Abstract

While artificial intelligence is slowly creeping into the mainstream and finally being recognised by the average person, AI has been around for longer than most people are aware of. One of the first examples of AI was back in 1951 with the video game "NIM" (Lucas et al, 2016). This was surprisingly 20 years before "Pong", and while the AI was rudimentary and not necessarily "smart", it was able to win games against almost all the higher skilled players.

This report will concentrate on the game "Connect Four" and the theoretical and practical approach of creating a perfect bot. We will be using two algorithms (Artificial Neural Network & Minimax) and evaluating the performance of both. To begin, we will elaborate on how each algorithm works within the "connect Four" rules, we will conclude with which of the two is better for the task and why, how the algorithms reached the solutions, and if there were any problems with the algorithms that hindered our project. The results we found were quite surprising with the Minimax being a clearly superior algorithm, but only if the time allocated to make a move was limited.

Introduction

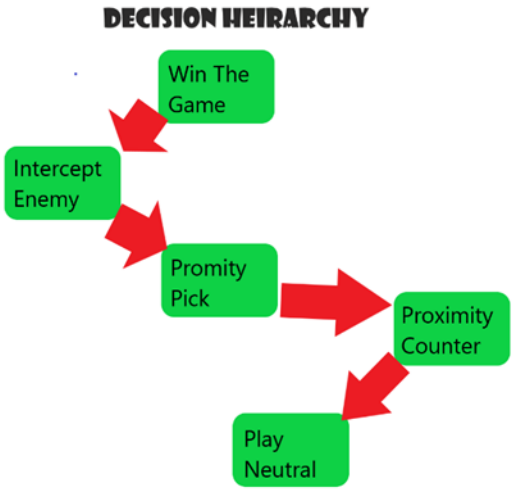
There have been many approaches to solving the Connect Four problem, with some being successful and others not so. Nonetheless it is a commonly used game for algorithm analysis with countless developers wanting to create the unbeatable bot and competitions being held regularly. While Connect Four seems simple on paper, there are over 4 Trillion ways to fill a Connect four field. Bruin, Pijls, Platt & Shaeffer (2017) mention that this number is extremely large and to have, for example, a rule-based system would be extremely ineffective. That turns us to the ANN and the Minimax algorithms. The motivation behind our choice of the minimax algorithm is because it is a widely used algorithm in relation to games, particularly 2 player games (checkers, tic-tac-toe, connect four). It was also the most recommended algorithm throughout the bot websites.

Connect Four was first solved by James D. Allen(Tromp, 2014) followed by Victor Willis 15 days later(Willis, 1988). Inspired by this, John Tromp set out to strongly solve the game by computing all the positions after 8 plays with their corresponding score. The score being 1, -1 or 0 if the game will end up being a win, a loss or a draw for the first player. The score is computed with the assumption being that each player plays perfectly. With that assumption, and with Connect Four satisfying the criterions of a Perfect Information sequence game, it is then clear that the current board configuration is the only determinant of the end-game result. The question is then, can a certain function $Y = f(X)$ be discovered that takes for input the current game-board configuration and outputs a score for the current player (the player who's turn it is to play)? Using an Artificial Neural Network with back propagation of errors seemed the best to discover such a function.

To begin with, this report will concentrate on each algorithm separately, describing the theory of each, how the AI interacts with the connect four board, how it places tokens, and how it will inevitably win the game. We will then produce a comparative study by placing each algorithm head to head in a real case scenario. This will allow us to accurately evaluate each algorithm's performance and ultimately conclude which algorithm is more effective.

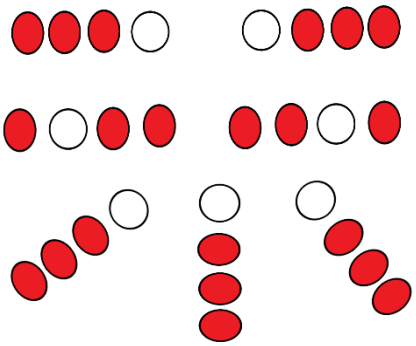
Research Approach

In order to design an intelligence system, we must first define the rules of connect four & the heirarchy of these rules so that the system can determine which move to make.



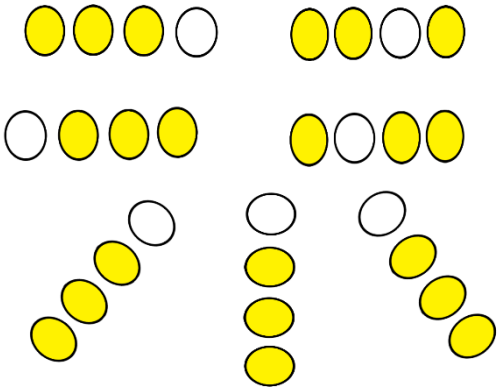
WIN THE GAME:

The diagram below shows the combinations where the bot will attempt to "Win the Game".



