

Project - Tic-Tac-Toe

Instructions

1. Answer the below question in the boxes if needed.
2. Code on your computer and zip all your code before submission.
3. Please submit the assignment through TalentLabs Learning System.

Part 0 Project Introduction

In this project, you are going to build a Tic-Tac-Toe game in the command line interface. Two players, X and O are gonna take turns in picking their next move by entering the numbers in the command line (See the screenshot on the right hand side.)

To help you in building the project, we have break down the project into 2 parts:

- Part 1: To build all the utility functions for this game including:
 - markBoard
 - printBoard
 - validateMove
 - checkWin
 - checkFull, and
 - A constant variable storing all the winning combination (winCombinations)
- Part 2: To complete the game play using the utility functions you built in Part 1.

```
Game started:
1 | 2 | 3
-----
4 | 5 | 6
-----
7 | 8 | 9

X's turn, input: 1

X | 2 | 3
-----
4 | 5 | 6
-----
7 | 8 | 9

O's turn, input: 2

X | O | 3
-----
4 | 5 | 6
-----
7 | 8 | 9

X's turn, input: 
```

Part 1 Build All the Utility Functions for the Game

1.1 Get familiar with the data structure

In this project, we are going to represent the tic-tac-toe game board using an object, with 9 properties. Each property represents 1 box in the game board, in sequential order. (See below on the usage of the data structure)

The name of the property represents the position of the box, while the value of the property represents who put their mark in the box (either "X" or "O" or "").

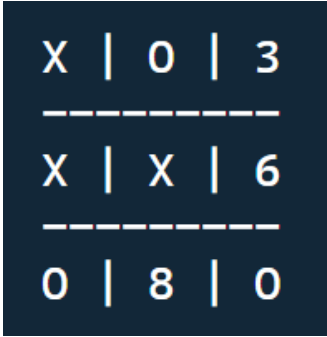
X	O	3

X	X	6

O	8	O

```
let board = {
  1: 'X', 2: 'X', 3: ' ',
  4: 'X', 5: 'O', 6: ' ',
  7: 'O', 8: ' ', 9: 'O'
};
```

1.2 Get familiar with the utility functions that you are going to build

Name of the Function	Requirements
markBoard(position, mark)	<p>Takes 2 input:</p> <ul style="list-style-type: none">• position - which is a number representing the box number• mark - the player just picked the box (X or O) <p>This function should fill in the box picked with the mark X or O.</p>
printBoard()	<p>Print out the game board in the format below. If a box is unoccupied, then you should print the number of that box.</p> 
validateMove(position)	<p>This function takes in the position that player picked as an input, and validates for the following error. This function should return true or false, with true denoting correct input.</p> <ul style="list-style-type: none">• Wrong Input (invalid position, not entering a position)• Out of bound position (smaller than 1, or larger than 9)• The position is already occupied
checkWin	<p>Check if the input user is the winner by returning true or false. True denotes the player wins.</p>
checkFull	<p>Check if the game is in a tie situation, i.e. all boxes occupied with no winner.</p>
winCombinations (not a function, but a constant variable)	<p>List out all 8 winning combinations as an array of arrays. The first one is already done for you</p>

1.3 Learn how to test your functions

We have already created some automated tests for the above function. All you need to do is to run the code file in the command line using `node tic-tac-toe_Part1.js` command, and the output in the console will tell you which test you are failing. It would report one failed test every time.

The test cases specifications start from line 69 of the `tic-tac-toe_Part1.js` file. We built the automated tests using a JavaScript Library named "assert". You don't have to worry about the setup of it, you only need to know how to read the test cases.

Reading the test cases

Test the **return value**

Compare it to the **expected value**

If they are not equal, raise an error with **error message**

```
var assert = require('assert');

board = {
  1: 'X', 2: ' ', 3: ' ',
  4: 'O', 5: 'X', 6: ' ',
  7: 'O', 8: ' ', 9: 'X'
}

assert.strictEqual(checkWin('X'), true, "checkWin() didn't work as expected with input : 'X'")
```

Reading the failing test cases

```
assert.strictEqual(checkWin('X'), true, "checkWin() didn't work as expected with input : 'X'");

assert.js:100
  throw new AssertionError(obj);
        ^
AssertionError [ERR_ASSERTION]: checkWin() didn't work as expected with input : 'X'
    at Object.<anonymous> (E:\TalentLabs\L10\project-1\tic-tac-toe_Part1.js:116:8)
    at Module._compile (internal/modules/cjs/loader.js:1133:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:977:32)
    at Module.load (internal/modules/cjs/loader.js:977:32)
    at Function.Module._load (internal/modules/cjs/loader.js:877:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:74:12)
    at internal/main/run_main_module.js:18:47 {
  generatedMessage: false,
  code: 'ERR_ASSERTION',
  actual: undefined,
  expected: true,
  operator: 'strictEqual'
}
PS E:\TalentLabs\L10\project-1>
```

error: Line 116 char 8.

