Faculty of Engineering and Technology

Electrical and Computer Engineering Department

ARITIFICAL INTELLIGENCE

ENCS3340

Project

Magnetic Cave Game

Prepared by

Usama Shoora 1200796

Instructor

Dr. Yazan Abu Farha

Section **2**

June 20th 2023

I certify that this submission is my original work and meets the Faculty's Expectations of Originality.
Usama Shoora, 1200796, June 20th 2023

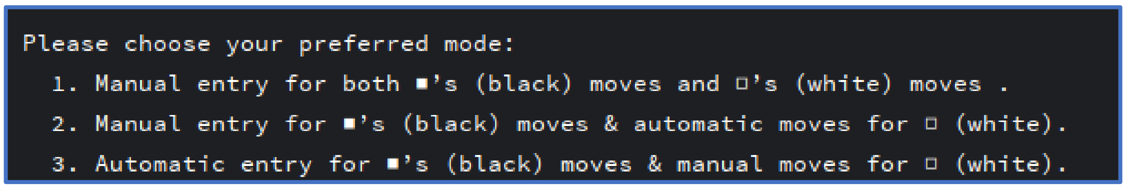
Abstract

This program is a game called Magnetic Cave, it can be played with two players, or one player and the other is automated using an AI. The minimax algorithm was implemented in building this game so that the AI chooses reasonable moves that guarantee its win or at least a tie.

Table of Contents

1. Running the game

To start the game, we run the program and choose from three playing modes; manual entry for both players' moves, manual entry for black player's moves and automatic for white player, or automatic entry for black player's moves and manual for white player.



```
Please choose your preferred mode:
  1. Manual entry for both ■'s (black) moves and □'s (white) moves .
  2. Manual entry for ■'s (black) moves & automatic moves for □ (white).
  3. Automatic entry for ■'s (black) moves & manual moves for □ (white).
```

Choosing the first mode lets the user(s) enter moves for both players in turns. The second mode lets the user enter moves for the black player and the white player's moves are chosen by the AI. The third mode lets the AI decide for the black player and the user enters the moves for the white player.

Trying to insert a brick where another brick already is displays an error "illegal move!". And when a player wins a message saying "Player White/Black won!". And if the board gets filled with bricks without any player winning then a message saying "It's a tie!" is displayed.

2. Main Functions and Classes

1. **Game state class**: a class to represent the game board.
2. **Node class**: a class to represent a game state/move in a tree to be used for the minimax function.
3. **Ai_move**: a function to let the AI decide next move using the *minimax* function with alpha-beta pruning and depth of 3. Depending on the minimax function's returned value, the next legal move with the highest score gets chosen.
4. **Minimax**: a function to implement the minimax with alpha-beta pruning algorithm on the game by generating a tree of states/moves. It works by evaluating every legal move by exploring the game state tree up to a limited depth. It calculates the score of each state using the *evaluation* function.
5. **Evaluation**: a function responsible for implementing the heuristic used to evaluate how favorable a game state/move is. It evaluates the state of each move and returns how favorable it is.
6. **Is_legal_move**: a function to check if a move is legal. Legal moves are those that are either on left/right edges, or stacking next to an already placed brick.
7. **Check_if_win**: a function to check if a current state results in a win for the player by examining the rows, columns, and diagonals for five consecutive bricks of the same player.

3. Data Structures

1. The game board is implemented using a **nested list** which acts like a 2d array.
2. The game state used in the minimax function is represented using a **tree**.

4. Heuristic

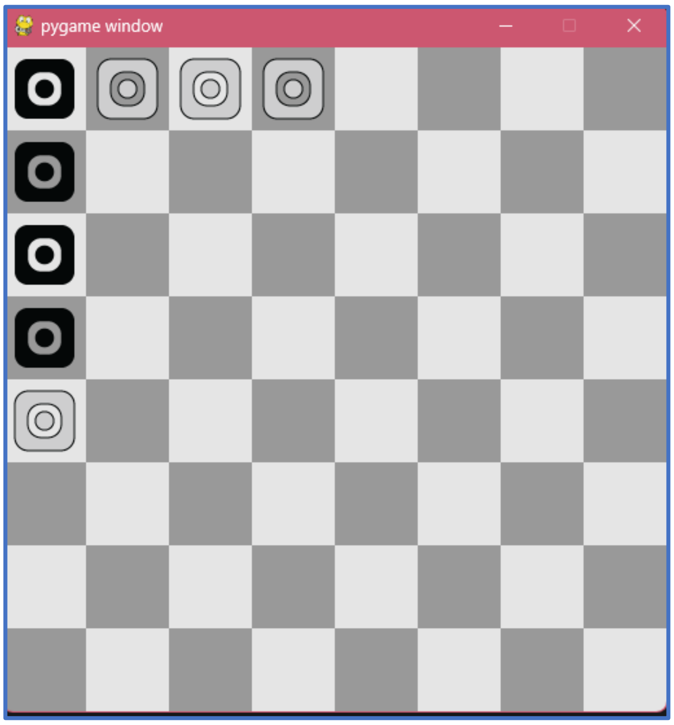
The heuristic used to evaluate a game state works by checking every row, column, and diagonal for 5 consecutive bricks of the same player, and for each 5 consecutive bricks it increments the score to the according player by one. Then it returns the difference of both scores according to the automatic player. This is an admissible heuristic since it ensures the return of a high score for favorable game states.

