Humanization of Computational Learning in Strategy Games

By Benjamin S. Greenberg

S.B., C.S. M.I.T., 2015

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 2016

Copyright 2016 Benjamin S. Greenberg. All rights reserved.

The author hereby grants to M.I.T. permission to reproduce and
to distribute publicly paper and electronic copies of
this thesis document in whole and in part in
any medium now known or hereafter created.

Author_____
Department of Electrical Engineering and Computer Science
May 19, 2016
Certified by_____
Andrew Grant, Lecturer/Technical Director, MIT Game Lab, Thesis Supervisor
May 19, 2016
Accepted by _____
Dr. Christopher J. Terman, Chairman, Master of Engineering Thesis Committee

Humanization of Computational Learning in Strategy Games

by Benjamin S. Greenberg

Submitted to the Department of Electrical Engineering and Computer Science

May 19, 2016

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

I review and describe 4 popular techniques that computers use to play strategy games: minimax, alpha-beta pruning, Monte Carlo tree search, and neural networks. I then explain why I do not believe that people use any of these techniques to play strategy games. I support this claim by creating a new strategy game, which I call Tarble, that people are able to play at a far higher level than any of the algorithms that I have described. I study how humans with various strategy game backgrounds think about and play Tarble. I then implement 3 players that each emulate how a different level of human players think about and play Tarble.

1 A Microcosm of Thought

Understanding human thought is one of the largest research ventures in human history. Since the days of Aristotle, people have been wondering how we think. What makes humans so special? And in more recent years, how would one go about replicating human intelligence? These questions probe into some of the deepest topics ever considered, and many people, including myself, consider them to be fundamental to achieving the true extent of human capabilities. Ever since Alan Turing introduced the basic ideas and possibility of artificial intelligence in his paper “Computing Machinery and Intelligence,” the computational world has been enthralled with the idea of a computer that would actually be intelligent (Turing 1950).

There are many different definitions of what intelligence might be. Turing proposed the idea for a test in which a machine holds a conversation with a human with the goal of deceiving the human into thinking that they are talking to another human. DARPA asked for a machine that could correctly identify various actions such as running, eating, and jumping. IBM implemented Watson, a very successful Jeopardy playing program. All of these projects can be considered goal oriented thinking. Goal oriented thinking is the way that most engineering projects are accomplished. The basic idea is that a team comes up with a goal, such as playing Jeopardy as well as possible. The team then figures out the most efficient way to accomplish the goal. Finally, they implement whichever method they determined to be ideal for accomplishing the task. Goal oriented thinking has dominated the field of artificial intelligence thus far. However,

I have decided to attempt a slightly different approach. I have implemented a strategy game player, a well known and studied area of artificial intelligence, but I have implemented the game player in a different way from how most academic research projects implement strategy game players. I have studied the ways that humans think about and play strategy games, and I have replicated the way that they play the game, rather than just focusing on the goal. This problem is actually similar to one that industry game developers solve when they try to make digital players that make people feels as if they are playing against other humans.

There were multiple areas that I could have chosen as the area in which I would attempt replicate human intelligence. I decided to choose strategy game players because I believe that games often act as microcosms for the way that people think about the world. I see this project as a step towards a program that actually thinks in the same ways that a human might think, and I believe that games are a good stepping stone towards this goal because games often act as a simplified version of the real world. I specifically chose turn based strategy games as the area of concentration because I wanted to simplify the experiment to focusing on the way that people think. I did not want to get distracted with more physical embodiments of game play such as precision or reaction time. While these may be important skills, they are difficult to replicate and would have greatly complicated the task at hand. The turn based nature of the games that I focused on allowed me to concentrate solely on the strategy of the game. I wanted to replicate the planning and decision making that goes into how people

play games because that is the part that I believe provides the greatest insight into the minds of the players that I am studying.

2 How Do Computers Play Games

I am far from the first person to be interested in artificial game players. It has been a popular topic of research since Arthur Samuel's work with checkers in 1959. Many different techniques and algorithms have been developed for different types of turn based strategy games, and research continues to this day for certain more complex games such as Go. There has been a great deal of success in the field of strategy game playing over the years; however, while there are certainly examples of researchers attempting to model the way humans play the games, most of the work has been performed with the goal of creating a player that could compete at the highest levels of competition. I did want the player that I designed to be competitive, but I wished to accomplish this goal through modelling the ways that people play games. It would however be remiss to not mention the multiple successes that have been accomplished in the field of artificial strategy game players.

2.1 Checkers and Minimax

Among the first and most influential studies of artificial strategy game playing was the work done with checkers by Arthur Samuel in his paper "Some Studies in Machine Learning Using Checkers." In 1949 Samuel began working on a computer program that

could play checkers. He was interested in checkers because it was relatively simple, but it had a depth of strategy. Similar to myself, Samuel believed that understanding games could be useful because it would allow people to build upon the work in order to approach general problems. Samuel's work is actually a perfect example of this phenomenon as his signature minimax and learning algorithms have been applied to all sorts of different planning problems.

The main contribution from Samuel's work was his minimax algorithm. The minimax algorithm is a form of tree search where each level of the tree represents a player, and each player is attempting to optimize a different objective function. The objective function is a numeric value assigned to each board state that determines how likely it is for a given player to win. For instance, in checkers a simple objective function for the red player might be the number of pieces red has captured minus the number of pieces that black has captured.

The algorithm works so well for turn based games like checkers because the objective function for one player can be represented as the negation of the objective function for the other player. Taking the example above, the objective function is greater when red has captured more pieces than black, implying that red is winning. The negation of this function is the number of pieces black has captured minus the number of pieces red has captured. This function is greater when black has captured more pieces than red, implying that black is winning. This feature stems from the fact that one player winning causes the other player to be losing.

The way that the algorithm works is that a search tree of all of the possible moves over the next few turns is generated. An objective function is then used to evaluate the board state at each of the leaves of the tree. The algorithm then assigns each level of the tree either a minimize or maximize function. This minimize or maximize function is used to decide which direction the player will most likely move. The algorithm starts at the leaves of the tree, which are already assigned a number, and assigns values to the second lowest layer equal to the best value that can be obtained for that level's optimization function from the children of that node. This process is then repeated until the top node, representing the current state of the game, is the only node yet to be assigned. The player then makes the move that would allow them to obtain the optimal value. A visual representation of the algorithm can be seen in figure 1 below:

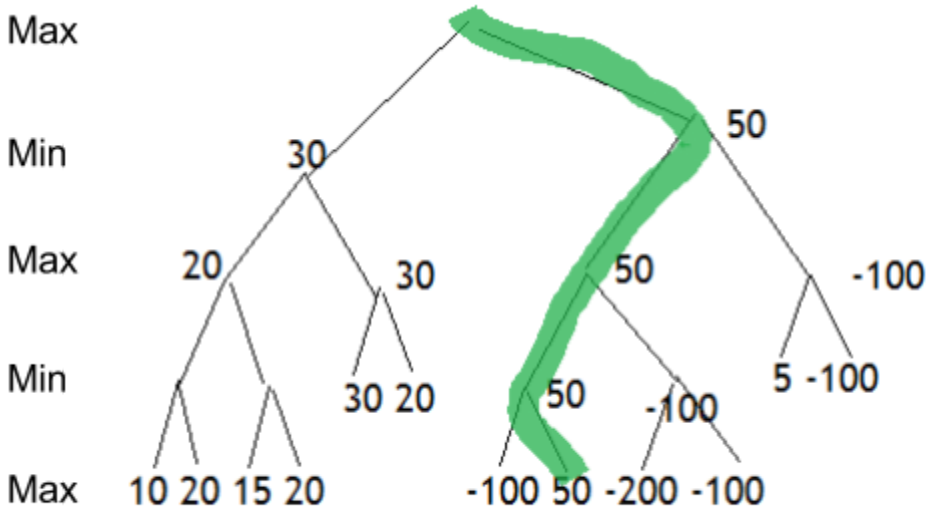

Figure 1: A visual representation of the minimax algorithm. The leaves would be filled out ahead of time and each level is assigned a value that optimizes the decision rule written to the left of the tree. The optimal path is highlighted in green.

While the minimax algorithm was undoubtedly Samuel's largest contribution to the field of artificial intelligence, there were a few other useful ideas that he implemented in his checker playing algorithm. Samuel introduced the idea of rote learning. Essentially, the program would remember positions that it had already seen along with the terminal value of these positions in order to artificially extend the length of the search tree. This gave his program better performance at a time when computational power was such a scarce commodity. In addition, while Samuel, a mediocre checkers player at best, developed the original scoring function for his program, later iterations had professional checkers players help design the scoring function. Both of these ideas would later be used to allow computers to play more complicated games at a high level (Samuel 1959).

Samuel's program wound up becoming the premier artificial checkers player by the time he ended the project in the mid-1970's. While this level of play would not be considered an accomplishment today, it was the first program to defeat proficient human players in any turn based strategy game. The minimax algorithm continued to be used in different forms for many similar projects for years after Samuel completed his research, and the program paved the way for later ventures in artificial game playing and artificial intelligence as a whole.

2.2 Chess and Alpha-Beta

While checkers is certainly an interesting game, Samuel chose checkers to study due to the relative simplicity compared with other turn based strategy games. The

minimax algorithm is an incredibly powerful tool. However, it could potentially take a massive amount of time to run depending on the complexity of the game and the depth of the search tree. A game of checkers has approximately 10 different moves that the computer must choose from in any given state of the game. This mean that on a single ply a player will have an average of 10 different possible moves that they can make. A ply is one player's turn, which in checkers or chess is defined by moving a single piece. This means that if a player wants to look 5 ply ahead, that player would have to look at all of the possible moves they could make, all of the possible responses their opponent could make to each move, and continue this train of thought for 5 turns.

Because the player has to consider all possible responses to all possible moves, and checkers has approximately 10 possible moves per ply, there are approximately 10^5 different board states that could result from looking 5 ply ahead, and each board state must be evaluated by the objective function. Chess has approximately 30 possible moves that the computer must choose from in any given state of the game. This means that if a player wants to look 5 ply ahead in chess, there are 30^5 different board states that must be evaluated. This number becomes incalculable very quickly, and chess was considered unconquerable for many years. As late as 1976 Senior Master and professor of psychology Eliot Hearst of Indiana University wrote that "the only way a current computer program could ever win a single game against a master player would be for the master, perhaps in a drunken stupor while playing 50 games simultaneously, to commit some once-in-a-year blunder."

However, despite the incalculable number of states that must be assessed in order use the minimax algorithm to play a game of chess, researchers refused to give up and the challenge only made them try harder. Alexander Brudno came up with a variation on the traditional minimax algorithm called alpha-beta pruning that would propel turn based strategy games into the new era of artificial intelligence (Brudno 1963). The idea of alpha-beta pruning is that if you evaluate the possible moves in order from left to right, there are often times where a certain branch of the tree will never be taken. This knowledge allows the algorithm to ignore branches that will never be taken. An example to better explain this concept is shown in figure 2 below:

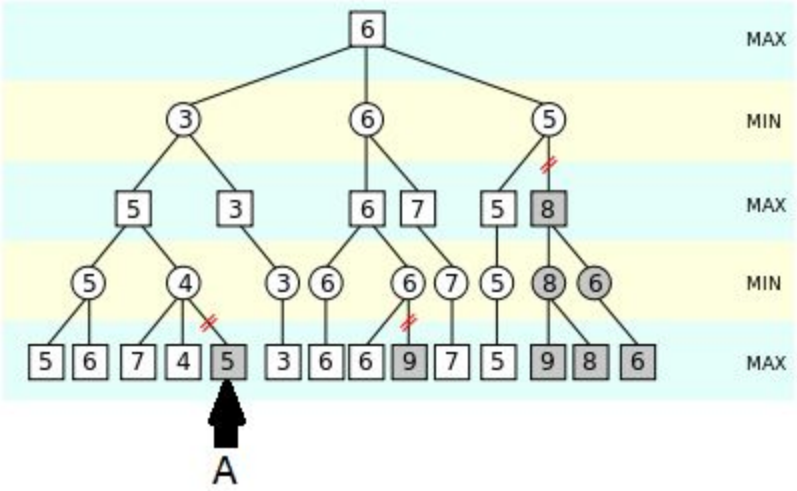

Figure 2: A min-max tree being searched by the alpha -beta algorithm. The greyed out branches are never evaluated due to the pruning of the algorithm.

In the figure above the greyed out branches are never evaluated by the alpha-beta algorithm. I will explain how this applies to the node marked A as an example. Each parent observes its children nodes from left to right. Therefore, the first

leaf nodes evaluated are the 2 leaf nodes at the bottom left hand corner of the image. After evaluating these leaf nodes, their parent finds it can achieve a minimum value of 5. That node's parent finds that it can achieve a maximum value of 5 or greater because one of its children has a value of 5. Node A's parent then evaluates its first child, which has value 7, and determines that it can achieve a minimum value of 7 or less. Next it evaluates its second child, which has value 4, and determines that it can achieve a minimum value of 4 or less. This node's parent already knows that it can achieve a maximum value of 5 or greater, and this node can achieve a minimum value of 4 or less. Therefore, there is no value of node A for which node A's parent will be the path chosen. The algorithm can then save time by not evaluating node A.

This pruning process saves a significant amount of time. With an intelligent ordering of nodes, the algorithm can actually prune close to half of the branches in a large search tree. This allows computer strategy game players to search much farther in a tree and make more intelligent moves without taking more time for each move. While the increase in computing power over the past 30 years has been massive, computers would likely have had significantly more difficulty playing chess at a high level had it not been for alpha-beta pruning.

One of the greatest accomplishments in the field of artificial intelligence did in fact come in the form of a chess player using the alpha-beta algorithm. Deep Blue, a chess program developed by IBM, has become one of the hallmark success stories in artificial intelligence. In 1996 Deep Blue managed to become the first computer program to win a game against a world champion chess player when it lost a 6 game match 4-2

against Gary Kasparov, and just a year later Deep Blue managed to defeat the world champion picking up 3 wins and a tie to take a 6 game match. While Deep Blue had one of the most complex heuristic functions ever developed and a team of professional chess players and performance engineers on its development team, the core of the program was just the alpha-beta algorithm being run on a massive supercomputer. Without the algorithm, programs might still be having trouble playing chess at the highest levels (Campbell, Hoane, Hsu 2002).

2.3 Go, Monte Carlo, and Neural Networks

After Deep Blue's 1997 victory, the general belief was that computers would soon completely dominate the world of strategy games. However, there was one important strategy game that proved far harder to crack, Go. From a purely tree search standpoint, Go is significantly more complex than both chess and checkers. As mentioned previously, checkers has about 10 possible options per move and chess has about 30 options per move. Go on the other hand can have almost 200 options in early game situations. Go has also proven significantly more difficult to develop a heuristic function for because the game is far more focused on positioning and far less focused on material advantage. Many professional Go players have sighted intuition as a key factor to success in Go, and intuition is far more difficult to replicate than a proven, well defined strategy.

However, while the traditional techniques may not have had the power to play Go at a high level, the power of human ingenuity proved itself once again as researchers

developed a plethora of strategies more fit to the playstyle of Go. The first major successes with Go playing algorithms began appearing with the advent of Monte Carlo randomized tree search. The first paper on using Monte Carlo tree search for Go playing appeared in 2006 when Rémi Coulom published his paper, “Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search.” In this paper, Coulom discusses the current state of the Monte Carlo algorithm along with the improvements that he made in order to allow the algorithm to better play strategy games, along with coining the phrase Monte Carlo tree search.

The basic idea behind Monte Carlo tree search is that instead of developing a specific heuristic and creating a minimax tree for each state of the game, the algorithm starts with little or no knowledge of the strategy behind the game. It knows the current state and the rules of the game, but that is basically all that it knows. The algorithm then plays out random games until it finds a winner in order to explore the space of moves that it can make. The algorithm remembers whether it won or lost after making each move, and uses this knowledge to more intelligently choose a move the next time. It runs through many such games with this randomized decision making process until it has decided that it has explored a large enough portion of the tree to have an idea of what a good move would be. Finally, it makes the move that it has found to be most successful in its simulations.

The algorithm has a few key advantages over traditional minimax tree search:

- The developer does not need to supply a heuristic function because the algorithm will figure one out on its own.

- A larger state space, such as the one present in Go, can be explored without taking too much time or not looking farther than a few moves ahead.
- The algorithm can even self improve by remembering the results of exploring the game tree the last time it found itself in the current state.

All of these advantages helped Monte Carlo tree search players succeed in Go at levels that had never been previously achievable. However, while Deep Blue had defeated the chess world champion, until recently the most successful Go playing algorithms were only playing at a high amateur level (1-7 Dan).

