Monte Carlo Tree Search for Dots-and-Boxes

Said Krebbers

Supervisors:
Walter Kosters & Hendrik Jan Hoogeboom

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

22/01/2021

Abstract

Dots-and-Boxes is a pen and paper game usually played by two players. The players alternate drawing lines on a rectangular grid of dots, trying to complete boxes. If a player completes a box they must do another move. The goal is to claim more boxes than the opponent by the end of the game. In this thesis we examine the performance of Monte Carlo Tree Search (MCTS) in the game of Dots-and-Boxes and compare it to the methods used in Berlekamp's book. We found that the MCTS algorithm, as is, performs decent against a random agent and poorly in Dots-and-Boxes compared to the other more simple algorithms. The pure MCTS agent could find or gets close to several solutions to problems in Berlekamp's book.

Contents

1 Introduction

Dots-and-Boxes is a pen and paper game, most often played by children that have no access to an iPad or internet connection. The rules of the game are simple, but underneath the simple exterior hides a complex game with various levels of strategy. The leading work on the subject is the book written by Berlekamp [Ber00]. In this book he explains the best strategy known for this game.
For computers the game of Dots-and-Boxes is hard because of its high branching factor. The Monte Carlo Tree Search (MCTS) algorithm has frequently been used in the last years in order to create Artificial Intelligent (AI) agents. The most noticeable of these is the agent written by Deepmind named AlphaGo [SSS+17]. This AI agent and its successors were able to compete with and beat the top players in the game Go, a very complex board game. Go is somewhat similar to Dots-and-Boxes. They are both perfect information two player games with high branching factor.
In this thesis we examine the performance of MCTS in the game of Dots-and-Boxes and compare it to the methods used in Berlekamp's book. We found that the pure MCTS agent performs poorly compared to most other, simpler, agents. However, in two out of three problems from Berlekamp's book which we tested, the pure MCTS agent chooses or gets very close to the optimal move. An enhanced MCTS agent performed better against the other agents than the pure MCTS agent, though not by much. However this agent never picked the optimal move in the tested problems from Berlekamp's book.

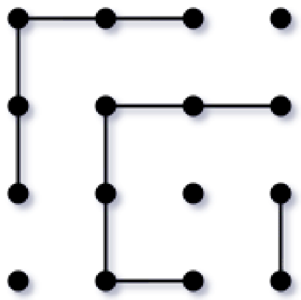

Figure 1: Example of Dots-and-Boxes game in progress.

This thesis has the following layout. Section 2 describes the rules and the most important game-specific terminology. Section 3 discusses related work. Section 4 describes the algorithms used. Section 5 describes the experiments and their outcome. Lastly, in section 6 we conclude this thesis and discuss future research.
This bachelor thesis was written at the Leiden Institute of Advanced Computer Science (LIACS) at Leiden University and was supervised by Walter Kosters and Hendrik Jan Hoogeboom.

2 The Game of Dots-and-Boxes

Dots-and-Boxes is an impartial, zero-sum, perfect information game and is usually played by two players. It is impartial because both players are allowed to do the same moves at any state of the game. The configuration of the playing grid dictates which moves are available [BK12]. If a player completes and claims one box, the other player loses the opportunity to claim this same box. This makes it a zero-sum game. In all possible configurations of the playing field, the players can access all information in the game. At their turn a player knows all moves that have previously been done by the opponent and himself, which makes Dots-and-Boxes a perfect information game [Mye97]. In this section we will explain the rules of Dots-and-Boxes, introduce the most relevant game-specific terminology and describe strategies frequently used my human players.

2.1 Rules

Dots-and-Boxes is a game that is usually played by two players. It is played on a rectangular grid of dots. Players take turns drawing horizontal or vertical lines between two adjacent points. When a player draws the fourth line to make a box, as seen in Figure 2, this player claims this box. Often, players write their initials in the box to mark the claim. If a player is in the position to finish a box, this player may or may not decide to do so. After claiming a box, that player must do another move and draw another line. A player is not allowed to pass their turn.

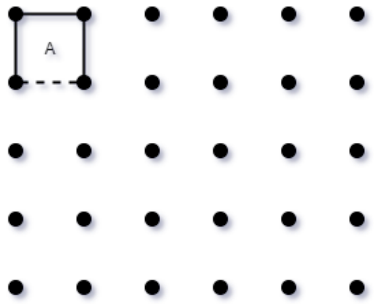

Figure 2: The claiming of a box.

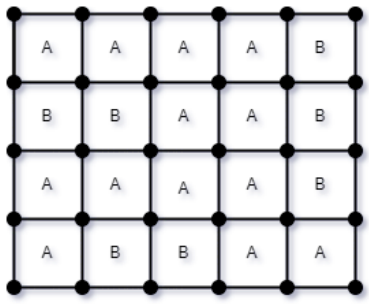

Figure 3: Victory for player A.

It is important to note that as this game is impartial, any player can claim any box given that this player can draw the fourth line. The game ends when all the possible lines have been drawn. At this stage all the possible boxes will have been claimed. The goal of the game is to claim more boxes than the opponent. On a grid with an even number of boxes, a tie is possible. An example victory for player A is shown in Figure 3. Here, it shows that player A has obtained thirteen boxes, whereas player B has obtained only seven boxes. Therefore, player A has won.

2.2 Variations

The Dots-and-Boxes game has multiple variations. Firstly, the game can be played by more than two people which increases the level of complexity. Also, different variations of the starting grid can be found, sometimes depending on the country the game is played in. The standard version starts with an empty grid of dots while the Swedish version starts with a grid that has the outer edges drawn at the start [Ber00], an example of which can be seen in Figure 5. In this thesis we will only look at the two player variant on the standard, empty starting grid, as is shown in Figure 4. The size of the grid is variable and depends on the preferences of the players.

