Implementing Monte Carlo Tree Search in Variations of Connect Four

Nicholas Abad

May 25, 2018

Abstract

With the recent emergence of the Monte Carlo Tree Search algorithm within artificial intelligence, there have been a number of recent works that have successfully implemented this algorithm within a game setting, all of which have helped push this field to new heights. Taking inspiration from these past papers, this specific paper implements the Monte Carlo Tree Search algorithm within the popular board game Connect-4 in an attempt to research how the altering of heuristics effect an agent's chance of winning a match. In particular, the optimal exploration constant that is used within the UCB algorithm, the number of simulations that an agent is allowed to take before selecting a best action, and the different Connect-4 board sizes are all explored through the continuous play of two intelligent agents. Once these experiments concluded, an analysis of the results was then conducted as well as a discussion as to what can be improved upon in future works.

1 Introduction

Within the field of artificial intelligence, an extensive amount of research has been previously conducted on the Monte Carlo Tree Search (MCTS) algorithm, which is a search algorithm that finds optimal actions in a specific domain through the use of multiple simulations throughout an action space, which is depicted by a tree. By treating potential actions as arms within the age-old multi-armed bandit problem, this algorithm has helped push this field to new heights, specifically within game settings. Within the artificial intelligence community itself, it has been well documented and mentioned a countless amount of times that the Monte Carlo Tree Search algorithm was the main proponent used by DeepMind in order to create the first computer program, AlphaGo, that was able to accomplish what was thought to be an impossible task at the time by defeating a professional Go player on a regular 19 x 19 board without any handicaps back in October of 2015 [1]. In addition to this original implementation within the game of Go, Monte Carlo Tree Search algorithms have also been implemented in other board, real-time, and non-deterministic games such as Hex[2], Poker[3], Havannah[4], Pac Man[5], Total War: Rome II[6], Magic The Gathering[7], and Settlers of Catan[8] to name a few. Within these games, both research groups and individuals alike have created their own variations of this original method in order to tune and optimize the MCTS search algorithm and come up with the optimal decision that best suits them and their problem, many of these variations of which can be found in the *Related Works* section of this paper. However, despite how decisions are made in all of these works, one thing remains constant: games play an integral part in the implementation of the Monte Carlo Tree Search algorithm.

Due to wanting to continue this trend of experimenting within a game setting, one specific area that this paper explores is the implementation of this search algorithm on the classic game Connect-4, which, for those who are unfamiliar with the game, is a simple two-player, complete information, zero-sum, connection game in which each player takes turns dropping a board piece into one of the vertical columns in an attempt to vertically, horizontally, or diagonally, create a line of 4 discs of the same color. In an attempt to discover what heuristics work best within this game, this paper will touch upon how the number of simulations, the exploration constant within the Upper Confidence Bound (UCB) algorithm, and the board size effects the overall winning percentage of each of the players (otherwise known as agents) involved. In order to do so, two intelligent agents were subsequently created from scratch and put against each other, one of which whose heuristics were held constant while the other had its heuristics varying. By recording the

final details of all of these experiments, results and conclusions have been made and could thus be found in *Section 4: Results and Analysis*, *Section 5: Discussion and Future Works*, and *Section 6: Conclusion*.

2 Related Works

Before delving deep into the *Methodology*, *Experiments*, and *Results* sections, it is important to note that there have been previous studies conducted that have went into great detail about the Multi-Armed Bandit Problem, the Upper Confidence Bound Method, and Monte Carlo Tree Search within a game setting, all of which are things that will be heavily discussed within this paper. Because of this, it is of great importance to briefly note the main crux of these papers as well as how this relates to the experiment at hand.

Given a Connect-4 board state at a certain time, choosing the optimal action could be likened to that of the original Multi-Armed Bandit Problem, which was originally introduced by Herbert Robbins in 1952 [9]. In the context of this work, Herbert Robbins proposed a problem in which one has a fixed amount of resources that he/she wants to allocate to a number of choices in such a way that maximizes that person's gain. In essence, the agent choosing an action has to choose between whether it wants to explore unplayed actions or to exploit previously played actions, which is typically known as the exploration-exploitation trade-off. Because of the importance of this work and its centrality to the experiment at hand, this paper goes into far greater detail of Robbins' work, particularly in *Section 3.1: The Multi-Armed Bandit Problem*. Because of this, this section will leave out the mathematical details of this work temporarily.

