|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **ID** | **Full Name** | **المستوى الدراسي** | **Department** |
| **1** | **202000863** | **مروان احمد عبدالتواب جاب الله** | **Level 3** | **IS** |
| **2** | **202000555** | **عبدالله نبيل ابراهيم عيد** | **Level 3** | **IS** |
| **3** | **202000199** | **بسمله سامح عبدالله عبدالغني** | **Level 3** | **IS** |
| **4** | **202000964** | **مي ممدوح قناوي حسين** | **Level 3** | **IS** |
| **5** | **202000345** | **رؤى محمود عبدالجواد محمود** | **Level 3** | **IS** |
| **6** | **202000363** | **زينب حسام الدين مصطفى محمد** | **Level 3** | **IS** |

Shared file: -

<https://drive.google.com/drive/folders/1-xqwlRBF8esbRfZASGhcdks0KJ9IzSSQ?usp=sharing>

1) Project idea in details: -

Connect-Four is a [tic-tac-toe](https://mathworld.wolfram.com/Tic-Tac-Toe.html)-like two-player game in which players alternately place pieces on a vertical board 7 columns across and 6 rows high. Each player uses pieces of a particular color (commonly black and red, or sometimes yellow and red), and the object is to be the first to obtain four pieces in a horizontal, vertical, or diagonal line. Because the board is vertical, pieces inserted in a given column always drop to the lowest unoccupied row of that column. As soon as a column contains 6 pieces, it is full and no other piece can be placed in the column.

Connect Four (also known as Four Up, Plot Four, Find Four, Captain's Mistress, Four in a Row, Drop Four, and Gravitrips in the Soviet Union).

Both players begin with 21 identical pieces, and the first player to achieve a line of four connected pieces wins the game. If all 42 men are played and no player has places four pieces in a row, the game is drawn.

The game has been completely analyzed, so it is known that if both players play with optimal strategies, the first player can always win (Allis 1988). The numbers of possible positions after n=0, 1, 2, ... have been played is 1, 7, 56, 252, 1260, 4620, 18480, 59815, 206780, ... (OEIS [A090224](http://oeis.org/A090224)).

2) Main functionalities: -

Game Engine:

|  |  |  |
| --- | --- | --- |
| ID | Function | Description |
| 1 | Create board | Create a numpy array of zeroes to represent the Connect 4 board. This will be populated with numbered pieces throughout the game. |
| 2 | Is valid location | To check if there is a valid location in the chosen column. |
| 3 | Get next open row | To check which row the piece can be placed into (i.e. the next available open row). |
| 4 | Print board | Reverse the order of board along axis 0 (up/down). |
| 5 | Check winning move | Check horizontal locations for win, Check vertical locations for win, Check positively sloped diagonals, Check positively sloped diagonals. |
| 6 | Evaluate window | the look-up table is calculated according to the evaluate window function below. Here, the window size is set to four since we are looking for connections of four discs. Considering a reward and punishment scheme in this game. If four discs are connected, it is rewarded for a high positive score (100 in this case). When three pieces are connected, it has a score less than the case when four discs are connected. When two pieces are connected, it gets a lower score than the case of three discs connected. Finally, when the opponent has three pieces connected, the player will get a punishment by receiving a negative score. Indicating that it is not an optimal move for the current player. |
| 7 | calculate all scores for each possible move(score position) | With the scoring criteria set, the program now needs to calculate all scores for each possible move for each player during the play. The function score\_position performs this part from the below code snippet. The AI player will then take advantage of this function to predict an optimal move. |
| 8 | Is terminal node | Before implementing the minimax algorithm, the two game-terminating states need to be defined as terminal nodes. If there is a winning move from either player, or if the board fills up without a win (leading to a draw), the game will end. |
| 9 | Define minimax algorithm | First, the program will look at all valid locations from each column, recursively getting the new score calculated in the look-up table (will be explained later), and finally update the optimal value from the child nodes. Notice that the alpha here in this section is the new\_score, and when it is greater than the current value, it will stop performing the recursion and update the new value to save time and memory. |
| 10 | Get valid locations | To get all locations in the board that could contain a piece (i.e. have not yet been filled). |
| 11 | Pick best move | Return the best column. |
| 12 | Draw board | Set up the board to print out in the terminal in a way that makes it visually easy to play with the computer. |

Player And AI:

|  |  |  |
| --- | --- | --- |
| ID | Function | Description |
| 13 | Drop piece | to place a piece in the next available row, in the chosen column |

(3) Similar applications in the market: -

- Teeko

is an abstract strategy game invented by John Scarne in 1937 and rereleased in refined form in 1952 and again in the 1960s. Teeko was marketed by Scarne's company, John Scarne Games Inc its quirky name, he said, borrowed letters from Tic-tac-toe

**explaining of game:**

The Teeko board consists of twenty-five spaces arranged in a five-by-five grid. There are eight markers in a Teeko game, four black and four red. One player, "Black" plays the black markers, and the other, "Red", plays the red. Black moves first and places one marker on any space on the board. Red then places a marker on any unoccupied space; black does the same; and so on until all eight markers are on the board. The object of the game is for either player to win by having all four of his markers in a straight line (vertical, horizontal, or diagonal) or on a square of four adjacent spaces. (Adjacency is horizontal, vertical, or diagonal, but does not wrap around (the edges of the board.) If neither player has won after the "drop" (when all eight pieces are on the board), then they move their pieces one at a time, with Black playing first. A piece may be moved only to an adjacent space.

**Online version :**

[**https://en.m.wikipedia.org/wiki/Teeko**](https://en.m.wikipedia.org/wiki/Teeko)

- source four

Score Four is a "three dimensional" abstract strategy game similar to Connect Four (Milton Bradley 1974). It was first sold under the name "Score Four" by Funtastic in 1968. Lakeside issued 4 different versions in the 1970s. Later Hasbro sold the game as "Connect Four Advanced" in the UK.

**explaining of game:**

The object of Score Four is to position four beads of the same color in a straight line on any level or any angle.[1] As in Tic Tac Toe Score Four strategy centers around forcing a win by making multiple threats simultaneously, while preventing the opponent from doing so.

**Online version:**

[**http://intensecomputers.com/portfolio/scorfor/ScorFor.html**](http://intensecomputers.com/portfolio/scorfor/ScorFor.html)

- 3D Tic-Tac-Toe

3D tic-tac-toe, also known by the trade name Qubic, is an abstract strategy board game, generally for two players. It is similar in concept to traditional tic-tac-toe but is played in a cubical array of cells, usually 4x4x4. Players take turns placing their markers in blank cells in the array. The first player to achieve four of their own markers in a row wins. The winning row can be horizontal, vertical, or diagonal on a single board as in regular tic-tac-toe, or vertically in a column, or a diagonal line through four boards

**explaining of game:**

On the 4x4x4 board, there are 76 winning lines. On each of the four 4x4 boards, or horizontal planes, there are four columns, four rows, and two diagonals, accounting for 40 lines. There are 16 vertical lines, each ascending from a cell on the bottom board through the corresponding cells on the other boards. There are eight vertically-oriented planes parallel to the sides of the boards, each of these adding two more diagonals (the horizontal and vertical lines of these planes have already been counted). Finally, there are two vertically-oriented planes that include the diagonal lines of the 4x4 boards, and each of these contributes two more diagonal lines—each of these including two corners and two internal cells.

The 16 cells lying on these latter four lines (that is, the eight corner cells and eight internal cells) are each included in seven different winning lines; the other 48 cells (24 face cells and 24 edge cells) are each included in four winning lines.

**Online version:**

<https://apps.apple.com/eg/app/3d-tic-tac-toe/id1474860497>

----------------------------------------------------------------------------------------------

(4) An initial literature review of Academic

publications (papers) relevant to the idea: -

- Paper 1

**Introduction**: -

* The concept of the Connect 4 game is to get, before your opponent, four chips in a row, arranged either diagonally, *vertically*, or horizontally.

There are several levels of AI difficulty:

1- random (randomly picks a play and is the easiest to beat).

2- defensive (makes blocking a win a priority).