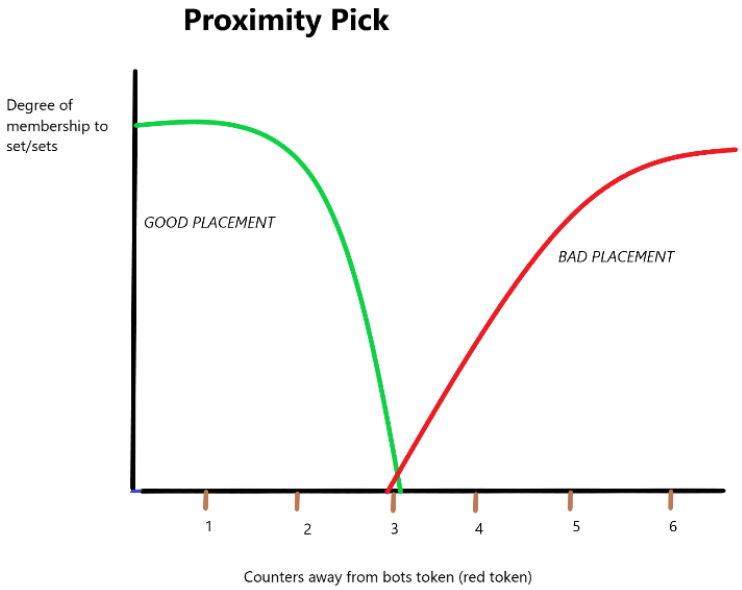
INTERCEPT ENEMY:

The diagram below shows the combinations where the bot will attempt to “Intercept Enemy” as the enemy is one move away from winning.



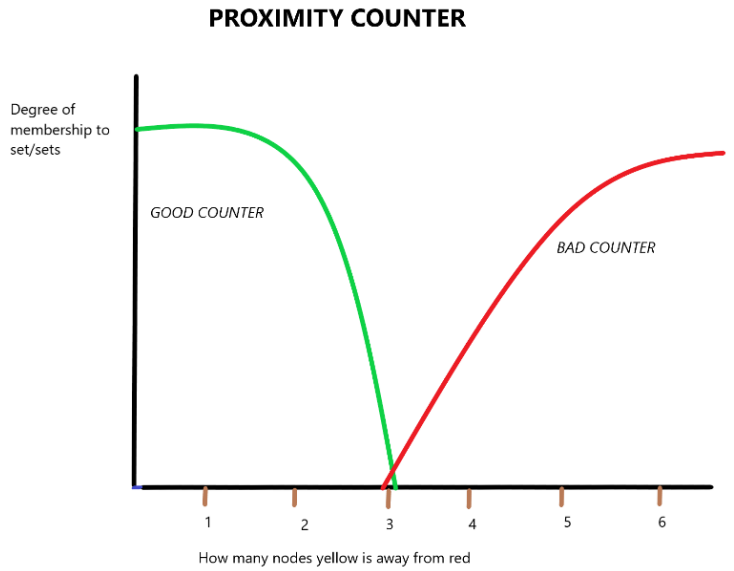
PROXIMITY PICK:

The diagram below provides criteria for the bot to determine the worth of a follow up move. If the move is within bad placement (>3), the move is not executed. If there are multiple good moves the Intelligent system must determine which move to make.



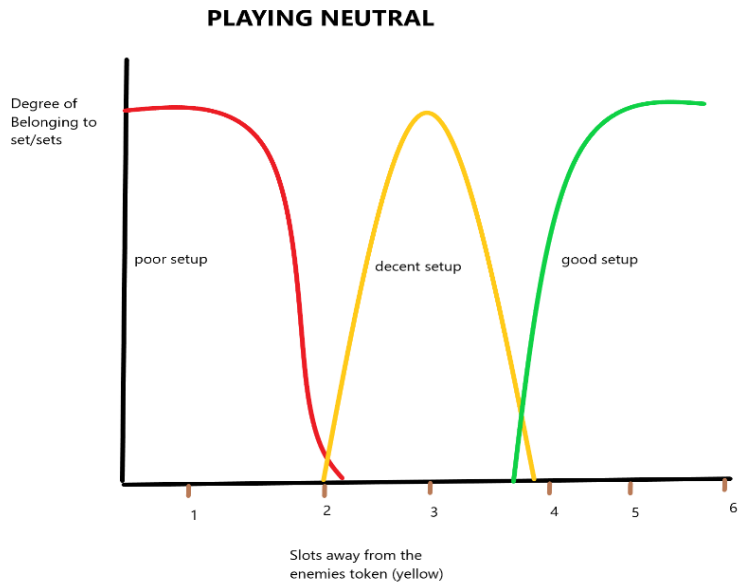
PROXIMITY COUNTER:

The diagram below provides criteria for the bot to determine the worth of a counter move (a move to intercept the enemy where the enemy will place a token in order to build up tokens). If the move is within bad counter (>3), the move is not executed. If there are multiple good moves the decision is hierarchy based. If there are multiple good moves the Intelligent system must determine which move to make.



PLAYING NEUTRAL:

The diagram below provides criteria for the bot to determine the worth of a move in the games neutral state such as the start of the game. In this process regardless of the set the move is executed, where the bot performs the move closest to the good setup set. In the case of the two bet moves having the same value the move is determined by the intelligent system.



Minimax Algorithm

This part of the report will describe the ins and outs of the minimax algorithm, particularly the minimax algorithm in Connect Four. To truly understand how minimax works we must first break it down into understandable blocks. We must also understand that minimax is a recursive algorithm and is called upon at the start of each round.

Displayed below is a simple minimax algorithm flowchart with each section indexed for later explanation.

