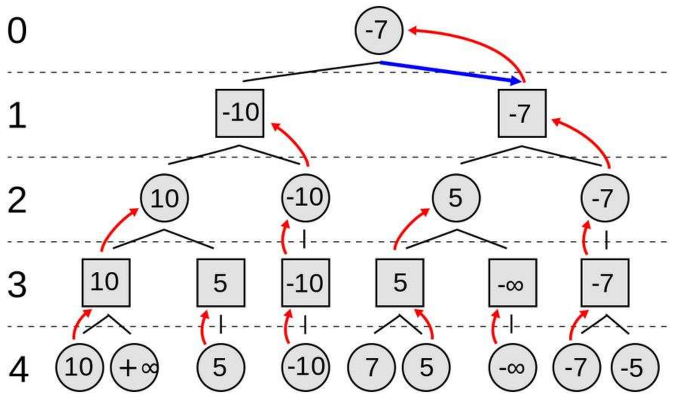
1. Introduction

This homework assignment requires us to develop advanced adversarial search algorithms we have learned in the course for the Connect Four game. The primary objective is to implement and evaluate two types of agents: Minimax and Alpha-Beta Pruning. Ultimately, we are expected to apply the techniques we have learned, along with the provided homework specifications, to develop a Stronger agent with a better heuristic function that outperforms the Alpha-Beta Pruning agent.

2. Implementation

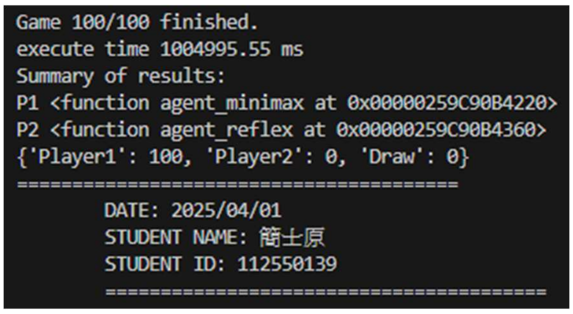
2.1. Minimax:

- **Algorithm Explanation:**
  - **Recursive Exploration:** The game tree represents all possible game states, which the Minimax algorithm constructs and explores recursively to determine the optimal move. In this assignment, the algorithm expands the game tree up to a depth of 4. At each level, it alternates between the maximizing player and the minimizing player, simulating the alternating turns of the game.
  - **Move Selection:** For maximizing nodes, the highest evaluated score move is selected; for minimizing nodes, the lowest evaluated score move is selected.



  - **Heuristic Evaluation:** When the search reaches a terminal state or the maximum depth, the board is evaluated using the get_heuristic(board) function. This function assigns a score to the current board state. The score is assigned by a predefined evaluation formula (e.g., an instant win will score a massive number, and 3 or 2 connected pieces will score less but relatively large compared to a single piece), and the sign will be positive for the maximizing player and negative for the minimizing player.

- **Results & Evaluation:**

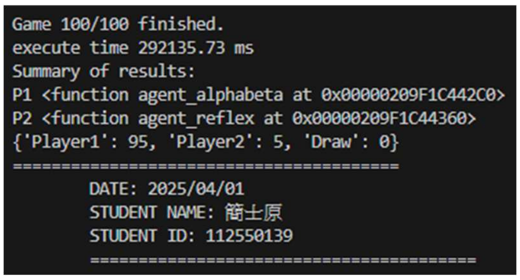


```
Game 100/100 finished.
execute time 1004995.55 ms
Summary of results:
P1 <function agent_minimax at 0x00000259C90B4220>
P2 <function agent_reflex at 0x00000259C90B4360>
{'Player1': 100, 'Player2': 0, 'Draw': 0}
========================================
        DATE: 2025/04/01
        STUDENT NAME: 簡士原
        STUDENT ID: 112550139
        ========================================
```

2.2. AlphaBeta

- **Optimization Explanation:**
  - **How α-β pruning reduces unnecessary node expansions:**
    The algorithm maintains two parameters: alpha (the best score of the maximizer) and beta (the best score of the minimizer). When beta ≤ alpha, it means the current branch can no longer yield a better value for the parent node. Therefore, the algorithm stops exploring this branch, which is known as pruning, to reduce unnecessary node expansions.
  - **When and how pruning occurs in your implementation:**
    In my implementation, alpha and beta are passed as function parameters with initial values set to -inf and inf. These 2 parameters are updated as the recursions and iterations progress (alpha increases and beta decreases when they find a better score). When beta ≤ alpha, the loop that explores the remaining child node (possible game states) will break, and the function will return the best score so far.

