Distributed Systems SOFE 4790

Final Project Report:

Echo Hunt Game

Due: December 4, 2016

Aaron Peters 100483612

# Primary Design

The initial design was to have a singular server that clients would connect to through RMI. Clients would send their information to the server which will handle all the game actions and keep a list of all players in the game and will report the results back to the players. Server will act as the client and will notify the correct clients of specific events, such as the start of turns and the end of turns and games. One or more client must connect to the server. A game was proposed to be played with this.

# Final Design

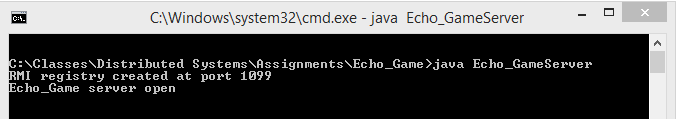
The final design still uses RMI to communicate to between clients and the server. Players will connect to the server and register their player name and receive a player id that the server will use to identify each player. Once registered players can enter commands to the server, such as /who or /games. These will be detailed later in the report. Players can create and join lobbies which will set them as specific clients to receive messages about each lobby. Players in lobby 1 will not see the same messages as players in lobby 2, etc. Players can leave lobbies once they are joined but cannot leave midgame. An assumption is made that once a game starts all players will remain in game until the end and no connection issues will occur. There is no way to reconnect to a game in progress if a disconnection occurs, and you cannot join a game in progress. Players can communicate to other players in lobbies before games and see who joins and leaves lobbies. The final form of the project is written in java, and consists of six java files; Client.java, ClientServant.java, Server.java, ServerServant.java, Echo\_Game.java and Echo\_GameServant.java. All files will be explained further in the report. The game portion was proposed in the primary design, however was left out due to time constraints. However, the game portion of the project is not crucial to the distributed system design for this project.

# How to Run the Program

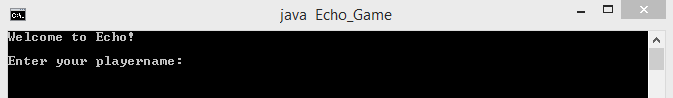
Assumption: All command windows are in the directory where all the .java files are located.

Compile the java files. If they are not compiled run the command on the command line: “javac Client.java Server.java ClientServant.java ServerServant.java Echo\_Game.java Echo\_GameServer.java”.

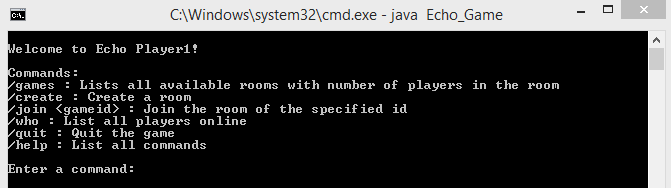
On the command line, run the command “java Echo\_GameServer”. The result should look like the picture below. Leave this window open, this is the server that clients will connect to.



In a separate command line window run the command “java Echo\_Game”. The result should look like the picture below. This is the client window which is attempting to connect to the server.



Enter a playername and press enter, if the playername is not taken yet the server should accept it and show a welcome screen and a list of commands that you can enter. All commands are entered with a / in front of them to differentiate them from chat messages.



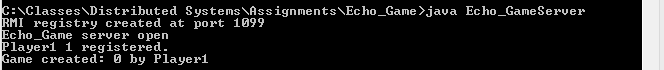
The player name should be registered on the server as well.

https://i.gyazo.com/426b6280231657770b89482dcdae6316.png

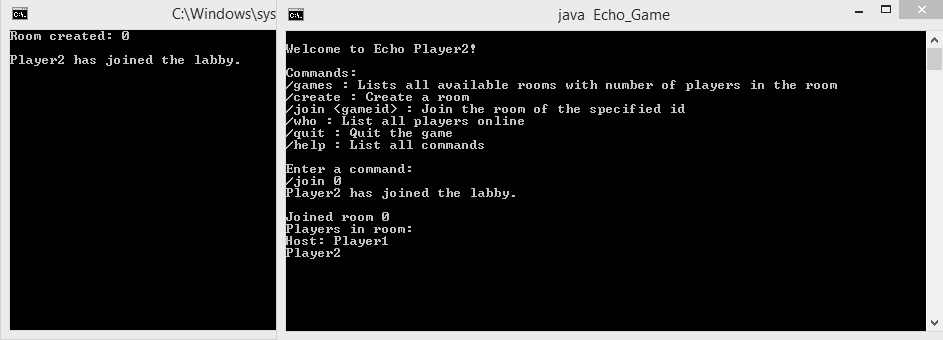
Type “/create”. This will create a room.

https://i.gyazo.com/b688b3b25634b40e45309e77bf60761c.png

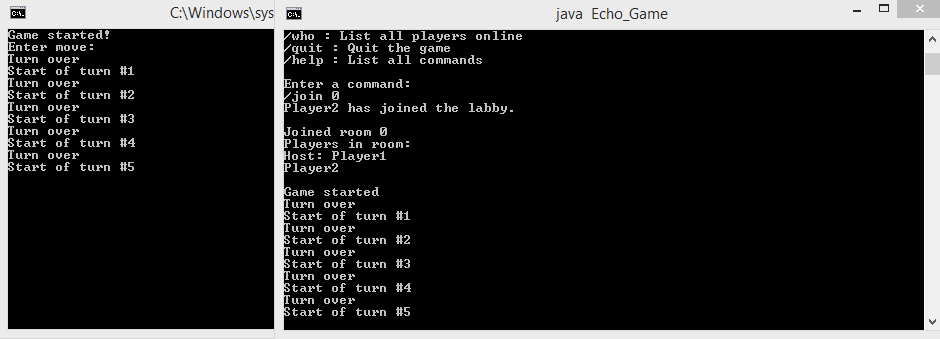
The server will show a record of the lobby being created.



