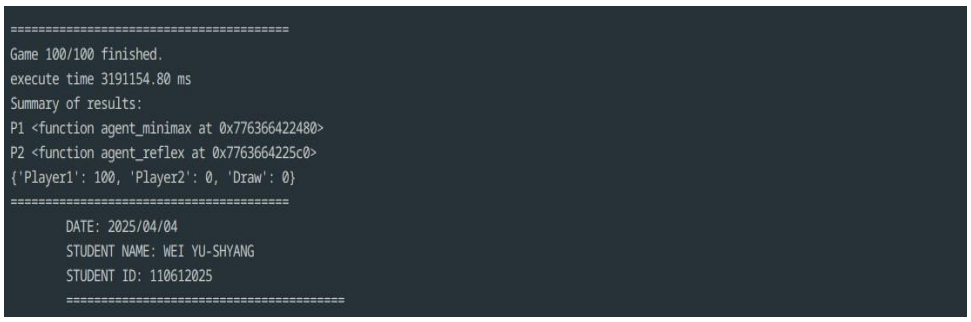
- Part I. Introduction

  HW2 is to develop and implement advanced adversarial search techniques for the game "Connect four". My goal is to build a intelligent agent that are capable to evaluate board position, planning ahead and making optimal moves by implementing classical search algorithm.

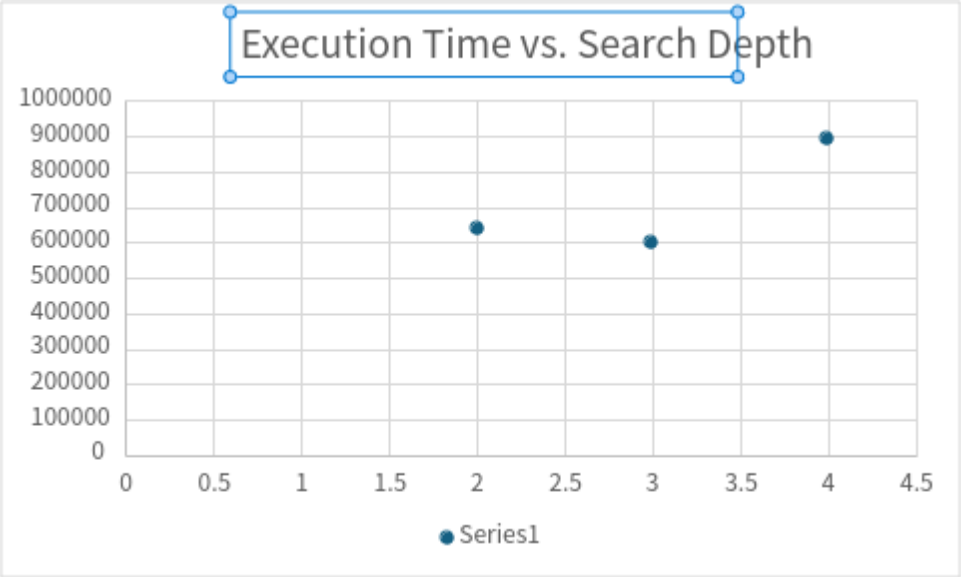
- Part II. Implementation
  - Part 1. Minimax search
    - It keeps exploring the every possible moves for each depth, and will return the heuristic value when the depth is 0 or the result is determined, from the heuristic value we can know which move is best or worst for the AI.
    - Minimax key idea is to assume the opponent will choose the best step for themselves, that is the worst for us. So when it is Maximizing Player's turn we choose the maximum value in the child node. When it is opponent's turn they will choose the minimum value in the child node.
    - "*get_heuristic(board)*", this function help us evaluating what's the situation in current board, give us a standard to know this situation is good for us or opponent.
    - 