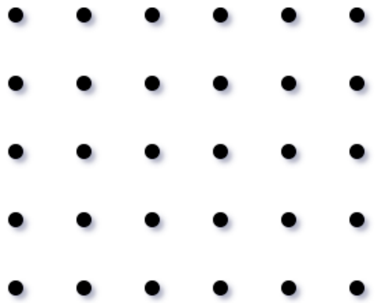
Figure 4: Starting position in the standard version.

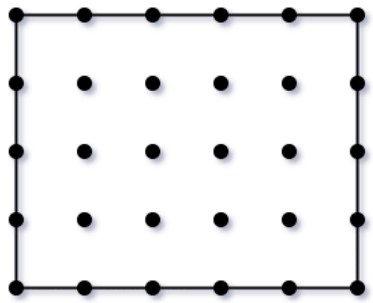
Figure 5: Starting position in the Swedish version.

2.3 Game-specific terminology

This section will introduce the most relevant game-specific terminology that will be used throughout this thesis [BCG04].

Chain: A chain is a set of boxes that could be taken one after another by a player when their opponent makes a move in this chain. A long chain is a chain of at least three boxes. A short chain is a chain of fewer than three boxes. An example of a chain of length five is seen in Figure 6 and Figure 7. In Figure 7 the dotted lines represent the moves that would allow the opponent to take all boxes in the chain.

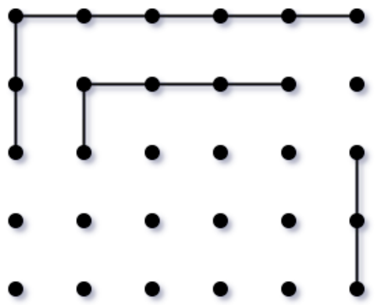
Figure 6: Example of a chain.

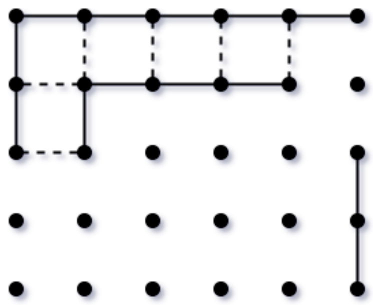
Figure 7: Possible starts of chain.

Loops: A loop is a chain of which the head and tail connect. A loop has to be of at least size four and is always a long chain. An example of a loop can be seen in Figure 8 and Figure 9. In Figure 9 the dotted lines represent the moves that would allow the opponent to take all boxes in the loop.
Handout: A handout is a move that allows the opponent to take boxes. A handout has two variations. If the move gives the opponent the choice whether or not he takes the boxes, it is referred to as a *half-hearted handout* (Figure 10). If the opponent has little choice but to take the offered

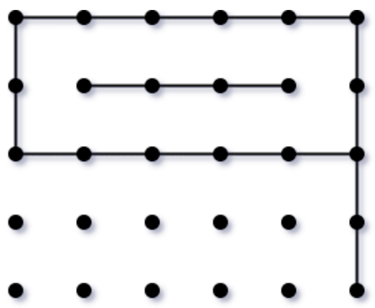
Figure 8: Example of a loop.

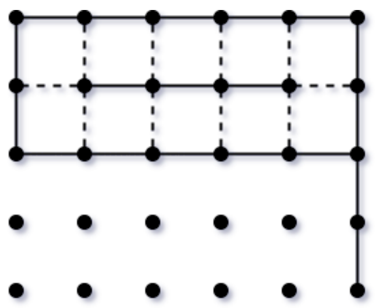
Figure 9: Possible starts of loop.

boxes and make a move elsewhere, it is referred to as a *hard-hearted handout* (Figure 11). The opponent can choose to ignore a handout, but this often leads to unnecessary loss of boxes.

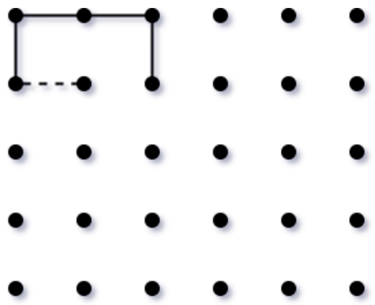
Figure 10: Example of a half-hearted handout.

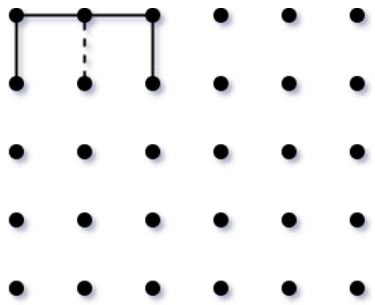
Figure 11: Example of a hard-hearted handout.

Double-dealing move: A Double-dealing move forces the opponent to do a *Doublecrossed* move, through which the opponent claims two boxes in one move. After this, the opponent has to make another move. Often, this move will result in the start of a chain. A double-dealing move could therefore be seen as a small sacrifice by a player to gain access to the larger chain. An example of a double-dealing move is shown in Figure 12. Here, player A forces player B to claim the boxes in the top-left corner. Player A does this by drawing the red line. After drawing the dotted line player B will have to draw a line somewhere in the chain without the ability to claim a box in this chain. This results in player A claiming the larger chain.
Loony Move: This is a move that allows the opponent to choose whether they should do a double-dealing move or hard-hearted handout. When the opponent is able to do this they can choose whether they want to do the first move on the remainder of the board or if they want their opponent to do the first move. If the opponent can look far enough ahead, they are able to see which move will result in the largest number of boxes and choose this move. A loony move gives the opponent the opportunity to get at least half of the remaining boxes. Examples of loony moves are the first move in a large chain or the half-hearted handout.

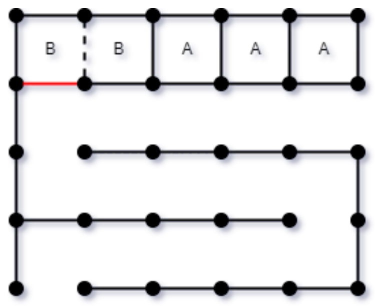

Figure 12: Example of Doublecrossed move.

