

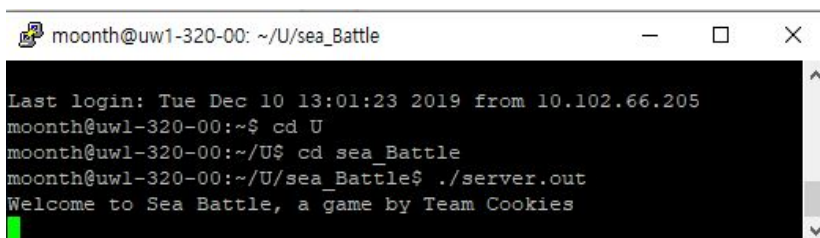
Overview:

Our team has designed a battle across the sea! Two players in a match will try to guess the location of each others' ships, and the goal is to sink the other's ships before having their own be sunk.

Each player sees their own board of empty squares, is randomly populated with ships. They can also see their opponent's board, but not the ships on that board. Players alternate turns entering locations they believe the opponent's ships reside, and the boards are updated to show hits and misses. Once a player has correctly guessed all opponent ship locations, the game ends, and that player wins the game.

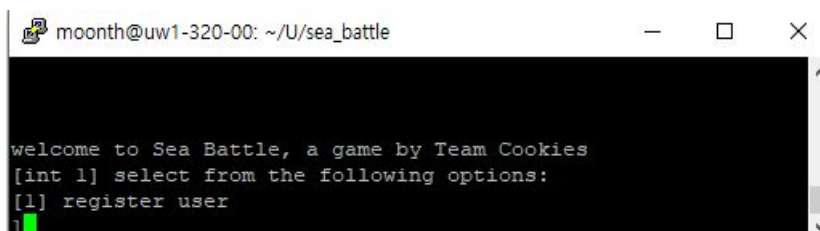
Instructions on Compiling:

1. Download the zip folder from Canvas assignment submission page or download from Google Drive (https://drive.google.com/open?id=1XCPamb3o6QrdmObe8q8_bjeZG7o3-B9I)
2. Move the zip folder to a remote Linux Lab computer. Our program runs on any "uw1-320-##" remote computer so it does not matter which remote computer you choose to move the zip folder too.
3. Run ./buildscript.sh to compile the Server, Client, and Peer applications.



```
moonth@uw1-320-00: ~/U/sea_Battle
Last login: Tue Dec 10 13:01:23 2019 from 10.102.66.205
moonth@uw1-320-00:~$ cd U
moonth@uw1-320-00:~/U$ cd sea_Battle
moonth@uw1-320-00:~/U/sea_Battle$ ./server.out
Welcome to Sea Battle, a game by Team Cookies
```

Figure 1: Window after running ./server.out



```
moonth@uw1-320-00: ~/U/sea_battle
welcome to Sea Battle, a game by Team Cookies
[int 1] select from the following options:
[1] register user
1
```

Figure 2: Window after running ./client.out

How to play the game:

1. Open 3 terminal windows, one to run the server and two to run the clients.
2. Run the server first. Navigate to the folder and run ./server.out
3. Run the two clients afterwards. Navigate to the folder and run ./client.out
4. Clients are prompted to enter the server's hostname. This is the name of the linux lab machine running the server.
 - a. This is printed on the terminal, ex: "your hostname is: uw1-320-00". [Enter "uw1-320-00" on both clients]
5. Clients are prompted to register. [Enter "1" on both clients]
6. Once registered, one client must create a game. [Enter "3" on the first client.]
 - a. It is now waiting for a connection and may be set aside until the game begins.
7. The second client may now join the first client's game. [Enter "4" on the second client]
 - a. Open games are shown, ex: "names: Illloyd". [Enter Illloyd on the second client]
8. The game has begun! Bring back the client that has been set aside, it is now a peer.
9. To launch an attack, enter a coordinate pair on each peer. [a-f][1-6]
10. The game ends when all of one's player's ships have been sunk.
11. Exiting the game will return to the menu. ['ctrl-c']
12. Exiting the menu will return to the console. [Enter '5'] or ['ctrl-c']

