The University of Nottingham

UNIVERSITY OF NOTTINGHAM

School of Computer Sciences

Designing Intelligent Agents (COMP3004)

Submitted: May 2024

Concept:

A Minimax AI in the game of Connect 4

Submitted by:

Nderon Beqiri (20365426)

Word Count:

4000

Contents

Introduction

What is an intelligent agent?

An intelligent agent is a system that perceives its environment and takes actions that maximise its chances of achieving its goals. These agents can range from being implemented via simple to complex algorithms, which our project looks into in order to see the discrepancy in their performances. They are designed to interact with their environment.

What is an environment?

An environment, refers to everything outside the agent that can potentially affect its behaviour and any other factors that might influence the agent's decision-making process.

The relationship between an agent and its environment is dynamic and reciprocal, with the agent continuously sensing and acting upon the environment in pursuit of its objectives. This interaction forms the basis of intelligent behaviour, as the agent adapts and learns from its experiences to better achieve its goals over time.

Connect 4- the agents and their environment

Connect 4 is a classic two-player strategy game where the objective is to be the first to connect four of one's own discs vertically, horizontally, or diagonally on a 7 column x 6 row grid. Players take turns dropping coloured discs into a vertically suspended grid, and the game ends when one player achieves four in a row *(as shown in Figure 1)* or when the grid is full with no player achieving four in a row, resulting in a draw *(as shown in Figure 2)*.

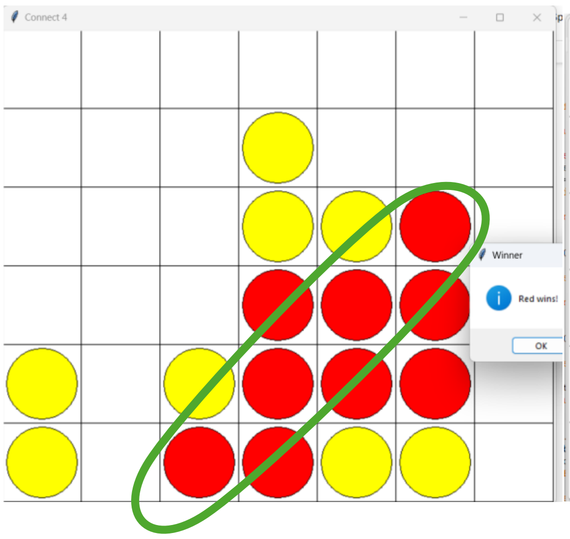

Figure 1: A game with a winner

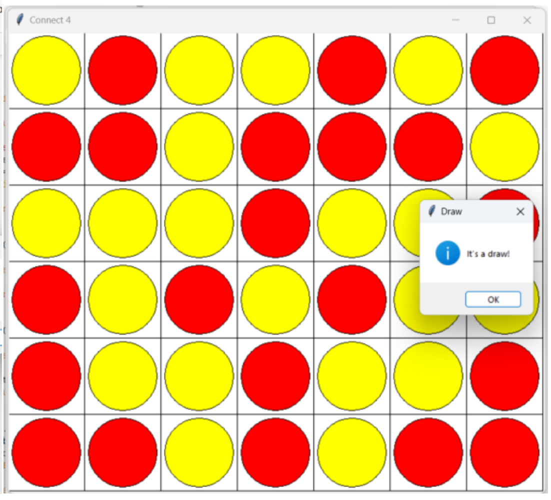

Figure 2: A draw (game with no winner)

It's a game that's easy to grasp but challenging to master, as it requires strategic thinking and planning ahead to outmanoeuvre your opponent. Due to the complex nature of the game, AIs have been developed to such a great extent that the game has been solved. Nevertheless, the game remains as a benchmark for testing ideas in artificial intelligence.

A brief summary of solving Connect 4

The game of Connect 4 was solved in 1988 by James D. Allen, who demonstrated that with perfect play, the first player can always win [1]. Allen employed a brute-force approach, analysing all possible game positions to determine the optimal move at each step. His analysis was a monumental computational feat, involving the examination of over 4.5 trillion positions. As part of his exhaustive analysis, Allen created an endgame database, which stored perfect play strategies for specific board configurations, aiding decision-making during gameplay.

