HalliGalli Game

Team03

(Team Leader)20180203 배재영

20186491 송치현

20183901 김상민

20180023 김승후

20181269 박현우

20180353 박성배

1. Description

We decided to program HalliGalli Game by using java socket programming also, using Thread. HalliGalli is originally a boardgame you can play with cards and bell. The rule of the game is like this: Every player gets equal number of cards and each open their card one by one. In the card, there are 1-5 number of same kind of fruit, Banana, Strawberry, Lime, Grapefruit. So, if the number of a single fruit in opened cards is 5, the first person who rings the bell wins a single match. If you win a match, all of the opened cards are carried into winner’s deck. The only person who has deck, who has 56 cards wins the game. Our HalliGalli game is played by 4 players and since we cannot put a bell, we replaced it by entering random word. We made a Server that controls 4 Clients.

1. System requirement

1.requirement for Server program

To run Project HalliGalli Game, we need both Server program and Client Program. If the state that 4 clients have accessed is checked, the game is executed.

Since, both Server program and Client program are written in java language, it is executed above JVM. Therefore, JVM environment is required to execute both programs and shared internet which is applied to both programs is also required

1. requirement for client program

Client program mainly focused on how graphic interface for game user is rid. To make graphic environment, we used Swing component provided by java. Additionally, we used several library classes provided by java like Thread and to compile these programs, eclipse environment is needed. Client program is written in one source code, ClientEx.java, and we need images to be used in program.

1. Functionality

In this project, Client program and Server program exists separately, and built project to make it exchange information through Socket communication.

1. server program

In class MainServer, every time client accepts it sends information about its number. Also, every ID of accessed client to every client. When number of client becomes 4, it creates User interface and begins the game. At the beginning of the game, server gives same number of cards to Users. 5 seconds after that, timer is created and the game begins.

The circumstance after beginning is divided into 3 cases: A B C D

1. 0<=time<=1, beginning of turn
2. 1<=timer<=10, in the middle of turn
3. 10<timer, end of turn
4. When CheckSum returns true

When A->B, Server lets all of the clients to know which turn it is. When turn is on B, when client sends protocol “P# SAY BUTTONPRESSED”, PlayerDraw is called, and circumstance changes to A. Also, if client enters wrong word from B, playerPenalty functions is called to that certain client and gives penalty. When B->C, playerDraw is called and set Timer=0 and circumstance goes to A. Then, it checks ueser’s deck and open\_deck and calls function updateLife which changes user.state. After that, “P#\_BUTTONOFF” is sent to client which turn points and make it repaint screen. Also, when C->A, it receives random Word and send message to every client that current answer is Word. After CheckSum is executed, if it becomes D, it doesn’t go into A and wait for clients to enter answer Word. If answer is entered, it becomes A.

At this moment, static functions of Mainserver are following ones.

Public static void initialize(User[] user)

Receive user which has 4 User instance which contains 4 clients game process information as parameter and return Deck instance which has 14 cards which are created randomly without overlap.

Public static Deck merge\_open\_deck(User[] p)

Receive user which has 4 User instances which contains 4 clients game process information as parameter and return Deck instance which gathered all cards from each user’s open\_deck.

Public static boolean checksum(User[] p)

Receive user which has 4 User instances which contains 4 clients game process information as parameter and return true if opened cards have same fruit number of 5 and else return false.

Public static int getAliveNum(User[] p)

Receive user which has 4 User instances which contains 4 clients game process information as parameter and check each user’s state and return the number of users which has state of true.

Public static void updateLife(User[] p, GameRoom room)

Receive user which has 4 User instances which contains 4 clients game process information and GameRoom’s instance room which has socket information of all 4 clients as parameters and make user’s state to false if both user.deck and user.open\_deck is 0 and send message to every client that that certain user is dead.

Public static int nextTurn(User[] p, int turn)

Give turn to next client.

Public static void playerDraw(User[] p, GameRoom room, int target)

User which target points draws one card from Deck and puts it into opdn\_Deck. After that, it calls broadcast\_currentNum.

Public static void playerPenalty(User[] p, GameRoom room, int target, String msg)

User which target points draws one card from Deck and puts it into every other player’s Deck which means targeted User draws 3 cards in total. Then, it sends “P”+target+”\_\_SAY\_\_”+msg to every clients and call broadcast\_currentNums.

Public static void broadcast\_currentNums(User[] p,GameRoom room)

Notify every clients the number of cards in every user’s Deck.

Client program consists of three classes.

First, public class ClientEx. Once program is executed, main method in ClientEx class creates Client object and also creates one Thread object.

Second, class Client. This class extends JFrame and implements Interface Runnable. This class is responsible for most of the program’s functions. This class includes Socket object which is responsible for socket communication, I/OStream objects, and class GUI objects as member variable.

Once program is executed, main method in ClientEx calls constructor from Client class. Client class makes Frame for users to see as they are created. If user enters Server’s IP address and game ID, socket communication initializes. EnterGame in Client class is responsible for this. As socket communication begins, Thread is also initialized. Thread will make class which implements Runnable be ready for run(); Through Thread, Client and Server can continuously send and receive information. As server sends Card information which user has, Client class shows screen according to commands from Server.

Lastly, class GUI. Class GUI extends JLayeredPaned. This draws several GUI elements in screen.

1. Implementation issue

When scheduling Thread, the problem was synchronizing main Thread and ServerUser Threads’ execution. For example, in main function, if user[0].id is accessed right after user[0].start(), proper string is not allocated to user[0].id and sometimes null comes out. At this time, until ServerUser finishes Start, mainThread is synchronized by sleep. Also, when in loop, when accessing inst.size() when storing commands from ServerUsers in instruction object, there are somes cases that 0 is referenced. We also solved this problem by synchronizing mainThread by sleep when in while loop.

The main implementation issue in implementing Client program was socket communication and the use of Thread. The main reason why we decided to use Java is because of socket communication. It is programmed to initialize socket object by using getText() when game user enters IP address in JTextBar. It is created by receiving I/O stream from initialized socket object.

When communicating with server, one important factor is to utilize Runnable interface. Since, server continuously sends information to client, client should always be ready to receive information. To do this, we executed Client which implements Runnable interface by Thread and was able to create it by receiving information from server.

After receiving information from server, client checked which information is inserted by switch case statement and made to take action that follows.

1. UML
2. Execution result
3. Applied object-oriented concepts

* ServerUser extended Thread to allow multithreading. We didn’t had to create multithreading by reusing code resulting high productivity.
* After creating user, deck, card objects in server program, we added each object’s information to each object’s field. For example, card’s fruitname and fruitnumber are stored in field and made Deck to manage each card objects with stack. Declared each information in private and methods as public resulting using data encapsulation.
* By using Runnable interface, we overrided run() method. By using functions which interface provides, we redefined run() to do functions we want.

1. Conclusion.