HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE & ENGINEERING

COMPUTER ARCHITECTURE ASSIGNMENT REPORT

Text-based Tic-tac-toe game for two players

Instructor: Prf. Phạm Quốc Cường

Author:

Trương Tân Hào Hiệp

ID: 2011211

Semester 211, October 20, 2021

1. INTRODUCTION

This report aims to demonstrate the author's implementation of the famous Tic-tac-toe board game in a text-based version for two players using assembly language. Before proceeding to the next section, we will get a quick review about the game itself.

Tic-tac-toe is a turn-based game for two players who take turns making the spaces in the three-by-three grid with X or O. The player who succeeds in placing three of their marks in a horizontal, vertical or diagonal line is the winner.

2. OVERVIEW

As presented below in ***Figure 1***, the program starts by displaying the introduction and the universal rules for the game. Next, the program will clean up the board for potential rematch (initially, the board contains example numbers that need to be cleared). Player 1 (play as X) will always start first (as the game only have PvP mode, the player can determine their turn pre-game). For every turn, a player will be prompted to input an integer corresponding to any cell on the board as shown below.

1	2	3
4	5	6
7	8	9

The program shall proceed to check the input's validity. A move is considered valid when the input is within the set of number required and that cell is not yet occupied. If that move is valid, the cell will be marked with a symbol corresponding to the player or else the player will need to enter another valid move. Then the new board will be shown to the players so as to inform them the latest move. If the turn counter is greater than 4 (means that there is a potential winner), we will check if a player fits the winning condition, else the program will continue with the next player's turn. Should there be a winner, the program will inform the player of the victor and ask for a rematch. In case the board is filled yet there is a winner (the turn count is 9), the program will call it a tie and ask for a rematch. To accept the rematch

proposal, player can enter 0; otherwise, player can enter 1 (or anything else) to stop the program.

