Lancaster University

An Investigation into the Energy Consumption Benefits of Including Domain Knowledge into AI Agents

Declaration

Name: Theo Fraser
Date: 22/03/24

Abstract

Energy consumption is a global issue that has massive effects on the world. Increased energy demand leads to more fossil fuel emissions which contribute to other global issues such as climate change. The rapid development of AI has led to them using more energy than ever before. This project is designed to evaluate the effectiveness of a series of AI ‘agents’ by measuring their performance against their energy consumption. We choose to evaluate the agents in the game of Connect4. The agents are all based of Monte Carlo Tree simulations which are described in later sections. Each agent is differentiated from another by having levels of domain knowledge implemented in various means. We find that the more domain knowledge an agent has the better its efficiency. The results of this project can hopefully be used for further AI development in the future.

Introduction

The power of computers has changed unimaginably in the last fifty years. There are many mind-boggling statistics like “computing power has increased a trillion times since the 1950s” or “a single Google search consumes about as much computing power as was used for all the Apollo space missions”. Computers and computing devices are now found everywhere and the energy they *consume* is becoming a concern.

Why is the energy consumed by computing of interest?

In 2022, the Parliamentary Office of Science and Technology published a report Energy Consumption of ICT that stated that information, communication, and technology (ICT) accounted for an estimated 4-6% of global electricity use in 2020.

Artificial Intelligence (AI) algorithms, programs that show “intelligent” behaviour, are becoming ever more common and widespread. Sometimes the user knows that they are interacting with AI, like OpenAI’s ChatGPT which was released in 2022 and made over 100 million users within 2 months. Other times the user may not be so aware, for example Google search, which receives over 5 billion visits per day, and has been upgraded to use generative AI to improve search results.

The benefits that AI programs deliver are large however, AI programs can require significant energy to run. In 2018, OpenAI found that the amount of computation power used to train the largest AI models had doubled every 3.4 months since 2021 (MIT Technology Review[7]). The New York Times[3] reports that by 2027 servers running artificial intelligence algorithms could alone consume 0.5% of the world's total electricity – up to 134 terawatt hours per year!

Project Aims

The aim of this project is to explore the relationship between energy consumption and performance (i.e. effectiveness) for different AI algorithms / learning techniques and to demonstrate how increased performance can be delivered for less energy by altering the way algorithms are implemented / applied.

The domain for this investigation is the well-known game of Connect4. Connect4 was chosen because it is familiar, easy to understand and relatable as well as being sufficiently non-trivial. However, the principles and learnings of this investigation can be applied more generally and to more complicated areas.

During the investigation, different AI algorithms will be evaluated against one another playing Connect4. The energy consumed by the algorithms will be measured against performance (i.e. how well they play the game) and compared.

Background

Today, there are many different types of artificial intelligence algorithms that use different approaches to solving a given problem. AI algorithms are often used for problems that are difficult to program or describe procedurally because they don't require specific instructions on how the program should solve the problem.

This property makes them very useful for problems where the desired outcome is understood, the rules or parameters are known, where there is often domain knowledge available but nevertheless it is difficult to describe exactly how to go about solving the problem. Games, such as Chess, Go or Connect4 are good examples. The object is simple, win the game, the rules are relatively easy to understand (and program), there is lots of domain knowledge available but programming or playing a game like chess is not easy.

Another example might be driving a car. Most people can do it but that doesn't mean that it is easy to program a computer to do it. In fact, it is a very difficult thing to do.
Knowledge-based algorithms, for example expert systems, require significant upfront investment in human resources to capture human knowledge about a domain in the system. Thereafter, the system requires relatively few resources to run but without adding more knowledge it doesn't get any better and can't adapt to a changing problem space.
Some AI algorithms need no previous "knowledge" of the problem space or any training to operate effectively within it. A flexible and powerful example of such an algorithm is the basic Monte Carlo Tree Simulation (MCTS) algorithm which is described in the next section and is central to this paper.

Genetic Algorithms (see below subsection) are different again, like the MCTS algorithm they do not have to be programmed to solve a problem however, unlike the MCTS they require lots of upfront training before they become useful.

The game Connect4, which is described below, has been chosen to demonstrate how the performance of AI algorithms can be enhanced by the inclusion of domain knowledge, at the same time as reducing the energy consumption.

Monte Carlo Tree Search Introduction

Monte Carlo Tree Search (MCTS) is a search algorithm that uses random playouts to "derive" the best option – in this case the best Connect4 move. MCTS is used a lot for games, like Go and chess. It has the advantage that it doesn't need any prior knowledge or training making it extremely easy to implement however, it can use a lot of memory and CPU time to produce effective results. Browne and other provide a comprehensive overview of MCTS methods in their work cited as [10].

MCTS is comprised of four main stages: selection, expansion, simulation, and backpropagation. Selection begins at the root node of a search tree where the algorithm will choose the most promising child node based on a selection strategy like Upper Confidence Bound (UCB). Figure 1 shows how to calculate the UCB value.

$$\frac{w_i}{s_i} + c\sqrt{\frac{\ln s_p}{s_i}}$$

- w_i : this node's number of simulations that resulted in a win
- s_i : this node's total number of simulations
- s_p : parent node's total number of simulations
- c : exploration parameter

Figure 1

The selection strategy performs a balance between exploration (trying less visited nodes) and exploitation (selecting nodes with high estimated value). Expansion occurs when a leaf node (a node with unexplored children) is selected. The algorithms expand it by adding one or more child nodes to the leaf node which represent possible feature states or moves.

Simulation (Rollout) happens when a node is selected and has no visits. This involves playing out the game from that node until reaching a terminal state. The simulation is typically done randomly or by using some heuristic strategy if applicable.

Backpropagation is performed after the simulation is complete. During backpropagation the result (say, win, loss, or draw) is backpropagated up the tree to update the values of all the nodes visited during the selection and expansion phases. The values for each node typically include the number of visits and the total accumulated rewards. In figure 2 it shows a detailed image of how the MCTS works.

In summary, MCTS iteratively selects promising nodes, expands the search space, simulates potential outcomes, and updates node values based on the results, ultimately guiding decision-making in complex environments.

In its pure form, without any domain knowledge, MCTS is required to start from scratch each time it is run i.e. it doesn't start with any knowledge advantage or learn or get better over time. Its performance is largely determined by the available resources.
In this project, the MCTS algorithm can be throttled by controlling the resources available to it. The unit of resource is the number of back propagations that is available to the algorithm for each move.

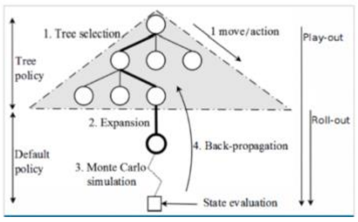


Figure 2 [pick]

Genetic Algorithm Introduction

Genetic Algorithms (GA) mimic, in code, the process of natural selection to "evolve" an effective solution. Solutions are evolved over generations by combining the fittest (i.e. best) solutions together and adding in elements of mutation and crossover.

The genetic algorithm starts by generating a random population of chromosome that represent the characteristics to be evolved. Then the algorithm evaluates the relative fitness of each member of the population by performing an evaluation function e.g. playing a game of Connect4.

The algorithm creates a new population selecting two parent chromosomes based on their fitness using a selecting strategy. Once the chromosomes have been selected there is a chance for the chromosomes to do a cross over this chance is dictated by the cross over probability if the cross over is not performed the chromosomes are still and exact match of the parents.
Once the cross over occurs there is now a random chance that the chromosome mutates which is affected but the mutation probability.

After the mutation we insert the new chromosomes in a new population. We do this until the new population is the size of the old population. Then the algorithm evaluates the fitness of the new population and does this all again until a certain number of generations. A literature review of a genetic algorithm is covered by Lambora and others [11]. In figure 3 It shows a very simple image of how a genetic algorithm works.

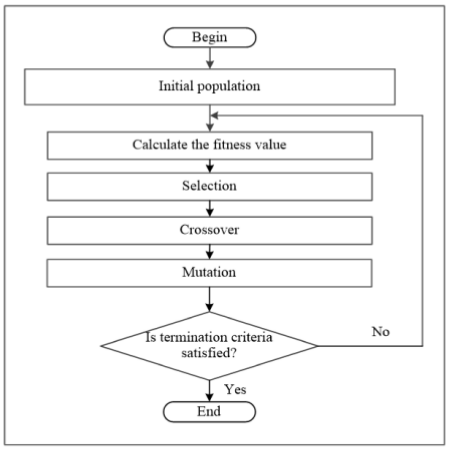

Begin
Initial population
Calculate the fitness value
Selection
Crossover
Mutation
Is termination criteria satisfied?
No
Yes
End

Figure 3

The natural selection process of a GA can take significant time and consume many resources however, this process needs only to be performed one time. Once the selection process has completed, usually by evolving and comparing solutions over many generations, the improvements developed by the process have become embedded in the solution making them available every time the solution is run.

Connect4 Basic Analysis

An analysis of the problem space, in this case Connect4, is an obvious first step. Connect4 was invented by Howard Wexler and Ned Strongin and was first sold to the public as a game in 1974. It was solved in 1988 independently by James Dow Allen and Victor Allis. The solution algorithm is procedural and can be applied from any board position however, it is not easy for a human brain to execute. Nevertheless, the outcome of any game can be determined, using a piece of paper or a computer implementation of the solution from any position including the starting position.

The solution for Connect4 is such that Player1 (P1) can always win if playing optimally. Playing optimally for P1 requires that the middle column is played first i.e. d1 in figure 4. If P1 does not play d1 first then Player2 (P2), if playing optimally, can always force a draw on the 42nd move. Player 1 will lose never if playing optimally even if it did not play position d1 on the first move.

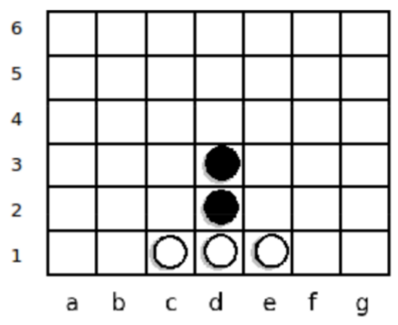


Figure 4

The first 5 optimal moves with white as P1 is shown in figure 4.

Note that it is not the purpose of this investigation to programmatically implement the solved solution and it is not significant that the game has a solution. The author's personal experience of Connect4 was gained whist playing as a child and my skill never went beyond the basic tactics and playing mostly "reactively".

Connect4 was chosen because it is well-known and has well understood rules and is very intuitive. However, the game is not trivial in that there are many possible board combinations (several trillion) which makes a brute force approach to solving infeasible. It is also easy to measure the effectiveness (performance) of the algorithms discussed in this paper simply by playing them against one another and determining the win rate. This is an important characteristic of the problem domain because it allows quantitative performance information to be collected. However, this does not mean that the approach and any findings cannot be applied to problem domains where there isn't always a "correct" answer e.g. driving a car or calculating the weather.

Another important aspect of the game is that, like in many domains, there exists information that helps you play better. Some of this information may be simple and often obvious do's and don'ts. For example:

- Do block an opponent's win where possible.
- Don't set up a win for opponent on your turn.

There are also more subtle tactics that are not so black and white such as:
- Try and set up a "fork".
- Try to block opponent "threats".

These do's and don'ts and tactics represent Connect4 domain knowledge that is readily available in the public domain – and whilst the do's and don'ts may be obvious to humans and easy to program, tactics are more difficult to implement due to their subtle and non-specific nature. Most problem spaces have some amount of domain knowledge but, that does not necessarily mean that it can be easily turned into program code.

Related Work

Modifying the basic MCTS algorithm is not a new idea. Several studies have demonstrated benefits of integrating other techniques into the basic algorithm.

Yang, Z [12] integrated domain knowledge into the basic MCTS algorithm and applied it to Real-Time Strategy (RTS) games. The results showed that their learned model that applied bias both the tree node selection rollout operations performed best. This showed that MCTS improves with the application of domain knowledge. This arrangement of applying domain knowledge to tree node selection and playout is investigated in this document from an energy consumption point of view.

Kim and Ahn[13] proposed a model that combined Genetic Algorithms with MCTS. Their hybrid approach allows for efficient problem-solving in real-time scenarios i.e. using GA to refine MCTS processing in real time. In this project GA is used to pre-process domain knowledge which is then included into the modified MCTS algorithm.

Chen, K.H. and Zhang[14] identified the importance of using domain knowledge with MCTS to effectively play the game GO with increased performance.

There are other examples of modifying the basic MCTS algorithm including the following:

Benbassat, A. and Sipper, M[8] uses Genetic Algorithm techniques to improve the MCTS by evolving board evaluation functions for playouts. They domain for their work was the game Reversi.

Macolino[15] used a system where different agents play in the simulation phase of UCB Monte Carlo Go which increase the quality of the simulations. They could beat the original system they were trying to overcome which was Fuego.

Chaslot and others[16] developed a program called MoGo that recently won a game against the 9 Dan Pro Go player, Zhou ZunXun. MoGo. They improved the MCTS simulations by including specific domain knowledge into the algorithm.

Methodology

The project will devise and develop different agents / algorithms to operate within the domain i.e. playing Connect4. To evaluate the strengths and weaknesses of the different agents, a series of tournaments will be run. Each tournament consists of playing games between different agents and/or agents with different parameters (e.g. available resources).

The number of games in each tournament is required to be sufficiently high that the results are statistically significant. Through trial and error, the figure of 1000 games per tournament was found to provide statistically significant results without being overly high / excessive.

For each tournament various data will be collected as follows:

- Number of games (usually 1000)
- Player1 & 2 agent
- Player1 & 2 win %
- Player1 & 2 average moves per game
- Player1 & 2 average time per game (micro secs)
- Player1 & 2 total energy consumed (mJ)
- Number of draws

When we come to evaluate the performance of the agents, we will do so against a baseline agent with fixed resources playing as the advantaged Player1 (P1).
For different Player2 (P2) agents, we will vary the number of resources available, measuring the performance as we do so. This approach will allow us to determine the relative performance of the agents and enable us to rank them.

There are many candidates for AI agents that could be applied to this problem. Different algorithms have distinct characteristics that will influence not only how well they play but how well they play given restricted / limited resources, how much energy they consume and whether pre-training – which also carries a one-off energy cost - is required. Also, different algorithms also require varying effort and knowledge to implement. A summary of the agents in scope for this project are discussed in the following sections.

Random

The Random agent simply picks at random, a move from the set of available valid moves. There is no inherent domain knowledge or in-built intelligence. This agent was developed primarily to help with testing the game framework and data collection functions however, it is also useful for demonstrating the P1 advantage bias discussed above.
This advantage bias can be demonstrated by playing 2 Random agents against one another.

