## horizontal line



**Applications of Monte Carlo Tree Search and Artificial Neural Networks in Chinese Checkers**

02.17.2018

**─**

Sravan Jayanthi

Ishaan Chawla

# **ABSTRACT**

Chinese Checkers represents a complex, unsolved board game that can utilize advanced artificial intelligence to demonstrate the effectiveness of machine learning and simulation algorithms in performing complex decisions. An application of Monte Carlo algorithms, Monte Carlo Tree Search (MCTS), and Artificial Neural Networks (ANN) in an interactive game of chinese checkers. From an initial game state, the computer plays the game against another player using moves it determines to be most advantageous. This process is conducted through MCTS - a simulation is used to compute possible moves which lead to certain game outcome states. These possibilities can be visualized as nodes in a tree that branches outwards from the original game state. The computer play responds to the moves made by its opponent by analyzing the game board and responding with its own move accordingly. After an advantageous move is found, it is used to increase the computer’s chance of winning. However, merely calculating moves may not be enough to beat human players with experience from past games. Therefore, the computer collects information on gameplay and uses ANN to effectively teach itself how to improve its playing in future games. The computer trains a neural network based off specific game state inputlined to the best possible outcomes to certain moves and utilizes experience from MCTS simulations and previous games to develop the most successful algorithm to make moves. Thus, the system sets up an A.I. computer player that successfully wins at least ninety percent of its games against its human and other computer opponents.

# **Introduction**

The object of study within this paper is the application of Monte Carlo algorithms and artificial intelligence in the optimization of multivariable systems with complex inputs and critical decision outputs. The board game, Chinese Checkers, is a board game of German origin where two, three, four or six players compete. The game is played on a hexagram shaped board and the objective is to move all the player’s pieces from the ‘home’ side of the board to the opposing side. Each player starts with ten pieces and utilizes strategic positioning and moves to ‘jump’ across the board. If a piece is next to another piece, the piece may jump and continue jumping if there are adjacent pieces. Chinese Checkers has a board of one hundred and sixty one spaces and can see up to sixty pieces on the board11, as shown in Figure 1.

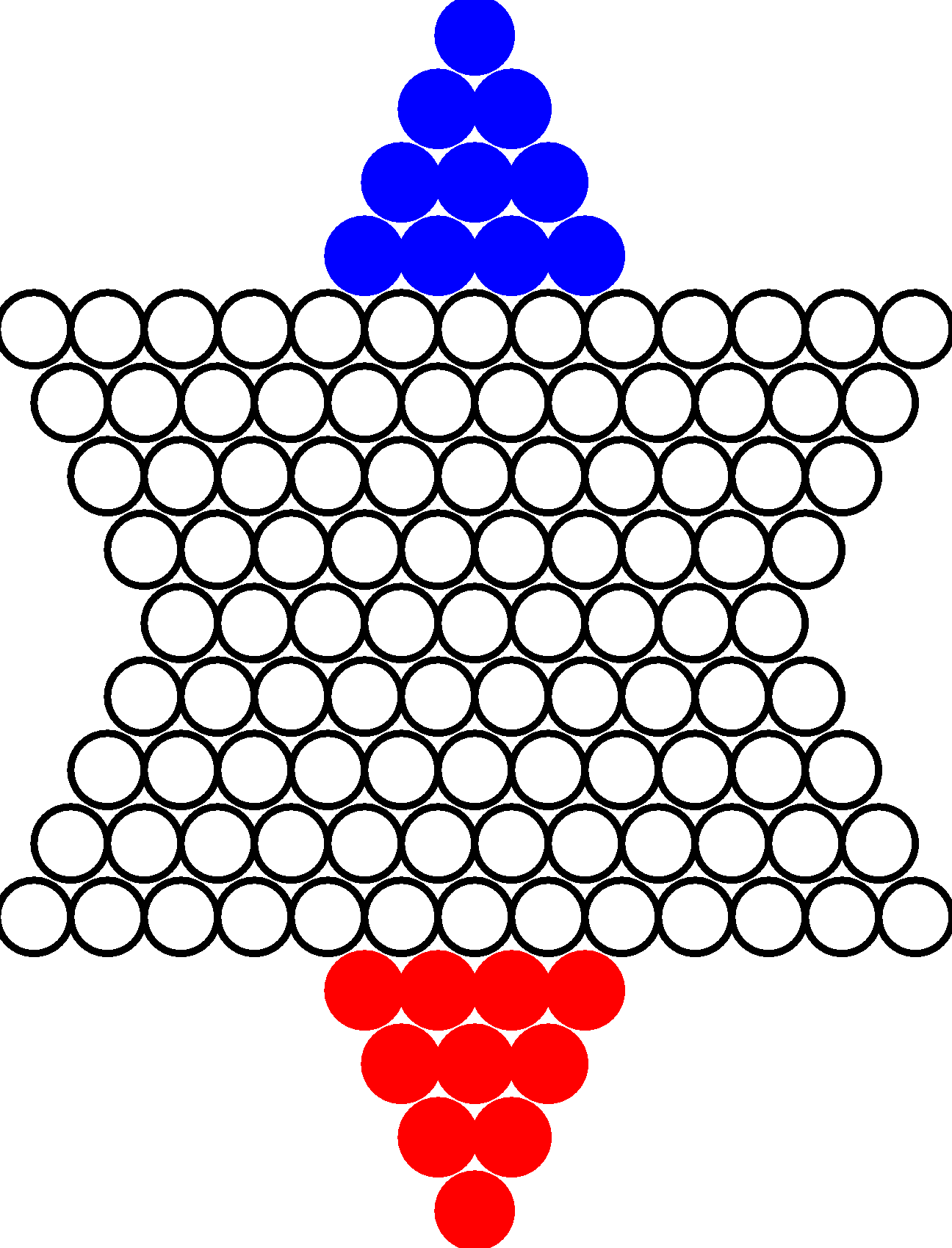


Figure 1. Example GUI of Chinese Checkers game in starting position.

The game of Chinese Checkers is designed so that despite the player interactions being simple, the strategy of the game is complex. The geometry of the board is not square but rather hexagonal with six stars on each segment of the hexagon. The spaces are not perpendicular but rather at angles to each other meaning movement to the final destination is not direct. The pieces can move to any spaces adjacent to the space it currently occupies, usually up to six spaces. There are at least ten rows of spaces a piece must traverse before landing at the goal. The interesting element about Chinese Checkers is the fact that pieces can ‘jump’ over each other. The position of pieces becomes of tremendous importance to allow for friendly pieces to quickly traverse the board by jumping and preventing opponent’s pieces from moving about on the board. Due to this, there are many billions of game situations that can be created in Chinese Checkers11.

There exist strategies that pertain to the beginning stages of the game that provide limited advantage for the player. These create formations that set up pathways to allow pieces to jump across each other. These formations also tend towards concentrating the movement of the pieces in the center of the board, as to minimize the amount movements required to traverse the board. However, beyond the first few moves of a game, these strategies have limited application and do not account for the inevitable situations players will face once into the game. The threats to the player’s success are evident once the pieces enter the hexagon portion of the board. Opponent’s formations can prevent movement, spread out pieces make it harder to jump, and weak formations can allow the opponent to breach into home territory.

There is no extensive record of the game, Chinese Checkers, being ‘solved’, where the outcome of the game can be predicted if all players play correctly12. Players playing correctly refers to the process where players make moves that best suit their interest: winning the game. The players are not making moves that would prevent them from achieving their goal. Thus, ideal strategies for playing Chinese Checkers have yet to be discovered. If it is discovered, then the best possible player of the game can be manifested. Therefore, the goal of this project is to utilize Monte Carlo, Monte Carlo Tree Search, and Artificial Neural Networks in application of a computer program to create the best Chinese Checkers player.