In an attempt to come up with a solution to the exploration-exploitation problem introduced by Robbins, Sutton and Barto [10] proposed the famous ϵ-greedy approach, which is a simple rule-based approach when deciding which action to take. As an agent plays the different actions that are available to it, the agent keeps track of the average payout of each of the actions and thus selects the machine with the highest average payout with a certain probability that is denoted by

$$(1 - \epsilon) + (\frac{\epsilon}{A}) \tag{1}$$

For the other sub-optimal actions, these actions are then selected with probability $\frac{\epsilon}{A}$. In this algorithm, ϵ is chosen to be a small value, such as 0.10, while A denotes the total amount of actions that are available to an agent at that time. As one could probably infer by the algorithm's name, the ϵ-greedy approach is greedy in the sense that an agent typically chooses the optimal action but occasionally chooses sub-optimal actions.

Another attempted solution to the exploration-exploitation problem comes from the Softmax Strategy [11] (also known as Boltzmann Exploration), which deals directly with exploring new actions. Unlike the ϵ-greedy approach which chooses each sub-optimal action with uniform probability, the Softmax Strategy selects an action a with

$$P(a) = exp\{Q(a)/\tau\}/\sum exp\{Q(a)/\tau\} \tag{2}$$

in which $Q(a)$ denotes the Q-value and τ represents a constant greater than 0 that specifies how likely actions are to be chosen. If τ is relatively high, actions will be chosen with roughly the same probability and consequently, as $\tau \to \infty$, the best action will be chosen.

Thirdly, another proposed solution to the Multi-Armed Bandit Problem is the Upper Confidence Bound (UCB) algorithm, which was proposed by Peter Auer [12]. This algorithm chooses the action that corresponds to having the greatest UCB value, which can be calculated by the following:

$$UCB1_a = \bar{X_a} + C\sqrt{\frac{2log(N)}{n_a}} \tag{3}$$

where $\bar{X_a}$ denotes the average reward of this action, C is a given constant, N denotes the total amount of overall plays, and n_a denotes the amount of times action a has been played. Similar to the first related work by Robbins, this paper goes into extensive mathematical detail and explanation regarding this topic due to its centrality of the experimentation in the *Methodology* section, in particular *Section 3.2: Upper Confidence Bound*, so for now, this paper will not yet go into heavy detail about this work and will save it for later.

In addition to creating this algorithm, Auer, Cesa-Bianchi, and Fisher [12] also developed another algorithm, conveniently called UCB1-Tuned, in which they claimed that this algorithm performed better than the original UCB1 algorithm in practice and could be calculated by the following:

$$j(t) = \arg\max_{i=1...k} \{\hat{\mu_i} + \sqrt{\frac{ln(t)}{n_i} \min\{\frac{1}{4}, V_i(n_i)\}}\} \tag{4}$$

where $V_i(t) = \hat{\sigma}_i^2(t) + \sqrt{\frac{2ln(t)}{n_i(t)}}$. In essence, this algorithm takes into account both the mean as well as the variance of each action.

In addition to these specific papers, Browne [13] also quickly glosses over many of the other UCB and UCT variations that have been proposed throughout the years in his survey paper. Among these variations that he lists that have been created by other authors, he lists other potential UCB variations, tree policy enhancements, as well as other enhancements that may be critical in other experiments that also use the Monte Carlo Tree Search algorithm.

Within the game of Connect-4, there have also been several other papers that have successfully implemented the MCTS algorithm, two of which have been authored by Baier and Winands [14] [15]. Within their first joint work, rather than just plainly using the MCTS algorithm, Baier and Winands created several variations of the algorithm, all of which incorporated the minimax algorithm at different times of the their Monte Carlo Tree Search. The first variation came when implementing minimax rollouts during the the rollout phase, the second variation came when implementing minimax in the selection and expansion phases, and the third variation came when incorporating the minimax algorithm within the back propogation phase. When using these variations within the game of Connect-4 as well as within the game of Breakthrough, it was experimentally proven that all three of these variations worked better than the original MCTS algorithm by a significant margin. Secondly, Baier and Winands [15] set out to discover what the optimal parameters were in each of the aforementioned games, specifically when using the Monte Carlo Tree Search algorithm. By giving each agent 20 seconds to choose an action, they found that by using the idea of stopping the traversal of a tree once it hits a depth of 5 with probability 0.9, they were able to successfully win in 65.0% of the matches that the program was playing against, which in turn was a general version of Connect-4.

These works, in particular, have been excellent resources before, during, and after the experimentation phases simply due to their extensive background research, detailed methodology, results, and analysis sections, and their overall important to the the algorithms at hand.