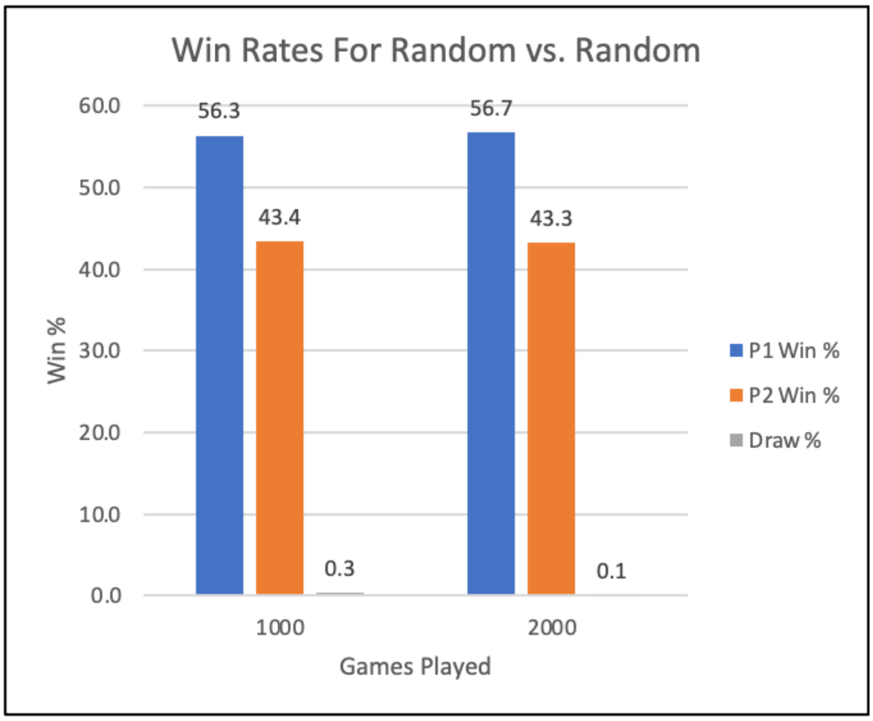


Figure 5

The graph in figure 5 shows that there is roughly a 13% bias to Player1 when both agents operate randomly. The point at which Player2 beats Player1 i.e. wins more than 50% of the games will be a significant measurement in the below results.

There are very few draws. The average number of P2 moves across all games was 10.3 (out of a maximum of 21). The results are quite consistent between 1000 and 2000 game tournaments. 1000 game tournaments are used for the remainder of the data in this document.
The Random agents consume very little energy, too little for the total consumption to register when playing 3000 games. The average time to make a move was << 1 microsecond.

Naïve

The Naïve agent implements a very simple algorithm based on four pieces of "domain knowledge" – like a very simple expert system. The domain knowledge used is very basic, readily available, and easy to understand. The knowledge takes the form of absolute do's and don'ts and so can be thought of as "rules" for the Naïve player.

The Naïve agent applies four very basic rules, in a sequence, which are easy to evaluate and implement. The agent acts on the first rule in the sequence that applies. If no rules apply, a move is selected at random from the list of available moves. The only "intelligence" in this agent is contained in the logic of the rules and the order in which they are applied.

Naive Rules:

1) Take a winning move if one is available (offensive).
2) Block opponent winning on next turn (defensive).
3) Don't create a winning move for opponent on next turn (defensive).
4) Play bottom centre (d1) if available (offensive).

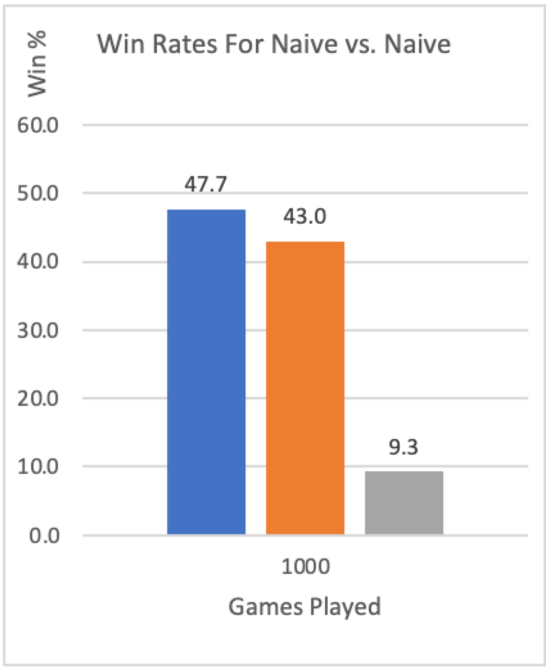


Figure 6a

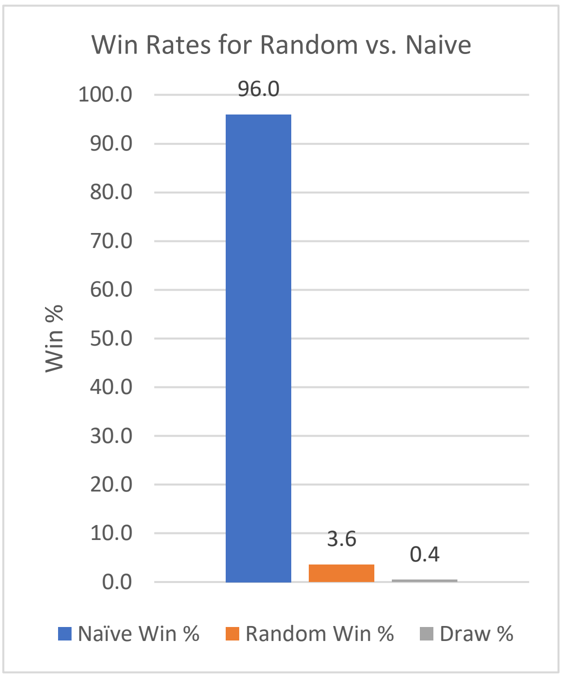


Figure 6b

Playing the Naïve agent against itself shown by figure 6a sees the Player1 advantage drop to just under 4% and the draw rate increase significantly indicating that the defensive logic rules are effective at defending a loss at least some of the time. Despite being crude and having little by way of intelligence, when Naïve is played against Random shown by figure 6b it achieves a 96% win-rate even as the disadvantaged Player2 (and 97% as Player1).

The Naïve agent consume very little energy, too little for the total consumption to register when playing 1000 games. The average time to make a move was 140micro seconds.

Determining Monte Carlo Tree Search Baseline

In figure 7 the chart demonstrates how the MCTS (P2) win % increases with available resources. It is not until the MCTS agent has 60 back propagations available to it that it wins more than 50% of the games against the Naïve agent which has the P1 advantage.

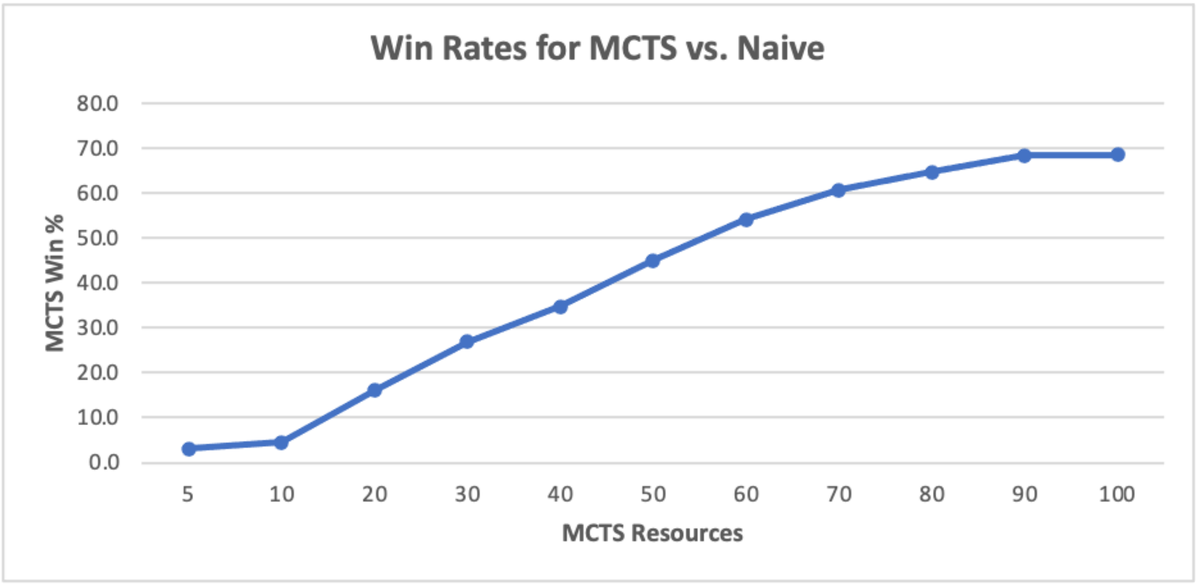


Figure 7a

The effectiveness of the throttling and the associated cost in terms of energy consumption (right y-axis) is shown in figure 8 where the win % (left y-axis) of a Player2 MCTS agent with varying resources (x-axis) plays against the Naïve agent.

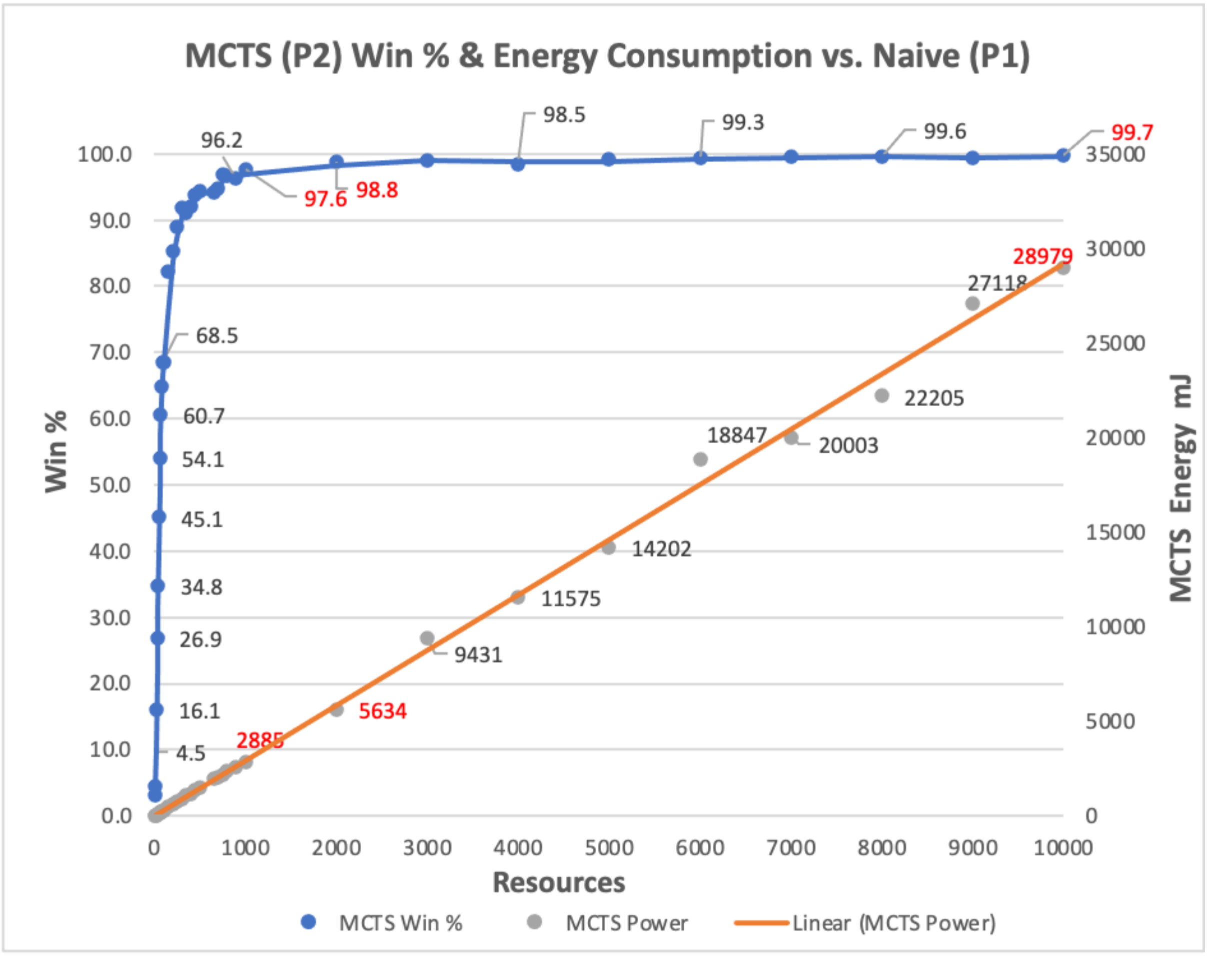


Figure 7b

The MCTS performance (the blue line) rises sharply with resources. The orange line shows how much energy is used for different resources.

With 1000 resources (back propagations) MCTS (P2) wins 97.6% and uses 2885 mJoules (mJ). Doubling the resources to 2000 gives only a 1.2% increase in performance (98.8%) but consumes almost twice as much energy, 5634mJ. Giving ten times the resources, 10000 back propagations, raises the performance by just over 2% (99.7%) but uses ten times as much energy 28979mJ.

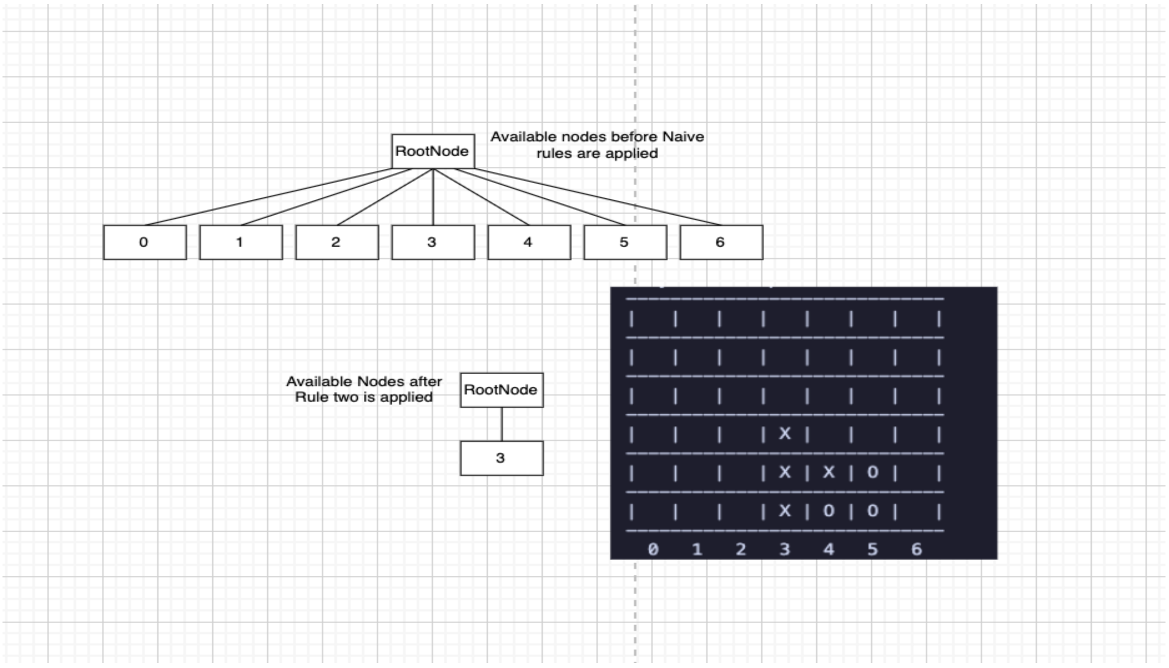
The shape of the graph clearly shows that the gains, which increase greatly to begin with, level off sharply around 1000. An MCTS agent playing with 1000 resources (MCTS:1000) is reasonably fast, it performs moves in approximately 0.2 seconds, and good at playing Connect4 against a human. MCTS:1000 is also good value in terms of the number of resources it consumes.
For these reasons a Player1 basic MCTS with 1000 resources MCTS:1000(P1) is used as the benchmark to beat going forward i.e. can we devise agents that perform better and with fewer resources, as Player2, than MCTS:1000(P1)?