Within the last few months a new Go playing algorithm, AlphaGo, has been able to defeat the world's top Go players. AlphaGo uses Monte Carlo tree search, but it combines this strategy with a neural network that is capable of learning complex intricacies in the game in order to enhance its play. A neural network is a form of machine learning that has been gaining immense popularity lately in the fields of vision and natural language processing. Neural networks are very powerful because they can learn a concept without having to be supplied any prior information by the developer, and they are universal classification tool. This means that they are theoretically capable of learning any finitely complex objective function, such as the intuition behind professional Go players actions.

AlphaGo was trained by observing professional Go players and playing thousands of games against itself in order to figure out what strategies are effective. AlphaGo is so powerful and effective because it is able to improve and learn from its

mistakes between every game. The *Nature* article, “Mastering the game of Go with deep neural networks and tree search,” discusses the intricacies of how the algorithm works. In March, AlphaGo was able to defeat Lee Se-dol, 18 time Go world champion, and solidify itself as the premier artificial strategy game player in the world. AlphaGo’s victory has shown that there may be no limit to what artificial intelligence is capable of accomplishing.

3 A Game Only Humans Can Play

Most of the research in artificial intelligence can be broken up into two schools of thought, goal based thinking and method based thinking. The major difference between the two is that goal based thinking seeks to optimize some parameter in order to perform optimally at a given task. An example of this form of thinking would be Deep Blue, IBM's chess playing program mentioned earlier. The goal of Deep Blue was to play chess at the highest level possible. Success was measured by whether Deep Blue was able to defeat world class chess players not what algorithms were employed in order to succeed.

An example of method based research would be the work done by Patrick Winston at MIT in the field of story understanding. Winston's goal is to create computers that can understand stories on an intellectual and emotional level. He believes that humans learn a great deal through stories and story understanding is one of the keys to artificial intelligence. His work is mainly method based thinking because he is attempting

to imitate human understanding of stories rather than attempting to pull every possible detail out of the story. For instance, he refuses to utilize popular machine learning methods because he believes “we need biological inspiration to help us understand nonsymbolic computing.” Additionally, Winston put a significant amount of effort into designing a system that could explain how it came upon its train of thought. He took the time to implement this feature because he believes that humans learn a great deal from explaining their thoughts to others. A vast majority of his work is inspired by human thinking rather than accomplishing a certain goal (Winston 2012). The two major methodologies underlie much of the work done in artificial intelligence, and my project falls distinctly into the latter field of thought.

The goal of my project is to imitate human planning in strategy games rather than developing a system that is proficient in strategy games. However, a majority of the work done in artificial intelligence has been goal based rather than method based because goal based development is a more natural way to solve problems. In order to bridge this gap, a common technique among method based researchers has been to develop a goal that requires the desired method to be completed. Most artificial strategy game players are severely limited by the branching factor, the number of possible moves from each state, of the game being played. Chess is considered more difficult than checkers because the branching factor is about tripled, and Go was considered unconquerable until recently due to the massive branching factor associated with the game. Therefore, in an attempt to make a strategy game player that was more

human-like, I created a game with the purpose of being difficult for traditional strategy game playing algorithms to play well.

The game that I created is called Tarble. Tarble is a 2 player competitive strategy board game where each turn is broken up into N distinct moves where N is the result of a roll with a 6 sided die. Each of the N moves has approximately 50 options, and each option leads to a new distinct board state with a similar number of options. Therefore, a roll of 6 leads to a turn with a branching factor of about 30,000,000,000 (compared with the branching factor of Go which is about 200). However, while all of the traditional artificial strategy game playing algorithms that I have mentioned failed to play Tarble at a reasonable level, humans seemed to have little trouble picking up the nuances and basic strategies of the game. With such an immensely computationally complex game, I was able to concentrate on developing the best artificial player that I could create for Tarble, and this player necessitated imitating the thinking styles that I observed in humans because other techniques were simply ineffective.

3.1 Tarble

The game of Tarble is a two player competitive strategy game played on a hexgrid with side lengths of 6. Both players begin the game with 7 pieces, 4 in the middle of the player's back row and 3 in the middle of the player's second to back row.

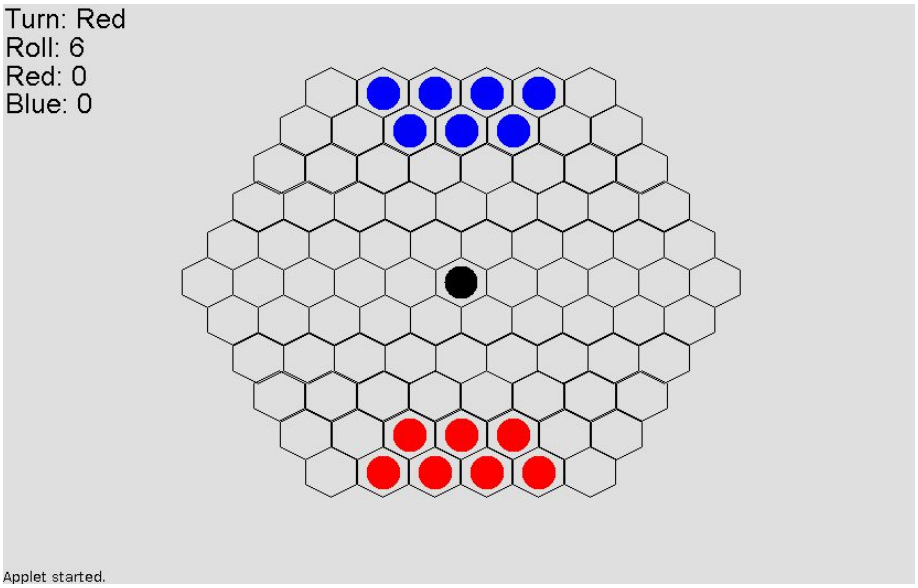

Figure 3: The starting state of a Tarble game. It is currently the red player's turn. The first roll was a 6.

There is also a neutral piece placed in the center of the board. During a player's turn, the player rolls a 6-sided die and must make as many moves as the result of the die roll. A move consists of moving any of the player's pieces a single unit in any direction or jumping another piece. Jumping another piece is defined as moving from one side of a piece to the other side (a 2 unit move) and removing the piece. Only opposing and neutral pieces can be jumped.

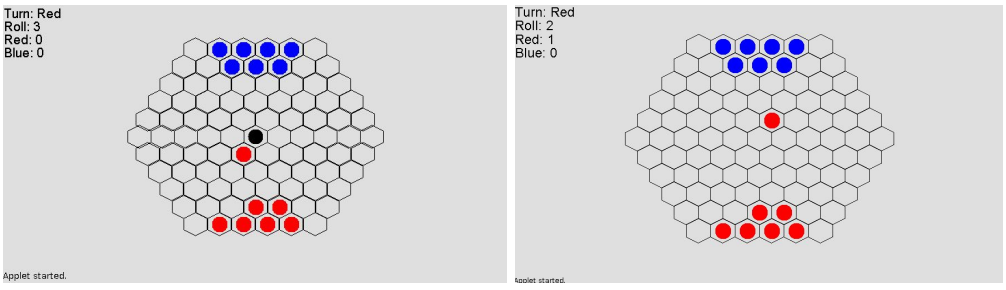

Figure 4: The red player starts in the situation on the left. The player jumps the center piece, scoring 1 point and removing the jumped piece from play resulting in the situation on the right.

A player may not move to the location of a current piece of any type. For instance, if a player's opponent had a piece located on the other side of the neutral piece from the player's piece, the player would not be able to capture the neutral piece on that move.

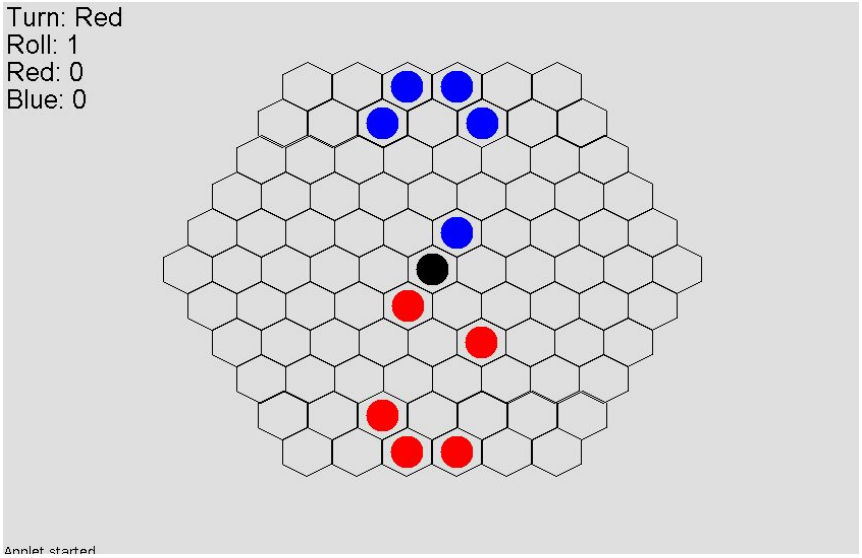

FIgure 5: Red can not capture the neutral piece because the blue player has a piece at the location the capturing piece would land.

Any pieces located on a corner or in the center of the board are removed at the end of the controlling player's turn. The neutral piece is replaced at the end of each player's turn as well if it has been captured.

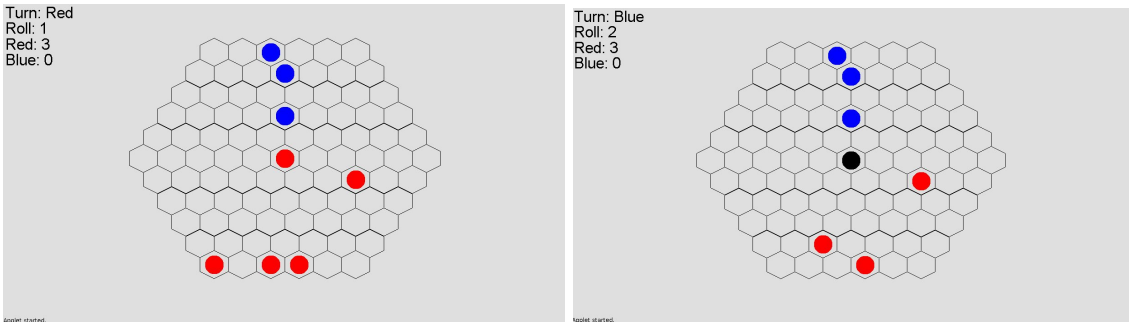

Figure 6: The game starts in the situation pictured on the left. After the red player moves it proceeds to the situation pictured on the right. The red player has a piece in the center and a piece in the bottom left-hand corner of the board. Both of these pieces are removed after the player's turn. Additionally, the center piece is replaced after the red player completes their turn.

A player receives 1 point for jumping any piece, opposing or neutral, (pieces removed by ending a turn on a corner or center location do not result in points for either player) and the winner of the game is the player with the most points at the end of the game. The game ends when either player has no pieces remaining on the board. In the event of a points tie, the player with pieces remaining on the board wins. In the event of a points tie, both players have no pieces remaining on the board, the game ends in a tie (while theoretically possible it has never occurred in a game of Tarble that I have observed).

By designing my own game, I had the freedom to adjust the game play to better study the features of human intelligence that I was interested in studying. I wanted a strategy game because these games offer the purest form of insight into human planning (as opposed to a more reaction based game where physical characteristics could have affected the study). I also introduced a significant measure of randomness

into Tarble, the random die rolls at the beginning of each turn, because I was interested in how humans dealt with randomness. Randomness also hurts many of the common strategy game algorithms, such as minimax and alpha-beta tree search, because these algorithms assume a deterministic expansion of the game tree based off each player making the move that most benefits him/her in any given situation. Most importantly, as described previously, Tarble has a branching factor far larger than any other turn based strategy game that I have found. This feature thwarts many of the common strategy game playing techniques and forces me to utilize other methods in order to create a competitive artificial Tarble player. Finally, by creating my own game, I am able to collect an unbiased testing group. Many people have experience with games such as chess and Go, and even if I found people with very little or no experience with these games, these people still might know various pieces of accepted strategy from popular culture. However, no subjects that I studied had ever played or heard of Tarble prior to my study. Therefore, I was able to observe players learn and explore the game unencumbered by preconceived notions of what consisted of a good strategy.

4 Common Artificial Intelligence Techniques

While Tarble was specifically designed to make it difficult for common machine learning techniques to play, the only way to support this statement is to actually implement these algorithms. I implemented the minimax, Monte Carlo randomized tree

search, and neural network algorithms in order to support the claim that commonly used strategy game playing algorithms are ineffective at playing Tarble.

4.1 Minimax

The first algorithm that I implemented in an attempt to design an artificial Tarble player was the minimax algorithm. I did not need to implement the alpha-beta pruning optimization because I knew that I was not going to be able to look further than a single turn ahead. Because the alpha-beta algorithm prunes based off of the opposing optimization techniques of the minimizing and maximizing players, the alpha-beta algorithm would act identically to the basic minimax algorithm when only looking ahead a single turn.

While Tarble does have a massive branching factor, the game does have a notable advantage over certain other games that have proven difficult for the minimax algorithm such as Go. Due to the point based scoring of Tarble, it is not difficult to develop a reasonable heuristic algorithm. Material based heuristics have proven very effective in chess and checkers, and these simplistic heuristics are a large part of what made chess and checkers playable with look-ahead algorithms. Go on the other hand does not have an easily expressible heuristic. Many top Go players have described their plays as intuitive more than anything else, and intuitive play is difficult to capture in a general heuristic function.

However, despite a relatively simple heuristic function, the minimax algorithm is simply unrealistic for Tarble. The algorithm that I implemented was only looking a single

turn ahead. It was not even considering the possible responses by the opponent. Yet, on a roll of six the algorithm still took an unacceptably long time to run, understandable as there are over 20,000,000 possible moves with a roll of 6 from the starting state of the game. The opening move took about 2 minutes with a roll of 6 and as the game progressed this increased to as much as 10 minutes for a single move. Furthermore, under certain conditions, generally when both players had spread their pieces out across the board but without capturing many pieces, the program would even throw an out of heap space exception as the number of moves increased to greater than the maximum size of a Java Hash Set, the data structure that I was using to store the possible moves. While I could have developed an on-line algorithm to evaluate all of the possible board states, this would have taken an even greater amount of time to run, and the runtime was already greater than what I would consider an acceptable amount of time.

The big problem with the minimax algorithm is that there are simply too many possible moves to consider on a single turn when the roll is large. The key insight is that this restriction only applies for rolls of 5 or larger, and Tarble conveniently plays in a recursive fashion. Therefore, a roll of 5 can be thought of as the player getting 2 turns in a row where the player first rolls a 4 followed by a 1. As a result, I implemented a minimax algorithm that only looked a maximum of 4 movements ahead. If the roll was a 4 or less, the algorithm looked ahead an entire turn. However, if the roll was a 5 or a 6, the algorithm would only look ahead 4 movements, choose the best move, and then look the remaining number of movements ahead in order to finish the move. This

adjustment fixed all of the runtime issues that I was having with the original minimax algorithm.

Even though the algorithm was now runnable in an acceptable amount of time, I never observed the algorithm taking more than 30 seconds on a single turn, and the out of memory exceptions had been removed, the player that resulted from this algorithm was not even close to competitive with humans. I tested the algorithm against 4 human players who had never before played Tarble. Each player played a best of 5 games match with the algorithm. The algorithm did not only lose 15 straight games, it lost a majority of the games by 10 points or more (a major blowout in Tarble). While I did test many different iterations of heuristic functions, none of the functions seemed to make a major difference in the ability of the algorithm.

The minimax algorithm attributes much of its success to the ability of the algorithm to look further ahead than humans would ever be able to look. Thus, even a minimax player without a complex checkers heuristic function can be successful because it is looking farther ahead than the humans that it is playing against. My algorithm was not even looking a full turn ahead. Humans are more than capable of planning out a full turn, but the algorithm has no way of filtering moves that can be considered ridiculous. Therefore, the minimax algorithm is forced to evaluate every possible move, which due to the massive branching factor of Tarble eliminates the major advantage that the algorithm provides the computer.

4.2 Monte Carlo Tree Search

The second algorithm that I implemented in an attempt to make a competitive artificial Tarble player was the Monte Carlo tree search algorithm. This algorithm has had a great deal of success playing Go recently, and I believed that the results would be significantly better than the minimax algorithm. Monte Carlo tree search had been developed specifically for the purpose of playing games with a large branching factor.