3 Methodology

3.1 The Multi-Armed Bandit Problem

Originally introduced by the American statistician Herbert Robbins in 1952 [9], the Multi-Armed Bandit Problem is one in which an agent is required to maximize its total reward or subsequently minimize its regret by choosing a single action a from one of the $a = 1, 2, 3, ..., N$ possible choices within the action space, each action of which corresponds to a reward, with the most common example being likened to an individual trying to choose between a number of slot machines at a casino. After an agent chooses an initial action a, the agent receives a reward that comes from an unknown probability distribution function, but for the remaining $N - 1$ actions, still nothing is known about the rewards behind these actions, which is where the crux of the problem lies. On the second iteration, should the agent decide to exploit the board and choose the action that it already played in Iteration 1 once more or should the agent decide to explore one of the $N - 1$ remaining actions in an attempt to find an action with a higher reward?

Also known as the exploration-exploitation trade-off, the agent choosing an action needs to find a technique in order to balance whether it chooses to exploit what is known or explore what is unknown at each iteration. Too little exploration may result in consistently choosing a sub-optimal action, while on the other hand, too much exploration can result in the agent not deciding to play the optimal action with the high reward, both of which are problematic. With this in mind, one could clearly see how this can relate back to a game setting, in particular within the game of Connect-4. By considering an iteration to be a single turn, an agent ideally needs to determine an optimal action to play that maximizes its reward, which can be depicted as either a win, a loss or a draw.

3.2 Upper Confidence Bound

In order to come up with an algorithm that tries to optimize the aforementioned exploration-exploitation trade-off and as noted previously in *Section 2*, Peter Auer [12] created the Upper Confidence Bound (UCB) algorithm that utilizes confidence bounds to elegantly choose an action. Because an agent receives a reward that is based on that reward's probability distribution function, the consequent reward of the action will not always be consistent. In general, choosing an action will not always guarantee the same reward. However, despite this being the case, by continuously choosing the same action and receiving its reward, the agent develops a certain confidence interval in which the reward lies within. As this action is chosen more and more, the confidence interval thus shrinks since the agent is more certain of the reward that it will receive and thus the average reward tends to the true mean of the reward's probability distribution function. On the other hand, however, if an action is only chosen once, the confidence interval is quite large since this reward might have just been an aberration and may not accurately depict the reward's true mean value.

Mathematically, if there are N possible actions that an agent can take at a certain state, each action a is given a UCB value, which is computed by the following formula that has previously been defined in *Equation 3*:

$$UCB1_a = \bar{X}_a + C\sqrt{\frac{2log(N)}{n_a}} \tag{5}$$

where $\bar{X}_a$ denotes the average reward of this action, C is a given constant, N denotes the total amount of overall plays, and n_a denotes the amount of times action a has been played. It should be noted that within this equation, the $\bar{X}_a$ term is typically called the exploitation term while the $C\sqrt{\frac{2log(N)}{n_a}}$ is known as the exploration term. Naturally, it's evident to see that as you increase the constant C, this would imply that the you are increasing the value of the exploration term. Additionally, it should be noted that if an action has not been played, this action's $UCB1$ value is evaluated to be at ∞.

Once each and every action's $UCB1$ value has been calculated, the action that is chosen is thus the action that has the maximum value when compared to its counterparts:

$$ChosenAction = \arg\max\{UCB1_1, UCB1_2, UCB1_3, ..., UCB1_N\} \tag{6}$$

For the next iteration, the chosen action is then played and this process repeats itself during each and every iteration.

3.3 Upper Confidence Bound for Trees and Monte Carlo Tree Search

By expanding upon the aforementioned UCB algorithm, Kocsis and Szepesvari [11] created the Upper Confidence Bound for Trees (UCT) algorithm, which essentially implements the UCB algorithm within decision trees. By envisioning the given board state as a node and having all of its subsequent actions being child nodes of this original node, the UCB algorithm could thus be used to choose the optimal action by treating each child node as an arm or action of the aforementioned Multi Armed Bandit Problem. As one traverses the tree downwards, each node represents a new action based on the board of it's parent node.