Monte Carlo Tree Search + Naïve

It is possible to include domain knowledge into an MCTS implementation to try and improve its performance given fixed resources; this project explores the effectiveness of different approaches of embedding domain knowledge into MCTS.

The MCTS + Naïve agent implements the four pieces of domain knowledge from the Naïve agent into the MCTS algorithm. The idea is to "prune out" tree branches that represent bad moves according to the Naïve rules; this helps ensure that MCTS resources are not consumed investigating "bad" moves are instead focussed on "good" moves. In this algorithm, the Naïve rules are only applied during the process of determining available children and are not applied during the playout process (this variation in considered later).

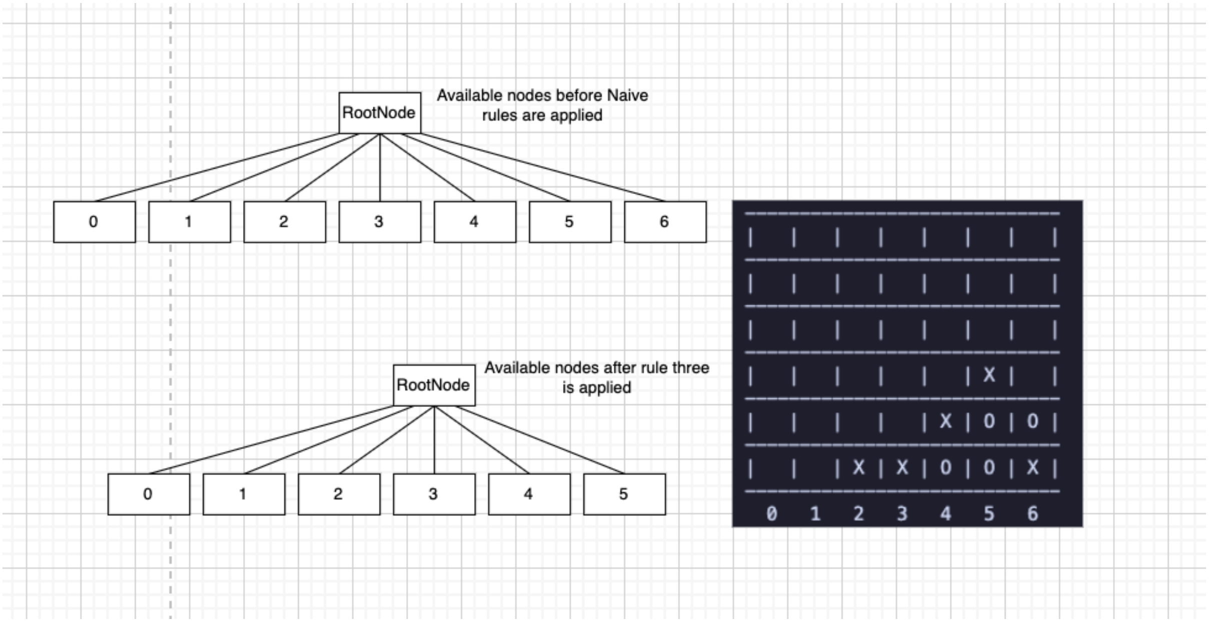
For example, if all moves (0-6) are available to the agent but, Naive rule 2 (Block opponent winning on next turn) can be satisfied by playing move 3 then all moves other than 3 will be pruned. This can be shown by figure 9.


RootNode
Available nodes before Naive rules are applied
0
1
2
3
4
5
6
Available Nodes after Rule two is applied
RootNode
3

```
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   | X |   |   |   |
-----------------------------
|   |   |   | X | X | O |   |
-----------------------------
|   |   |   | X | O | O |   |
-----------------------------
  0   1   2   3   4   5   6
```

Figure 9

Similarly, if all moves (0-6) are available to a player but, playing 6 means Naive rule 3 (Don't create a winning move for opponent on next turn) is broken then, move 6 will be removed from the list of available moves and node selection will be performed from the remaining nodes based on standard UCB evaluation. This can be shown by figure 10.

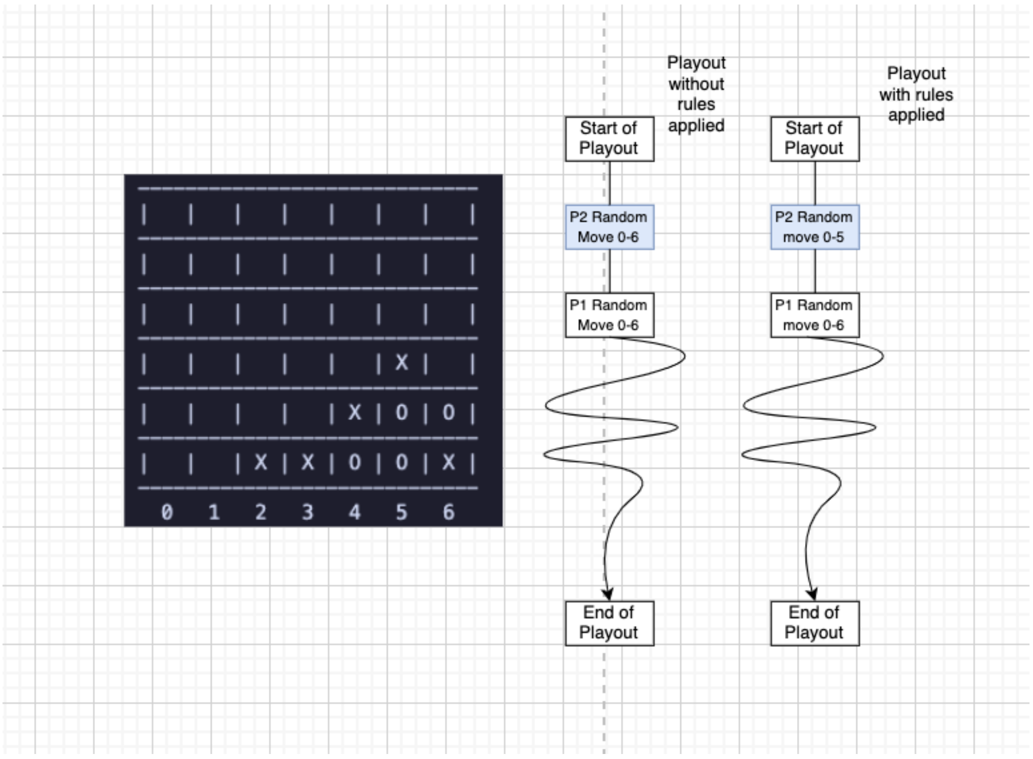

RootNode
Available nodes before Naive rules are applied
0
1
2
3
4
5
6
RootNode
Available nodes after rule three is applied
0
1
2
3
4
5

```
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   | X |   |
-----------------------------
|   |   |   |   | X | O | O |
-----------------------------
|   |   | X | X | O | O | X |
-----------------------------
  0   1   2   3   4   5   6
```

Figure 10

Monte Carlo Tree Search + Naïve Playout

This agent is like the previous one except that the Naïve rules are applied for each move in the MCTS random playout. This can have the effect of significantly reducing the number of moves in a playout and making the playouts "more realistic".
This is demonstrated in the diagram below where the move 6 becomes unavailable to Player2 during the playout.



```
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   | X |   |
-----------------------------
|   |   |   |   | X | O | O |
-----------------------------
|   |   | X | X | O | O | X |
-----------------------------
  0   1   2   3   4   5   6
```

However, although the cost of evaluating the Naïve rules is relatively low, this approach requires that the Naïve rules are evaluated many thousands of times consuming energy as it does so.

Genetic Algorithm Derived Strategies

In this implementation, a GA is used to refine the relative importance or significance of several common Connect4 "strategies" representing domain knowledge. This is achieved by evolving weights that represent the proportional importance of the strategies over the generations. The weighted strategies are then incorporated into an MCTS implementation to speed up and improve / focus the node selection process.

Strategies are different from the Naïve rules in that they are not definitive do's or don'ts and so are not easily or intuitively implemented. Instead, the strategies represent guidance for playing one move over another. The importance or weights of different strategies can "stack" meaning that more than one can apply to a particular move in each situation. Also, unlike the Naive rules, there is no sequence applied.

Strategies:
1) Place on top of opponent (defensive move)
2) Block opponents two in a row (defensive move)

3) Create three in a row (offensive move)
4) Favour middle column (offensive move)
5) Create two in a row (offensive move)

In general, it is generally regarded that aggressive play is best for Connect4 and as such we might expect the offensive strategies to generally have higher weights than the defensive strategies. The above strategies are all have a positive influence and therefore one might expect the associated weights to be positive. Strategy weightings are only applied in the algorithm if there is more than one allowed move after processing the Naïve rules.

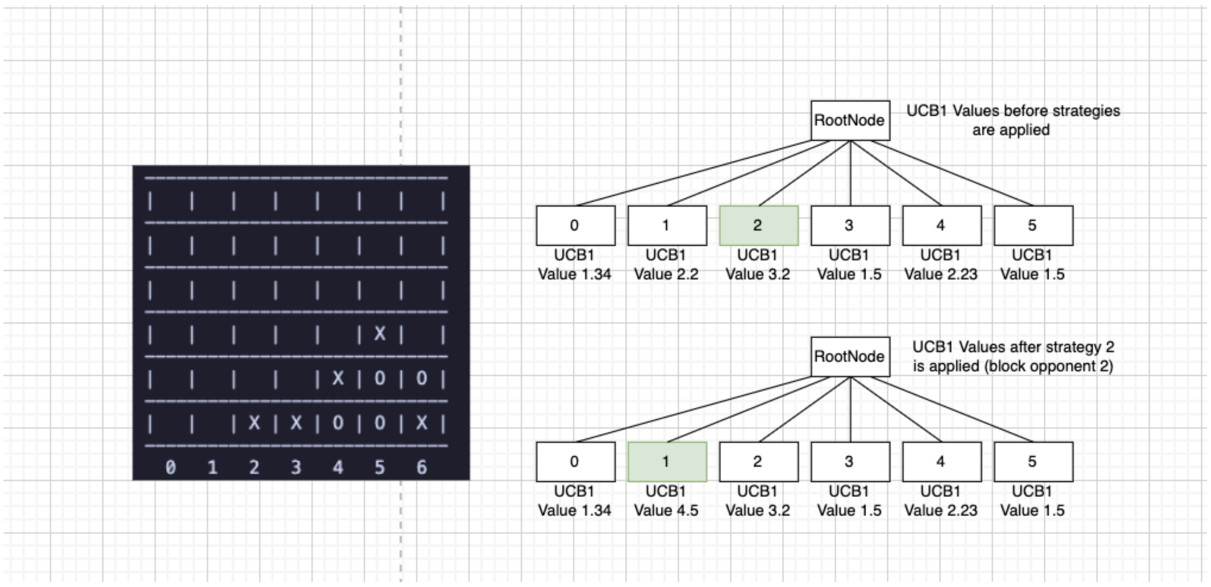
Monte Carlo Tree Search + Naïve + GA Strategies

This algorithm enhances the MCTS + Naïve implementation by additionally including weighted strategies where the weights were derived by a Genetic Algorithm.

The operation of MCTS is changed so that the exploration component of the UCB function is modified according to the GA weights on the applicable strategies. The exploration constant of the exploration component is adjusted incremented according to an adjustment function f that normalises the applicable strategy weights such that -1 <= f <= 1.

The adjustment function is intended to, in some cases, subtly influence the initial search path of MCTS. If the direction indicated by the application of the GA strategy weights is correct, then the resources available to MCTS should quickly confirm this and increase the UCB value further.

However, if the move identified by strategies is not applicable to play in a given position, then the MCTS algorithm will adjust (though at a resource cost) the UCB value to quickly overwhelm the influence of the incorrect GA strategy function.



```
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   | X |   |
-----------------------------
|   |   |   |   | X | O | O |
-----------------------------
|   |   | X | X | O | O | X |
-----------------------------
  0   1   2   3   4   5   6
```

Implementation

Technical Environment

Two languages were considered for this project, Java and Python. Java was selected as the best candidate because of the perceived speed benefits over Python – an important factor given the large amount of processing that would be required to play many hundreds of thousands of Connect4 games.

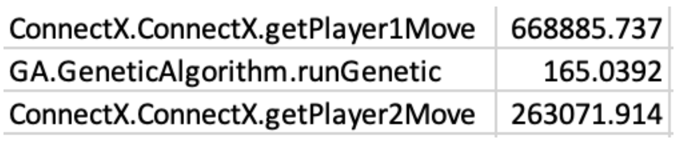
The Java version used was Java 17.0.2.jdk.

Visual Studio was chosen as the interactive development environment (IDE).

A key aspect of this project is to measure the actual energy consumption, at method level in many cases, of different algorithms. It is therefore important to have a consistent execution environment which was my Mac Pro Version 13.3.1. with a 2.4 GHz Quad-Core Intel Core i5 CPU.

To enable programs to be run over long periods, often overnight, the Max Pro was configured to consistently operate at full power with all sleep functions disabled.

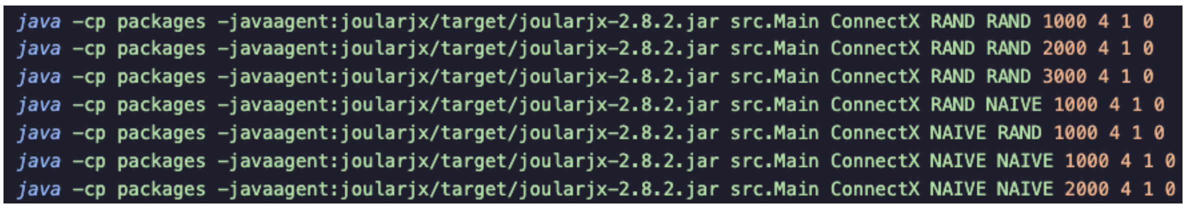
A java-based agent (JoularJX) was used to measure the energy consumption of the CPU. This agent produces log-files that record, down to the method level, the actual energy consumed by the CPU in mJoules. JoulaxJX can produce result log-files for all methods executed or else be filtered to track specific methods. An example log-file is shown below:



ConnectX.ConnectX.getPlayer1Move	668885.737
GA.GeneticAlgorithm.runGenetic	165.0392
ConnectX.ConnectX.getPlayer2Move	263071.914

Due to the repetitive nature of the tournaments and the large number of games to be played a framework was required that allows multiple, parametrised games to scheduled and executed unsupervised.