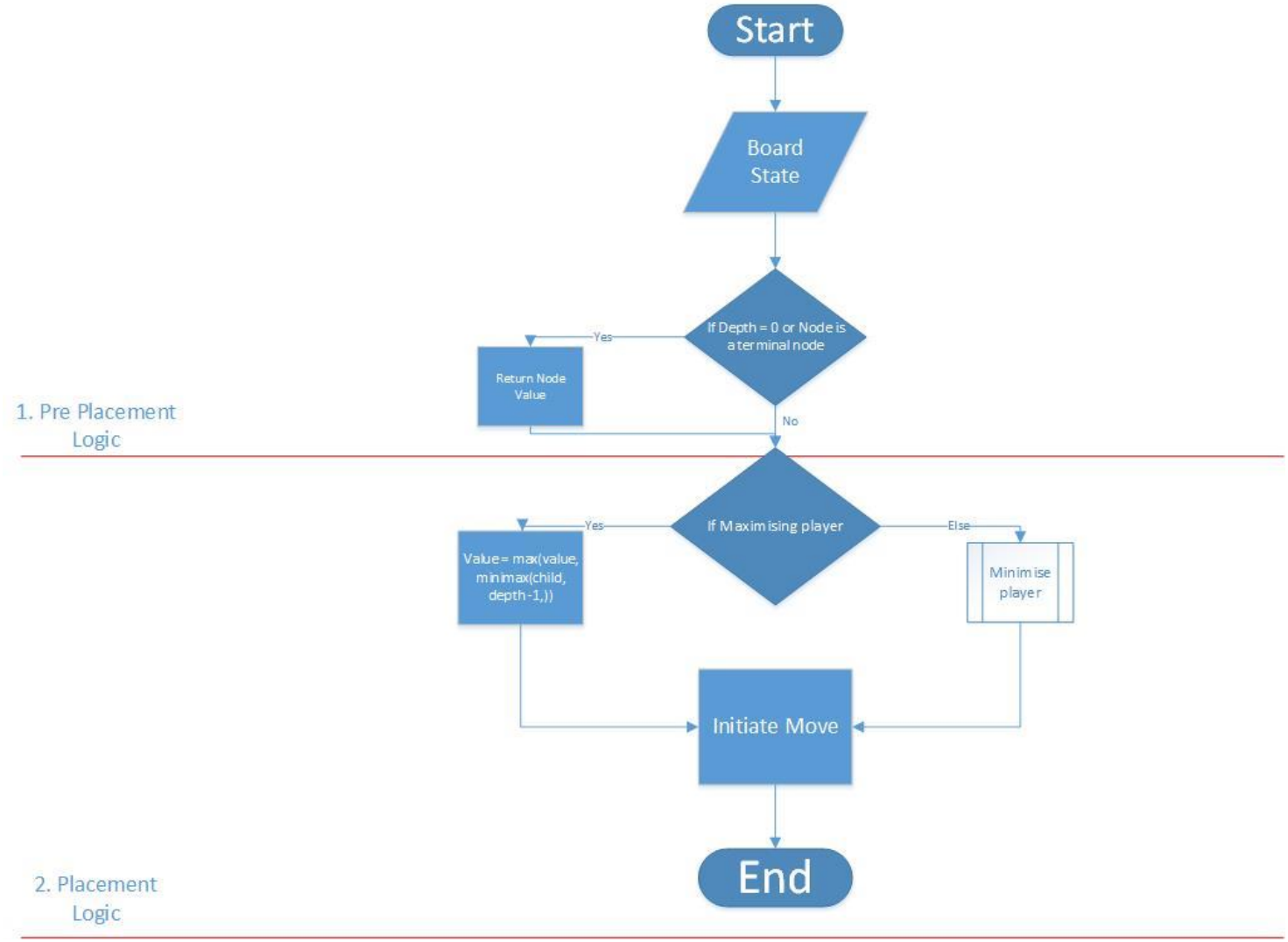


Figure 1Minimax Flowchart

Now that we have a visual representation of the minimax algorithm, we will begin to explain it in detail with the use of pseudocode and outsourced images.

The algorithm can be split into two sections, the “Pre Placement Functionality” and the “Placement Functionality”.

Pre placement Functionality

The “Pre placement Functionality”, section #1, is the simplest part of the algorithm but is also one of the most important (Bruin, Pijls, Platt & Shaeffer, 2017). It assigns a score to each position or state of the game by means of a position evaluation function.

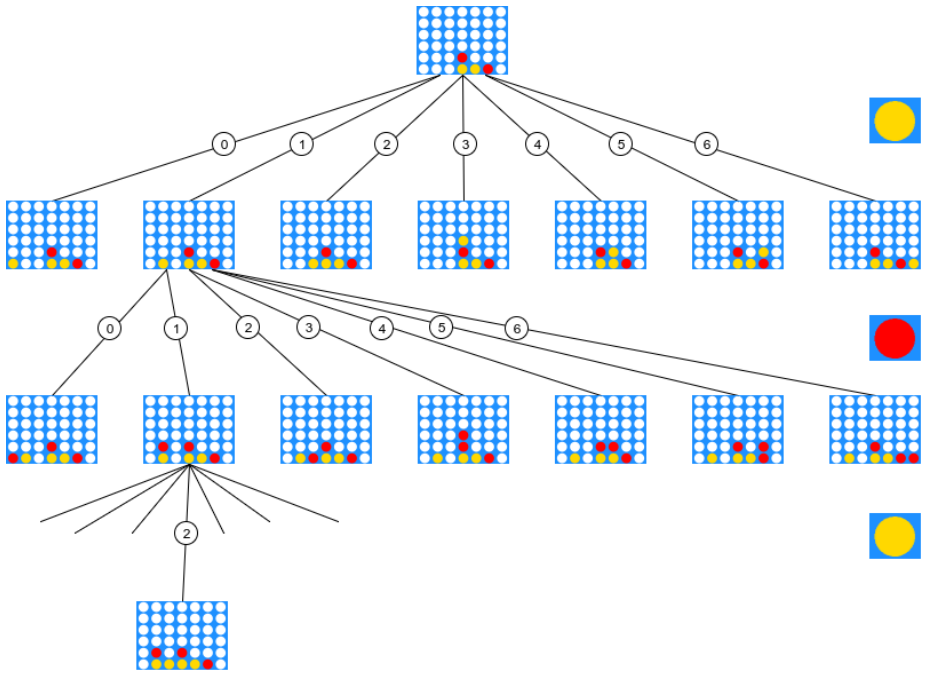


Figure 2 https://medium.com/@gillesvandewiele/creating-the-perfect-connect-four-ai-bot-c165115557b0

