BIRZEIT UNIVERSITY

Faculty of Engineering & Technology
Electrical & Computer Engineering Department

ENCS3340

Project 1 Report

Magnetic Cave

Adversary Game

Search for Optimal Solution by Minimax Algorithm

Prepared by:

Mohammad Salem - 1200651

Pierre Backleh – 1201296

Instructor: Dr. Yazan Abu Farha

Section: 2

Date: 20/6/2023

Program Implementation

We used Java to implement our program using Eclipse. Additionally, we utilized the JavaFX framework to create our FX-GUI views. Firstly, we created two classes: Move and HeuristicResult. The Move class has two attributes, 'row' and 'col', and it is used to define the function called 'findBestMove'. The HeuristicResult class has three attributes: 'detBit' (a Boolean detection bit used to determine the color of the player who has the turn), 'row location', and 'column'.

```
public class Move {
    3 usages
    private int row;
    3 usages
    private int col;

    1 usage
    public Move(int row, int col) {
        this.row = row;
        this.col = col;
    }

    6 usages
    public int getRow() { return row; }

    6 usages
    public int getCol() { return col; }

    18 usages
    public void setRow(int row) { this.row = row; }
```

```
public class HuristicResult {
    5 usages
    private boolean detbit;
    3 usages
    private  int row_blankLocation;
     3 usages
     private  int column_blankLocation;

     public HuristicResult() {}

        2 usages
        public HuristicResult(boolean detbit) { this.detbit = detbit; }

        10 usages
        public HuristicResult(boolean detbit, int row_blankLocation, int column_blankLocation) {...}

    no usages
    public boolean isDetbit() { return detbit; }

    no usages
    public int getRow_blankLocation() { return row_blankLocation; }
```

Problem Formalization:

*Initial state: -

Empty colored (by black and grey) chess board, 8x8 magnetic cave, implemented by 2 D Array initialization by zeros, that appear at interface as empty colored chess bord. We gave flexibility to any player who starts, that appear in the option at interface and in any color, whether with agent (computer) or with another human player.

*Terminal Test: -

We check the Goal State.

If click counter reach 64(fill Board) or if any player (whether agent or human player) achieves a win.

Terminate when not achieved the definition configuration for winner tie case.

*Successor Functions and scissure fringe: -

We implemented a methods called isMoveAvilable , First it checks if the cell is empty or not empty and check if the cell is stacked directly on the left or right wall , and check if the cell is stacked to the left or right of another brick , and other more cases mentioned in the code.

*How to play and Rules: -

1. The game is played on a board called the "cave," which is initially empty.
2. There are two players: Player ■ and Player □.
3. Player ■ always moves first, followed by Player □.
4. Under the following conditions, players can only place a brick on an empty cell in the cave: a- directly on the left or right wall of the cave, b- stacked to the left or right of another brick.
5. The game is won by the first player to line up five consecutive bricks in a row that is either horizontal, vertical, or diagonal.
6. The game ends immediately and the player who achieved the winning configuration becomes the winner.
7. The game ends in a tie if neither player is able to create a winning configuration and the board is full.

*Evaluation Functions and Heuristic evaluation:

*win/*loss/*draw(tie)

When does the player win, lose or draw?

When the player makes a bridge, whether it is of 5 consecutive bricks in a row, column or diagonal.

In artificial intelligence, a heuristic is a strategy or general guideline that supports problem-solving or decision-making. Heuristics are widely used in AI algorithms to make informed judgements or approximations of answers when finding a perfect solution could be difficult or time-consuming.

Heuristics are frequently developed utilizing patterns discovered in the problem domain or domain-specific information. They aim to provide a useful and efficient approach to problem-solving by focusing on essential information and narrowing the search field.

We found a pattern that helps us describe the problem and works to solve it.
Also, we noticed how we can Make the computer play with us strongly and what moves are available that bring the computer agent closer to winning.
This observation was as it appears in the code comments as follows:

```
public  HuristicResult  hueristic_h1(int arr[][],int detBit) {

    HuristicResult hr_inh1_defult = new HuristicResult( detbit: false);

    /**
     *  //case1: check for rows       Bricks,Bricks,Bricks,Bricks,blank
     *  //case2: check for rows      Bricks,Bricks,Bricks, blank,Bricks
     *  //case3: check for rows       Bricks,Bricks,blank,Bricks,Bricks
     *  //case4: check for rows      Bricks,blank,Bricks,Bricks,Bricks
     *  //case5: check for rows       blank,Bricks,Bricks,Bricks,Bricks
     */
```

