



Faculty of Engineering & Technology Electrical
& Computer Engineering Department
ENCS3340

Project 1 Report
Magnetic Cave

Prepared By:

Aya Dahbour 1201738 Alaa
Shaheen 1200049

Instructor: Dr. Yazan Abu Farha

Section: 2 & 4

Date: 20/6/2023

† About The Project:

This project aimed to develop an intelligent Magnetic Cave game-playing software, starting from an empty board as the initial state. The successor function allowed for diagonal, horizontal, or vertical movements. The terminal test determined the outcomes of win, lose, or draw. The utility function measured the proximity to winning or the opponent's distance from winning. The game board was represented as a 2D array called “board”, with the 16 initial states positioned on either the far right or the far left due to their magnetic nature. In order to ensure quick decision-making and action, we employed the minimax algorithm with alpha-beta pruning, which allowed the computer to make intelligent decisions by minimizing the evaluation of less important possibilities. We conducted experiments with multiple depths to determine the most suitable one for achieving our goals.

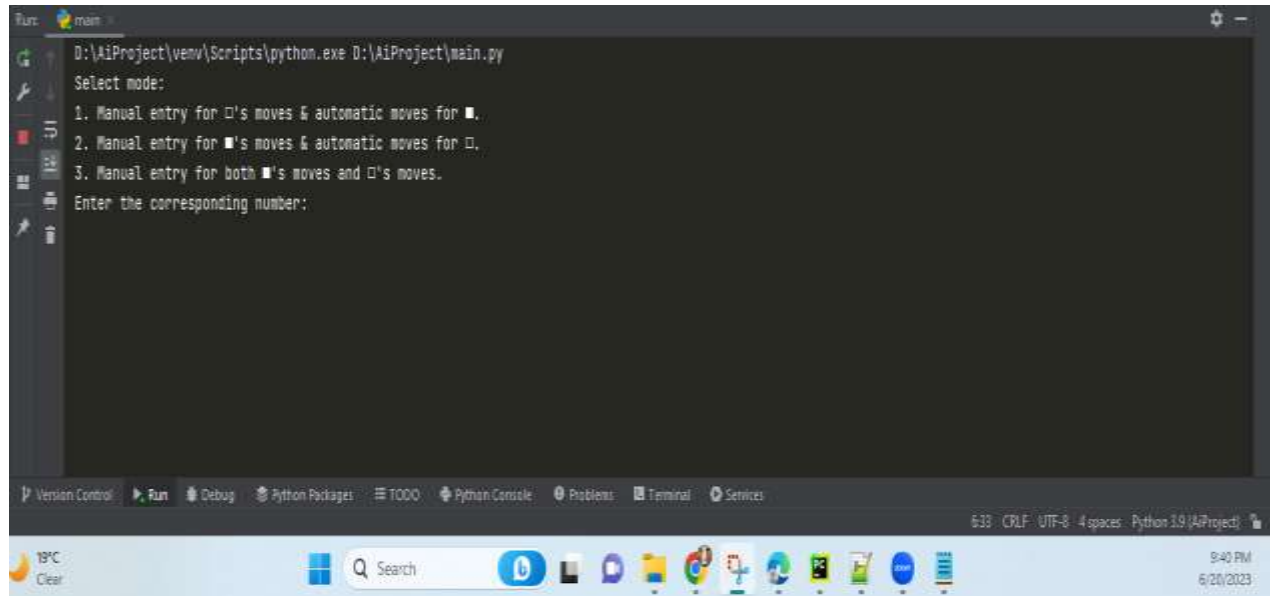
Our objective was to create a software that could effectively compete against human players and other AI opponents, providing an enjoyable gameplay experience. To achieve this, we utilized a heuristic-based approach, taking into consideration factors such as piece movement, board location, and stability during the decision-making process. Additionally, we implemented a graphical user interface (GUI) using Tkinter to enhance the user experience by providing a visual interaction with the game.