In computer analysis of highly complex situations with exponentiating systems, it is often impractical to analytically determine the most beneficial move to be made3. To mitigate this issue, Monte Carlo methods defer to simulations of potential outcomes, and each simulation utilizes elements of “randomness”, and these factors provide distributions of results to identify the most optimal solution to a problem deterministic in principle. Through many simulations, the algorithm trends to increasing precision because the resulting distribution more clearly identifies the solution10. The ultimate application of the Monte Carlo method in simulation is for optimization, finding the best element (based on a set of criteria) out of a set of elements simulated. The elements simulated are done so randomly so that the population of elements is a representative sample of all possible elements.

In order to enhance the quality of simulations provided by the Monte Carlo method, the application of Markov Chain Monte Carlo (MCMC) sampling models are utilized. The principle of Markov Chain simulation is described by a possible sequence of events in which the probability of the event is determined by the state achieved in the previous event. This form of simulations manifests itself in Monte Carlo modeling as an equilibrium distribution is determined and the simulations of a variable are done to create unique distributions. Each simulation utilizes the information on the accuracy of the previous simulation in the empirical mean to the equilibrium distribution to modify parameters of the simulation to tend towards the equilibrium distribution. If a problem has a probabilistic interpretation, the probability distribution of a variable can be parameterized and a MCMC sampler can be utilized. Random states of the samples derived from MCMC can utilized to approximate a stationary distribution4.

Inherent in decision based problems is that in any situation, there can be derived a best solution based on the variables that govern the situation. The system can explore different situations and link each to the most ideal solution. This is the theory behind Artificial Neural Networks (ANN) and is modeled similarly to the processing mechanisms of biological neural networks in animals and humans. The program is designed to learn and develop its own processing algorithm to find the solutions based off a variety of inputs5. This allows it to no longer require specific programming to address a certain task as the processing algorithm is designed not to require such inputs. In ANN, artificial neurons, a collection of connected nodes, transmit information between each other in the form of real numbers. Each of the connections have different weights when processing inputs the through a nonlinear function9. There are multiple layers of nodes and eventually the output is delivered. The nodes are structured and given their weights based off trials where the system is given an input and directly associated with its successful output1. The nodes are layered and given weights accordingly and this process continues with many trials8, refining the system to work when it is required to provide a successful output for given inputs.

# **Preliminaries**

## **Traffic Simulation**

In beginning research, Original Monte Carlo (OMC) and Markov Chain Monte Carlo (MCMC) methods and experiments were studied profusely and put into action using Java programming. The famous Nagel-Schreckenberg model7, known for its use of OMC to model random driver behavior in traffic, was recreated using Java to gain a greater understanding of the application of Monte Carlo theory.

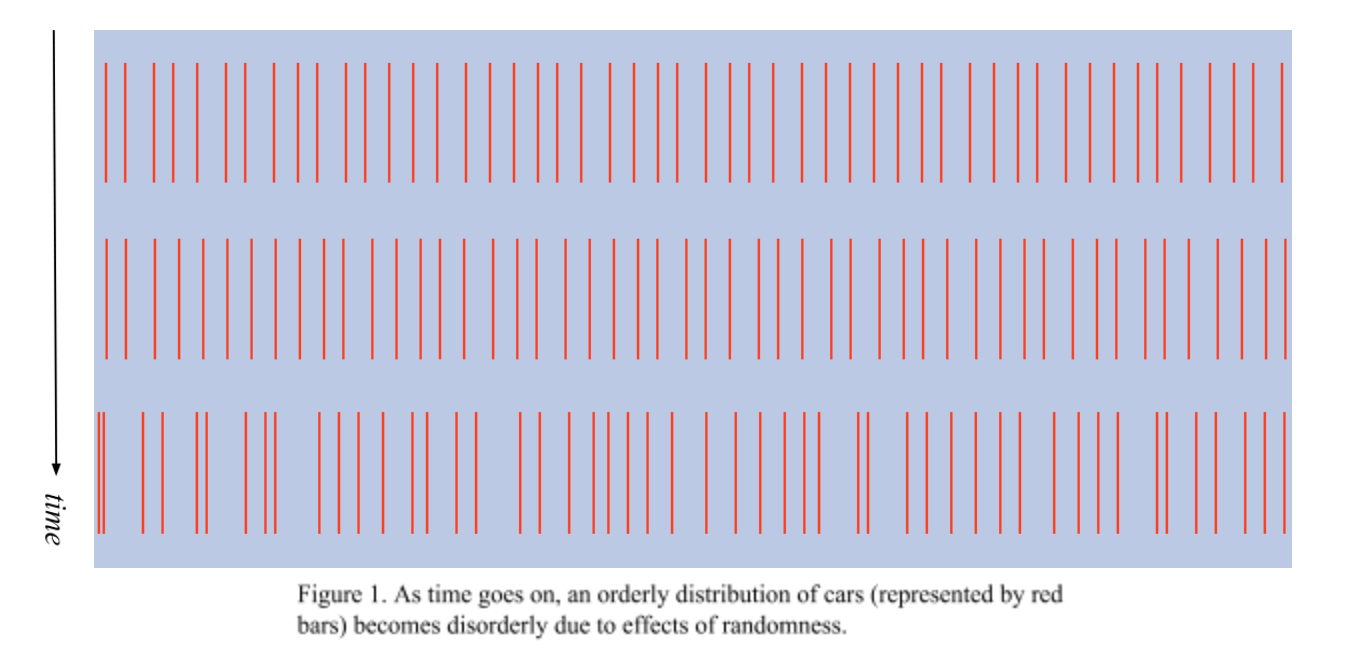
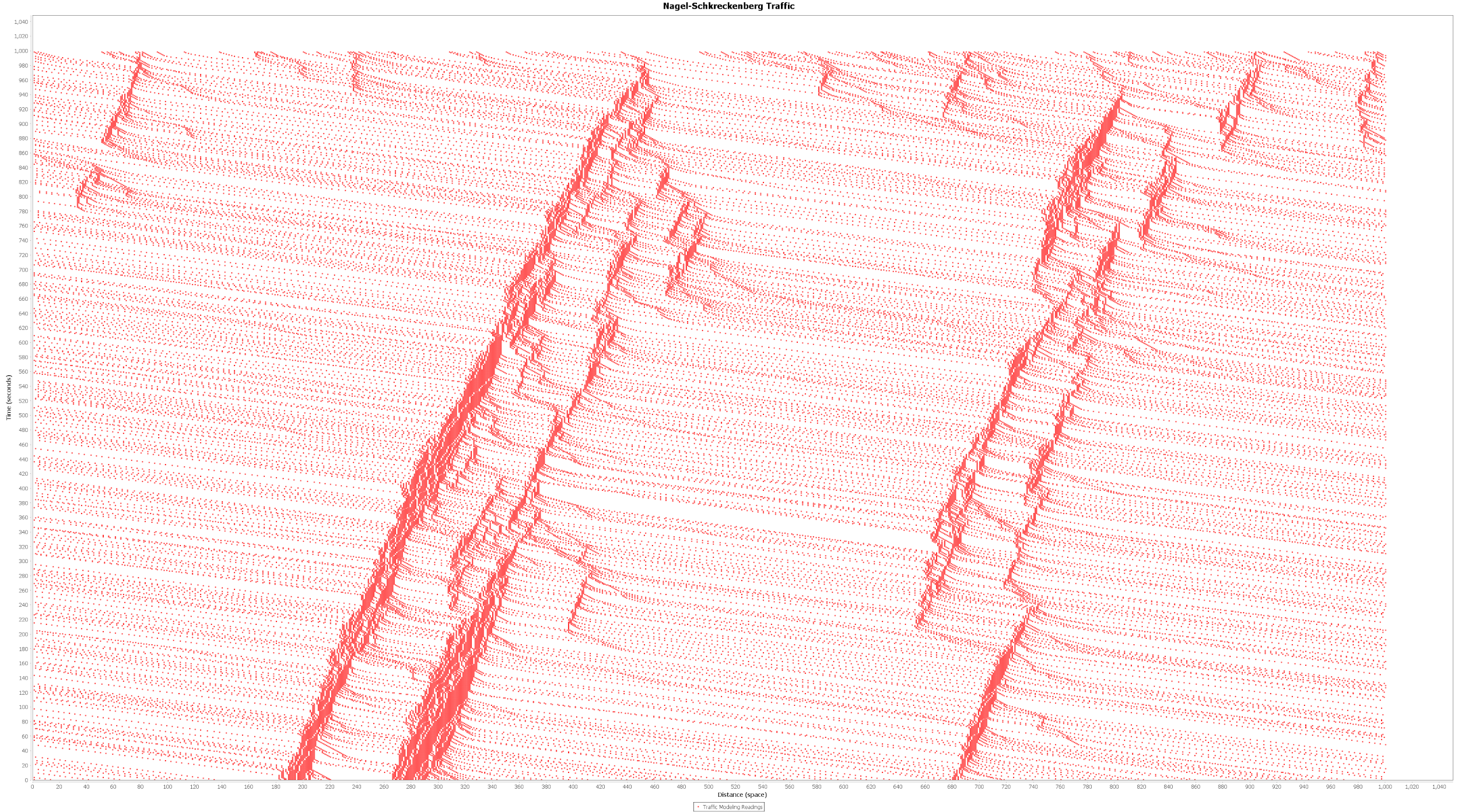