first heuristic h1

```
public  HuristicResult hueristic_h2(int arr[][],int detBit) {
    // defult value
    HuristicResult hr_defult = new HuristicResult( detbit: false);

    /**
     check for columns
     ------------------------------------------------------------------------------
     case1:           case2:           case3:         case4:          case5:
     ------------------------------------------------------------------------------
     Bricks,           Bricks,         Bricks,         Bricks,        __blank__
     Bricks,           Bricks,         Bricks,        __blank__        Bricks,
     Bricks,           Bricks,         __blank__       Bricks,         Bricks,
     Bricks,           __blank__        Bricks,        Bricks,         Bricks,
     __blank__          Bricks,         Bricks,        Bricks,         Bricks,
     ------------------------------------------------------------------------------
     */
```

Second heuristic h2

In two previous pictures we detection the blank location whether it is in the row (in first screen shot h1 function) or it is in the (second one h2 function), and the methods return objects (instance from Heuristic Result class) it has three attributes,1. Boolean value to determine if the trap has occurred by adverts player or not, 2.row and 3. columns to determine the blank location, to fill that blank by agents, to prevent human player win, whether it is occurred in row or column.

*Search Space: -

Tree search space with min max algorithm.

We use min max algorithm with alpha, beta pruning.

The minimax method with alpha-beta pruning, which is widely used in game theory and artificial intelligence, may be used to find the best move in a game with two players, who are commonly referred to as the maximizer and the minimizer. The algorithm, after exhaustively exploring the game tree and considering all possible movements and

their consequences, eventually selects a move that maximizes the minimizer's odds of winning while reduces the maximizer's chances of losing.

*Our program support: -

1. Player Interaction: The program allows two players to take turns playing the game. Player ■ and Player □ can input their moves and see the updated board after each turn.

2. Valid Move Validation: The program checks the validity of each move to ensure that a player can only place a brick in a legal position.

3. Winning player detection: The program determines whether a player has achieved a winning configuration after every move. When five bricks line up in a row, either vertically, horizontally, or diagonally, the game recognizes it and declares the winning player.

4. Tie detection: In case no player achieves a winning configuration and the board becomes full, the program detects the tie condition and ends the game.

5. Heuristic function: Heuristic functions are built into the program to assist the AI player in making decisions.

6. Minimax Algorithm: The program utilizes the minimax algorithm, a recursive search algorithm, to determine the best move for the AI player.

7. FX GUI: Our program has a JavaFX-built graphical user interface (GUI). Players can interact with the game using the GUI, which offers a user friendly interface.

*Data Structures used: -

1. Array List: We store all legal moves in an Array List data structure. As additional moves are made available on the game board, this list expands dynamically. When validating player moves or adding or removing moves, we can quickly access the list.

2. Two Dimensional array: The game board is represented by a two-dimensional array.

*Results: -

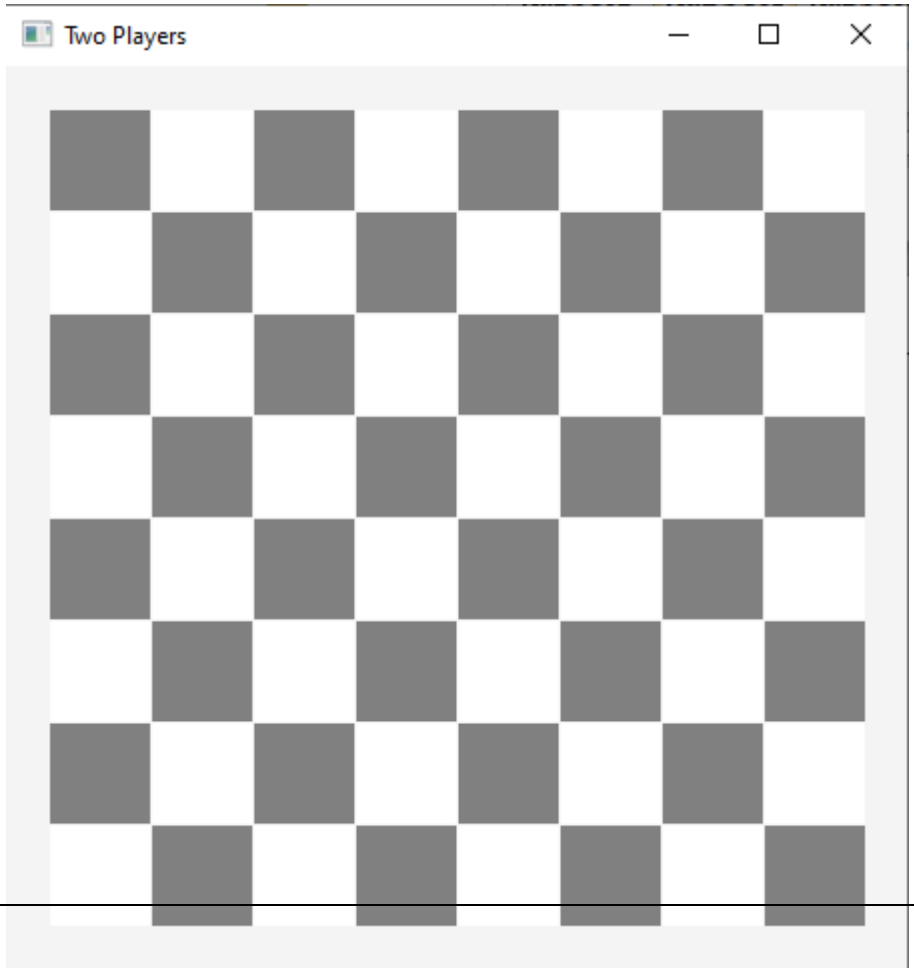
In the main menu, there are three options available for the players:

1. Two Players (Manually): This option allows two human players to participate in the game. Each player takes turns making their moves manually by selecting a valid position on the game board.

2. Player 1 (AI) vs. Player 2: This option enables a game between a human player and an AI player.

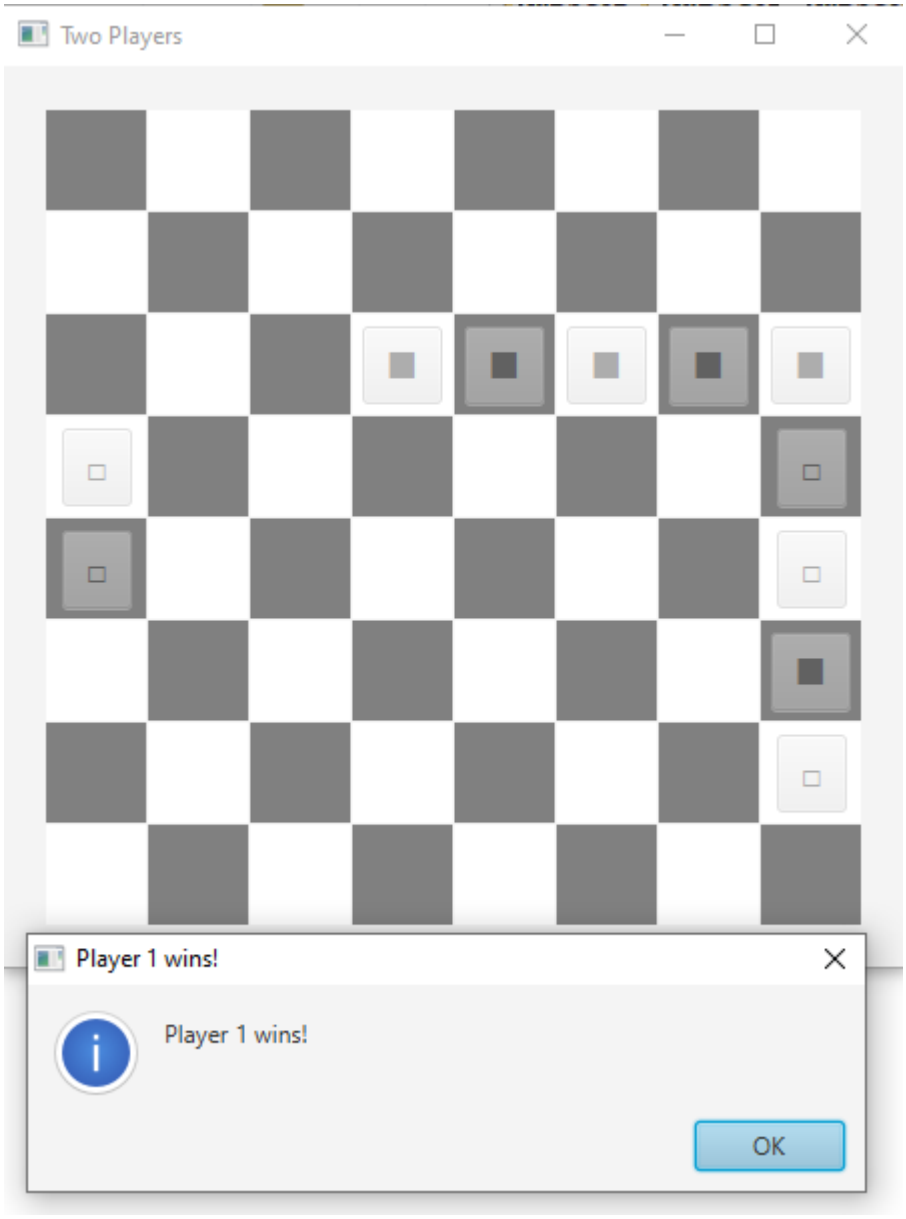
3. Player 1 vs. Player 2 (AI): This option enables a game between an AI player and a human player.