For example, a move that would result in the player winning, will have higher score value than a move that results in the player losing (Duchi, Wainwright & Zhang 2015). That score is then used to determine what move would be the best to move to maximise the players score, hence the name minimax. The pseudocode for this functionality is shown below:

```
# @player is the turn taking player
def score(game)
    if game.win?(@player)
        return 10
    elsif game.win?(@opponent)
        return -10
```

```
    else
        return 0
    end
end
```

Pseudocode for an empty game (first move):

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
```

To understand even further, Duchi, Wainwright & Zhang (2015) define the formal definition:

$$\underline{v_i} = \max_{a_i} \min_{a_{-i}} v_i(a_i, a_{-i})$$

Placement Functionality

The next component of the algorithm is the "Placement Functionality". It uses the data output from the "Pre-emptive Functionality" to make the next move, resulting in the maximum possible score for the player. Elaborating on this, the best move position is stored in a variable, in this case @choice, which is used to place the token in the optimal position.

"Placement Functionality" Pseudocode below:

(Note that in a connect four board, a column is simply a location or address on the board. Example: column [3] is the 4th column)

```
def minimax(game)
    return score(game) if game.over?
    scores = [] # an array of scores
    moves = []  # an array of moves

    # Populate the scores array, recursing as needed
    game.get_available_moves.each do |move|
        possible_game = game.get_new_state(move)
        scores.push minimax(possible_game)
        moves.push move
    end