Figure 2. As time goes on, an orderly distribution of cars (represented by red bars) becomes disorderly due to effects of randomness

The experiment was setup with a random variable that would influence the motion of driver behavior. As seen in Figure 2, eventually, the distribution of cars deviates from the starting conditions due to the influence of the random variable. The motion of the cars are governed by four rules: each car seeks to increase speed, their speed cannot exceed the speed limit, they must slow down if there is a car ahead, and the probability *p* of the car slowing down. In each time step, the cars operate on these rules and are repeated until the simulation is complete. The final rule, the reduction of speed randomly, is where driver behavior will be simulated in causing potential traffic jams.



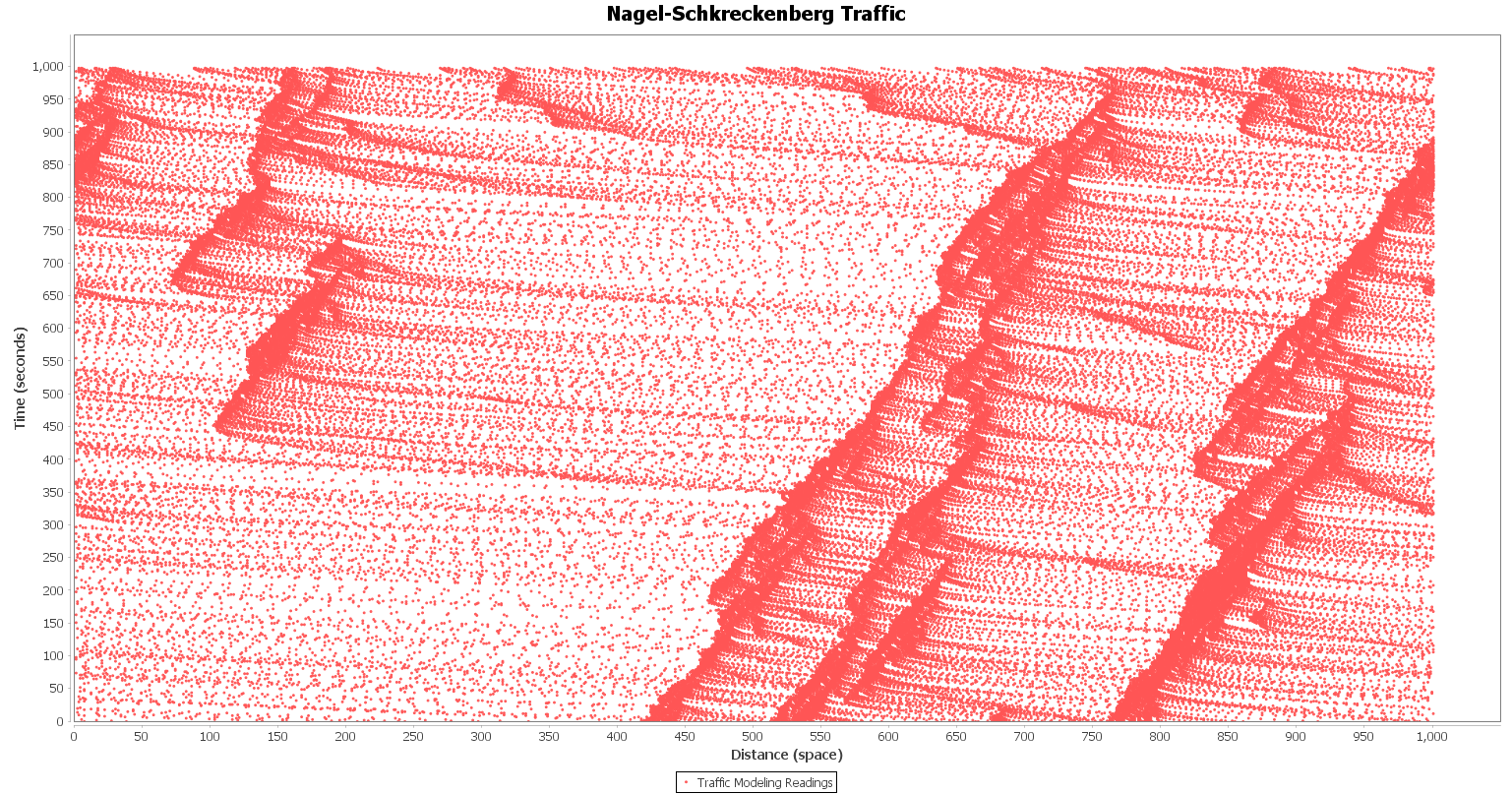


Figure 3. Two trials of the Nagel-Schreckenberg experiment are shown. As time goes on, the spacing between cars becomes increasingly disorderly and varies randomly in separate trials.

The results of the simulations in Figure 3 give insight into why traffic develops and how it propagates through a highway. The random behavior of the driver to slow down has a direct chain effect on the motion of all the other drivers on the highway. Thus, the darker areas demonstrate areas of high traffic that sees many cars forced to slow down to prevent collision. The traffic eventually dissipates at the area of formation but the jam continues on going towards the back of the highway. The traffic propagates in the reverse direction of motion leading to many more cars getting stuck. However, it is the probability that the car will not slow down that allows the other cars to escape the traffic.

The implications of the simulation in demonstrating the MCMC theory are that the algorithm utilizes uncertainty in the inputs to show the “risk” of behavior in instigating traffic. This extends the application by generating a random distribution in effort to approximate the mean distribution. The random element in the traffic simulation will generate different distributions but through many samples, the mean distribution can be reached. Then, the problem of determining the factor that causes traffic can be determined, thus reaching a solution to the deterministic problem.

