Homework 2: Connect Four

JUN-KAI CHEN, 111550001

April 1, 2025

1 Introduction

In this report, I applied different algorithm to the connect four game agent, testing their winning rate and efficiency. The performance of all the implemented algorithms is also recorded.

For reference, the experiments were tested on my computer. The processor is Intel i5-12600KF, with 32 GB RAM. The software environment is Python 3.10.6, numpy 2.2.4, and pygame 2.6.1 .

2 Implementation

There are three algorithms implemented in this homework. Basically, an algorithm should be consist of two part: evaluation and search function.

2.1 Minimax Search

In minimax algorithm, the provided heuristic function is used. I only implemented the search algorithm.

2.1.1 Explain the algorithm

Minimax algorithm is based on recusive function. The first step is always checking for termination. If the depth of recursion reaches bottom, the function needs to return heuristic value which is provided `get_heuristic(board)` function. This function quickly calculates the score of a column based on the given board argument.

For normal situation, the selection of minimax algorithm depends on its own identity. As a player, it should maximize the score as much as possible. As an opponent, it should minimize the score. The score here originally comes from `get_heuristic(board)` function, and selected by each recursion.

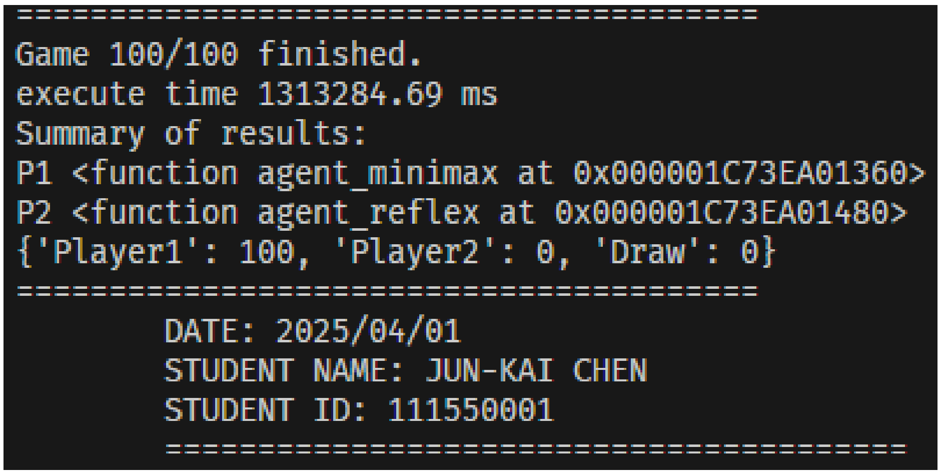
In my design, the minimax algorithm goes through all the valid columns and get each of the best score and corresponded moves.

2.1.2 Results & Evaluation

The result of 100 games is shown in Figure 1. Thought the winning rate of minimax algorithm is 100% in the figure, I have tested it repeatedly on my computer several times and there are still very few cases where the minimax algorithm loses the game. However, the overall winning rate is close to 100%.

2.2 Alpha-Beta Pruning

Alpha-Beta Pruning algorithm is based on minimax algorithm, which was introduced in last section.



```
========================================
Game 100/100 finished.
execute time 1313284.69 ms
Summary of results:
P1 <function agent_minimax at 0x000001C73EA01360>
P2 <function agent_reflex at 0x000001C73EA01480>
{'Player1': 100, 'Player2': 0, 'Draw': 0}
========================================
        DATE: 2025/04/01
        STUDENT NAME: JUN-KAI CHEN
        STUDENT ID: 111550001
        ========================================
```

Figure 1: The result of 100 game with minimax agent.