    # Do the min or the max calculation
    if game.active_turn == @player
        # This is the max calculation
```

```
            max_score_index = scores.each_with_index.max[1]
            @choice = moves[max_score_index]
            return scores[max_score_index]
        else
            # This is the min calculation
            min_score_index = scores.each_with_index.min[1]
            @choice = moves[min_score_index]
            return scores[min_score_index]
        end
    end
```

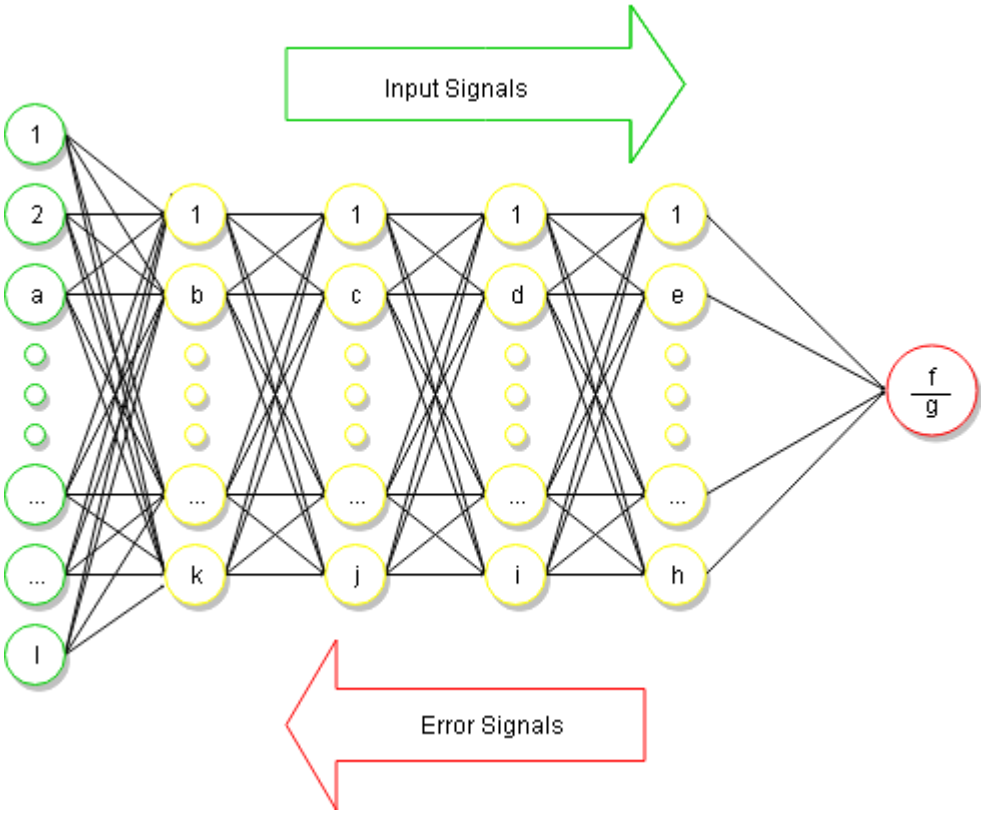
Improvements Didwania, R., Kumar, V., Maji, S., & Nasa, R. (2018)

An important nuance is in relation to how the bot plays if it knows it's going to lose. Should it prolong its defeat or should it finish the game as quickly as possible to get into the next one? This takes us into the "depth" of the algorithm. Depth is referred to as how far the algorithm "looks" down the tree. As mentioned earlier, a Connect Four board of dimensions 6x7 has 4 trillion possible board states. This means we cannot have unlimited depth because the computational time would be extremely high. To solve this, we have to specify a desired depth. Does the player have a limited amount of time to make a move? If so, the depth must be limited. Another interesting approach is using alpha-beta pruning. Didwania, Maji & Nasa (2018) all agree that A-b Pruning and minimax can be used in conjunction to create a more efficient algorithm by limiting the depth based on conditions, this could be elaborated on in a future report.

These are important questions to ask regarding improvements to the algorithm and how to truly make it unbeatable.

Before building an ANN, we first must determine if the problem is a Classification problem or a Regression problem. We start by setting up the board of the game so that each cell contains either 1, -1 or 0 if it contains, respectively, a token by the first player, or one by the second player, or none. The outcome of each position can then be computed to a win, loss, or draw, numerically represented by; 1, -1 or 0. Since the outcome of the ANN is to compute discrete values, it can be said to be a Classification problem.

ANN Structure



