CS4000

Homework #4: Mini Battleship Using Sockets

due Saturday, March 23rd, 2019 (at end of day)

(60 pts.)

Introduction

A classic 2 person "board" game called Battleship (see wiki page here) by Milton Bradley involved placing ships on a 10 by 10 grid and iteratively trying to sink an opponent's ship by firing at specific positions on the board. Each player would say "hit" or "miss", depending on whether a shot landed on one of their ships. When a ship was hit at all of its grid positions, it sank. The objective of the game is to sink all of your opponent's ships. In a well-played game, your opponent would not know the exact positions of your ships at the start, but would gain information during each round.

For this assignment, you will create an interactive, networked (client/server) "mini" version of the battleship game that uses the the curses and sockets libraries for the game play. The mini version game will be played on a 4 by 4 grid, and each player has exactly one ship. The game begins by players placing their ship (3 grid cells long) on their board. Unlike the traditional game of "battleship", your ship can be placed horizontally, vertically, or diagonally.

Game Play

Your client will specify, in the command line, the web-address and port number of the server. As the first, you will place your ship. For simplicity, use the arrow keys to find a location to place the ship. Once a starting location is selected, the player hits the space key. The player can then give the orientation of the ship by rotating it around the selected location via the left and right arrow keys. (Hitting the escape key should allow the user to reselect the starting location.) When the ship is positioned correctly, the user hits the space key to finalize the location.

```
Your ship:
+-+-+-+-+
|x| | | |
+-+-+-+-+
| |x| | |
+-+-+-+-+
| | |x| |
+-+-+-+-+
| | | | |
+-+-+-+-+
```

Once the ships are positioned, each player will select positions on the grid to fire upon. (Space == fire, arrows keys move the cursor). Hits will be indicated by an "H", and misses will be indicated by an "M".

```
Your ship:
+-+-+-+-+
|x| | | |
+-+-+-+-+
| |H| | |
+-+-+-+-+
| |M|x| |
+-+-+-+-+
| |M| | |
+-+-+-+-+

Your opponent:
+-+-+-+-+
|M| | | |
+-+-+-+-+
| |M| | |
+-+-+-+-+
| | | | |
+-+-+-+-+
| | | |H|
+-+-+-+-+
```

When one on the player's ships has been destroyed, the game ends, and both clients are informed of the result.

The Server

For part of this project, you will design a server that uses sockets to communicate with the battleship clients. At a minimum, the server will accept two connections (one for each client) and act as a referee between the two battleship clients. The server will receive the two initial boards from the two clients and iteratively accept game moves from each of the players (clients). The server will communicate to each of the clients whether each player's move resulted in a hit or a miss. The server will also communicate to the clients when one of the players has won the game.

Once the game is complete, the server can halt.

An example server is found in `rsp_server.cc`

Example Client

I have provided two example "client" programs. The first, `battle_example.cc`, is an example program that uses `ncurses` to control the terminal. To compile this code, issue the

command `g++ battle_example.cc -lncurses`. The second example is `rsp_client.cc`. This code uses sockets in the `boost` library to communicate with the `rsp_server.cc`. A provided `Makefile` shows how to compile this code.

Submission

Submit both your client and server code via blackboard. Make sure to provide a makefile that demonstrates how to compile your code.