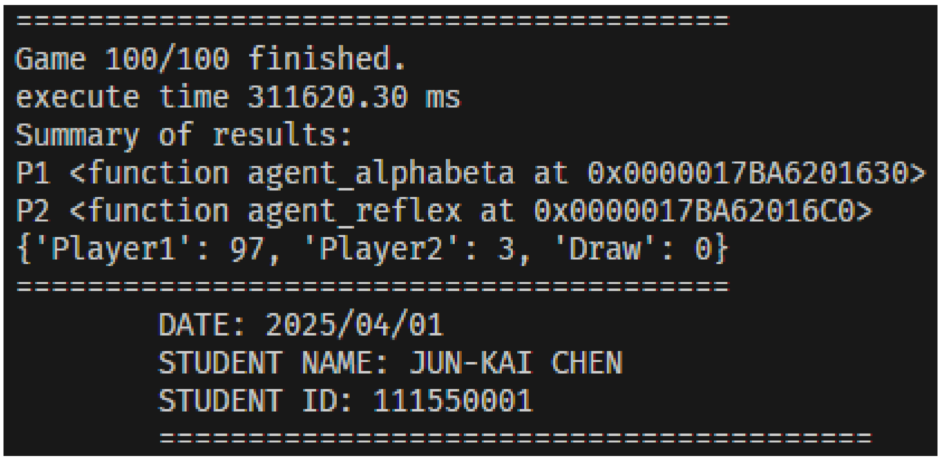
2.2.1 Explaing the algorithm

The α-β pruning is a method to decrease the expansions of unnecessary node. Its core is still based on minimax algorithm. The main difference from the former is that when the program traverse all valid nodes and obtain their scores, it updates the alpha or beta value according to its identity.

When it plays as a player, it sets α to the maximum score that can be obtained by recursion, which means that these branches can achieve the best result so far. When $\beta \leq \alpha$, the opponent won't choose this path, so it can be pruned directly. And as a opponent, β is set to the minimum value, representing the worst possible outcome for the player. Similarly, when $\beta \leq \alpha$, the player won't allow this path to occur, so pruning can also be performed.

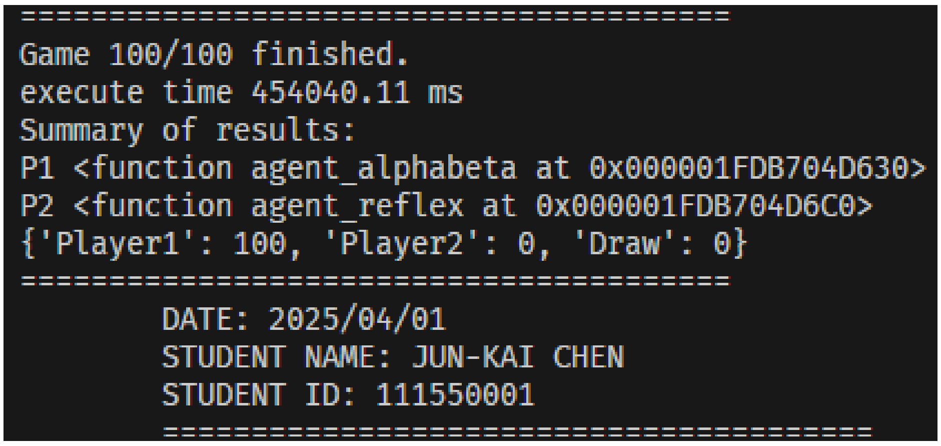
2.2.2 Results & Evaluation

When I implemented this function, I found that the winning rate of α-β pruning couldn't reach 100% sometimes. I found the answer at the Notion page, which was discussed by Q.10. If I use $\beta \leq \alpha$ as the criteria for judging whether to prune, some of the valuable branches might be pruned and results to a wrong answer. The results of two pruning strategies are shown in Figure 2 and 3.



```
========================================
Game 100/100 finished.
execute time 311620.30 ms
Summary of results:
P1 <function agent_alphabeta at 0x0000017BA6201630>
P2 <function agent_reflex at 0x0000017BA62016C0>
{'Player1': 97, 'Player2': 3, 'Draw': 0}
========================================
        DATE: 2025/04/01
        STUDENT NAME: JUN-KAI CHEN
        STUDENT ID: 111550001
        ========================================
```

Figure 2: The result of 100 games with α-β pruning (use $\beta \leq \alpha$ as criteria).



```
========================================
Game 100/100 finished.
execute time 454040.11 ms
Summary of results:
P1 <function agent_alphabeta at 0x000001FDB704D630>
P2 <function agent_reflex at 0x000001FDB704D6C0>
{'Player1': 100, 'Player2': 0, 'Draw': 0}
========================================
        DATE: 2025/04/01
        STUDENT NAME: JUN-KAI CHEN
        STUDENT ID: 111550001
        ========================================
```

Figure 3: The result of 100 games with α-β pruning (use $\beta < \alpha$ as criteria).

