# Tic Tac Toe Plan

## Rules

- Two players choose a symbol “o” or “x”
- Each player takes a turn by marking a space in a 3x3 grid
- If a player gets three of their symbols in a row either horizontally, vertically or diagonally then the player wins
- If the board is full then the game is a tie
- A user can only mark empty spaces

## Variables and constants and their types

The actual game board will be initialised as a list of whitespace representing each empty space on the board. Throughout the game items in the list will be overwritten by the players as they select a space.

To allow the users to take turns I will use a boolean variable called `x` and after a players turn I will invert the value of the variable by using the line `x = not x` to change the boolean value to allow for the other player to take their turn.

I will use two user inputted variables to hold each player names as strings, and any input will be accepted.

## Use of data structures and files

The only data structure I will use is a list to hold the gameboard and the users turns. This is useful as I can easily use list indexes when I am checking the game board for winning patterns.

## User interface - how the user will interact with the program. The messages they will receive.

The user will input their move as an integer 1-9 which corresponds to a square on the board. As the list I am using has 9 objects yet begins at zero, I will take away 1 from the users input to correspond with the list index for that input, rather than confusing the user with an input of 1-8.

If a user tries to choose a place in the list that has already been taken then the program will tell them that this space has been taken and ask the user to re-input their choice. The user will also be alerted when the input is invalid, for example if it is out of range, or an incorrect data type.

When a winning combination is detected then the user is told that they have won the game.

## Algorithm flowchart

In folder

## Test plan

| Plan | | | | |
|---|---|---|---|---|
| Test Num | Purpose of Test | Test type | Test Data | Expected outcome |
| 1 | Test invalid input | Range | 13 | Invalid input message |
| 2 | Test invalid input | Range | -1 | Invalid input message |
| 3 | Test invalid input | Valid boundary | 1 | Invalid input message |
| 4 | Test invalid input | Valid boundary | 9 | Invalid input message |
| 5 | Test invalid input | Invalid boundary | 0 | Invalid input message |
| 6 | Test invalid input | Invalid boundary | 10 | Invalid input message |
| 7 | Test invalid input | Data type | “one” | Invalid input message |
| 8 | Test invalid input | Data type | “yes” | Invalid input message |
| 9 | Test valid input | Valid | 2 | User selection is marked |
| 10 | Test valid input | Valid | 5 | User selection is marked |
| 11 | Test taken index | Valid | 1 (x2) | Invalid input message |
| 12 | Test taken index | Valid | 7 (x2) | Invalid input message |
| 13 | Test win combination | Valid | | Player win message given |
| 14 | Test win combination | Valid | | Player win message given |
| 15 | Test win combination | Valid | | Player win message given |
| 16 | Test win combination | Valid | | Player win message given |
| 17 | Test win combination | Valid | | Player win message given |

| | | | | |
|---|---|---|---|---|
| 18 | Test win combination | Valid | | Player win message given |
| 19 | Test win combination | Valid | | Player win message given |
| 20 | Test win combination | Valid | | Player win message given |