



**Faculty of Engineering & Technology**  
**Electrical & Computer Engineering Department**

ENCS3340

Project Report

AI module for Magnetic Cave game

---

**Prepared by:**

Alaa Saleem                      1200001

Nidal Zabade                    1200153

**Instructor:** Dr. Yazan Abu Farha

**Section:** 2

**Date:** 15/6/2023

## Project idea

To Implement Magnetic cave game using programming language (C, C++, Java, Python), Implement the minimax algorithm, choose appropriate heuristic function and create two player mode either a (human vs human) or (human vs AI)

## Table of Contents

Project idea .....	I
Table of Contents .....	II
Table of Figures .....	II
Code Description .....	1
Dashboard .....	1
Dashboard Design .....	1
Insertion and Update .....	1
Move validity .....	1
Data Structure and related functions .....	2
Tree .....	2
Generate next valid moves .....	2
Generate The Tree .....	2
AI and minimax algorithm .....	3
Heuristic function .....	3
Minimax algorithm .....	3
AI move .....	3
Tournament .....	4
Reasons to win .....	4
Reasons for loss .....	4
Appendix .....	5

## Table of Figures

Figure 1: Dashboard Design .....	1
Figure 2: Tree Code .....	2
Figure 3: Minimax function attributes .....	3
Figure 4: AI best move .....	3

## Code Description

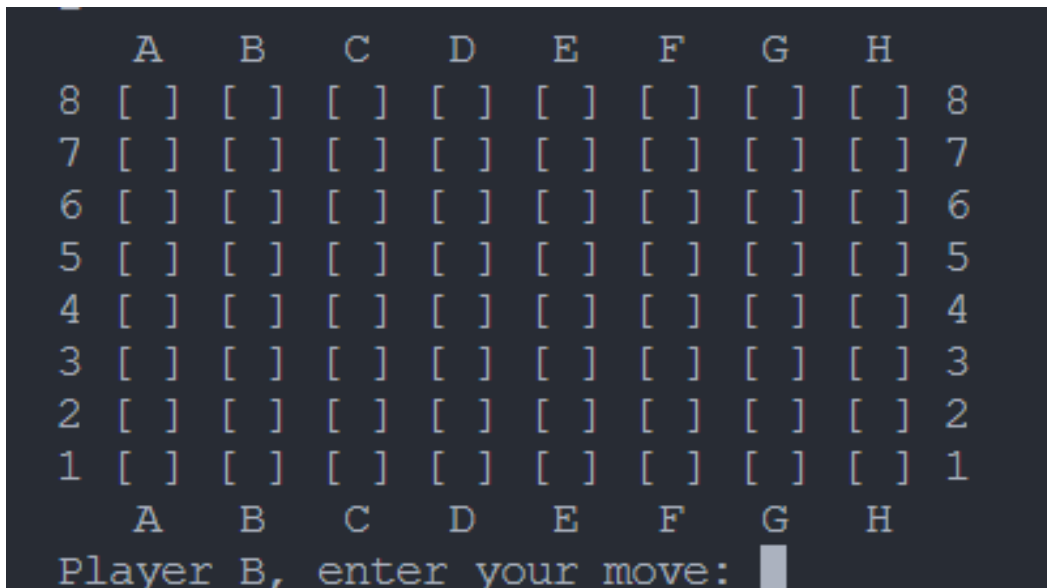
**Note:** All functions described below you can see its implementation in the Appendix section.

### Dashboard

#### Dashboard Design

The dashboard consists of 8 rows (1-8) and 8 columns (A-H) so in total we have 64 tiles that can be filled with small magnets depending on player turn either black (B) or white (W).

The *display\_cave* function is responsible for this operation



	A	B	C	D	E	F	G	H	
8	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	8
7	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	7
6	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	6
5	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	5
4	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	4
3	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	3
2	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	2
1	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	[ ]	1
	A	B	C	D	E	F	G	H	
Player B, enter your move:									

Figure 1: Dashboard Design

#### Insertion and Update

The game started with black turn then white and so on until one of the players win or the game ended with draw. We used the function *enter\_move* to insert new move by the user and *update\_cave* to add the new move to the board

#### Move validity

The move is said to be valid if it follows some rules, can only place a brick on an empty cell 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 (of any color). Check *is\_valid\_move* function.

## Check if the game ends either a draw or one player win

The game ends if one player align 5 consecutive bricks in a row, in a column or in a diagonal, see *check\_win* function. Or the board got filled.

## Data Structure and related functions

### Tree

We implemented Tree Data Structure each node has the current state of the board, root move and the children of the current states (next possible moves).

```
class Node:
    def __init__(self, board):
        self.board = board
        self.last_move = None
        self.children: Node = np.array([])
```

Figure 2: Tree Code

### Generate next valid moves

We created the function *get\_possible\_valid\_moves* to get all the next valid moves to generate the children for each node in the tree

### Generate The Tree

*generate\_tree* is the responsible for creating the tree from the current state

## AI and minimax algorithm

### Heuristic function

Our function calculates the score of a state (board) depends on different attributes, If the state is a winning state for the AI, then give it a score of infinity else Calculate all consecutive of the AI color bricks and set it as the score for the state, if neither (the human is winning) the move which leads to draw get the highest score.

### Minimax algorithm

The minimax function takes the generated tree for the current state, the depth which we want to reach while searching, if the player is maximized or not, the color of the AI player and the color of the human player. the function stated search on the tree recursively and apply the minimization of the human and maximization for the AI player.

```
def minimax(node, depth, is_maximizing_player, maximizing_player,
            minimizing_player):
```

*Figure 3: Minimax function attributes*

### AI move

For AI to find the best move first it generates the tree from the current state, then apply the **get\_best\_move** function that's contain calling the minimax function, finally returns the best move.

```
def AI_move(board, player, depth):
    root = Node(board)
    generate_tree(root, player, depth)
    best_move = find_best_move(root, depth, True, player, "W" if player == "B"
else "B")
    return best_move
```

*Figure 4: AI best move*

# Tournament

## Reasons to win

I think we have a good chance of winning the tournament due the way we implemented the project considering the following:

- 1- Time complexity: we improved the execution time by using NumPy library in python, instead using normal list in python we use the NumPy array, instead of using nested loops we tended to use one loop combined with NumPy functions. NumPy library is a python library which consists of C code wrapped by python code, you can check it by the link: [numpy/numpy: The fundamental package for scientific computing with Python. \(github.com\)](https://numpy.org/doc/stable/index.html)
- 2- Space complexity: we done some calculations to reduce memory usage as much as possible, for example, we know each state has at max 16 next state in the tree so for depth  $n$  we have  $16^n$  each is  $8*8$  board so 64 tile what we can control is the data type in the  $8*8$  board, assume we want to search to depth 3 and we store 0 for empty tile 1 for black and 2 for white if we calculate the space we need  $16^3 * 64 * (4 \text{ bytes [each integer is 4 bytes]}) = 1 \text{ Mb}$  seems not a lot but if we want to go deeper the space will increase in very rapid way so just small improvement instead of store an integer or a Unicode like "■" or "□" which takes also 4 bytes we used just normal character "W" for white and "B" for black each is 1 byte size so that will reduce the total space.
- 3- Decent heuristic function and fast implement for minimax

## Reasons for loss

- 1- Maybe some students implemented their project using faster programing languages like C or C++.
- 2- Better heuristic function
- 3- ChatGPT

## Appendix

[NidalZabade/AI Project1 \(github.com\)](https://github.com/NidalZabade/AI_Project1)