The basic Monte Carlo algorithm required me to make a few decisions that were not explicit from the problem that I was trying to solve. When I implemented the minimax algorithm, it was clear that 10 minutes was too long for a single turn, as this is far longer than any human takes, but 30 second was not an issue. The Monte Carlo algorithm performs better the more time that it is allotted to expand the search tree. I decided that 3 minutes, still longer than any of the human players that I observed took for a turn, was an acceptable amount of time for a move. I was also unable to list all of the possible moves that the algorithm could make for its turn and have the algorithm randomly choose one because the algorithm would be unable to complete even a single randomized play through using this common technique. Therefore, the algorithm listed all of the possible game states 1 movement (only moving a single piece) ahead, and the algorithm chose one of these options during its randomized playthroughs. The algorithm would then look a full turn ahead and choose the best move of all of the paths that it had explored when making its actual move for the turn. I felt that this optimization would be beneficial because it would allow the algorithm to explore all of the possible game trees as quickly as possible while sacrificing as little strategic knowledge as possible.

I began tested this algorithm by allowing it 3 minutes to explore the game tree, the amount of time that I had previously deemed acceptable for a single move. While the algorithm performed far better than the minimax algorithm, it was still not even close to a competitive level for human players. The Monte Carlo algorithm managed to defeat the minimax algorithm in 12 out of the 13 games that I played them against each other. However, as I watched the games, there were clear mistakes that I noted in the Monte Carlo player's moves. On multiple occasions the Monte Carlo algorithm failed to make obvious capturing moves. I believe that its advantage over the minimax algorithm stemmed mainly from its ability to see the bigger strategic picture of the game rather than just the single move it was making. I tested this algorithm against only a single human player because after watching the games against the minimax algorithm, my own testing against the player, and the Monte Carlo algorithms defeat in 3 straight games by over 10 points to a brand new Tarble player, I knew that the algorithm could not compete with human players.

The shortcomings of the Monte Carlo player stemmed mainly from its inability to search a significant enough portion of the game tree in the allotted time. This can be seen from the algorithms failure to make obvious capturing moves on multiple occasions where any reasonable human player would have certainly executed the capture. However, later in the game when there were less pieces remaining on the board the algorithm played far better than it did in the early game. I believe this improvement to come from being able to search a larger percentage of the search tree in the allotted time due to the game having a significantly smaller search tree when

there are less pieces on the board. In order to counteract this issue, I allowed the algorithm to think for up to 10 minutes before making a move (far longer than I felt was an acceptable amount of time and the amount of time that the minimax algorithm took before throwing out of memory exceptions). While the player allowed to think for 10 minutes defeated the player allowed to think for only 3 minutes in 5 out of 5 games, 10 minutes was still not enough time as many obvious moves were still missed in the early game and the player was still unable to play competitive game with me (it lost 3 straight games by 8, 7, and 13 points respectively). While the Monte Carlo tree search algorithm clearly played at a higher level than the minimax algorithm, it was still unable to play Tarble at a human level even when given significantly longer than the human to decide upon its moves.

4.3 TD-Gammon

The final algorithm that I tested in an attempt to utilize common machine learning techniques to play Tarble was the TD-Gammon machine learning algorithm. In 1992 Gerald Tesauro approached the problem of finding an ideal heuristic function from an entirely different perspective. Up until this time, heuristic functions had generally been determined by recruiting experts in the game of interest and having them develop an easily computable heuristic for any possible state of the game. Nobody had up until that point created a successful backgammon player due to the number of possible moves from each state of the game and the difficulty in creating an accurate heuristic function.

Tesauro decided to approach the problem from a novel machine learning point of view (Tesauro 1995).

Tesauro began by randomly initializing a neural network. The inputs to the network did nothing more than describe the state of the game. He then had the neural network play games of backgammon against itself. The neural network played by looking 2 ply ahead, the current player's turn followed by its opponents response, inputting the possible board positions into the neural network, using the output of the neural network as the heuristic function, and using alpha-beta to find the ideal move. After each time step, a single move by one player, the network would update itself using the temporal difference algorithm:

$$w_{t+1} - w_t = \alpha (Y_{t+1} - Y_t) \Sigma^{t}_{k=1} \lambda^{t-k} \Delta_w Y_k$$

where: w_t is the weight matrix for the neural network, Y_t is the output of the neural network with an input of the current game state, α is a small constant called the "learning rate" parameter (essentially equivalent to step size), and λ is a constant that affects how much board evaluations should feed back to previous estimations. $\lambda=0$ makes the program correct only the previous turns estimate, and $\lambda=1$ makes the program attempt to correct the estimates on all previous turns. Values of λ between 0 and 1 specify different rates at which the importance of older estimates should "decay" with time. Temporal difference learning can be thought of as a version of stochastic

gradient descent that takes previously observed training points into account as well as the current training point (Tesauro 1995).

After 300,000 training games, which took months to complete, TD-Gammon had become the most successful backgammon playing program ever created. It dropped less that 0.25 points on average per game against various professional backgammon players across 51 games. This would be equivalent to an advanced human level of play that could expect to have a decent chance of winning a local or regional tournament. These results are in contrast to other top backgammon programs at the time, which would drop about 1 point per game on average against top human players, equivalent to a weak intermediate level player (Tesauro 1995). There were later experiments run where more complex inputs were designed and moves were considered out to 3 ply all with increasing success, but for the purposes of this paper I will be focusing on the initial experiment described above.

4.3.1 Tic-tac-toe

The temporal difference algorithm is significantly more complicated than either of the other algorithms that I implemented. Therefore, I decided to implement a Tic-tac-toe player in order to test the algorithm before releasing it on Tarble. I was able to do this due to the features of the algorithm. As long as I can represent the two games in a one dimensional array, which can be used as the input to the neural network, the algorithm does not require any specific adjustments by the designer. Tic-tac-toe is interesting due its simplicity. I was able to rapidly iterate on the algorithm because I can evaluate the

results of a Tic-tac-toe algorithm by simply observing it play. This allowed me to work out the bugs while it played Tic-tac-toe, and then only move on to Tarble once the algorithm was working as intended for the simpler problem. This algorithm is also interesting because it actually relies on the randomness of Tarble in order to function properly rather than being harmed by it as basic minimax and alpha-beta tree search are. I will go into more depth later in the paper about how this randomness affects the performance of the temporal difference algorithm.

4.3.2 Designing a Neural Network

I programmed my own neural network for this project in Java. I had originally planned to use Neuroph, Java's open source neural network library, for this project, but after further research I found that I could not change the weight updating algorithm to use the temporal difference algorithm. I was therefore forced to implement my own version. Computational efficiency was a major consideration due to the massive number of games and possibilities that would have to be evaluated. In order to train the neural network in a timely manner I used Java's DoubleMatrix library, a multi-threaded matrix library written in Fortran that performs the fastest matrix multiplication possible in Java, for storing weights and performing forward propagation. I also used my own multi-threaded implementation of backpropagation that shards by the weights that need to be updated in each layer for calculating the gradient of the output with respect to the weights.

4.3.3 Playing Tic-tac-toe

Once I was satisfied with my implementation of a neural network, I designed a simple interface for playing Tic-tac-toe in Java. I designed a player that uses alpha-beta and looks at all of the possible moves 2 ply ahead of the current game state. It then turns all of the possible game states into 1x10 vectors that can be used as inputs to the neural network. The output of this neural network is used as the objective function for alpha-beta. 9 of the dimensions represent positions on the board, and the final dimension represents whose turn it currently is. I use a 1 to represent the first player, a -1 to represent the second player, and a 0 to represent an empty board position (it must always be someone's turn). I then feed this vector into the neural network and choose the optimal move. Player 1 is trying to maximize the output and player 2 is trying to minimize the output. After every move I update the neural network according to the temporal difference algorithm. If player 1 wins the game, the final state, Y_{t+1} on the final iteration of the temporal difference algorithm, is 1, and if player 2 wins the game, it is 0. I begin by randomly initializing the neural network with weights between -1 and 1. I ran all of these algorithms to convergence where I defined convergence as when the algorithm plays 2 identical games against itself 1,000 games apart.

I ran the algorithm, which continuously played games of Tic-tac-toe against itself and updated after each game. However, the games were taking longer than I expected (about 75 games per second). This was not an issue for Tic-tac-toe, but I knew that once I started playing Tarble the algorithm would be unusably slow. The reason that it was so slow was that I was calculating the gradients after every move, up to 9 times per

game. In order to solve this problem I decided to make a small adjustment to the temporal difference algorithm. I first tried:

$$w_{t+1} - w_t = \alpha\,(1(\text{winner}) - Y_t)\,\Sigma^{t}_{k=1}\lambda^{t-k}\,\Delta_w Y_k\,, \text{ for all } t$$

where 1(winner) is the indicator function equal to 0 if player 2 wins and 1 if player 1 wins. I used this equation once at the end of the game, and I calculated each gradient once by storing a variable called gradient sum. After each update I would multiply gradient sum by λ and add the next gradient. I ran the algorithm with this equation, but found that the neural network did not learn to play a reasonable game of Tic-tac-toe.

The reason that it struggled was that my algorithm had a fundamental flaw. I calculated each gradient once for a given set of inputs and outputs, but as the weights are updated, the gradients change. To solve this issue I needed to make a second update to the algorithm. I now broke the temporal difference algorithm down to what it was attempting to accomplish. The temporal difference algorithm is attempting to update the weights such that the neural network prefers states visited by the winning player and avoids states visited by the losing player. It also puts more weight on more recent states. After realizing this goal, I updated the algorithm one more time to look like the following:

$$w_{t+1} - w_t = \alpha\,\Sigma^{t}_{k=1}\,(\lambda^{t-k}\,\Delta_w Y_k\,(1(\text{winner}) - Y_t))$$

This algorithm was run a single time at the end of each game, and it does exactly what I realized was the goal of the temporal difference algorithm. The λ^{t-k} term ensures that more weight is placed on later states, and the $\Delta_w Y_k (1(\text{winner}) - Y_t))$ term moves the network towards preferring winning states and avoiding losing states.

I now had to find the parameters α and λ that I would use for this algorithm. I found that with an α greater than 0.005, the algorithm would not converge in a reasonable amount of time and a λ less than 0.5 failed to lead to interesting early game discoveries (such as the middle and corner positions being better first moves than the middle edges). I therefore only considered these ranges when evaluating α and λ. The results from this experiment can be seen in figures 7 and 8 below. I first found the ideal α with λ=0.7 because α has a greater effect on the time to convergence than λ. I found α=0.001 to be ideal. I then varied λ with α=0.001 and found λ=0.9 to be ideal (note that figure 7 shows varying alpha with λ=0.9 for illustrative purposes, and α=0.001 is still the ideal value for α).

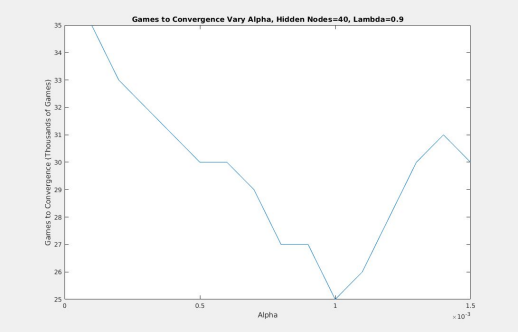

Figure 7: I varied the value of alpha in my adjusted temporal difference algorithm in Tic-tac-toe with the optimal values of lambda and number of hidden nodes. I found

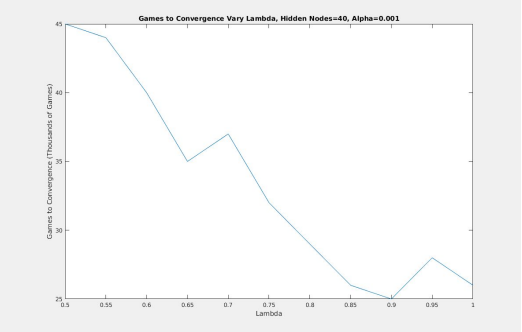

Figure 8: I varied the value of lambda in my adjusted temporal difference algorithm in Tic-tac-toe with the optimal values of alpha and number of hidden nodes. I found

the optimal value of alpha to be 0.001, which converged after 25,000 games.

the optimal value of lambda to be 0.9, which converged after 25,000 games

Finally, I varied the number of hidden nodes present in the neural network. I found that with less than 20 hidden nodes, the algorithm would fail to converge to a reasonable, tying, state. I also realized that while the number of games to convergence monotonically decreased with the number of hidden nodes used, the time to convergence did not. This behavior is because the time per game increases with the number of hidden nodes. A graph relating number of hidden nodes to convergence time is shown in figure 9 below. All of these tests were done with λ=0.9 and α=0.001 I found 40 hidden nodes to be the ideal number. With these parameters the algorithm converged after 25,000 games.

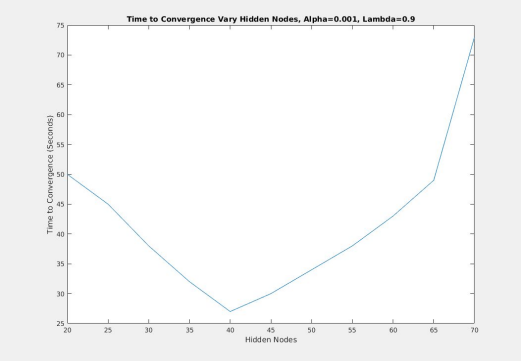

Figure 9: I varied the number of hidden nodes in my adjusted temporal difference algorithm in Tic-tac-toe with the optimal values of lambda and alpha. I found the optimal number of hidden nodes to be 40, which converged after 27 seconds.

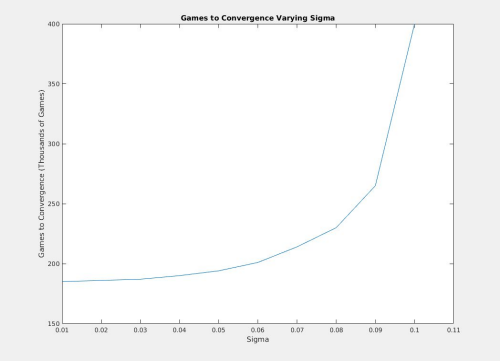

Figure 10: I varied the value of sigma in the Gaussian noise that I used to select a move in Tic-tac-toe with the optimal values of alpha, lambda, and the number of hidden nodes. I tried to balance the time taken from increasing sigma with the added exploration that I obtained from the randomness. I chose 0.05 as

the sigma that I would use because I felt it balanced these concerns well. This sigma led to convergence after 194,000 games.

4.3.4 Gaussian Noise

This new algorithm ran games incredibly quickly (about 1,000 games per second), and it converged to reasonable states where the games all ended in ties. However, after convergence, I played a few games against the computer to test its ability. I found it severely lacking. The computer seemed to choose moves almost at random, and it missed places where I had 2 pieces in a row regularly. I realized that it had not been given a chance to fully explore the state space of the game.

Backgammon conveniently has a die that introduces a random element to the game. This means that no two games of backgammon will be exactly the same, and the computer will see different states even if the exact same algorithm plays the game twice in a row. Certain academics, such as Jordan Pollack and Alan Blair, even believe that "TD-Gammon is successful because of the randomness of backgammon," which Tic-tac-toe does not have (Pollack and Blair 1997). I therefore decided to introduce a randomness to the game in order to more fully explore the state space.

Each turn the computer player now looks ahead 2 ply and evaluates every possible board state using the neural network, exactly the same as before. However, it

now multiplies each board evaluation by a Gaussian with mean 1 and varying standard deviations, and chooses the new min or max respectively. This process allows the algorithm to more fully search the state space by not always selecting the ideal move.

I then had to redefine convergence, which I did by running a deterministic game (one without the Gaussian noise) every 1,000 games and terminating when 2 consecutive deterministic games are identical. Convergence took much longer to achieve, but there were very interesting results.

Firstly, I had to decide on the proper variance to use. I found that with variances greater than 0.1, the algorithm does not produce reasonable results because the games are too random. It would sometimes converge to a state where one player would win for instance. A graph relating variance to time to convergence can be found in figure 10 above. I decided to use a variance of 0.05 because this allows enough noise to fully explore the space without taking too long to converge or leading to unreasonable results. It converged after 194,000 games.