Victor Allis independently solved the game just 15 days later, providing further validation of Allen's solution. Allis utilised a heuristic approach, developing a program that used knowledge-based strategies and heuristics to evaluate game positions and determine optimal moves. His program evaluated positions based on features such as the number of potential winning lines for each player, the presence of threats, and the potential for future moves to create multiple threats simultaneously. Allis's solution was notable for its efficiency in terms of computational resources required. Additionally, Allis contributed a mathematical proof demonstrating that under optimal play conditions, the first player can always win [2].

Other researchers, such as John Tromp, made significant contributions to the understanding of Connect 4. In 1995, Tromp extensively solved the game, providing outcomes for any 8-ply position within a database [3]. This work extended the understanding of the game beyond the endgame phase, providing insights into mid-game strategies.

Today, numerous online solvers exist where players can explore perfect play scenarios and see how a game will play out from any position [4]. Common strategies include the recognition that if the first player opens with a move adjacent to the middle column, the opposing player can always force at least a draw. Conversely, if the first player starts by playing a move in one of the four outer columns, the second player can force a win. These strategies underscore the depth and complexity of Connect 4, despite its seemingly simple rules.

Although these solutions exist, many researchers build AIs to see how they fare on solved games such as Connect 4. One example being Mark Winands, known for his contributions to the field of artificial intelligence, particularly in the realm of game theory, with a focus on Monte Carlo Tree Search (MCTS). He has explored techniques for enhancing the efficiency and effectiveness of AI algorithms in playing Connect 4 optimally. MCTS takes distant consequences of moves into account, giving it a strategic advantage in many domains over traditional depth-limited minimax search with alpha-beta pruning. However, MCTS builds a highly selective tree and can therefore miss moves and fall into traps. His work focuses on MCTS-minimax hybrids that employ shallow minimax searches within the MCTS framework [5].

This is simply one of many researchers who have applied their algorithms to Connect 4 to test its performance, with others experimenting with other ideas in artificial intelligence such as reinforcement learning- AlphaZero for Connect 4 [6], and more.

Upon doing all of this background research I learned that the most effective algorithm was minimax with alpha-beta pruning and wanted to challenge myself to see if I could implement it in Python taking inspiration from those who attempted this before me [7], but implementing it in my own way with a more efficient AI, better evaluation function, scoring system and the use of Tkinter.

What will our project be about?

Our project aims to answer the question **“can we develop an AI to consistently beat humans in a game of Connect 4 whilst balancing time and performance?”**.

How will we approach our goal?

We will delve into this question by beginning with simple agents in the environment of a Connect 4 board, such as an AI that plays random moves. This will extend to a greedy AI that implements methods of playing a winning move or blocking an opponent’s winning move. We will see how this fares against an AI that resorts to a heuristic strategy which prioritises the centre column and then evaluates other available moves based on their scores. This method assesses the current position on the board by considering the number of consecutive pieces in rows, columns, and diagonals, and will build on where there is the most.

An issue with all of these AIs is that they do not look ahead and view what will happen after their move, after the opponent’s move and so on. However we can implement a minimax AI with foundations found from these AIs such as prioritising a winning or blocking move as done by the greedy AI, or a more complex evaluation function that extends the heuristic AI and considers playing where there are currently more pieces in a row, but also other factors.

Connect 4 Environment- how was it designed?

The environment within our project is the Connect 4 board designed from scratch using the aid of online resources [8] and Tkinter. The environment allows humans to interact with the board, placing their own moves and altering the agent’s response. Similarly, if the human does not play, they are able to watch the AI play itself move after move and view the game's progress as the game state within the environment changes.

The main component of this window is the drawing surface for rendering the Connect 4 board. The Connect 4 board itself is represented as a grid of cells, mimicking the physical game's layout. Each cell can be either empty or contain a game piece, represented by circles. These game pieces come in two colours: red and yellow.

Players interact with the game by clicking on the columns of the board where they wish to drop their pieces. This interaction is managed through event bindings in Tkinter, which captures mouse clicks.

The game's logic, including checking for wins, handling turns, and validating moves, is implemented within the Python classes defined in the code. It adheres to the standard rules of Connect 4, where players take turns dropping their coloured pieces into columns, aiming to connect four of their pieces either horizontally, vertically, or diagonally.

The environment supports various game modes, including Human vs. Minimax AI, Minimax AI vs. Minimax AI, and Minimax AI vs. Human. Before starting the game, players can select the desired game mode through a user interface embedded within the GUI window. This interface comprises Tkinter widgets such as labels, dropdown menus, and buttons, enhancing the user experience by allowing easy configuration of game settings. In any of the modes if the AI player goes first, the user is able to choose the amount of random moves it plays, meaning that the higher the random moves the more likely the game state inside the environment is to be different from the beginning. As Connect 4 has a large number of outcomes, the moves played from start to finish is likely to never be the same.