5. Minimax with alpha-beta pruning

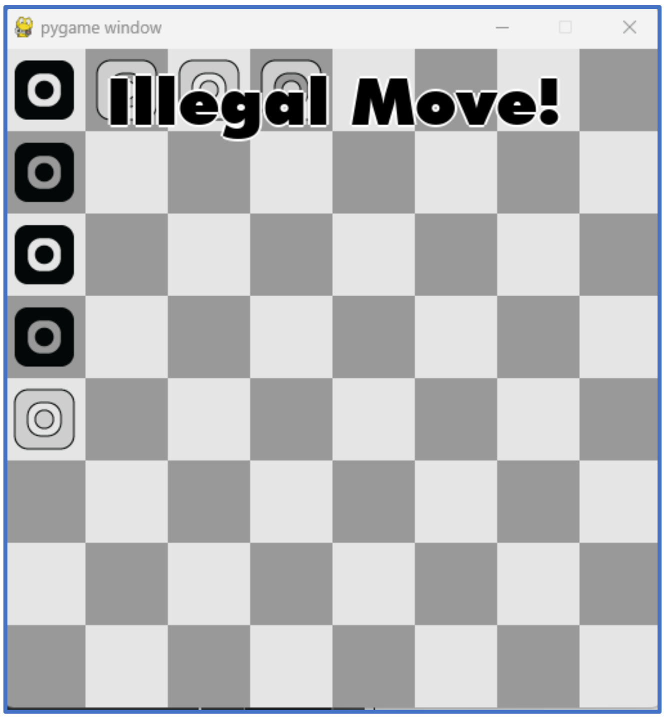
The minimax algorithm used explores the game state tree recursively and prunes out unwanted branches of the tree. With the help of the evaluation function that uses the previously mentioned heuristic, it ensures the AI never losing. The AI move is determined by the minimax algorithm with the depth of 3. An additional line was added in the code to optionally use **iterative deepening** with the minimax function.

6. Examples of gameplay

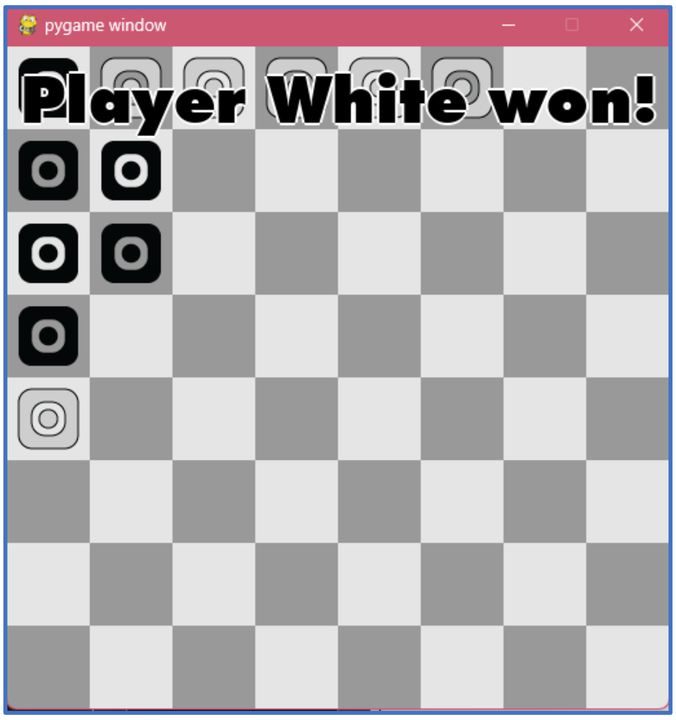
- The AI player (white) blocked me (black) from winning with 5 vertical bricks.



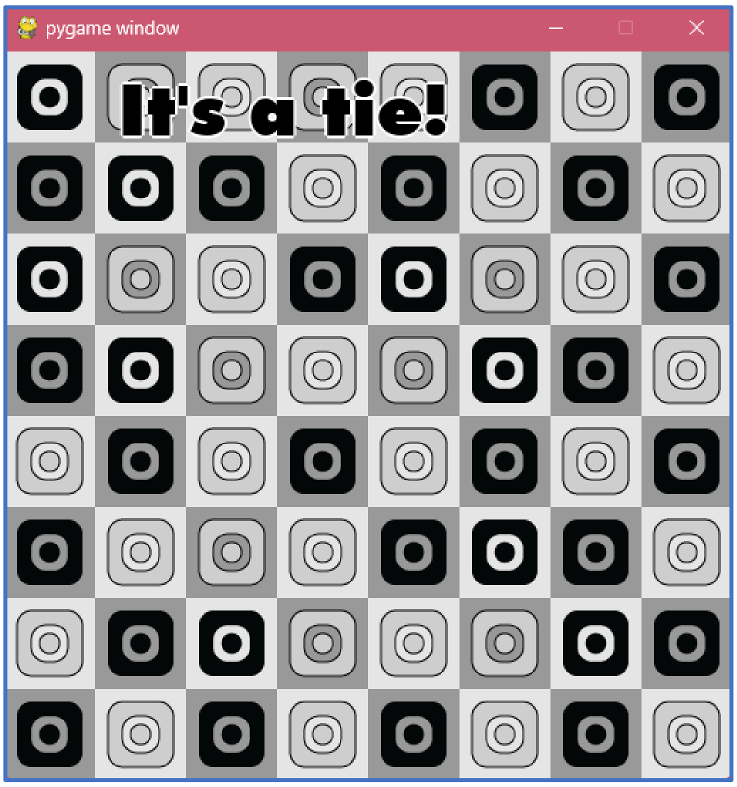
- Choosing an illegal move in the middle of the board results in a warning



- White player (AI) winning.



- The game ending with a tie.


pygame window
It's a tie!