4.3.5 Strange Convergence Game

When I began this project, I assumed that the algorithm would converge to a “perfect” game of Tic-tac-toe when the noise was removed. While all games should end in a tie, a “perfect” game, shown in figure 11, could lead to player 1 winning the game even if player 2 successfully blocks player 1’s 2 in a rows whenever possible. If player 2's second move is to a corner position, usually preferred to a center edge, player 2 will lose the game. However, the algorithm converged to a different solution seen in figure

12. I dismissed this behavior at first because when I played against the algorithm the game ended in a tie, and if I made a mistake it ended with a loss for me. Therefore, I assumed it was playing intelligently even if it was not converging to the expected state. However, the algorithm was actually performing better than the perfect solution that I assumed it would converge to. In the perfect solution, player 1 must block player 2 from winning twice before the game ends in a tie. Because the players choose moves in a noisy fashion, this behavior is not guaranteed. In the solution that the algorithm actually converged to, the only time that player 1 needs to block player 2 is when player 1 has no other choice and can therefore not make a mistake. Further, player 1 forces player 2 to block 3 times, which player 2 will not always be able to do due to the noise in the system. I found this result incredible because the algorithm actually discovered a better solution than I arrived at for the situation that I had asked it to optimize for, playing as and against this noisy player.

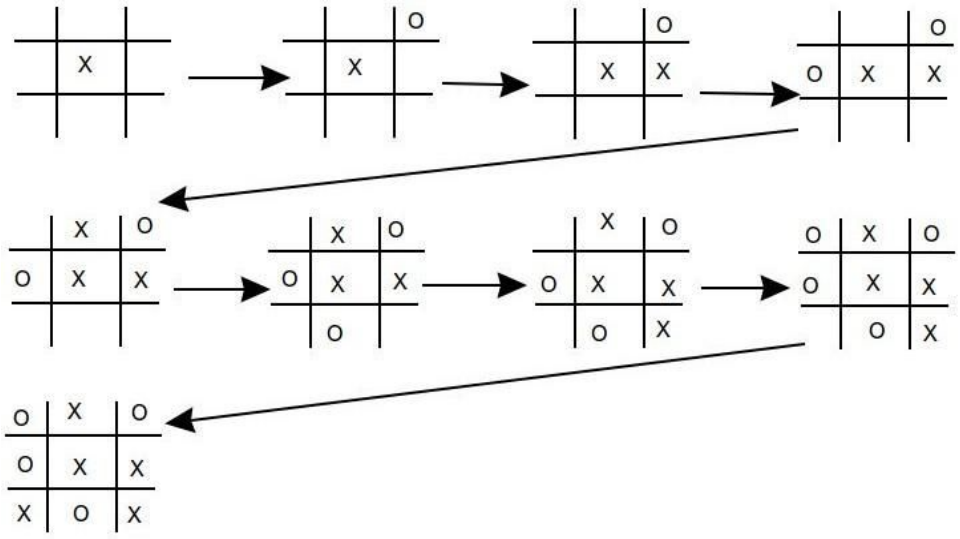

Figure 11: A “perfect” game of Tic-tac-toe.

This is a perfect game because if O's second

move is to a corner, O will lose, which may not seem obvious to a novice player. This is the only situation that could lead to a win assuming that both players block all 2 in a rows. However, notice that X must block O from winning twice (the last move does not count because there are no other options).

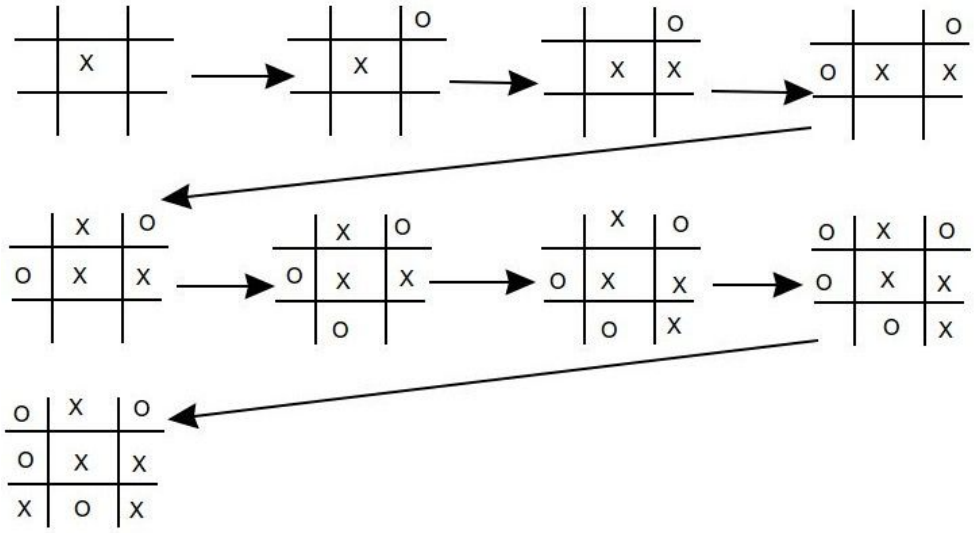

Figure 12: The deterministic game that my algorithm converged to. While all the O has to do in order to win the game is block all of X's 2 in a row, O will sometimes fail due to the noise added to each player's moves. Further, X never has to block a win from O until the final move when X does not have a chance to mess up. O must block X's win 3 times, the maximum possible.

4.3.6 Tarble

After testing the algorithm on Tic-tac-toe, I was able to use the neural network to play Tarble, a far more complex game. Tarble, like backgammon, uses a die to

determine the number of moves that players may make on his/ her next turn. This leads to two important factors, randomness and massive branching factors. The randomness is a very good quality illustrated by the fact that I had to artificially introduce it into Tic-tac-toe. However, the large branching factor makes it incredibly computationally intensive to perform alpha-beta search to a large depth. While Tesauro did perform search to a ply of 2 in his TD-Gammon project, the branching factor on Tarble is so large that even ply 1 is often too complicated to handle. There were a few tricks that I used in order to make Tarble a more reasonable game to perform temporal difference learning on. First I utilized the rotational symmetry of the board. Because 2 positions rotated 60 degrees are identical, I oriented the board such that rotating it 60 degrees would lead to the same input to the neural network. Using the rotational symmetry does not actually decrease the time required per game, but it should help the algorithm converge to a reasonable result more quickly. I also used the recursive nature of the moves to trim the branching factor significantly. Similarly to the strategies that I employed in minimax search, I looked ahead one movement rather than 1 turn. I essentially counted a roll of 4 as 4 individual rolls of 1. This technique allowed me to more quickly evaluate all possible next moves and make a selection. I used the same α and λ values that were found to be ideal for Tic-tac-toe because it would have been impossible to reevaluate the results of training in order to optimize for these parameters in a reasonable amount of time. I also used 50 hidden nodes because I figured that Tarble was significantly more complex than Tic-tac-toe and would therefore require more hidden nodes for computation. The input vector used was again only the minimal

amount of information required to describe the game. Each board position was given a dimension with the same -1, 0, 1 structure as previously. The score of each player, whose turn it currently is, and whether or not the black neutral piece in the center was captured were also inputs. I then oriented the board so that a 60 degree rotation would have no effect on the input vector.

4.3.7 Results

While I could not determine a reasonable convergence criterion for the temporal difference algorithm applied to Tarble, I ran the algorithm for 15 hours before deciding that it would be unreasonable to expect an algorithm to train for significantly longer. It ran over 276,000 games over the course of those 15 hours, and the later games went significantly faster than the early games due to the algorithm actually making relevant choices to advance the game. Early games could involve hundreds of turns before a single piece was captured, but in later games the algorithm was actively trying to capture its opponent's pieces and advance the game.

I then played against the player resulting from this training, and while the moves were clearly not being chosen randomly, the player played at what I would consider a sub-human level. It did not sacrifice any pieces and attempted to organize its pieces into reasonable defensive positions, but because the player did not look a full move ahead when making decisions, it missed some positions where it was clear that it could capture my pieces without me being able to capture back.

To solve this issue, I decided to use the neural network trained by looking one movement ahead, but I would allow it to look a full turn ahead when evaluating moves.

The only exception was when there were over $2*10^6$ possible moves after looking 5 movements ahead on a roll of 6. In these cases I would evaluate all of the possible roll 5 moves and choose only the top 1,000 to expand. While this algorithm ran much slower, as there were significantly more moves to consider, the algorithm still performed at what I would consider an acceptable human speed, about 30 second per move, and the play skill was significantly improved. I even lost 1 out of the 5 games that I played against the player, and while it definitely rolled significantly better than I did throughout that game, this game does mark the first Tarble loss that I had to a computer algorithm. I would put the play skill for this player somewhere around an inexperienced strategy game player who has never played Tarble previously. I also pitted this algorithm against the Monte Carlo tree search algorithm that I designed allowing the Monte Carlo player 10 minutes per move, and the temporal difference player won 10 out of 13 games. While this is certainly not conclusive evidence, I do believe that the temporal difference player is significantly more skilled, and it moves about 5 times as quickly.

4.3.8 Full Turn Training

After the success from looking ahead an entire turn while playing, I attempted to train the neural network by looking ahead an entire turn both during play and training. I used the same branch thinning strategy mentioned earlier for moves with too many options. I started running the trainer with this new strategy. Unfortunately, the algorithm had completed only a single game after an hour of running. I realized that the issue was that the players were making semi-random moves due to the random initial conditions of

the neural network. This meant that games would take hundreds of turns to complete.The long games were not an issue for the previous iteration of the neural network player, but these players took significantly longer per move, which yielded unreasonably long games. In order to prevent games from going too long, I decided to use my partially trained neural network from the single movement look ahead training as the initial conditions for this training method. The single movement look ahead network had proven to play a reasonable game of Tarble when it was allow to look ahead a full turn. Therefore, I was hopeful that the games would end in 20-30 turns, the average length of a Tarble game.

4.3.9 Mapreduce

The games were now running much more quickly, but they were still taking longer than I thought would be reasonable for retraining the neural network. The games were taking between 5 and 10 minutes, which would not allow me to get the sheer number of games that I would need in order for the neural network to relevantly update itself. One solution that I attempted was using mapreduce, a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster, to find the optimal next move. Mapreduce is a common technique used by engineers dealing with big data and was first proposed in Jeffrey Dean and Sanjay Ghemawat's paper "Mapreduce: Simplified Data Processing on Large Clusters." Because the decision of where to move next is simply a maximization problem, I was able to shard the possible next moves into 10 groups, find the best move

out of these groups, and then compare the 10 best moves to find the optimal solution. Because the optimal solution is guaranteed to have a larger evaluation value than the optimal solution from the other 9 groups and the optimal solution from each group is guaranteed to have a larger evaluation value than anything else in those groups, the optimal solution is guaranteed to have the largest evaluation value of all the options considered. Therefore, it would have been chosen by the simple maximizer that I had been using thus far. The games run using the partially trained initial conditions and the mapreduce algorithm finally ran at a speed that I was comfortable running over night.

4.3.10 Final Results

After 10 hours I stopped this final run. It had run through 957 games in the 10 hours that I had allowed the algorithm to run. I then played it against the neural network that looks ahead a full turn but had only been trained by looking ahead a single movement. This new neural network won 11 out of the 13 games that they played. However, when I played against it, I did not notice any significant improvements in its ability. I believe that it would have improved if I had allowed it to continue running for many days. However, I do not believe that the algorithm actually learned very much in such a small number of games, which is why I did not notice a major difference. What it did learn was how to beat older versions of itself that had no experience looking a full turn ahead. This is why I believe the new player was able to defeat the old player so handily.

Overall, neural networks were the most successful traditional artificial Tarble player that I implemented. The player also taught me a great deal about how to think about artificial Tarble players, and it forced me to employ new techniques in order to successfully play at the highest level possible. However, even the most successful neural network player was not playing at the level that a brand new human player would be able to play. At this point it seemed obvious that a new method of implementation would be required if I wanted an artificial player to have the ability that I desired. Therefore, I decided that the player would imitate human behavior and try to play the game like a human would play the game.

5 Human Evaluations

5.1 Notation

Before implementing a player that utilized human strategy, I had to identify what human strategy entailed. I ran human tests in order to find what types of strategies and tendencies that people used while playing. The first step of doing this evaluation required creating a notation to record the games. I wanted a notation system that would be easy to understand, able to fully recreate a game, and be as undisruptive to the natural flow of the game as possible. I used the notation system of chess as an example of what I wanted to accomplish.

The first issue that I faced was the need for a coordinate system. The chess notation system uses the simplicity of the grid coordinate system to allow users to easily

refer to the position that they would like to move their pieces. However, Tarble is not played on a square grid like chess. Tarble is played on a hexagonal grid, which does not have a universally accepted coordinate system like the cartesian coordinate system. There are multiple coordinate systems used for hexagonal grids. Polar coordinates (each location is referred to using an angle from vertical and a distance from the center), tri coordinates (where each location is referred to by 3 numbers rather than 2), and a skewed cartesian coordinate system (where locations are referred to similar to the grid cartesian system but with the x and y axis meeting at a non-right angle).

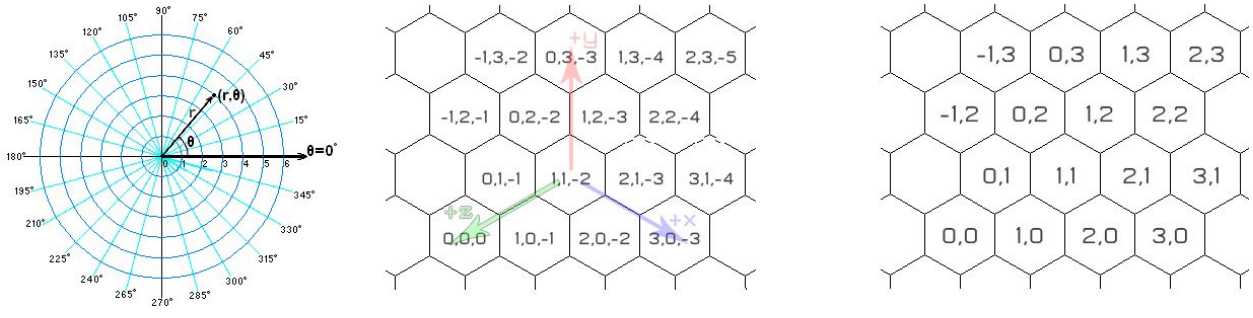

Figure 13: Examples of the 3 coordinate systems discussed above. Images of the polar (left), tri (center), and skewed (right) are shown to illustrate how each is represented. Note that the polar coordinate system is drawn without a hex-grid. A hex-grid can be Imposed over the image by matching origins in order to obtain the hex-grid coordinates.

I chose to use the skewed cartesian coordinate system for my notation because I thought that people would have experience working with grid cartesian coordinates that could be translated to this system. Polar coordinates tend to be difficult to grasp for

people who have never worked with them, and I did not want players worrying about calculating coordinates while playing. This unfamiliarity could lead to more time spent notating the games rather than playing, and calculating the coordinates could distract the players from the actual game, leading to suboptimal play. Furthermore, people understand bi-coordinate systems. While the tri-coordinate system makes sense for hex-grids, many players may not easily grasp the purpose of the third coordinate immediately. This could lead to possible mistakes in notation, and the third coordinate is also more writing for the person notating the game.

After deciding on a coordinate system, I had to find a minimalistic way to notate a single move of the game. I determined that there were 3 things needed to understanding a turn of the game. I needed:

- The roll
- The pieces moved
- The pieces captured.

The path that a piece took during the turn should be irrelevant. Notably, the state of the game can be fully captured without the roll, but if a player chooses to make a move with a roll of 6 that would have been legal with a roll of 4 this is an interesting observation that I wanted to know about. Therefore, on every turn of the game the notation includes the roll made, a list of location pairs where the first element of the pair refers to the location of the piece being moved and the second element of the pair refers to where that piece was moved, and a list of locations referring to pieces that were captured on the turn. It does not matter which piece executed the capture because if the turn ends in

the same state it should not matter which piece did the capturing. An example of a turn would be: 3 | [(0,1),(0,-1)], [(2,1),(3,2)] | (0,0). On this move the roll was a 3, two pieces were moved, and the piece at location (0,0) was captured. This notation system captures all of the relevant actions taken during the turn with as little writing as possible.

5.2 Graphic User Interface

While I spent a great deal of time trying to make the notation system as simple as possible, it was still more complicated than I would have liked. I did run 16 playtest games using a physical board and the notation system described above. However, the recording proved to take about as long as the turns, even when I was the recorder. I often slowed the game down by asking the players to wait for me to finish recording a turn, and I both understood and was familiar with the notation system. When I asked other players to record the games for me, they would often spend more than twice as much time recording the games as they would playing them, which was exactly the issue that I was trying to avoid. Further, I witnessed multiple errors in notation made while observing games that other people were recording. All of these issues convinced me that a graphic user interface, or GUI, would need to be implemented if I wanted any significant amount of people to play recorded games of Tarble.