Our major goal throughout the project was to create a clever and formidable player that would provide a hard gameplay experience and pose a considerable challenge to opponents. We successfully constructed the primary game logic, created a heuristic assessment function, and held a tournament to assess the program's performance against different opponents. Using the aforementioned principles, we were able to design software that displayed intelligent gaming and effectively met our aims.

The “EvaluateBoard” function in our code implements the heuristic function, which is used to evaluate the current state of the game board. This method examines the game board's current condition and assigns it a score, indicating the advantage of the first player ('■') over the second player ('□'). The heuristic function calculates the difference in winning positions between player '■' and player '□'. It counts the number of winning locations in the board's rows, columns, and diagonals for both players and returns the difference between these counts.

❖ The Game Output:

After running the code, a menu will display to allow you to choose the mode. As shown in the figure below:

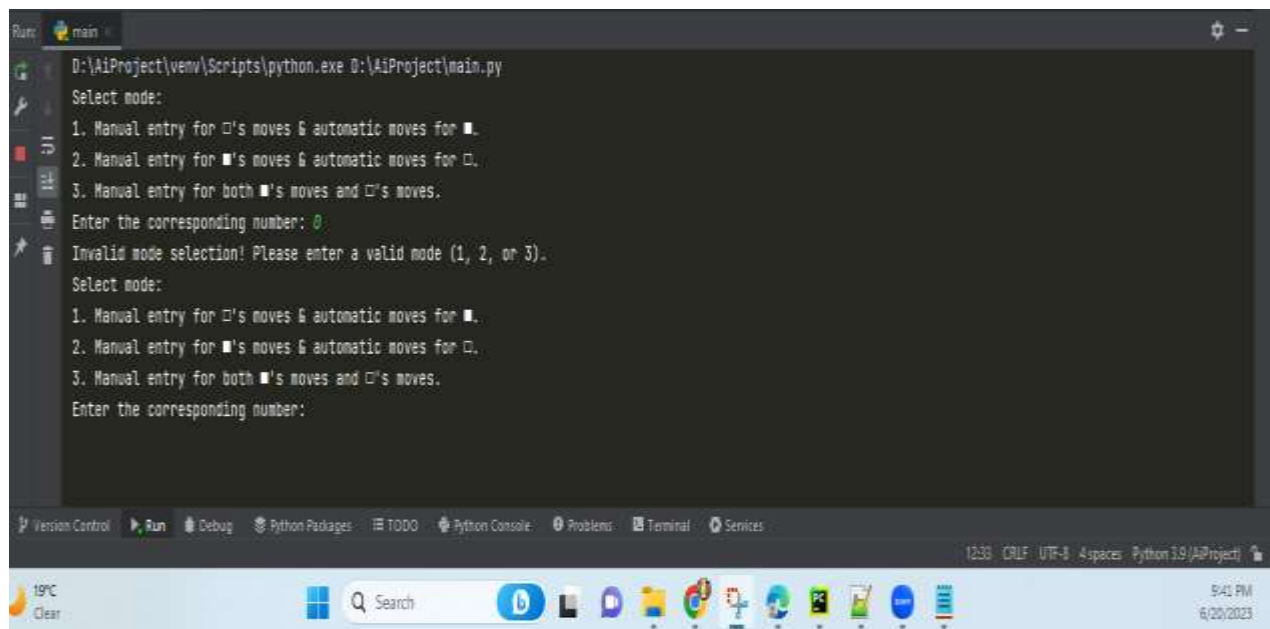


```
Run: main
D:\AiProject\venv\Scripts\python.exe D:\AiProject\main.py
Select mode:
1. Manual entry for □'s moves & automatic moves for ■.
2. Manual entry for ■'s moves & automatic moves for □.
3. Manual entry for both ■'s moves and □'s moves.
Enter the corresponding number:
```

The screenshot shows a Windows 11 desktop with a terminal window titled 'main'. The terminal displays the output of a Python script. The script prompts the user to 'Select mode:' and lists three options: 1. Manual entry for □'s moves & automatic moves for ■., 2. Manual entry for ■'s moves & automatic moves for □., and 3. Manual entry for both ■'s moves and □'s moves. The prompt 'Enter the corresponding number:' is shown at the bottom. The terminal window is part of a larger application with a sidebar and a bottom status bar. The status bar shows '6:33 CRLF UTF-8 4 spaces Python 3.9 (AiProject)'.