In order to intuitively explain the Monte Carlo Tree Search algorithm, it would be best to split the algorithm up into four separate portions: Selection, Expansion, Simulation, and Back Propogation. The first step in this algorithm is the Selection phase in which a child policy is recursively called, starting from the root node, in order to descend through the tree. Once a node is chosen, that node is then added to the tree if it has not already been added previously. If the node has been previously added to the tree, however, the Selection phase is implemented again, searching that specific node's children once again using the child policy. On the other hand, if the node has not been previously added to the tree, a random simulation, starting from this node, occurs up until a terminal node has been reached (i.e. up until the game has finished). Once a terminal node has been reached, the final reward (1 for a win, 0 for a draw, -1 for a loss) is then back propogated up the tree until it returns to the root node. This process is done iteratively until a certain number of simulations have occurred within the tree or until a certain time has been reached. A full single iteration can be seen in the figure below [13]:

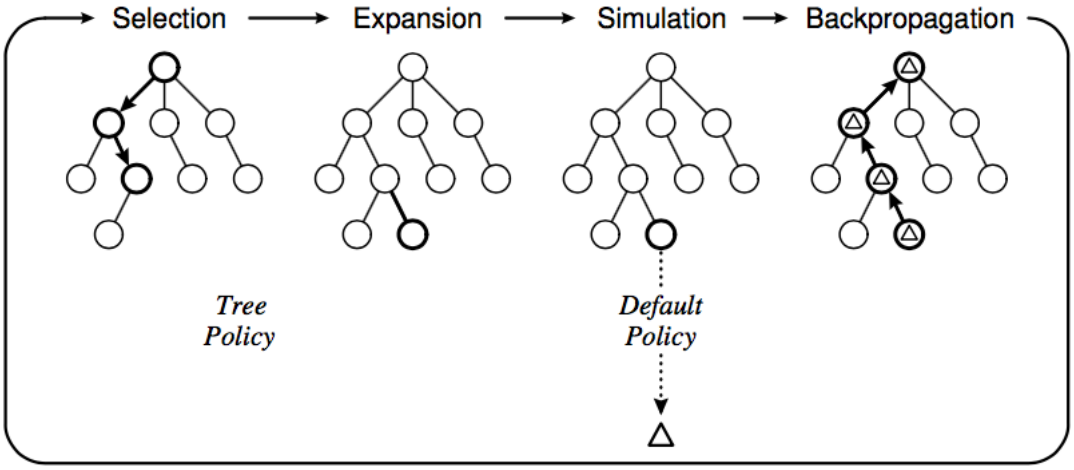

Figure 1: The MCTS algorithm

Because Connect-4, like many other games, is a turn-based, zero-sum, adversarial game, after a node is selected by say Agent 1, the next move would then belong to Agent 2, while the move after that would belong to Agent 1, etc.. With this in mind and in these types of games, an individual agent wants to ideally maximize its own reward while minimizing the reward that its opponent receives. In the context of a decision tree and because Connect-4 is turn-based, at first the agent moving wants to choose an action with the highest confidence bound when it is its turn to play and consequently, wants to choose an action with the lowest confidence bound when it is its opponents turn to play. This, in particular, plays a crucial role in the back propogation phase specifically because of its necessity to minimize the other player's rewards.

4 Results and Analysis

4.1 Exploration Constant

When conducting these experiments, the first objective that this paper wanted to investigate was the overall effect of the exploration constant in the win percentage within the standard Connect-4 board setting, which is a simple 7 x 7 board. In order to do so, Player 2, which was the agent that stayed constant throughout each iteration, was allowed 500 simulations per turn in order to choose a best action, and had an exploration constant of $\frac{1}{\sqrt{2}} \approx 0.7106781$, which was previously used as the exploration constant in the paper written by Kocsis and Szepesvári [11].

On the other hand, Player 1 was also allowed 500 simulations per turn in order to choose the best action but its exploration constant constantly increased after every 100 games by 0.1, beginning at 0.1 and concluding at 1.0. After every 100 games, the statistics were then recorded in terms of the amount of wins for Player 1 and the amount of wins for Player 2 for this specific value of an exploration constant for Player 1. When conducting the 1,000 games (100 games per 10 different exploration constants) it should also be noted that there were no draws. A graph of the results can be seen in *Figure 3*, in which the y-axis measures the total amount of wins that Player 1 received during these games while the x-axis had the value of Player 1's exploration constant. Within this graph, the vertical, dotted orange line represents the value of the exploration constant of the opposing player, Player 2, which is located directly at $\frac{1}{\sqrt{2}} \approx 0.7106781$.