In summary, the Connect 4 board environment provides a dynamic platform for AI research and development as it serves as a testbed for evaluating AI algorithms, in our case the Minimax algorithm with alpha-beta pruning, in the context of strategic decision-making and adversarial gameplay. The comparison of AI agents against each other or against human players in our diverse modes allows for the assessment of AI capabilities in both competitive and collaborative settings. As a result, the Connect 4 environment offers versatility for AI experimentation and analysis, enabling us to delve into the intricacies of AI decision-making and strategic planning within the context of a classic board game.

Minimax AI agent- how was it designed?

The AI that we will be looking at utilises the minimax algorithm with alpha-beta pruning to make intelligent moves in the game of Connect 4. This method prunes branches of the game tree that are deemed irrelevant, significantly reducing the computational load and improving the AI's efficiency.

The AI implements a maximum depth parameter to which the algorithm should search the game tree. This is the parameter we will be experimenting with to see if we can find an optimal depth that allows the AI to defeat humans whilst playing quickly when responding to human moves.

The AI considers various scenarios during gameplay, firstly checking for immediate winning moves and looks to block opponent's potential winning moves. If neither is found, it employs the Minimax algorithm with Alpha-Beta pruning *(as shown in the pseudocode section)* to efficiently navigate through the game's decision tree and select the most advantageous moves.

The evaluation of the game state is based on heuristics that assess the current board position for both the AI and its opponent, considering factors such as the presence of winning combinations and potential threats. Through analysing patterns of discs in rows, columns, and diagonals, the AI can assign scores to different board positions, aiding in its decision-making process. It assigns higher scores to states where the AI player is closer to winning and lower scores to states where the opponent has the advantage. By considering various possible moves and their potential outcomes, the AI ultimately selects the move that maximises its chances of winning or prevents its opponent from winning, representing the most favourable outcome for the AI player.

Unlike the simpler AIs, it considers both its own moves and the opponent's potential responses, ensuring a balanced strategy that anticipates counter-moves.

Overall, the AI offers a formidable challenge as it dynamically adapts its decision-making process based on the current game state, adjusting its strategy to optimise its chances of winning.

Challenges of designing the environment and agent

Designing the environment and agent for a game like Connect 4 presented several challenges. Firstly, it's crucial to represent the game state efficiently, devising a suitable data structure to manage the grid of cells and handle player moves effectively. Generating valid moves, essential for both human and AI players, required careful implementation to ensure adherence to game rules.

AI decision-making introduces another layer of complexity, demanding the design of algorithms capable of making intelligent moves within reasonable computational limits. Tuning parameters like search depth and evaluation functions was necessary to strike a balance between decision quality and resource efficiency.

Evaluating game states accurately to assess positions and potential outcomes posed a significant challenge, especially in assessing strategies such as blocking opponent moves and pursuing winning positions a few moves down the line.

Integrating game logic seamlessly with the user interface is also important, requiring careful synchronisation and event handling to maintain a smooth experience.

Lastly, thorough testing and debugging were essential to identify and address any bugs or inconsistencies, ensuring the AI behaves as required such as not losing within the first 4 moves if the user attempts to play 3 in a row at the bottom, allowing the next move to be played on either side if one is blocked. Note how in *figure 3* the weak AI was too late to block red's strategy but in *figure 4* it prevents red from creating 2 angles they can win from.

