Team 23: 12:30-1:45 section

Nabeel Haq - [nhaq2@uic.edu](mailto:nhaq2@uic.edu)

Juan Moraza - [jmoraz2@uic.edu](mailto:jmoraz2@uic.edu)

Hector Sanchez Rios - [hsanch21@uic.edu](mailto:hsanch21@uic.edu)

Project 5 Final Documentation

Section 1:

This program allows players to play a 4 player game. It is a dice roll game which is fairly simple. Each of the 4 players will click the button on their GUI that has a picture of dice on it. This will give them a random number between 1-6. After each player has rolled their dice, the player with the highest number is the winner. For this game, we will be using Java with JavaFX using sockets and threads. The client server relationship will be implemented similar to how we did it in project 3 and 4.

The server GUI will have a text field where you can enter a port number, along with a connect button to connect to the server. It will also have a text area to display winners and information about the game being played. For the client GUI, there will be a text field for the port number and the IP address, with a connect button that will connect the client to the server. It will have a button to roll the dice, along with a text field and text area to send and receive messages. Again we might add more later. The clients will each be on a separate thread, so that each player will be unique. The game will not be able to start until at least 4 clients are connected. A random number will be assigned to the client when the die roll button is clicked. When each of the players has clicked the button, the client GUI will automatically display the winner. It will also show the points that each player hasOne round is when any player gets 1 point (we might change this later). A new game will automatically be started after the round is over, and the clients die roll numbers will be reset. When a client exits the GUI, each remaining client and the server will be notified. Any message sent from a client will show up on every client GUI and the server GUI.

Section 2-3:

ArrayList<ClientThread>: this array list of client threads allows us to keep track of each client thread created when a new client enters the game. Whenever a client thread is created (a new player enters the game), it is added to this ArrayList. This allows us to send information to each client in a simple manner.

connthread: this object is an instantion of the ConnThread class which extends the Thread class. It creates the ServerSocket and the threads which are key parts of the program. Whenever a new player enters a game, a new thread is started dedicated to that player, making it much easier to separate each player from one another.

callback: this object is of type Consumer<Serializable>, so it allows data to be sent between the server and client, which is important when playing a game between the two. Each client/player will use this object to send and receive data. The data in our case will be the dice roll, and the messages sent at the end of the game (winner, win percentage, points, etc.)