I implemented a Java applet that could be used to play games of Tarble between 2 people or a person and an algorithm. The applet used the recording system that I had

developed, and recorded each game without the players even knowing that the game was being recorded. When the game was completed, the algorithm would then upload the recorded game to a database that I could access when I wished to review the games.

The applet provided many advantages over the previous notation system.

- I could play against computer algorithms much more quickly. Previously, the algorithms would output a move in the notation system provided above, I would make the move for them on a physical board, and I would enter my move in the notation system to the algorithm. Games that would normally take 15 minutes could take an hour using this system.
- Players could play recorded games without having to worry about learning a notation system and recording the game.
- It was impossible to make notation errors
- Players could no longer accidentally make illegal moves.
- I could quickly replay games played by any of the players. This was the biggest advantage of the new system. This ability allowed me to observe the same games over and over again searching for patterns in the movements.

Without the applet, it is doubtful that I would have been able to observe nearly as many patterns as I wound up observing. While I ran 16 play testing games with the hand-written notation system, there are 146 digitally recorded game stored in the database and even more games were played as there were issues early in the system and some games were lost.

5.3 Experimental Design

I designed my experiment to take place in two different phases. The first was a training phase in which I familiarized my subjects with the game of Tarble by having them play games against other participants in the study. The second phase consisted of presenting the subjects with a series of tactical situations that could arise in a game of Tarble. The games were originally played on paper, but after the implementation of the Java GUI, the games were played exclusively digitally. The situations were all tested digitally.

I started off by familiarizing my subjects with the rules of Tarble. In order to standardize my results, none of the subjects had exposure to the game prior to the experiment. After explaining the rules each of the subjects played games against other subjects. These games were recorded using the Java application to be analyzed later. Although the main reason for playing these two games was to familiarize the subjects with Tarble, these games also had some very interesting results in their own right. The games told me more about the general play styles of my subjects and how my subjects developed their skills and strategies as they played.

After the two games I had my subjects play against each other, I gave each subject six situations and recorded their moves. These situations took the form of a single ply (or turn), which generally consisted of multiple movements. The following is a short description of each situation:

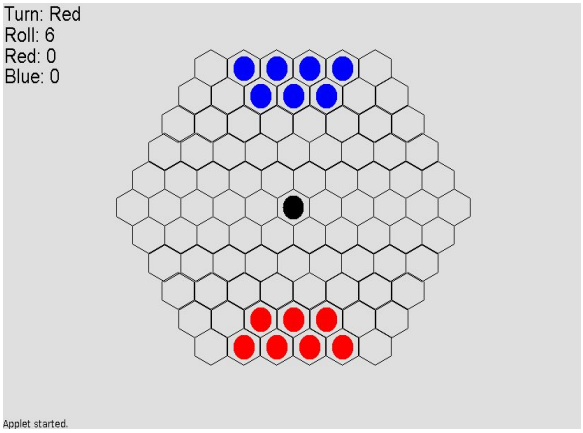

Figure 14: Situation 1. This is the starting configuration for Tarble with a roll of six. I wanted to see what the initial strategies were for each of the subjects with the maximum possible roll.

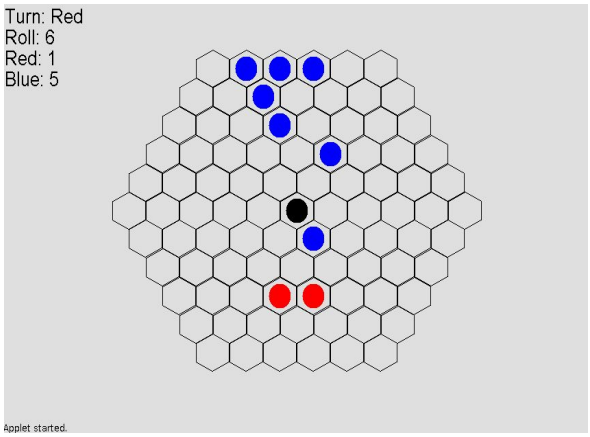

Figure 15: Situation 2. The player is significantly down in score in this situation, but has the material to make up some ground. This situation is illuminating as to how people play differently from a losing situation.

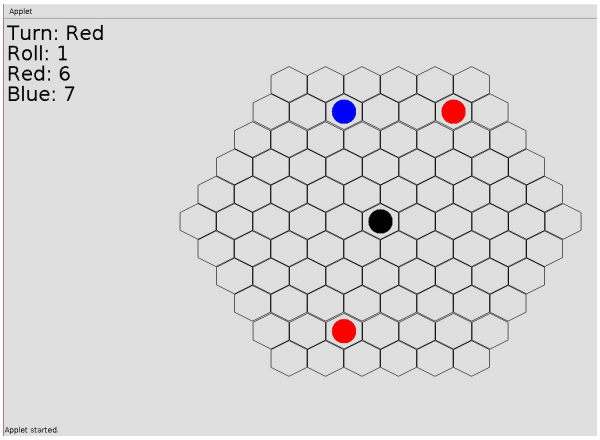

Figure 16: Situation 3. The player is also losing in this situation, but not by as much. However, the opponent has an opportunity to win next turn with a roll of 2 or more by sacrificing his piece into the corner.

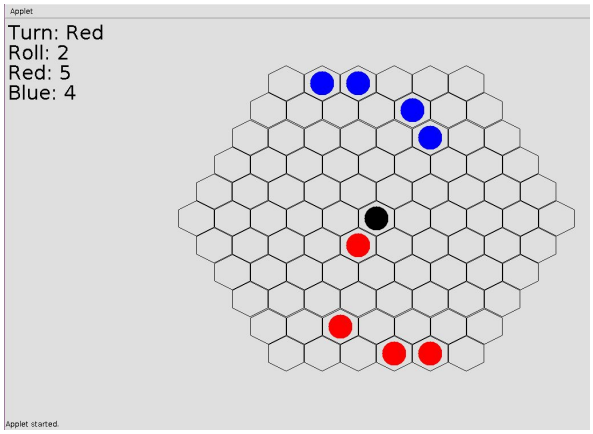

Figure 17: Situation 4. The player is winning here and has the opportunity to start sacrificing his/ her pieces to win the game. I was interested if the subjects would pursue this strategy when the margin of the game was so close.

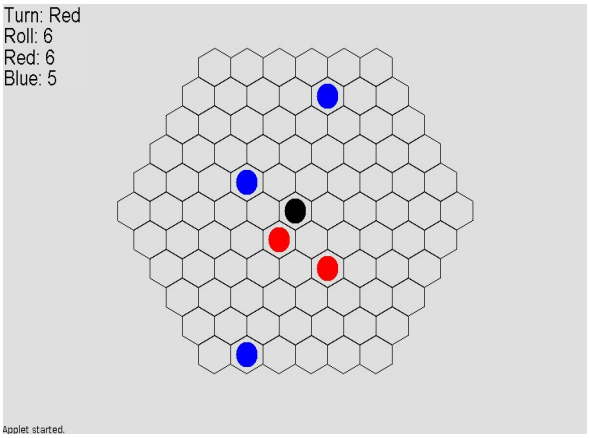

Figure 18: Situation 5. The player here has the opportunity to take up to two pieces, but must choose which two.

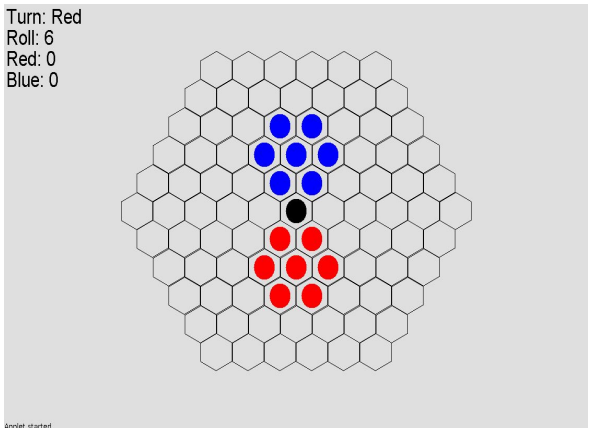

Figure 19: Situation 6. This is a stalemate situation where neither player can take his/ her opponent's pieces, only the middle piece. I was curious to see how the subjects would react to not having an opponent's piece to take or much opportunity to gain a better position.

5.4 Human Testing Results Part I: Full Games

After gathering the data from my subjects, I analyzed it by replaying each situation and game that I observed. I viewed all of the games through the lens of trying to figure out what the players were looking to achieve in their moves.

The first thing that I did was evaluate the games that my subjects played. There were some very interesting situations that occurred organically in the games without any interference on my part. For instance, one of the first things that I noticed was that players seemed to assume that both they and their opponents would roll a 3 on all future rolls. I do not think that it was a conscious decision, but time and time again players would positioning themselves 3 spots away from the center piece, allowing them to capture it on their next turn with a roll of 3 or greater, and 4 spots away from the opposing pieces, giving them a safe distance in order to avoid being captured. I refer to

this phenomenon as the rule of 3 because of player's seemingly unconscious decision to play as if future rolls will be 3s. A perfect example of this behavior can be seen in figure 20 below. Figure 20 shows a move made by one of my subjects where there was no clear reason to retreat back to exactly 4 spaces away. The increase in expected safety from moving from 3 to 4 spaces away is exactly the same as the increase in expected safety from moving from 4 to 5 spaces away. In both cases the chance of being captured decreases by 16.7% after the move. However, the subject deemed that after 4 spaces, her pieces were safe enough to develop her positioning elsewhere. Situations similar to this arose constantly throughout testing.

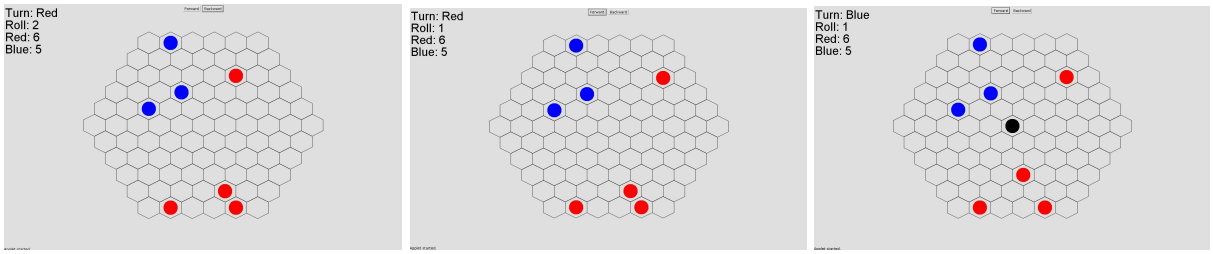

Figure 20: The progression of one turn. The board started in the position on the right, and ended in the position on the left. Red has rolled a 2 and chosen to retreat with her first move and develop board presence with her second. Note that the retreat brings her 4 spaces away from her opponent and the development brings her 3 spaces away from the center thus illustrating the rule of 3.

Another common trend that I noticed was that people looked at the number of spaces away a piece was from an opponent's piece in determining whether they were at a safe distance rather than actually counting the roll required for an opponent's piece to

capture them. I will refer to this distance measured in spaces as board distance and the distance measured by the required roll to capture as jumping distance. An example of the difference between the two can be seen in figure 21 below:

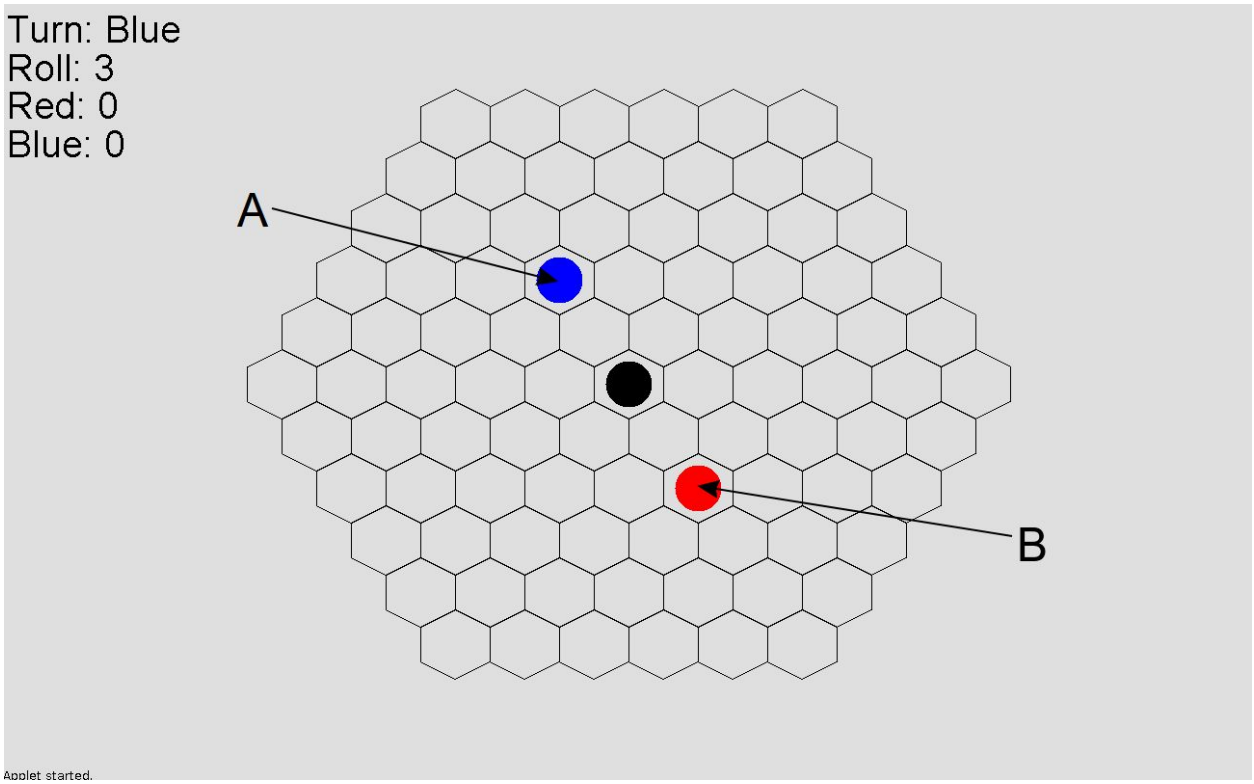

Figure 21: The board distance between the blue piece labeled A and the red piece labeled B is 4 because it would take 4 moves on an empty board for piece A to reach the location of piece B. However, the jumping distance is 3 because piece A can capture piece B in just 3 moves.

Once I notice the rule of 3, described above, I began specifically noting how close a player allowed their pieces to be to an opponent's piece at the end of the player's turn. I found that while 4 was the most common distance, 55% of the time that a player had the choice to end with all of their pieces at least 4 moves away from their

opponent's pieces they chose to do so, 3 was also abnormally common. At first I believed that players were content with 3 but 4 was ideal. However, after more careful analysis, I realized that most of the places where people ended 3 moves away were when a jump from the opponent would be required to reach the current player's piece with a roll of 3. An example of this can be seen in figure 22. I took this to mean that people were not actually counting the moves required to capture their pieces, but they were instead looking at the board and visually determining if their pieces were a "safe" distance from their opponent's pieces. This visual measure of distance is what I call board distance. Board distance appeared to be far more important than game distance when the players were considering positioning.

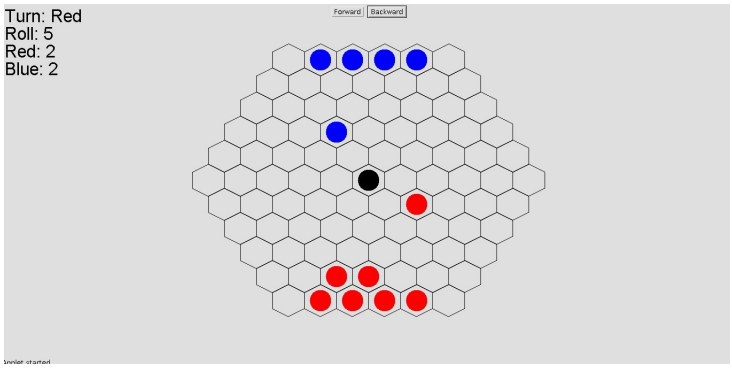

Figure 22: Blue has just ended her turn. Any roll of 3 or greater from Red will allow Red to capture Blue's piece. However, Red would have to jump the center piece in order to capture with just 3 moves.

