BattleFleet Documentation

Susanna Byun && Alex Carswell

Instructions

Compiling and Running Code

1. Simply run the executable that is attached to the submission (make sure that the Data folder and .dll file are in the same folder as the executable which are also in the submission).

How to play game

1. For first time users, choose a username and password and click register.
   For users that have registered previously, enter your username and password and click log on.
2. Enter the ip address of the server (this will be given in real time as the server is run in different places each time).
3. Once logged on, either create a game or join a game that already exists.
4. After you have joined the game, you will be taken to the battle fleet placement page. Place your battle fleets on the grid to the left. The grid to the right represents your opponent's battle galaxy. Once done with placing all battle fleets, press the button "Finished Placement" to let the server know you are ready to play. The game will only proceed when both players have finished placement of fleets.
5. Once both players are ready, the game starts. If it is your turn, click one square coordinate on the right side (your opponent's side) to fire. If it is a hit, the square will turn red. If it is a miss, the square will become gray.
6. Continue firing shots until one of the players have lost all battle fleets. However, if one of the players don't choose a place to fire for 30 seconds, timeout will occur, the game will end, and the opponent will win.
7. The leaderboard will be displayed. Click close on the leaderboard screen to return to the game lobby.
8. Click exit game to log out and go back to the main menu.

Protocol Description

General Overview of Server

- In the main function of the server, the socket constantly listens for a client on port number 8888 and the local IP.

- Each time the socket accepts a new TCP client, the main function starts a new thread to handle that client, using the ClientHandler function that is run as a thread.
- The main thread runs in an infinite loop to continuously listen for new connections.
- Once the ClientHandler thread has started, you start listening for a message, which is read through a ReadMessage function. This happens in an infinite while loop.
- Within the while loop, the message that is read is analyzed further down and the parameters of the message are passed to the appropriate function that is called to handle the message. More information will be given down below.
- In order to have centralized data that can be reached by all the different threads, there exists a static class called GameData. Inside this class, there are several Lists, Dictionaries, and Queues that can be accessed by all threads.
- Inside the GameData static class, there is the PlayerDictionary, which acts as a small, local database of all players that are registered.
- Inside the GameData static class, there is a GameDictionary, which keeps track of all the games that have been created and are not yet over.
- Inside the GameData static class, there is a ConnectionList, which is actually a dictionary where the key is the player and the NetworkStream that is used to read and write to the player is the value.
- Inside the GameData static class, there is a LobbyPlayersList, which consists of all the people in the lobby and not yet in a game. This is so that a thread can read and write to a player at any point in the game, even if the sender is not given the network stream as a parameter (it may have been easier to just send the networkstream as a parameter, but this was an initial design decision that was chosen for a different reason that restricted the passing of multiple parameters. However, we decided not to do it that way, no longer restricting us, but making it a hassle to change the way we've already implemented it).
- Inside the GameData static class, there is a ChangeQueue, which is a queue of all changes and chat messages that need to be pushed to all people in the lobby.
- There is a room counter that determines the room number for any newly created game room.
- All these things are instantiated when the server starts running. If the server closes, all data is lost, but if the server is left running, all the data is saved and used to make the appropriate decisions.

Register

The protocol for register is of the form “reg username password(encrypted)” which is sent to the server whenever a player enters a username and password and clicks the “Register” button in the opening scene of the game.

Login

Login is sent in the form “login username password(encrypted)” every time a user presses the login button and has filled in the correct input fields.

List Games

Each time a player enters the game lobby, the server automatically sends them a list of messages of the form “game update room# hostName gameName gameStatus.” This series of messages is only sent when the user enters the lobby.

Exit Lobby

Exit lobby is sent to the server before a player exits the lobby when creating a game, it is in the form “exitlobby userName.”

Create Game

When a player creates a game by inputting a game name and pressing the “Create game button” on the lobby page they are moved to the game scene. Here, when they first join, they reconnect and re-authenticate with the server. They then send a message of the form “creategame userName.”

Join Game

When a player joins a game they are first moved to the game scene where they re-authenticate and reconnect with the server. They then send a message in the form “joingame userName room#.”

Exit Game

The exit game closes the connection on the server end and takes the player back to the main menu.

Unregister

The unregister protocol removes the player from the player dictionary and closes the game back to the main menu.

Application Specific Protocol

Fire

The fire protocol is sent when a player fires at a coordinate on the battlefield. Then, that same message is sent to the player that is being hit, to notify that they have been fired at. The message is in the form of “fire [username] [room number] [coordinate letter]”.

Move

Once a player receives a Fire message, they send back a message detailing whether that was a hit or a miss, based on the placement of their ships. The message comes in the format of “move [hit/miss] [cont/lose]”. The hit and miss determines whether the player that fired gets a score added to their local game score. If a hit move was a winning move, the loser sends a lose parameter and the person that fired gets a win parameter.

GameReady

When one client tells the server that they are ready, it will wait in a loop checking if the game room object has both players being ready. Once the second player submits the message that they are GameReady, both players are set to ready and the while loop is released. Then, the message to MakeMove is sent to the host player.

Make move

This is sent to one client when one turn cycle has been completed. It is of the form “makemove” and simply lets the client know that they can now take their turn.

Global chat

Global chat isn’t currently working in the game’s current state, however, the message we designed and began to work on implementing is of the form “globalchat userName message.” It would be sent whenever a user typed into the message bar in the lobby scene and pressed enter. Due to thread desynchronization issues we were forced to remove it from the game in its current state.