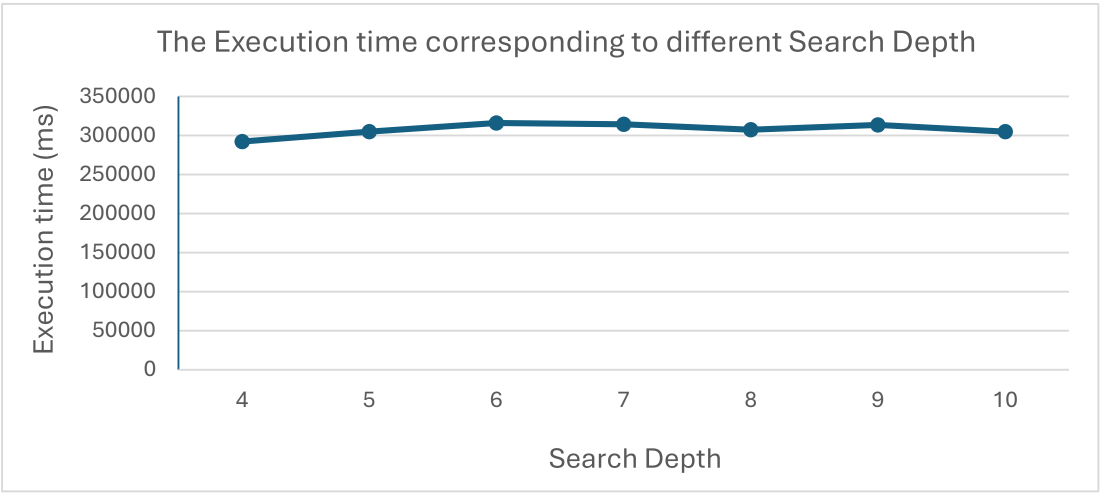
- **Results & Evaluation:**
  - **Execution time of Minimax vs. Alpha-Beta:**
    The execution time of Minimax is 1004995.55 ms, while that of Alpha-Beta is 292135.73 ms. This shows that Alpha-Beta pruning is approximately 3 times more efficient than Minimax. This improvement is attributed to pruning, which eliminates many unnecessary branches, allowing the algorithm to explore the game tree more efficiently.



```
Game 100/100 finished.
execute time 292135.73 ms
Summary of results:
P1 <function agent_alphabeta at 0x00000209F1C442C0>
P2 <function agent_reflex at 0x00000209F1C44360>
{'Player1': 95, 'Player2': 5, 'Draw': 0}
========================================
        DATE: 2025/04/01
        STUDENT NAME: 簡士原
        STUDENT ID: 112550139
        ========================================
```

  - **Execution Time vs. Search Depth:**

As the plot displays, although the game tree grows exponentially with increasing depth, the execution time remains relatively stable for Alpha-Beta. This is because pruning significantly reduces the number of nodes that need to be explored.

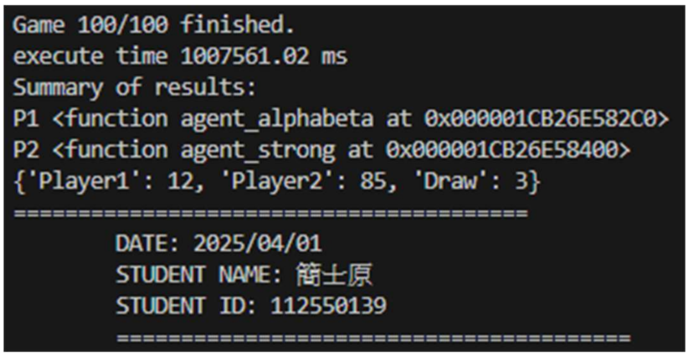


2.3. Strong Agent

- **Techniques Used:**
  Alpha-Beta pruning forms the foundation of my agent. In addition, move ordering is implemented within the algorithm. Specifically, the algorithm evaluates all possible moves using a heuristic, then sorts and stores these evaluations in a list for later iteration. This ensures that the most promising moves are explored first, leading to more efficient pruning and allowing the algorithm to search deeper without excessive computation time. Furthermore, the heuristic function has been modified (as described in the next section) to produce more accurate evaluations, which in turn helps the strong agent to better identify favorable moves.
- **Advanced Heuristic Function Evaluation:**
  The get_heuristic_strong() function improves move evaluation over the basic get_heuristic() by:
  - **Prioritizing Forced Wins:** If an immediate win is detected, the recursion is terminated, and infinity is returned.
  - **Defensive Play:** The function penalizes board states where the opponent has immediate winning moves.
  - **Board Control:** A position matrix is constructed to assign a favorable score to each board position based on its potential for forming a 4-connected combination (dominance). This matrix is then incorporated into the final evaluation.

