Poker game

(Sprint3 Retrospective)

https://github.com/srhee91/PokerGame

Team 3

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

Sprint3 Retrospective

1. Description of tasks

Implemented and working:

1. Integrate the already-existing *OngoingMode* animations to correctly respond to received *GameState* from the host to show each player's actions.

   Specific animations are activated based on differences in state variables between a newly-received *GameState* and the previous *GameState*. Additional *GameStates* and other data are sent by the host to trigger different steps in longer animations (e.g. when winnings are distributed, each *sidepot* requires an animation) so that pauses between them can be handled by the host where it's easier to implement. For simplicity, all animations are triggered by the receipt of a G*ameState* and never directly by a GUI button.

2. Expand the *GameState* class to include each player's most recent action so that the *OngoingMode* GUI can accurately display status labels for each player.

   We added a new variable of type *UserAction* called *latestAction* for each players so that we can save the most recent action of each player. If there was no latest action or the new round starts, *latestAction* is set to Null and the GUI will only display the user's *latestAction* if it is not null.

3. Add a timer in *OngoingMode* that starts when a player's turn starts so the GUI can show how much time that player has remaining to make a decision. If the timer expires for the controlling player's turn, the GUI will execute the fold action automatically.

   A timer was added on the host side in *HostMessageHandler* by using Java's Timer class. A variable *allowedPlayer* stores the name of the player of the current turn. It represents the player who can send an action in this turn. An *autoResponse* thread is created when that player's turn starts that will send a "fold" *UserAction* to Host when it is called. Specifically, in the send method, if the game is in *OngoingMode* and an action is received from *allowedPlayer*, it will start counting 25 seconds using timer schedule method. Timer will be cancelled when *HostMessageHandler* receives the *allowedPlayer's* action.
   A timer was also added the client side for the purpose of showing the player how much time someone has left. The timer was implemented by polling Java's system time before rendering each new frame. If the timer expires, a "fold" *UserAction* is sent to the host as if the player has folded.

4. Add a Quit button to the *OngoingMode* GUI that takes the player back to *StartMode*.

   A "Leave" button was added to *OngoingMode* and *LobbyMode* that would show a popup asking the user for confirmation. If confirmed, the *ClientMessageHandler* was closed and the GUI switches to *StartMode*.

5. Improve the host to be able to handle disconnecting clients (either from willingly quitting or from network errors) and to update the *GameSystem* accordingly, removing that player from the game.

   A timer was added to *HostMessageHandler* to fold for a player if they do not respond. If a player disconnects, this timer will eventually expire and that player will fold. At the end of every hand, the Host will check the connections to all players and will detect that a player has left. Their name will be removed from the list of players. Future *GameStates* that are sent to the clients will reflect this change, allowing every player's GUI to no longer display the name of the player who has left.

6. Improve the host to be able to handle disconnecting clients (either from willingly quitting or from network errors) and to update the *GameSystem* accordingly, removing that player from the game.

   HostMessageHandler knows whenever a client gets disconnected. In the backend, the host checks every end of each hand for the connection to each players and when it finds a disconnected player, the player becomes Null to be removed from the game.

7. Debug the GUI portion of the game by play-testing the game with all possible user inputs.

   The Host and *OngoingMode* have been modified repeatedly due to new animation bugs in the GUI being discovered. The Host game sequence has now been modified to specifically work with *OngoingMode* animations. The major changes made to the game sequence in Host has eliminated all major animation bugs on the client side.

8. Debug the *GameSystem* portion of the game by play-testing the game with all possible user inputs.

   We created the HostTestInText2 class that uses the major parts of *GameSystem* portion of the game. In HostTestInText2, random *UserActions* with random bet amounts were sent to the *GameSystem* until the game ended. While playing the game, we artificially created special circumstances such as split-pot, showdown, fold-by-winner, etc. for testing purposes.

9. Debug the Network (*ClientMessageHandler*, *HostMessageHandler*, *HostBroadcaster*, *HostSearcher*) portion of the game by simulating network issues and disconnects and host crashing.

   We found a way to improve network performance. Instead of broadcasting frequently (every 1 second), we made the *HostBroadcaster* broadcast with a longer interval (10 seconds). When a new *HostSearcher* starts, it checks all the IP addresses through the whole subnet once. When a new *HostBroadcaster* starts, it broadcasts to all the IP addresses through the whole subnet once. This change reduce the amount of data on network significantly, which reduced the failure rate when searching for game lobbies significantly.

Implemented but did not work well:

N/A

Not Implemented:

N/A

Extra tasks done:

1. Card Sound
   Sounds were added for the card and chip animations using *AudioStream*.

2. Animation
   In the GUI, the winning hand of the winner(s) of each pot/*sidepot* is displayed as labels before that pot is distributed.

3. Executable
   We created an executable so users can run it with a double click, instead of run java in command-line with a lot of obscure arguments.

2. How to improve

1. Find more realistic sound effects to use for the chip and card sounds.

2. Make the project executable on Mac by writing an extra loader in Objective C.

3. Make the game window resizable and have the GUI elements properly reposition themselves.