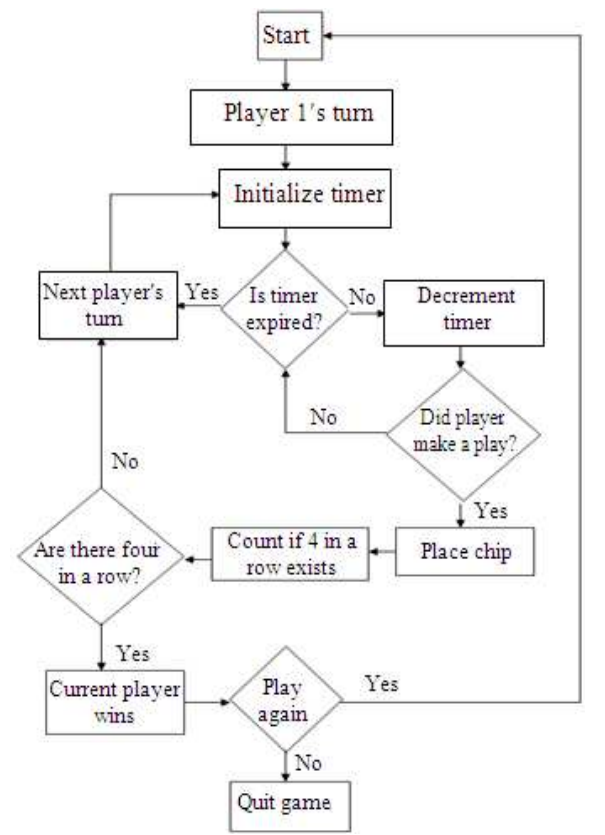
3- aggressive (makes winning a priority)

Defensive and aggressive are harder to beat than the random AI.

Solution algorithms for AI are:

1. minimax (it’s the hardest to beat)
2. minimax with alpha-beta pruning
3. A\* (may not be a fit for this real time game because the A\* algorithm may not meet the game deadline)

* This game was implemented as a real-time game. t. It was a 2 player game and had a 5 sec timer for play. The plays were made on an interactive GUI. Once the current player makes a play, the chip is placed in that column within 1 sec. When the chip is in place, the timer is reset. When the timer expires, the current player loses his/her turn.

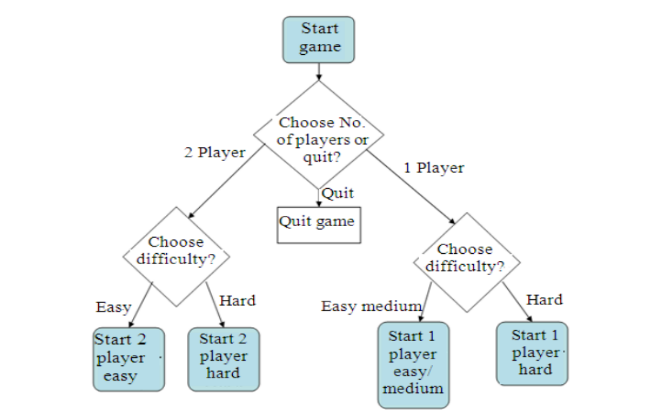


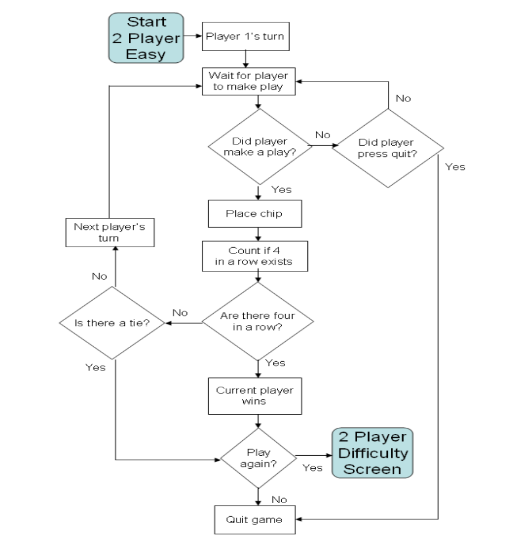
**Material and methods: -**

in Connect 4, only the top available spot in each column needs to be looked at not all the entire board unlike other games. The layout is a grid of six rows by seven columns.

this figure shows how the game is run.

Original software design flow chart



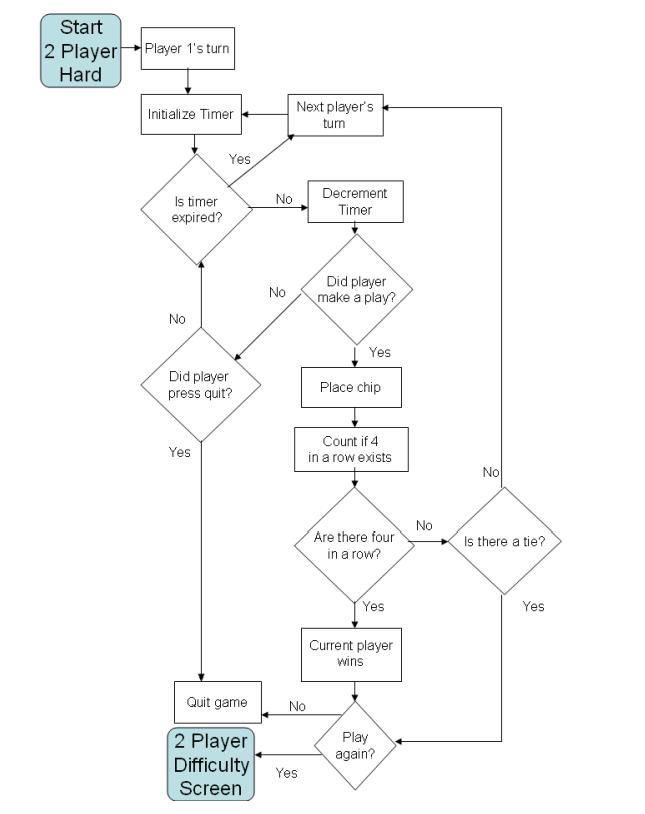
There are 3 levels of difficulty in this game that the player will choose from:

1. The easy level (is easier to beat):

It has no timer, the timer is invisible and disabled. The program will wait until the current player makes a move or selects quit. After a play is made, it checks if that player has won or caused a tie.

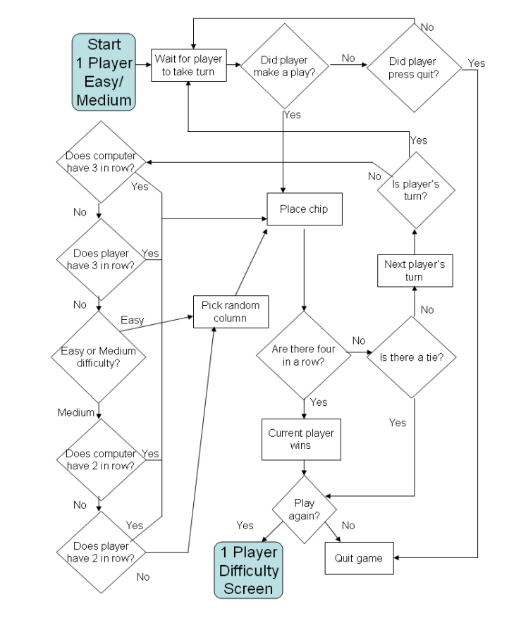
1. The medium level:

It includes the AI from the easy level and is modified making it harder to beat.



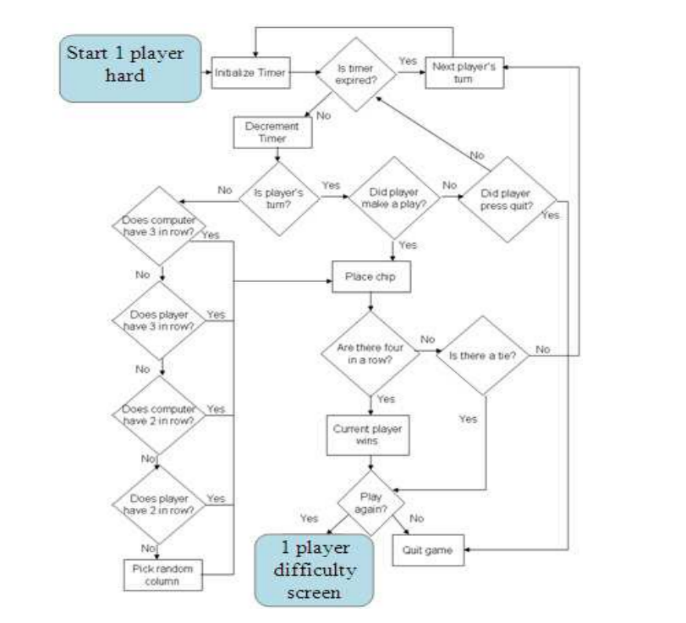
1. The hard level:

It initiates the timer and makes it visible to the players. It plays the same as the two player easy mode except for the lost turn when the timer expires.

There are 2 modes in this game: the one-player (between player and computer’s player) and the two player.