As can be seen in the two figures, the execution time of algorithm has been decreased with the pruning. Compared with the unoptimized minimax algorithm, the efficiency of the two pruning methods is 2.89 and 4.38 times that of the original ones respectively. The difference between the two methods is 1.51 times. However, the efficiency is also reflected in the accuracy of the two. The more rigorous pruning method (the version that doesn't use equality) has a better win rate, but is much slower. Based on the answer of Q.33 on the Notion page, $\beta \leq \alpha$ is used as criteria.

It's important to note that the execution time of the entire experiment is closely related to the works that the computer is currently processing. If a computer is processing other task during the experiment, such as writing a report, downloading files from the Internet, etc. , the execution time will be longer. The figures I provided are designed to allow the computer to process only the experiment as much as possible.

2.3 Stronger AI Agent

2.3.1 Techniques Used

I used the original `agent_strong()` function and α-β pruning as decision making.

While I was testing the performance of different searching algorithm such as MCTS against the original α-β pruning, I found that most algorithm can't defeat it easily. In theory, the minimax algorithm which is used by player 1 always finds a best choice if the depth of recursive is deep enough. The advantages player 2 can take is the heuristic value. Thus, I decided to stick on minimax and modify the evalution function.

There are a lot of websites, videos, or even article on the Internet that teach you how to implement an AI agent for Connect Four based on minimax. Some of them gave good ideas, such as using an 2-dimension array as weights to calculate heuristic value. I eventually chose a method introduced in an article: "Research on Different Heuristics for Minimax Algorithm Insight from Connect-4 Game" [1].

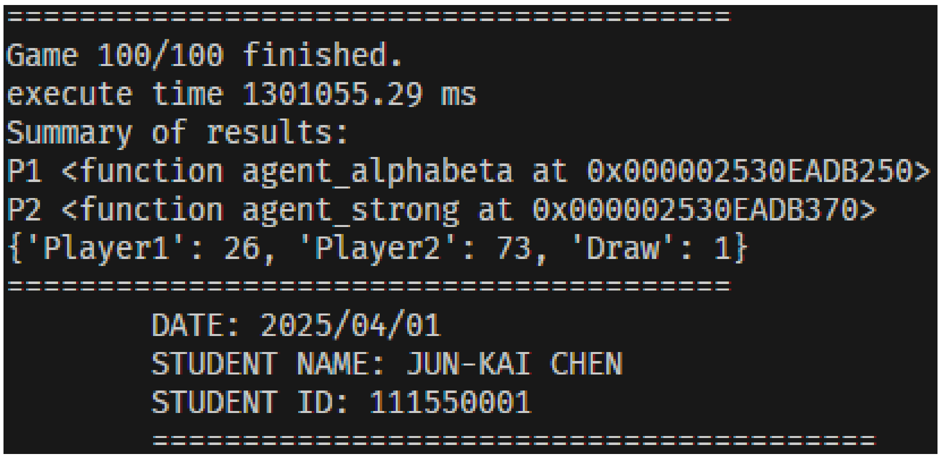
2.3.2 Advanced Heuristic Function

Following the article, I tried to take the number and position of individual pieces into consideration. Compared to the original function, it is not only more detailed, but also can weight the center according to the column where the chess piece is located.

2.3.3 Results & Evaluation

To prove the `get_heuristic_strong()` function is efficient, I need not only the result of α-β pruning and strong agent, but also the result of two players use the same strategy.