## **Tic Tac Toe/Connect 4**

To continue the study, further literature of Monte Carlo Tree Search (MCTS)2 was examined to gain an understanding of potential applications of OMC and MCMC to complete tasks of increased difficulty. In order to better understand MCTS, an original experiment was designed with a virtual simulation of Tic Tac Toe. The MCTS application pertained to utilizing an OMC simulation with MCMC theory to generate many random possible simulations of possible moves and determining the ideal move for the player. The Tic-Tac-Toe game has two players who alternate turns and fill in a space in the three by three board until they reach three in a row of their piece. The game often ends in a draw however, there are certain formidable formations that give players distinct advantages in winning the game. Similarly, Connect 4 has two players who instead seek to reach four pieces in a row and can only place the pieces at the open space in the bottom of each column. The goal for both would be for the computer player utilizing MCTS simulations to beat an opponent.

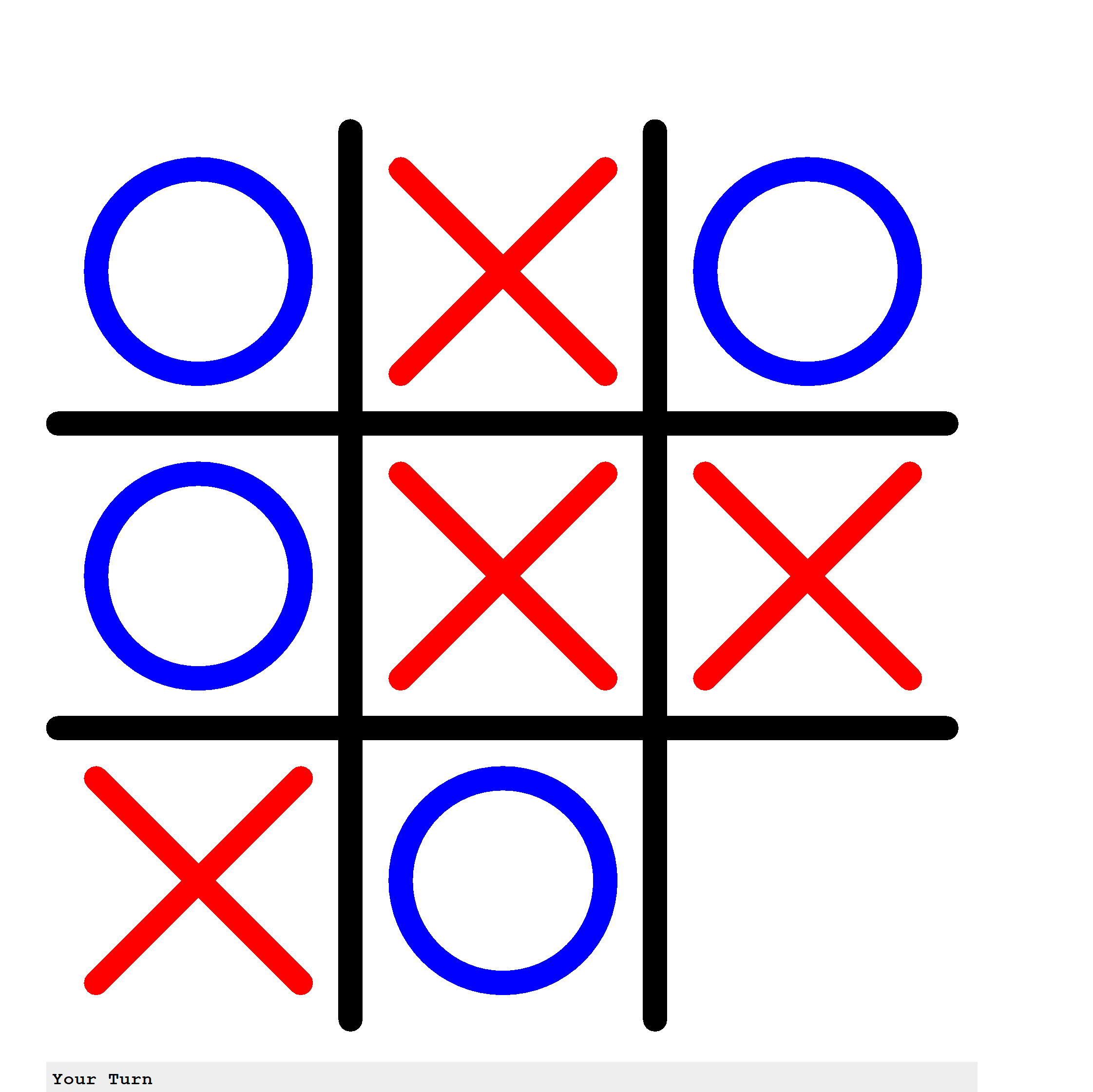


Figure 4. Example GUI of Tic Tac Toe game at a draw. Computer player is blue and human player is red.

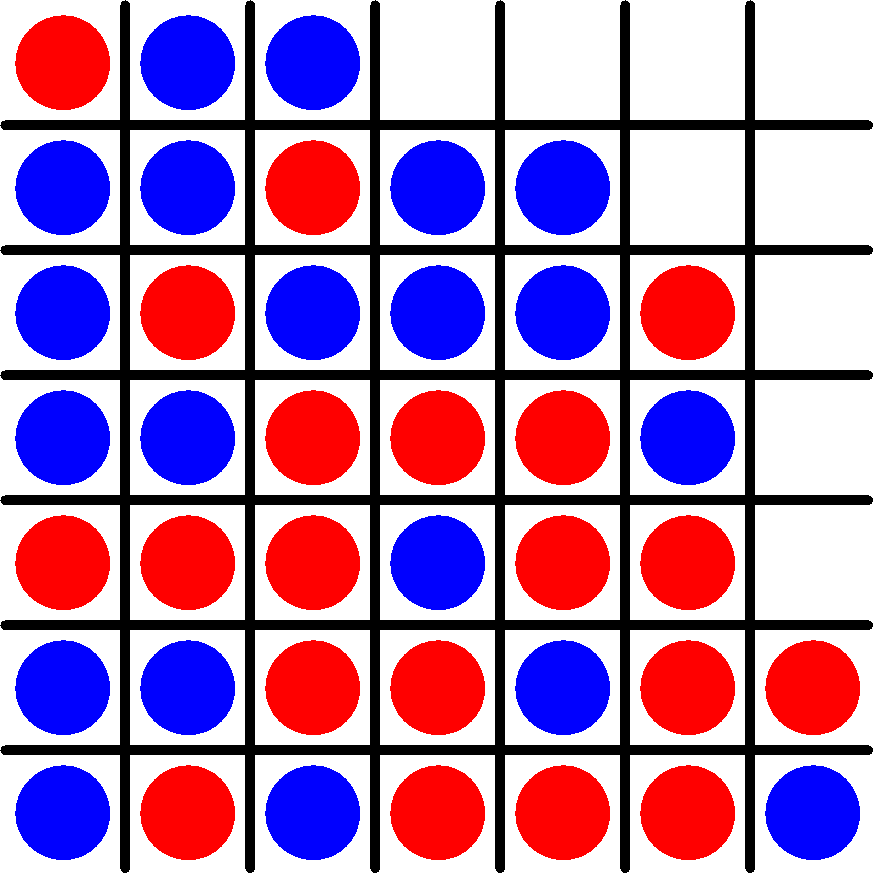


Figure 5. Example GUI of Connect 4 game, computer wins. Computer player is blue and human is red.

