

CS116 - Intro to Programming, C++

Summer 1998

Project 3

Due: Friday, August 28, 1998

This short project involves completing the program for a TicTacToe game. The user plays against the computer. At first, both the user's and the computer's moves are read from the keyboard, i.e. you will play both for the user and the computer. Later you will teach the computer to play according to some strategy.

Copy the relevant files into your own directory via:

```
cp -pr ~behfar/code/project3 project3
```

To submit your finished program, do `submit project3`.

1 Getting Started

The program uses three classes:

- Class `TTT`. This is the actual game object which plays the TicTacToe game. It has a 3 by 3 board of cells, each of which is either empty, contains an `X`, or contains an `O` (this is expressed by the *state* of the cell). Member functions are provided to get the next move from the user/computer (depending on whose turn it is). Evaluation of the board is done by another member function, whose completion is your first task.
- Class `TTTBoard`. This is the 3 by 3 board on which the game is played. It keeps track of a two dimensional array of cells, and provides an interface allowing to read/write the state of the individual cells. Row/column indexing ranges from 0 to 2, e.g. the center cell has row 1 and column 1.
- Class `TTTCell`. The actual cells, of which there are 9 on the board.

Important: Take time to read the code provided for you. It is simple and self-explanatory. The `main` function simply defines a `TTT` object and runs it. The function `TTT::play()` is a good starting point to get an overview of what is going on. You will be happy to know that you don't need to change anything

in the `TTTBoard` and `TTTCell` classes, but only need to understand how they are used by the `TTT` class. The comments in the code explain this.

Test Drive. Type `make` to create the executable `tictactoe` and run it to see what it does. Moves are entered by typing in the row and then the column. For example, to place your `X` in the center, type `1 1 <return>`. At this point the only winning configuration the program recognizes is if the top row is all `X`'s.

2 Evaluating The Board

Complete `TTT::evaluateBoard()` so it correctly recognizes wins by either player. Don't forget to include the two diagonals. Also, make sure you recognize a draw, i.e. when there are no empty cells left, and there is no winner.

3 Disallowing Bad Moves

There are two types of bad moves allowed by the program at this point. The first is if a player enters bad coordinates, e.g. row 5 and column 12, in which case the program tries to write into those nonexistent array cells, causes a segmentation fault, and gets killed. The second is if a player tries to put a mark in a nonempty cell. Fix the first bug by doing proper range checking, and fix the second bug by making sure that the target cell is empty. A bad input should be rejected and the player should be prompted for another input.

4 Computer Strategies

Instead of having to type in the computer's moves, we would like to have the computer actually play the game itself. A very simple strategy would be for the computer to place its marks into randomly chosen squares. To this end, the function `random()` will be useful: it returns a random integer, which you can *mod* with a small integer n to obtain a random number in the range $\{0, \dots, n-1\}$. For example, `(random() % 10)` will have a random value between 0 and 9. *Note:* Don't forget that the computer may choose a square which is not empty, in which case it will have to try again.

5 Extra Credit

- Implement a better strategy for the computer. Optimally, the computer should always be able to either win or at least tie against any player (this is a property of the TicTacToe game). How close can you come to this?
- Once you have implemented a good strategy for the computer, you can use it to give the user a hint upon request (the way chess programs do).

Your program would allow the user to request a hint, and it would then use the computer's strategy to make a suggestion for the user's next move.

- You can try and expand the board by making it say 8 by 8. The main difficulty here will be the board evaluation method. You don't want to check every possible sequence of 3 squares, since there are too many of them on the board. Instead you only need to check the sequences of squares which include the most recently marked square. Also, on a bigger board you may want to decide that winning requires 4 or 5 consecutive squares of the same mark instead of just 3.
- Implementing a bigger board will make for longer games, so you may want to give the user the option to switch places with the computer at any given time.

Good luck!