Assignment 3 - Tic Tac Toe
Spring 2025
CPSC 428 : Artificial Intelligence

Due Date: April 22, 2025

Objective

Your task is to implement the core logic for a **Tic Tac Toe** game, where **Player X** is controlled by the user, and **Player O** is controlled by an AI using the **Minimax** algorithm. You will implement the 'TicTacToeGame' class to manage the game state, handle turns, and integrate the AI to play optimally.

Deliverables

- A fully implemented 'TicTacToeGame' class in Unity with complete game logic and AI using the Minimax algorithm.
- A working Unity scene with UI components (buttons) for the Tic Tac Toe grid.
- Submit the complete Unity package along with all scripts and assets.
- A demonstration of the game during the class hours.
- A video of your project demonstrating the game, including a working AI that plays optimally.
- Submit the Unity project and video in the D2L dropbox before 11:59 PM on the due date.

Required Functions

You are required to implement the ‘TicTacToeGame‘ class with the following functions:

1. **Start()** *Prototype:* `void Start()` *Description:* Initializes the game. Resets the board and sets up the grid for the Tic Tac Toe game. The initial turn should be assigned to Player X.

2. **OnCellClicked(int x, int y)** *Prototype:* `public void OnCellClicked(int x, int y)` *Description:* This function is called when a cell in the Tic Tac Toe grid is clicked by the user (Player X). - Check if the clicked cell is empty and if it is Player X's turn. - Place Player X's mark ("X") in the clicked cell. - After the player's move, check if there is a winner. If not, pass the turn to Player O (AI).

3. **CheckForWinner()** *Prototype:* `bool CheckForWinner()` *Description:* This function checks if there is a winner. It should check rows, columns, and diagonals to determine if any player has won.

4. **UpdateBoardUI()** *Prototype:* `void UpdateBoardUI()` *Description:* Updates the user interface (UI) to reflect the current state of the board after each move. It should update the text on the buttons to display "X" or "O" based on the current state of the board.

5. **AITurn()** *Prototype:* `void AITurn()` *Description:* This function is called when it's the AI's (Player O) turn. -The AI will decide its move using the **Minimax** algorithm. -Once the AI selects a move, update the board and pass the turn back to Player X.

6. **Minimax(TicTacToeBoard board, bool isMaximizing)** *Prototype:* `int Minimax(TicTacToeBoard board, bool isMaximizing)` *Description:*

   This function implements the **Minimax algorithm** to determine the optimal move for Player O (AI). -It evaluates the board and recursively looks ahead at all possible moves. -If its Player Os turn, it tries to maximize the score (winning). -If its Player Xs turn, it tries to minimize the score (preventing Player O from winning).

7. **ResetGame()** *Prototype:* `void ResetGame()` *Description:* Resets the game, clearing the board and starting a new game from the initial state.

Starter Code

Below is the **starter code**. Your task is to implement the missing logic for the functions based on the provided prototypes.

TicTacToeBoard.cs

```
public enum CellState
{
    Empty,
    X,
    O
}

public class TicTacToeBoard
{
    public CellState[,] board = new CellState[3, 3];

    // Initializes the board by resetting all cells to empty
    public void ResetBoard()
    {
        for (int x = 0; x < 3; x++)
        {
            for (int y = 0; y < 3; y++)
            {
                board[x, y] = CellState.Empty;
            }
        }
    }

    // Checks if the board is full (no empty cells)
    public bool IsBoardFull()
    {
        foreach (CellState cell in board)
        {
            if (cell == CellState.Empty)
                return false;
        }
```

```
        return true;
    }
}
```

TicTacToeGame.cs

```
using UnityEngine;

public class TicTacToeGame : MonoBehaviour
{
    public TicTacToeBoard board;
    public bool isPlayerXTurn = true;  // Player X starts

    // Initializes the game
    public void Start()
    {
        board = new TicTacToeBoard();
        board.ResetBoard();
        UpdateBoardUI();
    }

    // Called when a cell is clicked by the user
    public void OnCellClicked(int x, int y)
    {
        if (board.board[x, y] == CellState.Empty && isPlayerXTurn)
        {
            // Player X places their move
            board.board[x, y] = CellState.X;
            isPlayerXTurn = false;  // Switch turn to AI
            UpdateBoardUI();  // Update the UI

            if (CheckForWinner())
            {
                Debug.Log("Player X wins!");
            }
            else if (board.IsBoardFull())
            {
                Debug.Log("It's a draw!");
```

```
        }
        else
        {
            // Now it's AI's turn
            AITurn();
        }
    }
}

// Checks if there is a winner
public bool CheckForWinner()
{
    // Implement logic for checking rows, columns, and diagonals
}

// Updates the UI to reflect the board state
public void UpdateBoardUI()
{
    // Implement logic for updating the UI with the current board state
}

// Makes the AI (Player O) take its turn
public void AITurn()
{
    // Implement logic for AI's move using the Minimax algorithm
}

// Implements the Minimax algorithm to evaluate possible moves
public int Minimax(TicTacToeBoard board, bool isMaximizing)
{
    // Implement the Minimax algorithm
}

// Resets the game for a new match
public void ResetGame()
{
    board.ResetBoard();
    isPlayerXTurn = true;  // Player X starts
```

```
        UpdateBoardUI();
    }
}
```

Instructions

- Implement the 'TicTacToeGame' class as described in the specification. You must write the logic for handling player input (Player X), AI decision-making (Player O using Minimax), and updating the board after every move.
- Implement the **Minimax** algorithm to allow the AI to make optimal decisions. Your AI should be able to win or tie every time it plays.
- Your Unity scene should have a grid of buttons (3x3) representing the Tic Tac Toe board. Each button should display either "X" or "O" based on the current state of the game.
- After implementing the game, test it by playing it. Make sure the AI plays optimally using the **Minimax** algorithm.
- Record a video of your project showing how the game is played, the AI's behavior, and the decision-making process. Make sure the video demonstrates both player moves and AI moves.
- Package your Unity project and include all relevant files (scripts, assets, etc.) in the Unity package.
- Submit the Unity project and the video in the D2L dropbox by the deadline.
- Be prepared to demonstrate your project during class hours on April 22, 2025.

Bonus Challenge (Optional):

Implement **Alpha-Beta Pruning** to optimize the **Minimax** algorithm. This will improve the performance of the AI by pruning branches in the game tree that are not needed, making the AI run faster.

Grading Criteria:

The assignment will be graded based on:

- Correctness of the game logic (40%).
- AI implementation using the **Minimax** algorithm (30%).
- Quality of the Unity project (UI, functionality, etc.) (20%).
- Video demonstration and ability to explain the project (10%).