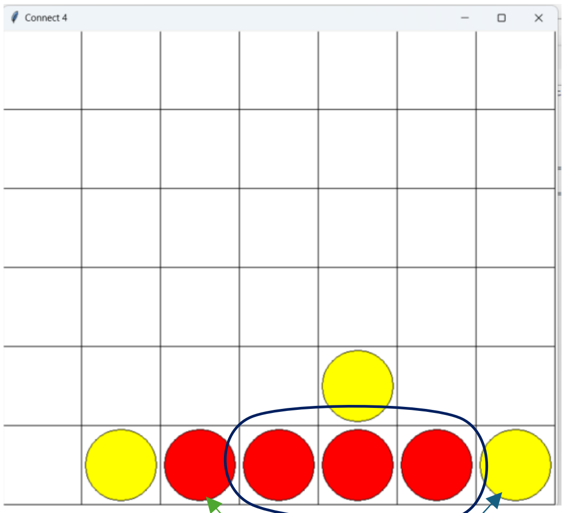

Figure 3: A weak AI that doesn't recognise early strategies

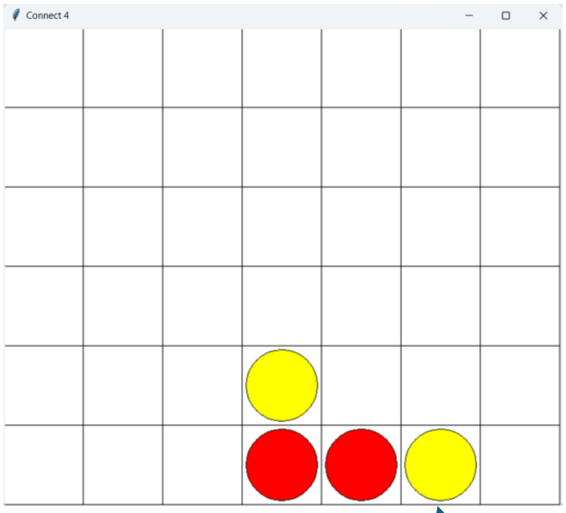

Figure 4: Minimax AI that counters early strategies

Blocked on 3rd move but human can now play here to win

Blocked on 2nd move, game continues

Pseudocode

```
function minimax(node, depth, maximizingPlayer, alpha, beta)
  if depth == 0 or node is a terminal node then
    return H(node)
  end if
  if maximizingPlayer then
    bestValue = -∞
    for each child of node do
      score = minimax(child, depth + 1, False, alpha, beta)
      max_score = max(max_score, score)
      alpha = max(alpha, max_score)
      if alpha >= beta then
        break
      end if
    end for
    return max_score
  end if
  else
    bestValue = +∞
    for each child of node do
      value = minimax(child, depth + 1, True, alpha, beta)
      min_score = min(min_score, score)
      beta = min(beta, min_score)
      if beta <= alpha then
        break
      endif
    end for
return min_score
```

Questions and Experiments

1) **At what depth is the Minimax AI able to consistently outperform human players?**
   In order to carry out this experiment, 5 human players will take turns to play a variety of games against the Minimax AI.
   The experiment will begin with the AI having a depth of 1.
   The players will play 2 games as player 1 and 2 games as player 2 for each depth.
   The depth of the AI will then be incremented by 1 and the player will play until the depth is too great for the AI to make quick moves.
   Results of each depth will be displayed.
   Human players will have no time limit to make moves.

2) **When is the depth demanding too much computational resources for machines to handle?**
   This experiment corresponds to experiment 1 where we will be able to get a maximum depth value that the AI can reach in a well-responsive amount of time.

3) **Does the Minimax AI perform better against humans as the first or the second player?**
   We will compare the results from experiment 1 to answer this question.
   See *figure 5*.

4) **What happens if the Minimax AI plays against itself?**
   As our AI is deterministic, we will implement a scheme where the first x moves are random for player 1.
   We will play 3 games for each random moves i.e. as we are going up to 3 random moves there will be 9 games in each set (Depth y vs Depth z).
   This will ensure that different games are played out as opposed to the same game with the consistent same outcome.
   We will also have the AIs play each other with the same and different depths to see how they fare against one another.

5) **Can any of the simple AIs beat the Minimax AI?**
   The AI will play a series of games against the greedy and heuristic AI up to reasonable depth, similar to experiment 1.
   This experiment will allow us to see if a greater depth make the AI more tactical and thus set up winning opportunities quicker with fewer moves.

Experiments 1 & 2 & 3

Depth = 1	W-D-L
AI as Player 1	8-5-7
AI as Player 2	10-2-8
Average Move Time	<0.02s

Depth = 2	W-D-L
AI as Player 1	12-1-7
AI as Player 2	12-3-5
Average Move Time	<0.03s

Depth = 3	W-D-L
AI as Player 1	14-2-4
AI as Player 2	15-2-3
Average Move Time	<0.2s

Depth = 4	W-D-L
AI as Player 1	15-2-3
AI as Player 2	17-1-2
Average Move Time	<0.7s

Depth = 5	W-D-L
AI as Player 1	16-1-3
AI as Player 2	18-0-2
Average Move Time	<3.0s