Given the high number of features (42 cell for the game board), it seems logical to assume the ANN would need to have multiple hidden layers to properly extract a coherent relationship. after some experimentation, 4 seemed to best balance the minimisation of the lost function and the computing time.

Overall representation of each layer:

- Each layer is represented as an nth dimension NumPy array. (ref NumPy documentation.)
- The neurons in each layer are initialised to zero.
- The weights are represented as a 2D NumPy array with the following shape:
    - (Right, Left), meaning the outer-most array would be the same length as the layer on the right.
    - This was done this way to simplify the backpropagation function which was costlier than the activation function.
    - Each weight was activated by using a random normal distribution:
        - equation

The first layer is the Input Signal:

- Each neuron represents each cell of a connect4 game. Meaning there are 42 neurons in the first layer.
- No processing is done at all. Meaning the input to each neuron is the same as its output.
- Programmatically, the layer is represented as a 2d NumPy Matrix, containing the input and the output to the neuron. It was done this way as it helped running the Activation function more uniformly over all layers.

The second to fifth layer are the Hidden Layers:

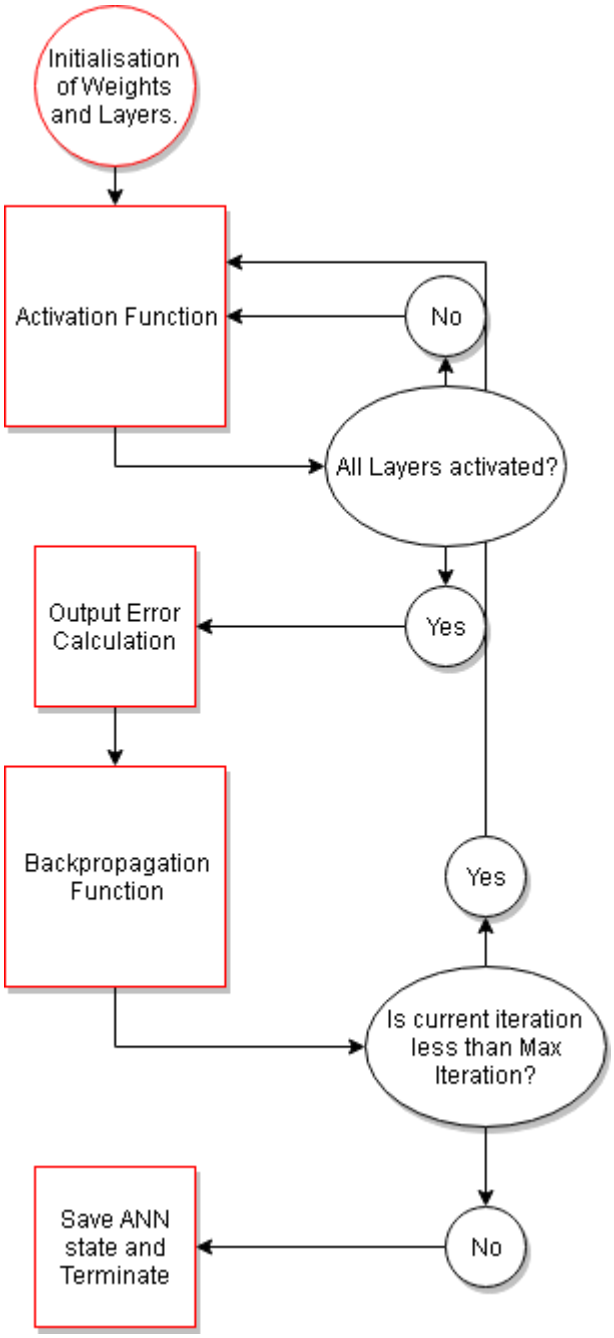
- They each have 24 neurons
- The input to each neuron is calculated using: $X = \sum_{i=0}^{i=j} y\,i * wki$ *eq.1*
    - X being the input of the neuron
    - i being the output of the i^{th} neuron of the previous layer
    - w_{ki} being the weight connecting the two neurons
- Their output is calculated using: $y = \tanh(X)$ *eq.2*

The sixth layer is the Output Layer:

- It consists of only one neuron
- The input and output are calculated using the same functions as the hidden layer.

ANN steps

As seen on the right, these are the main steps the algorithm goes through. After Initialisation, an array containing the features and the desired output is passed to the Activation algorithm. It's a recursive function that calls itself by passing the calculated output of a Layer that will act as input to the next layer. This is done automatically for all Layers. Once the output to the final layer is calculated, this is returned and passed to a simple function that returns the error between desired and actual output. The error is then passed