Also, to ensure that the Joular agent recorded energy statistics correctly in individual log files for each tournament, each tournament was required to be executed as a single java command. Unix BASH files were used to control and schedule the execution of multiple tournaments.

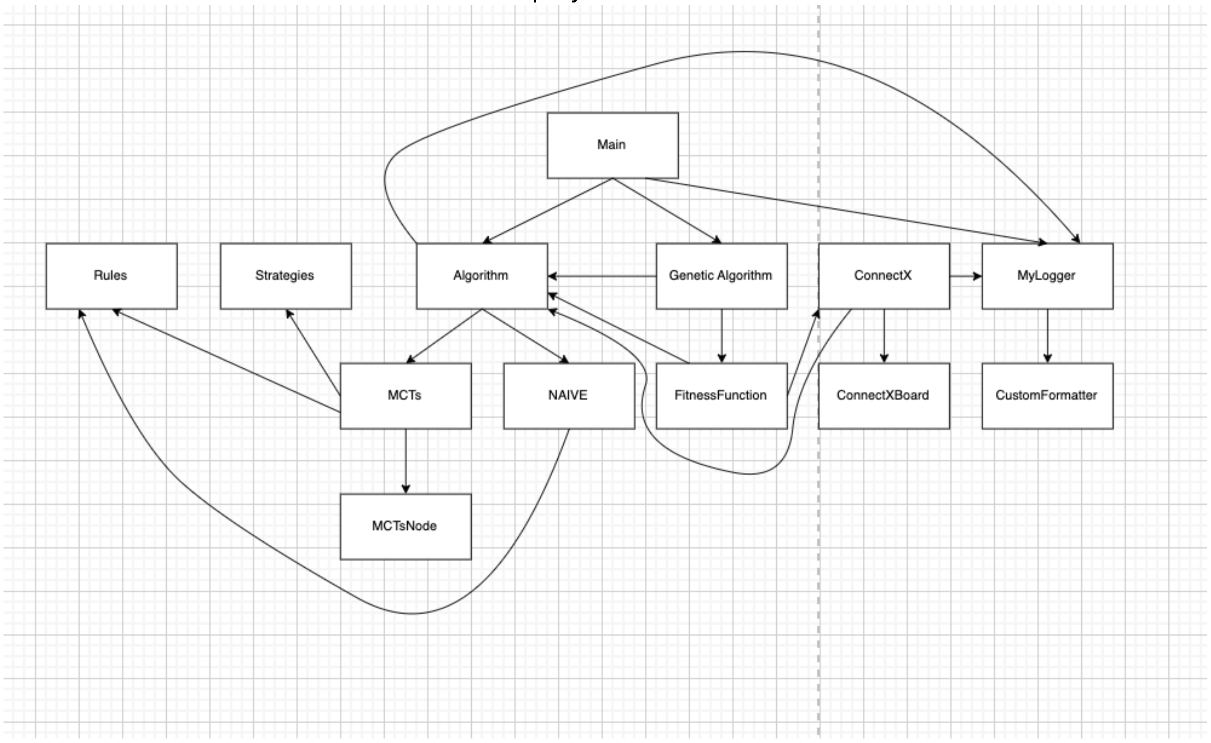


```
java -cp packages -javaagent:joularjx/target/joularjx-2.8.2.jar src.Main ConnectX RAND RAND 1000 4 1 0
java -cp packages -javaagent:joularjx/target/joularjx-2.8.2.jar src.Main ConnectX RAND RAND 2000 4 1 0
java -cp packages -javaagent:joularjx/target/joularjx-2.8.2.jar src.Main ConnectX RAND RAND 3000 4 1 0
java -cp packages -javaagent:joularjx/target/joularjx-2.8.2.jar src.Main ConnectX RAND NAIVE 1000 4 1 0
java -cp packages -javaagent:joularjx/target/joularjx-2.8.2.jar src.Main ConnectX NAIVE RAND 1000 4 1 0
java -cp packages -javaagent:joularjx/target/joularjx-2.8.2.jar src.Main ConnectX NAIVE NAIVE 1000 4 1 0
java -cp packages -javaagent:joularjx/target/joularjx-2.8.2.jar src.Main ConnectX NAIVE NAIVE 2000 4 1 0
```

Microsoft Excel was used to support the data analysis and visualisation of the logfiles of the results of each tournament.

Class Model

An overview of the class model used for the project is shown below.



Main
The main class parses user arguments to determine whether to set up Connect X or the Genetic Algorithm (GA).
Rule
The Rule class applies game rules to the current board state and returns a list of valid moves.
Strategies
The Strategies class utilizes the current board state to assess applicable strategies and assigns weights to potential moves accordingly.
Algorithm
The Algorithm class provides access to various algorithms.
MCTS
The MCTS class employs the Monte Carlo Tree Search algorithm to identify the best move at a given board state.
MCTSNode
The MCTSNode class stores data for nodes in the MCTS algorithm, such as wins and visits.
Naive
The Naïve class utilizes a simple algorithm to select a move.
GeneticAlgorithm
The GeneticAlgorithm class executes the genetic algorithm based on user-provided arguments, such as population size and number of generations.
FitnessFunction
The FitnessFunction class calculates the fitness of a selected member in the population by simulating Connect X games.
ConnectX
The ConnectX class manages the Connect4 game. It is parametrised to support boards of different sizes e.g. Connect5.
ConnectXBoard

The ConnectXBoard class handles the creation of a Connect X board and verifies if a move is legal (e.g., within the grid boundaries) and checks for a win or a full grid.

MyLogger

The MyLogger class manages the creation and modification of log files, utilizing the format specified in the CustomFormatter class.

CustomFormatter

The CustomFormatter class defines the format for writing to log files.

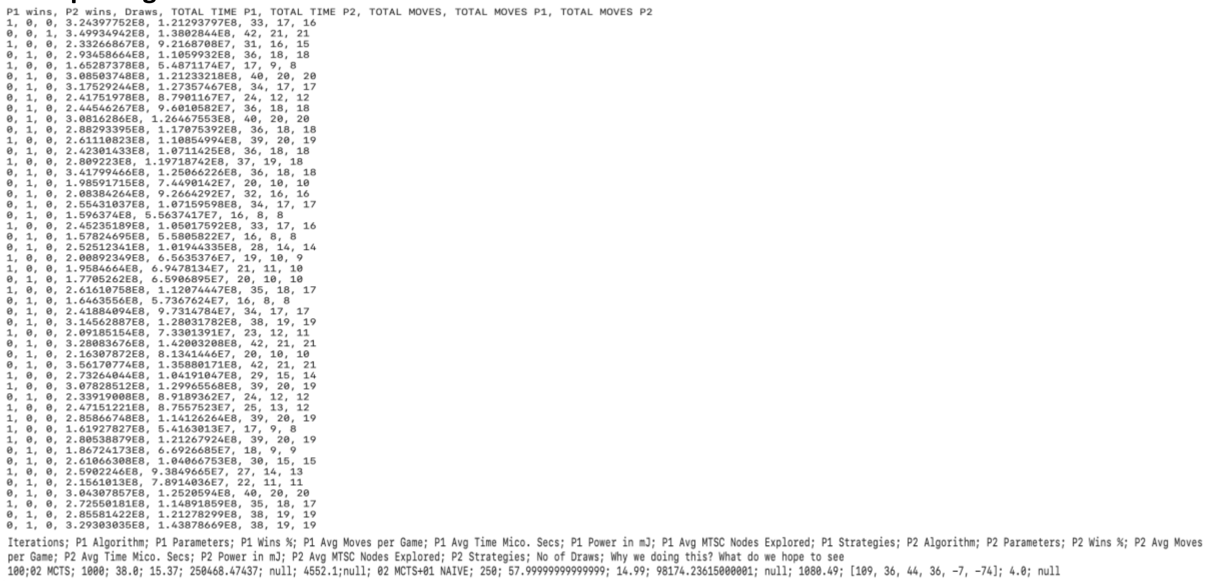
Program Framework

The program framework allows different agents/algorithms to play against one another in tournaments of multiple games. The agents can be computer programs or humans. The framework enforces rules of Connect4, prohibiting illegal moves and stopping when the game ends.

The framework also allows for multiple tournaments between non-human agents / players to be scheduled and played unsupervised via batch file execution. After each game in a tournament the framework records various metrics for subsequent analysis in a log file:

- the outcome, when a terminal state is reached: P1 win, P2 win or draw (after 42 moves).
- The number of moves is played.
- The actual time P1 and P2 agents took to run.
- The energy consumption of the P1 and P2 agents.
- Agent specific measurements (e.g. MTSC tree nodes explored).

Example Log File



```
P1 wins, P2 wins, Draws, TOTAL TIME P1, TOTAL TIME P2, TOTAL MOVES, TOTAL MOVES P1, TOTAL MOVES P2
1, 0, 0, 3.24397752E8, 1.21293797E8, 33, 17, 16
0, 0, 1, 3.49934942E8, 1.3802844E8, 42, 21, 21
1, 0, 0, 2.33266867E8, 9.2168708E7, 31, 16, 15
0, 1, 0, 2.93458664E8, 1.1059932E8, 36, 18, 18
1, 0, 0, 1.65287378E8, 5.4871174E7, 17, 9, 8
0, 1, 0, 3.08503748E8, 1.21233218E8, 40, 20, 20
0, 1, 0, 3.17529244E8, 1.27357467E8, 34, 17, 17
0, 1, 0, 2.41751978E8, 8.7901167E7, 24, 12, 12
0, 1, 0, 2.44546267E8, 9.6010582E7, 36, 18, 18
0, 1, 0, 3.0816286E8, 1.26467553E8, 40, 20, 20
0, 1, 0, 2.88293395E8, 1.17075392E8, 36, 18, 18
1, 0, 0, 2.61110823E8, 1.10854994E8, 39, 20, 19
0, 1, 0, 2.42301433E8, 1.0711425E8, 36, 18, 18
1, 0, 0, 2.809223E8, 1.19718742E8, 37, 19, 18
0, 1, 0, 3.41799466E8, 1.25066226E8, 36, 18, 18
0, 1, 0, 1.98591715E8, 7.4490142E7, 20, 10, 10
0, 1, 0, 2.08384264E8, 9.2664292E7, 32, 16, 16
0, 1, 0, 2.55431037E8, 1.07159598E8, 34, 17, 17
0, 1, 0, 1.596374E8, 5.5637417E7, 16, 8, 8
1, 0, 0, 2.45235189E8, 1.05017592E8, 33, 17, 16
0, 1, 0, 1.57824695E8, 5.5805822E7, 16, 8, 8
0, 1, 0, 2.52512341E8, 1.01944335E8, 28, 14, 14
1, 0, 0, 2.00892349E8, 6.5635376E7, 19, 10, 9
1, 0, 0, 1.9584664E8, 6.9478134E7, 21, 11, 10
0, 1, 0, 1.7705262E8, 6.5906895E7, 20, 10, 10
1, 0, 0, 2.61610758E8, 1.12074447E8, 35, 18, 17
0, 1, 0, 1.6463556E8, 5.7367624E7, 16, 8, 8
0, 1, 0, 2.41884094E8, 9.7314784E7, 34, 17, 17
0, 1, 0, 3.14562887E8, 1.28031782E8, 38, 19, 19
1, 0, 0, 2.09185154E8, 7.3301391E7, 23, 12, 11
0, 1, 0, 3.28083676E8, 1.42003208E8, 42, 21, 21
0, 1, 0, 2.16307872E8, 8.1341446E7, 20, 10, 10
0, 1, 0, 3.56170774E8, 1.35880171E8, 42, 21, 21
1, 0, 0, 2.73264044E8, 1.04191047E8, 29, 15, 14
1, 0, 0, 3.07828512E8, 1.29965568E8, 39, 20, 19
0, 1, 0, 2.33919008E8, 8.9189362E7, 24, 12, 12
1, 0, 0, 2.47151221E8, 8.7557523E7, 25, 13, 12
1, 0, 0, 2.85866748E8, 1.14126264E8, 39, 20, 19
1, 0, 0, 1.61927827E8, 5.4163013E7, 17, 9, 8
1, 0, 0, 2.80538879E8, 1.21267924E8, 39, 20, 19
0, 1, 0, 1.86724173E8, 6.6926685E7, 18, 9, 9
0, 1, 0, 2.61066308E8, 1.04066753E8, 30, 15, 15
1, 0, 0, 2.5902246E8, 9.3849665E7, 27, 14, 13
0, 1, 0, 2.1561013E8, 7.8914036E7, 22, 11, 11
0, 1, 0, 3.04307857E8, 1.2520594E8, 40, 20, 20
1, 0, 0, 2.72550181E8, 1.14891859E8, 35, 18, 17
0, 1, 0, 2.85581422E8, 1.21278299E8, 38, 19, 19
0, 1, 0, 3.29303035E8, 1.43878669E8, 38, 19, 19
Iterations; P1 Algorithm; P1 Parameters; P1 Wins %; P1 Avg Moves per Game; P1 Avg Time Mico. Secs; P1 Power in mJ; P1 Avg MTSC Nodes Explored; P1 Strategies; P2 Algorithm; P2 Parameters; P2 Wins %; P2 Avg Moves
per Game; P2 Avg Time Mico. Secs; P2 Power in mJ; P2 Avg MTSC Nodes Explored; P2 Strategies; No of Draws; Why we doing this? What do we hope to see
100;02 MCTS; 1000; 38.0; 15.37; 250468.47437; null; 4552.1;null; 02 MCTS+01 NAIVE; 250; 57.99999999999999; 14.99; 98174.23615000001; null; 1080.49; [109, 36, 44, 36, -7, -74]; 4.0; null
```

Batch File Execution

There is a batch-file execution capability that allows any number of Connect4 tournaments or Genetic Algorithm training to be scheduled and executed sequentially without intervention. Both accept parameters that allow different scenarios to be executed.
For example, the following command executes a Connect X tournament with:

- Player1 MCTS agent with resource allowance of 1000 back propagations.
- Player2 MCTS agent with resource allowance of 90 back propagations and Naïve rules in the tree search, no Naïve rules in the playout.
- Player2 uses the supplied 5 strategy weights for the 5 strategies.
- The tournament length is 1000 games.
- The game to play is Connect4.
- Log file capture is on.