Finally, I noticed that after a player has made their first move of the turn, they are much more likely to move the same piece on subsequent moves. There is nothing in the rules that benefits a player from moving the same piece twice in a row. However, in

88% of the moves of roll 2 or greater that I observed, the player moved the same piece with both their first and second second moves.

5.5 Human Testing Results Part II: Situations

I also used the six situations described in the experimental design section to further study my subjects and observe their reactions to interesting situations. However, after gathering the results, I observed that some of the situations had much more interesting results than others. I will limit the discussion to these situations: specifically situations 1, 3, and 5.

The first situation that I studied was Situation 1, the starting state of the game with a roll of 6. I considered the first move of the game to be very important because it is the only state that is visited in every game. I used a roll of 6 as I considered it to be the most interesting first move scenario due to the player having the maximum number of options available to them. Almost all of the subjects (> 90%) took the middle piece and then retreated back to safety. One such example is the situation shown below:

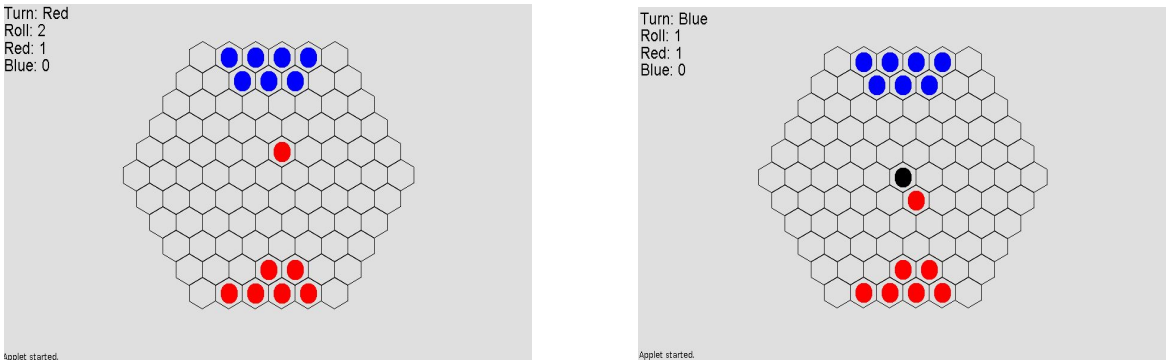

Figure 23: From the starting position red captures the middle piece (left) and then retreats back two spaces (right)

All of the players who employed the capture and retreat strategy only moved a single piece on this first turn despite having the ability to move up to 6 different pieces if they desired. However, greater than 90% of the subjects only moved a single piece. This result is consistent with the observation that players are more likely to move a piece that they have just moved as discussed above. After capturing the middle piece the subjects may have not seen a clear move going forward, there being no other pieces to take. Instead of developing their other pieces to the middle of the board the subjects kept moving the same piece. Earlier I only analyzed whether a player would move the same piece on their first two moves of the turn. However, this evidence extends the idea to incorporate all the moves of a turn, especially the later moves.

Another interesting behavior that I observed in this situation was that *every* person who took the middle piece moved to the right afterwards. The board is symmetric, so there is no logical reason to favour one side over the other. However, every single one of the subjects that captured the center piece and then retreated, retreated to the right.

The next situation that I studied was Situation 3, an end game state where the opponent has a good chance to win on the next turn by sacrificing his remaining piece. I found this situation interesting because there is an optimal move in this situation, moving toward the opponent's remaining piece as shown:

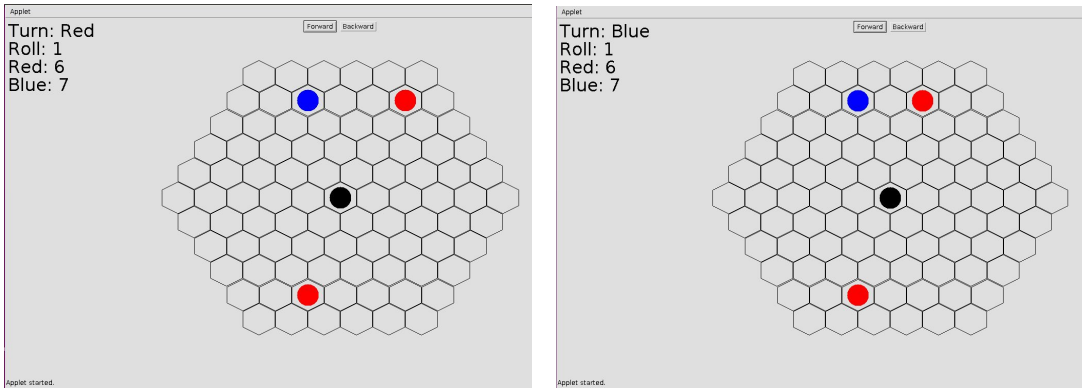

Figure 24: the starting position for situation 3 (left) with the optimal move (right)

This move has a 1/9 chance of winning, given ideal play by the blue player. If blue rolls a 1, he/she can not win on the next turn. Red then has a chance to win by rolling a 3 or higher and taking blue's single remaining piece. Any other move by Red in this situation would still lose on the next turn if Blue rolls a 2 or higher. However, if Blue rolls a 1, Red would have to roll a 4 or greater on his/ her next turn to guarantee a win with any other move.

However, none of the participants employed this strategy. The most common move was instead to move the other red piece towards the center of the board. This move got the red player within 3 spaces of the middle piece, a result found to be preferable to players in the full game observations. I believe that the reason for this behavior is that the optimal move looks dangerous. It is rare that a player would ever want to end the turn with his/ her piece just 2 spaces away from an opponent's piece. It

would be easy for the blue player to capture the red piece if the red player moves towards the blue player. Therefore, the move is glossed over even though it is the mathematically correct move.

Finally, I found situation 5 interesting as well. This situation provided the subjects with up to 2 different pieces to capture. The subject had to find which 2 pieces they wanted to capture. They can capture the middle piece and any of the blue pieces, or they can capture both blue pieces on the top half of the board. I considered capturing two blue pieces to be the optimal play, as shown below:

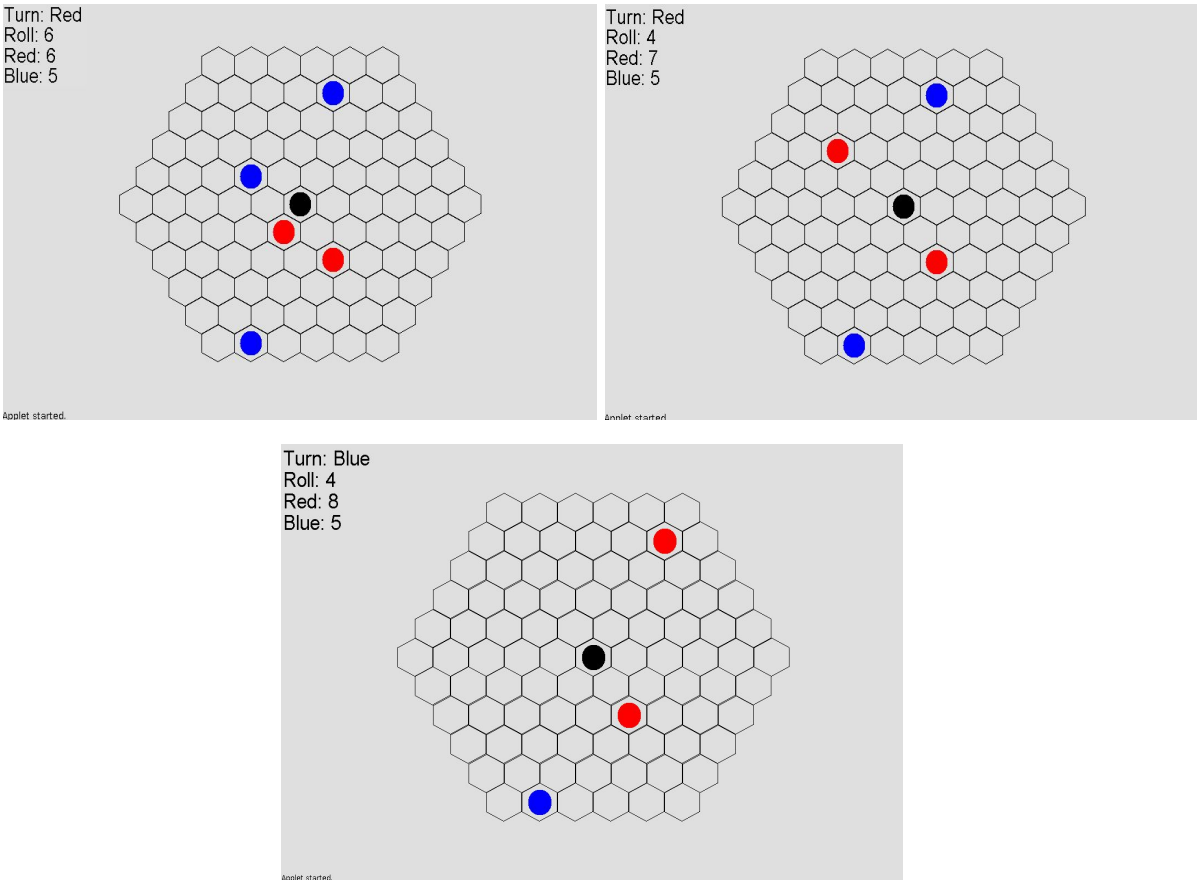

Figure 25: Starting position for Situation 5 (top left), taking the first blue piece (top right), followed by taking another blue piece (bottom).

However, while I believed the above sequence to be the clear best solution, only one of my participants chose this sequence. Almost all of them captured the middle piece and one of the blue pieces (there was one who only captured the middle piece and retreated, but I consider this to be an outlier because he was the only subject to make this move). I theorize that the reason for this behavior was that none of the subjects had enough experience with the game to see this complex of a move. Capturing 2 blue pieces requires a player to switch directions mid move, which requires extensive experience with a hex-grid in order to see quickly. It also requires the player to ignore the neutral piece, which lies right next to one of their pieces. It is rarely the case that capturing the neutral piece is incorrect. Therefore, most players will see this option immediately and continue looking for a move with the assumption that capturing the center piece must be their first action.

The subjects moves did follow the tendency to stay exactly 4 spaces away from their opponent with alarming accuracy. An example is shown below:

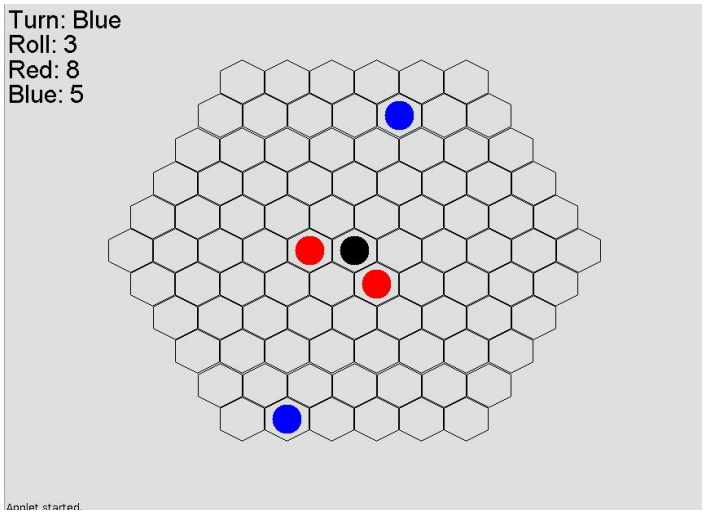

Figure 26: The move performed by one of the subjects. Note that both of the red player's pieces are exactly 4 moves away from the blue pieces.

Although this tendency was overwhelmingly corroborated, this situation was different than the previous situations and the games in that players consistently moved more than one different piece. After capturing one piece with one of their pieces, the subjects often went on to capture the other piece with their second piece, or in some situations move their second piece further from the opposition.

5.6 Discussion

There were 3 major observations that I made about the play styles of my testing subjects while conducting this study:

- Players seem to assume that a 3 will be rolled on all future rolls.
- Players seem to evaluate distances visually rather than from the perspective of the game.
- Players tend to move the same piece multiple times in a row whether it is strategically correct or not.

There were also 2 other interesting observations that I made, which may not have as much strategic relevance but still deserve mentioning:

- Players to move to the right rather the left in situation 1.
- Under certain situations newer players seem unable to notice seemingly superior moves.

The first observation that I made was that when players set up positions that they preferred at the end of a turn, they seemed to make their decisions assuming that a 3 would be rolled on the next turn. 4 spaces away appeared to be a safe distance to avoid being captured, and 3 spaces appeared to be an acceptable distance from the center piece to allow players to earn the free point for capturing the center piece on their next turn. It makes sense that there is a distance at which players should feel safe or unsafe, but why is 3 specifically the distance that was chosen by so many players?

This problem is explained fairly well in the context of Gott's rule and one and done decision making. These theories basically say that there is a known prior on the way that a value will be chosen, in this case it is reasonable to assume a uniform distribution for the roll of a die, and the subject would like to predict what the value will be (Vul, Goodman, Griffiths, and Tenenbaum 2014). Humans tend to predict that the value will be the expected value of the posterior distribution, 3.5 in the case of a fair die. Obviously a roll of 3.5 is impossible, but if the player assumes that the next roll will be greater than a 3 and less than a 4, the observed behavior aligns perfectly with the predictions. Players will assume a distance of 4 from the opponent is safe because that is greater than the expected value, and players will assume that a distance of 3 from the center will allow them to capture it on the next turn because 3 is less than the expected value.

Another very interesting observation is that players tend to cue into board distance rather than jumping distance when applying the distancing strategies discussed above. This strategy is obviously not ideal if a player would actually like to be

safe given that their opponent rolls a 3. However, there were a couple of interesting theories that I have which could explain this behavior. None of the players that I studied have had previous experience with Tarble. The most games of Tarble that any of the players have played at the end of the study is 2. Therefore, players may have noticed that pieces that look a certain distance away from another piece tend to not get taken, which could be a very quick observation after just a couple of examples. However, the players have not had the experience to realize the suspicious coincidence that the times when their pieces were taken at that distance were when the center piece was between their piece and their opponent's piece. It would be interesting to see if more experienced players act in a similar manner or notice the flaw in this reasoning.

Another possible explanation for this behavior is the idea of a visual heuristic. The branching factor of the game, the number of moves possible from any given state, is absolutely massive. It is so large in certain cases that even a computer can not hold all of the possible moves in memory. Therefore, it would be infeasible for a player to look at all of the possible next moves that their opponent could make in order to form a reasonable heuristic. Therefore, more primitive methods must be used. One such method that I believe players use all of the time is the visual heuristic. Players basically look at the board and decide if a given board state "looks good enough." This is obviously a very loosely defined idea. However, players can definitely tell when they believe one board state is better than another by just looking at it. There is no complex evaluation that needs to be done. One such thing that could make a board state look better to most people is having pieces that are more than 4 spaces away from your

opponents and less than 3 spaces from the center. If this is the method that is used to determine moves rather than a look ahead method, it would make sense that players would value board distance more than they would value actual game distance.

Furthermore, players tended to move the same piece continuously throughout a turn rather than moving multiple pieces. Strategically this allows players to be more aggressive in the early game but at a cost to later game development. At more advanced levels of play, not considering moving multiple pieces can be massively detrimental. However, it could be a valid method of thinning the search tree in order to make the space of possible moves more comprehensible. Players could be exhibiting this behavior not for strategic benefit, as it is almost surely a less than ideal strategy, but more due to a “cognitive blinders” effect. Each turn provides the player with so many different choices. Even the computer had trouble keeping track of all of the possible moves and angles of attack involved in a single turn. Therefore simplifying measures need to be taken. One such method that people employed is to choose a piece every turn, and move that piece and only that piece in whatever pattern the player saw fit. The choice of which piece to move is difficult. Certain rules like the closest to an enemy piece or the least developed piece may apply, but once the piece is chosen, the turn becomes much simpler to handle. If a player assumes that they can only move a single piece, the number of choices decreases dramatically. A new player can now think through all of the choices that they have and choose the “best” one because they have reduced the problem to one that they can cognitively handle.

I believe that people utilize similar cognitive shortcuts in everyday life all the time as well. If a person flips a coin 30 times and it comes up heads every time, the person will assume that the coin is somehow weighted. Most people can come up with this explanation rather quickly. However, there are many explanations for why these flips could have occurred. The person could have just been lucky (or unlucky as the case may be), gravity could have changed in that one area, the person might be flipping the coin so that it will always land on the side that it was flipped from. However, the person has reduced the possibility space of what could have happened to just something being wrong with the coin because the coin being weird is the most obvious explanation. Once the possibilities have been reduced to just the coin being different, deciding that the coin is weighted is a rather simple conclusion. This example is obviously a toy example, but there are plenty of things in the world that have too many possible explanations to list. However, people make intuitions about these situations easily by reducing the number of things being considered, and the strategy involved in Tarble is not different.