2.4 Common strategies

There are several distinct strategies human players of Dots-and-Boxes often use. Weaver categorizes five of these possibly overlapping strategies [WB96]:

- Random play
  Moves are chosen at random.

- Box completion
  This strategy pertains finishing a box when it is possible to do so. If it is not possible, the player does a random move. This strategy requires the player to be able to see which moves can finish a box.

- Third side avoidance
  A player will not allow a box of three sides to be available to their opponent if this is possible. This means the player will not leave boxes with three sides unfinished. These players will also avoid to add a third side to a box if this means the opponent would get it next turn.

- Learning to optimally concede boxes
  This requires the player to be able to look ahead and identify what chains are on the grid and how many points each one is worth. Conceding a small chain in order to get a chain of boxes that is worth more points is favorable.

- Learning to ignore a box
  This requires the player to be able to look even further ahead. The player wants to start claiming the first chain after the opponent did a loony move and opened the chain for claiming. The player will want to stay in control and double-cross the opponent, claiming the majority of all boxes from all chains.

This last strategy is at the core of the method explained in the Berlekamp book [Ber00], which is the method the performance of the Monte Carlo Tree Search algorithm will be compared to.

3 Related Work

This section will discuss some of the previous research on MCTS, the leading book on strategies for Dots-and-Boxes written by Berlekamp and the application of the MCTS algorithm to the Dots-and-Boxes game.

3.1 Monte Carlo Tree Search

The meta-analysis by Browne et al. is a survey of literature on Monte Carlo Tree Search (MCTS) [BPW+12]. MCTS combines the precision of Tree Search methods with the randomness of Monte Carlo techniques. The algorithm starts with an empty game tree (with the current situation in the root) and gradually expands this tree using a number of iterations of the following four steps:

- Selection: A node in the tree is selected using some selection policy. This policy is often the Upper Confidence bound applied to Trees (UCT) algorithm. A child node is selected from the current node. If this child node has been visited before and it is not a terminal state, we continue with the expansion step. If the selected child node has not been visited or is a terminal state, we continue with the simulation step.
- Expansion: In this step the selected node has no child nodes. A child node is added for every possible move that can be done from the selected node. One of these child nodes will be selected for the simulation step. As none of these child nodes have been visited yet, the choice is arbitrary. Often the first child node is selected.
- Simulation: A set number of simulations are run from the game state that is associated with the selected node.
- Back propagation: The results of the simulation step are added to the selected node and the parent nodes all the way to the root of the tree.

The four steps are explained in more detail in Section 4. These steps are done for a set number of iterations or until time runs out. When the algorithm is to choose the next move it chooses the most promising child from the root node according to some selection strategy.
An advantage of the MCTS algorithm compared to most other algorithms is that it can be stopped at any time and it will be able to select a move according to its own calculations. When the search is stopped or the allowed time has passed the MCTS algorithm has to determine which move to do. In their survey Browne et al. names four methods for deciding which root child will be selected for the actual move:

- Max child: The root child which promises the highest reward.
- Robust child: The root child which has been visited most.
- Max-Robust child: The root child which has both been the most visited and promises the highest reward.
- Secure child: The root child which maximises a lower confidence bound.

3.2 The book by Berlekamp

The leading work on the Dots-and-Boxes game is the book written on Dots-and-Boxes by Berlekamp [Ber00]. In his book Berlekamp explains the mathematical foundations of Dots-and-Boxes. This book also explains the best strategies known to play and win this game.
The player that gets to do the last move that is not a handout is usually the player that is in control for the remainder of the game, and is most likely to win the game. The strategy in this book explains how to take control of the game, stay in control and win. Once the board is filled with only chains the player that is to move has to open a chain. The second player can then take many boxes and double-cross the opponent. The opponent must now open a new chain for the second player. The book describes how one can increase the chance of not being the one that has to open the first chain. The rule of thumb is to try and make the number of initial dots plus the number of long chains even if you are the starting player and uneven if you are not. The book also explains how the game is similar to the game nimstring and how Sprague-Grundy theory can be used to gain an advantage in Dots-and-Boxes. However, this is outside the scope of this thesis.

3.3 Monte Carlo Tree Search and Dots-and-Boxes

At the moment of writing, relatively few papers have been published comparing the performance of the MCTS algorithm with other algorithms for the Dots-and-Boxes game. Dührsen et al. [DFF+20] compared the MCTS algorithm to multiple algorithms such as Game Tree, rule based and MiniMax algorithms. In their tests they concluded the MCTS algorithm performed the poorest. No explanation has been given as to why it performs poorly.
Many papers have been written on how to improve the MCTS algorithm for Dots-and-Boxes. Prince [Pri17] summarizes this and explains how one can reduce the size of a game state by representing the playing grid with a string of ones and zeros, which denote drawn and not yet drawn lines respectively. This adjustment reduces the amount of memory needed, which means more data can be stored in the same space. The author also shows that a hash table is useful for reducing the number of identical nodes in the game tree. This reduction in identical nodes means less memory is used to store the tree. The paper also expands on some potential improvements such improving the simulation algorithm, which could result in more accurate predictions, or parallelizing simulation in order to do the same amount of work in less time.

Both the selection policy [Pri17] and the simulation policy [LZDP19] can be replaced by a neural network to improve the selection and simulation steps respectively. Improved selection would mean allotted resources, for example calculation time, are used in more effective places. An improved simulation policy ensures that the predictions made from the simulations are more accurate. AlphaGo, a very successful MCTS agent in Go and other games, and its successors show an effective combination of deep neural networks, reinforcement learning and MCTS [SSS+17].

4 Algorithms

In this section we will describe how we implemented the Dots-and-Boxes game. We will explain the used algorithms and how these were implemented in the Dots-and-Boxes game.

