GADE 7321 PART 2

By Jano Frederick Kemp (St10087979) and Adam Miles Luttig (ST10069132)

Table of Contents

HIGH CONCEPT STATEMENT

Tic Tac Toe, the blueprint for turn-based games, but with a twist. In this battle, strategy and chance meet each player must use their wits to gain the upper hand in this exciting twist on Tic Tac Toe. The stage is set on a 4x4 grid as opposed to a the traditional 3x3 grid and the starting player will be decided by chance. The players need to strategies and think carefully where they want to make their move and to aid them, they will be granted a special power up to help tip the scales in their favour. Whether it be a game-altering power boost or an ingenious counter to their opponents move. Each move will shape the path to victory or dismay in this exciting game of strategy and surprises.

Game Specifications/Rules

- 4x4 grid
- Players decided amongst themselves their game symbol. They have choice between X or O.
- The starting symbol will randomly be chosen and then the respective player must make their move.
- A player can choose to place their symbol anywhere on the 4x4 grid by clicking on that place they.
- The game is turn-based so once the starting player has made their move, the other player must choose where they want to place their symbol and so the process will repeat until someone wins or the game results in a draw.
- After each player has made their first move, they will be given 1 random power up that they can use on their turn at any time during the game.
- There are 2 power ups:
  - A replace power up which when played, the player who played it can choose to replace anyone of the opponents' symbols.

    - A Double turn power up which when played, the player who played it can places the their symbol twice in a row, effectively skipping the opponents turn.
- Game outcomes:
    - Win: To win, a player must align 4 of their symbols either vertically, horizontally, or diagonally.
    - Draw: All cells of the 4x4 grid have been filled and neither player was able to align 4 of their symbols.

Game State Representations

- The size of the gameboard will be a 4x4 grid.
- Each cell on the gameboard will be a button.
- Initialising game function:
    - This function is responsible for the setting up the initial state of the game.
    - It will consist of two for loops that will loop through each row and collum of that gameboard that consists of a 4x4 grid of buttons. It will find each buttons' game object and then store a reference to the button in its respective cell in the 2D array. For example it will store button 1 in buttonArray[0,0].
    - This function will also randomly select either symbol X or symbol O and assign it to a currentPlayer variable to indicate which player must start. After this it will just switch back and forth between X and O until the game is done.
- Button click function:
    - This function will deal with the actual gameplay of the game and it will be called every time a button is clicked.
    - When a player clicks on a button the text of the button will change to the symbol of the current player (either X

    or O) and the buttons’ intractability will be set to false which means it cannot be clicked gain.
    - There will be two if statements to check if it is a draw or if there is a winner.
    - In the if statement that checks if there is a winner it will call a checkWinner function and if the value is true, it will display the winner via text on the screen.
    - In the if statement that checks if it is draw it will call a drawFunction and if the returned value is true, it will display that it is a draw via text on the screen.
    - If there is no winner or draw, it will check if the current player is X or O and it will assign the opposite value to indicate who’s turn it is and the game will continue. For example, if the currentPlayer = “X”, it will assign “O” to currentPlayer.
- Check Winner function:
    - This function will use a for loop to check if there are 4 of the same symbols in a row and if there is it will return true.
    - It will use another for loop to check if there is 4 of the same symbols in a column and return true if there is.
    - It will use 2 if statements to check if there are 4 of the same symbols diagonally and return true. For example, if [0,0]; [1,1]; [2,2]; [3,3] are all the same symbols it will return true.
- Check Draw:
    - This function will check if all the buttons have clicked and if they have then it will return true to indicate if it is a draw or it will return false if it is not a draw.

Utility function

The game state utility function we have chosen to go with is the Linear Utility Function. Tic tac toe is a linear game, there is no need for exponentials. This shows the normalized utility score of each move, which is all we need.

$U = \frac{X}{M}$

(X) represents the input values, in this case the number of inputs that would be directed towards a winning move. (***M)*** represents the maximum number of inputs that is possible for any given move. In this case the number of winning configurations, meaning the number of winning rows, columns, or diagonals.

The breakdown of a 4x4 Tic tac toe and how the maximum value is calculated for the first move goes as follows:

1. Four horizontal rows
2. Four vertical columns
3. Two diagonals

We take these into account and combine all three of these columns, rows, and diagonals via addition. Such as 4 + 4 + 2 = 10 and therefore on the first move of our game has the maximum input of ten.