Finally, there were a few other less clear observations that I observed in this study as well. For instance, I found in situation 1 that players tended to move to the right when given a choice between right and left despite no discernible advantage to moving right instead of left. I also found in situation 5 that players tended to not see a move that I thought clearly ideal. I do not have any definite explanation for why these phenomena would occur, but there are a few hypothesis that I would like to make.

Firstly, why would people move to the right rather than the left? The most obvious explanation is that I just happened to observe players who made this move.

However, if it were truly random the probability of all of the players making the same 50/50 move would be very low. I collected enough data that I can be pretty confident that it was not random chance that lead to all of the players retreating to the right in situation 1. One possible explanation is that everyone that I observed was a native English speaker. English reads from left to right. Therefore, the players might naturally move things in their mind from left to right, which leads to moving from left to right in the game. Native Hebrew and Arabic speakers would need to be observed playing in order to make any conclusive results, but this is one possible explanation. Another possible explanation is that the people playing were right handed. I do not know if the players were in fact right handed because it did not occur to me to ask them at the time of the study, but if right handed players are more likely to move to the right and left handed players are more likely to move to the left, this would at least weight the odds in the favor of moving to the right, which could explain the behavior that I observed. Future studies would need to be conducted in order to bring any validity to either of these explanations, and the behavior is strategically neutral enough that it is beyond the scope of this paper.

In addition, I found that players tended to be unable to find the ideal move in situation 5, despite it not requiring looking ahead to the opponent's turn or moving different pieces. However, it does require players to navigate a hex-grid effectively. I believe that this behavior can be marked up to people simply not understanding how to properly navigate a hex-grid without experience and an inability to look at all of the possible moves that a player can make even with just one piece on a roll of 6. The

optimal move would also require ignoring the center piece, which for most players is simply too juicy to pass up. For these reasons I believe that players did not even consider the option of capturing both red pieces in situation 5.

6 Implementation

After completing the human studies, I was finally ready to begin implementing an artificial Tarble player that played the game like a human. However, there were still a few issues that I had to work out before I could begin implementing a final player. The most obvious issue, which I had overlooked up until this point, was that different people play differently. Even amongst new players, there are people with different prior experiences with strategy games, different ages, personalities, and even cultures or handedness, which could all affect how different people would play. All of that does not even begin to examine the differences involved with differing levels of experience with Tarble. More experienced players obviously play at a higher level than less experienced players, but I also observed various successful strategies emerge and local metagames within different playgroups. There were 3 playgroups of 4, 6, and 6 people with whom I tested at least 10 games per player, and each of these playgroups seemed to develop differing strategies based upon what seemed effective early on in testing.

All of these possible differences led to my decision to implement 3 different human-like artificial Tarble players. The first would be what I called my “Joey” player. Joey, my 9 year old cousin, was the youngest and least experienced of all of the players that I observed. This player would represent my inexperienced subjects, and

implementing the Joey player should be simpler than the rest. This would give me a good fallback if the others proved too difficult to implement. The second player would be called my “Sarah” player. Sarah, a senior at Suffolk University and moderate strategy game player, was the top player in one of play groups that I observed. I have also observed her playing 23 games against both human and artificial opponents, more than any other person. These games along with multiple interviews about her thoughts and strategies led me to choose her as one of the players to emulate. Finally, the last player that I will implement will be called “Brian.” Brian is a graduate student at the Massachusetts Institute of Technology with copious amounts of strategy game experience. Brian was a competitive chess player as a child, and has maintained a healthy interest in various strategy games. I have recorded Brian playing 11 games of Tarble and observed him playing many more. I have also had multiple extended talks about Tarble strategy with Brian. He is easily the best Tarble player of the people that I have studied. Brian himself also played a significant role in identifying his own strategies and heuristics, which will be discussed later. With these three players, I hoped to create a varied artificial player base that would be able to capture the variation among different players and hopefully combine them in some way to create a human-like artificial Tarble player.

6.1 Joey

The first player that I implemented was the “Joey” player. I recorded the real Joey playing 3 games, and while he definitely improved across the 3 games, his general

strategy was basically blind aggression with little thought to how the opponent might respond. Joey's strategy essentially boiled down to maximize the number of points that he could score and hope that it was more than the opponent had when the game ended. It is unclear whether Joey relied on board distance or game distance when making his decisions because the only time that he would actively retreat is when he is next to an opposing piece, but he is unable to capture the piece. Joey also interestingly seemed to ignore the fact that a player only needs 1 piece near the center to capture the center piece. His development involved moving the closest piece to the center towards the center once all capturable pieces had been taken.

While Joey definitely had the most simplistic strategy of the players that I observed, there still existed no easy way to imitate his strategy. I could tell a human to retreat if they are next to an opposing piece and move towards the center, but getting the computer to understand these abstract concepts is significantly more difficult. As such, I needed to create a library of heuristics that could be used to capture what these things meant. I could then use these heuristics to computationally capture the meaning of moving towards, running away, attacking, and defending.

The first heuristic that I implemented was board distance. While it was unclear whether Joey was using board distance, it would both give me the information that I needed to implement his strategy and be useful for future players. Using the skewed euclidian coordinate system discussed above, board distance is actually a very simple calculation. The board distance between any 2 points on the board is simply equal to the maximum of the distance in the x direction and the distance in the y direction. For

instance, the board distance between the center location, (0,0), and the location (2,3) would just be max(2-0, 3-0)=3 units. Having a closed form equation to measure board distance allows me to find the board distance between 2 locations on the board in O(1) time, which was indispensable for the implementation of the human-like artificial players.

Once board distance was implemented, finding the closest piece to the center was simple. The program iterates through a list the locations of each of its pieces (only a maximum of 7 pieces), and chooses the piece visually closest to the center location, (0,0). I was also able to utilize the same strategy for defining attack and retreat strategies. Attacking involved finding the closest pair of pieces between the player and its opponent, and moving towards the closest piece. Defending involved finding the same pair, but moving away. Moving towards or away was simple because once the piece to be moved was identified, there are only a maximum of 6 possible moves that any given piece can make. By finding the board distance between the piece's new location and the piece of interest, the program can minimize or maximize this quantity respectively in order to attack or retreat. Because board distance is a constant time calculation, this entire process can be done in quadratic time, which is very manageable due to the limited size of the board. This is a massive improvement to the exponential time and complexity with which the previous artificial players dealt.

After the simple heuristic and strategy functions were implemented, implementing the actual Joey player could not have been simpler. All that I had to do was use the heuristics in order to find out when it was appropriate to utilize a specific strategy. The

final Joey player captures any piece that it can as soon as it can. It then checks if any of its pieces lie on the locations that will lead to them being destroyed at the end of the turn. If any pieces are, it moves off of the location and towards the center. The only exception to this rule is if the piece would then end the turn next to an opposing piece or if the piece was already in the center location in which case it moves as far away from opposing pieces as possible. Once these edge cases are taken into account, the player finds the nearest opposing piece and determines whether it can capture the piece this turn. If it can capture the piece, it attacks that piece. If it can not, it attacks the center with all pieces not already next to the center location. Finally, if it would end the turn with a piece next to an opposing piece, it retreats. Once the heuristics and high level strategies are in place, the player is actually remarkably simple. The description of the player actually reads out very similarly to how I would describe Joey's play to another human that I wanted to imitate it. This makes the player easy to understand and update.

After implementing the Joey player, I tested it against the Monte Carlo tree search player, both neural network players, a human who was new to Tarble, and myself. The results are shown below:

Opponent	Games in Match	Games Won	Game Win Percentage
Monte Carlo	13	13	100%
Neural Network Single Step	13	12	92.3%
Neural Network Full Turn	13	10	76.9%
New Human	5	3	60%
Me	5	1	20%

The match against the new human player marked the first 5 game Tarble match ever won by a computer. The Joey player also decided moves in under a second, far more quickly than any of the other artificial players that I had implemented thus far. The first human-like artificial player was a huge success, and I had only begun to explore what could be done with powerful high level heuristics and strategies.

6.2 Sarah

The second human-like artificial player that I implemented was the “Sarah” player. Sarah the person played the single most games in my study and discussed with me the strategies that she used while playing these games at length. Sarah interestingly was the player that I was observing when I discovered the rule of 3. While it is a tendency that was observed in a majority of players, nobody followed it quite as explicitly as Sarah. Sarah also approached the game from a very interesting angle. She was the most conservative player that I observed in the study by a great deal. Sarah would actively refuse what appeared to be obvious captures in order to end the turn at least 4 spaces from the opponent. Sarah also notably appeared to use board distance almost exclusively in her decision making. However, despite many of these seeming inefficiencies in her play, Sarah still managed to be among the highest level of players involved in the study. The conservative strategy was also found to be rather common among female players, especially those who considered themselves

non-confrontational. Therefore, I felt that Sarah embodied a group of players that deserved to be represented.

The key difference between the “Sarah” player and the “Joey” player is that the “Sarah” player looked at the turns of the game wholistically. One common pitfall that the “Joey” player fell into was that the player would often spend its turn attacking, and then on its last move of the turn, the only move where it thinks about defensive play, the player’s pieces were already so far out of position that it did not matter that the last move was played defensively. The “Joey” player’s game would subsequently end very quickly, and the player would often make unfavorable trades. The “Sarah” player on the other hand, did not just look at its moves individually, but it actually finds the positions that it will end up in given certain move. This player had to look at concepts such as favorable and unfavorable trades, 2 for 1 trades, and safe distance. The “Sarah” player was also the first time that piece sacrificing was implemented. When player first begin playing Tarble, they often decide from the outset that losing pieces to the corner and center board locations must always be a negative. However, these players often learn that there are situations where ending the game as quickly as possible is preferred. In these situations, sacrificing pieces could be a legitimate strategy. The “”Joey” player will never sacrifice a piece unless it stems directly from a capture. However, if the “Sarah” player is winning by a large enough margin, it will actively sacrifice pieces.

The first heuristic that I needed to implement for the “Sarah” player was a full distance matrix heuristic. This heuristic stemmed nicely from the distance metrics that I defined in the previous section. However, instead of simply finding the distance between

2 pieces, I found the distance between every pair of pieces on the board. The reason I needed all of these distances was that the “Sarah” player needed to not only know if it could capture a piece, but it also needed to know how far it would be from the opponent after the capture. The “Sarah” player would only make a capture if:

- The player could retreat to the “safe” distance of 4 spaces away from an enemy
- The player could capture multiple pieces while only risking a number of its own less than it captured (risking being defined as can be captured with a roll of 3).
- The player is risking an equal number of pieces to the number it can capture, and it is winning by 2 or more points prior to all captures.

All of the risk evaluations were done using board distance. The “Sarah” player captures much less recklessly than the “Joey” player, and all captures must be made only after evaluating the possible responses of the opponent (with a roll of 3).

The other major algorithm that I had to develop fro the “Sarah” player was the concept of development. The “Joey” player did not concentrate on positioning enough to care about anything other than points and capturing. The “Sarah” player was concerned with the board position. Therefore, when it has captures that meet the above criteria and its pieces are safe enough that retreating is not required, the “Sarah” player develops its board position. Development however, was not such an obvious concept to develop. Capturing and retreating are fairly straightforward, but development is a less well defined concept. From watching and playing the game, I knew that a well developed board involves having pieces close to the center that are well protected and a safe distance from the opponent. Therefore, I decided that development would involve 3

pieces, a primary, secondary, and tertiary piece. Each of these pieces have a goal location, and the score of the position is a function of their distances to their goals. The primary goal is the location adjacent to the center piece that is furthest away from opposing pieces. The primary piece is defined as the piece closest to the primary goal. The secondary and tertiary goal locations are defined as the locations adjacent to the goal location, but neither adjacent nor opposite to the center location. A figure depicting the goal locations is shown below:

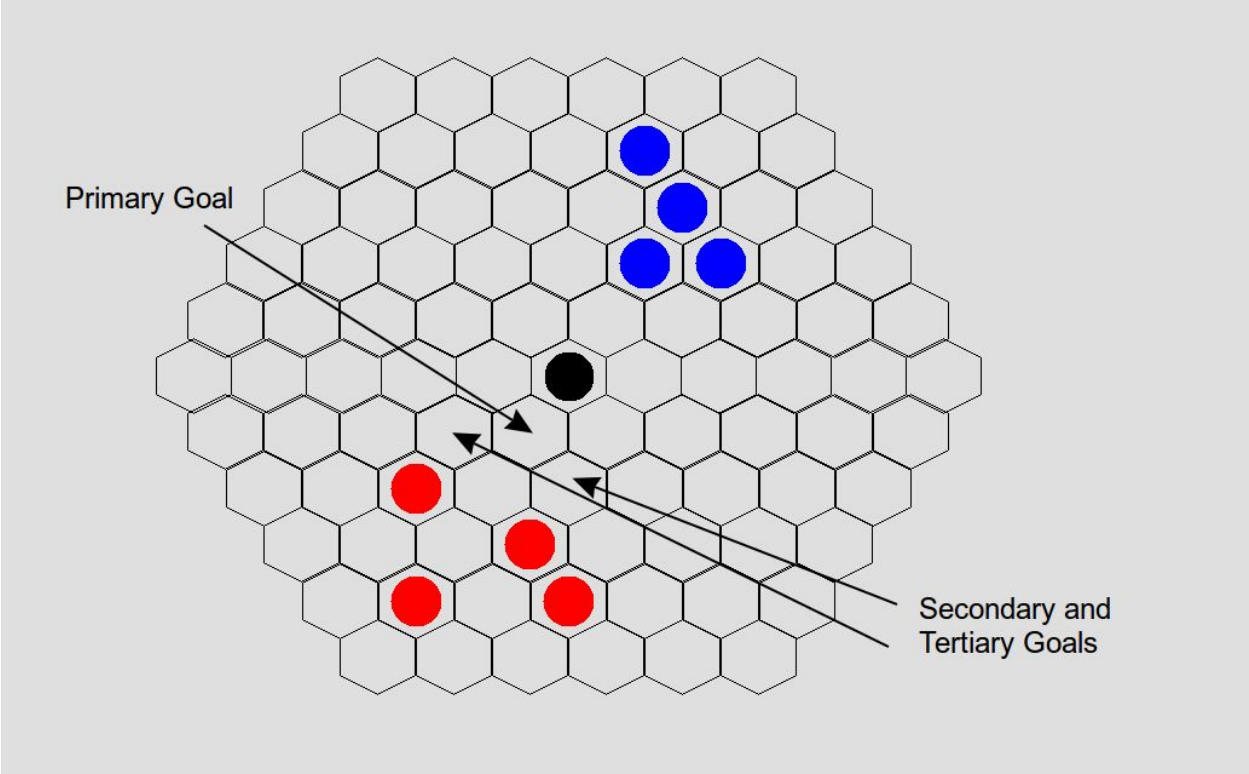

Figure 27: A figure depicting the primary, secondary, and tertiary goal locations for the red team in the current situation. Note that these locations are subject to change depending on where the blue pieces are located.

The secondary and tertiary goals are differentiated by finding the location closer to one of the player's pieces (other than the primary piece). The location with a closer closest piece is the secondary goal, and the location with the further closest piece is the tertiary goal. Finally, the secondary piece is the piece, other than the primary piece, closest to the secondary goal, and the tertiary piece is the piece, other than the primary and secondary pieces, closest to the tertiary goal.

Once all of the goals and locations have been found, I find the distance between each piece and its respective goal. The score of the position is then defined as 3 times the primary distance squared plus 4 times the secondary distance plus 2 times the tertiary distance. The coefficients represent how important the piece is towards the goal of ideal development. The primary goal is squared because it is the most important of the goals, but the importance does not scale linearly. For instance, the difference between being 3 and 4 spaces from the center is greater than the difference between being 1 or 2 spaces from the center because in the latter case the player is "close enough," again defined by the rule of 3. Finally, development moves are determined by generating all of the possible moves 1 roll away and choosing the move that yields the smallest score function. The only complication is that it filters out moves that put a piece at risk (less than 4 spaces from an opposing piece). Using this strategy the artificial players were able to develop in a reasonable manner that greatly resembled the development strategies of many of my subjects. The "Sarah" player develops whenever none of its pieces are at risk and the attacking criteria listed above is not met.