4.1 Game engine

The game engine we used was made from scratch for this thesis. It was written in C++. The code is available on Github on the following page: `https://github.com/kurebasu/Dots-and-Boxes`. This language was chosen for its efficiency and our own proficiency in the language. The program takes several parameters as input. The parameters are the following:

- Height: Height of the playing field counted in boxes;
- Width: Width of the playing field counted in boxes;
- numberOfGames: Number of games to be played between the players;
- MC iterations: Number of iterations the MC players are allowed to do;
- Player1: Regulates which agent will control the first player;
- Player2: Regulates which agent will control the second player;
- print: Whether to print output other than the final score;
- printMCTStree: Whether to print the tree built by a MCTS player;
- MCTSEnhancement: Whether to use an enhanced simulation policy;
- MCTSMoveChoice: Whether to prioritize moves that promise higher reward or to prioritize stable moves.

The program outputs the tree produced by a MCTS player in the DOT language. This is a language used to visualize graphs. There are many interpreters available for this language, one of these is `https://edotor.net/`.

4.2 AI agents

In order to test the performance of the MCTS agent we wrote multiple other agents to play against. All agents will be described in this subsection. All active agents will have access to all functions and information in the Dots-and-Boxes game. These include, but are not limited to, viewing the state of the board, checking how many moves are possible and both doing moves with specific coordinates and some xth move out of all possible moves. The possible moves are ordered so every possible move has a number associated with it. The horizontal lines come before the vertical lines, the lines in top rows come before the lines on lower rows on the board and the lines on the left go before the lines on the right. This means that the 0th move is the horizontal line on the top left if this one is free and the last move is the vertical line on the bottom right of the board, if this is free. Figure 13 shows an example of how moves are numbered.

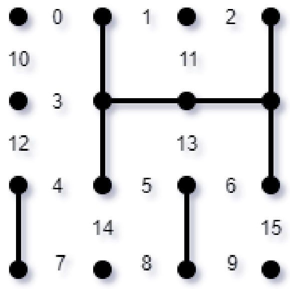

Figure 13: Example of Ordering of Possible Moves.

4.2.1 Random agent

The random agent uses the option to do a numbered move. It checks the amount of possible moves and chooses a random number between and including zero and the amount of moves minus one. All moves are equally likely to be chosen.

4.2.2 Greedy agent

The greedy agent is a simple rule based agent. It checks all potential boxes and checks whether it has three sides drawn and can therefore be completed. If such a box exists it will always complete this box. If multiple such boxes exist, the agent will complete the first box. In the next turn it will find the other boxes to complete them. If no such box exists it will do a random move as described in the random agent subsection. This agent does moves based on the coordinates of a box and a specific side of that box.

4.2.3 Pure Monte Carlo agent

The Pure Monte Carlo, or briefly Monte Carlo (MC), agent gets a number of allowed computations from the input parameters of the program. When this agent is to move, it will test every possible move from the current game state. It does this by trying every move and simulating the end result based on the standard "random policy". Using this policy the game will be played until the end as if played by two random players. These playouts result in a win, a loss or a tie for the Monte Carlo agent. These results are stored for the end evaluation for this move. In the end, the move that resulted in the most wins will be chosen. Other selection policies could be used, for instance favouring moves with low variance or the move with least losses.

4.2.4 Monte Carlo Tree Search agent

The Monte Carlo Tree Search (MCTS) agent combines the Monte Carlo Algorithm with Tree Search. The MCTS algorithm uses a specific data structure to store its computations. This structure is a tree of nodes. Each of these nodes represents a specific game state. One of these game states includes the position of the board, the amount of points scored by each player until that point and who the next player to move is. These nodes have links, or pointers, to their parent node and child nodes. An example of what a tree made of nodes looks like can be seen in Figure 14. The numbers inside the node refer to the results of the simulations. The first node (root node) has value 20/7/3, meaning all simulations in this tree add up to 20 won, 7 lost and 3 tied games for the MCTS agent.

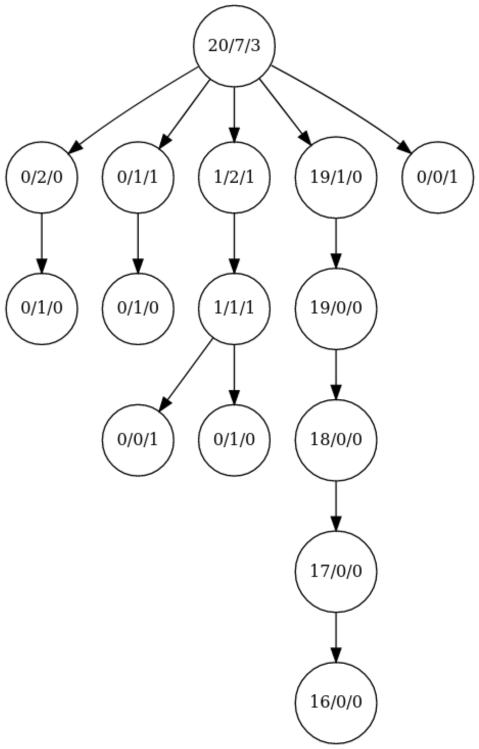

Figure 14: Example of a tree build by MCTS.

In this example it is easy to see that the 4th move (19/1/0) is the move that looks most promising from the current game state.
The MCTS agent builds this tree using four steps. These steps are selection, expansion, simulation and back propagation, which have been explained before in Section 3. The selection step uses the Upper Confidence bound applied to Trees (UCT) algorithm which was introduced as an algorithm that can balance between exploitation and exploration [KS06]:

$$UCT_i = \frac{w_i}{n_i} + C\sqrt{\frac{\ln N_i}{n_i}}$$

This formula consists of two parts:

- w_i/n_i — This is the exploitation factor which is used to select the most promising node. In this part w_i is the amount of wins in node i and n_i is the amount of simulations done in i and its children.
- $C\sqrt{(\ln N_i)/n_i}$ — This is the exploration factor which is used to give unexplored nodes a better chance. In this part Ni is the amount of simulations done by the parent of this node and the parents children. The constant C is traditionally set to $\sqrt{2}$. However we have chosen, based on our own empirical review, to set it to 0.6.

