Advanced Object Oriented Programming
Asteroids Report

Jeewon Heo s3766861
Hwajun Lee s3543609

October 29, 2019

1 Problem description

Our goal is to create a 2D asteroids game. The game should be available for multiplayer game as well as single player game. More specifically, the task requires to provide stable multiplayer platform based on UDP method. Following are the basic information about the required features:

1. Singleplayer game
2. Hosting multiplayer game
3. Joining in multiplayer game
4. Sepctating multiplayer game
5. Showing scores of singleplayer game scores

Game Rule The collision of different kind of Game Object destroys each other, but collision of the same kind do not. Both type of games are automatically restarted when game is over. The scoring system in singleplayer game is by counting the number of asteroids shot with bullet and the score is reset ever restart of the game. For multiplayer game, we decided to give a point to the last standing player of each game. In multiplayer game, players can kill each other's ship using their weapon.

2 Problem analysis

Upon starting the game, a main menu is created with options to player singleplayer game, host multiplayer game, join multiplayer game, spectate multiplayer game, and see top ten scores of the singleplayer game. When selecting **Start Singleplayer Game**, the user is asked to put in his or her nickname, and then the game starts. when selecting the **Host Multiplayer Game**, the user is asked to put in his or her nickname as well. Then the server lobby frame is created which monitors connected players. It also displays its IP address and port number. Then, the user can click on **Start Game** button to initiate the multiplayer game. When selecting **Join Multiplayer Game**, the user is asked to put in his or her nickname and the server IP address. Then, it displays a default screen and waits for the server to start game. Selecting **Spectate Game**, the user is asked to put in server IP address. Once again it displays an default screen and waits for server to initiate the game. In any case, the menu frame is maintained so that the user can go back to it select again from the options. Users can quit program by closing main menu frame.

3 Program design

3.1 GamePacket

For efficient networking, we have created a class GamePacket which contains the wanted data. There are 6 types of GamePacket: REQUEST_CONNECTION_JOINER(0), REQUEST_CONNECTION_SPECTATOR(1),

DISCONNECT(2), REJECT_CONNECTION(3), ACCEPT_CONNECTION(4), SHIP(5), and GAME_MODEL(6). The first 4 are type of MessagePacket. SHIP is a PlayerModelPacket type and GAME_MODEL is a SeverModelPacket type. When creating a new DatagramPacket, we make the first byte to denote its PacketType. This design makes it easir for the receipient to process later on. The static methods in Network class provides a simple sending methods, that upon calling, automatically creates a GamePacket and sends it to destination address.

3.2 Networking and Sharing Game Objects

The **entity** package in **model** has all the participants and their supporting classes. Server, Client, Joiner, Spectator are the participating entities of UDP networking. Address is for easier access to the IP address and port address. InputHandlers and OutputHandlers for Joiner and Spectator manages the input and output of the data. The threads start running when a new Joiner or a Spectator is created. In order to flag when start or quit a certain loop, we created a boolean fields such as connected and shipMoved. connected signifies that client has successfully connected to the server. and allows the InputThread to constantly listen to data sent from server and OutputHandler to move on to the next step. As for the Spectator, the next step is to wait user or program prompts it to disconnect from server. As for the Joiner, the next step would be sending its data to the server. For this matter, we chose to send the Spaceship controlled by the Joiner. In order to decide when to send a new Spaceship to server, we made a boolean field shipMoved. shipMoved is set to true when Joiner's Observer, PlayerKeyListener notifies that the joiner pressed or released a game key. Thus, shipMoved signifies the update of the model, which commands JoinerOutPutHandler to send the Spaceship model to server.

Inside Server class, we have put methods regarding input from the clients, such as receiveConnectionRequest, receiveDataFromClients, processGameData, and etc. Upon receiving connection request from a client, it performs method processConnection which proceeds according to the type of the connection message. When it is a connection request from Joiner or Spectator, the Server adds address to the clientsAddresses list and sends back ACCEPT_CONNECTION message. For the former, the Server proceeds to create a new Spaceship for its game model and sends back the data (color of the Spaceship) back to the Joiner along with ACCEPT_CONNECTION message.

Server's output is dealt mostly in MultiplyaerGameUpdater thread. When the server user clicks on the button "Start Game," it prompts the start method in game, and creates and run the MultiplayerGameUpdater thread, which updates the game model. Whenever it is time to display an updated game model on screen, the updater thread sends the updated game model to the clients so that the client side display is renewed as well as that of the server side.

In this manner, Server deals with input within its own Runnable and runs a game updater thread that deals with sending game model to clients.

3.3 Databse

Via MySQL, we created a database that contains all singleplayer game records. Inside Database class in **database** package, there are multiple static methods that allows us to modify the database. The connect method connects Java with MySQL. A new database is automatically created when it cannot find one. When every time a singleplayer game restarts, a new row of data is appended to the table all_scores with insert method. Also, getTopTenScorers method retrieves ten rows with highest scores and saves it a String format. This is used in RankingFrame to display high score list.

4 Evaluation

The program satisfies all the given requirements. It performs well in multiplayer game mode, which is the essential part of this program. However, if we had more time, we could have tried several things to improve our program, such as making database work for multiplayer game and allowing user to choose the color of their spaceships. Also, we could make the networking between server and clients more robust and provide game modes like 2 vs 2. We both contributed equally to making this project. In terms of

teamwork, we both contributed equally to making this project. Before starting, we discussed together how to approach and implement the program. We worked together to implement interesting UI. The basic structure of networking was suggested by Jeewon and Hwajun contributed to refine it. The database was created together. After the basic structure was implemented, we both worked together to refactoring and generally improving it.