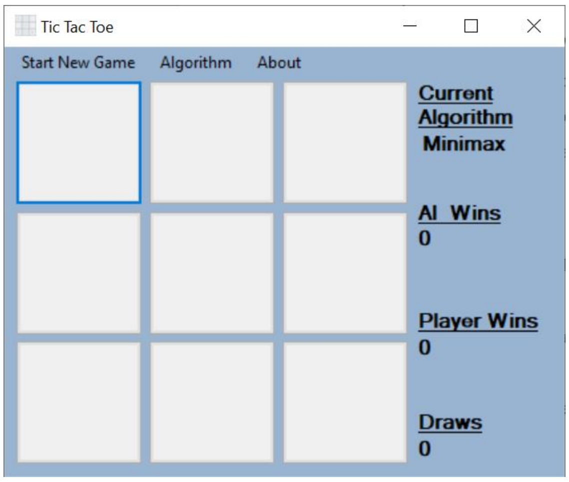
Brennan Coslett
11.05.2020
C# Project Documentation

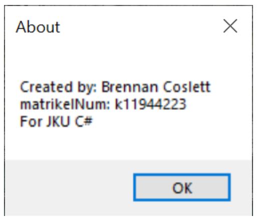
Directory Breakdown

This project was created using a WinForms C# project inside of Visual Studio 2019. Therefore, there is some code generated by the form designer in VS as well as the program logic. The source code for the logic is inside of **Form1.cs** and the WinForms design elements were created inside of **Form1.Designer.cs**. The generated .exe file is in **./bin/Release/TicTacToe.exe**.

WinForms Design



The basic form design is simple. There is the 3x3 grid of squares that make up the game play area. Each square is a button. On the right side of the play area are a series of automatically updated labels. The top label shows the current "AI" algorithm with the choice between Minimax and a Random square choice. This label is automatically updated when the user changes the algorithm against which they play. The other three labels all show the current win count for: the player, the AI, and how many game states have ended in a draw. The menu bar follows gives the user 3 options. "Start New Game" begins a new game, "Algorithm" allows the user to choose between the algorithms and "About" displays a small popup showing info about the project.


About

Created by: Brennan Coslett
matrikelNum: k11944223
For JKU C#

OK

Logic Design

The player always plays first as it gives the player a slightly higher chance to beat the minimax algorithm. The board does not do anything until it receives **Player_Turn** event sent by clicking on one of the buttons in the play area. The sender is captured and converted to a button. The text is updated to an "X" to show the player has placed their move there and the button is disabled so that it cannot be clicked by the player again. **BoardState** at that position is then updated to "-1". **BoardState** contains values of "-1,0,1" corresponding to Player, blank, and AI plays. **Turn_Count** is then incremented and if **!GameFinished(BoardState)** is checked. If the no win/draw conditions have been met, then the game continues and moves onto the AI's Turn.

GameFinished()/ CheckWinState()

The first thing *GameFinished* runs is *CheckWinState* which iterates through the **WinConditions** array which contains every combination of board indices that would win the game and checks if they are all the same value (-1,0,1) and if any of the corresponding buttons are disabled. This ensures that it does not also consider an empty row as a winning condition. *CheckWinState* returns a tuple containing the value of the winner (-1,1) and a bool for whether any of the buttons in the winCondition were disabled. If **Winner.Value** and **Winner.ButtonState** are both true *GameFinished* iterates through all Controls and if they can be cast to buttons will disable them. Then it displays a popup with the winner or stating there has been a draw and calls *ResetGame and returns true*. If no WinCon has been met it *returns false*.

ResetGame()

ResetGame will iterate through all Controls and if they are buttons reenable them and set their Text to blank. It then sets **Turn_Count** to 0 and clears the **BoardState** array. If **AlgorithmChange** is true it will also reset the winCount labels to 0 and will show a MessageBox stating all values have been reset.

AI_Turn()

AI_Turn creates a new array to hold a copy of the **currentBoardState** and creates a **ListOfButtons** that allows easy indexing of all of the buttons on the form. **UseMinimax** checks if **Algorithm** is "Minimax" and will run the correct algorithm based on that. **MoveIndex** is initialized to -1 to ensure that no current plays are overwritten. This game uses a modified version of the minimax algorithm that I believe makes the game slightly more interesting to play. The first turn (**AIFirstTurn = true**) the **MoveIndex** is generated using the *RandomNextMoveIndex* function. This ensures that the first move by the AI isn't always the center position. This means that the minimax algorithm isn't guaranteed a win I believe this makes the game more interesting to play against. Then it runs *TwoOfThree* on the **CurrentBoardState**. If any of the win conditions has 2 "X"s the AI should choose to block the player from winning. Next is the actual Minimax implementation.

MiniMax Algorithm

The original run of Minimax (inside *AI_Turn()* not the *Minimax()* function) is attempting to maximize the current turn which means the **OptimalScore** needs to be set to the MinValue. For each valid and unoccupied board state in **CurrentBoardState** we will attempt to change that position to be our move and then we will call **Minimax** with that board state, a depth of zero and **isMaxing** set to false. This will check if our move is a good fit with minimizing the effects of the players next move. If the **Score** generated by minimax is greater than **Float.MinValue** it changes **OptimalScore** to **Score** and adjusts the **MoveIndex** to the corresponding position. It does this for all available moves until the best move is decided.

Minimax()

This is a recursive algorithm with two modes Minimizing and Maximizing (therefore the name). It attempts to maximize the AI's turns and minimize the effects of the player's turn. For each position it will attempt a move and then call itself in the opposing version until it reaches a terminal node. This value is then carried up through the nodes and the comparison logic is used to determine the best move. To stop this from becoming an infinite loop the beginning of this function runs *CheckWinState* on whatever **BoardState** that has been passed to it and if it is a winning position it will either return the **result** if the **depth** is 0 or if the **depth** is greater than 0 it will return (**result / depth**). This will prioritize the winning conditions that will affect the game sooner over those which will have more of an effect later in the game.

RandomNextMoveIndex()

This function iterates through the **BoardState** and if any index is a blank space adds it to the list **ValidMoves**. If **ValidMoves.Count != 0** it will return a random index from the list.

UpdateAISpace()

This function takes the **MoveIndex** generated and will update **BoardState** at that index as well as the button Text and disable it as well as incrementing **Turn_Count**.