* The easy/medium -level one player:
* It starts by looking at the horizontal direction for the move it is scoring. It examines the space to the left of the move and sees if the player it is scoring has a chip there.
* If that player has a chip there, the count is increased. It continues to count until either the space is not occupied for the player being scored or it reaches the end of the board. It then looks to the right and does the same thing.
* The moves are then scored in the left and right diagonal directions. The scoring algorithm checks down and up the diagonal for the number of chips in a row that are connected to that move until they hit the other player's chip, an empty space, or the end

of the board

The hard-level one player:

Picking the hard level gives you the same AI that the medium level has. What makes it harder is that a 5 sec timer is added. The player has 5 seconds to make a play after the computer plays. This requires quick thinking in order to beat the computer. A delay was added for the computer player so that the move could be seen easily by the human.

The easy-level two player:

Select easy mode and play the game. Make sure that there are no bugs and that it flows as if you were playing with a live board and chips.

The hard-level two player:

Select the hard mode and make sure that it flows like the easy mode except for the timer. Make sure that when the timer expires that the current player changes. Make sure that when the current player makes a move that the next player is made current and the timer resets.

**The improvements in this version of the game over the older version are:**

* Timer: In this version, the timer is made an option whereas in the old program, the timer was always activated
* Computer player AI: There is now a choice between one and two players, whereas the old version was just a two-player game

**The conclusion:**

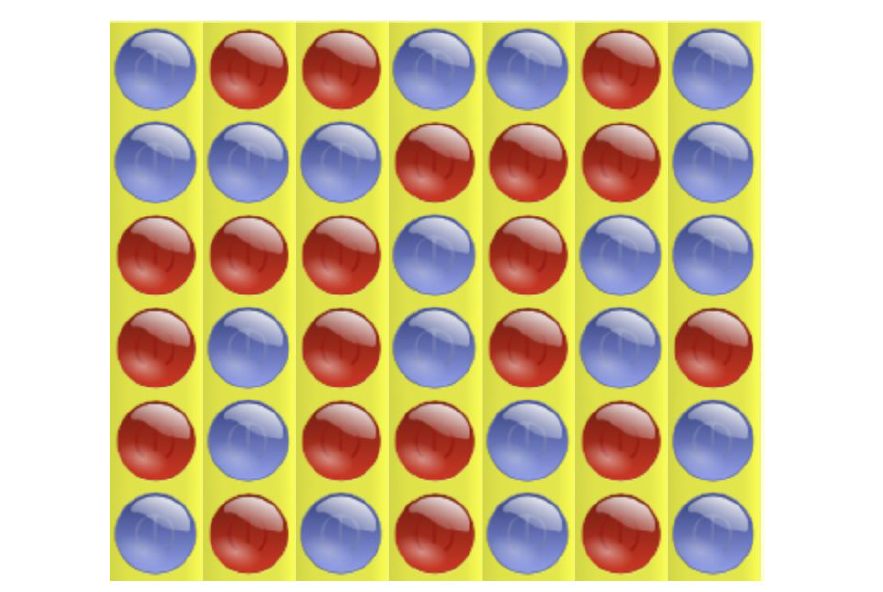
. The game was updated to include the choice of one or two players. If two players were chosen, there was a choice of having a timer. If one player was chosen, there was the choice of easy AI, medium AI or hard AI. The only level in the one player mode that has a timer is the hard level.

- Paper 2

**Introduction:**

* Connect Four is a well-known two-player strategy game. Each player has their own colored discs and players alternate in selecting a slot on the board to drop the disc in with the goal of getting four of their colored discs in a row, either horizontal, vertical or diagonal.
* Connect Four is played on a vertical board with six rows and seven columns, at the top vertical edge of the board, for each column there is slit where the pieces are slotted into. Once the piece has been slotted into a column it falls down the column to the lowest row or it lands the row above the piece that was last played in that column.
* It is a two player game and each player has 21 coin-like pieces in a different color to the other player (player one may have all red pieces and player two all yellow).
* The game begins by randomly deciding whether player one or player two will take the first turn. After the first player takes their turn, turns are then taken alternatively between the two players. A turn consists of a player dropping their colored piece into a column on the board. Additionally, once the piece has been played, it cannot be undone or removed from the board.

1. connect 4 board



* The objective of Connect Four is to be the

first player to connect four of their colored pieces in a

row (either vertically, horizontally or diagonally)

with no gaps on the board between the four piece . At this point the game is over and the

player who connected the four pieces wins. Alternatively, if all of the pieces have been played,

resulting in the board being full, then the game is called **a tie**

**Problem Description:**

Our project attempts to find an optimal

Connect Four playing strategy (the sequence of turn actions that

B) A tie game

maximizes the probability of a player winning a

single Connect Four game) given a Connect Four board.

**Action Space :**

An action is defined as (small action)dropping a piece

into one of the seven columns on the board. Because

the number of actions a player may take at a given

state depends on how many columns on the board are

full, we calculate the number of actions possible at a

given state as:

**Actions Possible = 7 - C**

**C = Number of full columns at current state**

**State Space:**

The state space for Connect Four is considerably larger than the action space The board with played pieces that a player sees. If the player starts the game, then the board will always have an even number of pieces. Alternatively if they are second player, the board will always have an odd number of pieces.

**LITERATURE REVIEW:**

Scientists have already proven that if the first player places their piece in the center column, with perfect play they cannot lose.

However, given a certain state, **what is the optimal move**? This requires some complex calculations.

* Solutions include using a **minimax algorithm**, which minimizes the maximum losses for a player. This is done by assuming that the opponent is going to make the best possible decision, and therefore ignoring all other possibilities. This significantly reduces the number of states that need to be calculated, while still ensuring optimal outcome for a player. However, it still takes a while, especially in the beginning of the game. **That is where alpha-beta pruning comes in**, a technique which prunes away the need to consider many subtrees. It involves considering whether the values in a subtree could possibly change the current action indicated by the tree, and if there is no value that could change the action then the subtree is not calculated and ignored.

**IMPLEMENTATION:**

1. **Docker:** is an open-source tool that makes it easier to create, deploy, and run applications through the use of containers**.**
2. **Tensor Flow:** is an end-to-end open source platform for machine learning. We used the core open source library to help develop and train deep learning models for Connect Four (enable learning reinforcement which makes agent how to play).
3. **Simulator:** we could set up the agent we were training to play itself or another agent, we could then analyze the agent’s performance.
4. **Q-learning:** this means that rather than using transition and reward models, we use the observed next state *s’* and reward *r*
5. **Sarsa:** unlike Q-learning, we use the current state to get Q value

**RESULTS:**

In implementing our project we decided to have two agents, one for Q-learning and another for Sarsa. Within each agent, we choose to vary both the rewards for the agent and their exploration factor.

For the purposes of our project. A test means an agent played 1000 games of Connect 4.

Whether an agent played against itself or the other agent, **the specific rewards allocated, and what the exploration factor was, where all set for each test** which we ran within multiple docker containers. We broke down the testing into three cases below, **Q-learning vs. Q-learning, Sarsa vs. Sarsa, and finally Q-learning vs. Sarsa.** From this we varied the exploration factor (05. - 0.9) and rewards (3 options) player 1 win percentage iscalculated as follows:

(player 1 wins + (0.5\*ties)) / (total games played)

in our results we noticed that every single test run results in the majority of wins between the two agents always went to player 1, regardless of which agents were playing. This is consistent with

the prior work that has been conducted on connect 4 showing that with the perfect decisions being taken, player 1 cannot lose. Player 1 does not win 100% of the time because it is not taking the optimal strategy every time, for it is learning how to play the game as it plays.

we noticed interesting differences in these percentages across algorithm, exploration rate, and reward values. **Starting with Q Learning Vs Q Learning** we see that there is a negative correlation between win percentage and exploration rate. There is an interesting outlier with an exploration rate of 0.8, which is actually the most successful exploration rate we tested, however the general trend is downwards. We suspect that lower exploration rates result in higher success rate because player 1 sticks to what works

, does not compromise its advantage by exploring sub optimal policies, that while they may pan out, more often give up the superior position that results from starting the game.

**Sarsa vs Sarsa** has the same negative correlation between exploration rate and player one

win percentage that we observed with Q learning, with the difference being a lack of an outlier and a

smaller spread between exploration rates.

The most intriguing discovery about exploration rate was the effect that it had on Q Learning vs Sarsa, and similarly Sarsa vs Q Learning Here, increasing exploration rate elevated the success of Q Learning over Sarsa. increasing exploration rate elevated the success of Q Learning over Sarsa.

With an exploration rate of 0.5, Q Learning and Sarsa won about 66% of games when starting,whereas with

an exploration rate of 0.9 both algorithms won around 72.5% of games when starting. This is curious because when playing against oneself, both machine learning algorithms fared worse with a higher exploration rate. However, when the algorithms played against each other, both benefited from a higher exploration rate when being player 1.

The second parameter that we varied was the reward. Looking at the data, the effect of this variable is hard to deduce.