to the back-propagation algorithm that calculates new weights. This algorithm is also recursive and calls itself by passing the error gradient to the next layer. It terminates by returning an array containing the new weights which are then used to replace the old weights. This process is repeated on the complete data set available and then the state of the ANN is saved. This includes the Structure and the weights.



Chronological Screenshots of tools used:

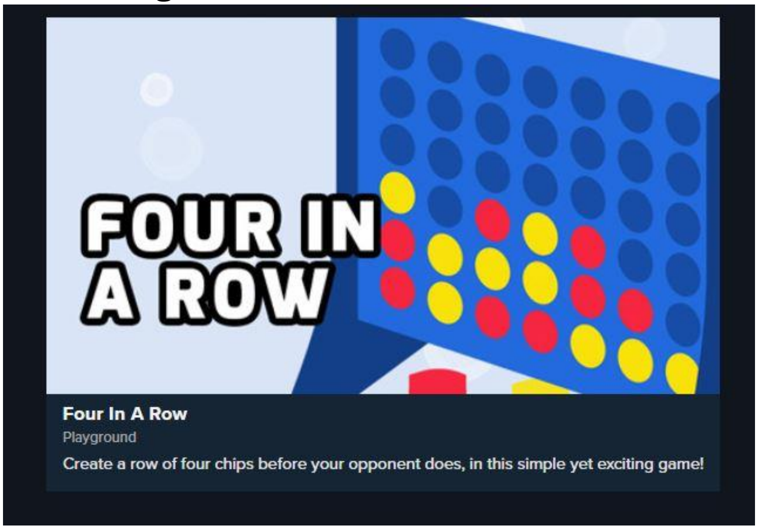


Figure 3 First starter bot downloaded from riddles.io

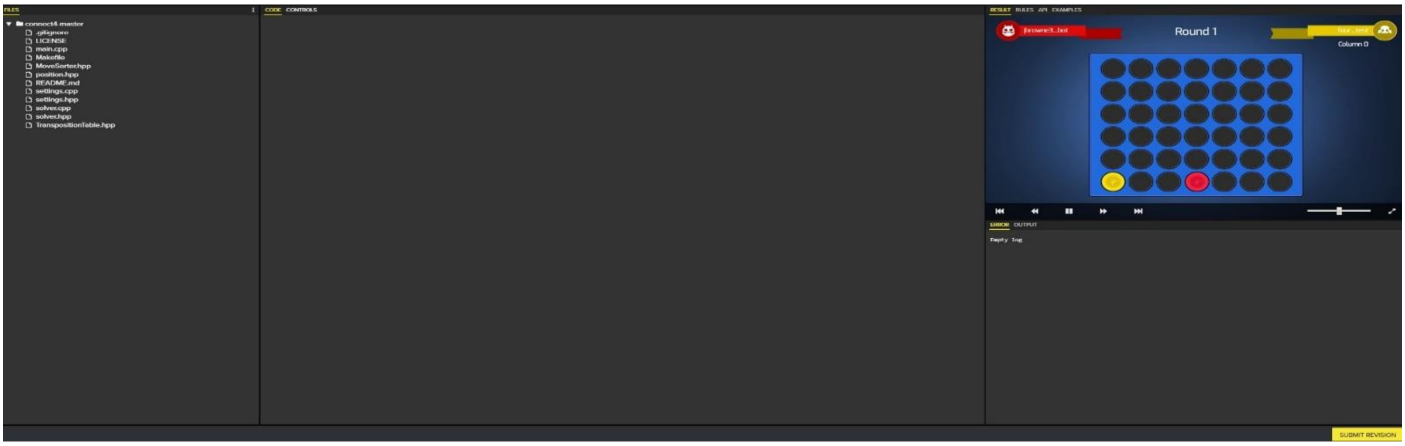


Figure 4 Implementation of minimax in riddles.io

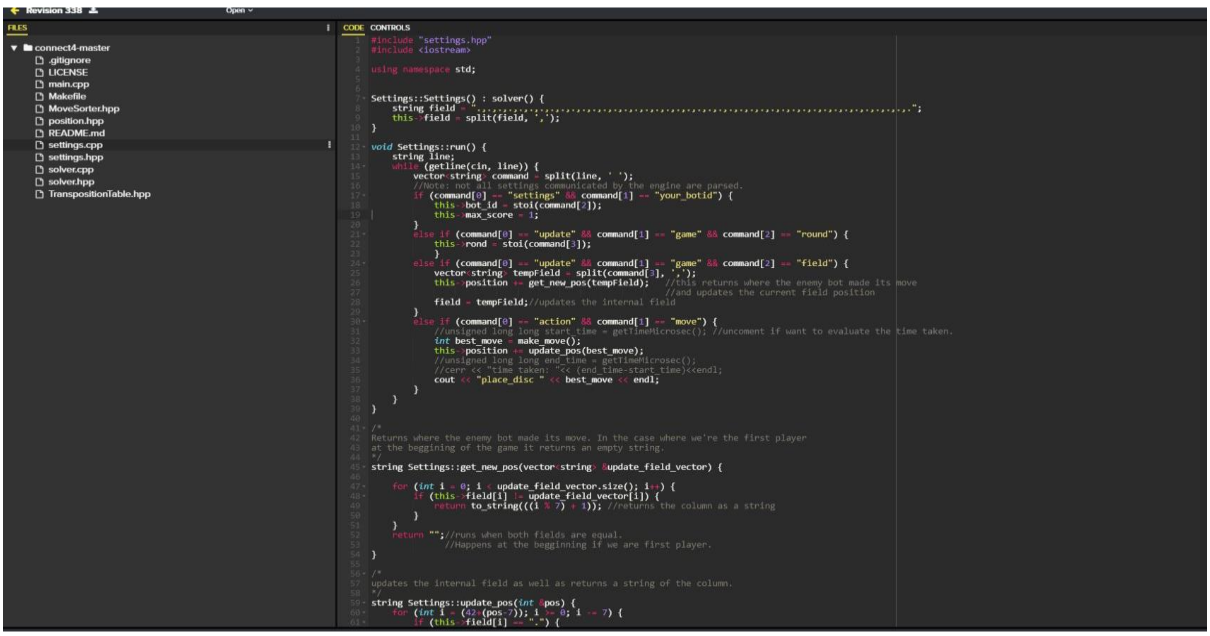


Figure 5 Sample of some of our minimax code

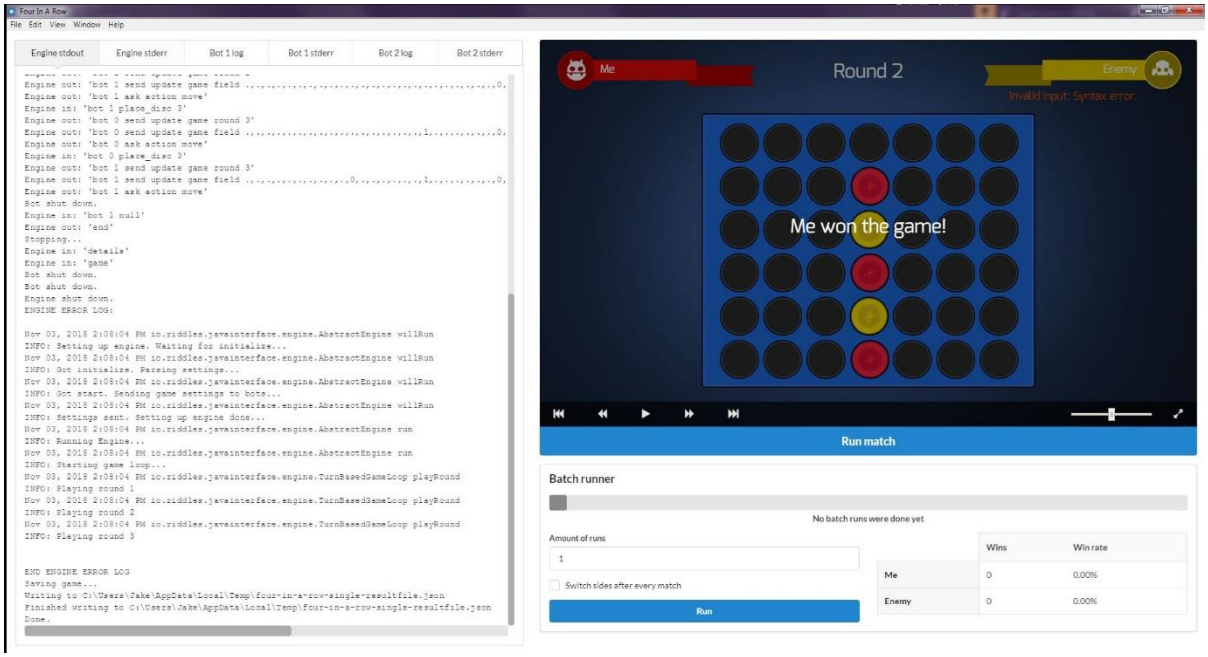


Figure 4 Exported minimax code into the Ai-Workspace

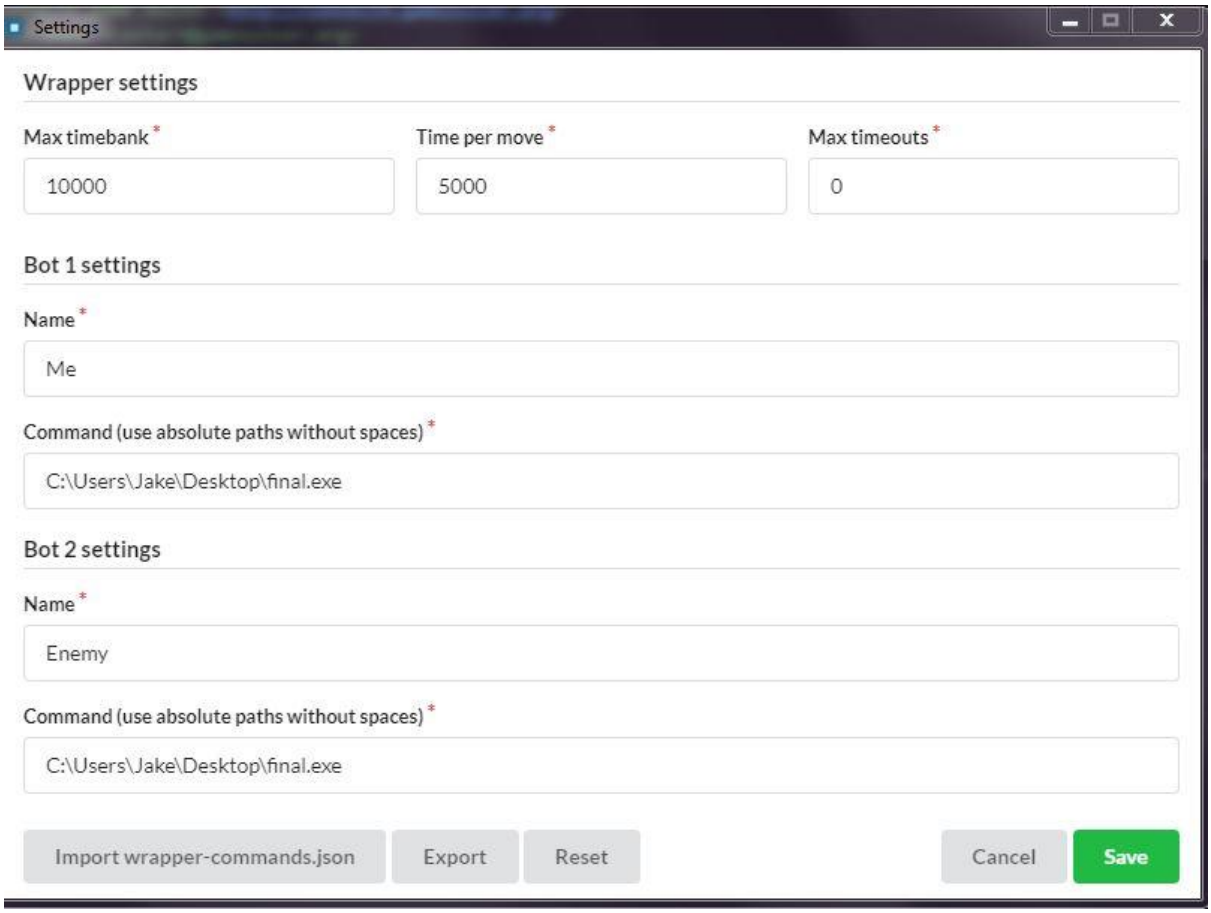


Figure 5 Uploading our bot to the AI workspace

(Note: As you seen from the image above, we can specify the "time per move". This will be mentioned further in the report in the comparative study section

Comparative study

The bots competed against each other on 3 sets of games; 10 games, 100 games & 1000 games. We also decided to use a standardised time each player has to make a move. Although it had no effect as the timer was in seconds not milliseconds. Across these results the average amount of wins was close to even with the Minimax algorithm having a slightly higher win rate in the 10 to 100 sets of games and the Artificial Neural Network having a slightly higher win rate at 1000 sets of games. This reflects that new data that the Artificial Neural Network has collected allowing it make more accurate decisions.

Conclusion

Summarise very clearly what was done and learned, mentions prospects and limitations and possible improvements.

In this report we designed & implemented two Intelligence systems, an artificial neural network & a minimax algorithm. The goal of these intelligence systems was simply to win a game of Connect Four. We had the systems play a set of games together to determine which one was the best. Theoretically the minimax algorithm should win every time if the artificial neural network takes too long to make a move as the algorithm has more time to calculate further down the tree in turn predicting the