Table 1: Exploring the Exploration Constants for a Standard 7 x 7 Connect-4 Board

Player 1 Wins	**Player 2 Wins**	**Player 1 Exploration Constant**	**Player 2 Exploration Constant**
23	77	0.1	0.710678119
23	77	0.2	0.710678119
27	73	0.3	0.710678119
30	70	0.4	0.710678119
38	62	0.5	0.710678119
50	50	0.6	0.710678119
52	48	0.7	0.710678119
55	45	0.8	0.710678119
98	2	0.9	0.710678119
99	1	1.0	0.710678119

As one may be able to conclude through this graph and as the exploration constant of Player 1 approaches the exploration constant of Player 2, the amount of wins for each agent gets more and more similar to one another at around 50 wins for each of the agents, which is expected, as all other constants have remained exactly the same. In essence, when Player 1 has an exploration constant value of 0.7, Player 1 and Player 2 are essentially identical agents due to their heuristics taking the same values.

However, it is interesting to note that when Player 1's exploration constant was lower than its counterpart, the amount of wins heavily favored that of Player 2. Because this was consistent throughout each of Player 1's exploration constant values (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, and 0.7), it is a reasonable conclusion to make that when keeping all other constants equal, the agent with the higher exploration value wins more often than not. Without loss of generality, it should also be noted that the converse of this sentiment could be concluded as well, particularly by looking at the amount of wins when Player 1's exploration constant value is greater than that of Player 2.

Before the experiment concluded, I personally thought that increasing Player 1's exploration constant would actually decrease the winning percentage of that agent , which was the complete opposite of what was found during the experiment itself. My reasoning behind my initial thought process was that because the game of Connect-4 does not have necessarily a large action space (when put into comparison to other board games such as Chess or Go), the algorithm would be best suited to exploit the moves that it has already seen. Additionally, because a winning combination is a string of four game pieces all attached to one another, I was under the assumption that the available moves to an agent that are closest to its own colored pieces would have a higher value, resulting in high UCB values for either the same column, the column directly to a same colored piece's right, and the column directly to a same colored piece's left.

Despite my initial reservations, however, I was quite pleased with the results simply because there does not to seem to be a lot of noise in the data, in the sense that the amount of wins Player 1 receives correlates heavily with the exploration factor used within the Monte Carlo Tree Search Algorithm.