Player 1 Map view

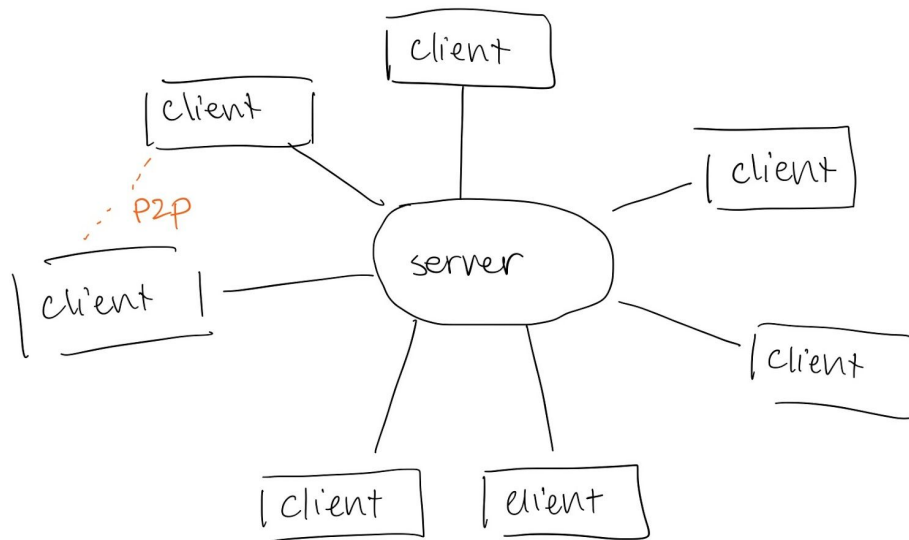
```
key: -sea    Oship    +miss    Xhit
Field:
  1 2 3 4 5 6
a 0 + 0 0 - 0
b 0 - - - - -
c - - - - -
d - 0 - - - -
e - - - - 0 0
f - - 0 - 0 -
Map:
  1 2 3 4 5 6
a - - - - -
b - - - - -
c - - - - -
d X - - - - -
e - - - - -
f - - - - -
press enter to continue
[char a-f][int 1-6] launch an attack on coordinate:
```

Player 2 Map view

```
key: -sea    Oship    +miss    Xhit
Field:
  1 2 3 4 5 6
a - - 0 - - -
b - 0 - - - -
c - - - 0 - 0
d X - - - 0 -
e - - - 0 0 -
f - 0 - 0 - -
Map:
  1 2 3 4 5 6
a - + - - - -
b - - - - -
c - - - - -
d - - - - -
e - - - - -
f - - - - -
press enter to continue
[char a-f][int 1-6] launch an attack on coordinate:
```

- The players play the game by entering coordinate pairs and trying to guess each other's battleships location.
- Field is the player's board. Map is the opponent's board.
- 0 marks battleships. X marks hits. - marks sea. + marks misses.
- The two clients will now establish a P2P connection. Board placement is chosen for the user. The field contains ships that you own, and the Map shows your attacks. Boards update each turn.

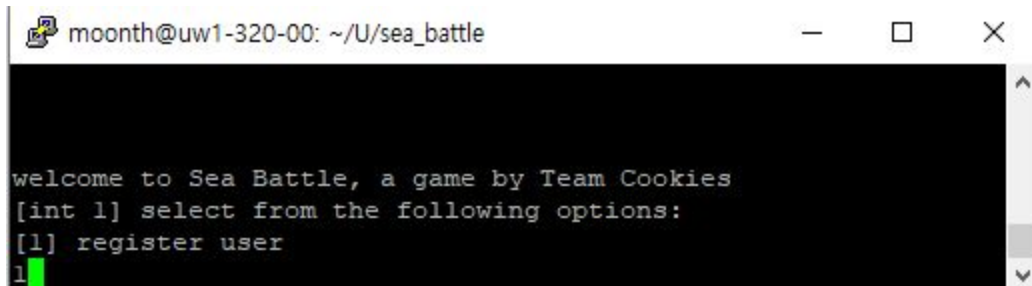
Network protocol:



- The server and client programs exist as a matchmaking service for players. They run on the server-client architecture, which is designed for a server to store a collected database of open games to distribute to clients. The server creates a pthread for each client, allowing individual request handling for many clients while accessing one shared database.
- Once clients have found matches, a peer-to-peer architecture is constructed. This design allows two players to request services from each other (i.e. player one receives player two's attack coordinates and vice versa).

Description of each method implemented:

1. Register user



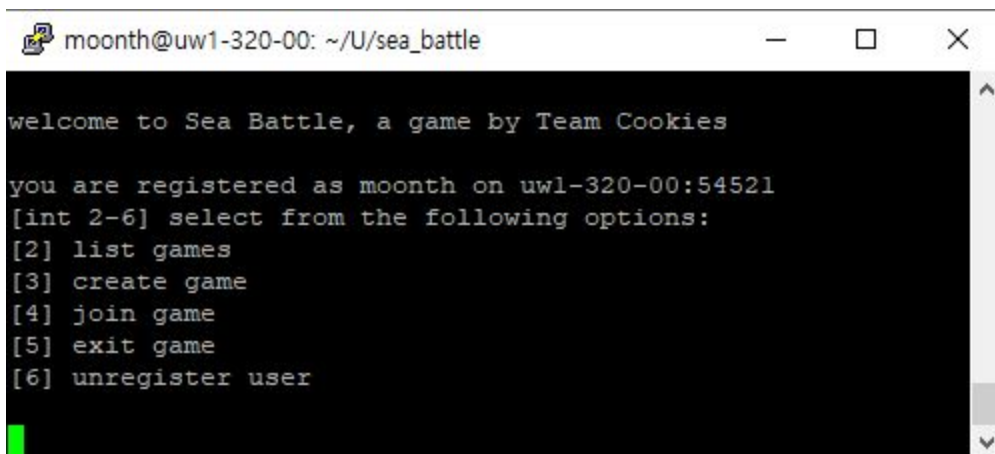
```
moonth@uw1-320-00: ~/U/sea_battle

welcome to Sea Battle, a game by Team Cookies
[int 1] select from the following options:
[1] register user
1
```

Figure 3: Client side after run ./client.out and registering

- Retrieves the user's login name using getlogin_r()
- Generates a random port number for the user using rand()
- Retrieves the computer's hostname using gethostname()
- Stores these values locally on the client, and marks it as registered

2. Game menu



```
moonth@uw1-320-00: ~/U/sea_battle

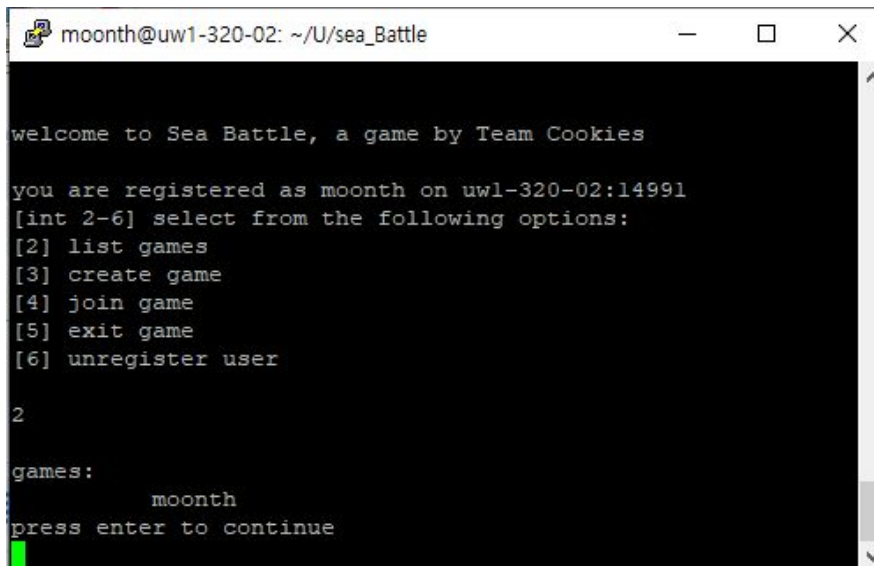
welcome to Sea Battle, a game by Team Cookies

you are registered as moonth on uw1-320-00:54521
[int 2-6] select from the following options:
[2] list games
[3] create game
[4] join game
[5] exit game
[6] unregister user
```