- On-screen debug messages are off.

```
java -cp packages -javaagent:joularjx/target/joularjx-2.8.2.jar src.Main ConnectX MCTS{1000} MCTS{90:1:0:97:37:126:78:-41} 1000 4 1 0
```

Similarly, Genetic Algorithm training can be scheduled using a command like the below where:

- The Genetic Algorithm will play the Naive algorithm against the baseline algorithm (MCTS:1000(P1)).
- The population size is 200.
- There will be 500 generations.
- Each member of the population will play 20 games to determine fitness.
- 6 strategies will be used.
- Connect4 is the chosen game.
- Log file capture is on.
- On-screen debug messages are off.

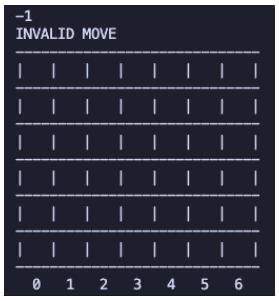
```
java -cp packages -javaagent:joularjx/target/joularjx-2.8.2.jar src.Main GA NAIVE 200 500 20 6 4 0 0
```

Testing

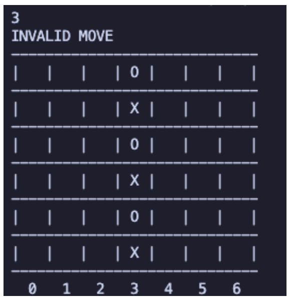
Testing the game environment

Initially, game environment tests were conducted to ensure that users could not make invalid moves, such as placing tokens outside the grid or in columns already filled with pieces. Initially this performed with the game configured such that Player1 and Player2 were human. This gave the best control for performing tests for different scenarios. In the below Player1 is always X and Player2 is O.

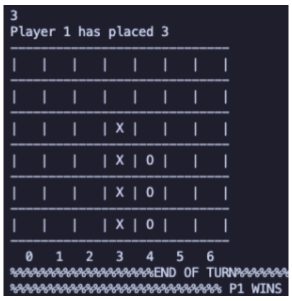
The first image depicts a test where an attempt to play outside the grid results in an "invalid move" message displayed in the terminal. The second image demonstrates that users are not allowed to place a piece on top of columns that are already full. An example of the game terminating correctly when a user executes a winning move is shown in image 3. Finally, I examined whether the game ended when the board reached full capacity.



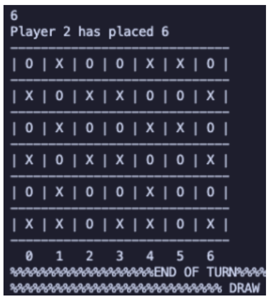
```
-1
INVALID MOVE
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
  0   1   2   3   4   5   6
```



```
3
INVALID MOVE
-----------------------------
|   |   |   | O |   |   |   |
-----------------------------
|   |   |   | X |   |   |   |
-----------------------------
|   |   |   | O |   |   |   |
-----------------------------
|   |   |   | X |   |   |   |
-----------------------------
|   |   |   | O |   |   |   |
-----------------------------
|   |   |   | X |   |   |   |
-----------------------------
  0   1   2   3   4   5   6
```



```
3
Player 1 has placed 3
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   | X |   |   |   |
-----------------------------
|   |   |   | X | O |   |   |
-----------------------------
|   |   |   | X | O |   |   |
-----------------------------
|   |   |   | X | O |   |   |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% P1 WINS
```



```
6
Player 2 has placed 6
-----------------------------
| O | X | O | O | X | X | O |
-----------------------------
| X | O | X | X | O | O | X |
-----------------------------
| O | X | O | O | X | X | O |
-----------------------------
| X | O | X | X | O | O | X |
-----------------------------
| O | X | O | O | X | O | O |
-----------------------------
| X | X | O | X | X | O | X |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% DRAW
```

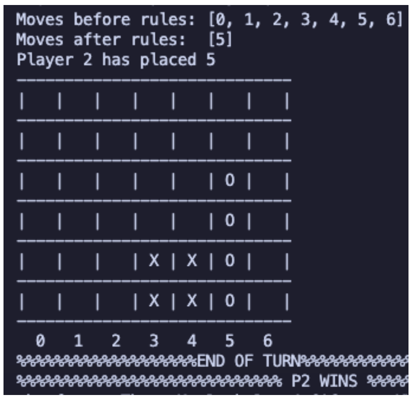
The Random agent was used to play many games unsupervised. This allowed me to check that the scheduling of games operated correctly as well as stress testing the program and confirm the collection of energy and other statistics in the log files.

Testing agents

The agents were tested individually using tests that were appropriate to their function.

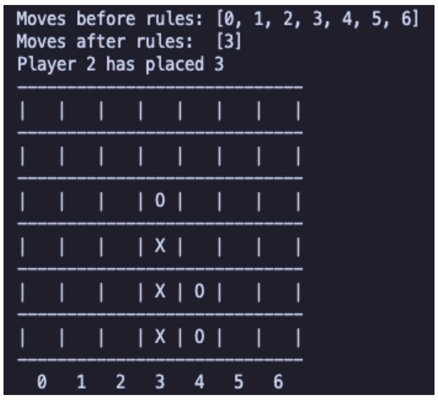
Naïve Testing

To validate the effectiveness of my Naïve agent implementation, different scenarios were created where the rules would come into play by playing as one of the players. Assessment of each rule performance was done one by one. For example, for rule 1 (placing a winning move) the figure below shows that Player2 could win by placing their piece in position 5. Before the rule were applied Player2 considered any move however once the rules were applied only position 5 remained available.



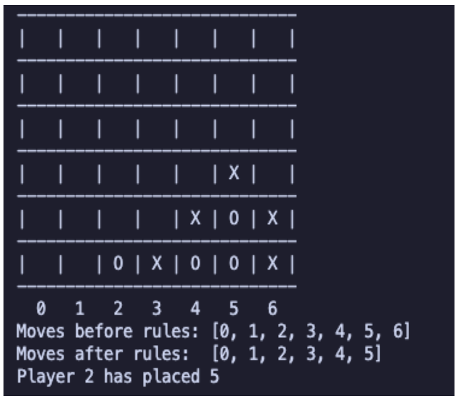
```
Moves before rules: [0, 1, 2, 3, 4, 5, 6]
Moves after rules:  [5]
Player 2 has placed 5
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   | O |   |
-----------------------------
|   |   |   |   |   | O |   |
-----------------------------
|   |   |   | X | X | O |   |
-----------------------------
|   |   |   | X | X | O |   |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% P2 WINS %%%%
```

In a separate scenario, I aimed to assess the effectiveness of rule two (blocking opponent's winning moves). To achieve this, I arranged a situation where Player1 could win on the next turn by placing a token in column 3. Intially Player2 considred any move from 0-6 but when the rules were applied the only avaialable moves left were position 3 meaning the rule worked.



```
Moves before rules: [0, 1, 2, 3, 4, 5, 6]
Moves after rules:  [3]
Player 2 has placed 3
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   | O |   |   |   |
-----------------------------
|   |   |   | X |   |   |   |
-----------------------------
|   |   |   | X | O |   |   |
-----------------------------
|   |   |   | X | O |   |   |
-----------------------------
  0   1   2   3   4   5   6
```

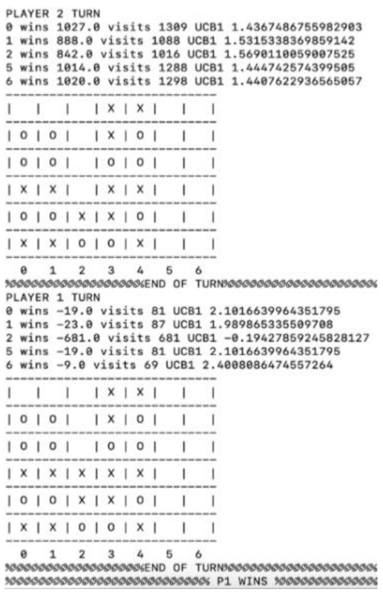
To test rule 3, which aims to prevent the creation of winning opportunities for the opponent, I constructed a game board where if Player2 placed their piece in position 6, it would allow Player1 to subsequently place theirs in position 6 and secure a victory. As shown if figure intially any move was considered but after the rules were applied position 6 became unavailable to Player2. This shows that the rule worked as Player2 cant play positon 6 resulting in a Player1 win.



```
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   | X |   |
-----------------------------
|   |   |   |   | X | O | X |
-----------------------------
|   |   | O | X | O | O | X |
-----------------------------
  0   1   2   3   4   5   6
Moves before rules: [0, 1, 2, 3, 4, 5, 6]
Moves after rules:  [0, 1, 2, 3, 4, 5]
Player 2 has placed 5
```

MCTS Testing

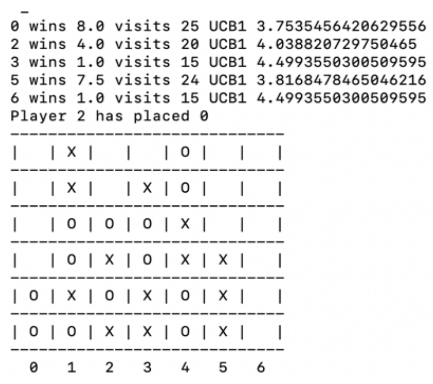
To test the MCTS algorithm, I played two instances of MCTS against one another to see their gameplay and analyse their UCB (Upper Confidence Bound) values by using debug mode. During testing and over time, several issues were found with my MCTS implementation as shown below.



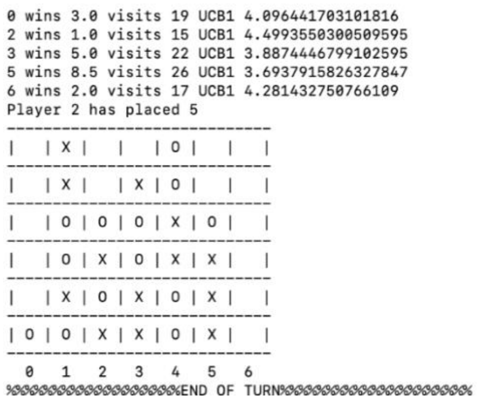
```
PLAYER 2 TURN
0 wins 1027.0 visits 1309 UCB1 1.4367486755982903
1 wins 888.0 visits 1088 UCB1 1.5315338369859142
2 wins 842.0 visits 1016 UCB1 1.5690110059007525
5 wins 1014.0 visits 1288 UCB1 1.444742574399505
6 wins 1020.0 visits 1298 UCB1 1.4407622936565057
-----------------------------
|   |   |   | X | X |   |   |
-----------------------------
| O | O |   | X | O |   |   |
-----------------------------
| O | O |   | O | O |   |   |
-----------------------------
| X | X |   | X | X |   |   |
-----------------------------
| O | O | X | X | O |   |   |
-----------------------------
| X | X | O | O | X |   |   |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%%%%
PLAYER 1 TURN
0 wins -19.0 visits 81 UCB1 2.1016639964351795
1 wins -23.0 visits 87 UCB1 1.989865335509708
2 wins -681.0 visits 681 UCB1 -0.19427859245828127
5 wins -19.0 visits 81 UCB1 2.1016639964351795
6 wins -9.0 visits 69 UCB1 2.4008086474557264
-----------------------------
|   |   |   | X | X |   |   |
-----------------------------
| O | O |   | X | O |   |   |
-----------------------------
| O | O |   | O | O |   |   |
-----------------------------
| X | X | X | X | X |   |   |
-----------------------------
| O | O | X | X | O |   |   |
-----------------------------
| X | X | O | O | X |   |   |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% P1 WINS %%%%%%%%%%%%%%%%%
```

In the figure above an issue was found where Player2 made a move in position 0, which was suboptimal, as it should have placed its piece in position 2 to stop Player1 from securing a victory. This error occurred because the win values were not reversed when the MCTS algorithm played as the other player. Note that at the point when Player2 made the move in position 0, the game was already lost for him because even if Player2 had blocked the potential four-in-a-row by placing their piece in position 2, Player1 could still win by placing a piece on top of Player2 to make a diagonal win.

Another problem occurred where Player2 failed to play in position 5 to block Player1 from winning (below). This issue occurred during my experimentation with small backpropagation values. To address this problem, the exploration constant was adjusted from 4 to 2. When the board was set up in the exact same state the algorithm successfully recognized that playing in position 5 was the correct move to prevent Player1 from winning.

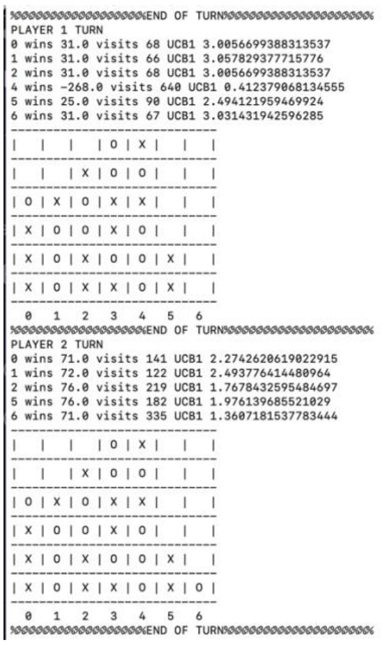


```
0 wins 8.0 visits 25 UCB1 3.7535456420629556
2 wins 4.0 visits 20 UCB1 4.038820729750465
3 wins 1.0 visits 15 UCB1 4.4993550300509595
5 wins 7.5 visits 24 UCB1 3.8168478465046216
6 wins 1.0 visits 15 UCB1 4.4993550300509595
Player 2 has placed 0
-----------------------------
|   | X |   |   | O |   |   |
-----------------------------
|   | X |   | X | O |   |   |
-----------------------------
|   | O | O | O | X |   |   |
-----------------------------
|   | O | X | O | X | X |   |
-----------------------------
| O | X | O | X | O | X |   |
-----------------------------
| O | O | X | X | O | X |   |
-----------------------------
  0   1   2   3   4   5   6
```



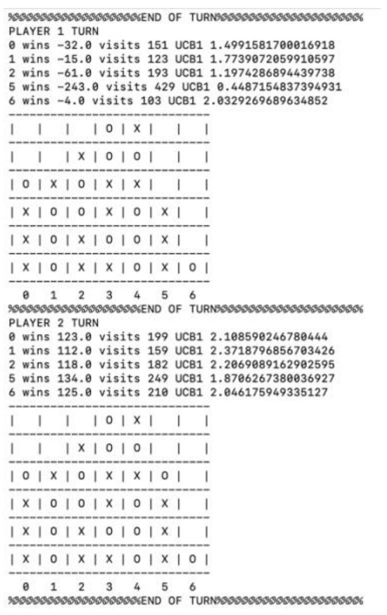
```
0 wins 3.0 visits 19 UCB1 4.096441703101816
2 wins 1.0 visits 15 UCB1 4.4993550300509595
3 wins 5.0 visits 22 UCB1 3.8874446799102595
5 wins 8.5 visits 26 UCB1 3.6937915826327847
6 wins 2.0 visits 17 UCB1 4.281432750766109
Player 2 has placed 5
-----------------------------
|   | X |   |   | O |   |   |
-----------------------------
|   | X |   | X | O |   |   |
-----------------------------
|   | O | O | O | X | O |   |
-----------------------------
|   | O | X | O | X | X |   |
-----------------------------
|   | X | O | X | O | X |   |
-----------------------------
| O | O | X | X | O | X |   |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%%%%
```