**CONCLUSION:**

Memorizing and determining the optimal move for every possible board state in a Connect Four game is humanly impossible. Our Q-learning agent was able to achieve a peak win percentage of 73.4% when starting as player one versus the Sarsa agent. Alternatively, when Sarsa started as player one versus Q-learning, our Sarsa agent was able to achieve a peak win percentage of 73.5%. Both algorithms never dipped below a win percentage of 50%, regardless of their opponent.

It was solved by showing that the first player can always win if they play the middle column first, we know that both our Q-learning and Sarsa method are ultimately suboptimal.

Both Q-learning and Sarsa methods have the ability to be modified, this means that the reward that is associated with reaching the goal is “propagated backward to the states and actions leading up to the

goal”

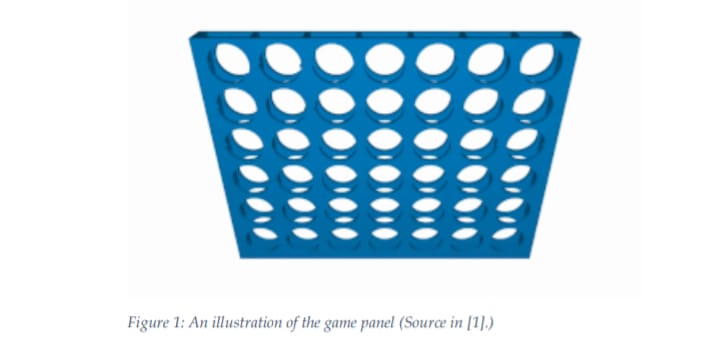
URL:

<https://web.stanford.edu/class/aa228/reports/2019/final106.pdf>

- Paper 3

Introduction

- Connect Four is a board game that is played by exactly two players, players in it are assigned to different colors and then take turns dropping colored discs into the suspended grid. The game’s grid has seven columns and six rows. The pieces fall straight down, occupying the lowest available space. Figure 1 llustrates the game panel.



-The main goal of the game is to be the first player to have either horizontal, vertical, or diagonal line of four same-colored discs. It is well known that Connect Four is a solved game, i.e. there is a specific known strategy by which the first player can always win by playing the correct plays. Hence, in this project, we try to play “semi-perfectly” against the AI and observe the results. Furthermore, unlike most of the card games, Connect Four does provide full information, where when a player at a time plays, all the two players get all the information regarding moves that have deterministically already taken place and regarding all moves that can take place for the next step, in a given game state. This makes implementing an AI player of the game more feasible for the purpose of this project. The project includes first, implementing the game environment itself in a suitable and user-friendly way, and second, an implantation of an AI player who we can configure its parameters and hardness. Finally, a general evaluation of the AI is presented.

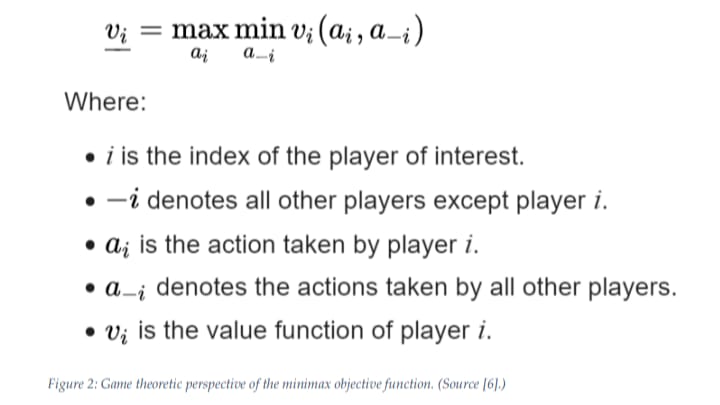
Theoretical Analysis

-Starting from the standard game, we will have a panel of 6 × 7 = 42 locations, each location has three possible states that can have; being empty, having the disc of the first player, or having the disc of the second player. That would directly give a rough upper-bound of 3 42 ≈ 1.1 × 1020 possible game situations that the search space of the game may have. However, we can compute a more accurate and tight upper bound of the possible situations by evaluating the sequence of possible games corresponding to a specific number of discs played so far, and then, just summing up the elements of this sequence of numbers. This sequence of possible situations after a number of turns will start from zero situations after zero turns, then seven situations after one turn (the first player has seven possible positions to place their disc on), then 7 × 7 = 49 positions after the second turn. After the second turn, it becomes a little bit trickier, calculating the number of possible situations after three turns is not trivial; the number of possible situations at that point will be 7 × 7 × 1 + 7 × 6 × 5 2 + 7 × 6 × 2 = 238 situations. Which is derived after considering that in some scenarios, different orders of placing the discs of the first player may produce the same situation of the game. Here, we consider all the cases. The whole sequence of possible situations after a specific number of turns, and starting from zero is given by the following sequence: 1, 7, 49, 238, 1120, 4263, 16422, 54859, 184275, … [2]. After calculating the number of situations after considering all 42 possible turns as well as the aero turn, we will end up with a total number of situations that is around 4.5 × 1012 situations. For a usual board game, an order of trillions of possible states in the search space is relatively in too large (like the search space in Chess or Go board games), and it is not too low that a simple brute-force algorithm that exhausts all the situations is feasible alone to do that. This proves that our choice to consider this game specifically was a good choice because implementing an AI agent that does not trivially exhausts all the possible scenarios would be useful in order to play against in this game.

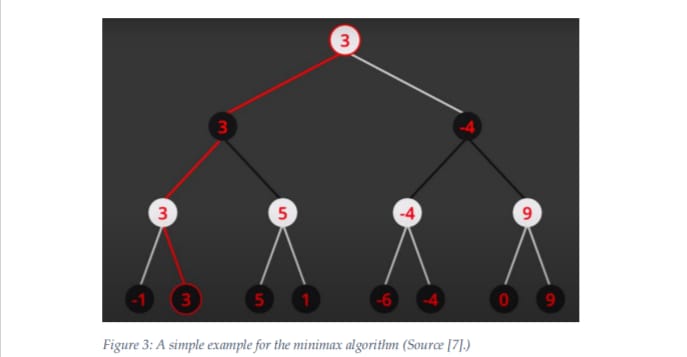
The Minimax Algorithm

-We implement an AI that mainly employs the minimax algorithm that we have learned in the class. The algorithm is simply implemented by making the players try to optimize some utility function that they have. 6 One player will try to maximize their utility function, which we will call the maximizing player. And the other player will try to minimize their utility function, which we will call the minimizing player. We can choose the utility functions for each player to be the complement of the other, so that, we would have a zero-sum game that is easier and more feasible to be implemented using minimax algorithm.

Figure 2 shows the algorithm from a game-theoretic perspective.



For the exact way the algorithm works, it is better to illustrate using an example; consider the simple example in Figure 3.

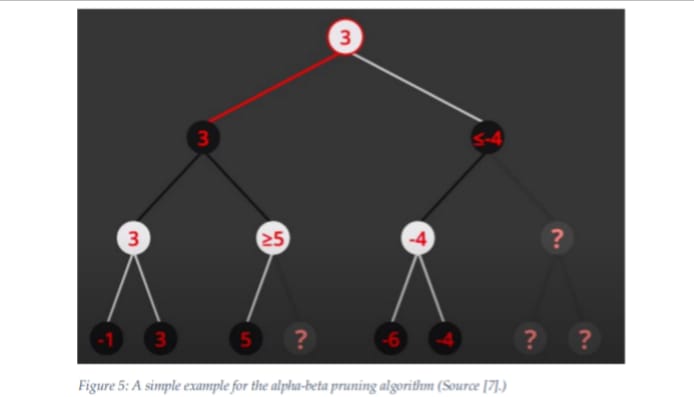


Here, we consider a simple game with a simple case where there are eight possible final situations of the game, where each one is assigned to a utility function that the first player (the white player) tried to maximize, and the second player (the black player tried to minimize.) The algorithm evaluates each leaf node using a heuristic evaluation function that we call the utility function, obtaining the values shown. The moves where the maximizing player has advantage are assigned with positive numbers, while the moves that lead to a situation where the minimizing player has advantage are assigned with negative numbers. As much the magnitudes of the numbers become larger, as much the advantage prevails towards one side.