The success of the computer within both games as shown in Figures 3 and 4 illustrates the effective performance of computer simulations to win (or prevent a loss). The MCTS algorithms analyze the possible moves for the player and utilizes the results of each simulation to create a win rate for each move. The simulations are organized similar to the structure of a tree, with the parent node being the current state of the game and the children nodes being the state of the game after each successive move by the player. The state of the game is defined by the locations of the pieces on the board. The search into the tree branches is random and thus instead of having to simulate each possible move to determine the best choice, the application of MCMC ensures that the random sampling will point towards the mean win rate for each move.

The application of MCTS shows the decision process implementations of Monte Carlo. The simulations are utilized to determine what move the computer will make when playing the game. The random element of the search allows for a broad mean to be determined for each possible move. The system is structured so that the quantity of random samples generated is sufficient enough to justify that further exploration would not significantly alter the results. Therefore, simulations through random sampling can be implemented for a decision-making system in order to determine the best move for the player.

# **METHODS**

The Chinese Checkers game was created utilizing the Java computer programming language in the Eclipse Integrated Development Environment. The premise of the application was to create an unbeatable Chinese Checkers player. The elements that would constitute the creation of the application would be the methods of Monte Carlo, Markov Chain Monte Carlo, Monte Carlo Tree Search, and Artificial Neural Networks that had been studied. The structure of play would be focused on a limited game between only two players. All rules and regulations for the play of Chinese Checkers would be maintained and the condition for winning would be when one player moves all their pieces to the opposite end of the board.

The game was created with a two dimensional Graphical User Interface that would represent the board. In addition, there was an object created to represent the board and the positions of each of the pieces at each time. Each player has ten pieces aligned in a triangular formation and could interact with any piece on the board and perform any moves that are legally sanctioned. In order to determine what moves were allowed for each piece, upon reaching a player’s turn, simulate the moves utilizing a random samples of possible moves. It would analyze the current state of the game board and see which spaces were opened and which spaces were occupied but the adjacent space was open so a piece could still jump over.

In application of the Monte Carlo method, a simulation is implemented in the process that determines the possible moves, as depicted in Figure 6. The parameters for which a piece can traverse are restricted by the size of the board, which is seventeen units in the horizontal direction (x\_space) and thirteen in the vertical direction (y\_space). Thus, any location a piece can move would be restricted to be inside the playable game board. Due to the fact that Chinese Checkers allows pieces to jump multiple times consecutively, it is unknown from an initial simulation all the possible locations where a piece could eventually land. Therefore, moves of a piece are simulated multiple different times and the final outcome of the pieces’ location is recorded after each simulation. The results of the simulation are compiled and the options are then presented for the A.I. to choose.

**procedure** legal\_moves

Array moves = **function** simulate\_moves

**If** x\_space < 17 **and** y\_space < 13 **Then**

**For** i := 0 **to** moves / 2 **do**

**If** move < board **Then**

**If** EMPTY **Then**

**set** move **to** array

**If** OCCUPIED **Then**

Array new\_moves = **function** simulate\_moves

**For** j := 0 **to** new\_moves / 2

**set** move **to** array

**end**

Figure 6. Pseudocode for simulation method of finding new game states.

The computer player for the Chinese Checkers game resembles that of the Tic-Tac-Toe or Connect 4 game as it involves the Monte Carlo Tree Search method with Markov Chain Monte Carlo. The principle behind its application, seen in Figure 7, is that it generates a specific win ratio for each possible move a piece can make and thus the computer choses to make the move with a certain piece that will maximize its chance for winning the game. The delta terms represent the change in position from each move. Each possible move that the player could make is designated as a Node object in the “Tree Search.” For each node, a Markov Chain simulation is conducted and the tree nodes are explored until it has reached a final outcome in a game. If it is a win for the computer, it transfers that information as a property of that node going back up the tree to the initial move to indicate that it has been successful. Thus, through many simulations, the distribution of win rates tends to the true mean for a certain move. The total explorations into a node will be recorded and the win rate will be determined accordingly. In addition, the initial step in each iteration of the simulation to determine which node branch to explore will be determined by the highest win rate node. The random element enters as for any move that does not have a win rate assigned to it, a random value will be generated and the highest value will be chosen to be explored. Figure 8 shows the process of exploring nodes and assigning weights based on future potentials of success on that pathway.

**function** simulate\_moves

**If** deltaY + 1 **mod**  2 **is** 0 **Then**

**If** deltaX **mod** 2 **is** 0 **Then**

moveY = predictY + ((deltaY + 1) % 2)

**If** deltaX **is not** divisible by 2 **Then**

moveY = predictY - ((deltaY + 1) % 2)

**If** move < board **Then**

**If** EMPTY **Then**

**set** move **to** array

**For** i := 0 **to** 4 **do**

**If** OCCUPIED **Then**

Array moves = function simulate\_moves

**For** j := 0 **to** moves / 2 **do**

**set** move **to** array

**return** array

Figure 7. Code for assigning win rates to different game state outcomes.

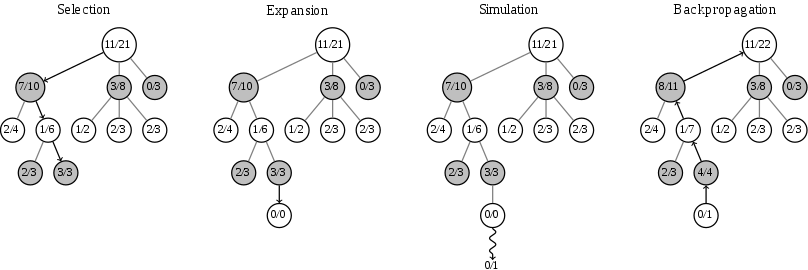
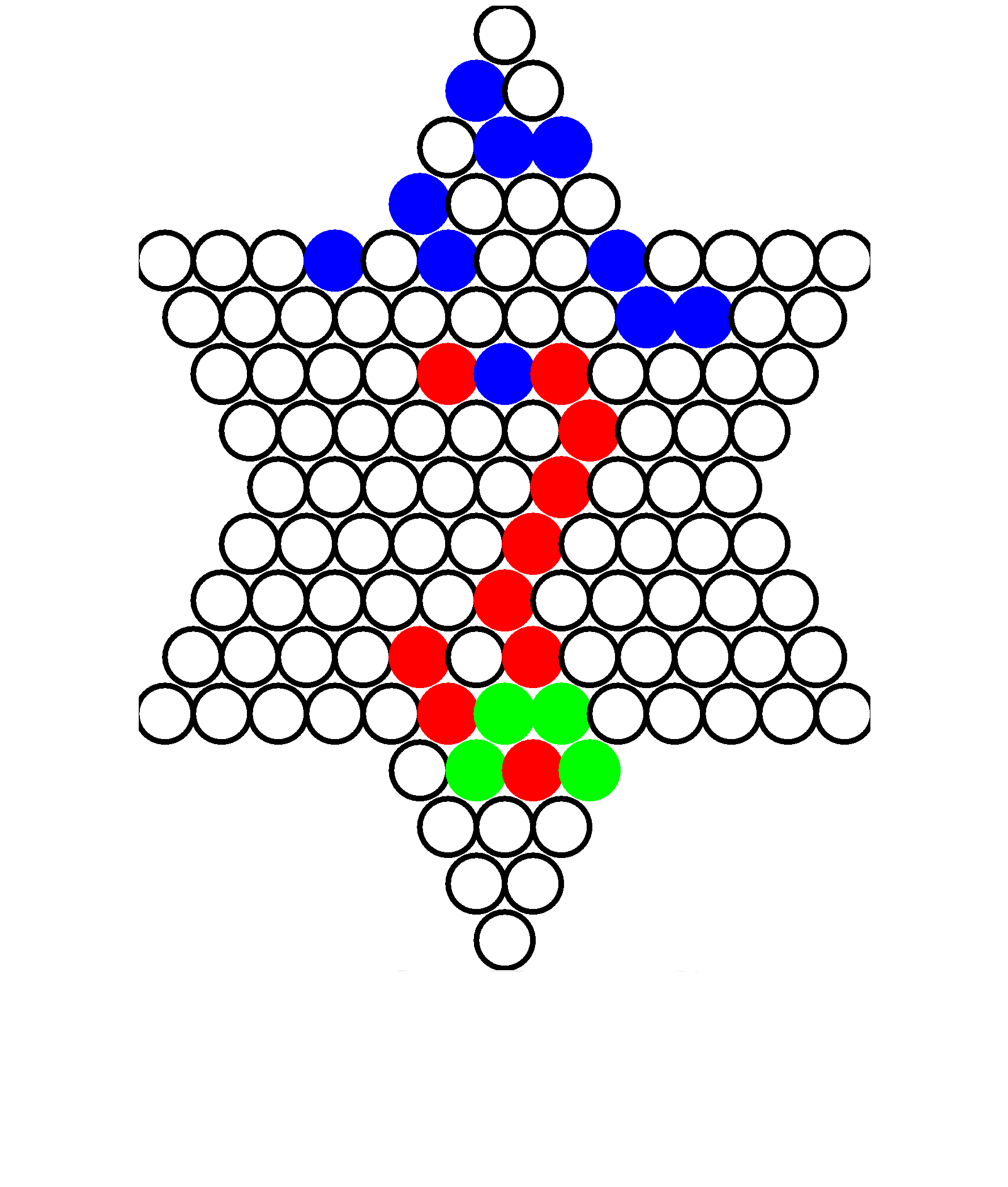