Figure 1: modes menu

Then when enter invalid choice the warning appears as follow:



```
Run: main
D:\AiProject\venv\Scripts\python.exe D:\AiProject\main.py
Select mode:
1. Manual entry for □'s moves & automatic moves for ■.
2. Manual entry for ■'s moves & automatic moves for □.
3. Manual entry for both ■'s moves and □'s moves.
Enter the corresponding number: 0
Invalid mode selection! Please enter a valid mode (1, 2, or 3).
Select mode:
1. Manual entry for □'s moves & automatic moves for ■.
2. Manual entry for ■'s moves & automatic moves for □.
3. Manual entry for both ■'s moves and □'s moves.
Enter the corresponding number:
```

The screenshot shows the same terminal window as Figure 1, but now it displays an error message: 'Invalid mode selection! Please enter a valid mode (1, 2, or 3).' This message appears after the user has entered '0' as the corresponding number. The prompt 'Select mode:' is repeated, and the three options are shown again. The status bar at the bottom shows '12:33 CRLF UTF-8 4 spaces Python 3.9 (AiProject)'.

Figure 2: invalid choice

When the user selects one of the options, the application displays the game screen as seen below:



Figure 3: Initial screen

If the user select choice 3 (manual for both players), the game it is natural and follows the decisions and actions of the participants, with no waiting time.

And If the player selects any square other than the sides or sides of a square, a failure message pops informing him that he must select another spot as follows:



Figure 4: wrong move

If the user chooses either 1 or 2 (one player manual & the other automatically) there are two possibilities either for player ■ to win or for player □ to win.

- Example for a case where the computer won and here it was represented by the player ■ (choice 2)
It builds a horizontal bridge.



Figure 5: when computer win

- Another example for the computer won with the player □ trying to build a bridge horizontally and vertically to win, despite that, the computer (player ■) was surprised that the other player beat him diagonally!

