

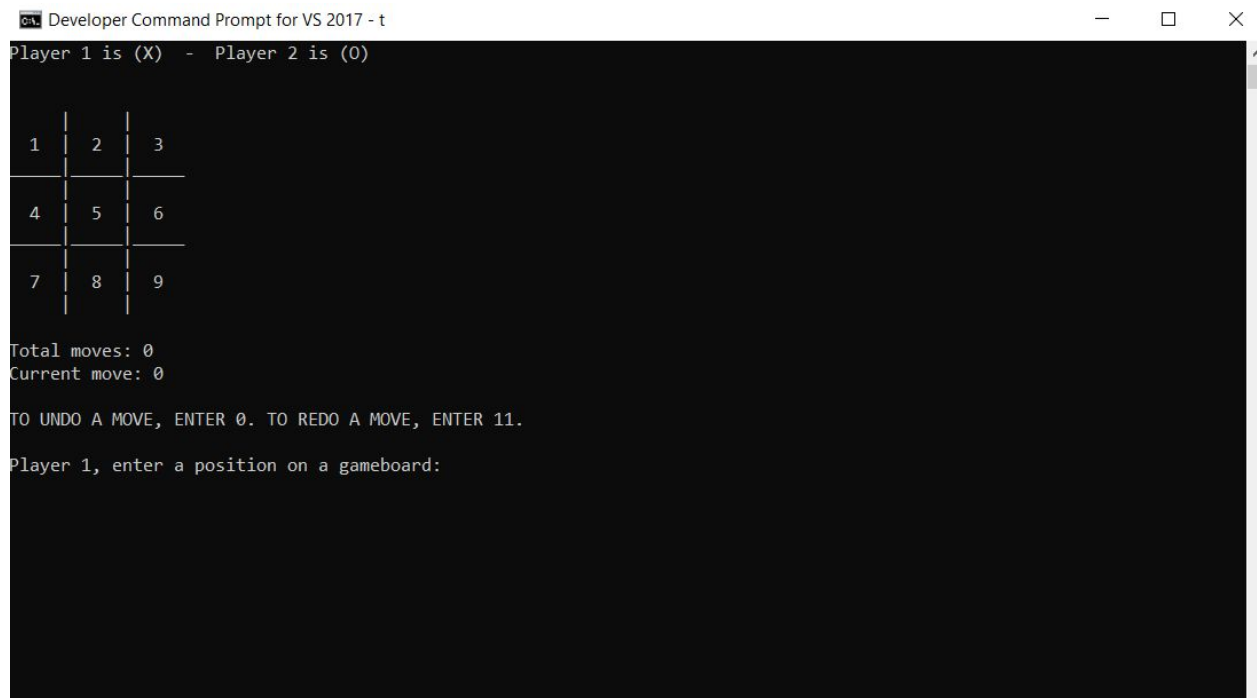
Tic Tac Toe Game in C language

Introduction:

The goal of this assignment is to make a stable & playable Tic Tac Toe game using the C language. The game should at least contain all the basics: 2 players, a board, positions and marks (X or O). In order to make the game more advanced, it should also contain a game saving feature (history of the previous games in a file for example), an undo feature (returning the game to a previous state), and a redo feature. In this report I'll explain how I implemented each feature and give an overview of how my tic tac toe game works.

Design:

I tried to design the game as simply as possible, without unnecessarily overcomplicating my codebase. When you start the game, you are brought to this screen:



```
Developer Command Prompt for VS 2017 - t
Player 1 is (X) - Player 2 is (O)

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

Total moves: 0
Current move: 0

TO UNDO A MOVE, ENTER 0. TO REDO A MOVE, ENTER 11.

Player 1, enter a position on a gameboard:
```

For convenience, I use system("cls") function in my code every time the game board is initialized. This particular function is very useful in apps like these, because it clears the screen every time it's called. In my case, it's in my gameboard() function, WHICH is in a while loop in

main, so every time the player provides input (makes a move) - everything is cleared and updated, and you get a refreshed game board (along with the move counters), without cluttering up the command prompt. Very convenient!

For this assignment I tried to stick to very simplistic data structures along with simple algorithms. You absolutely do not need to use advanced data structures & algorithms for a piece of software such as this, and I am sure that many would agree with me. In addition, I do not like coding apps like this in a very low level and old language such as C. Especially since this is a command line app, it's quite challenging to debug sometimes when something doesn't work as you expect it to. I personally had bad experiences with C, so I tend to avoid it unless I am cornered and absolutely have to use it. For most of my things, I used arrays, while loops and if-else statements. The base game (without the additional features such as game history, undo/redo) is very straightforward. The `gameboard()` function is a function that just prints the gameboard and refreshes it (`system("cls")`). There's a character array that has '1' - '9' (characters) stored in it. I use it in the `gameboard()` function in order to print it. There's a choice variable (which is an integer), and it's a user input. Based on what number the current player types, the board is updated with a mark in that position. The mark is a character (either X or O) and it depends on the player (either 1 or 2). I represent the player as an integer, and I just keep incrementing it and then checking if it divides by 2 without a remainder. If it does (even), `player = 1`, otherwise (odd) `player = 2`. That's just one of the ways to cycle between two players. The status of the game is also kept. It is an integer (a function that returns an integer), and it's set like so: bunch of if-else statements, checking all the possible combinations of 3 in a row, column and diagonally. If it detects a combination of 3 same characters, it returns 1 - which means either player 1 or 2 won. If the board is all filled with X's and O's, it returns 0 - which is a tie. Otherwise, it just returns -1 by default, which means that the game is in progress.

