. Computing machinery and intelligence. In *Parsing the turing test*, pages 23–65. Springer, 2009.

[73] Jos Uiterwijk. Predictions and Perfection: The Art of Solving Games, 2006.

[74] H Jaap Van Den Herik, Jos WHM Uiterwijk, and Jack Van Rijswijck. Games solved: Now and in the future. *Artificial Intelligence*, 134(1-2):277–311, 2002.

[75] Nees Jan van Eck and Michiel van Wezel. Application of reinforcement learning to the game of Othello. *Computers & Operations Research*, 35(6):1999–2017, 2008.

[76] Mark HM Winands. 6x6 LOA IS SOLVED. *ICGA Journal*, 31(4):234–238, 2008.

[77] Kang Xiaohan, Mohd Nor Akmal Khalid, and Hiroyuki Iida. Player Satisfaction Model and its Implication to Cultural Change. *IEEE Access*, 8:184375–184382, 2020.

[78] Shuo Xiong, Long Zuo, and Hiroyuki Iida. Quantifying engagement of electronic sports game. *Advances in Social and Behavioral Sciences*, 5:37–42, 2014.

[79] Shuo Xiong, Long Zuo, and Hiroyuki Iida. Possible interpretations for game refinement measure. In *International Conference on Entertainment Computing*, pages 322–334. Springer, 2017.

Publications

International Conference (refereed)

[1] A. Primanita and H. Iida. (2019). Nature of Probability-based Proof number Search. Sriwijaya International Conference on Information Technology and Its Applications (SICONIAN 2019), pp. 485-491, 2019.

[2] L. Jiangzhou, A. Primanita, M. Khalid and H. Iida. (2019). Analyzing The Improvement Process of Table Tennis Using The Game. Sriwijaya International Conference on Information Technology and Its Applications, pp. 437-442, 2019.

[3] K. Xiaohan, Z. Zhang, A. Primanita, M. Khalid and H. Iida. (2019). Analysis of Boardgames using Eye-tracking: Case Study with Gomoku," International Conference on Technologies and Applications of Artificial Intelligence, pp. 82-87, 2019.

[4] G. Naying, A. Primanita, M. N. A. Khalid and H. Iida. (2020). A Key Factor to Maintain Engagement: Case Study Using 'Login System'," Sriwijaya International Conference on Information Technology and Its Applications (SICONIAN 2019), pp. 492-497, 2020.

[5] C. Xiao, A. Primanita, M. Khalid and H. Iida. (2019). Analysis of Card Collection Game 'Hearthstone'," International Conference on Technologies and Application of Artificial Intelligence, pp. 76-81, 2019.

[6] Y. An, A. Primanita, M. Khalid and H. Iida. (2020). Ascertaining the Play Outcome using the Single Conspiracy Number in GoBang. 2020 IEEE Conference on Games (CoG), pp. 658-661), 2020.

Journal (refereed)

[7] A. Primanita, M. N. A. Khalid and H. Iida. (2020). Characterizing the Nature of Probability-Based Proof Number Search: A Case Study in the Othello and Connect Four Games, Information Journal, vol. 5, no. 11, p. 264-279, 2020.