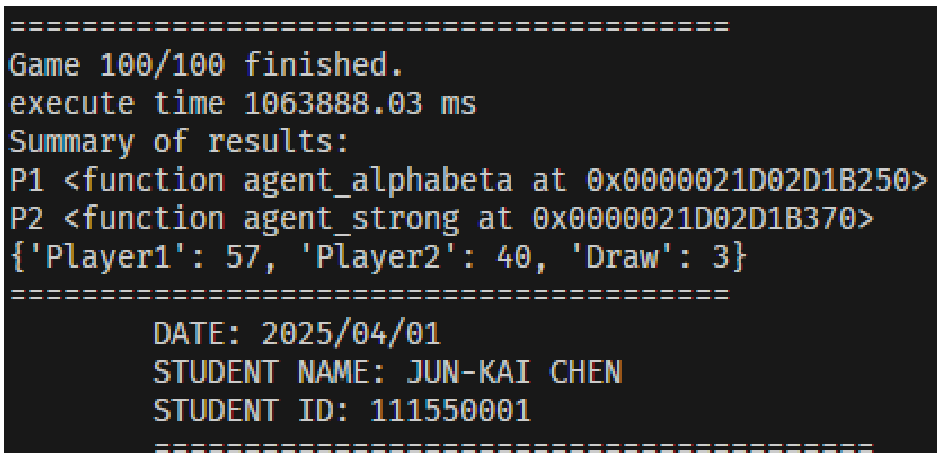
The performance of strong agent with the implementation of stronger evaluation function is shown in Figure 4. In fact, this function could keep the winning rate abover 60%.



```
========================================
Game 100/100 finished.
execute time 1301055.29 ms
Summary of results:
P1 <function agent_alphabeta at 0x000002530EADB250>
P2 <function agent_strong at 0x000002530EADB370>
{'Player1': 26, 'Player2': 73, 'Draw': 1}
========================================
        DATE: 2025/04/01
        STUDENT NAME: JUN-KAI CHEN
        STUDENT ID: 111550001
        ========================================
```

Figure 4: The result of 100 games with α-β agent as player 1 and strong agent as player 2.

If the strong agent calculates the heuristic value using the original function, the result is shown in Figure 5. It can be seen that under the same strategy, player 2 will not reach a winning arate of 50 %, which shows that the modification of the evaluation function is effective.



```
========================================
Game 100/100 finished.
execute time 1063888.03 ms
Summary of results:
P1 <function agent_alphabeta at 0x0000021D02D1B250>
P2 <function agent_strong at 0x0000021D02D1B370>
{'Player1': 57, 'Player2': 40, 'Draw': 3}
========================================
        DATE: 2025/04/01
        STUDENT NAME: JUN-KAI CHEN
        STUDENT ID: 111550001
        ========================================
```

Figure 5: The result of same experiment in Figure 4 but using same strategy.

3 Analysis & Discussion

3.1 Difficulties in Designing a Strong Heuristic

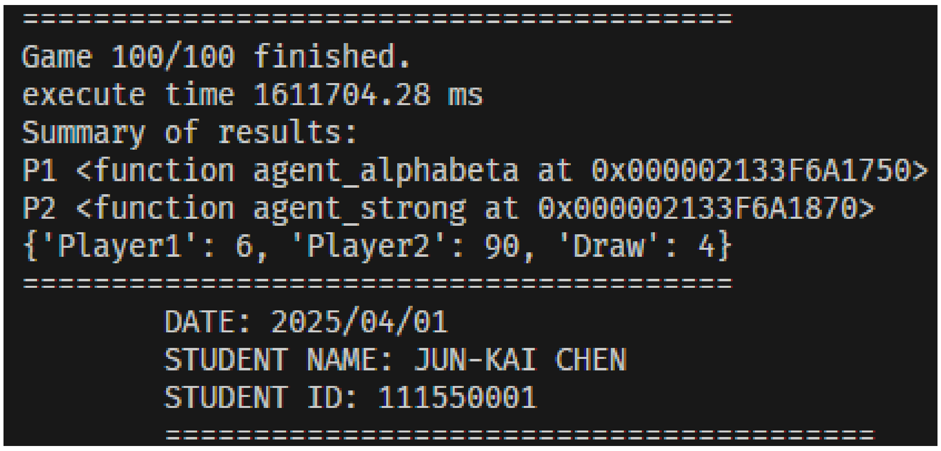
The difficulty in designing a strong heuristic lies in finding an appropriate way to adjsut the parameters. For example, whether to use addition or multiplication, and how to adjsut the values of parameters to ensure that each parameter can affect the result but retains the appropriate weight.