Figure 4: List of options after registering

- Registered users have access to 5 options. Each one has its own error handler. Chosen options are sent to the server, allowing it to run the corresponding server-side functions.

3. List Games



```
moonth@uw1-320-02: ~/U/sea_Battle

welcome to Sea Battle, a game by Team Cookies

you are registered as moonth on uw1-320-02:14991
[int 2-6] select from the following options:
[2] list games
[3] create game
[4] join game
[5] exit game
[6] unregister user

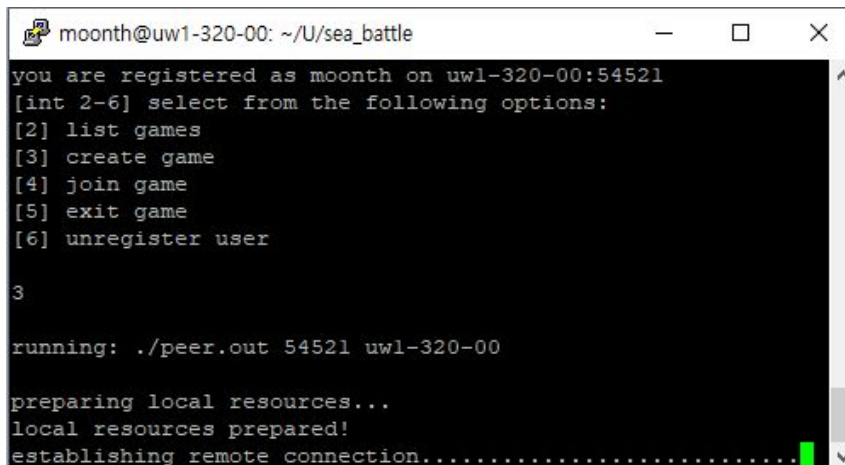
2

games:
    moonth
press enter to continue
```

Figure 6: List all games created

- Server parses the database of usernames in open games, and sends it to the client to view.
 - Open games are games which have been created, but not joined by a second player.

4. Create Game



```
moonth@uw1-320-00: ~/U/sea_battle

you are registered as moonth on uw1-320-00:54521
[int 2-6] select from the following options:
[2] list games
[3] create game
[4] join game
[5] exit game
[6] unregister user

3

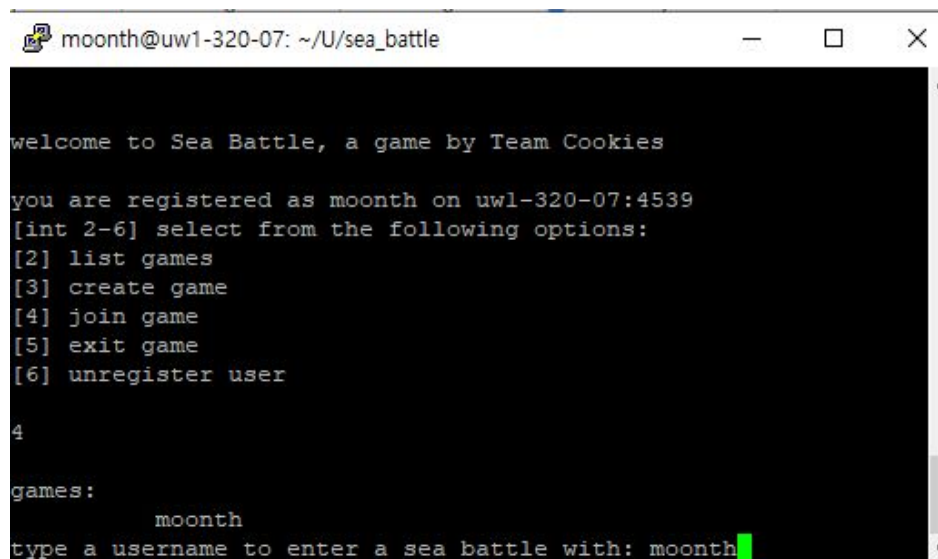
running: ./peer.out 54521 uw1-320-00

preparing local resources...
local resources prepared!
establishing remote connection.....
```

