Poker game

(Sprint2 Retrospective)

https://github.com/srhee91/PokerGame

Team 3

So Mi Choi, Bo Heon Jeong, Hanchen Li,
Sang Rhee, Yixin Wang, Yuying Wang

Sprint2 Retrospective

1. Description of tasks

Implemented and working:

1. Make the *GameSystem* class able to update the *GameState* depending on received *PlayerAction* according to the rules of Texas Hold'em. (e.g. updating chip amounts, calculating/distributing split pots, shuffling/dealing cards).

   We implemented a total four nested loops for the main flow of the *GameSystem* class: game -> hand -> round -> turn. The first step was to figure out where we change the game information. For example, when the game starts, the player info and the whole game system object need to be created, and for each hand, we need to create a new deck to shuffle and deal new hands and flop. After we figure out where everything goes to, we implemented all of them in Host by calling methods of the *GameSystem* class.

2. Make the *GameSystem* class able to determine the proper order of player turns based on the *PlayerAction* it receives and to send out *GameStates* prompting the players when it's their turn.

   In order to keep the proper order of player turns, we created a variable called *highestBetter*. Its initial value is set to an index of the first turn player which is the person next to dealer, or the person next to the big blinder. Whenever someone raise the bet, *highestBetter* is reset to the index of the raiser. The round will end when the turn is back to the *highestBetter*. At every turn, the *GameState* will be sent to every player and the host will request *playerAction* to its current turn player.

3. Implement Host class to the point where *GameSystem* can interact with *HostMessageHandler* to send *GameState* and receive *PlayerAction*.

   The Host class's main method has been implemented to run in its own process apart from the host client. After being invoked, it first establishes a connection with the host client, allowing it to retrieve the name of the hosting player. Then, it enables the *HostBroadcaster* to respond to the *HostSearcher* modules of any clients looking for game lobbies to join. Next, it awaits to receive the start signal from the host client, at which point it will send a start signal to all other clients. Next, the *HostBroadcaster* is disabled and the *GameSystem* takes over the polling and handling of *UserAction* from the clients.
   The reliability of the communications between the host and clients is in need of improvement. Receiving an object from the host sometimes takes up to 10 seconds for a client. We don't know if this is a result of Java.

4. Modify Rank class to be able to compare players' best-hands for all possibilities and finds the winner.

   The *findWinner*() method was implemented in Rank class, which merges the flop, turn, and river with the player's cards to calculate and identify the player who has the best-hand using *findBestHand*() method. If two or more players' best hands have the same rank, *compareHand*() method is called to identify highest-rank among them.

5. Integrate *ClientMessageHandler* methods into the GUI *ComponentListeners* so that button presses in the GUI will cause the proper *ClientMessageHandler* action to occur.

   ClientMessageHandler actions have been integrated into the GUI, which include enabling/disabling the *HostSearcher*, sending/receiving the host client's "start" signal, and sending *UserAction* to the host when one of the four action buttons in the GUI in *OngoingMode* is pressed.

6. Add a *searchForAvailableHost*() method that can search the whole subnet and return an array of IP addresses with Poker game lobbies. Concurrency will be used to make sure this can be done in a few seconds.

   It is able to search all available host IPs, by adding two classes and using sockets. One is *HostBroadcaster* class that opens another port on host player to broadcast the table information to everyone connected. The other is *HostSearcher* class that loops all IP addresses in the subnet trying to connect. It has a string array representing 255 IP addresses and storing the name returned by searched host. It is able to make search be done quickly by creating threads for each search.

7. Add a UI element (either in its own GUI mode or as a popup in *StartMode*() that will display a list of Poker game lobbies that a player can join after *searchForAvailableHost*() returns. The user can select the lobby they want to join.

   A new mode called *JoinMode* was added that is meant to appear between *StartMode* and *LobbyMode* if a player chooses to join a game. This mode shows a list of all game lobbies it found through the *HostSearcher* and displays the IP address and the name of the host of each lobby. The list of game lobbies is displayed 8 at a time, with two buttons for moving to the next or previous page of results. A Refresh button was implemented that retrieves an updated list of game lobbies from the *HostSearcher*.

Implemented but did not work well:

N/A

Not Implemented:

N/A

Extra tasks done:

N/A

2. How to improve

1. The network components could be improved to make the game respond more quickly and reliably by sending data packets more efficiently.

2. More exceptions and error conditions could be handled throughout the game such as handling clients leaving, the host crashing, or packets being lost.

3. Remove hosts that no longer exist or who have already started a game from the list of joinable lobbies in *JoinMode* or mark them as unavailable.