Regarding the implementation of game history, I simply used a file data type. At the start of my main, I declare it, give it a name, and open it with append ("a"). The append is important, because it lets me add new data to the same file without overwriting it, hence the concept of game history. It lets me store the data of all the games that have been ever played. In order to make this better, I also used a `time_t` data type in order to get the current time and date. I put this at the beginning of every game. It is just a way of separating each game from another. Every time a player makes a valid move, I use the `fprintf` function (writes to the file) to record the move by which player, which mark and which position has the mark been placed in. Simple.

Finally, regarding the implementation of the undo feature, I use an array of type `int` of size 9. This array stores the position played at the current move. Current move is just an integer as well. I just increment the current move each time a player makes a valid move. I have a bunch of if-statements, checking like so:

```
if(pos_played[current_move] == 1){square[1] = '1';}
```

The square[] is the main array, the one that corresponds to the board. That's what the board prints. So basically I just check it like so, all the way up to 9, and set the characters accordingly. I also make sure to only decrement current move if it's not 0. Otherwise it would go to the negatives, and we don't want that.

The redo feature follows more or less the same concept. I declare another integer called max_moves which represents the total number of moves that the game has. It never decrements. I need that in order to redo the data once the current move goes back. I also declare another array of size 9. This time it's a character array though, and records which mark was placed in each position. That way I know which mark to restore in a given position. So then if current_move < max_moves, we need to check if the particular mark in that particular position in the already recorded array corresponds to the mark that's currently is on the board. If so, set the mark on the board at that position to the recorded mark at that position.

```
if(mark_at_pos[pos_played[current_move + 1]] != square[pos_played[current_move + 1]]){  
    square[pos_played[current_move + 1]] = mark_at_pos[pos_played[current_move + 1]];  
}
```

And then we just increment current move by 1 outside of that statement, which should correspond to whether the current move is less than the total number of moves.

Enhancements:

Overall, I think the way I implemented my features are quite good. They are short and relatively simple. However, if I had more time, I would've tried to implement an intelligent computer AI player. It would basically be a big function, and the user would be prompted with a choice at the start of a program, whether he wants to play vs a player or a computer. By 'intelligent' I mean that it would make a move based on the players' moves and the current status of the board. Like, if the player already has 2 marks in a combo and needs 1 more, the computer would place its' mark there and prevent the player from winning. But then again, perhaps it wouldn't be possible to win against that computer. Unfortunately, I was extremely busy so my time was very limited.

Critical Evaluation:

One of the things/features that doesn't work well is the user input that the program takes with the scanf function. Basically, if the user inputs a character and not an integer, the whole app just

spazzes out and freezes for some reason. It's probably related to my user input variable being of type int. So when you input a character, it doesn't understand and goes crazy. However, the compiler doesn't give me any errors when I do something like this: `if(choice == 'u')`. So that's "if integer variable equals character". That technically should be incorrect but it doesn't give me any errors. Apart from this, everything works very well. I particularly like that `system("cls")` function that clears the screen. It just goes very well with the while loop that I have, preventing the prompt/game from being too cluttered.

Personal Evaluation:

To summarize, I wouldn't say that I learned much while doing this coursework. It wasn't harder than the apps i've made previously. Overall I just learned a few new C functions that I didn't know before, such as the `system("cls")`. I was quite stuck on the undo/redo features for a while, just trying to wrap my head around how I would implement it and make it work. Generally when something like that happens, I google for hours until I find information to help me come up with a solution. It honestly sounded much easier than done. I've asked around people for some ideas, I've even asked a few experts I personally know. One of them gave me an idea of doing a 2D/3D array for the undo/do, but I couldn't quite understand that I approach and how it would fit in my already existing codebase, so I came up with something on my own. A little primitive, but it works well. All in all, I think I did quite decent. Of course, I could've done better if I followed through with what I explained in the enhancements section above.

References:

I structured my code based on the concept that is shown in this video:

<https://www.youtube.com/watch?v=PtWMiEdtCrM>

I really liked this concept. Very simple and short, no complex data structures used and no complex algorithms applied.