Figure 5: The player creates the game and wait opponent for an opponent to join

- Sends the user's connection information to the server to be logged into the database of open games.
- Makes a system call to run `./peer.out` using the player's connection information.
- Players will be on the loading screen until someone joins their game

5. Join Game



```
moonth@uw1-320-07: ~/U/sea_battle

welcome to Sea Battle, a game by Team Cookies

you are registered as moonth on uw1-320-07:4539
[int 2-6] select from the following options:
[2] list games
[3] create game
[4] join game
[5] exit game
[6] unregister user

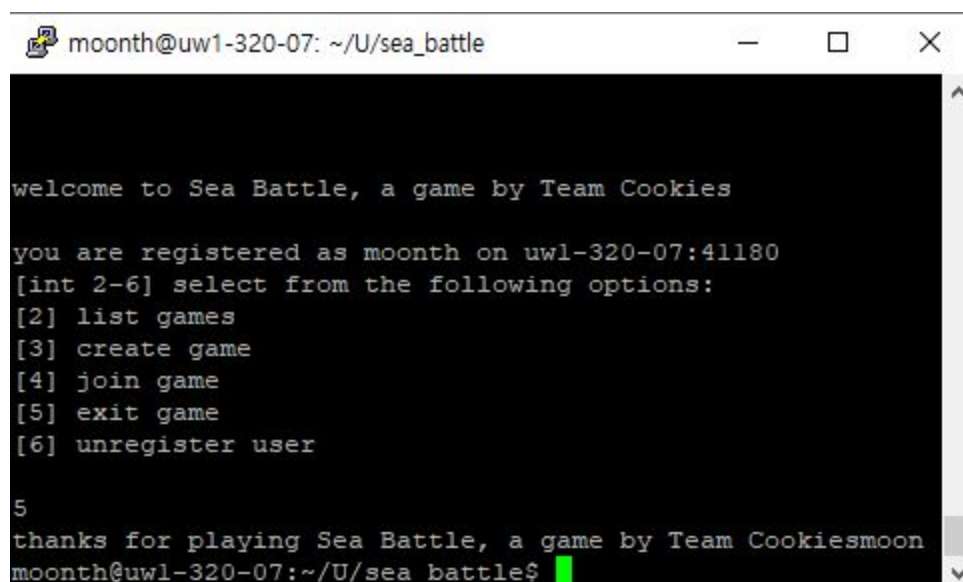
4

games:
    moonth
type a username to enter a sea battle with: moonth
```

Figure 6: After user has pressed option 4 to join game, the client will print usernames in the database (See: List Games) and then prompts the player to enter the username the opponent they want to play against

- Client calls the listGame method to shows open games that are available to play
- Prompts the player to enter the opponent's username
- Makes a system call to run ./peer.out using the player's connection information AND the opponent's connection information (retrieved by their username).
- Erases the opponent's entry in the database of open games