The Alpha-Beta Pruning Algorithm

-As clear from our previous analysis of the minimax algorithm, it would be very computationally expensive to search through the whole depth of the tree because of the exponentially-increasing number of leafs. Hence a very feasible way to reduce the search space is by trimming a part of the tree of the possible situations of the game. Alpha–Beta Pruning can be considered as a general search algorithm which has an objective of minimizing the number of explored nodes that have utilityfunction values calculated by the main heuristic function evaluator that is employed in the minimax algorithm

Alpha-Beta Pruning algorithm is an adversarial search algorithm that is usually employed to implement an agent playing two-player game, which makes it a very great fit for the purpose of implementing and AI for the Connect Four board game. The idea behind the algorithm is that it terminated the evaluation of the utility function values of the nodes when at least one possibility gets found that proves that the play in inquiry is not better than a play that has already been previously examined. Such plays do not need to have their utility function values calculated further. When applied the Alpha-Beta Pruning algorithm on a standard minimax search tree, it returns a result of the same move as the minimax would return without applying the algorithm, but a new trimmed tree will be considered that has some branches, which are impossible to possibly affect the final decision of the algorithm, cut out. Hence, a more tractable problem will be considered rather than the main computationally-expensive original one. Figure 5 depicts an example of an application for the Alpha-Beta Pruning algorithm



In Figure 5, an example where it is not important to calculate the utility function of three possible situations out of eight because, after we have evaluated the other five already, we became sure that whatever values those leafs may have, that will not affect the final result of the main minimax algorithm in anyway.

We can easily observe that, when looking at the left subtree, whatever the value of the fourth leaf is assigned, the weight node above it will not ever have a value less than five. This means surely that the black node above it will inherit the value from the left child, which has a value of three, instead of inheriting it from the child on the right, which will not have a value less than three, let alone five. The same exact principle is applied on the right subtree that gets more nodes trimmed at once after the information of the first five leafs have already been discovered.

References

[1]<https://oeis.org/A212693>

[2] <https://www.youtube.com/channel/UCq6XkhO5SZ66N04IcPbqNcw>

[3] <https://en.wikipedia.org/wiki/Minimax>

[4] <https://www.youtube.com/watch?v=l-hh51ncgDI&t=37s>

[5] <https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning>

[6] <https://www.pygame.org/docs/>

- Paper 4

**Keywords**Heuristics, Minimax Algorithm, Connect-4 Game

**Introduction**

Minimax algorithm has already achieved significant success in area of game including chess, backgammon and Connect-4

AI still does not play Connect-4 game in a super optimal way. Connect-4 was first solved by James Dow Allen, and independently by Victor Allis in  
1988

Connect Four, known as Captain’s Mistress, is a two-player connection game  
on a 6 × 7 board

there are two ways to play Connect Four: defensive and aggressive. Defensive AI prevents  
its opponent from winning, whereas aggressive AI makes every possible move to  
connect four in a row ahead of its opponent; this paper discuses a program that  
relies on the “aggressive” way

The two players are called  
MAX and MIN separately. MAX makes moves to maximize its score while MIN  
tends to minimize MAX’s score

Minimax search algorithm is good at predicting its opponent’s move and then  
beating it , but the runtime of minimax is always an issue.

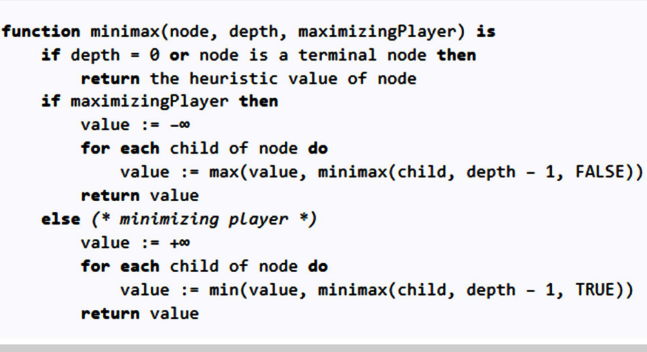
Minimax algorithm basically comes from the “Minimax theorem”, which was  
proposed by John von Neurmann in 1928. At first, minimax theorem was used  
in zero-sum game with two players knowing all moves that have taken place so  
far. However, John von Neumann improved and extended the minimax theorem  
that involving imperfect information games with more than two players, published this result in

**2.Methodology**

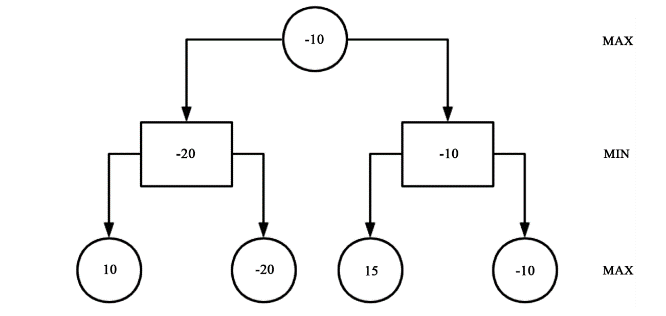
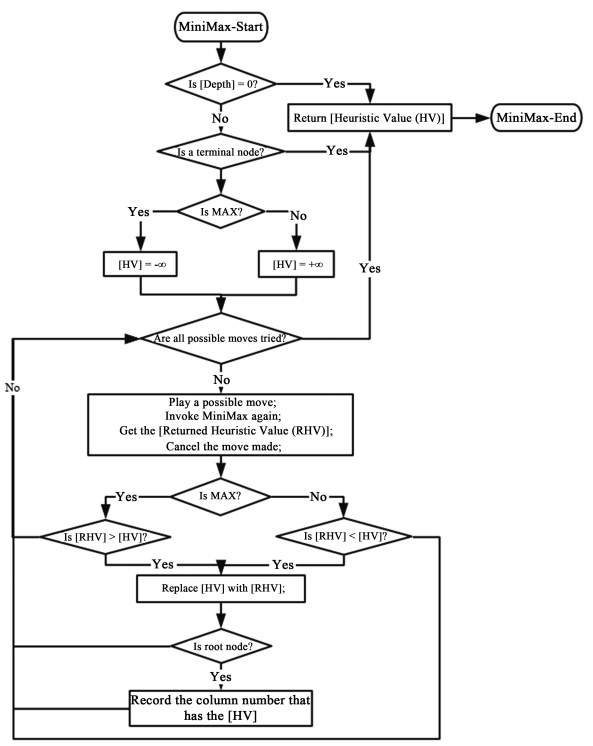
**2.1.Connect-4 Game**

Connect-4 game is a chess game on a board of 7 vertical columns of 6 squares  
each. Two players make their moves in turn till 4 men are connected horizontally, vertically or diagonally. Once a man is put in one of the columns, it will fall  
down to the lowest unoccupied square in the column. In our research, we designed an intelligent program to seek for the best move, using Minimax and heuristic search. The main steps for solving the best move.

Pseudo-code of minimax :

**2.2. Minimax**

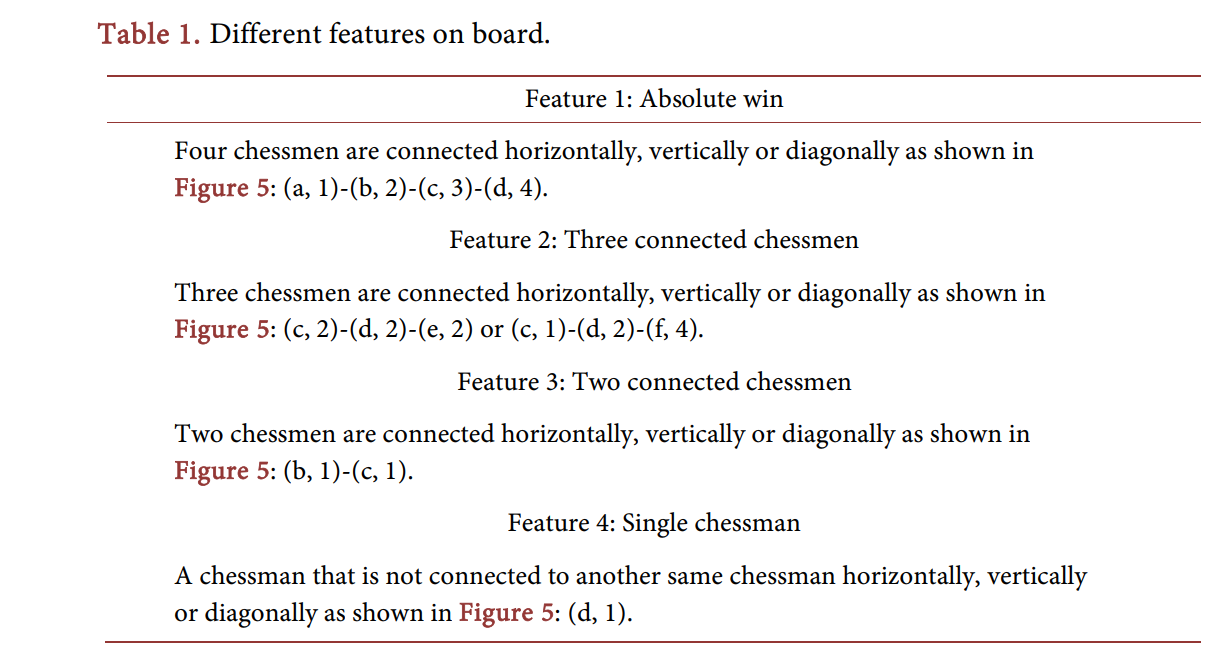
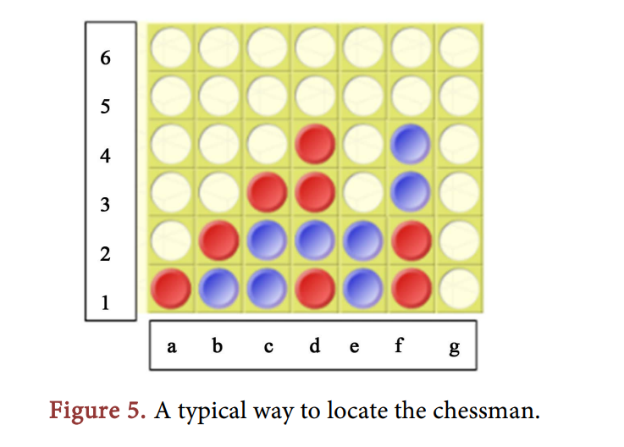
Minimax is used in artificial intelligence for decision making. In most cases, it is applied in turn-based two player games such as Tic-Tac-Toe, chess, etc. In our Connect-4 chess game, Minimax aims to find the optimal move for a player, assuming that the opponent also plays optimally. In Minimax, there are two players called Max and Min. Starting with Max trying its first move, Minimax algorithm will try all the possibilities of combination of Max’s and Min’s move. When either one wins or the game comes to a draw, an evaluation value of the board will be given to indicate the situation of the board. If some features on the board are in favor of Max, a positive value will  
be given to that feature. Otherwise, a negative value will be given. The final  
evaluation value is the summation of all the values of features. Max will choose the maximum evaluation value and Min will choose the otherwise. Eventually, Max will decide the best move. A descriptive figure is shown below.