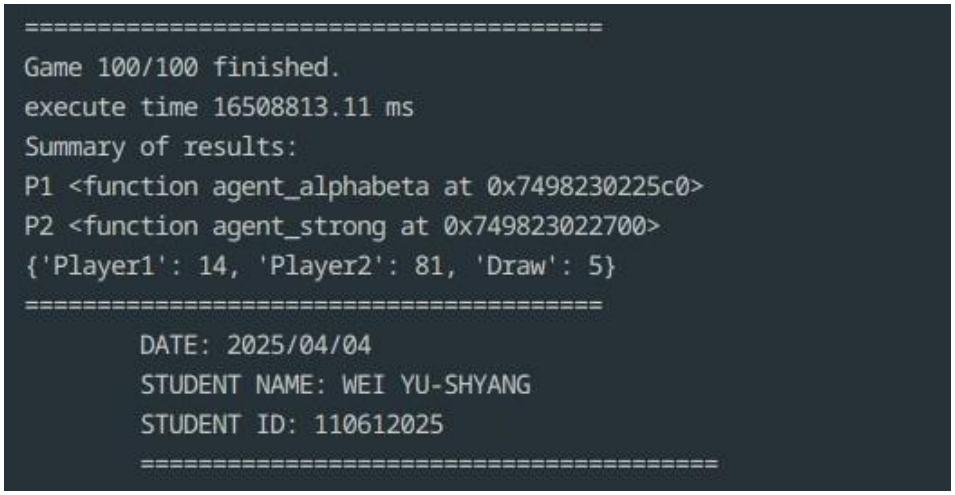
  - Part 2. Alpha-Beta Pruning
    - For each node, we give two additional information alpha and beta, alpha representing the maximum lower bound and beta representing the minimum upper bound. Once we make sure alpha >= beta, then we don't need to check the child node, since there's no possibility to be the best move.
    - In my code, alpha update when we are in the maximum layer, alpha is chosen the bigger value between itself and the highest value in the child node, on the other hand , beta update when we are in the minimum layer,

beta is chosen the smaller value between itself and the smallest value in the child node, once alpha >= beta, we don’ t need to explore that branch deeper, so we break.

-
```
========================================
Game 100/100 finished.
execute time 894578.38 ms
Summary of results:
P1 <function agent_alphabeta at 0x72a62f826520>
P2 <function agent_reflex at 0x72a62f8265c0>
{'Player1': 98, 'Player2': 2, 'Draw': 0}
========================================
        DATE: 2025/04/04
        STUDENT NAME: WEI YU-SHYANG
        STUDENT ID: 110612025
        ========================================
```

In minimax search time cost is 3191154.80ms, in alpha-beta pruning, time cost is 894578.38ms

- 

- Part 3. Stronger AI Agent(agent_strong())
  - In ***my_function(),*** I make the improvement based on the alpha-beta pruning, the difference is in following.
    - If the next move can win immediately then return the infinite value, and the move that can lead us to win, same way also works for opponent.
    - Set the order of the exlporing node based on the score of all the possible move, when it' s in my turn we choose the maximum score column, when it' s in opponent' s turn, they choose the minimum column to minimize my score.
  - The reason why it improves more evaluation then the older one is that we give certain strong move higher weight in the new evaluation function, for instance add the double threats move to encourage the player do the move that can make two "num threes" at the same time, or like block the opponent when their next move cause their winning, etc.
  - The way it counter alpha-beta is to priortizing
    - Winning potential: make the next move win have 1e8 weight
    - Defensive play: When the opponent' s next move will win, make the weight be - 1e8
    - Board control: I count the how many my coin are in the center bottom of the whole board, count the number and give it the weight of 1e3
  - 

  - Winning rate against alpha-beta pruning is 81%.

- Part III. Analysis & Discussion
    - What were the difficulties in designing a strong heuristic?

      Ans: How many weight should we give to the new add element like immediate win, center the element, if we prefer to give more weight to different move, it might cause total different move preference for AI to choose the next step.
    - Did agent_strong() have any weaknesses (e.g., high computation time, failure in some cases)?

      Ans: Yes, the biggest weakness is the high computation time, since we have to calculate the value of all the valid move to determine which branch we should explore next, it cost 2 times time than the original alpha-beta pruning.
    - How could agent_strong() be further enhanced?

      Ans: For further future, we can add different weight for each column in the chess board to get more accurate preference to the next step etc.

- Part IV. Conclusion

  Minimax algorithm help us mimic the whole possibility of the game, in our turn we always choose the maximum value, but in the opponent's turn they try to minimize or score, but it cost too much time to go through whole tree, so we use alpha-beta pruning to enhance the searching efficiency. In order to design the stronger agent to beat the alpha-beta agent, we revise the heuristic function by adding strong move more weight, like centering the piece or find the potential of creating double threats, and I improve the original alpha-beta pruning by calculating all the valid next step move, and in my turn explore the highest value move first, in opponent's turn, explore the lowest value first, but this cause loads of time, so maybe in the near future, there might be some algorithm to enhance the time usage.

- Part V. Reference
    - Minimax Algorithm introduction
    - Alpha-Beta pruning
    - Move Ordering in Search Algorithms