Figure 8. The process of exploring nodes and assigning weights based on future potentials of success on that pathway.

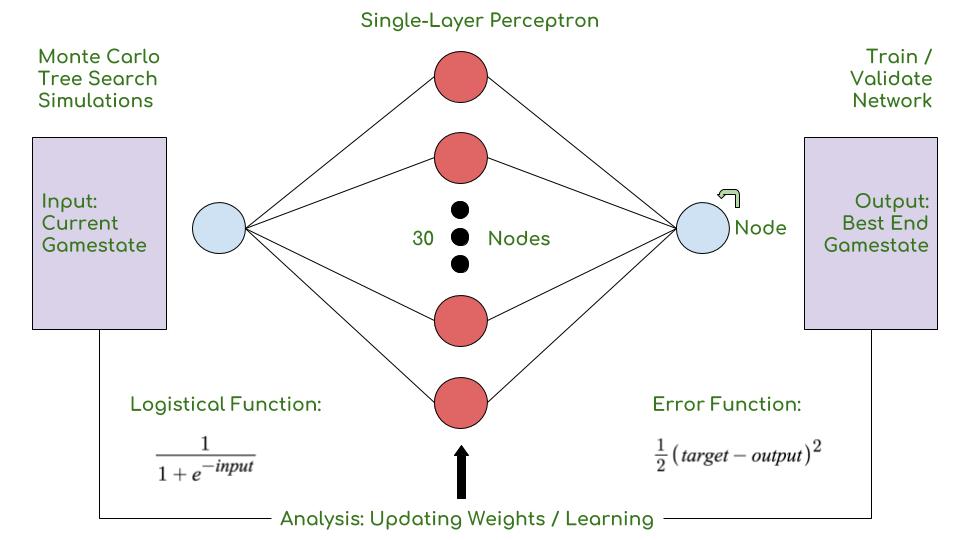
In the final step, an Artificial Neural Network is implemented as shown in Figure 9 in order to optimize the performance of the computer player. The objective of this is to analyze the state of games and the ideal moves in each and develop a series of evaluation functions, the weighted sum of perceptron inputs, within the neural network that would allow for the system when given the input of a board in a certain game state, can utilize the information to tell the player the best move to make in order to win the match6. The input for the Neural Network is the game state, a numerical sequence representing the locations of all the pieces on the gameboard. The game state is a single string of integers encompassing one hundred and twenty one locations on the game board. The two players are assigned separate integers, with player one being the integer 2 and player two being the integer 3. The first integer in the game state integer sequence would be the corresponding player’s integer. Subsequent integers each represent the state of a corresponding location on the game board. If the space on the gameboard is empty, the integer 1 will be used to represent that space and if the space is occupied by a player’s piece, the corresponding integer will be used for that space. The order in of the spaces on the game board represented in the game state number is from left to right in rows descending from the top to bottom. Figure 9 below demonstrates the visual game board that would be represented in the game state array: 31211222111111212112111111111111221111113231111111111311111111311111111311111111131111111113131111111111311111111131111111



# Figure 9: Example of Gameboard at a specific gamestate.

The array is mapped as an input into the neural network with the dimensions of one hundred and twenty two in width, representing the game state array’s length. Due to the fact that the output will resemble the same configuration, as a game state array, there is only a single layer in the network, and thus the input is mapped with one in height. There is only one neuron in the output layer and it will have mapped an array of the same dimensions as in the input. The sequenced input that represents the current gamestate first undergoes a convolution operation to prevent having the size of the inner layer of the network having to be the same length as the array. Thus, the operation requires that the amount of neurons in the network to be limited. The convolution operation is structured to chunk the data into groups of four, with the ends of the array having an extra value. Then a weighted sum of the values is taken and utilized as an input into a node in the network. Thus, the process of organizing the input ultimately allows for the size of the network to be only thirty nodes in size.

The initial stage of the network that includes the convolution operation is the root node, which sends input to each of the inner nodes of the network. The inner nodes have an activation function and an output function, each which consist of weights that are trained to optimize the output. The embedded nature of the node structure allows for a propagation function which passes the outputs to activation functions. Eventually, the final layer will produce end game state code that will document the final game state once the move has been made on the board. This end game state can be compared to the input game state to determine what move was made. The success of the outcome will be judged by a cost function that measures the expected outcome to the actual outcome and provides an error. For the project, the Neuroph framework was utilized to implement the neural network. The element neural network utilized was a multi-layered perceptron network that utilized nonlinear activation functions that added the necessary complexity required for the architecture to allow for unsupervised learning. Figure 10, below, is a specific graphic representation of the structure of the Neural Network. Figure 11, below, is the implementation of Artificial Neural Networks in code.



# Figure 10: Graphic to illustrate ANN perceptron

Depicted here is the logistic function used to take the net input (the sum of each weight multiplied by its respective input) and compute an output, between zero and one, which is then fed into the error function to determine each individual error value. These values are divided by the number of training examples to find total error. Then, the activation function is partially differentiated to determine new weights for future plays.

