Faculty of Engineering & Technology

Department of Electrical and Computer Engineering

ENCS3340

ARTIFICIAL INTELLIGENCE

Project #1

Magnetic Cave

Group members

Faris Abufarha	ID: 1200546	Section: 4
Rashid Al-Qubbaj	ID: 1202474	Section: 3

Date: 20/6/2023

Table of Content

Table of Content 2
Project Description 3
Setup 3
Main functions 4
Data structures 4
Heuristic 5
Results 6
Charter of Academic Honesty 7

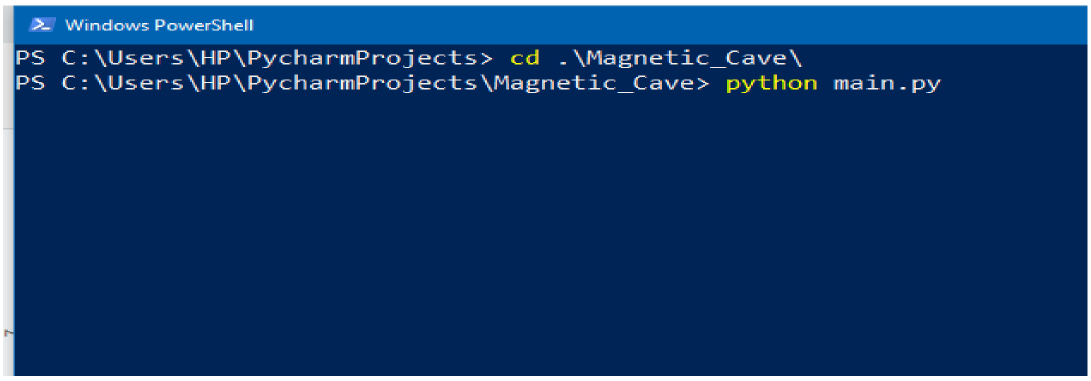
Project Description

The objective of this project is to develop an AI player for the game 'Magnetic Cave' using the Minimax Algorithm as an adversarial search technique. The Player has 2 options either to play against the AI opponent automatically, or play with another human as a 2-Player Game.

