Plan

Rules:

Tic Tac Toe has the following rules:

- Two players exist
- Players must alternate turns
- Each player has an assigned symbol – either a circle or cross
- The starting player is always assigned the cross symbol
- A 3×3 grid exists
- At each turn, the respective player must place their assigned symbol in an empty cell (of their choice) on the grid
- A player wins if three of his symbols align in a straight line. This could be either horizontally, vertically, or diagonally.
- A game draws if all cells are filled, and neither player has won

Planning Decisions:

a. *Variables and constants*

Name	Data Type	Usage
grid	GameGrid (custom type)	Game grid list and values of cells will be stored in this variable.
grid_dimension	int	Static constant, part of GameGrid class, which will define the dimensions of the grid. Initially, this will be set to 3, in order to generate a 3×3 grid. Grids of variable sizes can be generated by changing the value of this constant
turn_index	int	Variable which will store which user's turn it is. This will also be used to choose the appropriate symbol to be placed on the grid, when a player enters a coordinate.

b. *Use of data structures and files*

I will be using a personal data structure, GameGrid, which will be a class that deals with game mechanics. Internally, it will use a list data structure which will store the values for each cell of the grid. The list will only be one-dimensional, as it will use column-based ordering in order to map the two-dimensional game grid into a one-dimensional list.

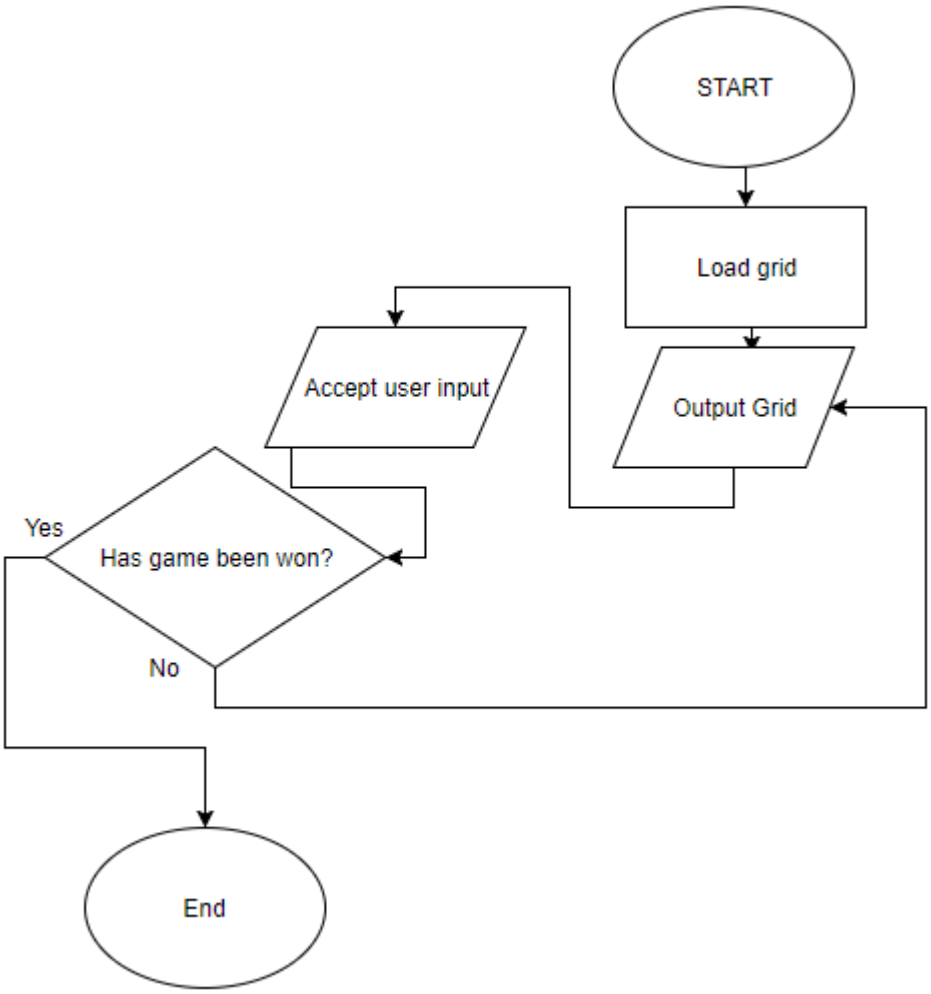
A basic list of reasons why one-dimensional lists will be used instead:

- This approach is more performant, as less objects have to be instantiated.
- The game grid is very small, so removing the need for extra lists reduces the complexity of the design.
- In C++ the use of two-dimensional arrays is discouraged. As a consequence, I prefer using one-dimensional array mappings.

c. *User interface*

My interface will be a simple CLI, which will print the grid at every turn, on the console. Users will be allowed to enter coordinates where they would like to place their symbol. They can either enter an absolute (1-dimensional) coordinate, or a 2-dimensional coordinate.

Algorithm:



Test Plan:

Test Num	Description of Test	Test Data	Expected Outcome
1	Check whether (x, y) coordinates are accepted, and accurately interpreted.	1, 2 (1, 2) 1 2	All three forms should be accepted, and a cross should accurately be placed in the middle-left square.
2	Check whether linear coordinates are accepted, and accurately interpreted.	9	9 should be interpreted as square (3, 3) – bottom-right corner.
3	Check whether the game is won accurately by a player who places 3 symbols diagonally	1 2 5 6 9	First player should win.
4	Check whether the game is drawn accurately when no lines of symbols match up.	1 2 3 4 5 6 8 7	Game should be drawn.

		9	
5	Check whether invalid data is discarded.	Hello 10 (1, 20) [empty line]	User should be requested input again, and the current round will be restarted.