Another issue became apparent as shown below. Player2 should ideally play at position 5 to stop Player1 from placing a piece in that position creating three tokens in a row which would force Player2 to place a piece on top allowing Player1 to play on top of Player2 and win. However, the MCTS algorithm chose position 6.



```
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%%%
PLAYER 1 TURN
0 wins 31.0 visits 68 UCB1 3.0056699388313537
1 wins 31.0 visits 66 UCB1 3.057829377715776
2 wins 31.0 visits 68 UCB1 3.0056699388313537
4 wins -268.0 visits 640 UCB1 0.412379068134555
5 wins 25.0 visits 90 UCB1 2.494121959469924
6 wins 31.0 visits 67 UCB1 3.031431942596285
-----------------------------
|   |   |   | O | X |   |   |
-----------------------------
|   |   | X | O | O |   |   |
-----------------------------
| O | X | O | X | X |   |   |
-----------------------------
| X | O | O | X | O |   |   |
-----------------------------
| X | O | X | O | O | X |   |
-----------------------------
| X | O | X | X | O | X |   |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%%%
PLAYER 2 TURN
0 wins 71.0 visits 141 UCB1 2.2742620619022915
1 wins 72.0 visits 122 UCB1 2.493776414480964
2 wins 76.0 visits 219 UCB1 1.7678432595484697
5 wins 76.0 visits 182 UCB1 1.976139685521029
6 wins 71.0 visits 335 UCB1 1.3607181537783444
-----------------------------
|   |   |   | O | X |   |   |
-----------------------------
|   |   | X | O | O |   |   |
-----------------------------
| O | X | O | X | X |   |   |
-----------------------------
| X | O | O | X | O |   |   |
-----------------------------
| X | O | X | O | O | X |   |
-----------------------------
| X | O | X | X | O | X | O |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%%%
```

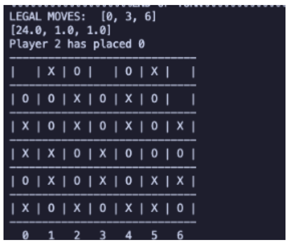
To address this problem, the reward for a draw was adjusted from 0.1 to 0.25. The previous reward for a draw was insufficient, causing the MCTS to play to aggressively and not aiming for a draw when necessary. This change successfully resolved the issue, as demonstrated in the below image above where the algorithm correctly determines that position 5 is the optimal move for Player2.



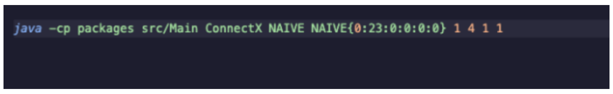
```
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%%%
PLAYER 1 TURN
0 wins -32.0 visits 151 UCB1 1.4991581700016918
1 wins -15.0 visits 123 UCB1 1.7739072059910597
2 wins -61.0 visits 193 UCB1 1.1974286894439738
5 wins -243.0 visits 429 UCB1 0.4487154837394931
6 wins -4.0 visits 103 UCB1 2.0329269689634852
-----------------------------
|   |   |   | O | X |   |   |
-----------------------------
|   |   | X | O | O |   |   |
-----------------------------
| O | X | O | X | X |   |   |
-----------------------------
| X | O | O | X | O | X |   |
-----------------------------
| X | O | X | O | O | X |   |
-----------------------------
| X | O | X | X | O | X | O |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%%%
PLAYER 2 TURN
0 wins 123.0 visits 199 UCB1 2.108590246780444
1 wins 112.0 visits 159 UCB1 2.3718796856703426
2 wins 118.0 visits 182 UCB1 2.2069089162902595
5 wins 134.0 visits 249 UCB1 1.8706267380036927
6 wins 125.0 visits 210 UCB1 2.046175949335127
-----------------------------
|   |   |   | O | X |   |   |
-----------------------------
|   |   | X | O | O |   |   |
-----------------------------
| O | X | O | X | X | O |   |
-----------------------------
| X | O | O | X | O | X |   |
-----------------------------
| X | O | X | O | O | X |   |
-----------------------------
| X | O | X | X | O | X | O |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%%%
```

Strategies Testing

To assess the effectiveness of the strategies implemented, a positive weight was assigned to one specific strategy while setting the weights of all other strategies to zero.

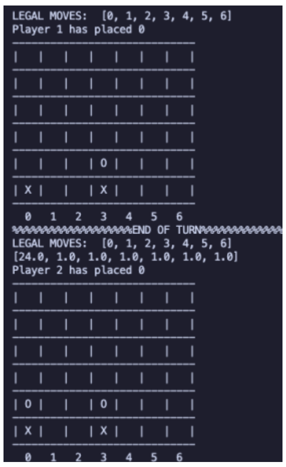


```
LEGAL MOVES:  [0, 3, 6]
[24.0, 1.0, 1.0]
Player 2 has placed 0
-----------------------------
|   | X | O |   | O | X |   |
-----------------------------
| O | O | X | O | X | O |   |
-----------------------------
| X | O | X | O | X | O | X |
-----------------------------
| X | X | O | X | O | O | O |
-----------------------------
| O | X | O | X | O | X | X |
-----------------------------
| X | O | X | O | X | X | O |
-----------------------------
  0   1   2   3   4   5   6
```



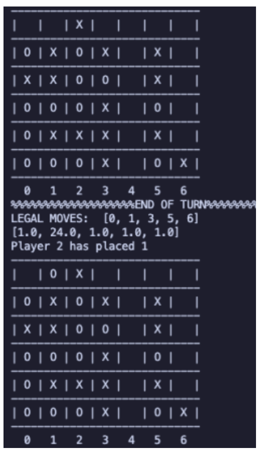
```
java -cp packages src/Main ConnectX NAIVE NAIVE{0:23:0:0:0:0} 1 4 1 1
```

In the image above, a weight of +23 was applied to strategy 2, with all other strategies having zero weight. Below, the strategy was correctly applied, as playing in position 0 gave a weight of +23, indicating the successful use of strategy 2, which aims to block an opponent's two.



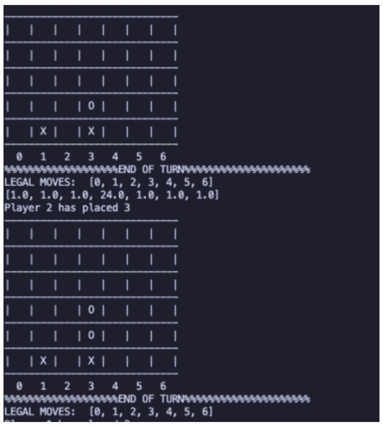
```
LEGAL MOVES:  [0, 1, 2, 3, 4, 5, 6]
Player 1 has placed 0
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   | O |   |   |   |
-----------------------------
| X |   |   | X |   |   |   |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%
LEGAL MOVES:  [0, 1, 2, 3, 4, 5, 6]
[24.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Player 2 has placed 0
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
| O |   |   | O |   |   |   |
-----------------------------
| X |   |   | X |   |   |   |
-----------------------------
  0   1   2   3   4   5   6
```

Next, strategy 1 was evaluated, which involves placing a piece on top of the opponent's piece. As shown in the image, the weight of +23 was assigned correctly to position 0, showing that the strategy was applied correctly as position 0 is placing on top of the opponent.



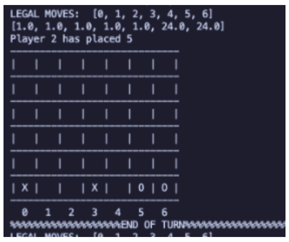
```
-----------------------------
|   |   | X |   |   |   |   |
-----------------------------
| O | X | O | X |   | X |   |
-----------------------------
| X | X | O | O |   | X |   |
-----------------------------
| O | O | O | X |   | O |   |
-----------------------------
| O | X | X | X |   | X |   |
-----------------------------
| O | O | O | X |   | O | X |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%
LEGAL MOVES:  [0, 1, 3, 5, 6]
[1.0, 24.0, 1.0, 1.0, 1.0]
Player 2 has placed 1
-----------------------------
|   | O | X |   |   |   |   |
-----------------------------
| O | X | O | X |   | X |   |
-----------------------------
| X | X | O | O |   | X |   |
-----------------------------
| O | O | O | X |   | O |   |
-----------------------------
| O | X | X | X |   | X |   |
-----------------------------
| O | O | O | X |   | O | X |
-----------------------------
  0   1   2   3   4   5   6
```

To evaluate strategy 3, which involves creating a sequence of three pieces for the player a specific board configuration was set up where strategy 3 would come into play. As shown by figure the weight of +23 was correctly assigned to position 1 which creates three pieces in a row for the player.



```
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   | O |   |   |   |
-----------------------------
|   | X |   | X |   |   |   |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%
LEGAL MOVES:  [0, 1, 2, 3, 4, 5, 6]
[1.0, 1.0, 1.0, 24.0, 1.0, 1.0, 1.0]
Player 2 has placed 3
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   | O |   |   |   |
-----------------------------
|   |   |   | O |   |   |   |
-----------------------------
|   | X |   | X |   |   |   |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%
LEGAL MOVES:  [0, 1, 2, 3, 4, 5, 6]
```

Next, was to verify the functionality of strategy 4, which involves placing a piece in the middle column. To accomplish this, a specific board configuration was set up where strategy 4 would come into play. As shown in the image, it's clear that strategy 4 was successfully employed, as evidenced by the assignment of a +23 weight to position 3, representing the middle column.



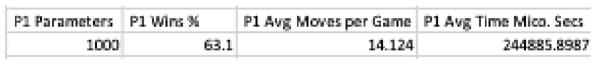
```
LEGAL MOVES:  [0, 1, 2, 3, 4, 5, 6]
[1.0, 1.0, 1.0, 1.0, 1.0, 24.0, 24.0]
Player 2 has placed 5
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
|   |   |   |   |   |   |   |
-----------------------------
| X |   |   | X |   | O | O |
-----------------------------
  0   1   2   3   4   5   6
%%%%%%%%%%%%%%%%%%%END OF TURN%%%%%%%%%%%%%%%%%%%%
```

Afterwards, the aim was to evaluate the effectiveness of strategy 5, which involves creating a sequence of two pieces for the player. To assess this strategy, a specific scenario was created, and observations were made to see if the strategy was applied correctly. As shown in the image above, strategy 5 was correctly implemented as a weight of +23 was given to position 5 and position 6 which creates a two for the player.

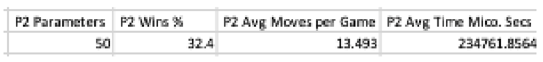
Naïve MCTS Testing

A significant issue was encountered when running the Naïve MCTS with only 50 back propagations, as it took an excessively long time to execute. Surprisingly, its performance was like that of the MCTS with 1000 back propagations, with both consuming similar amounts of energy.

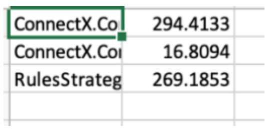
In the image below, the highlighted green box represents the energy consumption for MCTS 1000, while the other two boxes indicate the energy for Naïve MCTS. The rules and strategies used 269.1853 joules, while the entire MCTS 1000 consumed 294.4133 joules. In the figure below you can also see that the MCTS 1000 took 244885.9 microseconds while MCTS NAÏVE took 234761.85 microseconds.



P1 Parameters	P1 Wins %	P1 Avg Moves per Game	P1 Avg Time Mico. Secs
1000	63.1	14.124	244885.8987

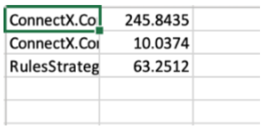


P2 Parameters	P2 Wins %	P2 Avg Moves per Game	P2 Avg Time Mico. Secs
50	32.4	13.493	234761.8564

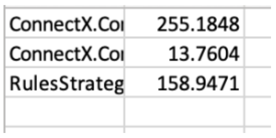


ConnectX.Co	294.4133
ConnectX.Coı	16.8094
RulesStrateg	269.1853

To fix this issue, rule 3 was removed which resulted in a reduction of energy consumption for Naïve MCTS. The energy dropped from 269.19 to 63.2 joules which is a drop of 77%. Then the efficiency of rules was enhanced by modifying them to now check if the opposing player has 3 pieces in a row before rule 2 and rule 3 are applied and to check if the player has 3 pieces in a row before rule 1 is applied. This reduces the times the rules are called. This optimization decreased the energy consumption from 269.19 to 158.94 joules, as shown in figure.



ConnectX.Co	245.8435
ConnectX.Coı	10.0374
RulesStrateg	63.2512



ConnectX.Coı	255.1848
ConnectX.Coı	13.7604
RulesStrateg	158.9471

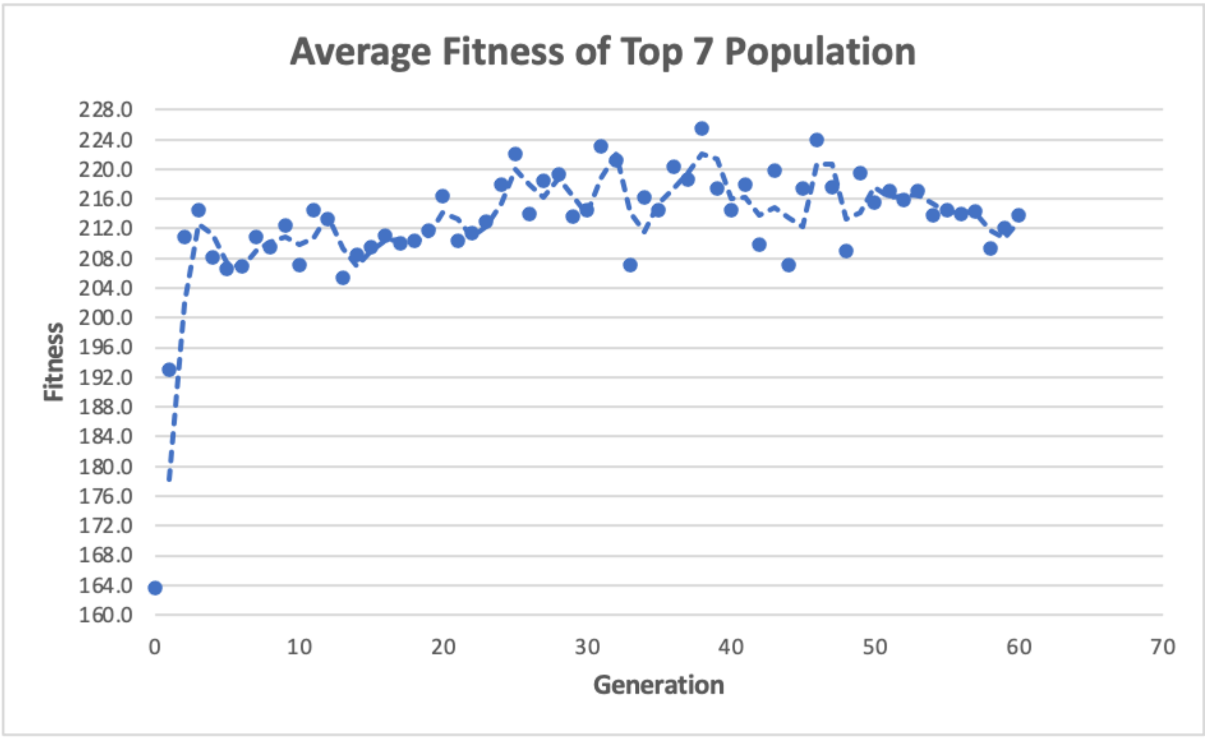
Training the Genetic Algorithm

Various combinations of Population Size, Games Played, Fitness function, Mutation Probability and Crossover Probability were tried to get the best convergence for the strategy weights. Due to the large number of games played for each combination it took a long time before the effectiveness of a particular combination could be determined. Various combinations were assessed meaning that training the Genetic Algorithm involved a great deal of time. The table below shows keys variations tested with the right column showing the final values that were used.