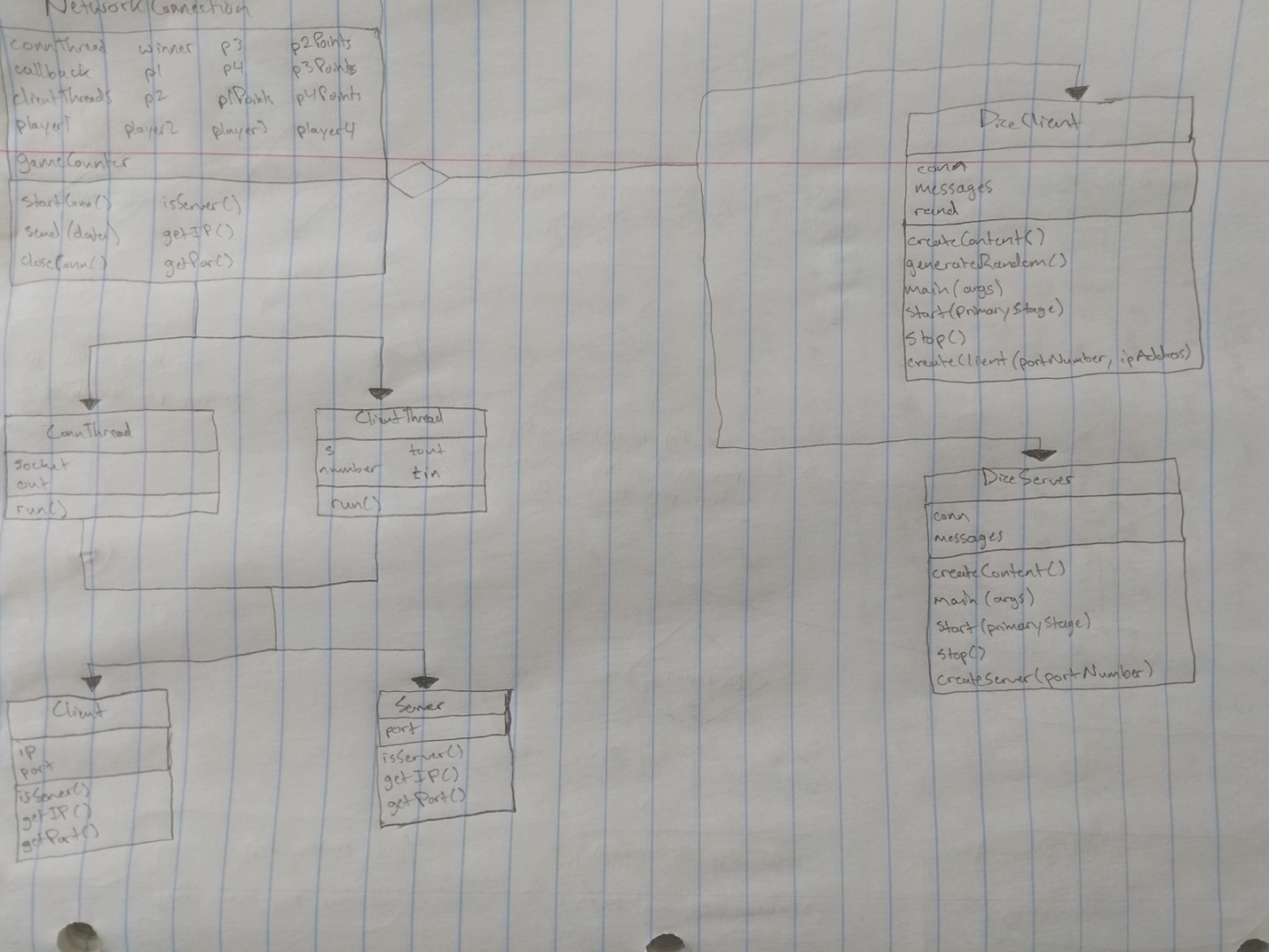
winner: string that states the winner of the game. To make it easy to keep track of the winner, we set this string to be “Player X”, where x is the winner of the game.