6. Exit Game



```
moonth@uw1-320-07: ~/U/sea_battle

welcome to Sea Battle, a game by Team Cookies

you are registered as moonth on uw1-320-07:41180
[int 2-6] select from the following options:
[2] list games
[3] create game
[4] join game
[5] exit game
[6] unregister user

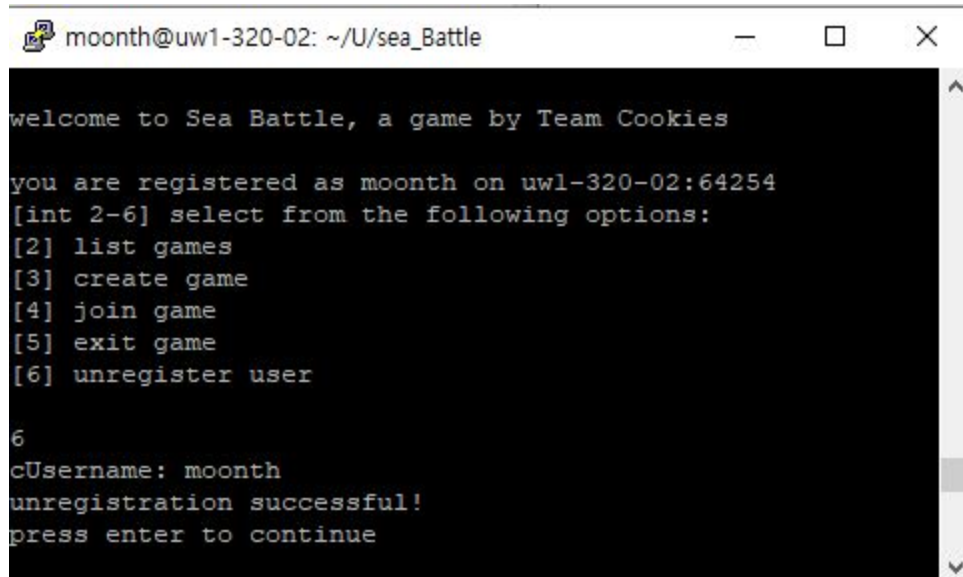
5

thanks for playing Sea Battle, a game by Team Cookiesmoon
moonth@uw1-320-07:~/U/sea_battle$
```

Figure 10: The player would return to the terminal after exiting game.

- While playing a game, the P2P connection is closed and the player returns to the main menu
- While at the menu, the client-server connection is closed. The client sends a signal to the server and the server will close the thread.

7. Unregister user



```
moonth@uw1-320-02: ~/U/sea_Battle

welcome to Sea Battle, a game by Team Cookies

you are registered as moonth on uw1-320-02:64254
[int 2-6] select from the following options:
[2] list games
[3] create game
[4] join game
[5] exit game
[6] unregister user

6
cUsername: moonth
unregistration successful!
press enter to continue
```

Figure 11: Unregisters the user from the game and return the player to the register screen.

- The player removes their connection information and marks themselves as unregistered.
- As an unregistered user, they only have one option on the menu (to re-register).

Additional files:

- Peer.cpp
 - This is the actual game which players play on a peer to peer connection.
 - Interestingly, it may be run standalone, by running ./peer.cpp with the connection information of both players.
 - Uses std::async to allow asynchronous functions to run, such as player one and player two entering coordinates (two functions running at the same time).
 - Handles the connection between the two players.
- Board.cpp
 - Creates the board for the players. Board is created on a 6x6 grid and battleships and the sea are randomized so each time, no game should be the same.
 - Handles attack requests for the players and lets the players know if the coordinate pair they entered was a hit or a miss.
- Static.cpp
 - A list of static functions used throughout many of the files. These may include function wrappers for error checking, setup methods for servers and clients, support for fancy console output, etc.

Things changed after the presentation:

1. Implemented exitGame. Allows players to exit the game cleanly. Exit game will return the user to the terminal screen.
2. Previously our game would crash if players would enter any invalid arguments or exit the game by closing or using "ctrl-c".
3. Lots and lots of error handling.
4. Added integrity checking to transmission between server, clients, and between peers. When data is sent, it will travel back and forth once. The returned data will be checked with the sent data and see if the data was sent correctly. The program will throw an error if the data was not sent correctly.