Faculty of Engineering & Technology
Electrical & Computer Engineering Department

ARTIFICIAL INTELIGANCE

'ENCS3340'

Project #1

Magnetic Cave Game

Prepared by:

Layan Shoukri 1201225

Amal Butmah 1200623

Instructor: Dr. Yazan Abu Farha

Section: 2

Date: 20-6-2023

Project Description

This code aims to program the magnetic cave game that is played on an 8x8 chessboard, which is played by two players. The game consists of three modes, the first is between two players whose movements are entered by the user, and the second mode is between a player whose movements are entered by the user and an automatic player who bases his movements on the MiniMax algorithm, While the third situation is the exact opposite of the second situation. The winning condition in this game is when any of the players places 5 consecutive blocks without any intersection, either horizontal, vertical or diagonal. When the game board is full of blocks without a player winning, the game ends.

Functions and Data structures

Functions:

The class **MagneticCave** represents the game and contains various methods for playing the game.

1. **MagneticCave()** constructor: Initializes the game by creating an empty board, setting the current player as player1 (■), and filling the board with blank spaces.
2. **printBoard()**: Prints the current state of the board, displaying the positions of players 1 (■), player 2 (□), and blank spaces (_).
3. **validation()**: Checks if a move is valid by verifying that if the move is placed on blank of the cave provided that the brick is stacked directly on the left or right wall or is stacked to the left or the right of another brick.
4. **isWin()**: Checks if a player has won the game by scanning the board for winning configurations in rows, columns, and diagonals.
5. **minimax()**: Implements the minimax algorithm to determine the best move for the AI player. It assigns scores to different moves and recursively evaluates the game state to choose the optimal move.
6. **makeMove()**: Determines the best move for the AI player by invoking the **minimax** function. It selects the move with the highest score and updates the board accordingly.

7. **evaluateMove()**: Evaluates the score of a move based on its impact on the game. It considers factors such as creating winning configurations and blocking the opponent's potential wins, and counting adjacent opponent's pieces around the move.
8. **adjacentbBricks():** Counts the number of adjacent pieces of a given type (player) around a specific move. It checks rows, columns, and diagonals.
9. **isFull()**: Checks if the game board is full, indicating a tie.
10. **playManualManual()**, **playManualAutomatic()**, **playAutomaticManual()**: These methods are responsible for playing the game in different modes. They handle the turns, input validation, making moves, and determining the winner.
11. **main(String[] args)**: The entry point of the program. It creates an instance of the **MagneticCave** class, displays the game menu for selecting a play mode, and calls the respective play function based on the chosen mode.

Data Structures:

- **board**: A 2-dimensional char array stores the state of the game board.
- **boardSize**: An integer variable defining the size of the board.
- **blank**, **player1**, **player2**: Char variables representing the symbols for a blank space, player1 (■), and player 2 (□), respectively.

In order to run the game, we created an instance of the **MagneticCave** class called **game**, and called each of the play modes methods (**playManualManual()**, **playManualAutomatic()**, or **playAutomaticManual()**) according to the selected mode from the user.

Heuristic Description

The heuristic used in our program is a simple evaluation function that assigns scores to different moves based on their potential to lead the automatic player to a winning state or to block the opponent's winning moves. The evaluation function aims to guide the automatic player in making intelligent moves.

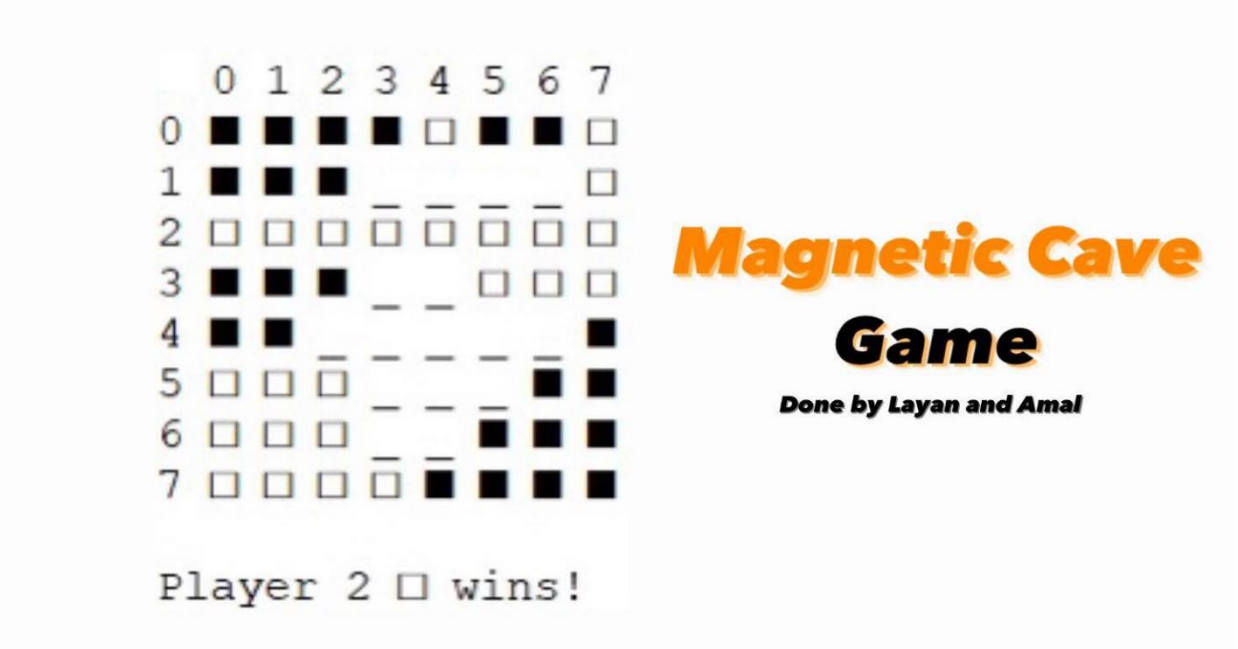
The evaluation function consists of the following components:

Winning Configuration (Score: 100): If the current move results in a winning configuration for the player, a score of 100 is assigned. This encourages the automatic player to prioritize moves that lead to a potential win.