Open a new command line window. Type “java Echo\_Game”. Enter a name. On the main menu type “/join 0”. The second player will join the room of the first player. The first player will be alerted of the second player joining the lobby. The first player and second player’s screens are shown below.



You can type anything without a / in front of it to send a chat message to the other players in the lobby. You can type “/start” in either of the command line. This will start the game. Every 5 second a turn will tick. The screen can look like this after a few turns.



That’s the core functions of the program. The other functions available in the main function are “/who” to see all the players connected to the server, “/games” to list all games, and the players inside each game, “/help” to see all the commands, and “/quit” to quit the program and removes the player from the playerlist and “/create” to create rooms. Inside the lobbies the available commands are “/start” to start the game, “/leave’ to leave the room and return to the main menu, “/quit” to leave the lobby, and quit the program and remove the player from the playerlist, “/who” to show who is all inside the lobby, and “/help” which shows the available commands.

# Presentation

The script I followed to show the program to the Professor/TA during the presentations can be found in Presentation.txt.

# Summary of code

## Client.java

Defines all the methods used in the Client interface

## ClientServant.java

### Important Variables

List<Integer> gamelist : list of games currently open, how many lobbies are open

HashMap<Integer,ArrayList<Pair>> playersingames : HashMap of which players are in each game lobbies. Key is gameid, values is an ArrayList of Pairs.

List<Pair> playerlist : list of players that are in the server. Players are saved as Pairs(name,id).

HashMap<Integer,ArrayList<Server>> gserver : HashMap of which clients are in each lobbies. Key is gameid, values is an ArrayList of Server objects.

Boolean[] games : Boolean array of games, if true, a game of that id exists.

Boolean[] gameinprogress : Boolean array of games in progress, if true, a game of that id is in progress.

### ClientServant()

Constructor for ClientServant

### sendMove

not implemented

### String joingame(int gameid, int playerid, Server gameserver)

Takes the gameid, playerid and the ServerServant object and joins a lobby. Returns a string for the client to print.

### int register(String playername)

Takes the playername and registers the player and assigns a playerid. Returns the playerid.

### int unregister(int playerid)

Takes the playerid and unregisters the player. Returns a success 0, or 1 for unsuccessful.

### int startgame(int gamenum, int playerid)

Takes the gamenumber and playerid. Returns a success 0, or 1 for unsuccessful.

### int leavegame(int gameid, int playerid, Server gameserver)

Takes the gameid, playerid and gameserver and removes the player from the game.

### int creategame(int playerid, Server gameserver)

Takes the playerid and gameserver and creates a lobby for players to join. Returns the gameid of the game created.

### String listplayers()

Lists all players connected to the server. Returns a string to print out to the client console.

### String listplayerslobby(int gameid)

Takes the gameid and returns a list of all players connected to the lobby. Returns a string to print out to the client console.

### Void lobbychat(int gamenum, int playerid, Server myserver, String message)

Takes the gamenum, playerid, gameserver and message you want to send, and sends it to all other clients connected to the lobby.

### HashMap<Integer,List<String>> listgames()

Lists all games that are open and all of their players inside the games. Returns a HashMap of List<String> with all the players insides games.

### Void debug(int playerid)

Method to test things as I was creating it.

### Int getPlayerIndex(int pid)

Takes the playerid, and returns the index from the playerlist where that playerid is stored.

### Class Pair()

Custom class that holds the name and id of a player.

## Server.java

Defines all the methods used in the Server interface

## ServerServant.java

### Important Variables:

Boolean[] games : Boolean array of games, if true the game exists

Int[] turntimers : integer array of turntimers, holds the value for the next endturn

Int[] turnn : integer array of turnn, holds the value of number of turns that has gone by

Timer timer : timer that ticks every second to see if a turn has ended

### ServerServant(Client gameclient)

Constructor for ServerServant with the Client obj passed in.

### Void gamestart(int turnnumber, int gameid, Boolean[] gameinprogress)

Takes the turnnumber, gameid and gameinprogress array. Starts the game timer.

### Void turnstart(int turnnumber)

Not implemented

### Void endturn(int turnnumber)

Not implemented

### Void playerjoins(String playername)

Takes the playername and prints out to all other people in the lobby.

### Void lobbychat(String playername, String message)

Takes the playername and the message, and sends the message to all other people in the lobby.

### Class turntimer

Runs every second and checks if any turn have ended, if any have send them a turn end message.

## Echo\_Game.java

### Void main(String args[])

Starts the program and connects to the registry. Registers to the server.

### Void mainscreen(int playerid, ServerServant gameserver, Client gameclient)

Runs the mainscreen part of the program. Can enter commands specified earlier in this document.

### Void inLobby(int gameid, int playerid, ServerServant gameserver, Client gameclient)

Runs the lobby part of the program when the player is in the lobby. Can enter commands specified earlier in this document.

### Void inGame(int gamid, int playerid, ServerServant gameserver, Client gameclient)

Runs the ingame part of the program when the game starts.

## Echo\_GameServer.java

### Void main(String args[])

Starts the program and creates the registry for the client to connect to. Connects all the clients together.