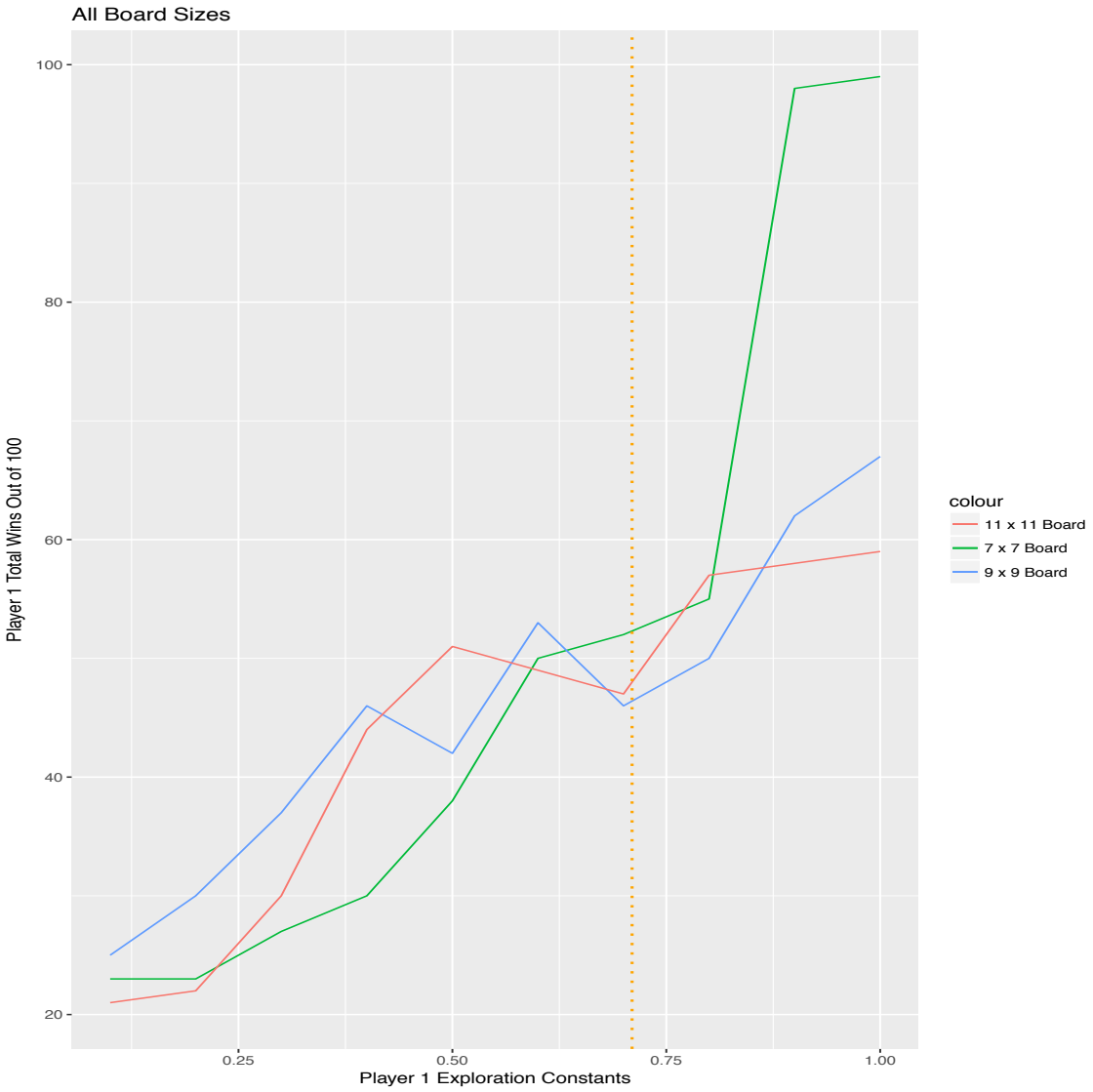

Figure 2: This graph shows how changing Player 1's Exploration Constant in each of the three different Connect-4 board variations effects the total amount of wins that the agent receives. The orange vertical line represents the exploration constant of Player 2, which is roughly 0.71.

4.2 Board Size

After exploring how Player 1's exploration constant effects its ability to win a game in a regular sized 7 x 7 board, it was natural to wonder whether or not the same trend would be prevalent for different sized boards as well. In particular, the two new board sizes that were investigated were a 9 x 9 board as well as an 11 x 11 board. Using the same exact heuristics that were previously mentioned in *Section 4.1: Exploration Constant*, the two experiments were then conducted and the results for these experiments could be found in *Table 2* and *Table 3* respectively.

After examining and graphing the results, which can similarly be found in *Figure 3*, the idea that the amount of wins that Player 1 receives increases as the exploration constants increase stayed consistent with all three boards, which I found to be reassuring. Although there was no large jump in the amount of wins going from one exploration constant to the next like in the 7 x 7 board, the trend line increased somewhat consistently, although there did seem to be some noise within the 9 x 9 board.

Despite the noise, I was once again quite pleased with these results because these tests alleviated my concern that something went wrong within the 7 x 7 board originally. Because of this, the conclusion that was previously made, which was that increasing the exploration constant directly

increases the amount of wins that an agent receives, could be once again experimentally proven in my implementation of Connect-4 as well as the Monte Carlo Tree Search algorithm.

Table 2: Exploring the Exploration Constants for a 9 x 9 Connect-4 Board

Player 1 Wins	Player 2 Wins	Player 1 Exploration Constant	Player 2 Exploration Constant
25	75	0.1	0.710678119
30	70	0.2	0.710678119
37	63	0.3	0.710678119
46	54	0.4	0.710678119
42	58	0.5	0.710678119
53	47	0.6	0.710678119
46	54	0.7	0.710678119
50	50	0.8	0.710678119
62	38	0.9	0.710678119
67	33	1.0	0.710678119

Table 3: Exploring the Exploration Constants for an 11 x 11 Connect-4 Board

Player 1 Wins	Player 2 Wins	Player 1 Exploration Constant	Player 2 Exploration Constant
21	79	0.1	0.710678119
22	78	0.2	0.710678119
30	70	0.3	0.710678119
44	56	0.4	0.710678119
51	49	0.5	0.710678119
49	51	0.6	0.710678119
47	53	0.7	0.710678119
57	43	0.8	0.710678119
58	42	0.9	0.710678119
59	41	1.0	0.710678119

4.3 Number Of Simulations Allowed

In previous works, there has been an extensive amount of research done, primarily by the aforementioned Baier and Winands within the Connect 4 game setting, in which the authors decided to give each agent a set amount of time in order to decide which move is best. However, due to implementing this from scratch, the amount of simulations that my created program could make in a reasonably short amount of time was not nearly as numerous as previous papers. For this reason alone, it was thus decided to give each agent a set number of simulations to run during each move, despite how long each move would actually take, rather than giving each agent a set amount of time.