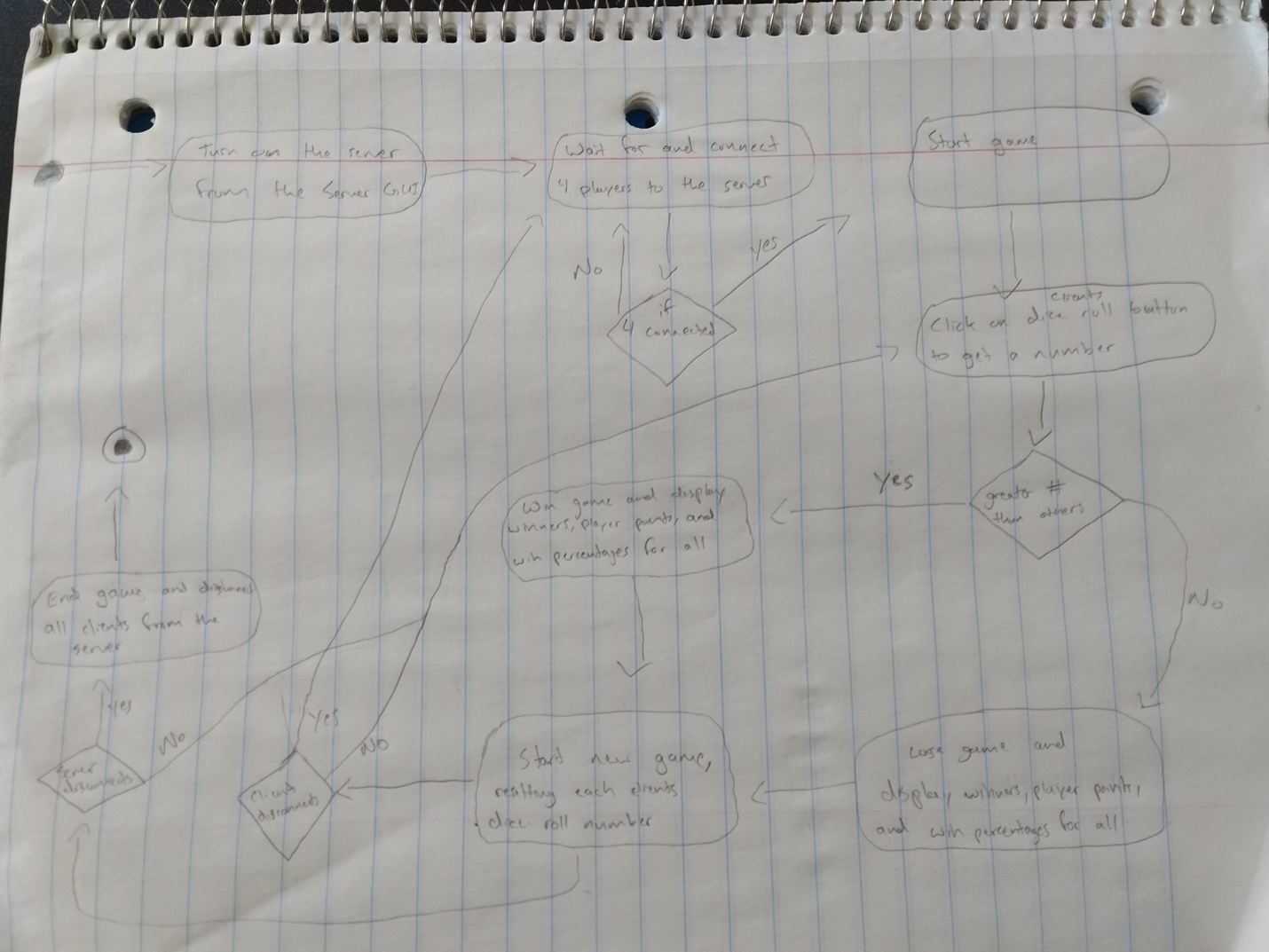
p1,p2,p3,p4: these int keep track of the number of the dice roll for each player. We use these to compare the number that each player rolled in their die roll. The player with the highes number is the winner. This makes its easy to keep track of the winner, as the random number generator in java also uses int.

p1Points,p2Points,p3Points,p4Points: these int keep track of each players points. This makes it easy to keep track of the winners points, as all we have to add is add 1 to it when the winner is determined. If there was an error in our program, the counter would display in an incorrect number.

player1,player2,player3,player4: these Boolean values set to true when the corresponding player connects to the server, so we can handle more than 4 players. At the start of the program they are automatically false. These make it easy for us to track a 5th or more player, because when a 5th player is connected he is told to wait for a slot to open. When one of these values is set to false after being set to true, it means a player has disconnected, opening up a slot for that 5th or more player.

gameCounter: this double keeps track of the number of games played, in order to calculate the win percentage of each player. This is a simple way to keep track of it because the program adds 1 to it when the current game is done. If there was an error in our program, the counter would display in an incorrect number.

UML:

Activity Diagram: 

Section 4:

Benefits

1. The design of the client GUI has the area to display messages, along with a dice roll button that has a picture of dice on it, which makes it look simple and elegant.
2. Server GUI has the port number, client GUI has port number and IP Address already inputted so no need to enter it unless you’re changing it.
3. Server GUI displays everything that the clients are getting in order to see maintain and see what each client is getting.
4. Allows game to start only when 4 players are connected.
5. Prevents 5 players from playing, making them wait for open spot.

Risks

The few risks we had have become non-risks now that we have completed the program. The first one being the counters of the game and player points displaying incorrect numbers based off an incorrect implementation of the client threads. The second one being that the server wouldn’t be able to receive the data from the 5th client. These have been resolved.

Assumptions

The only assumption we have has to do with the addition on more than 4 clients. When a client is connected to the server when 4 are already connected, the new clients will get a message telling them to wait for a slot to open up. Once a slot opens up, the assumption is that the waiting player takes the spot of the player that disconnected.