**procedure** run\_ann

tidouble **sigmoid**

1.0 / (1 + exp(-x))

//----- forwardpass I[i] -> y[j], y[j] -> y[k] ---------------------------------------

**NeuralNetwork** :: forwardpass( )

**for\_i**

x = x + ( I[i] \* w[i][j] )

y[j] = **sigmoid** ( x - wt[j] )

**for\_j**

x = x + ( y[j] \* w[j][k] )

y[k] = **sigmoid** ( x - wt[k] )

tidouble **error** = 0

**for\_k**

**error** = (**error** + (y[k] - O[k])^2)/2

**for\_k**

*dy*[k] = y[k] - O[k]

*dx*[k] = ( *dy*[k] ) \* y[k] \* (1-y[k])

//----- backpropagate *dx*[k], w[j][k] -> *dy*[j] -> *dx*[j] -> w[i][j], wt[j] ---------------

**NeuralNetwork** :: backpropagate()

**for\_j**

tidouble t = 0

**for\_k**

t = t + ( *dx*[k] \* w[j][k] )

*dy*[j] = t

*dx*[j] = ( *dy*[j] ) \* y[j] \* (1-y[j])

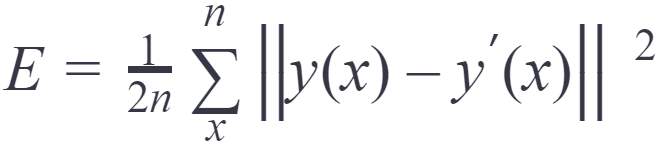
**for\_j**

**for\_k**

*dw* = *dx*[k] \* y[j]

w[j][k] = w[j][k] - ( **r** \* *dw*[k] )

Figure 11. Code for ANN implementation

The main learning paradigm for the ANN architecture would be supervised learning that uses an input and desired output. The ANN architectures implemented were all trained with the same dataset of initial gamestate numbers and ideal end state. In order to test the variation from the ideal output, the architecture implemented the Mean Squared Error loss function. The objective of this method is minimize the average squared error between the network's output, and the target gamestate value. Thus, in the backpropagation algorithm, the weights in the activation functions in each node are adjusted to reduce the error. The error function utilized: . The chart in Figure 12 shows the eight training attempts underwent to determine the appropriate learning rate, in order to make the training reliable and optimized. Also, the number of training epochs was adjusted to avoid skipping the local minima and smooth out variations in the descent of the gradient. The graph in Figure 13 demonstrated the loss curve for the initial training of the Neural Network while the graph in Figure 14 shows the loss curve for the sixth training of the Neural Network. Eventually, the sixth training attempt was determined to be the most ideal as it demonstrated the least error. This sixth method was trained with 173 epochs which was ideal to avoid underfitting the dataset in the neural network while preventing overtraining that would lead to memorization of desired outputs. The graph in Figure 15 shows the test and validation curves with respect to the training loss curve.

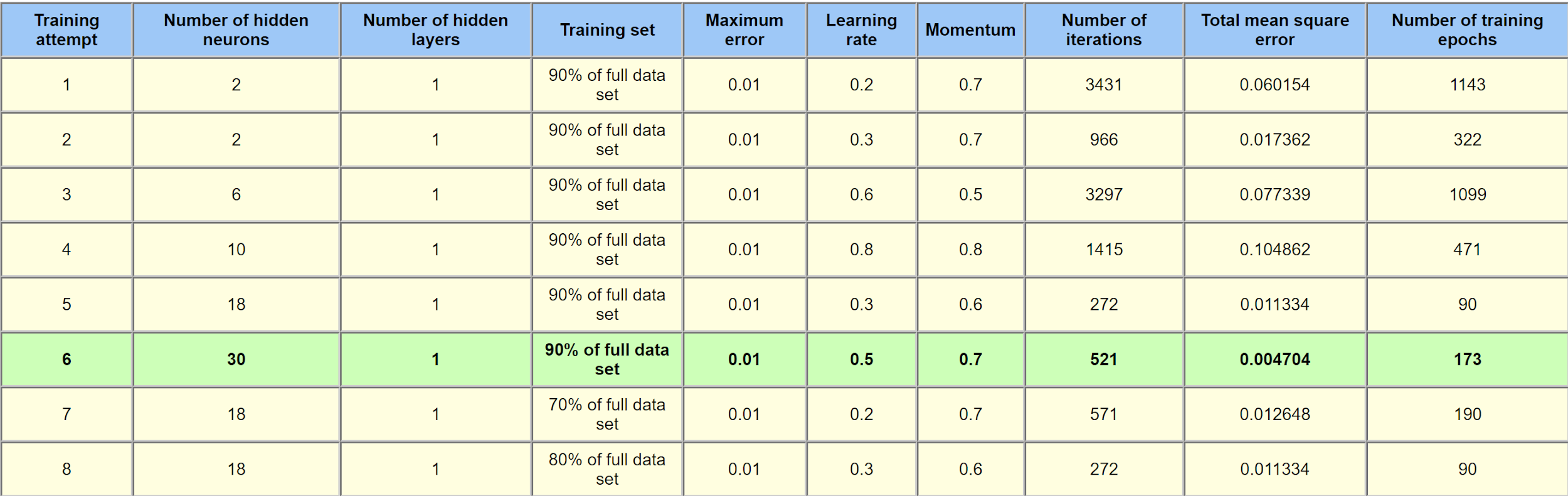


Figure 12. Chart of Supervised ANN Training Results

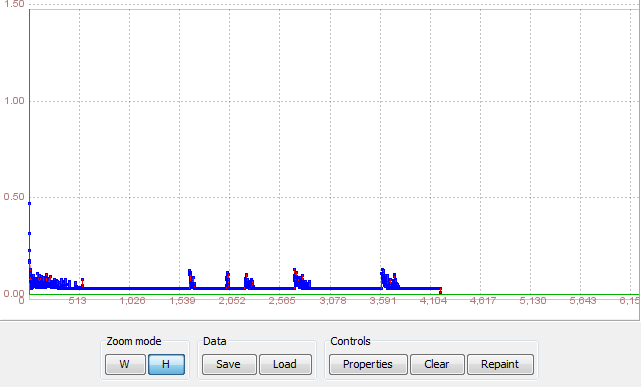


Figure 13. Loss curve of initial ANN Trial

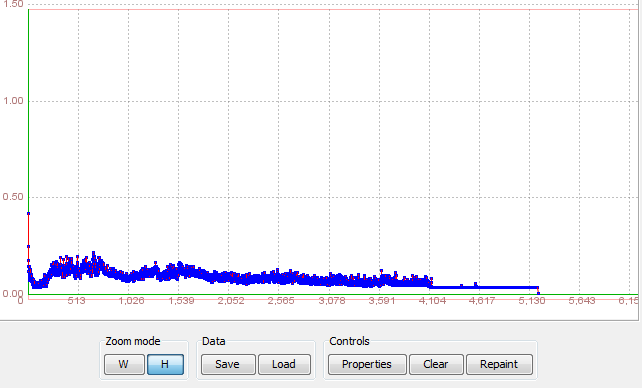


Figure 14. Loss curve of sixth ANN trial

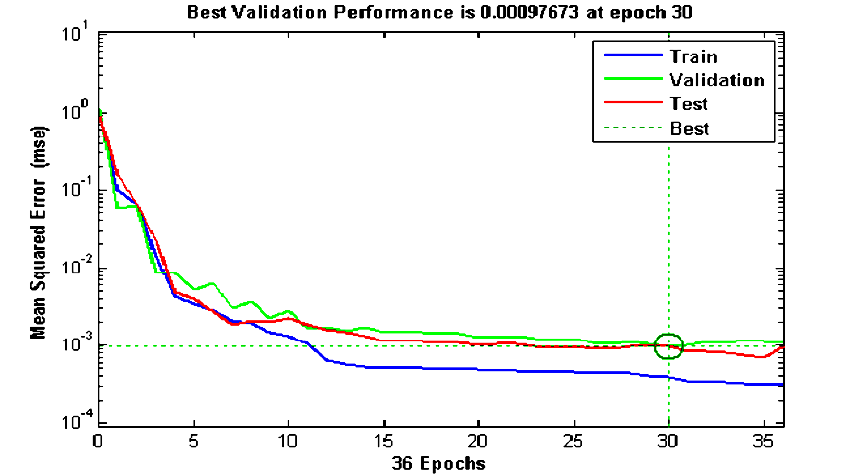
****

Figure 15. Validation, Test, and Training curves up at 30 Nodes

# 

# **RESULTS**

The test results after the creation of the Chinese Checkers computer player demonstrate the high competence of the Artificial Intelligence system. Each iteration of the computer player was tested against the other iterations to see the success rate of each player against each other. In addition, a beginner human player was introduced to play against each iteration of the computer player. Each matchup consisted of a hundred games played between each of the players and the success of a player was determined by the win rate of the player defined by the games won divided by the total games played.

