

**Faculty of Engineering & Technology**

**Electrical & Computer Engineering Department**

**ARTIFICIAL INTELLIGENCE -** **ENCS3340**

**Project one**

**Prepared by:**

x**Instructor:** Dr. Yazan Abu Farha

**Section:** 2

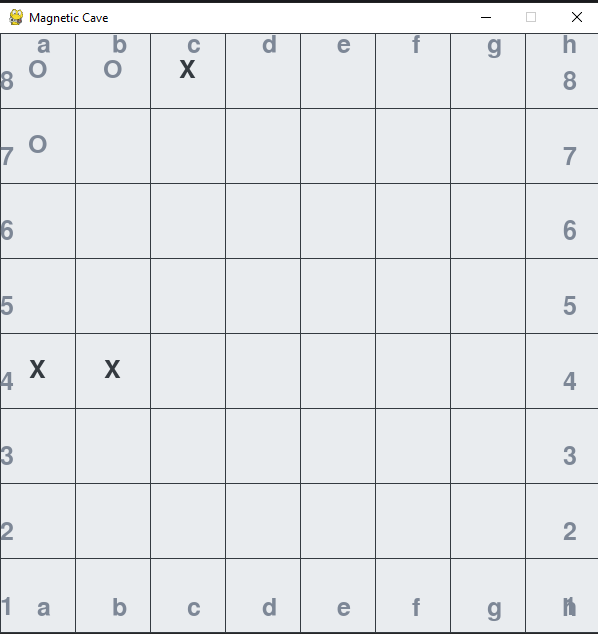
**Date:**20/6/2023

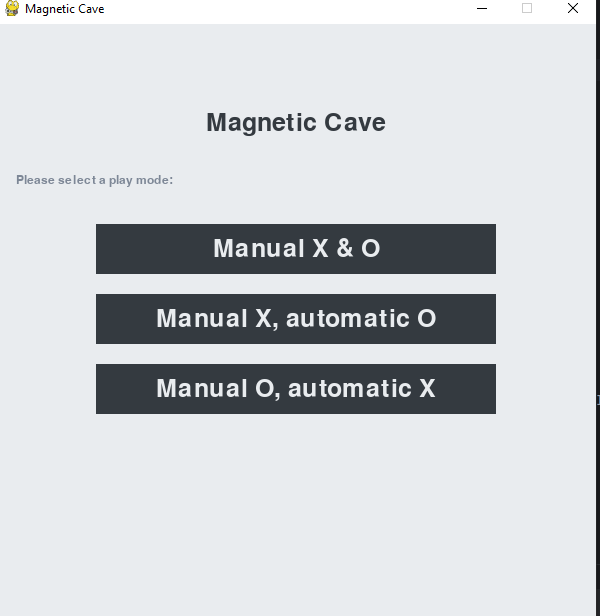
***Program Description***

*The provided code is a Python implementation of a game called "Magnetic Cave". The game is built using the Pygame library and uses a graphical user interface to display the game board and interact with the user.*

***How to Run the Program***

*To run the program, you need to have Python and Pygame installed on your system. Once you have these prerequisites, you can run the program by executing the Python script in your terminal or command prompt.*

**

**

***Main Functions and Data Structures***

*The main data structure in the program is the* ***Game*** *class, which encapsulates the state of the game and the logic for playing the game. The* ***Game*** *class has several key methods:*

* ***\_\_init\_\_****: Initializes a new game. The game board is represented as a 2D list of strings, with each string representing a cell on the board.*
* ***draw\_board****: Draws the current state of the game board on the screen.*
* ***is\_valid\_move****: Checks if a move is valid. A move is valid if the chosen cell is empty and is either on the edge of the board or adjacent to a non-empty cell.*
* ***has\_won****: Checks if the current player has won the game. A player wins the game if they have five or more of their markers in a row, either horizontally, vertically, or diagonally.*
* ***manual\_move*** *and* ***automatic\_move****: These methods handle player moves. The* ***manual\_move*** *method is used when the player is making a move manually by clicking on the screen. The* ***automatic\_move*** *method is used when the computer is making a move automatically using a simple AI algorithm.*
* ***run****: This is the main game loop. It handles user input, updates the game state, and redraws the screen.*

***Heuristic***

*The heuristic used in the* ***automatic\_move*** *method is a simple minimax algorithm with alpha-beta pruning. The algorithm explores the game tree up to a certain depth and uses the* ***evaluate*** *method to estimate the score of each game state. The score is based on the number of consecutive markers of the current player. The algorithm chooses the move that maximizes the minimum score that the opponent can achieve.*

***Tournament Results***

*The performance of the program in a tournament would depend on the strategies of the other players and the depth to which the minimax algorithm explores the game tree. If the other players use a similar or weaker strategy and the depth is sufficient to capture the key strategic decisions in the game, the program could potentially win the tournament.*

*If the program lost against an opponent, it could be because the opponent used a more sophisticated strategy or because the depth of the minimax algorithm was not sufficient to capture the key strategic decisions. The program could also lose if there were any bugs in the implementation or if the heuristic did not accurately reflect the strategic value of each game state.*

*If the program won the tournament, it could be because the strategy and heuristic used by the program were superior to those used by the other players. The program's ability to look ahead and plan its moves could give it an advantage over players who make decisions based only on the current game state.*