Two different simulation policies have been implemented. The *standard* random policy simulates the rest of the game as if two players were doing random moves. The *enhanced* simulation policy simulates the rest of the game as if two players claim boxes whenever this is possible and safe to do. If no boxes can be claimed the enhanced policy acts like the random policy. Note that this is different than the greedy agent, as the greedy agent does not check if a move is safe to do and just greedily takes it. In Figure 15 one can see the order in which the four steps are done.

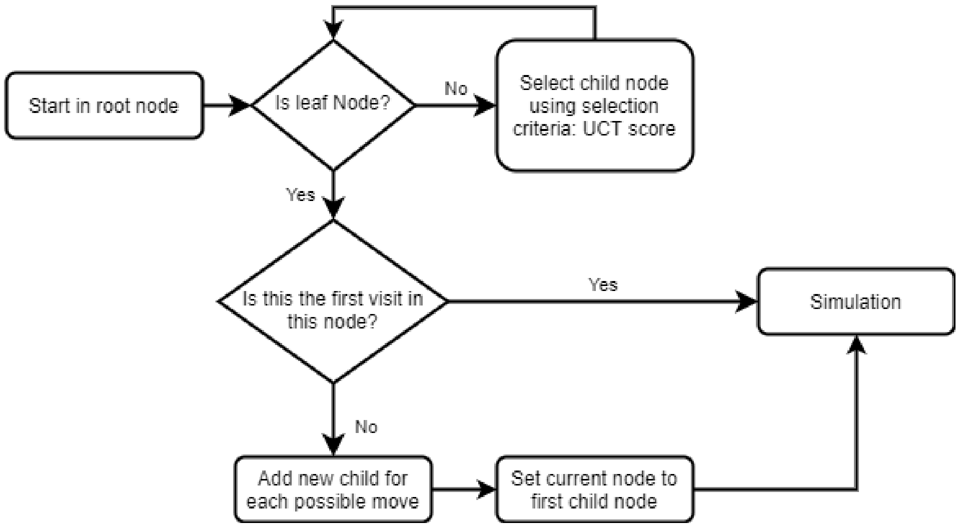
Start in root node
Is leaf Node?
No
Select child node using selection criteria: UCT score
Yes
Is this the first visit in this node?
Yes
Simulation
No
Add new child for each possible move
Set current node to first child node

Figure 15: Flow of the four MCTS steps.

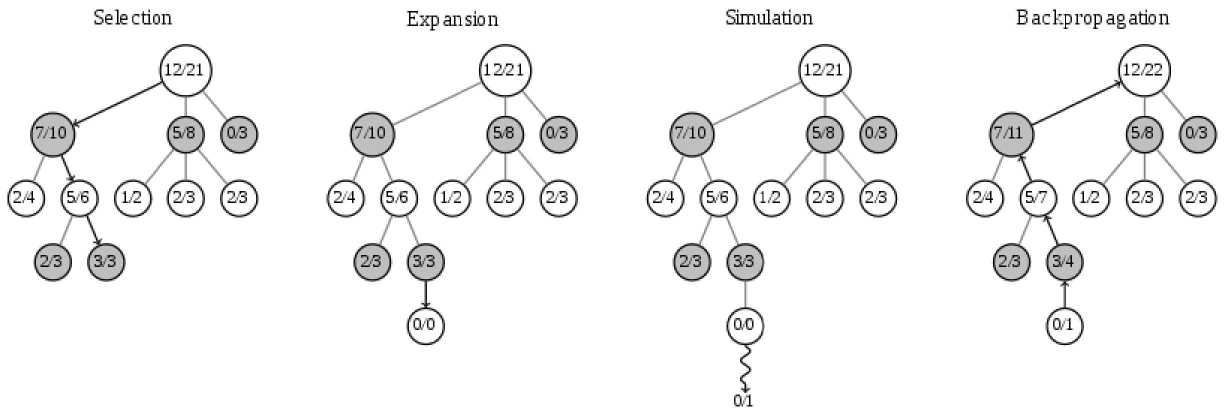

Figure 16: Flow of the four MCTS steps [Mci13].

The MCTS agent starts in the root node and checks whether it is a leaf node. If not, it keeps **selecting** a child according to highest UCT value and changing the current node to the selected child node until the current node is a leaf node. If it is a leaf node it will check whether the node has been visited before. If the node has not been visited before the MCTS agent will **expand** the tree by adding a child node for every possible move to the selected node. After this the first new child node will be selected as the current node. If it was the first time the node was visited or

after expansion a **simulation** is played out from the selected game state. This results in either win, a loss or a tie for the MCTS agent. The result of the simulation is then **back propagated** up the tree. An example of one sequence of the MCTS steps can be seen in Figure 16. In this figure only nodes that have been visited before are shown, as the many unvisited (0/0 nodes) would clutter the figure and make it unclear. In our program this loop will be executed as many times as allowed by the input parameter MC iterations. Once this number of iterations has been reached the algorithm will make a choice by selecting the most promising child node from the root node. We have implemented two functions to determine which child is most promising. These look for the *max child* and the *robust child*. The max child is the child that promises most reward. The robust child is the child that was visited most often.
In Figure 16 the numbers within the nodes refer to the won simulations and total amount of simulations respectively.

5 Experiments

In order to determine the performance of the MCTS algorithm we propose to do multiple tests. The first test consists of a comparison between several other AI agents for the Dots-and-Boxes game. These agents are described in Section 4. The second test is more specific to the Berlekamp book [Ber00]. This book contains several game situations and gives the optimal move for these situations. We will test some of these situations to see whether the MCTS algorithm compares to the method in the book.