The flowchart for Minimax algorithm is illustrated as previous Figure, In our research, this algorithm is specifically for Connect-4 chess game. To decide which column to play, it creates a game tree by trying out all the possibilities of combination of Max’s and Min’s moves. Each node of the game tree is initially given a heuristic value. If it is Max node, it’s given minus infinity. If it is Min node, it’s given infinity. When the pre-set depth is reached or the game ends, the heuristic value will be returned to its father node. The father node compares the returned value and its value and chooses the bigger one. When the root node  
receives a returned heuristic value, it will record not only the bigger heuristic  
value but also the column number corresponding to the value. After the root  
node tries out all the seven columns, the best column is recorded. Hence, the algorithm finds out the best move.

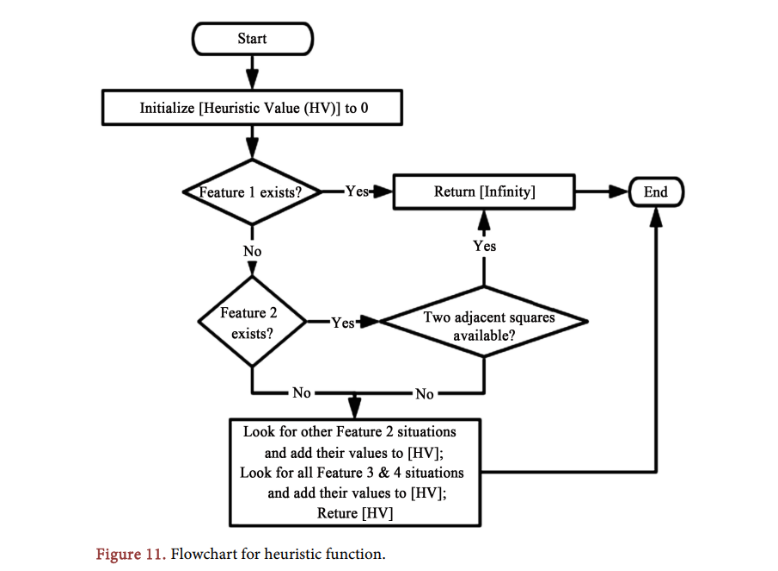
**2.3.Heuristics**Heuristic function is used in Minimax for evaluation of the current situation of  
the game. The final decision made by Minimax largely depends on how well the  
heuristic function is. Therefore, designing a reasonable heuristic function is paramount. In our research, we designed a heuristic function for Connect-4. To  
evaluate the current situation of the game, the heuristic function firstly looks for  
different features on the board and then gives them proper values. Finally, the  
heuristic function returns a summation of all the values of features on the chess  
board. We also introduced another heuristic function in [14]. It evaluates the  
board in a different way. It doesn’t look for features on the board. Instead, it  
evaluates each square on the board and gives them a proper value. We want the  
two heuristic functions to fight against each other so that we can assess them

**2.3.1. Heuristics-1**In our research, we look for 4 kinds of different features from the board. They  
are listed in Table1 below

  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
For these 4 features, we give them different values listed in Table 2 below.  
Now we will expand on the four features

Calendar

Description automatically generatedGraphical user interface

Description automatically generated  
  
**2.3.2. Heuristics-2**   
Different to heuristic 1, heuristic 2 doesn’t look for specific features on the  
board. Instead, it looks into every square on the board and gives them different  
evaluation values. If the square is more promising, it will get a higher value. The  
value for each square is shown in the following matrix.   
  
  
  
  
  
  
  
  
  
If the square is close to the middle column and row, it has a bigger value. For  
example, if a chessman is at (d, 3) or (d, 4), it has the biggest expansion space. It  
can form 4 connected men in its whole horizontal line, whole vertical line and  
whole diagonal line. However, if a chessman is put at (a, 1), it can only form 4  
connected men in its half horizontal line, half vertical line and half diagonal line.  
This square has much less possibility in forming 4 connected men than middle  
squares. Therefore, the values are corresponding to the square’s expansion space  
  
  
URL for full Research:  
<https://www.researchgate.net/publication/331552609_Research_on_Different_Heuristics_for_Minimax_Algorithm_Insight_from_Connect-4_Game>

- Paper 5

# 1.Introduction into Connect-Four

In this introduction the rules of the game Connect-Four are described, as well as some nomenclature used throughout this text.

## § 1.1. The Rules of the Game

Connect-Four is a game for two persons. Both players have 21 identical men. In the standard form of the game, one set of men is yellow and the other set is red. The game is played on a vertical, rectangular board consisting of 7 vertical columns of 6 squares each. If a man is put in one of the columns, it will fall down to the lowest unoccupied square in the column. As soon as a column contains 6 men, no other man can be put in the column. Putting a man in one of the columns is called: a move.

The players make their moves in turn. There are no rules stating that the player with, for instance, the yellow men should start. Since it is confusing to have to identify for each new game the colour that started the game, we will assume that the sets of men are coloured white and black instead of yellow and red. Like chess and checkers (and unlike go) it is assumed that the player playing the white men will make the first move.

Both players will try to get four connected men, either horizontally, vertically or diagonally. The first player who achieves one such group of four connected men, wins the game. If all 42 men are played and no player has achieved this goal, the game is drawn.

Diagrams 1.1, 1.2 and 1.3 show positions in which White has won the game:

White has won

●

●

●

Diagram 1.1.

White has won

●

●

●

●

Diagram 1.2.

White has won

●

●

●

●

●

●

●

Diagram 1.3.

In the position of diagram 1.1, White has made a horizontal winning group, while his winning groups were resp. vertical and diagonal in the other two diagrams.

A possible drawn position is shown in the diagram 1.4:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | ● |  |  |  |
| ● | ● | ● |  | ● | ● | ● |
|  |  |  | ● |  |  |  |
| ● | ● | ● |  | ● | ● | ● |
|  |  |  | ● |  |  |  |
| ● | ● | ● |  | ● | ● | ● |

Draw

Diagram 1.4.

# Different Approaches

## Complexity of the Game

In order to get an idea about the complexity of the game an estimate is presented of the number of different positions which can be achieved, if the game is played according to the rules. A position which can occur during a game is called a legal position, while a position which cannot be achieved is called illegal.

Each square can be in one of three states: empty, white or black. Therefore it is easy to see that the number of possible positions is at most 342 (≥ 1020). This upper bound is a very crude one, and can be brought into better proportions.

If the total number of occupied squares in a given position is odd, the number of white men is one more than the number of black men. If the total of occupied squares is even, these numbers are equal. Furthermore, if a column contains an empty square, all squares higher than this square are also empty. If a position contains four connected men, the position concludes a game. Since the last move ended the game, at least one of the four squares in the connected group must be the highest filled square in its column. If this is not the case, or both players have connected four men, the position is illegal. If one player has more than one connected group this position can only be legal if these groups share a square which contains the last man played. In the calculations we are going to make, we do not rule out positions in which are illegal for the reason mentioned above. We also do not rule out positions which are not legal, because they cannot be achieved, during normal play.