- ○ **Revaluation:** The weights of the board state are adjusted to more precisely evaluate the overall score, enhancing the importance of connected pieces (score for 3 and 2 connected pieces increase) while also considering positional advantages.

- **Results & Evaluation:**



```
Game 100/100 finished.
execute time 1007561.02 ms
Summary of results:
P1 <function agent_alphabeta at 0x000001CB26E582C0>
P2 <function agent_strong at 0x000001CB26E58400>
{'Player1': 12, 'Player2': 85, 'Draw': 3}
========================================
        DATE: 2025/04/01
        STUDENT NAME: 簡士原
        STUDENT ID: 112550139
        ========================================
```

3. Analysis and Discussion

- **What were the difficulties in designing a strong heuristic?**
  1. **Weighting Different Conditions:** It's hard to determine the relative importance of various conditions. For example, while 2 connected pieces might seem insignificant initially, they can be crucial for future wins. Conversely, 3 connected pieces might not always be valuable if surrounding positions are blocked. The position is also crucial for evaluation since the central part of the board can contribute more possibilities than the marginal part. Since the final score is a summation of all these factors, accurately weighing each aspect is a challenging task.
  2. **Complexity of Considerations:** To precisely evaluate various board state features, the heuristic must consider their future potential, such as its positional advantages or disadvantages. This often requires implementing multiple nested loops and even recursive evaluations to examine every possible threat or opportunity. For example, if the algorithm wants to determine whether certain 3 connected pieces are valuable or not, it needs a multiple layers nested loop to check whether it's blocked or not. Such complexity can lead to high computational costs.
  3. **Balancing Efficiency and Accuracy:** A more detailed and accurate heuristic generally requires more computational resources (time, memory, etc.). However, efficiency is crucial for implementation. Therefore, striking a balance between efficiency (efficient enough to allow deep search and real-time play) and performance (accurate enough to guide decision making) is a major design challenge.

- **Did agent_strong() have any weaknesses?**
  Yes, agent_strong() does have several potential weaknesses. It takes longer to compute than a standard Alpha-Beta agent due to its advanced heuristic function and move ordering, which increase computation time. Moreover, its performance is highly dependent on the precise tuning of the heuristic; if the parameters are not well-fitted, it may misjudge board states. Finally, the heuristic can be overly specialized, potentially overfitting to certain board scenarios and failing to adapt effectively to unusual or complex situations.

4. Conclusion

In this assignment, I implemented 3 types of agents for Connect Four: Minimax, Alpha-Beta Pruning, and a self-designed Advanced Strong Agent. Through this process, I gained hands-on experience writing these agents in Python and developed a precise understanding of how each algorithm works. I became more familiar with heuristic functions by designing and tuning my own, which enhanced my insight into how evaluation functions impact decision-making. Additionally, I implemented techniques such as move ordering in my strong agent, further improving its performance. Overall, the first two parts of the assignment enhanced my implementation skills, while the strong agent part demonstrated that subtle modifications in heuristic design and optimization techniques can have a profound effect on performance.

5. References:

Sebastian Lague. (2018, April 20). *Algorithms Explained – minimax and alpha-beta pruning* [Video]. YouTube. https://youtu.be/l-hh51ncgDI?si=_EMnevCE7BwxMJ6u

Wikipedia contributors. (2025, March 24). Minimax. In Wikipedia, The Free Encyclopedia. Retrieved 08:28, April 3, 2025, from https://en.wikipedia.org/w/index.php?title=Minimax&oldid=1282084908

Kang, X.Y., Wang, Y.Q. and Hu, Y.R. (2019) Research on Different Heuristics for Minimax Algorithm Insight from Connect-4 Game. Journal of Intelligent Learning Systems and Applications, 11, 15-31.

ChatGPT was used for linguistic refinement.