5.1 Comparison of agents

In order to test the performance of the MCTS algorithm we test it against other algorithms. The other algorithms are less complex than the MCTS algorithm and provide an insight in how well it plays against other algorithms. We tested on a board of 3×3 boxes. In every matchup we gave the MC and MCTS agents 1000 iterations for each move. We ran each matchup between two algorithms 1000 times. Every matchup was performed twice; if in the first test an agent had to do the first move each time, in the second test this agent had do the the second move each time. Both the standard version of MC and MCTS and enhanced versions, in which the simulations have been adjusted to always do moves to take boxes that have no consequences, have been tested. This testing required some calculation and took 114 minutes of computing time on a pc with an intel core i7 8700k processor. The results of these tests can be seen in Figure 17.
In this figure the first of the two players on the vertical axis is the one that is to do the first move. Both the MC and the MCTS agents use the same simulation policy. So both use the standard or both use the enhanced simulation policy. We expected the random agent to perform the least of the agents. From Figure 17 we can see that the random agent nearly always lost against the greedy agents. It wins a small portion of the games against the MC and MCTS agents, these win percentages are unexpected as it plays randomly against agents that actually do some calculations. It might be explained by the fact that randomness plays a big part in these agents as well. When the random agent is matched up with the enhanced versions of the MC and MCTS algorithms the win rates drop to nearly zero against the MC and it reduces to 3% against the MCTS.
The greedy agent is a simple heuristics based agent. This agent does simple checks on the game board to take boxes wherever possible. This seems to be a decent strategy as it scores fairly well in these tests. The greedy agent won nearly all games against the random player which is to be expected. It also won over 95% of the games against the standard MC and MCTS agents. It still performed well against the enhanced MCTS agent with a win rate of 75%. It performed worse against the enhanced MC algorithm with a win rate of 24%.
The MC agent has two versions, which are the standard and the enhanced versions. The standard version performs decent against the random player with a 91% percent win rate where the enhanced version won 99.3% of the games. The standard MC agent does not perform well against the greedy algorithm with a win rate of 3%. The enhanced version does better against the greedy version, but still not convincingly as the first player with a win rate of 40%. The standard MC agent seems to perform about as well as the MCTS agent. When looking at the enhanced versions of both of these, however, we see that the MC agent performs significantly better than the MCTS agent with a win rate of almost 85%.
We expected the MCTS agent to perform the best of the agents. Just like the MC agent, the MCTS

agent has both a standard and enhanced version. The MCTS agent has multiple options for the move actual move selection. These prioritize the max child or robust child as explained in Section 4. The differences between these options are negligible so we will look at the max child version. Both the standard and enhanced versions perform decently against the random agent with 87% and 90% win rate respectively. This is lower than expected as the MCTS agent checks many states and performs many calculations to do a move, whereas the random agent picks a random move. The MCTS agent performs poorly against the greedy agent with win rates under 1% for the standard and 3% for the enhanced version. With the MCTS agent as first player it seems to be comparable to the MC agent with the standard versions for both. However when looking at the enhanced version we see that the MCTS performs poorly compared to the MC with a win rate of about 5%.
When comparing the standard and enhanced versions of the MC and MCTS agents against other agents we see that the win rate is higher for the enhanced versions. When comparing the MC and MCTS agents against each other, we see that the standard versions are comparable in performance. However, with the enhancement the MC agent seems to perform significantly better than the MCTS agent. This indicates that the enhancement is more effective for the MC agent than for the MCTS agent.

5.2 Specific situations from the Berlekamp book

The Berlekamp book [Ber00] contains several problems and solutions for the Dots-and-Boxes game. These problems are a certain state of the game in which there is one best move. As explained in Section 3, a player should aim for a certain amount of long chains in the endgame. These problems are examples of how counting the chain helps in doing the best move.
We have selected three of these problems in order to compare the MCTS algorithm to the method in the Berlekamp book. These three problems have been chosen because of their small size of 3×3 and relative simplicity compared to the more advanced problems. For each of these problems we executed the MCTS algorithm with as starting state the game state as shown in the book. We allowed the MCTS algorithm to do 1000 iterations in order to determine which move would be best. From the tree built by the MCTS algorithm we could calculate what profit each move will give us. We present this in figures in which the moves have been replaced with this profit. The green number represents the move that the MCTS will choose. We decided to only look at the max child selection and not at the robust child selection because they behave nearly identical. We do differentiate between the standard, random, simulation policy and the enhanced one.