Keeping this in mind, the next goal of the experiment was to then research how the number of simulations effected that agent's win rate when put up against another agent with constant heuristics in 100 games. Similar to the process conducted in *Section 4.1: Exploration Constant*, Player 1 was the agent in which the the amount of simulations varied while the heuristic values of Player 2 were kept constant. To be explicit, Player 2 was given 500 simulations to learn from in order to choose the best action and was then given an exploration constant of 1.0, which was found to be the best exploration constant that happened to be tested previous sections. On the other hand, Player 1 was also given a exploration constant rate of 1.0 but in terms of the number of simulations this agent was given to choose a best move, it iterated through values of 10, 50, 100, 250, 500, 750, 1000, 1250, and 1500 simulations. The results of the experimentation as well as a graph of this can be seen in *Figure 4*.

As expected and as seen on the corresponding graph, the amount of simulations does indeed play an integral role in determining the amount of wins that an agent receives. Specifically, as the number of simulations that an agent is allowed to take increases, the probability of winning that

game seemingly increases as well. Surprisingly, however, once the amount of simulations reaches 500, the benefit of an increased amount of simulations does not quite grow as much.

One possible reason for this may be because at 500 simulations, an agent more or less learns what the best action is and cannot improve much upon what it already knows at this point. Similar to the reasoning behind what occurred in *Section 4.1: Exploration Constant* and because this is a "small" action space, the algorithm is able to learn the best moves relatively quickly and in a limited amount of iterations.

Table 4: Exploring the Effects of Increasing the Number of Simulations

Player 1 Wins	**Player 2 Wins**	**Player 1 Number of Simulations**	**Player 2 Number of Simulations**
6	95	10	500
3	97	50	500
27	73	100	500
43	57	250	500
54	46	500	500
55	45	750	500
58	42	1000	500
61	39	1250	500
59	41	1500	500

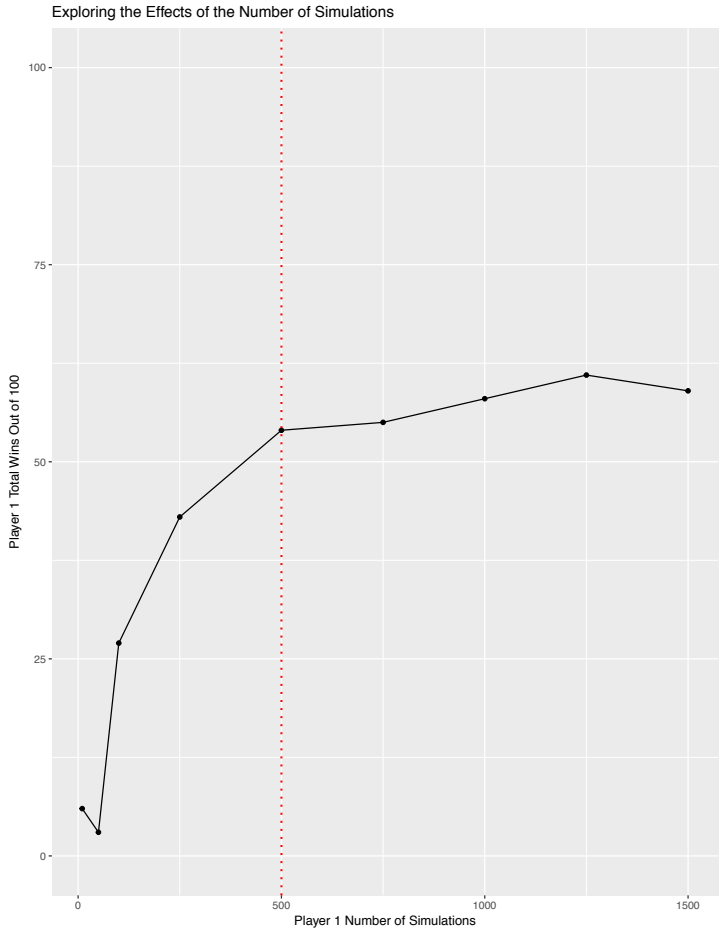

Figure 3: This graph shows the effect of changing the amount of simulations that Player 1 is allowed to take before choosing its best action. The vertical red line indicates the amount of simulations that Player 2 was allowed to make throughout every experiment.

