UNIVERSIDAD CATÓLICA BOLIVIANA "SAN PABLO"
UNIDAD ACADÉMICA REGIONAL COCHABAMBA
Departamento de Ciencias Exactas e Ingenierías

Práctica 3

Sistemas Inteligentes

Estudiantes:
Maria Belen Camargo Aramayo
Edwin Daniel Acero Rojas
Gary Jimmy Quispe Choque

Docente:
Joan Sebastian Gerard Salinas

Cochabamba – Bolivia
20 de octubre del 2023

Report

Installation guide.- First you should enter to GitHub repository link, then open “Proyecto_oficial.ipynb”, when you click on it, it suggests opening the project in Google Colab. To run the project, just execute the cells like any other Jupyter Notebook, especially the last one with the complete functionality to play with it.

1. Solution description

For the solution of the game, we created a Class called “Status” which is the Board. This is because all possible Boards (with the chips in different places) are in different states each. This board called “State” starts with a variable size, which we can choose for making the experiments and the project as it asks.

Some important functions of the Class “Status” are:

- make_move: This function receives a row, a column, and the player. Then changes the black space represented as a “.” to the symbol of the player on the position selected.
- get_valid_moves: This function receives a player and returns an array of all possible moves that the player has.
- is_valid_move: This function analyzes if it is possible to make a move without going out of the board.
- full_board: This function analyzes if the board is full.
- get_winner: This function counts the number of chips each player has and returns who is the winner depending on who has more chips.
- utility: This function returns the difference between black chips and white chips. If the number is positive, it means there are more black chips. If the number is negative, it means there are more white chips. If the number is 0, it’s a draw. This is the way we make Min Max work.

We also used the three different types of min max algorithms:

- Min Max

- Min Max Alpha Beta Pruning
- Heuristic Min Max

For the Heuristic Min Max, we experimented two different heuristic functions to try which one is the best:

1.1. **Counting Chips**:

This heuristic function counts the chips that every player has and evaluates the difference between chips. The most chips a player has, the better opportunities to win.

1.2. **Calculate Mobility and Stability**

This heuristic calculates “Mobility” and “Stability” together.

Stability is evaluated based on the position of the player's pieces on the board. Corners are a stable position, and being on a border is also relatively stable. The function adds 1 point of stability to the player that has a chip in a corner, and adds 0.5 of stability to the player that has a chip in the border.

Then Mobility, it calculates how many legal moves the player has. A higher mobility generally indicates a stronger position, given there are more options to choose from,

We combined these two parameters, “Mobility” and “Stability” for one heuristic algorithm. Individually, none of these heuristics are particularly effective, but combined, they synergize to create a robust response in the algorithm.

2. Experiments

2.1. Min Max

We experimented with various board sizes. We began with an 8x8 board size, but upon running the program, it took an impractical amount of time to respond, prompting us to cut the execution. Then we reduced the board size to 6x6, but the issue persisted. Eventually, we attempted the algorithm with a 4x4 board. Although it still took a considerable amount of time, it managed to complete the execution, expanding the entire tree in 68 seconds.

Figure 1: *Execution time*

```
The function took 68.02253079414368 seconds to execute.
```

This happened because the state space expands, and it is very big for the algorithm so it takes a lot of time to check all of it even if it is a simple 4x4 board.

68 seconds is a very long time to play with it, so we weren't able to test this algorithm in the playing.

2.2. Min Max Alpha Beta Pruning:

We tried to play the game with Min Max Alpha Beta Pruning and these were our experiments:

- First we tried it on a 8x8 board, but the algorithm took a lot of time, and we weren't able to play.
- Then we tried it on a 6x6 board, and the algorithm took a lot of time as well.
- Then we tried to play on a 4x4 board, and the algorithm started working faster than the other boards. We played and in every situation we weren't able to beat the algorithm.