opponents move & reacting in the appropriate manner. For the artificial neural network to contend against minimax it would need to be trained in an extremely reliable & efficient way, however it may be difficult for this theory to come to fruition as there are too many variables at play such as time it takes for the systems to make a move, efficiency & the reliability of the training for the artificial neural network.

Contribution of Each Team Member:

Research Approach (Minimax): Jake Browne

Research Approach (Minimax) & Introduction: Christian Watson

Abstract, Introduction, Conclusion, Comparative Study & Research Approach(Translating, ANN & Minimax): Bryce Wilkinson

Conclusion & Research Approach (Artificial Neural Network): Daniel

References

Bruin, A., Pijls, W., Plaat, A., & Schaeffer, J. (2017). A Minimax Algorithm Better Than Alpha-beta? Retrieved from https://arxiv.org/abs/1702.03401

Duchi, J., Wainwright, M., & Zhang, Y. (2015). Divide and Conquer Kernel Ridge Regression: A Distributed Algorithm with Minimax Optimal Rates. *Department of Electrical Engineering and Computer Science University of California, Berkeley, Berkeley, USA.* Retrieved from: http://www.jmlr.org/papers/volume16/zhang15d/zhang15d.pdf

Didwania, R., Kumar, V., Maji, S., & Nasa, R. (2018). Alpha-Beta Pruning in Mini-Max Algorithm –An Optimized Approach for a Connect-4 Game. *Dept. of Computer Science & Engineering, The National Institute of Engineering, Mysuru, Karnataka, India.* Retrieved from https://irjet.net/archives/V5/i4/IRJET-V5I4366.pdf

Lucas, M, S., Perez-Liebana, D., Samothrakis, S., Schaul, T., & Togelius, J. (2016). General Video Game AI: Competition, Challenges, and Opportunities. *University of Essex Colchester. Retrieved from http://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/download/11853/12283*

Luo, H. & Schapire, E. R. (2014). Towards Minimax Online Learning with Unknown Time Horizon. *Department of Computer Science, Princeton University, Princeton*. Retrieved from http://proceedings.mlr.press/v32/luo14.pdf

Tromp, J (2014). John's Connect Four Playground [Blog Post]. Retrieved from: https://tromp.github.io/c4/c4.html

Willis, V(October, 2018). *A Knowledge-based Approach of Connect-Four*. Retrieved from: http://www.informatik.uni-trier.de/~fernau/DSL0607/Masterthesis-Viergewinnt.pdf