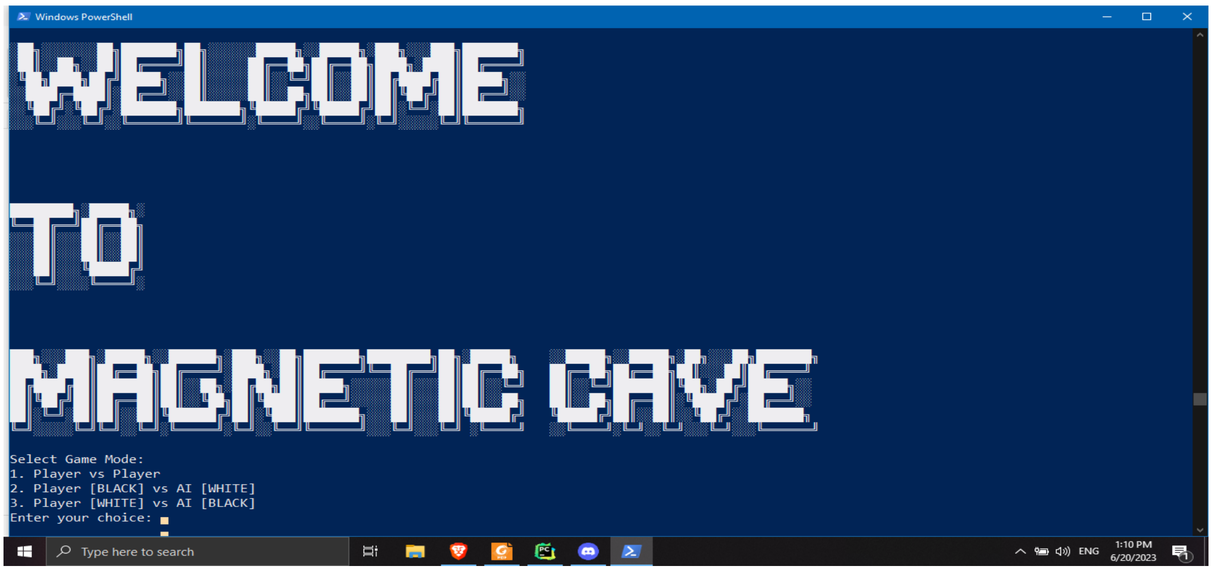
Setup

Running this program is surprisingly easy!

1. Clone the repository from Github or download it as a Zip file to your local machine.
2. Ensure that you have a Python interpreter installed on your system.
3. Additionally, make sure you have NumPy installed. NumPy is a Python library used for efficient numerical computations and is a requirement for this program.
4. Open a terminal or command prompt and navigate to the directory where you have cloned or extracted the program files.
5. Ensure that you are using a dark-themed terminal to ensure proper display of colors and visual elements in the game.
6. Do the following commands:



```
PS C:\Users\HP\PycharmProjects> cd .\Magnetic_Cave\
PS C:\Users\HP\PycharmProjects\Magnetic_Cave> python main.py
```



```
WELCOME

TO

MAGNETIC CAVE

Select Game Mode:
1. Player vs Player
2. Player [BLACK] vs AI [WHITE]
3. Player [WHITE] vs AI [BLACK]
Enter your choice:
```

Main functions

- **evaluate**: This function is responsible for evaluating the current board position. It employs a heuristic approach to assess the state of the game. Further details on the heuristic are provided in the relevant section.
- **possible_moves**: This function generates all the possible moves that can be made at the current position of the game.
- **mini_max**: The mini-max algorithm, enhanced with Alpha-Beta pruning, is implemented in this function. It is a recursive search algorithm that explores the game tree, considering all possible moves and their consequences. The AI player utilises this algorithm, along with Alpha-Beta pruning, to determine the best move to make. Alpha-Beta pruning helps in reducing the number of nodes evaluated, leading to more efficient searching and improved performance.
- **make_move**: Whenever it's the AI player's turn to move, this function is invoked to make the best move based on the evaluation from the mini-max algorithm. It ensures that the AI player makes optimal decisions during gameplay.
- **print_grid**: This function displays the current board position as a 2-D chess board. It helps to visualize the state of the game easily.
- **is_legal**: This function checks the legality of a move chosen by the player. It ensures that the moves made by the human opponent adhere to the game's rules and constraints.
- **is_over**: This function determines if the game is over by either resulting in a draw or a win for either player. It checks the current state of the game and evaluates the conditions for game termination.
- **ascii:** prints fancy Ascii Art

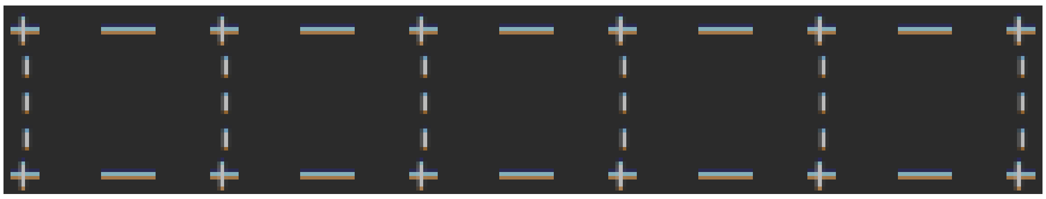
Data structures

- **Dictionary (HashMap)**: A dictionary is used to map between row letters, converting them to integers to store them in the 2D grid matrix. This mapping allows for efficient indexing and retrieval of elements based on their corresponding row letters.
- **List and Numpy.ndarrays**: These data structures are primarily used in the 2D matrix board representation of the game. Numpy.ndarrays are employed due to their faster iteration capabilities. They provide efficient storage and manipulation of the game board, allowing for quick access and modification of the game state.
- **Set**: A set data structure is used to store the legal moves without duplicates. Sets are ideal for this purpose as they automatically handle duplicate removal, ensuring that each legal move is recorded only once. This data structure enables efficient storage and retrieval of legal moves during gameplay.