The value returned by your function

The expected value returned by your function

1.4 Implementing the Part 1 of this project

Steps:

1. Open the `project` file you downloaded from TalentLabs Learning System.
2. You will need to first install the prompt-sync library. If you forgot how, please reference the lab in lab 9.
3. Start implementing the functions in `tic-tac-toe_Part1.js` following the sequence below. After implementing each of the following, run the test to make sure all the test cases of that function are passed. You can run the tests by running `node tic-tac-toe_Part1.js` in the command line.
 - `validateMove`
 - `markBoard`
 - `winCombinations` and `checkWin`
 - `checkFull`
 - `printBoard` (No automated test cases)
4. If you are seeing "All tests passed! Congratulations!" in the console output, then you have successfully implemented all the functions. Congratulations!

Part 2 Build the Game Play

In this part, you are going to implement the tic-tac-toe gameplay using the utility functions that you implemented in Part 1. Please follow the following steps for the setup.

2.1 Copy the utility functions you created to the project file

Steps:

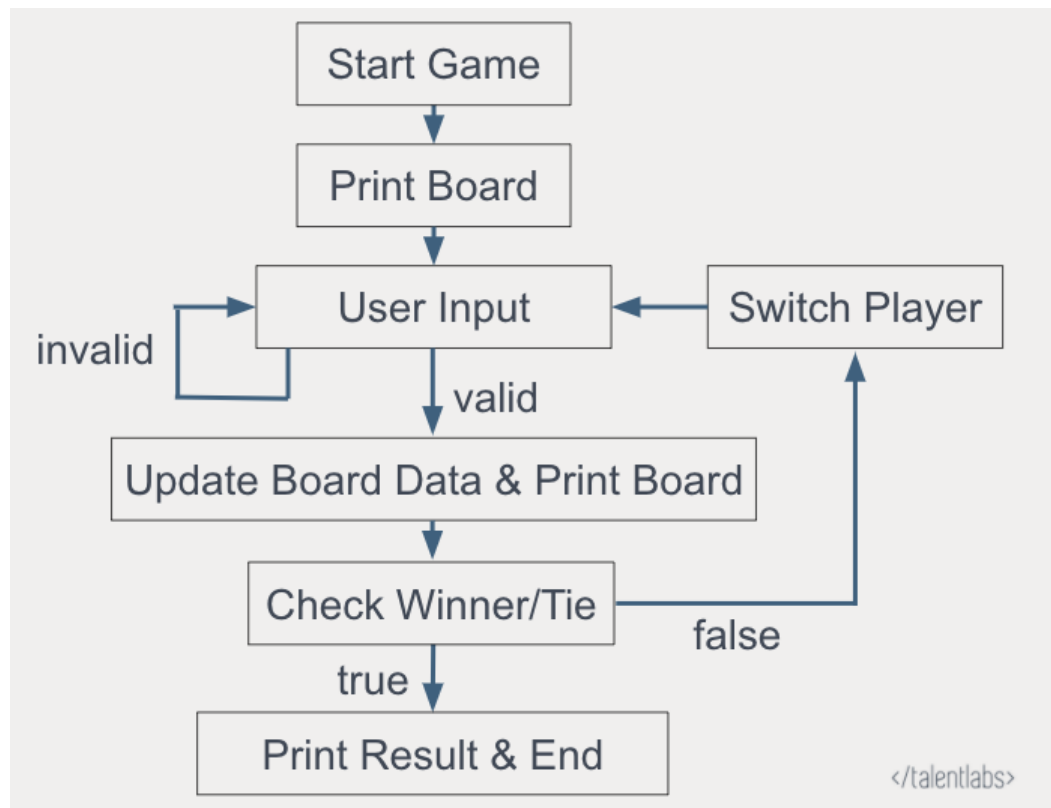
1. Open the `tic-tac-toe.js` file in the project folder in VSCode.
2. Replace all the functions in this file with the functions you implemented in Part 1 (i.e. remove line 1-63, and replace with the code from `tic-tac-toe_Part1.js` without test cases)

2.2 Implement the game play

Basically you will need to complete the game play implementation using the functions you created in Part 1. Most of the code would go into the function `playTurn(player)`. But you might also add code to the while loop at the end of the file if you find it necessary.

Try play around with the game after finishing the game to make sure it works for all cases (wrong input, winning scenarios, and tie scenarios)

If you are not too sure about the logic, you can take a look in the flowchart below. The flow chart described a recommended control flow of the game.



2.3 Bonus Point (Optional)

If you want to challenge yourself a bit, feel free to implement the restart feature for your game. Your game should ask users if they want to restart the game or not using prompt-sync. Users could respond by typing Y or N, and your program should respond accordingly.

</talentlabs>

Part 3 Submission

Zip the whole folder and upload the file to TalentLabs Learning Management System.