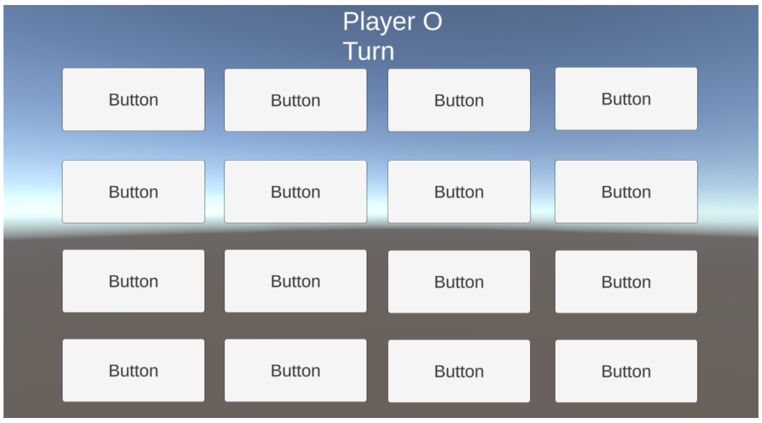

Figure 1: Starting Game State (Adam, 2024)

To further explain the function via examples. Figure 1 demonstrates the initial game state before any player has made any moves (Adam, 2024). In this current game state, there are ten possible winning input values for ***(X)*** and ***(M)*** so the equation will go as follows:

$U = \frac{10}{10}$

U = 1

Providing a value of one means that every single move as a utility value of one. Meaning that every single move is a winning move. This proves to be true as in a technical sense there is no other moves to predict yet, so the function is providing a useful output.

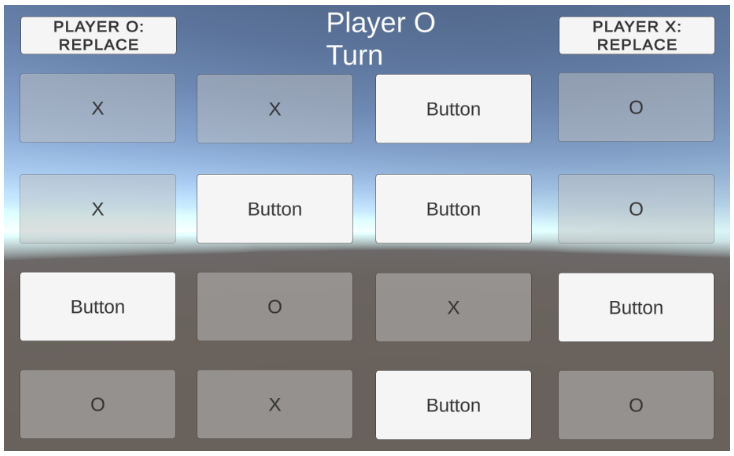

Figure 2: Game state where player O has two winning conditions (Adam, 2024)

In Figure 2 depicts a situation where player O has two winning conditions, the far-right column and the bottom left to top right diagonal (Adam, 2024). In this situation it is important for the utility function to calculate accurately as this should be a guaranteed winning game for player O. So, the utility function calculation goes as follows:

$U = \frac{X}{M}$

$U = \frac{1}{2}$

U = 0.5

(X) is equal to one, this being because in the column there is one input value possible. ***(M)*** is equal to two, as there are two possible winning configurations in this use case. To prove that this utility function calculates a viable best next move is to evaluate a non-winning move. The move chosen to evaluate is the first row third column. As an identifier key where the first number is row and second is column it would be written as [1,3]. The utility calculation goes as follows:

$U = \frac{0}{2}$

$U = 0$

Since there are no winning input values and zero divided by anything is equal to zero, means the utility function provides a viable output for our game evaluation. To evaluate an earlier game state accurately, you would need to check through a combined column and row pair, to identify the number of winning input locations, and whether the maximum number of winning configurations. In these cases, provided it is evident that the function works accurately.

References:

Adam, L. 2024. *Starting Game State* [Photograph] [Personal Collection]. Unpublished.

Adam, L. 2024. *Game state where player O has two winning conditions* [Photograph] [Personal Collection]. Unpublished.

Figure List