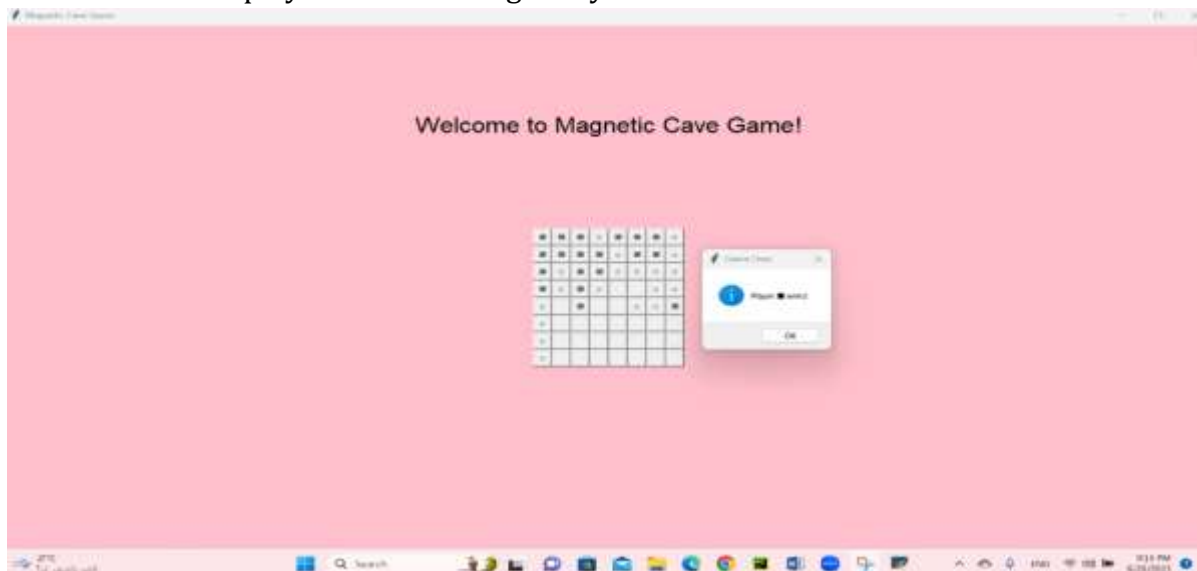


Figure 6: when computer win

- Then there is an example of the player □ winning by building a vertical bridge:

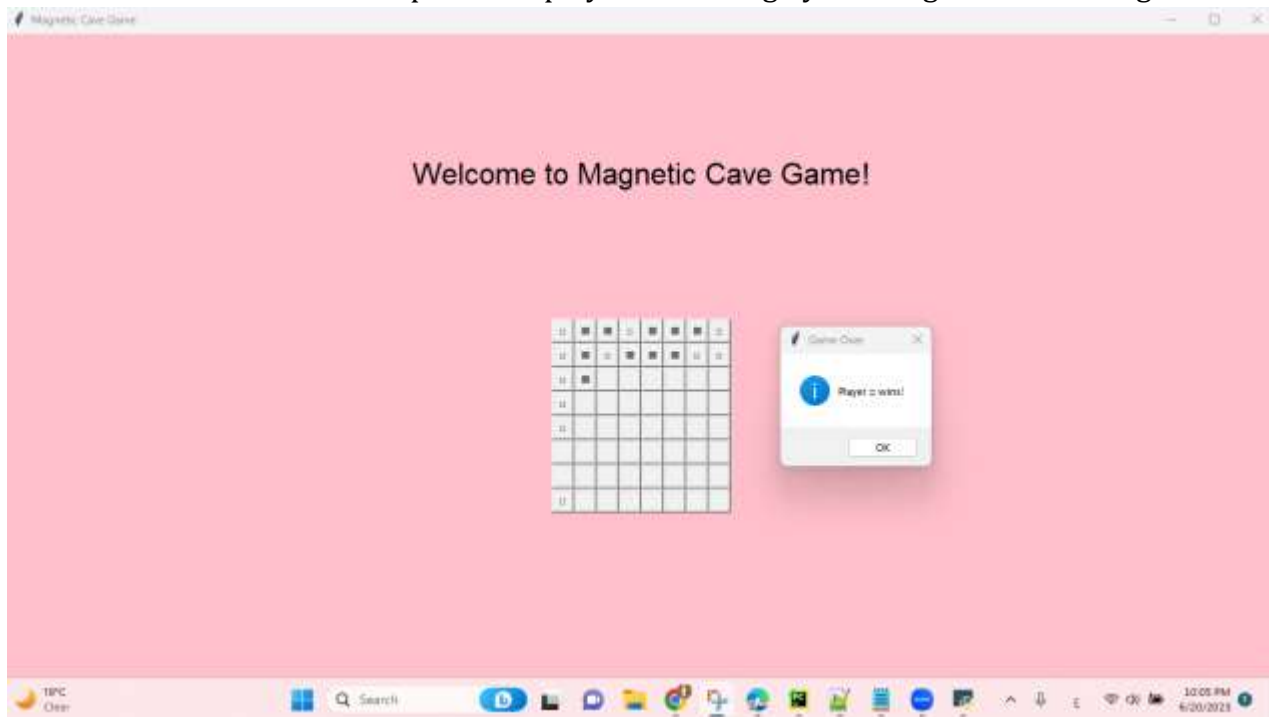


Figure 7: when the player win

“We certify that this submission is our original work and meets the Faculty's Expectations of Originality”