After implementing the “Sarah” player, similarly to the “Joey” player, I tested it against a number of different artificial and human players. The results are shown below:

Opponent	Games in Match	Games Won	Game Win Percentage
Monte Carlo	13	13	100%
Neural Network Single Step	13	13	100%
Neural Network Full Turn	13	13	100%
Joey	13	11	84.6%
New Human	5	4	80%
Me	5	1	20%

The “Sarah” player won the same number of games in our 5 game match as the “Joey” player. However, I felt that its performance in those games showed a significant improvement over the performance of the “Joey” player. In addition, while the “Sarah” player moves more slowly than the “Joey” player, it still moves faster than any human that I have observed playing the game. The improvement in performance between the “Sarah” and “Joey” players may not have been as great as I had hoped it would be, but the underlying concepts of the player improved greatly. The “Sarah” player gave me a more powerful and expressive heuristic model. I gained a strategy function in the

development method, and the “Sarah” player marked the first time that an artificial player was able to look at a turn as a whole without having to filter the search tree.

6.3 Brian

Finally, the highest level player that I created was the “Brian” player. Brian, the human, was one of the test subjects involved in my original Tarble playtesting session. Brian is a lifelong strategy game enthusiast, and he picked up on many of the subtleties of Tarble’s strategy rather quickly. While Brian did not play as many recorded games as Sarah, he was fascinated by the game and played a good deal of games without my observation. Brian and I also had many conversations about the various strategies that we observed people employ along with the benefits and disadvantages of these strategies. Brian even helped me come up with the situations that I asked people to solve from the human evaluation section. I believe that Brian represents an expert level player. He is at the very least far and away the person who has thought the most about Tarble other than myself. He even helped me think about how to implement the “Brian” player. Therefore, I felt that he was an important player to represent and act as my final human-like artificial player.

Brian was one of the most flexible players that I observed. He was able to adjust his strategy to fit each individual situation. He valued strong central control and board

position very highly, and he would often fail to capture pieces in order to obtain better board control with the hope that it would benefit him later. Brian was also one of the few players that I observed who did not only sacrifice pieces when he was winning and trying to end the game but as a form of retreat. If he could capture a piece, but he could not retreat to a safe distance he would often sacrifice a piece in order to deny his opponent the points. This strategy was particularly effective when he was winning. Brian was also one of the only players that I observed who used jumping distance, the number a player would have to roll in order to capture a piece, rather than board distance when positioning his pieces. I believe that this behavior stemmed from his experience with strategy games, Sid Meier's Civilization series in particular. These games gave Brian a good deal of experience navigating hexgrids, and they taught him the benefits of sacrificing immediate gain for long term rewards.

The first challenge that I encountered in the development of the "Brian" player was that I had to reimplement all of the board distance heuristic functions using jumping distance. This would be a fairly trivial problem if there existed a closed form solution to the jumping distance equation. Unfortunately, there is no such solution because different board positions could yield a different jumping distance between 2 locations. For instance, the distance between a piece and a space 2 hexes away from it is generally 2. However, if an opposing piece lies between the piece and the space, the piece can move to that location in 1 move by jumping its opponent. Therefore, I had to use a search algorithm in order to find the distance. I used branch and bound search with an extended set and a consistent heuristic because if a valid consistent heuristic is

found it is the fastest known search algorithm. The heuristic that I used was the ceiling of board distance divided by 2. This heuristic is consistent because it always provides an underestimate for the jumping distance between 2 points as a player can only jump 1 piece with each move, effectively doubling the speed at which the piece moves. Additionally, the difference in the heuristic between any 2 locations on the board and a goal location will always be less than the distance between those 2 locations. Using this algorithm I can find the jumping distance between any 2 locations on the board in O(N^2) time where N is the jumping distance between the 2 locations. Because N is maximally 10, the performance of the algorithm is not a major concern. After implementing the jumping distance calculator, reimplementing the strategy methods developed for the “Sarah” and “Joey” players was a trivial exercise.

The second major difference between the “Sarah” and “Brian” players is that the “Brian” player completely changes its strategy based upon the current state of the game. I broke the board position up into 4 different cases, early game, late game, end game, and conflict. Conflict is defined as any time that 2 opposing pieces are within 3 moves of each other, early game is defined as when both players have greater than 3 piece remaining, late game is defined as when either player has 3 or less pieces remaining on the board, and endgame is defined as any time that the player currently in the lead can end the game with a roll of 4 or greater on their next turn. Both early and late game only exist when the game is not in a conflict position.

Within each of these board situations the “Brian” player utilizes different strategies depending on whether it is far ahead, ahead, tied, behind, or far behind. Each

of these distinctions are defined by the point differential between the players. Far ahead is defined as 3 or more points ahead, far behind is defined as 3 or more points behind, and the other 3 distinctions have the resulting natural definitions. Without delving too deeply into the details of implementation, the basic idea is that the “Brian” player makes more conservative decisions when it is ahead and more aggressive decisions when it is behind. It gets very aggressive when it is in late game situations and behind because there is even less time to catch up. It is also more likely to sacrifice pieces, either to hasten the end of the game or to deny its opponent points, when ahead. The player even does a modified alpha-beta search in endgame situations when both players have less than 3 pieces on the board in order to more exhaustively search the possibilities. Brian, the human, told me that he often tries to search the gamestate as exhaustively as possible in these end game situations. All of this flexibility allowed the “Brian” player to perform at a far higher level than either or the other human-like artificial players that I implemented, and I believe that the ability to adjust strategies based on different board positions is the biggest factor that sets serious strategy game players apart the average player.

Finally, Brian employed one specific strategy that I found incredibly interesting and present in only the upper echelon of players that I tested. The idea that sacrificing piece can be a beneficial strategy generally takes players a number of games to realize unless their opponent utilizes sacrificing against them. However, even after 23 games, I never saw Sarah capture a piece and then sacrifice the piece that she had just used to capture in order to deny her opponent the point for the capture back. Sarah, along with

most of the players that I studied, would capture a piece only if they were willing to accept a trade or retreat back to a safe distance afterwards. However, Brian, especially when he had a lead, would often capture pieces that were protected well enough that he could not hope to retreat to a safe distance afterwards. Instead, he would use the sacrifice outlets on the corners and center of the board as infinite retreats. While he rarely used this strategy when he was behind because it naturally ends the game more quickly, when Brian was ahead he would actively look for these situations and often favor them over a capture and retreat.

The strategy was simple enough to implement because using the jumping distance calculator the “Brian” player simply had to check if it could capture a piece and then retreat to a sacrifice outlet before running out of moves. However, I believe that this strategy added another level of depth to the skill of both Brian, the human, and the “Brian” player and greatly contributed to their success.

The “Brian” player is far and away the most successful artificial player that I have implemented. The results from its testing games are shown below:

Opponent	Games in Match	Games Won	Game Win Percentage
Monte Carlo	13	13	100%
Neural Network Single Step	13	13	100%
Neural Network Full Turn	13	13	100%
Joey	13	12	92.3%
Sarah	13	13	100%
Brian	13	10	76.9%

Me	13	8	61.5%

My match against the “Brian” player marks my first match loss to a computer. The “Brian” player also defeated Brian, the human, implying that I may have been slightly overzealous in the implementation. The “Brian” player uses the strategies and thought processes of human players, but it has surpassed the skill level of any human that has played Tarble. The “Brian” player is as successful a player as I could have hoped to create.

6.4 Mixture Models

After completing the “Brian” player, there was one final player that I wanted to develop. I wanted a player that could begin as any new human player might begin and develop it skill across multiple games. This player could be used to help new players understand the game because it would scale up in skill level with the new player, constantly challenging the new player without becoming oppressive. I accomplished this goal with a mixture player. The player is initialized with a starting skill level and learning rate. The skill level can be any real number, but for simplicity I defined an easy, medium, and hard player with starting skill levels of 0, 30, 60 respectively, and the learning rate can be any positive real number. By default I set the learning rate to 10.

The skill levels determine how likely any individual turn is to be decided by the 3 human-like artificial players, and the learning rate determines how quickly the player improves. The program generates 3 numbers from independent variance 100 Gaussian

distributions with means 0, 50, and 100. The program chooses to use the “Joey” player to make the move if the mean 0 gaussian produced the closest value to the skill level, the “Sarah” player if the mean 50 gaussian produced the closest value to the skill level, and the “Brian” player if the mean 100 gaussian produced the closest value to the skill level. After each game the program updates its skill level based on the learning rate and its game history with its opponent. If it lost the last game, it will add the learning to the skill level. If it won the last game but lost the one prior, it will add half the learning rate to the skill level. If it won the last 2 games but lost the one prior, it will not change the skill level. Finally, if it won the last 3 games, it will subtract the learning rate from the skill level.

I had the medium level mixture player play 5 game sets with 3 different new players. One player said, “it started out too easy, but I like how it got better to keep up with me.” This sentiment seemed to be the consensus among the 3 players, and the fact that all 3 of them won the first 2 games and 2 of them won the third game back up this claim. It is possible that I could have made the medium and hard level players slightly more difficult, but all 3 of these subjects were also students at the Massachusetts Institute of Technology, which probably puts them in the higher tier of new players. I also considered it very successful that the final game between all of the testers appeared to be the most competitive, 2 of the players lost and 1 of the players won. Implying that the program found the correct level of play within 4 games. There are certainly improvements that could be made to this learning player. For instance, it shifts in ability far more so than a human between one game and the next, and it does not

learn anything during games only afterwards. However, the ability to adjust to the skill level of the opponent does accomplish the goal of giving new players an easy way to play competitive games and learn quickly.

7 Future Work

There are multiple directions that the results from this project can be directed for future work. One interesting idea that I did not have time to implement is to revisit the idea of machine learning artificial players from a higher level strategy point of view. After developing the “Joey,” “Sarah,” and “Brian” players, I have a library of heuristic and high level strategy functions that could be used to more compactly describe the gamestate. A developer could use some combination of high level heuristics as the input to a neural network and the network could choose between the strategy methods rather than all of the legal moves. For instance, the input would be something along the lines of distance to nearest opponent, development score, and the minimum roll required for victory along with other similar measurements. The output would be retreat, attack, develop, or any of the other strategy functions that I have implemented or a developer could create. A similar implementation could be considered for an alpha-beta player as well. Rather than search through all of the possible moves that a player could make on a given turn, the player could simply search through all of the high level strategies that could be

employed on a given turn and evaluate those situations. Either of these methods would greatly reduce the search space of the game making it feasible for these more traditional methods to be successful.

Another way that the research in this paper could be built upon would be to consider applying these concepts to something other than games. With the development of Tarble, I attempted to push the limits of what could be done in the world of games. However, if one leaves the field of strategy games, there are even more complicated problems that could be considered. Robotics is a field that provides the programs with near limitless possibilities. At any given time a robot move in any direction, move at any speed, and interact with the world in countless different ways. There are certainly researchers in robotics who have been studying the daily actions of humans for years in order to better understand how we navigate such a complex world. It is possible that some of the techniques used in this paper such as providing a few important heuristics and viable options could be a useful tool in this field. In many ways Tarble can be thought of as a simplification of the real world.

8 Contributions

Over the course of this project I designed Tarble, a game that made traditional game playing techniques ineffective; I evaluated how humans thought about and played Tarble; I implemented a variety of different artificial Tarble players using traditional

artificial game playing techniques; and I implemented 3 artificial Tarble players that imitated the way humans think about and play Tarble.

Tarble is a difficult game for traditional game playing techniques to play due to a number of reasons. The branching factor is absolutely massive compared to similar board games. A single turn can have billions of legal moves that are possible. Additionally, it is not obvious how a board state should be evaluated. The game is a points based game; therefore, points can be used as an indication of positional strength. However, the intricacies of what defines a well developed position or a dangerous position are far more complicated to assess. Moving even a single piece could massively change the strength of a position. Yet, despite all of these apparent challenges, humans seemed capable of understanding and playing the game with clear strategic patterns within minutes of learning the rules. A major part of this project was finding how to close the gap between the human players and the artificial players. Finally, the game had to be easy to understand and play in order to obtain the amount of testing required for the project. Most players were able to grasp the rules of the game in just a few minutes, and I implemented a graphical user interface that allowed subjects to play and record games with minimal effort.

One of the fundamental features of this project was that I did not assume how people think about and play Tarble. I did human evaluations and found test subjects to play the game, record the games for my future review, and attempt to solve interesting situations that I developed to test specific skills. These tests allowed me to find patterns in human play that I would never have been able to think of on my own. It also enabled

me to find 3 players that I felt embodied important subgroups of human players, and I used these players as baselines for my final players. One of the players even helped me implement a player that imitated him by explaining how he thought about various situations and giving me ideas for heuristics that he uses while playing Tarble. Without the solid foundation in human studies, this project would have almost surely yielded far different results.

In order to support the claim that Tarble is a difficult game for traditional artificial game playing techniques to play, I implemented a number of popular techniques and evaluated the shortcomings of each algorithm. The algorithms that I chose were minimax/ alpha-beta, Monte Carlo tree search, and temporal difference learning. I chose these algorithms because they are all popular and powerful game playing algorithms. The minimax and alpha-beta algorithms are 2 of the oldest and most abundant game playing algorithms. These algorithms have been used to defeat checkers and chess world champions and provide a basis for many of the other game playing algorithms that have since been developed. However, the branching factor of Tarble completely invalidated these algorithms. They could not run in any reasonable amount of time, and they were only able to evaluate moves up to a depth of 1, eliminating many of the strength inherent in the algorithms. Monte Carlo tree search is a much more recent algorithm the has had a good deal of success with game such as Go, which have a much higher branching factor than chess or checkers. However, even this algorithm could not handle the sheer size of the Tarble search tree. The algorithm must explore the space of possible moves in order to make reasonable decisions, but in the time that

I considered reasonable for a single turn it was unable to explore any significant chunk of the search tree. Finally, temporal difference learning has had success in backgammon, a game with a large branching factor and randomness similar to Tarble. A similar algorithm was also recently used to defeat the world champion Go player. However, again the large branching factor along with the difficulties involved with evaluating board positions, particularly board positions partway through a turn, lead to this algorithm performing below expectation. The temporal difference algorithm also has a massive training time, and while it was allowed to train for an entire weekend, it probably would have benefitted from even more training. That none of these algorithms succeeded in playing a game of Tarble as well as a brand new human player supports the claim that Tarble is very difficult for computers to play.

Lastly, I implemented 3 artificial Tarble players that imitated the way that humans play the game. I chose 3 of the test subjects that I had studied, and I used both data from their games along with common patterns that I found among all human play to implement artificial players that played like them. These players used simple high level heuristics like a human would rather than complex computations to evaluate board states, and each player chose a move from a preset list of strategies. These strategies involve actions that a person would use to explain their move such as attack or retreat in order to more closely approximate the thought process of a human player. I then combined these 3 players into a single player that could learn and adjust its skill level to match that of its opponent. The goal of this project was to use games as a lens through which I could gain a deeper understanding of human planning. Through studying and

imitating the thought processes of my subjects I have taken one small step towards understanding how people think about the world.

References

1. Brudno, Alexander. "The Alpha-Beta Algorithm." *Problemy Kibernetiki* (1963)
2. Campbell, Murray, Joseph A. Hoane, Jr., and Feng-hsiung Hsu. "Deep Blue." (2001)
3. Coulom, Rémi. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search." *Computers and Games Lecture Notes in Computer Science* (2007): 72-83. Web.
4. Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce." *Communications of the ACM Commun. ACM* 51.1 (2008): 107. Web.
5. Pollack, Jordan B., and Alan D. Blair. "Why Did TD-Gammon Work?" (1997)
6. Samuel, A. L. *Some Studies in Machine Learning Using the Game of Checkers II - Recent Progress*. Ft. Belvoir: Defense Technical Information Center, 1967. Web.
7. Silver, David, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. "Mastering the Game of Go with Deep Neural Networks and Tree Search." *Nature* 529.7587 (2016): 484-89. Web.
8. Tesauro, Gerald. "Temporal Difference Learning and TD-Gammon." *Communications of the ACM Commun. ACM* 38.3 (1995): 58-68. Web.
9. Turing, A. M. "I.—Computing Machinery And Intelligence." *Mind* LIX.236 (1950): 433-60. Web.

10. Vul, Edward, Noah Goodman, Thomas L. Griffiths, and Joshua B. Tenenbaum. "One and Done? Optimal Decisions From Very Few Samples." *Cognitive Science Cogn Sci* 38.4 (2014): 599-637. Web.
11. Winston, Patrick. "The Next 50 Years: A Personal View." (2012): n. pag. Web.