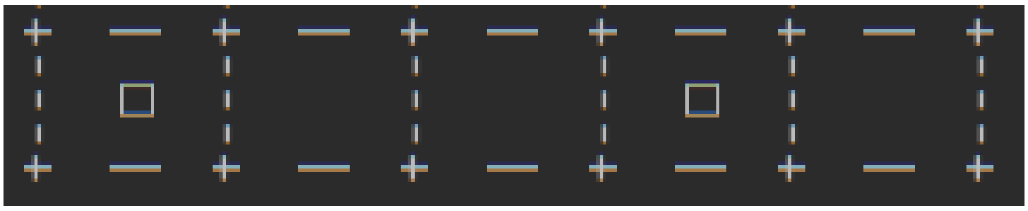
Heuristic

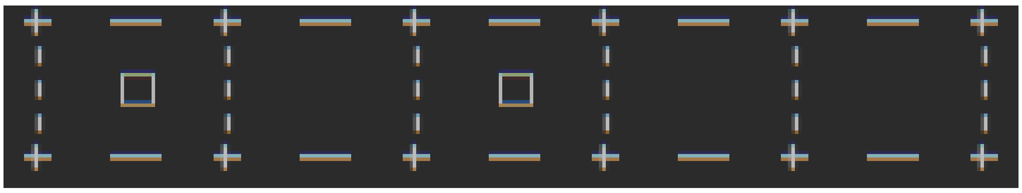
After further analysis we discovered that the game “Magnetic Cave” is very similar to the game “Connect Four”. So we were inspired by the following Github repository made by Keith Galli.

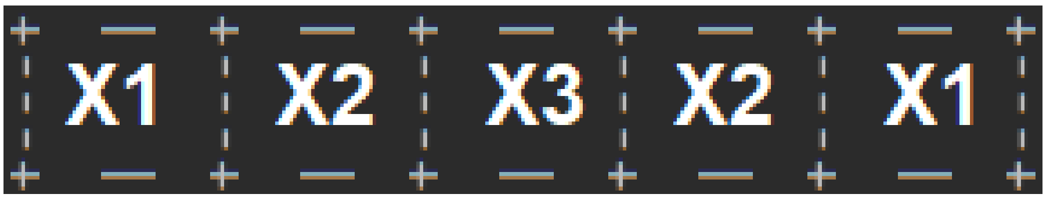
We decided to split the 8X8 board into a set of sliding windows of 5 squares as seen in image 1 on the horizontal, vertical, diagonal axis. We decided on the size of 5 for the windows since winning requires a connection of 5 pieces of the same kind.


(Image 1)

Then after considering possible variations of the window for example image 2 and image 3 We decided on the following values seen in image 4 assuming that the black pieces have a numeric value of 1 and the white pieces have a numeric value of -1 while empty space has the numeric value of 0.


(Image 2)


(Image 3)



(Image 4)

We also had to add an incentive to block connecting 5 for the opponent and to connect 5 for the AI by altering the scoring of said window to be 10000.

Results

We believe that we have a good chance at winning the tournament based on our experiments with the finished AI model.

We will not be disqualified since each move takes on average under 1 second to perform. We can assure that since the time required to make the move by the AI is always printed underneath the grid.

In most of the times the AI Wins, or at least doesn't let us making the winning move, due to the more moves he sees beyond us, and each winning move is scaling the evaluation function to the maximum

Charter of Academic Honesty

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

Faris Abufarha	ID: 1200546	Section: 4
Rashid Al-Qubbaj	ID: 1202474	Section: 3

Date: 20/6/2023