****

BERZIET UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

# ARTIFICIAL INTELLIGENCE– ENCS3340

**Project #1** Report

Prepared by

Miral Issa-1200527

Baraa Hamad-1191495

Section#4

Instructor

Dr. Yazan Abu farha

Program Implementation and heuristic

Our project was programmed in C#, and the program interface was done in Unity. our code at first initializes various variables and objects, including the game board, buttons, and game mode settings.

The main data structures used within the code are:

The Board is A 2D button array that serves as the gaming board. The move histories for the X and O players are kept in dictionaries called Xhistory and Ohistory, respectively. The values are arrays that represent the states of each row in that column, and the keys are column numbers, and mode is An enum used to describe the game mode (either AI vs. AI or 2-player).

and the main functions in the code:

When the script object is being loaded, the function Awake() is invoked. Initial values and references are established.

Start(): Before the initial frame update, this method is called. By constructing buttons and setting them to the Board array, it initializes the game board. CreateButton() creates a fresh instance of a button and returns it. The function GetPlayerSide() provides the current player's side (X or O). ChangeSide() flips the player's turn and, if necessary, deals with the AI move. EndTurn(): When a player makes a move, this method is invoked. It manages the game flow, updates the move history, and looks for winning circumstances. updateHistory(): This method makes changes to the player's move history. Heuristic values are determined for a specific move using the move history by the function heuristic(). It counts the number of successful moves in diagonals, rows, and columns.

When the game is over, the function GameOver() is invoked. It displays the game over panel and the necessary message. SetBoardInteractable() turns on or off the buttons on the game board's interactivity. Reset the game to its initial condition using the restartGame() function. And to Set the game mode and launch the game using setMode(). startGame() This function initiates the game by turning on the game board and, if necessary, handling an AI move. The game mode buttons can be enabled or disabled using the setModeButtons() function.

The move logic for the AI player will be implemented by the method AImove(), in this function the AI chose its move according to the minimax algorithm and with the help of alpha-beta that save some time by pruning the minimax tree.

Using the move history, the function checkWinningConditions() determines if a player has prevailed in the match and this function is to be used in the heuristic.

Results at the tournament

We weren’t able to make the AI work under 3s, and that has many reasons for it for starter The best move in a game can be determined using the recursive min-max algorithm. To discover the move that produces the best result, each move is recursively evaluated after all alternative moves have been taken into account. This procedure can take a long time, particularly for games with a lot of alternative moves. And it depends on the depth of the mini max tree we weren’t able to make the depth more than 2, that meant it only search for two moves ahead, we have tried to make the depth 5 but it has taken 11s to make the diction for the move. To make it work faster we have added the alpha-beta algorithm it has optimize the code but it didn’t make it work under 3s. and since the algorithm will not see more than two move ahead the horizon problem will accrue, The algorithm's restriction to just scanning a certain amount of moves forward in the game tree is referred to as the horizon problem. The algorithm may be unable to effectively assess the long-term effects of moves if the search depth is shallow, which could result in less-than-ideal decisions.