# Table 1: Performance by different players.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Players | Original Monte Carlo | +Monte Carlo Tree Search | +Artificial Neural Networks | Beginner Human (10 games) |
| Opponents |
| Original Monte Carlo |  | 0.84 | 1.00 | 0.4 |
| +Monte Carlo Tree Search | 0.16 |  | 0.92 | 0.2 |
| +Artificial Neural Networks | 0.00 | 0.08 |  | 0.0 |
| Beginner Human (10 games) | 0.6 | 0.8 | 1.0 |  |
| Net Win Rate | ~0.25 | 0.57 | ~0.97 | ~0.2 |

# Table 2: Performance of Artificial Intelligence with Artificial Neural Network against iteration with only Monte Carlo Tree Search

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Set of Games | 1-20 | 21-40 | 41-60 | 61-80 | 81-100 |
| Win Rate | 0.85 | 0.90 | 0.90 | 0.95 | 1.0 |

The win rates of each of the players after playing ten matches against each other demonstrate the strength of each of the systems in playing Chinese Checkers. The final iteration of the computer player that implemented Artificial Neural Networks was the most successful at playing Chinese Checkers based on win rate. It was able to win all its games in the matchups against Original Monte Carlo and the human beginner. The complexity of the Artificial Neural Networks allows it to analyze any situation it faces and through past experience, can determine the best move. As the system continues to play, it deepens the complexity of the network to allow it to produce stronger outcomes based on the current state of play. Thus, it performed the best out of all the players demonstrating its ability to not only account for all possible moves but to know which moves tend to lead to the best results. The other players, the Monte Carlo Tree Search iteration was also significantly better than its counterparts other than the Artificial Neural Network iteration. There is more nuance to the simulation methods of MCTS and this allows it to explore more of the best moves. Against the beginner human player, the MCTS and Original Monte Carlo performed well as they are both able to explore many possibilities while a human has limited insight into the game.

Along with the win rate, the competency of the computer player can also be judged by the time it takes for the player to complete a move. The less time needed to complete a move demonstrates the efficiency of the system in making decisions in a timely manner.

# Table 3: Average Time to Perform Move.

|  |  |
| --- | --- |
| Players | Average Time for Action (seconds) |
| Original Monte Carlo | 12.924 |
| +Monte Carlo Tree Search | 7.093 |
| +Artificial Neural Networks | 3.320 |
| Beginner Human | 3.758 |

The final iteration of the computer player was the most efficient player as it was able to complete its move with the least amount of time taken. This is due to the fact that simulation of all possible outcomes is not required. The human player has a similar decision making structure that resembles that of the Artificial Neural Network. The MCTS and OMC players take far longer to complete an action as part of their decision making process includes meticulous simulations which are relatively inefficient players.

Overall, the computer player that implements computational algorithms and decision-making processes of Monte Carlo, Monte Carlo Tree Search, and Artificial Neural Networks is a highly competent player as it is the most successful when it plays and is the most efficient in making moves.

# **DISCUSSION**

Within the final product of this research, a project which can effectively handle a two player simulation of Chinese Checkers and return moves with a strategy built on MCTS was engineered from scratch. Although strategies currently exist for human players, the strategy used by the computer within the virtual simulation is entirely based off of artificial intelligence applications alongside machine learning (ML) algorithms. The game is able to utilize a tree of nodes representing possible moves, and moves from those moves, and so on. This joint MCTS and ML function continues for a fixed period to allow the computer to determine possible moves but still generate efficient gameplay. After the time has elapsed, the computer determines the most advantageous first move from the first layer of nodes and uses that selection. The play then returns to the human player, and this pattern repeats in a cyclical manner.

In addition to the benefits of MCTS and ML, a third method to generate the most successful move is used. MCTS and ML have their abilities restricted to the confines of one game - every game, the strategy is largely the same, and the average results do not improve or worsen. Although MCTS introduces elements of probability in searching for possible nodes / moves to consider, there is no knowledge transferred from previous games, or even previous moves, that aid in making new decisions. In contrast, ANN acts using refining capabilities to enhance the system in the long term, making a computer player which can effectively teach itself to outperform its past performances and continue to improve its decision making in future moves within one game or even new games.

The applications of this research extend beyond games such as Chinese Checkers. Complex, multivariable situations exist in many pressing, real-world situations encompassing a wide variety of fields ranging from medicine to engineering to many more. As an example, to design fuel efficient airplanes, computational fluid dynamic simulations and testing is required. Such tests and computations require the use of exabyte computers which require large amounts of time and power. By utilizing systems incorporating MCTS and ANN, not only can probabilistic models find more optimal solutions sooner, but such modeling softwares can teach themselves to perform better over several tests. These ideas stretch to any field requiring computations and calculations beyond the scope of analytical computing.

# 

# **REFERENCES**

1- “A Basic Introduction To Neural Networks.” *Operating Systems: Three Easy Pieces*, University of

Wisconsin, (2006).

2- Chaslot, Guillame. “Progressive Strategies for Monte Carlo Tree-Search.” Maastrichtuniversity.nl. June 6, 2008. dke.maastrichtuniversity.nl/m.winands/documents/pMCTS.pdf.

3- Dalal, Gal. “Safe Exploration in Continuous Action Spaces.” Arxiv.org. 2017.

arxiv.org/pdf/1801.08757.pdf.

4- Diaconis, Persi. “The Markov Chain Monte Carlo Revolution.” Ams.org. 2008. www.ams.org/journals/bull/2009-46-02/S0273-0979-08-01238-X/S0273-0979-08-01238-X.pdf.

5- Gherrity, Michael. “A game-learning machine.” *University of California*, (1993).

6- Mazur, Matt. “A Step by Step Backpropagation Example.” Mattmazur.com. November 21 2017,

mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/.

7- Nagel, K. and Schreckenberg, M. “A cellular automaton model for freeway

traffic.” *Journal de Physique I*, (1992): 2221–2229.

8- Reidmiller, Martin. “Learning by Playing – Solving Sparse Reward Tasks from Scratch.” Arxiv.org. 2018. arxiv.org/pdf/1802.10567.pdf.

9- Thrun, Sebastian. "Learning to play the game of chess." Papers.nips.cc. 1995.

https://papers.nips.cc/paper/1007-learning-to-play-the-game-of-chess.pdf.

10- Tesauro, Gerald. "Comparison training of chess evaluation functions." Dl.acm.org. 2001. https://dl.acm.org/citation.cfm?id=644397.

11- “The Rules of Chinese Checkers.” Mastersofgames.com. 2018.

www.mastersofgames.com/rules/chinese-checkers-rules.htm.

12- Yuan, Yang, and Jiafu Wu. “An AI Agent to Play Chinese Checkers.” Stanford.edu. 2016.

web.stanford.edu/class/cs221/2017/restricted/p-final/jiafuwu/final.pdf.

**Code Repository:**

https://github.com/SVJayanthi/ChineseCheckersAI