Another difficulty is that the results of the function are not what I expect. The specification mentions that the weights of the center column and recursion depth can be adjusted. However, after actual attempts, my original implementation actually reduced the winning rate of the strong agent. That's the reason why I referred to other materials.

3.2 Weakness of `agent_strong()`

There are some issues with the new function of strong agent. It requires more computing time. In addition, due to alphabeta pruning, the strong agent may also prune the correct answers, causing it to make mistakes in some situations where it should be able to win or avoid losing.

Just for reference, I tried to use a stricter pruning strategy for testing, and the strong agent can achieve a win rate of about 90%. I present the results in Figure 6.



```
========================================
Game 100/100 finished.
execute time 1611704.28 ms
Summary of results:
P1 <function agent_alphabeta at 0x000002133F6A1750>
P2 <function agent_strong at 0x000002133F6A1870>
{'Player1': 6, 'Player2': 90, 'Draw': 4}
========================================
        DATE: 2025/04/01
        STUDENT NAME: JUN-KAI CHEN
        STUDENT ID: 111550001
        ========================================
```

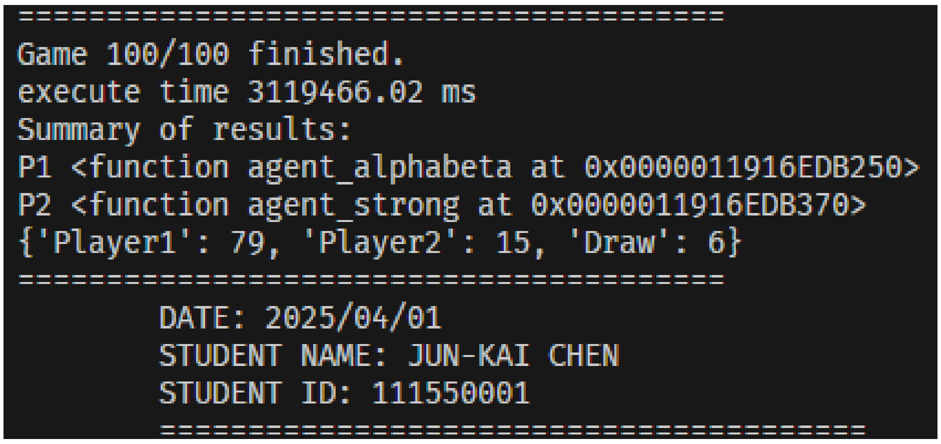
Figure 6: The result of same experiment in Figure 4 but strong agent uses stricter pruning stragey.

3.3 Further Enhanced

To improve strong agent, I can modify the board calculation. For example, in addition to columns, the weight of rows should also be considered, which similar to a two-dimensional weight array, or take the recursion depth into consideration.

4 Conclusion

From the tests in this report, it can be seen that for the minimax algorithm, the main factors that affect correctness are pruning conditions and calculation of heuristic value. The former has a greater impact, because once the algorithm mistakenly deletes the correct answer, the possibility of being defeated increases significantly. An example is shown in Figure 7, which shows the results when both players use strict pruning conditions, where player 1 has a win rate of about 80%. It turns out that the impact of pruning is greater than getting a good heuristic.



```
========================================
Game 100/100 finished.
execute time 3119466.02 ms
Summary of results:
P1 <function agent_alphabeta at 0x0000011916EDB250>
P2 <function agent_strong at 0x0000011916EDB370>
{'Player1': 79, 'Player2': 15, 'Draw': 6}
========================================
        DATE: 2025/04/01
        STUDENT NAME: JUN-KAI CHEN
        STUDENT ID: 111550001
        ========================================
```

Figure 7: The result of same experiment in Figure 4 but both use stricter pruning stragey.

5 Reference

[1] Kang, X. , Wang, Y. , and Hu, Y. , “Research on different heuristics for minimax algorithm insight from connect-4 game.” Journal of Intelligent Learning Systems and Applications, 11, 15-31, `https://doi.org/10.4236/jilsa.2019.112002`.