1 Purpose

Most everyone has played a computer game. How do these work? From our forays into code so far this semester hopefully you’ve gained an appreciation of the complexity that might be necessary. For this project we will work on a text-based game, the classic Tic-Tac-Toe. This game has historical implications. By making the supercomputer play tic-tac-toe, Matthew Broderick saved the world from imminent Thermonuclear destruction (Wait, was that a movie? It seemed real to me. War Games anyone??) Well anyway, the purpose of this project is to give some practice in the ideas behind programming computer games, especially representation of game objects in a computer environment.

2 Background

Tic-Tac-Toe is a game involving a 3 by 3 grid. The game is played between two players. One is ’X’ and the other is ’O’. The ’X’ player starts by placing an ’X’ in one of the nine spots on the grid. The ’O’ player then gets to place an ’O’ in one of the remaining 8 blank spots on the grid. The game continues until either the whole grid is filled up or one player has 3 of their symbols in a row: diagonally, vertically, or horizontally. I assume everyone has played tic-tac-toe.

So how can we implement Tic-Tac-Toe in C++? First, let’s think about playing the game on a piece of paper (sans computer). At each stage in the game what is the important information that is known? First there is the state of the board. On paper, we know what spots have been filled, which player filled those spots, and what spots are still open. How about using a 2-D char array to represent the board? The second piece of information that it is important to keep track of is the current turn. As long as the game is not over, the turn can either be X or O. How about using a single char variable that we call currTurn to keep track of this.

Next how should the game flow? I have attached some starting code that goes over the flow of the game and provides some function prototypes.

3 Requirements

For this project you will need to write a program in C++ which implements a simple tic-tac-toe game as described above. The following must be present in your project:

- One source code file named tictactoe.cpp
• Use `cin` (or `getline`) and `cout` to create a simple user interface to show the tic-tac-toe board after each turn. Allow the user to enter next move by picking a number between 0-8 as shown in the sample output below.

• Game is played by the rules stated in the background section above. (ie. Player 'X' starts first, etc.)

• Program uses a 2D array to represent the board.

• Program uses a while loop to allow multiple games to be played (clearing the board each time).

• At least five functions (prototypes given in the starter code):
  – `printBoard()`
  – `winningBoard()`
  – `clearBoard()`
  – `placeMove()`
  – `printMoveBoard()`

4 Grading

The following table shows the scoring that will be used for this project:

<table>
<thead>
<tr>
<th>Area</th>
<th>Max Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiles in Dev-C++</td>
<td>25</td>
</tr>
<tr>
<td>Report and Source Code Documentation</td>
<td>25</td>
</tr>
<tr>
<td>Correctness</td>
<td>25</td>
</tr>
<tr>
<td>Style</td>
<td>25</td>
</tr>
</tbody>
</table>

There are a couple chances for extra credit on this project:

• Extend the game to allow play against a computer opponent. (Would need to allow an option for this at the beginning of the gameplay as well as a decision system for the computer’s moves.) With a bit of cleverness, one can make some rules that will make winning a game impossible for the human player. (10 Points for Computer play, 10 Points extra for an unbeatable algorithm!)
5 Sample Output

The program output should look something like:

Welcome to Tic-Tac-Toe!
Enter moves as follows:
0|1|2
-----
3|4|5
-----
6|7|8

| | |
-----
| | |
-----
| | |

Player X, enter move: 0
X| | |
-----
| | |
-----
| | |

Player O, enter move: 1
X|O| |
-----
| | |
-----
| | |

Player X, enter move: 8
X|O| |
-----
| | |
-----
| | |

Player O, enter move: 4
X|O| |
-----
| | |
-----
| | |

Player X, enter move: 7
X|O| |
-----
| | |
-----
| | |

Player O, enter move: 6
X|O| |
-----
| | |
-----
| | |

Player X, enter move: 2
X|O|X
-----
| | |
-----
| | |

Player O, enter move: 5
X|O|X
-----
|O| |
-----
| | |

Player X, enter move: 3
X|O|X
-----
|O|O
-----
|X|X

Player X, enter move: 6
X|O|X
-----
|O|O
-----
|X|X

Welcome to Tic-Tac-Toe!
Enter moves as follows:
0|1|2
-----
3|4|5
-----
6|7|8

| | |
-----
| | |
-----
| | |

Player X, enter move: 0
X| | |
-----
| | |
-----
| | |

Player O, enter move: 1
X|O| |
-----
| | |
-----
| | |

Player O, enter move: 5
X|O|X
-----
|O| |
-----
| | |

Player X, enter move: 3
X|O|X
-----
|O|O
-----
|X|X

The game resulted in a tie.
Here is the final board:
X|O|X
-----
|O|O
-----
|X|X

Player X, enter move: 2
X|O|X
-----
|O| |
-----
| | |

Player O, enter move: 4
X|O| |
-----
| | |
-----
| | |

Player X, enter move: 7
X|O| |
-----
| | |
-----
| | |

Player O, enter move: 6
X|O| |
-----
| | |
-----
| | |

Would you like to play again? (Y/N)
6 Starter Code

#include <iostream>
using namespace std;

Function Prototypes

void printBoard(const char board[][3], int size);

char winningBoard(char board[][3], int size);

bool placeMove(char board[][3], int size, int move, char player);

void clearBoard(char board[][3], int size);
Main Function - Starting point for the program
 *******************************************************/
 int main()
 {
    //Set up the relevant variables
    const int BSIZE = 3;
    char myBoard[BSIZE][BSIZE];
    char currTurn = 'X';

    //Now the flow of the game
    /*---------------------------------------
     1. First, need to initialize the board
        We have a function to do this.
        ---------------------------------------*/

    /*---------------------------------------
     2. Secondly, Provide an introduction to
        the user, print out a board to show how
        to place moves.
        ---------------------------------------*/

    /*---------------------------------------
     3. Until the game is over, keep on doing:
        a. Ask player for move, check if valid
           re-ask if not valid.
        b. Place the move.
        c. Check if the game is over
           d. Switch the currTurn (X to O and vice-versa)
        ---------------------------------------*/

    /*---------------------------------------
     4. When the game is over,
        a. Print out a winning message.
        b. Ask if they would like to play again
        c. If so, go back to Step 1 (also need to reset currTurn)
        d. If not, let the program exit.
        ---------------------------------------*/
    system("PAUSE");
    return 0;
}