Diagram 2.1 shows such a position.

Illegal position

●

●

●

Diagram 2.1.

Although the position looks perfectly normal, it is clear that Black has made the first move.

Therefore it is not legal.

We have calculated the number of different positions, including all illegal positions which contain too many connected groups of four men, and illegal positions as shown in diagram 2.1. For this purpose a program was written in the C programming language, which can be found in appendix A.

For the standard, 7 x 6 board, the program found an upper bound of 7.1*\**1013.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  | ● |  |  |  |
|  | ● |  |  |  | ● |  |
| ● |  | ● |  | ● |  | ● |
| ● |  | ● | ● | ● |  | ● |
|  | ● |  | ● |  | ● |  |
| ● | ● | ● |  | ● | ● | ● |

To determine the amount of memory needed to construct a database for Connect-Four this upper bound is useful. In order to show that such a construction takes too much memory, we need a lower bound instead of an upper bound. If we want to find a (good) lower bound of the number of possible positions, we have to make sure that each position we count is legal. Therefore all positions which cannot be achieved during normal play, e.g. diagram 2.1, should be ruled out. We will show that this is not a trivial task. For this reason we have not determined a lower bound. Diagram 2.2 illustrates the difficulties we are faced with in determining if a position is legal.

Illegal position

Diagram 2.2.

The position of diagram 2.2 is a draw. Although at first sight it might look like a normal position, it cannot be achieved during normal play. This can be seen as follows: the first move White made must have been d1. If Black played as his first move one of b1, d2 and f1, there is no possible second move for White. Therefore Blacks second move was one of a1, c1, e1 and g1. Suppose Black played a1, White then must have played a2 as second move, giving the position of diagram 2.3:

Black to move

●

Diagram 2.3.

Now Black still cannot have played b1, d2 or f1, for the same reason as before. The move on a3 is not possible either. Therefore Black must have played one of the remaining c1, e1 or g1. After one of these, and after White’s answer to it, the position did not get any better. The farthest we can get with this game is shown in diagram 2.4.

Black to move

● ● ● ●

Diagram 2.4.

In this position Black has to move. For all seven columns, the lower two squares should be filled by black men. Therefore after the next move of Black there is no move White can make which will eventually result in the position shown in diagram 2.2. Therefore that position is illegal.

This diagram shows that it can be rather difficult to detect if a position is illegal. It is equally difficult to show which of the positions counted by the program of Appendix A, are not legal because more than one group of four connected men is present. We therefore assume that a database should contain a large number of illegal positions. We believe that in that case the order of magnitude of the upper bound presented before, is a good estimate for the magnitude of the database. This number is by far too big, to think seriously about making a database for Connect-Four. To see this, we have to consider the number of positions which must be stored at the same time, when we build the database. When a retrograde analysis is applied, as has been done for many endgames in chess [5], we need not necessarily store the positions consisting of, say, 20 men, as long as we have not yet determined the value of all positions of 21 men. When we have determined the value of these positions, we no longer need the positions consisting of 22 men or more. Therefore we only need to be able to store all positions of n and n+1 men at the same time. For the 7 x 6 board, this means that we must be able to hold all positions of 36 and 37 men at the same time, a total of over 1,6.1013 positions. Appendix C contains a table for the number of different positions for each number of men. We can store the value of a position in 2 bits, since we have 4 possible states: win for White, win for Black, draw or not checked (we can use the address of the 2 bits as identification for the position). This way we need at least 4 Terabyte. Therefore making a database does not seem realistic yet.

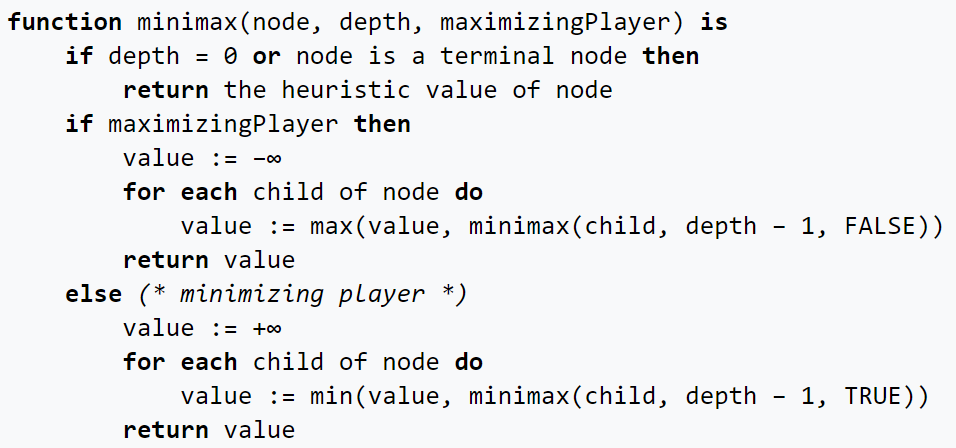
URL for all research:

<http://www.informatik.uni-trier.de/~fernau/DSL0607/Masterthesis-Viergewinnt.pdf>

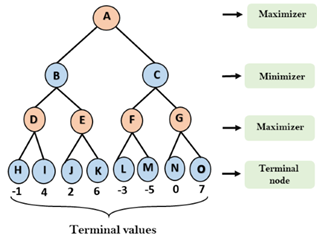
----------------------------------------------------------------------------------------------------------------------------------------------

(6) Details of the algorithm(s)/approach(es): -

* Alpha-beta is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
* is used in decision-making and game theory especially in AI game. It provides optimal moves for the player, assuming that the opponent is also playing optimally. For example, considering two opponents: Max and Min playing. Max will try to maximize the value, while Min will choose whatever value is the minimum. The algorithm performs a depth-first search (DFS) which means it will explore the complete game tree as deep as possible, all the way down to the leaf nodes. The algorithm is shown below with an illustrative example.



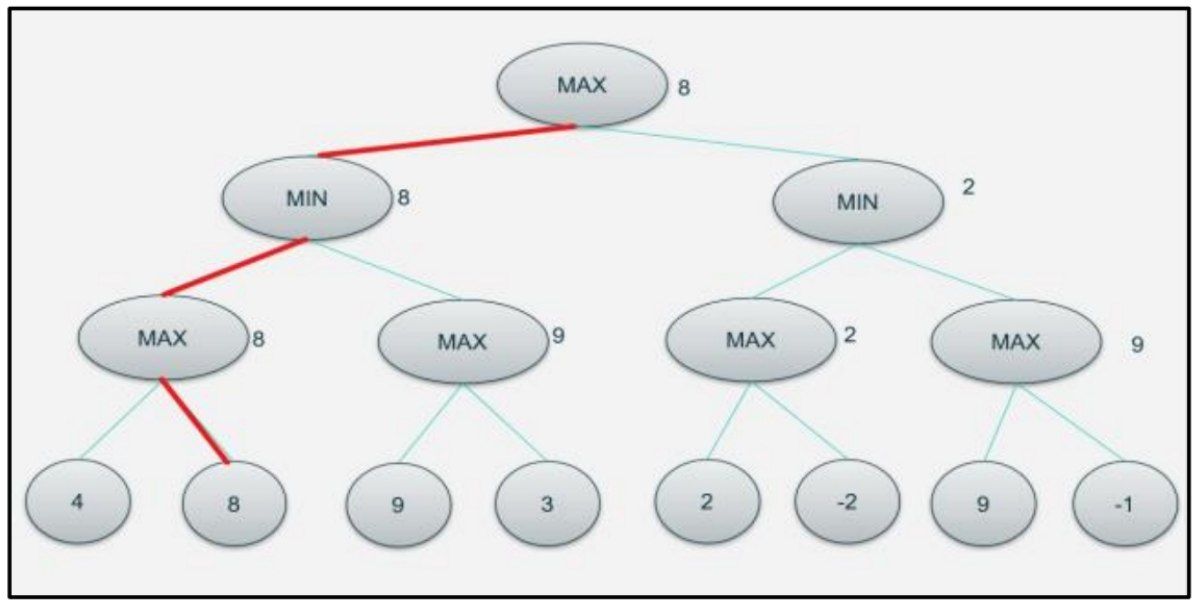
Initially, the algorithm generates the entire game tree and produces the utility values for the terminal states by applying the utility function. For example, in the below tree diagram, let us take A as the tree's initial state. Suppose maximizer takes the first turn, which has a worst-case initial value that equals negative infinity. Then, the minimizer will take the next turn, which has a worst-case initial value that equals positive infinity.