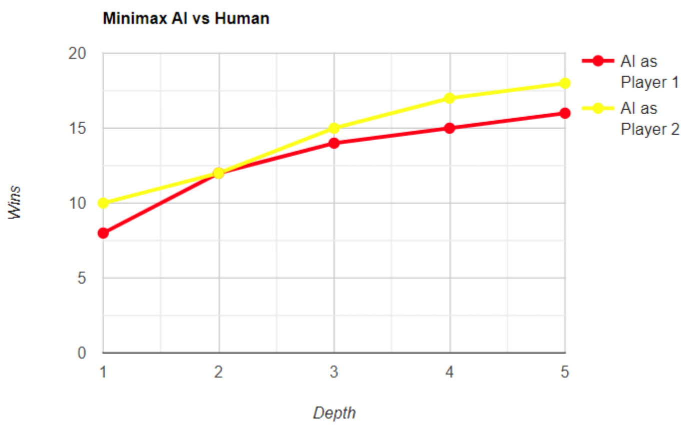

Figure 5: Results of Minimax AI vs Human

Moves started lasting over 10 seconds when depth was increased to 6 *(answer to question 2)*. Depth 5 would last only a few seconds and remained relatively playable. Depth 4 was quicker and the rest of the depths played moves near instantly.

However I did play a few games against Depth 6 to see if it plays optimal moves, comparing it to the "show solution" of the Connect 4 solver [4], and most of the time it did. It proved to be unbeatable to someone of my level.

We can see that as the depth increases so does the likelihood of the AI winning. The difference from depth 1 to depth 5 is immense as it is able to traverse more of the tree and produce better scores upon evaluating. If looking for a competitive and well responsive depth we see our program provides this at depth 4 *(answer to question 1)*. If looking for a challenge and a slightly slower response, then our program provides this at depth 5.

We also see through *figure 5* that the AI performs better as the second player. This is perhaps due to the fact that we made the first move the AI random when it was player 1 for more variety leading to more calculations *(answer to question 3)*. Of course referring back to Connect 4 being solved, with a higher depth and more computational power, this would likely not be the case.

Experiment 4

The minimax AI played itself in the form of sets:

Depth 1 vs Depth 1, Depth 1 vs Depth 3, Depth 1 vs Depth 5

Depth 3 vs Depth 1, Depth 3 vs Depth 3, Depth 3 vs Depth 5

Depth 5 vs Depth 1, Depth 5 vs Depth 3, Depth 5 vs Depth 5

where each set consisted of 3 games with 1 random move, 3 games with 2 random moves and 3 games with 3 random moves. Below are the results.

	Random Moves (Player 1 Only)	Score
D1 vs D1	1	2-1
	2	1-2
	3	0-3
D1 vs D3	1	1-2
	2	1-2
	3	1-2
D1 vs D5	1	0-2 1 DRAW
	2	0-3
	3	0-3
D3 vs D1	1	2-1
	2	3-0
	3	2-1
D3 vs D3	1	2-1
	2	2-0 1 DRAW
	3	0-3
D3 vs D5	1	1-2
	2	1-2
	3	0-3
D5 vs D1	1	3-0
	2	3-0
	3	2-1
D5 vs D3	1	2-0 1 DRAW
	2	2-1
	3	1-2
D5 vs D5	1	2-1
	2	1-1 1 DRAW
	3	0-3

Looking at the results of minimax playing against itself we see how much of a negative impact introducing randomness on the first player has (as expected). The only time it performs well with several random moves is when it has a distinct advantage in depth. With minimal randomness i.e. only the first move of the game, we see that the AI with the greater depth is more likely to win but it is not always the case.

When the AIs are the same depth with low randomness we can see that player 1 tends to perform better as it has the advantage of dictating the game.

Also in this experiment, the AI was moving quicker than it did when playing against human players, likely due to the fact it anticipated the move the opponent would play whereas humans are more unpredictable *(links to question 2)*.

Experiment 5

Here we will be playing the minimax AI against the greedy AI that only looks to play a winning move or a block an opponent's winning move, and then against the heuristic AI that prioritises central control and then placing pieces where there are the most in a row, column or diagonal.

It will play 5 games as player 1 against both (starting with 2 random moves), and 5 games as player 2 against both.