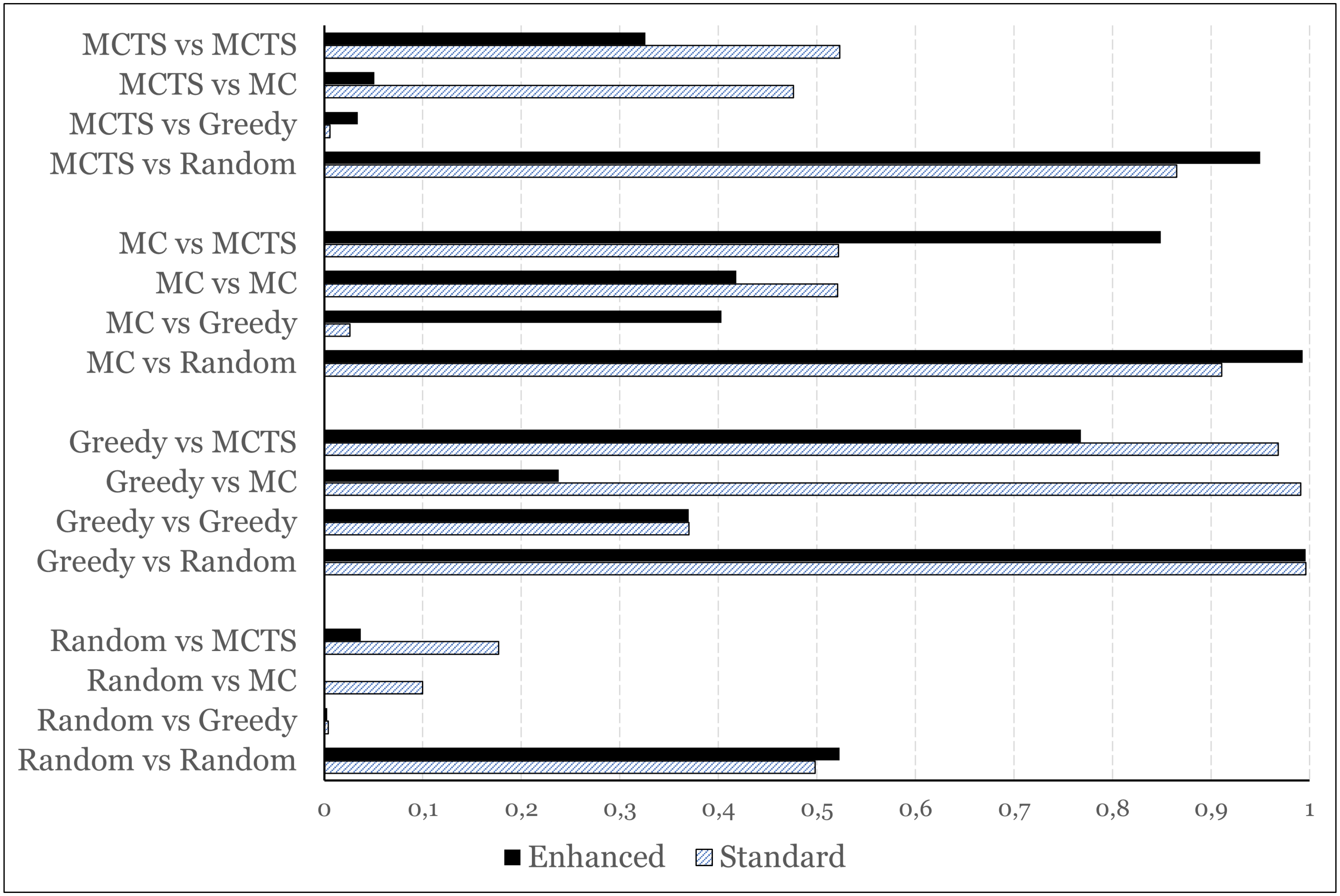

Figure 17: Win rate of the first player against the second player.

5.2.1 Problem 1

In the first problem ten lines have already been drawn and no boxes have been claimed. This means it is the first player's turn. The problem is illustrated in Figure 18 and the unique best move according to Berlekamp is shown as a dotted line.

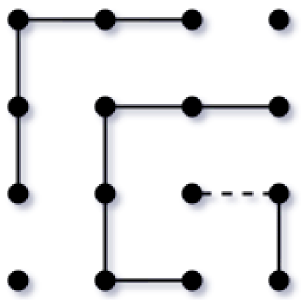

Figure 18: Situation 1. Best move according to Berlekamp.

According to Berlekamp the first player should aim to create an even number of initial dots plus the number of long chains. There is an even number of dots, so the first player should aim for an even number of long chains. There is a long chain in the top left of at least three boxes. The dotted move creates a second long chain in the bottom right and sets the first player up for certain victory. In Figures 19 and 20 the scores calculated from the MCTS tree are shown for each move for the standard and enhanced versions respectively. The green scores show the moves that will be done.

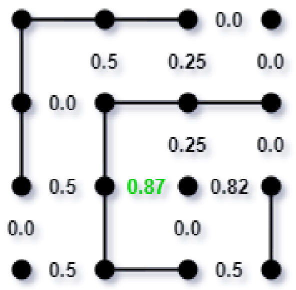

Figure 19: Standard MCTS move chance.

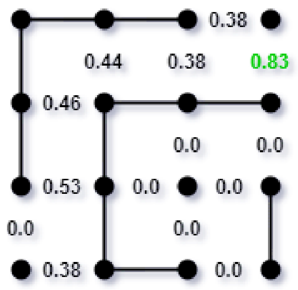

Figure 20: Enhanced MCTS move chance.

In this situation the random simulation policy shows two moves with high scores. The move specified by Berlekamp has a high winrate of 82%. Another move, however, has a higher win rate of 87%. This move removes the chance for two long chains to be created and ruins the chance to win against perfect play. The enhanced simulation policy shows very different chances. Only one move has a high win rate and it is not the move advised by Berlekamp. This move allows the opponent to ensure only one long chain is possible by doing the same move done by the standard version, which eliminates any chance of victory against perfect play.

5.2.2 Problem 2

In Problem 2 twelve lines have already been drawn and no boxes have been claimed. This means it is the first player's turn. The problem is illustrated in Figure 21 and the best move according to Berlekamp is shown as a dotted line.

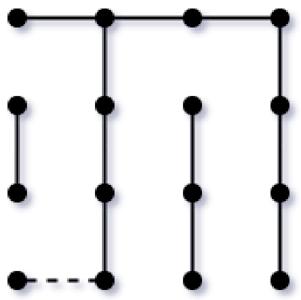

Figure 21: Situation 2. Best move according to Berlekamp.

There is an even number of dots, so the first player should aim for an even number of long chains. There is one long chain of six boxes on the right side and a potential long chain of three boxes on the left. The move suggested by Berlekamp completes this long chain of three on the left. This forces the opponent to do the first move in a long chain allowing the first player to claim some boxes and do a double dealing move, also claiming the other chain at the expense of two boxes.

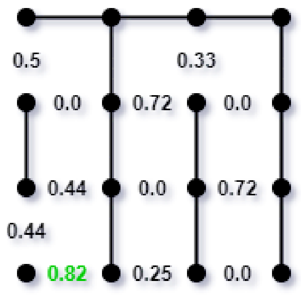

Figure 22: Standard MCTS move chance.

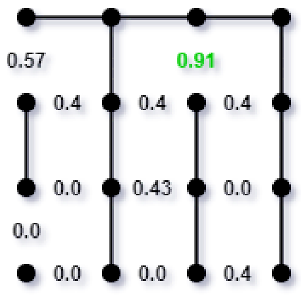

Figure 23: Enhanced MCTS move chance.

In Problem 2, the random simulation policy would choose the same move as Berlekamp would with a win rate of 82%. The enhanced simulation policy chooses a move that immediately gives a long chain of six boxes out of nine to the opponent, losing the game in the process.