Generations	500	500	500	50	200	60	60	**100**
Population Size	200	200	200	100	40	40	35	**60**
No of fittest previous generation opponents played	20	20	20	20	0	0	0	**0**
Games Played per opponent	1	1	10	10	20	20	100	**32**
Mutation Probability	0.4	0.05	0.05	0.1	0.2	0.2	0.2	**0.2**
Crossover Probability	0.6	0.6	0.6	0.3	0.4	0.4	0.4	**0.4**

Initially, opponents used when playing games to determine fitness, were sourced from the best performers of the previous generation. Due to poor convergence and overall performance, after several variations, the strategy was changed such that Player1 was always the baseline MCTS:1000. This gave a more representative and relatable metric against which to measure performance.

The fitness of a population member was determined by the number of games won, drawn or lost, earning 3, 1 and 0 points respectively.



The graph demonstrates that the fitness saw rapid improvement in the first 4 generations. For future generations improvement was scarce, slowly trending upwards with a lot of variance.

By comparing both the value of the weights and the difference for the weights for each strat at intervals of 10 generations we can interpret the and effects of the strategies and the effectiveness of the convergence of the weights.

	Differences in weights for each strategy				
Differences between:	Strat 1	Strat 2	Strat 3	Strat 4	Strat 5
Generations 0,10	-30.0	50.0	31.0	-4.0	-118.0
Generations 10,20	116.0	-26.0	2.0	-8.0	44.0
Generations 20,30	-8.0	14.0	4.0	8.0	-18.0
Generations 30,40	-12.0	-35.0	0.0	0.0	2.0
Generations 40,50	-98.0	1.0	-2.0	-33.0	-30.0
Generations 50,60	-52.0	-1.0	-30.0	-10.0	102.0

	Weights for respective strategy				
Generation	Strat 1	Strat 2	Strat 3	Strat 4	Strat 5
0	7.0	73.0	89	127.0	67
10	-23.0	123.0	120	123.0	-51
20	93.0	97.0	122	115.0	-7
30	85.0	111.0	126	123.0	-25
40	73.0	76.0	126	123.0	-23
50	-25.0	77.0	124	90.0	-53
60	-77.0	76.0	94	80.0	49

Table 1+2 demonstrate a very loose sense of convergence for the weights of strategies 2,3 and 4. The weights for these strategies remained quite high and positive indicating that following these strategies give a player a better chance of winning a Connect4 game. However, convergence is too poor to be able to determine the limit of convergence for any of these 3 weights.

In the case of the weights for strategies 1 and 5, Table 1+2 demonstrate no sense of convergence. These weights fluctuate greatly taking both positive and negative values with no discernible pattern. This may indicate that these strategies do not actually significantly influence the outcome of a Connect4 game.

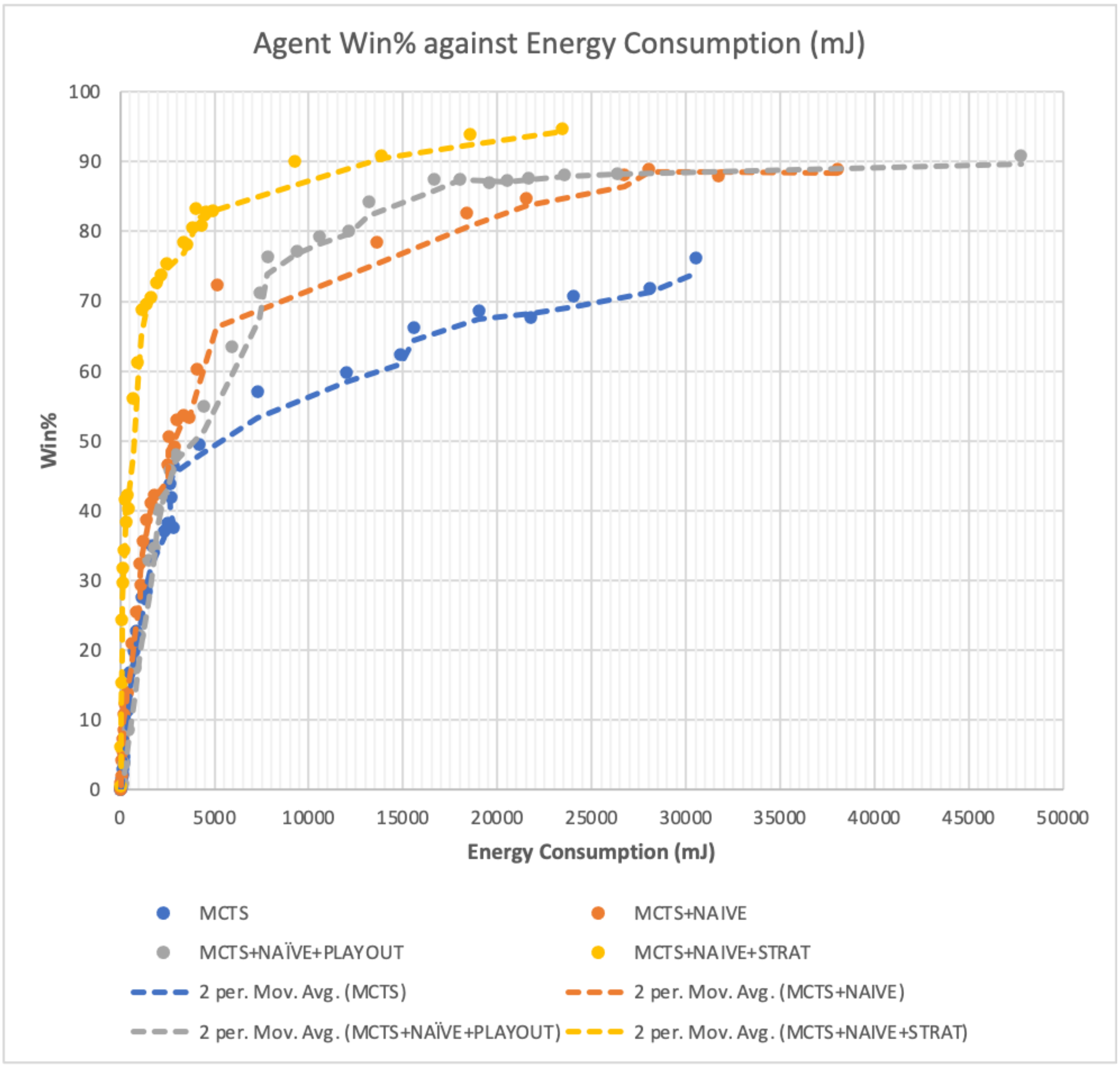
The lack of convergence can be partially attributed to the randomness involved in the agents playing connect 4. MCTS involves random processes meaning that an agent with theoretically ‘better’ weights didn’t always win more than a worse agent. Convergence could potentially be improved by playing more games to achieve a more accurate average fitness of each member of the population during each generation.

NAÏVE+STRATS was used to determine fitness of weights because although it is not one of the agents compared in the results section, it is much faster to run, giving more time to determining optimal mutation and crossover probabilities. With more time MCTS+NAÏVE+STRATS could be used in determining fitness to achieve more ‘accurate’ weights as they would be determined by the same agent they would be used for.

For the final weights used in MCTS+NAÏVE+STRATS we chose the best performing weights found over all generations in NAÏVE+STRATS.

Results

Comparison of Agent Performance



The above graph shows the difference of Win% against energy consumption for the 4 tested agents. This is the main statistic we use to compare the effectiveness of the agents.

It is important to note that the points don't 'match' on the x axis because in testing we used available resources (number of back propagations) as our independent variable and both win% and energy were dependent variables.

All agent Win%'s follow a similar pattern of starting at 0 then rising rapidly until energy consumption is about 5000mJ. The Win% gradient then starts to level out receiving diminishing effectiveness benefits with more energy consumption.

We can see that the basic MCTS performs the worst at all energy consumptions and never reaches a win rate above 80%.

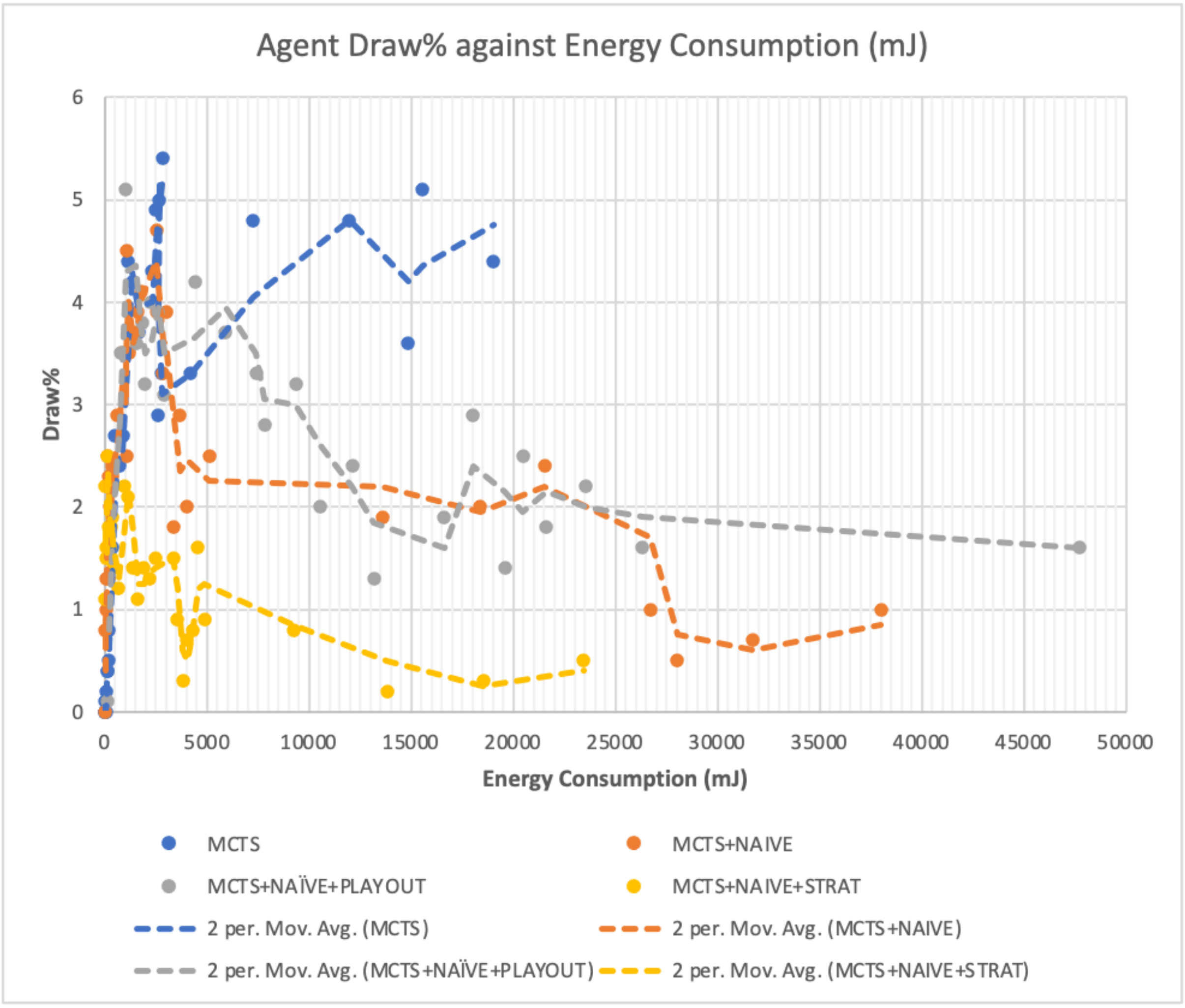
The MCTS+NAÏVE and MCTS+NAÏVE+ROLLOUT perform very similarly. With MCTS+NAÏVE performing better until 7500mJ of energy consumption when then the MCTS+NAÏVE+PLAYOUT performs better until 27500mJ where they then both perform similarly.

The MCTS+NAÏVE+STRATS performs the best at all energy consumptions. The MCTS+NAÏVE+STRATS achieves a Win% of 50% at around 750mJ of energy. The Win% for all the other agents is below 25% at this energy. It's Win% is about 75% at the energy consumption required for the next agent to reach 50%. At 14000mJ of Energy Consumption no other agent can match MCTS+NAÏVE+STRATS Win% for any measured energy consumption.

Although the MCTS+NAÏVE+STRATS performs the best by far it is also dependent on prior training to find the weights used in the agent. In total during testing, it took 932122J to train the MCTS+NAÏVE+STRATS.

A Win% of 75% for the second-best agent, MCTS+NAÏVE+PLAYOUT, requires 7500mJ of energy, whereas MCTS+NAÏVE+STARTS only requires 2500mJ. This means that 186,423 games would be needed to be played to justify the training cost of using MCTS+NAÏVE+STRATS over MCTS+NAÏVE+PLAYOUT.

This demonstrates that domain knowledge is useful for these agents when considering win% against energy only in the case that the agent would be used many times in the future.



The above graph shows the different Draw% against Energy Consumption for the 4 tested agents. All agents Draw% follow a similar pattern of starting at 0, rapidly rising before decreasing with increasing energy consumption.

We can see that in general, the agents with a higher effectiveness are the agents with a lower Draw% for any given Energy Consumption. The peak of the Draw% also occurs at a lower energy consumption the better the effectiveness of the agent.

The draw rate reflects the growth in effectiveness of the agents with respect to increased energy. With 0 energy consumption, and hence zero time to “think”, the opponent MCTS1000 always wins. As the energy consumption of the agents increases from 0 the agents improve allowing them to win and draw games. This results in a sharp increase in the Draw%. However, as the energy consumption of the agent increases further, they start to win almost all their games, leading to fewer draws.