Below are the results of the games with Minimax AI having a depth of 1.

vs Greedy AI	W-D-L
AI as Player 1	5-0-0
AI as Player 2	5-0-0

vs Heuristic AI	W-D-L
AI as Player 1	5-0-0
AI as Player 2	5-0-0

As expected the Minimax AI won every game *(answer to question 5)* as it implements the positive aspects of both AIs however adds to them with more complex evaluation functions. As a result, even with the lowest depth possible, simple AIs prove no challenge for our implementation of the Minimax AI. I did some further experiments and increased the depth to watch how the games would play out, and it led to minimax winning more efficiently in less moves, setting up more strategies and winning combinations.

What do the experiments tell us?

Overall from the results of our experiments we can see that the higher the depth of the AI, the higher the challenge towards the opponent, but at a cost of a greater time to play each move. We also see that against humans, the AI tends to perform better as player 2 but when playing against itself it is a lot more competitive as player 1. The more random moves that the AI opens up with, the less likelihood it has of winning, however this was necessary for our experiments as this is a deterministic AI and we did not want repeat results.

Additional observations during experiments

Watching the AI play at a low depth I noticed an inefficiency when it came to strategy. It was able to win games quicker but prolonged these wins. Whereas when depth was increased, the AI would set up tactics and traps, forcing players to play a certain move even though the AI would win on the next one (as shown in the video).

Furthermore, the reason the AI performs so well against humans is due to it not having "blind spots". When playing against the AI myself, or watching others, once every few games the human would lose by simply not blocking the AI's winning move as they would not have noticed it. This does not happen with the AI as it is coded to prioritise these moves before it goes to alpha-beta pruning as it is always the best move to play since it stops you from losing on the next turn.

Addressing our main question through the experiment findings

Through our experiments we can now answer the following question our project was set out to answer: **"can we develop an AI to consistently beat humans in a game of Connect 4 whilst balancing time and performance?"**. The answer is simply yes. Although it was a complex to develop AI with many algorithms and different evaluation metrics, it performs at an efficient level that most computers should be able to handle at lower depths. Even the minimal depth that plays near instant moves would tend to marginally outperform human players *(as shown through experiment 1)*, and as the depth increased, it became more clear that most humans stood little chance against the AI we developed.

Conclusion

In conclusion the research from our project shows we are able to create an AI that can play at a high level against humans without sacrificing neither time nor performance. It has proven to be a great success as through our experiments, we can see that between our agent and the humans that played it, the agent is the stronger player.

It would have been interesting to see the AI play at even greater depths but limitations of computational power prevented this. As depth increased it played with more efficient strategies and would tend to win in less moves, so an additional increase in depth would likely continue the trend.

Given more time I would perhaps implement other methods of evaluation to see which is most beneficial to the AI. Furthermore I would develop more AIs capable of playing Connect 4 and have them play against each other to see which is best, and have them face humans to maintain consistency throughout our experiments.

Video- example of me playing a game vs the AI

Note- should be able to run the code (Minimax AI from Moodle) and test it out yourself (common libraries used)

References

[1] Connect Four [WWW Document], n.d. URL https://fabpedigree.com/james/connect4.htm

[2] Allis, V., 1988. A Knowledge-Based Approach of Connect-Four: The Game Is Solved: White Wins. ICGA J. 11, 165–165. https://doi.org/10.3233/ICG-1988-11410

[3] John’s Connect Four Playground [WWW Document], n.d. URL https://tromp.github.io/c4/c4.html

[4] Connect 4 Solver [WWW Document], n.d. . Game Solver. URL https://connect4.gamesolver.org/en/

[5] Baier, H., Winands, M.H.M., 2013. Monte-Carlo Tree Search and minimax hybrids, in: 2013 IEEE Conference on Computational Inteligence in Games (CIG). Presented at the 2013 IEEE Conference on Computational Inteligence in Games (CIG), pp. 1–8. https://doi.org/10.1109/CIG.2013.6633630

[6] Kim, L., Godrej, H., Nguyen, C.T., n.d. Applying Machine Learning to Connect Four. https://cs229.stanford.edu/proj2019aut/data/assignment_308832_raw/26646701.pdf

[7] connect_four/ai_agent.py at master · AhmedEssammm/connect_four [WWW Document], n.d. . GitHub. URL https://github.com/AhmedEssammm/connect_four/blob/master/ai_agent.py

[8] Connect-4-Tkinter/connect4module.py at master · saiduc/Connect-4-Tkinter · GitHub [WWW Document], n.d. URL https://github.com/saiduc/Connect-4-Tkinter/blob/master/connect4module.py