5 Discussion and Future Works

After conducting the multiple experiments, I found some of the results to be surprising but consistent throughout the entire experiment, specifically when increasing the exploration constant. If given the opportunity and more time, future work would include the testing of both smaller and larger boards to see if the aforementioned conclusion still does hold.

Additionally, I thought that it was quite puzzling as to why the amount of games won by Player 1 did not steadily increase after 500 simulations. As noted previously, the only explanation that I could possibly come up with is that the agent "learned" as best as it could at that moment so adding more and more simulations would not quite increase the win rate at the same rate as going from 10 simulations to 500 simulations. If possible, I would have tried even more simulations to see in particular if this result would hold further.

With these two individual future possibilities at hand, it would also be interesting to combine them and experiment with changing both of these heuristic values (exploration constant and board size). By doing so, I hypothesize that changing the exploration constant would be more influential than its counter part but without running extensive experiments to test this, one would not be able to know if this were to be the case.

6 Conclusion

In conclusion, it was confirmed through extensive experimentation that altering the heuristics of the Monte Carlo Tree Search algorithm as well as altering the sizes of the Connect-4 Board itself both play a critical part in determining the outcome of a game. As shown in *Section 4: Results and Analysis* and specifically within *Table 1* and *Figure 4*, with a exploration constant of 0.1, Player 1 only won 23 out of the 100 games that were played on a standard board in comparison to winning 99 out of the 100 games when having an exploration constant of 1.0 with all other variables held constant.

Although the other board sizes, in particular a board size of 9 x 9 as well as a board size of 11 x 11, showed a similar upward trend in wins when altering the exploration constants of Player 1, the change in the number of wins was not quite as drastic as the standard 7 x 7 board but still gave evidence to come to the conclusion that when increasing the exploration constant within a Connect-4 game setting, the amount of wins should consequently increase as well, despite the size of the board.

In terms of the amount of simulations that were allotted to an agent in this specific program, there was indeed a significant increase in wins by Player 1 when only being able to run 10 simulations in comparison to being able to run 500 simulations but this steady increase did not quite hold any further for more simulations. Surprisingly, it was found experimentally that between 500 simulations and 1500 simulations, there only seemed to be a 5 win increase from 54 wins to 59 wins, respectively. Although this may be a slight increase, one could therefore argue that the additional time spent simulating through the tree may not be worth the time, specifically if time is of great importance and/or the computational power is not ideal.

All in all, I found this entire experiment to be an overwhelming success as I was able to successfully create a general Connect-4 board as well as successfully implement the Monte Carlo Tree Search algorithm in addition to experimentally arriving at great results.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[2] B. Arneson, R. Hayward, and P. Henderson, "Mohex wins hex tournament," *ICGA journal*, vol. 32, no. 2, p. 114, 2009.

[3] M. Buro, J. R. Long, T. Furtak, and N. R. Sturtevant, "Improving state evaluation, inference, and search in trick-based card games.," in *IJCAI*, pp. 1407–1413, 2009.

[4] T. Ewals, *Playing and solving Havannah*. PhD thesis, University of Alberta, 2012.

[5] X. Gan, Y. Bao, and Z. Han, "Real-time search method in nondeterministic game—ms. pac-man," *ICGA Journal*, vol. 34, no. 4, pp. 209–222, 2011.

[6] A. J. Champandard, "Monte-carlo tree search in total war: Rome ii's campaign ai," *AIGameDev. com: http://aigamedev. com/open/coverage/mcts-rome-ii*, 2014.

[7] C. D. Ward and P. I. Cowling, "Monte carlo search applied to card selection in magic: The gathering," in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on*, pp. 9–16, IEEE, 2009.

[8] I. Szita, G. Chaslot, and P. Spronck, "Monte-carlo tree search in settlers of catan," in *Advances in Computer Games*, pp. 21–32, Springer, 2009.

[9] H. Robbins, "Some aspects of the sequential design of experiments," in *Herbert Robbins Selected Papers*, pp. 169–177, Springer, 1985.

[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1. MIT press Cambridge, 1998.

[11] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*, pp. 282–293, Springer, 2006.

[12] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[13] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in games*, vol. 4, no. 1, pp. 1–43, 2012.

[14] H. Baier and M. H. Winands, "Monte-carlo tree search and minimax hybrids," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pp. 1–8, IEEE, 2013.

[15] H. Baier and M. H. Winands, "Monte-carlo tree search and minimax hybrids with heuristic evaluation functions," in *Workshop on Computer Games*, pp. 45–63, Springer, 2014.