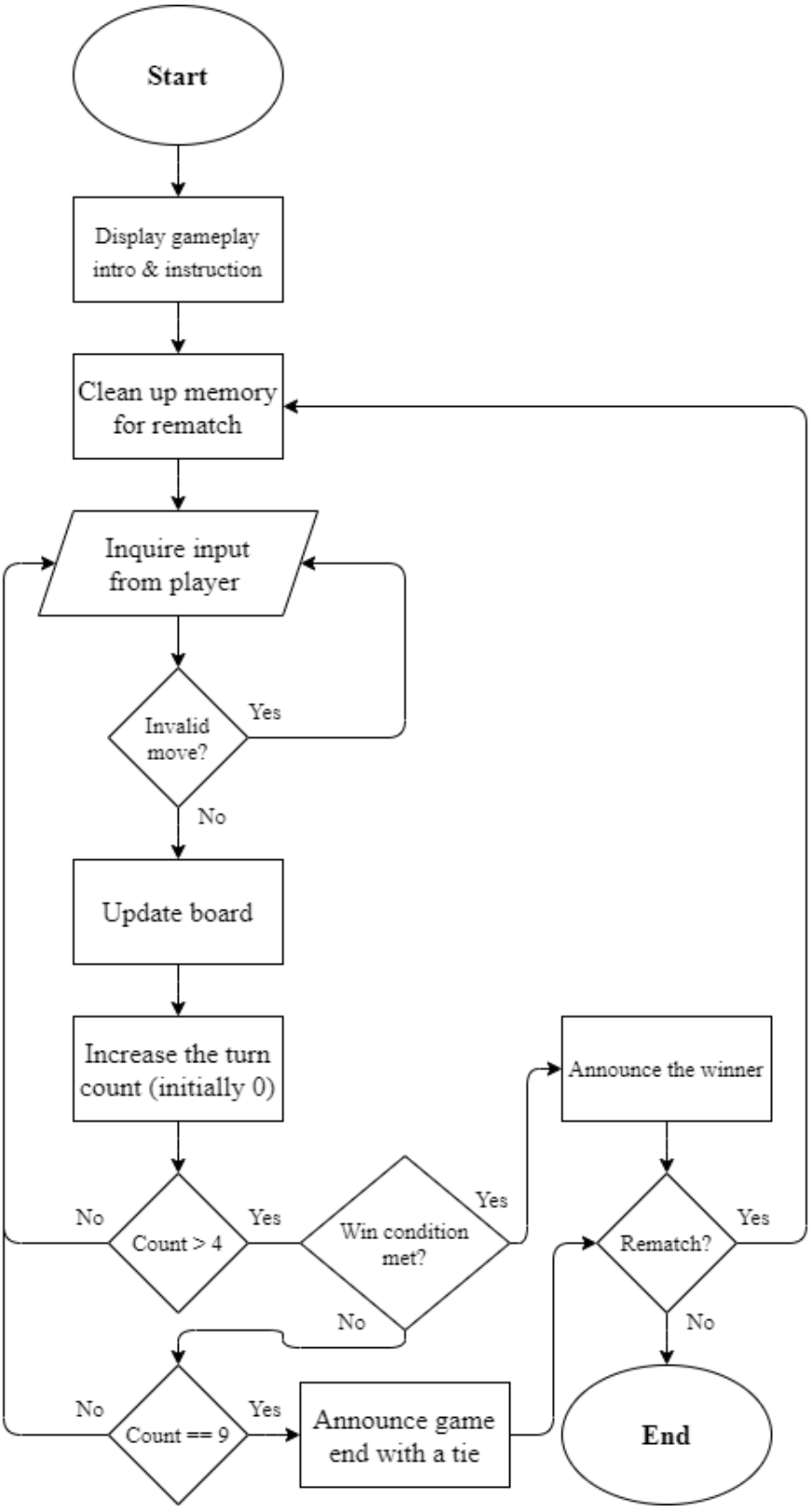

Figure 1. Overview of my program in a flowchart

3. IMPLEMENTATION

In this section, we will delve into the details of the methods used for the program, which was simplified (or omitted) in the flowchart in **Section 2**.

3.1. Board setting

The board is by far the most important thing in this program as it shares a great connection to how the program work. Unlike the initial approach I had in mind which is to create an array of characters of length 9, I instead work on a string of character which representing the board itself (both the grid and the cell) as I noticed how distributed the string can be and ***I want to try more in terms of string manipulation***. In particular, I created a string to demonstrate the following board:

```
 1 | 2 | 3
---+---+---
 4 | 5 | 6
---+---+---
 7 | 8 | 9
```

As stated above, the cell can be distributed in an ordered way as I came up with the respective formula for the cell:

$$cell\ address = string\ address + 24 * (input\ div\ 3) + 1 + 4 * (input - (input\ div\ 3) * 3)$$

Explaination:

Before proceeding, my formula work with the set from 0 to 8 so I will always reduce the input by 1. As for the player convenience, they are not required to input from 0 to 8 which can potentially cause confusion.

As there are three row, I divided the formula into three stage controlled by the key factor $24 * (input\ div\ 3)$ as I purposely create the distance between each row exactly 24 and the division for integer will not be rounded up which result in the expected level (every level has 3 elements and the set started with 0).

Because the string starts with an empty space so the increment by 1 is needed.

For every invidual element in a row, they are exactly 4 unit apart; and to get the correct position in each row, we subtract the input for the last element in the previous row.

Because I interact with the board itself so there is no need for any update method regarding the board. The result is not that much of a difference from using an array considering the small scale of the game.

3.2. Player turn's check

While having the turn counter, it is easy to control the player's turn as the first player will always goes on ***even*** turn count and vice-versa. Such thing happen because I initialize the counter as 0.

3.3. Invalid input check

The conditions to be deemed invalid are clear, the input exceed the range from 1 to 9. If the input is invalid, player will be inquired to enter another integer number until the condition is satisfied. I don't particularly want to purnish the player for any spam so there will be no penalty which can potentially create an infinite loop but with input requirement.

3.4. Occupied condition

Given that we have the cell address, compare its content with an empty character is no big deal. Should the cell be occupied, player will be inquired to enter another integer number until the condition is satisfied.

3.5. Check for winner

The condition for winning the game is clear, a player have to cross at least one row, one column or one diagonal. As stated above, I wanted to execute this program only by using string but it is not optimal so I use an array to label the board, which help a lot in this task.

In details, for every time I modify the board, I respectively change the corresponding cell index to the value 1 (player 1) or 2 (player 2). When this function is called, the value is loaded to registers and the program starts to check for every row, column and diagonal that result in a total of 8 cases (3 rows, 3 columns and 2 diagonal).

The checking process is simple yet effective, we using AND operator to compare the elements in the same row, column or diagonal respectively. If there exist a match, the program will inform the player and ask for a rematch. To determine the winner, see **Section 3.2.**