Figure 2: *Execution time and results of the Min Max Alpha Beta Pruning*

```
AI wins, you lost!
Number of X discs: 1
Number of O discs: 10
Average AI response time: 0.02 seconds
```

We played as X and AI was playing as O. The board was a 4x4 board. Algorithm used is Alpha Beta Pruning. We lost 10-1 and the average time of response in each move was 0.02 sec.

2.3. Heuristic Min Max

The last algorithm performed better than the other ones. We tried it directly on a 8x8 board, and we were able to play. We put our best to win but weren't able to. This happened with both of our heuristic algorithms. Another important detail is that we cut the tree on level 5 of the leaves. We tried on a deeper level, at first we also tried cutting at a level 6, but it took more time. We found out that level 5 was a perfect cut.

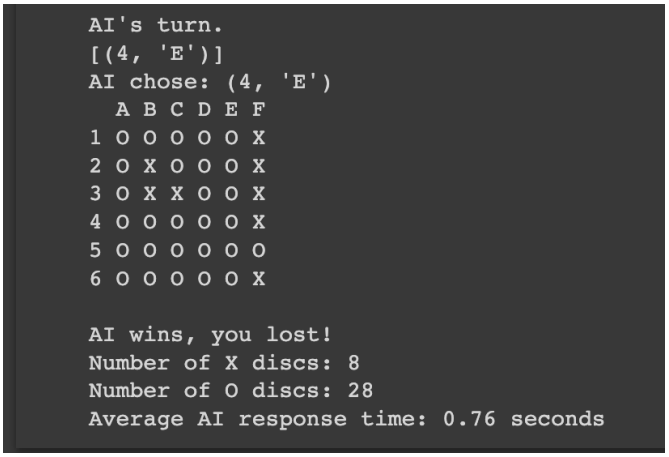
Here is one of similar resulting experiments we did with our first heuristic function “Counting Chips”:

Figure 3: *Execution time and results*

```
AI wins, you lost!
Number of X discs: 1
Number of O discs: 59
Average AI response time: 2.29 seconds
```

We played as X and AI was playing as O. The board was an 8x8 board. The Heuristic Function is Counting Chips. We lost 59-1 and the average time of response in each move was 2.29 sec.

Figure 4: *Execution results with 6 size board*



We played as X and AI was playing as O. The board was a 6x6 board. The Heuristic Function is “Counting Chips”. We lost 28-8 and the average time of response in each move was 0.76 sec. The average time decreased because the board was smaller

Here are some experiments we did with our second heuristic function “Calculate Mobility and Stability”:

Figure 5: *Execution time and results of the second Heuristic*

```
AI wins, you lost!
Number of O discs: 11
Number of X discs: 53
Average AI response time: 6.11 seconds
```

Figure 5: We played as O and AI was playing as X. The board was an 8x8 board. The Heuristic Function is “Calculate Mobility and Stability”. We lost 53-11 and the average time of response in each move was 6.11 sec.

3. Conclusions

Firstly, comparing Min Max and Min Max Alpha Beta Pruning, we found out that Alpha Beta Pruning is better than normal min max as it takes less time to execute. We were able to execute a 4x4 board with both algorithms, but normal min max took 68 seconds to execute, compared to alpha beta pruning that took 0.02. We also tried 6x6 boards but took a lot of time to execute, so in conclusion, the bigger the board, the bigger the state space.

Finally, we will explain the differences between each heuristic algorithm that we used, in all these cases the AI wins, we applied heuristic function Counting Chips the response time was 2.29 seconds for a 8x8 board, if we reduce the size of the board to 6x6 and apply the same function, the response time decreases to 0.76 seconds. We can say this is the best heuristic, because the algorithm takes the lowest time and it wins with a big number of chips.

If we apply the second heuristic function to calculate Mobility and Stability for a board of 8x8, the time response is 6.11 seconds, the time becomes longer than the first one. Even though this is a good heuristic, it doesn't allow the algorithm to reach its highest point and defeats us in a fast way like “Counting Chips” did.

Exploring these algorithms was an enjoyable experience. It allowed us to create a game that not only deepened our understanding of adversarial search but also provided valuable insights into the interplay of games between AI and users.