**MVC model**

* **How we collaborate**

In our project, we make use of MVC model, where we separate the game model, UI and control into different part. In this way, changing implementation of each part won’t affect each other since we use the same interface to send parameters to each other. Therefore, we can easily work on our own part and put them together at the end easily. In our main controller, we stored the model and UI object and wrote plenty of connection methods to allow UI to access the information it needed to show without knowing any information on what model structure is. Furthermore, model also can trigger an update on UI when the data change just by sending the data back to the controller.

**Model**

* **Game Model**

For our game model, we separate it into client part and server part. The reason for that is client only need to know just enough of information they need to know to prevent too much data calculation that might desynchronized the data. In client model, it keeps track of all game information such as user list, game status and chat history. All data operation will be executed through data pack from the server. In server model, we keep track of the remain points to give to the next guesser and the guessing word dictionary because the clients do not need these data to run the game. Furthermore, all game mechanic handling such as word guessing and switching between game state are also handle in the server model. They will all be processed on server and then sent the result back to the client.

* **User Model**

Within our game model, we contain a list of users in the room. On client side, we only store a list of client user model with their name, current score and their assigned ID. However, on server side, we store more information such as their player status and whether if they have guessed out the word.

**Control**

* **Config File**

When user first launch our game, the default config file will be generated into our root directory of the project. Even though there isn’t much custom things user can change for their preference, having a config file allows us to store data that we can use in the future. First of all, users game name will be remembered after they first join or create the game. Therefore, they don't need to enter their name repeatedly for further different game. Furthermore, the address and port they have joined last time will be recorded. In this way, user can play with their friends’ multiple times easily without need to re-enter the server information. If user is a host, the dictionary file will also be stored. The config file allows us to carry out the concept of user friendly, which we keep complex process of repeating entering data as a single time only. Therefore, the user can spend more time to play with their friends instead of checking the address every time they open the program.

* **Connection between multiple clients and server**

In our game, as we want to give user a clear visualization of what is going on, we use multithreading to handle the network connection part. The client is same as what we have done for online battleship game. The client will have a separate thread to try connecting to the server. If connection failed, user will be sent back to the setting panels and they can check if they input the information correctly. If connection success, they will enter the room and wait for game to get started. For server, since we need to handle more than one client, we need to have one thread for each client in order to communicate with all clients without blocking. As implementation, we use a thread to run the main server socket, where we listen the port and accept all the connections from the clients. Each time we receive a connection, we will store the socket into our server and create a new thread to read the message from the client. Therefore, we can react with any message we receive from the clients without blocking any part of our programs. Server can chat in the room while waiting for other players to join.

One challenging part of our project is dealing with synchronizing data between different clients. To do that, we make use of protocol technique. Server will receive the data pack from the client and decode them through the protocol. After server determined that the data pack is valid, the command will be sent back to controller to do further calculation. As a result, most parts of the logic processing of game state are actually happening on server and the client will receive data pack each time server determined that they need to get a notification on things change.

* **Main Game Logic**

When the player first enters the room, they will be notified with a data pack from the server, which contains all information of user currently in the room. The player’s chats will be first sent to the server and then distribute back to every client in order for all clients to receive the information. When the game began, the server will send a notification data pack and all clients will update their game state to running. During the game, a timer will be shown to tell user how much time they have left for drawing or guessing the word. When the player sends a message, it will first be processed by server to see if they already guess the right word. If they do, the message won’t be broadcast to other clients since we don't want the user to spoiler the answers. If they don’t, the message will then be compared with secret word to see if they get the right word. If the words match, a score data pack will be sent to all clients to let them know that someone guess out the word and earn the score. If words do not match, the server will just broadcast the raw message from the client. If all guesser guesses out the answer or the time run out, the game will end and the score will be added to drawer depend on how good they make others guess out the word. However, if all players guess right, it means that the drawing is too obviously and there’s no challenging in guessing the words, which we will not give any points to the drawer. Therefore, the graph should be challenging enough while some people are still able to find out the meaning. Since our project is kind of party game, there’s no win and lose and no ending of game. We still provide a scoreboard for players if they’re interested in how much scores they earn.

application prog using java - csci 4963 - 01