Blocking Opponent (Score: 50): The evaluation function also considers the opponent's potential winning configurations. If the current move blocks one of the opponent's winning configurations, a score of 50 is added. This encourages the automatic player to strategically block the opponent's progress.

Adjacent Opponent's bricks (Score: -1): The evaluation function takes into account the number of adjacent opponent's bricks around the current move. If there are adjacent opponent's bricks, the score is decreased by 1 for each adjacent brick. This encourages the automatic player to avoid moves that create opportunities for the opponent to form winning configurations.

These components of the evaluation function help the automatic player in making decisions by assigning scores to different moves. The automatic player then selects the move with the highest score, maximizing its chances of winning or blocking the opponent. The depth-limited minimax algorithm is used in combination with the evaluation function to determine the best move for the automatic player.



Result Explanation

The figures attached below are for some of the project results.

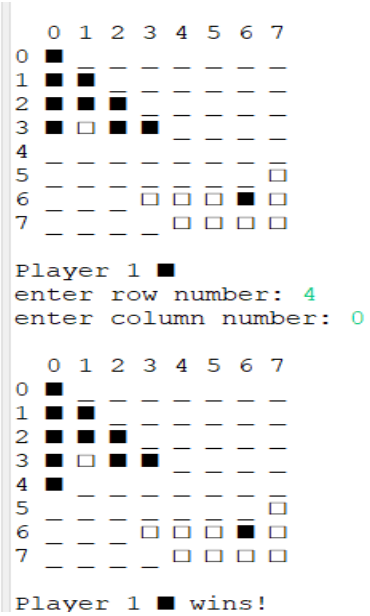


Figure 1: Vertically won

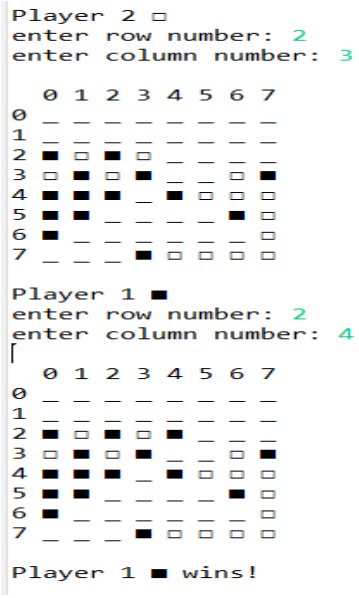


Figure 2: Diagonally won

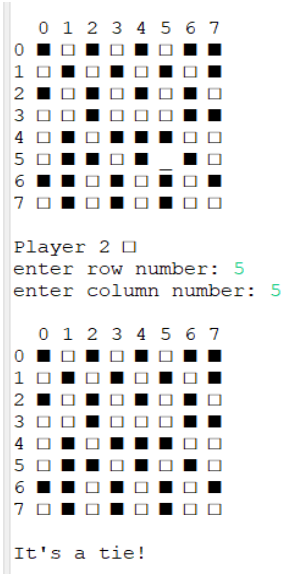


Figure 3: Board is full

The figures above show how the first player won in the manual manual mode, in which the moves of each of the players are entered. The first player wins after making 5 consecutive moves vertically and diagonally. Also, the last figure shows the output "it's a tie" when the board is full.

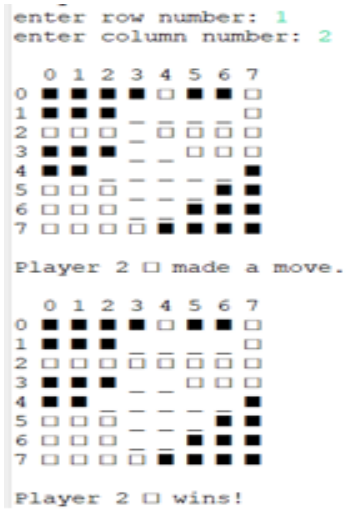


Figure 4: Automatic won

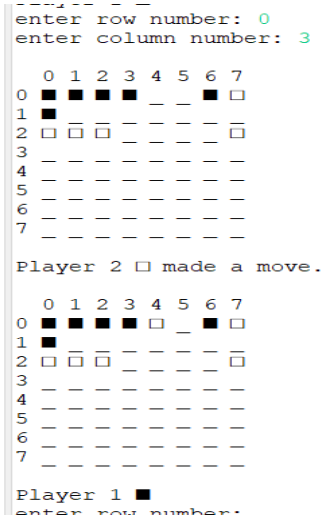


Figure 5: The automatic player prevents the manual player from winning

In the above figures, the first figure shows how the automatic player represented by the brick □ won, while in the second figure it represents how the same player ruins the process of winning the manual player by adding a move to him in the place where the manual player was going to add his fifth brick.

"We certify that this submission is the original work of members of the group and meets the Faculty's Expectations of Originality"

Amal Butmah 1200623

Layan Shoukri 1201225

Date: 6/20/2023