CS 162C Self Evaluation for Lab 1 – Tic Tac Toe

Your name: Long To Lotto Tang	Date: 6/1/2022
Are you willing to allow your code to be used in example debugging demonstrations or documentation? ☒ Yes ☐ No	

Instructions – Part 1

This document is to be turned in alongside solution of this lab. You will use this document to indicate your status on the lab, as well as areas where you are struggling conceptually or in converting concept to code. Please use the space underneath each evaluation criteria to describe any errors you are receiving or challenges you are having implementing the required functionality for your code.

Functionality

Basic Expectations	***Completed***
Does the program compile and run?	**Yes**
Yes, the program compiles and runs correctly.	
Does the program run correctly and return the expected result?	Yes
Yes, the program runs correctly and returns expected result.	
Are there comments explaining what the program and various functions are doing?	Yes
Yes, comments have been made in header, function source and main files to explain the intention of the code.	
Functions	***Completed***
Are all required functions implemented?	Yes
Except checkDraw, I have used checkWin to replace checkDraw. As in the last round (round 9), if checkWin returns false, it means that draw situation is spotted. The program will only check if there is draw situation in round 9.	
Do all functions properly return values of the appropriate type? (bool, int, void…)	Yes
Yes, all functions properly return the appropriate values and their respective type.	
Are user inputs properly validated?	Yes
Yes. getNumber() and playAgain() will ask for user data input. getNumber() will validate int type value within the user defined range, while playAgain() will validate the string and transform to lowercase to check if it is within ['yes', 'y', 'no', 'n'].	
Is the game board array properly passed into and out of functions when required? (Can't change where it shouldn't change, can change where it should)	Yes
For functions requiring the modification of the boardArray[], (e.g. initBoard(), getMove(), getBotMove()), the parameters passed without "const", while the other functions (e.g. botCheckWinLose(), showBoard()), the array parameter is passed with const as there is no need to change the content of the array.	
Does the display function properly display the board?	Yes
Yes. The 1st output will be the empty 3x3 grid, while the updated 3x3 grid marked with 'X', 'O' will be displayed through showBoard().	
Does the get move function check for an empty space before accepting an input?	Yes
Yes. The code "if (boardArray[location - 1] == NULL)" checks if the location is null, then 'X' or 'O' will be marked in the boardArray[location – 1].	
Does the check win properly check for all eight possibilities?	Yes
Yes, the program checks the horizontal (3 cases), vertical (2 cases) and diagonal (2 cases) to determine whether game_status is true from the function checkWin().	
Does the check draw properly check for no more moves?	Yes
As aforementioned, checkWin() will be used to check draw situation if round > 9 && game_status == false, indicating a draw game is found.	

Does the program offer users the opportunity to play again?	Yes
Yes, playAgain() and the outermost do-while loop terminating with “while (nextGame == true)” will determine whether user wants to play again.	

Instructions – Part 2

Please answer the following questions, in your own words, regarding your experiences throughout this lab.

Experiential Review

What aspects of this lab did you find most challenging?

The function to checkWin() is the most challenging.

```
241     //check horizontal
242     do
243     {
244         //case for position {(0,1,2),(3,4,5), (6,7,8)} and boardArray[index] is not empty
245         if ((boardArray[index] == boardArray[index+1]) && (boardArray[index] != NULL))
246         {
247              if (boardArray[index+1] == boardArray[index+2])
248                 //check 3 'O' or 'X' in a row
249                 game_status = true;
250             else
251                 //check for the next row
252                 index += 3;
253         }
254         else
255             index += 3;
```

At first, I did not include the code boardArray[index] != NULL and the code in 250-252. The program ran with logical errors as empty grids (e.g. location 0, 1, 2) were empty but the program returned game_status with true. Once I figured out the problem, the code 250-252 was critical as I missed them previously, the program checked code 245 with true, while 247 with false, the program ignored the remaining code (254-255) and jumped to check the vertical case (missing the remaining 2 rows).

The other challenging problems are similar, with the design of the code to make sure the program runs without logical errors. The checkBotWinLose() function also share the same problem. While in this function, there are much more cases to consider (e.g. check in a row by having 2 'X' or 'O' in a row, with a total 3 cases for a single row to consider – [0,1], [0,3], [2,3]). I have to modify the program based on the result generated in run-time and correct them respectively.

```
429         do
430         {
431             //check horizontal if a row having 2 'O' or 'X' to seek potential win/ lose
432             if ((boardArray[index] == boardArray[index+1]) && (boardArray[index] == choice[choiceIndex]))
433             {
434                 //if the potential win/lose grid is empty, mark the corresponding symbol
435                 if (boardArray[index+2] == NULL)
436                 {
437                     game_status = true;
438                     botIndex = index + 2;
439                 }
440
441                 else
442                     //check for the next row
443                     index += 3;
444             }
445
446             else
447
```

What concept from this lab do you feel you have the best grasp on now?

The use of array and consolidated the knowledge learnt in CS161C. At first, I have to consider whether should I use 2D array instead of 1D, but the assignment note restricted my idea. The use of array starting from index 0 is sometime confusing, but it is okay when I get used to it. The other previous concepts are data validation, random number generation, if/else are much more familiar now after this exercise.

What are some of the most common bugs encountered when using arrays? How can you avoid them?

1: index out of bound – user may read or write to array that is out of the designed memory slot. For instance arrayExample[5], but user read or write elements into arrayExample[6]. User should restrict and validate the input within the [0 – SIZE] to make sure the read and write of array is within correct memory location.

2: without initializing the array – as array is passed by reference, the same array passing through functions multiple times may under many changes. If the user wants to use that array, but he intended to use the empty array, he may get logical error at this time. He should initialize the array first if a new empty array is needed.

3: write elements with different types – array should store the same type of data. User should validate the input and store respectively.

What is exception handling and why is it used?

Exception handling is to check and response to the occurrence of exceptional cases that requires special treatments in the program, in order to prevent crashes to the program. For example data validation to prevent infinite loop in the program (e.g. switch case for the main() to check reference !=4), the preference is validated by the function getNumber() so that the number must within [1-4]. The other examples are validating char/string and see if they match with the desired result ['yes', 'y, 'no' ,n']. If there are no excepting handling, just like in the data validating cases, the invalid input may block the input buffer in console and further input will be affected and cause unexpected result. One more example is the division by 0 (not in this program), and these exception cases will causes infinite loop or crashes.

Please summarize the basic information on what Arrays are and how you can use them:

Array is a data structure consisting of a fixed size collection of elements (with same type), and can be accessed directly through a index. The allocation of memory for array is consecutive (e.g. arrayExample[0], arrayExample[1] should follow each other) in theory.

Without array, we may declare multiple variables with the same type. It may be troublesome if we have 100 marks to be stored in the program for 100 students. By using array, int studentScore[100] can be declared to minimize the clumsy declaration process. Usually, using for loop (int i = 0; i < 100; i++) to initialize, read and write elements into the array. Therefore, we can store the same type contents with less effort and we can also read particular element by studentScore[index].