5.2.3 Problem 3

In Problem 3 eleven lines have been drawn and no boxes have been claimed. This means it is the second player's turn. The problem is illustrated in Figure 24 and the best move according to Berlekamp is shown as a dotted line.

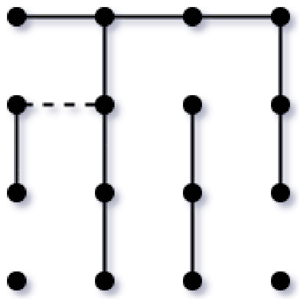

Figure 24: Situation 2. Best move according to Berlekamp.

There is an even number of dots, so the second player should aim for an odd number of long chains. There is one long chain of at least five boxes on the right side and a potential long chain of three boxes on the left. The move suggested by Berlekamp interrupts this long chain of three on the left. This forces the opponent to do the first move in a long chain allowing the second player to claim the long chain.

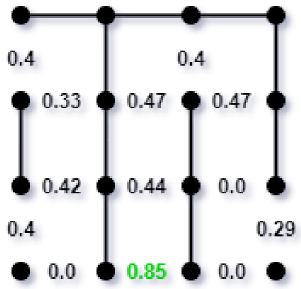

Figure 25: Standard MCTS move chance.

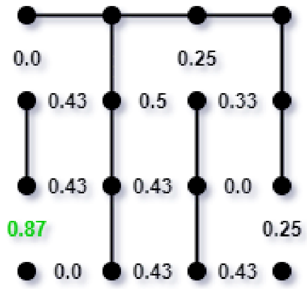

Figure 26: Enhanced MCTS move chance.

In Problem 3 the standard MCTS version does the first move in a long chain of five boxes out of nine boxes, effectively losing the game right there. The enhanced MCTS version completes the long chain of three on the left side, making the number of long chains in this game even. This allows the opponent to do the last free move on the bottom right. The MCTS will have to do the first move in a long chain, giving away all but two boxes.

5.2.4 Summary

In some situations the standard MCTS does or comes close to doing the move Berlekamp suggests. In the shown situations the enhanced MCTS seems to never do the optimal moves. This is strange as the enhanced MCTS shows higher win rates than standard against random and greedy agents.

6 Conclusions and Further Research

In this thesis we look at the Dots-and-Boxes game. We aim to analyze the performance of the Monte Carlo Tree Search (MCTS) algorithm for Dots-and-Boxes and in particular compare it to the leading work on the field by Berlekamp [Ber00]. We expected the MCTS agent to perform well against the other agents and find the correct solutions. Infact, we find that the MCTS algorithm performs better than the random agent, similar to the regular MC algorithm and worse than a greedy agent, a simple rule base algorithm. This shows that the MCTS algorithm without any special improvement performs worse than expected. We also find that the MCTS with enhanced simulation policy performs better than the pure MCTS agent against the other agents except for the enhanced MC agent. The MC agent with the same enhanced simulation policy performs better than the MCTS, which indicates it benefits more from the enhancement. We find it odd that the MCTS agent does not outperform the MC agent, as the MC essentially only looks at the top level of the game tree and the MCTS looks deeper. A reason might be that the MCTS does not investigate broad enough.

We have tested the performance of both the pure MCTS agent and the MCTS agent with enhanced simulation policy on three problems presented in Berlekamp's book. The pure MCTS agent found the solution to Problem 2 and came close to finding the solution in Problem 1. The enhanced MCTS agent did not find a single solution. This finding was unexpected as the enhanced MCTS agent seems to outperform the pure MCTS agent against the other agents.

While tweaking the MCTS algorithm, we notice that the constant in the UCT formula has significant effect on the overall win rate. We see that a constant near 0.6 works best. As the standard constant for the exploration factor in the UCT is $\sqrt{2}$ it seems that the exploration factor is not as important as the exploitation factor for MCTS for Dots-and-Boxes.

For future work it might be interesting to try other simulation policies. This could lead to more accurate predictions and trees, which in turn causes the algorithm to pick better moves. Because of time and hardware limitations we decided to keep the size of the board contained to 3×3. It might be interesting to look at how MCTS performs on a larger board. We doubt however that it will perform better without better enhancements.

References

[BCG04] E. Berlekamp, J. Conway, and R. Guy. *Winning Ways for Your Mathematical Plays*, volume 1–4. AK Peters, 2nd edition, 2001–2004.

[Ber00] E. Berlekamp. *The Dots and Boxes Game, Sophisticated Child's Play.* Routledge, 2000.

[BK12] J. Barker and R. Korf. Solving dots-and-boxes. *Proceedings of the Twenty-sixth AAAI Conference on Artificial Intelligence*, pages 414–419, 2012.

[BPW+12] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Liebana, S. Samothrakis, and S. Colton. A survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1:1–43, 2012.

[DFF+20] M. Dührsen, I. Fabris, F. Feyzi, L. Gauthy, C. Giepmans, and E. Simon. Intelligent agents for solving the game dots and boxes, 2020.

[KS06] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Machine Learning: ECML 2006*, pages 282–293. Springer Berlin Heidelberg, 2006.

[LZDP19] S. Li, Y. Zhang, M. Ding, and Dai P. Research on integrated computer game algorithm for dots and boxes. *The Journal of Engineering*, pages 601–606, 2019.

[Mci13] Mciura. Steps of Monte-Carlo Tree Search. Chess Programming Wiki, 2013. [https://www.chessprogramming.org/File:MCTS_(English).svg; accessed December 29, 2020].

[Mye97] R. Myerson. *Game Theory, Analysis of Conflict.* Harvard University Press, 1997.

[Pri17] J. Prince. Game specific approaches to Monte Carlo Tree Search for dots and boxes. *Honors College Capstone Experience/Thesis Projects*, 2017. Western Kentucky University.

[SSS+17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

[WB96] L. Weaver and T. Bossomaier. Evolution of neural networks to play the game of dots-and-boxes. *Artificial Life V: Poster Presentations*, pages 43–50, 1996.