Discussion / Conclusion

Review of Project Aims

The aim of this project was to explore the relationship between performance and energy consumption using the game of Connect4. Through experimentation it was shown that the agents that used more domain knowledge achieved significantly better effectiveness over those that used less. Though this domain knowledge comes at a steep energy cost It provides a permanent improvement to the agents at all energy consumption levels.

Future Work

Applying other AI learning techniques, for example Neural Network / Deep Reinforcement Learning (DRL), to MCTS could deliver similar energy consumption benefits over the basic MCTS approach. Similarly, for real-time scenarios where the problem space is changing constantly, such as driving a car, the inclusion of other techniques into MCTS may give a better energy consumption outcome.

Lessons Learned

This project has demonstrated how difficult it is to be efficient at managing time, estimating and keeping to planned timescales. I also learned valuable information that testing all ways takes much longer than expected. A skill I learned in this project was how to analyse data using Excel which I used for that purpose and for creating all my graphs. I encountered many challenges programming this project especially with the genetic algorithm, but I learned how to face the challenges and push on. In the end this project was a very valuable experience.

Overall Conclusion

This project explored the relationship between AI performance against energy consumption using the domain of Connect4. This document shows that domain knowledge can be introduced with different AI techniques to provide better effectiveness. The completion of this project allowed me to develop key skills that will surely help me in the future. Exploring the subject of AI learning was interesting and made the Final Year Project a rewarding experience.

References

[1]Energy Consumption of ICT, The Parliamentary Office of Science and Technology https://researchbriefings.files.parliament.uk/documents/POST-PN-0677/POST-PN-0677.pdf

[2]How to stop data centres from gobbling up the world's electricity https://www.nature.com/articles/d41586-018-06610-y

[3]A.I. Could Soon Need as Much Electricity as an Entire Country – New York Times https://www.nytimes.com/2023/10/10/climate/ai-could-soon-need-as-much-electricity-as-an-entire-country.html

[4]Connect Four overview Connect Four - Wikipedia MCTS overview https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

[5]Genetic algorithm https://en.wikipedia.org/wiki/Genetic_algorithm

[6]JoularJX Multi-Platform Software Power Monitoring Tools by Adel Noureddine JoularJX (noureddine.org) https://www.noureddine.org/research/joular/joularjx

[7]MIT Technology Review https://www.technologyreview.com/2019/11/11/132004/the-computing-power-needed-to-train-ai-is-now-rising-seven-times-faster-than-ever-before

[8] Benbassat, A. and Sipper, M., 2013, August. EvoMCTS: Enhancing MCTS-based players through genetic programming. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)* (pp. 1-8). IEEE.

[9] Perez, D., Samothrakis, S. and Lucas, S., 2014, August. Knowledge-based fast evolutionary MCTS for general video game playing. In *2014 IEEE Conference on Computational Intelligence and Games* (pp. 1-8). IEEE.

[10] Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S. and Colton, S., 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, *4*(1), pp.1-43.

[11] Lambora, A., Gupta, K. and Chopra, K., 2019, February. Genetic algorithm-A literature review. In *2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon)* (pp. 380-384). IEEE.

[12] Yang, Z., 2022. *Integrating Domain Knowledge into Monte Carlo Tree Search for Real-Time Strategy Games*. Drexel University.

[13] Kim, M.J. and Ahn, C.W., 2018, July. Hybrid fighting game AI using a genetic algorithm and Monte Carlo tree search. In *Proceedings of the genetic and evolutionary computation conference companion* (pp. 129-130).

[14] Chen, K.H. and Zhang, P., 2008, June. Monte-Carlo Go with knowledge-guided simulations. In *Computer Games Workshop 2007* (p. 103).

[15] Soriano Marcolino, L & Matsubara, H 2011, Multi-agent Monte Carlo go. in AAMAS '11 The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 1. vol. 1, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, pp. 21-28. http://dl.acm.org/citation.cfm?id=2034396

[16] Chaslot, G., Fiter, C., Hoock, J.B., Rimmel, A. and Teytaud, O., 2010. Adding expert knowledge and exploration in Monte-Carlo Tree Search. In *Advances in Computer Games: 12th International Conference, ACG 2009, Pamplona Spain, May 11-13, 2009. Revised Papers 12* (pp. 1-13). Springer Berlin Heidelberg.

Appendix A – Project Proposal

Navigating Through a Swarm using MCTS Abstract

This project will aim to investigate what is the best algorithm for an agent to navigate through an unknown enemy swarm with no prior knowledge under time constraints.

The first steps of the project are trying to implement the Monte Carlo Tree Search (MCTS) into the infiltration game. This will allow me to collect data on how well the algorithm does against many unknown swarms ranging from easy to hard and compare to many different algorithms.

Intro

Swarms in nature refer to large groups of animals or organisms that exhibit coordinated and synchronized behaviour. Swarms can be found in various species across the animal kingdom, and they provide several advantages to the individuals within them. These are some examples of swarming behaviours in birds, and fish along with the advantages they offer

1. Birds: Birds use swarming to find food more efficiently. If one bird finds a reliable food source it will alert the flock and the entire flock benefits [1].

2. Fish: When fish are in a swarm it is called a school and fish in a school have a lower chance of being eaten by a predator than fish not being in a school.[2]

Swarm robotics is the study of how to design a group of robots that operate without any external infrastructure or centralised control [3]. The primary point of using robotic swarms is to enhance scalability, adaptability, and resilience in various applications. Examples of robotic swarms include:

1. Search and Rescue: In disaster-stricken areas, robotic swarms of drones or ground robots can efficiently search for survivors and gather information about the environment. These swarms can cover large areas quickly, improving the chances of locating and helping those in need.

2. Precision Agriculture: Robotic swarms can be used in agriculture to monitor crops, apply pesticides, or perform other tasks. By working in unison, these robots can optimize resource usage and crop yield.

Consider the challenge of navigating through a traffic jam during rush hour. The complexity of swarm behaviour arises from the individual agents' interactions, making it difficult to predict their movements. Attempting to avoid the swarm may result in several problems, such as:

1. Unpredictable Behaviour: The collective behaviour of a swarm is emergent, and individual agents often respond to their immediate environment and the actions of neighbouring agents. This leads to unpredictability, making it challenging to determine the best avoidance strategy in real-time.

2. Dynamic Nature: Swarms, whether natural or robotic, exhibit dynamic behaviour that changes over time. The positions and velocities of agents constantly evolve, further complicating avoidance strategies.

Avoiding a swarm deterministically in real-time is virtually impossible due to the complex and dynamic nature of swarm behaviour. Predicting the exact positions and movements of individual agents within a swarm at any given moment is an impractical task. As a result, real-time avoidance strategies need to be adaptive and probabilistic.

To tackle the problem of avoiding swarms, methods like the Monte Carlo Method can be applied. This statistical technique involves running simulations with randomized variables to estimate possible outcomes. For instance, in traffic management, the Monte Carlo Method can simulate various scenarios to determine optimal routes for vehicles to navigate through congested areas. By generating many possible outcomes, this method provides probabilistic solutions for avoiding swarms, considering the unpredictability and complexity of swarm behaviour.

background

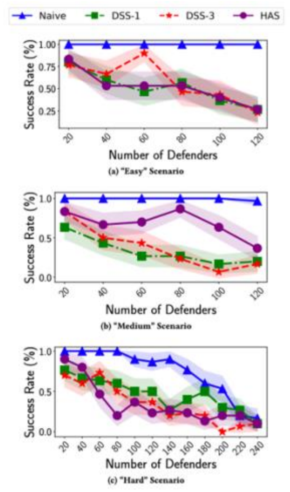
This project is based on this paper about Real-time Learning and Planning in Environments with swarms [4]. The paper proposes two novel algorithms for real-time swarm avoidance they are called Hierarchal Abstraction (HAS) and Direct swarm simulation (DSS).

HAS works by modelling the swarm as an ellipse instead of individual members then it uses ARIMA (Auto-Regressive Integrated Moving Average) to work (estimate) out the values for the ellipse in the future.

DSS works with the assumption that we know what the algorithm is, but we don’t know the list of parameters. We work out an estimation for parameters explained in the paper. Then it works out the waypoints of the swarm by putting the points into a tipple and looking at the distance between point 1 and 2 (V1), and then point 2 and 3(V2) and if the angle between them is bigger than a constant it is a waypoint. Then it moves onto the next tipple.

Then it compares the performance between the two algorithms and a very basic naïve algorithm. The naïve algorithm performs the best in the easy scenarios with a pretty much 100 percent success rate. However, in the hard scenarios against 240 defenders it has a success rate below 25 percent which is quite low.

DSS and HAS both use Monte Carlo simulations to overcome the problem of not being able to calculate all outcomes due to the size of problem space and the real-time requirements.



Aims and objectives

Monte Carlo tree search (MCTS) is a method that relies on intelligent tree search that balances exploration and exploitation.[5] It differs from Monte Carlo method as MCTS uses a tress structure for its decision making while Monte Carlo doesn't. MCTS is an algorithm that builds a search tree node by node according to the outcomes of simulated playouts.[6]. It can be broken down into 4 steps Selection, Expansion, Simulation and Backpropagation.

Selection starts at a root node (R) and then repetitively select the most favourable child node until a leaf node is reached (L).

Expansion is when the leaf node is not a node that does not end the game (terminal node) then generate one or more new child nodes from L and proceed to select one of them and call them C.

Simulation is when a run a hypothetical playthrough starting from node C and continue until you reach an outcome or result.

Backpropagation incorporates the outcome of the simulated playthrough into the existing sequence of moves, making modification as necessary based on the new information. This step involves integrating the results of the trial into the ongoing strategy or plan.

Every node in the tree should store two critical pieces of data: a predicted value derived from the outcomes of simulated trail and the count of how manty times it has been explored or traversed.

There are many achievements of the MCTS algorithms one of them is when Alpha Go used MCTS to defeat the three-time Go European Champion Fan Hui in October 2015[7].

For this project I aim to build on the works of Leandro's paper by incorporating a MCTS into the DSS and HAS algorithm in place of the Monte Carlo simulations. Once the method has been implemented, I will measure the performance of the updated DSS and HAS with the MCTS algorithm against DSS and HAS using the Monte Carlo Method and the naïve algorithm. I will be using the similar test that original paper used such as comparing the success rate of the algorithms and the time taken.

methodology

The project will use the common waterfall approach to software development. There will be an analysis and design phase followed by development and testing and finally an evaluation exercise.

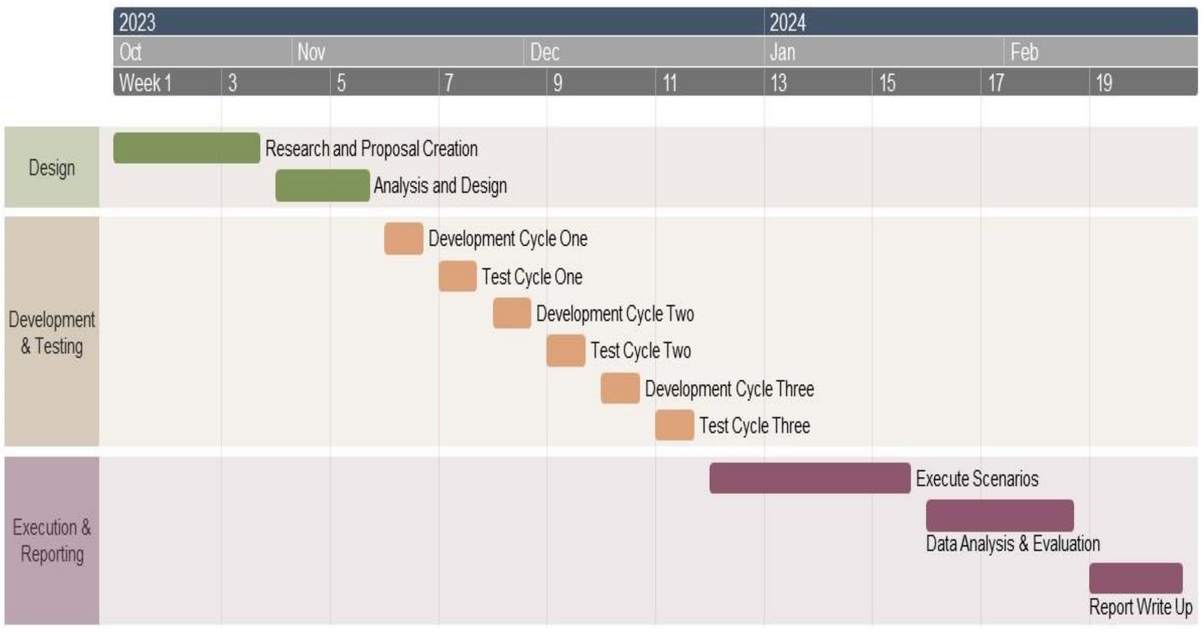
The development phase will use one of two existing frameworks that have implemented the DSS, HAS and naïve algorithms. One of the frameworks is written in Java and the other one in Python.

The decision as to which framework will be used will be made during the design phase.

Implementing into the existing frameworks will allow the new Monte Carlo Tree Search algorithms to be evaluated in the same way as the existing Monte Carlo Method algorithms.

Programme of work

A high-level project plan for the proposed work is provided below.



design phase

The design phase covers initial research and the creation of this proposal to create a rough scope of work. Afterwards, a more detailed analysis phase will be carried out to make sure that the problem is understood. A design of the how the solution will be implemented will then be created and this will include identifying key aspects like classes and the language (Java or Python) that will be used for development. This will take about two weeks.

development and testing

The development and testing phase will take about six weeks. I have allowed for three separate development and testing cycles. Errors found during testing will be fixed in the next development cycle and then testing will be repeated. The actual testing to be done will be defined during the first test cycle.

execution and reporting

After successfully finishing testing, formal scenarios will be run to create data that shows how the new algorithms perform. Once the result data has been created it will be analysed in a way similar to the current algorithms this will then allow the new algorithms to be compared in terms of performance.

All the results and any conclusions made will then be written up in the final report. There are about eight or nine weeks to do this, but this includes some extra time to use as contingency in case other phases are delayed.

Resources

This project will be built into one of the two existing frameworks, one in Java one in Python, that support the current implementation of HAS, DSS and naïve algorithms.

This will support direct comparison of performance of the new algorithms against the existing ones. As mentioned above the selection of the framework will be made during the design phase.

Reference:

[1] https://www.wwt.org.uk/news-and-stories/blog/marvellous-murmurations-why-do-birds-flock-together/

[2] https://www.sciencedirect.com/topics/medicine-and-dentistry/fish-school

[3] http://www.scholarpedia.org/article/Swarm_robotics
[4] https://ifaamas.org/Proceedings/aamas2020/pdfs/p1019.pdf
[5] https://builtin.com/machine-learning/monte-carlo-tree-search

[6] https://www.cs.swarthmore.edu/~mitchell/classes/cs63/f20/reading/mcts.html

[7] https://www.techopedia.com/google-deepminds-achievements-and-breakthroughs-in-ai-research#:~:text=AlphaGo%20also%20achieved%20a%209,most%20effective%20response%20or%20outcome.