After selecting one of the options from the main menu, the game board will be displayed on the screen. The game board is represented by a two-dimensional grid, typically an 8x8 grid.



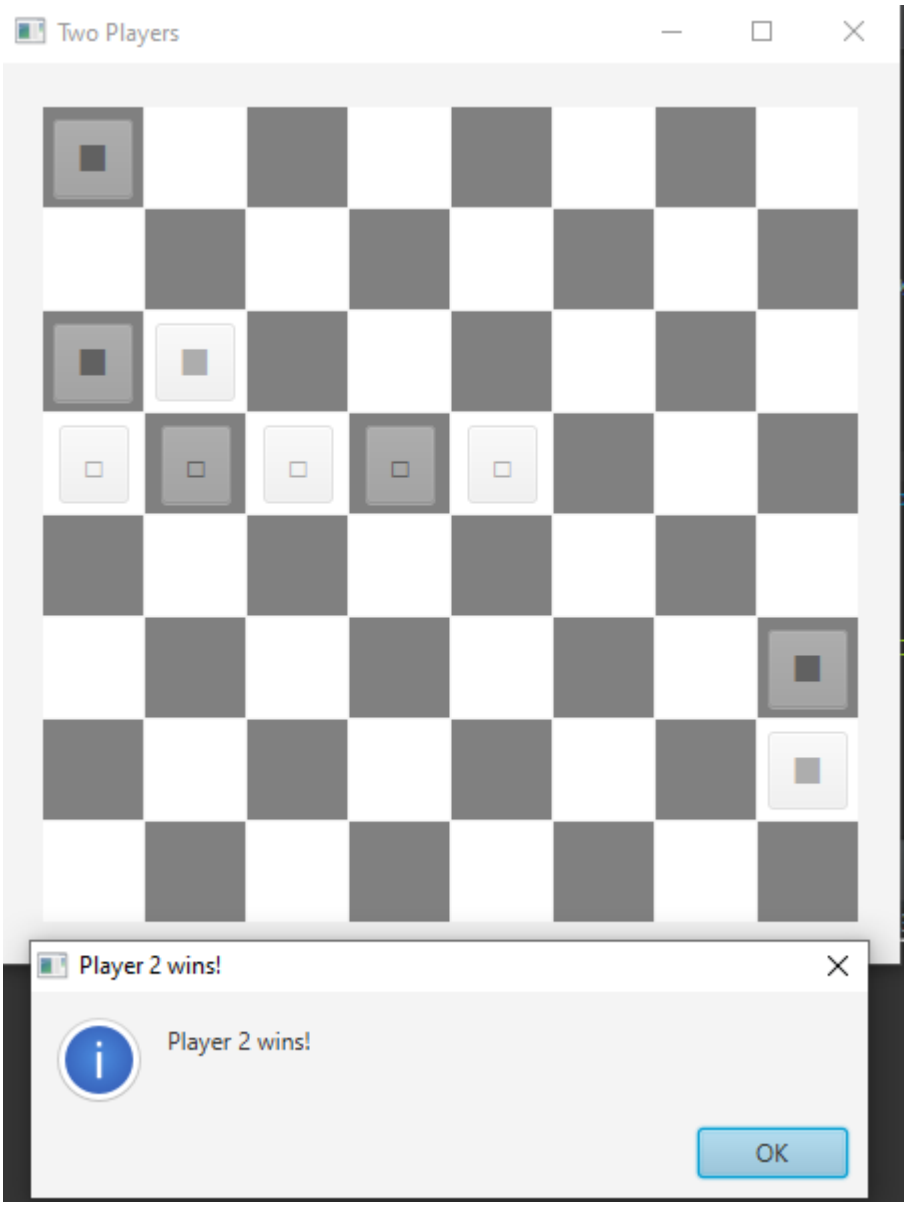
If the player 1 wins the game:

If Player 1 wins the game by achieving a winning configuration of five consecutive bricks (horizontally, vertically, or diagonally), the game will display a victory message as shown below:

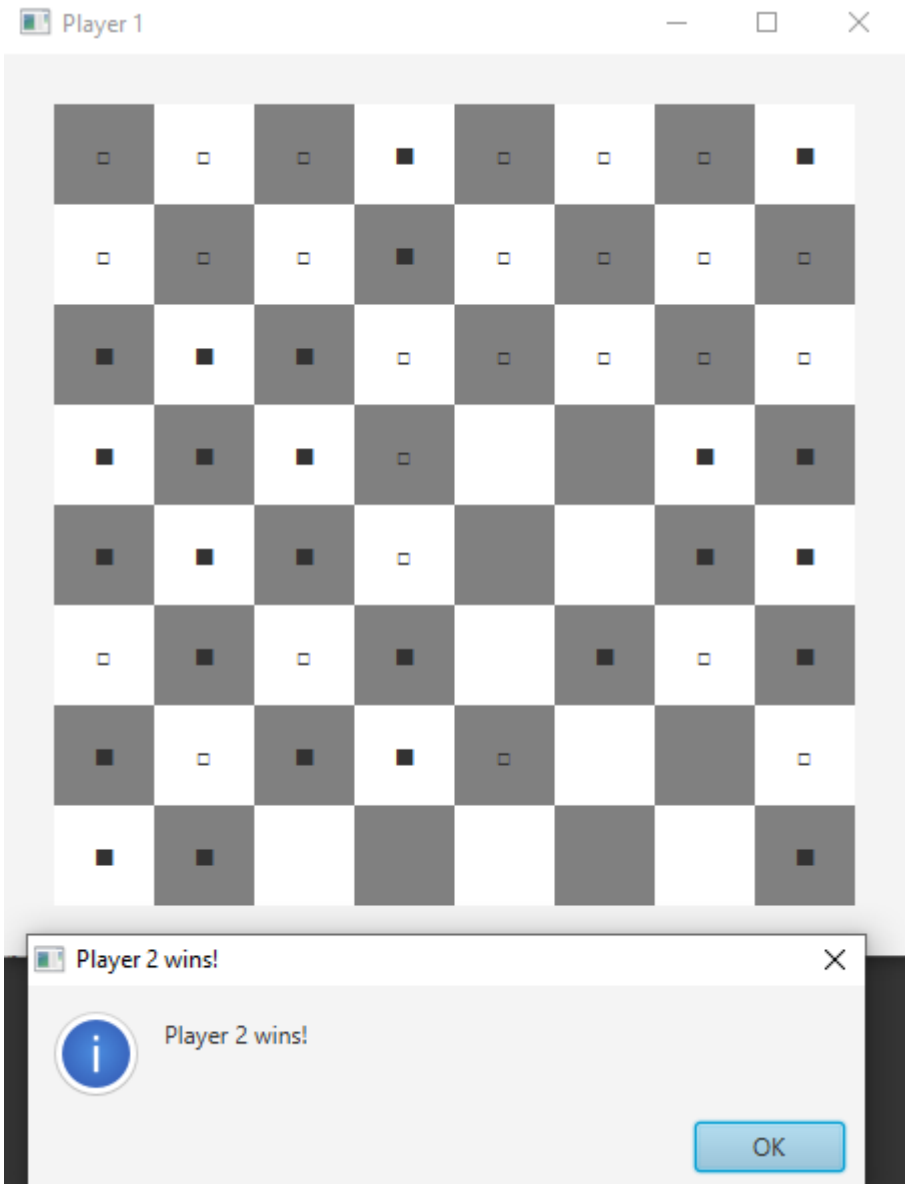


If the player 2 wins the game:

If Player 2 wins the game by achieving a winning configuration of five consecutive bricks (horizontally, vertically, or diagonally), the game will display a victory message.



If the AI wins the game by achieving a winning configuration of five consecutive bricks (horizontally, vertically, or diagonally), the game will display a victory message.


Player 1
Player 2 wins!
Player 2 wins!
OK