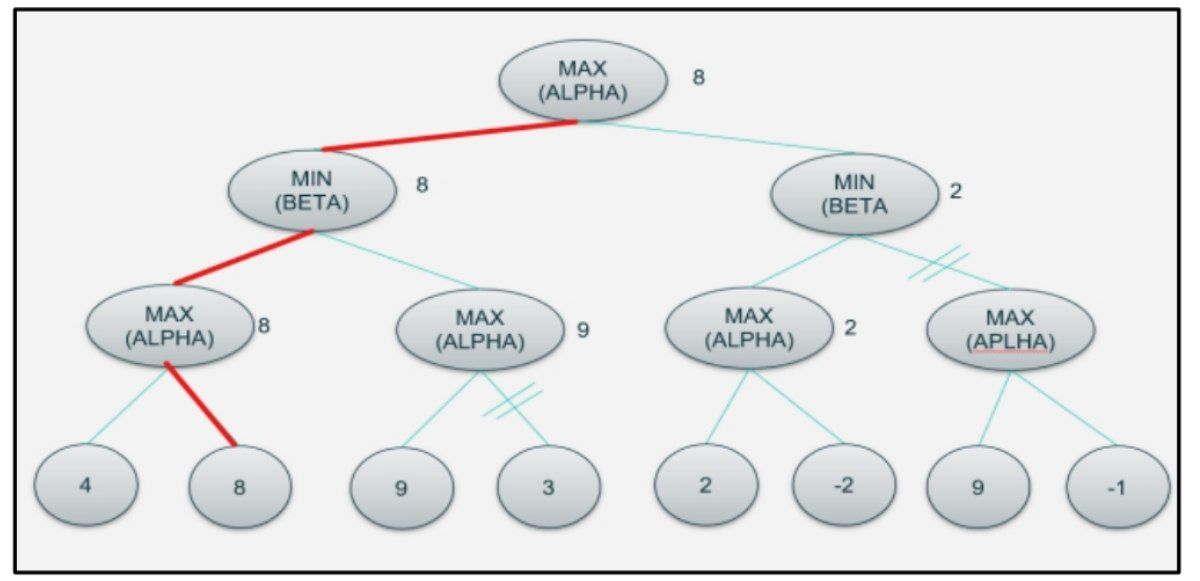
**in Connect Four**

**-------------------------------**

The algorithm that has been implemented into the Connect-4 game is classical Alpha-Beta pruning with MiniMax algorithm. Mini-Max is known as a backtracking algorithm where it will predict the next move and get the optimal move In a multiplayer game such as Connect-4, there are two players where each will be known as the maximizer and the opponent will be known as the minimizer. The maximiser will try to get a higher score as they could and the minimizer will do the exact opposite and obtain the lowest mark possible. As the depth increasing there will be more branches to be explored to get the optimal move which results to longer time taken. Hence Alpha-Beta pruning has been implemented into the Mini-Max Algorithm to optimize the algorithm. Alpha Beta pruning is the powerful version of Mini-Max algorithm where two more extra parameters will be added to the code which is known as Alpha and Beta . The implementation of Alpha Beta will help to significantly drop the searching time to get an optimal as not all branches will be explored. Pruning of subbranches of a particular node will be done when a better path has been explored . Which mean that the path that has been pruned would not be explored as there is no need for it since it would not change the top node result.



when the Mini Max Algorithm has been used each level top node will be given a label either it is a Maximizer or a Minimizer. It goes sequentially where if the top node is a Maximizer the below node will be the opposite and act as a Minimizer and it goes on. The Maximiser will try to obtain the highest value and Minimizer will obtain the lowest value. It will perform a depth-first algorithm and reach the final node to get value. When it performs the depth-first search the first value that will be captured is 4. Then the next value would be 8 and these two values will be compared to see which has the higher value which in this case is 8. Thus 8 will be passed and the max node. This will go on until the final node value has been explored and compared. Based on the values then a path will be formed wherein Fig 2. is the highlighted red path.



The Alpha Beta pruning does exactly the same as MiniMax Algorithm where each node will be categorised either as a maximizer or minimizer. The only difference is in AlphaBeta Pruning, not all nodes will be explored where since there is an extra parameter that has been passed which is Alpha and Beta. Alpha will store the highest value and beta will be storing the minimum value. As shown in Fig 2., there has been pruning occurred in a sub-branch. The pruning occurs as when 9 has been passed to the Alpha parameter it will be compared with the existing Beta parameter value which is 8 at the top node. Hence, when the Alpha value is higher than the Beta value there will be no need to explore the branch anymore as that path would not even be needed as it would not lead to the optimal move. Which result in no need for further checking of the nodes value and the branch will be prune. This will save a lot of time as right now the depth is not even deep enough so it is easier to visualise but in the actual game the search tree will be huge and long as the depth gets deeper making it to have more branches that need to be explored. When pruning occurs it would not need the system to check through the branch anymore. The highlighted red path is the optimal move which is the same as Fig 1. that uses the Mini Max algorithm where the same result is obtained but in a faster time which makes Alpha Beta Pruning much more powerful to be used in this game.

7) Development platform: -

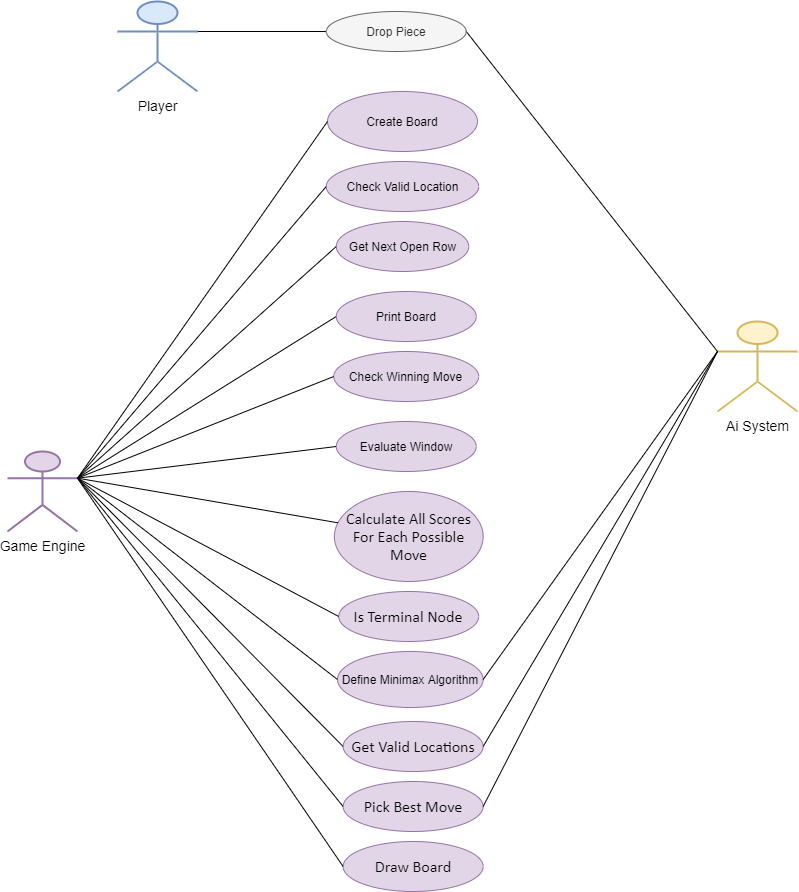
**Tools: Pycharm.**

**Programming Languages : Python .**

**Python Libraries :** numpy  
random  
pygame  
sys  
math

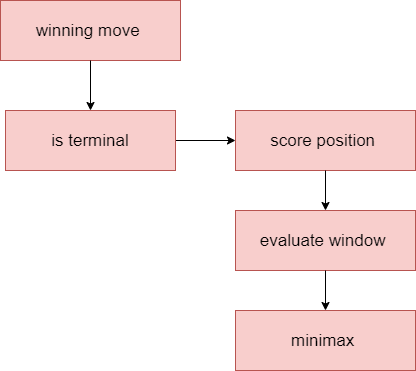
Extra Requirements: -

Use case Diagram: -

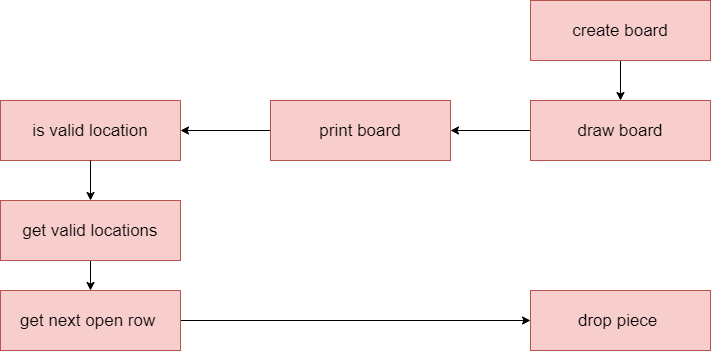


Block Diagram

- Minimax Algorithm Block Diagram: -



- Game(System) Block Diagram: -



Flowcharts

