Asteroids Report

Robert Patrona (S3322211) and Yona Moreda (S3324338)

Team 1

October 29^{th} 2018

1 Problem description and analysis

This asteroids assignment mainly deals with topics such as networking and concurrency. In addition, design patterns and re-factoring are explored. In order to fulfill the requirements of the assignment, several classes implementing thread functionality are created.

In order to re-factor, the code methods are extracted, descriptive comments are added, variables are renamed and encapsulated. Some methods and fields are added to different classes to satisfy the requirements of the given assignment.

2 Design description

The project is split into three main packages corresponding to the Model-View-Controller architectural pattern. The model contains all the major classes that are mainly involved with data and their corresponding methods. The View package describes all the classes necessary for displaying the model and the manipulations done by the user on the View is applied onto the Model through the classes described under Controller.

Two design patterns are used, namely, the builder pattern for building instances of the class `Client` and the factory pattern for constructing successors for large and medium asteroids.

The builder pattern for creating clients allows the creation of client object using a dedicated client builder class. The builder pattern allows us to reduce the number of parameters that are not necessary when constructing a client. The creation of clients is more readable and clear because of the use of builder pattern. The builder patter allows us to set the parameters of client explicitly and step by step.

We also implemented the Factory Pattern. We created a new class AsteroidFactory, which makes use the Factory Pattern in order to produce Asteriod objects. We decided to use this pattern because we wanted to delegate the creation of new Asteriods from the Asteriod classes to a dedicated Factory class, thus improving the quality of the code. After an Asterod has collided, in our AsteroidFactory class, if we pass it a Large Asteriod, it will return two Medium Asteriods, and similarly, passing a Medium Asteriod will return two Small ASteroids.

The networking is implemented in different ways for the different game modes and is described as follows:

Spectator mode:
In spectator mode, the host will instantiate its Server which is a thread that takes requests from Spectators for the host game model. The Server sends the host's Game model and the Spectator sets the properties of its game with the received game. The Spectator sets its spaceship, bullets, asteroids and array of clients from the received model. The Spectator then sends another request and receives and game model after updating itself.

NOTE: In order to spectate a multi-player game, the host or the joiner should select host spectate in the menu and then any client can spectate after.

Multi-player mode:
In Multi-Player mode, the host starts its JoinerServer thread that receives connection/disconnection requests from clients and adds/removes a new Client based on the received request. A Client class is defined to encapsulate and bind properties such as space ship, bullets, address and port. Whenever a client connects, its name and address is received and a client is constructed and added to the array list of clients in game model. Then, on the client side (the joiner), a key listener that is linked to the client frame sends its key presses (ClientMoves) to the JoinerServer.

The JoinerServer then identifies the address of the packet from client and updates the client with that corresponding address inside the array list of clients in game model. The JoinerServer then sends the model of the host to the right address. The Joiners update their array list of clients, asteroids, their ship and bullets based on the received model. In a short summary, the Joiner sends controls to JoinerServer and the JoinerServer sends its model to the clients using the UDP protocol.

3 Evaluation

The implementation of the spectator and multiplayer modes is relatively stable. The performance of the game drops for the joiner and a small bit for joinerServer

in multiplayer mode.

The game normally performs at around 25 frames per second and with the multiplayer mode the performance is lower and has a bit more stutter. There are some bugs that may occur when the host suddenly closes the program on its side.

The stuttering can be fixed with an implementation that involves the joinerServer being an observer to the host's game model and whenever the game model changes, the joinerServer is notified of that change and sends the updated game model to its clients. We worked on this implementation and it works as expected for 2 players with no stutter but when it comes to 3 or more players the joinerServer fails to identify the client with its address and the program fails to continue. We reverted back to our current implementation as a result and it has more stutter but works as it should for any set amount of players.

4 Team work

Robert came up with the implementation of clients and how the client side implementation behaves. Robert contributed the classes Spectator and Joiner and how the frame is drawn. Yona contributed with the implementation of host side and the classes Server and JoinerServer. The controllers ClientMoves and Execute are made by Yona as well. The idea behind the whole exchange that happens between client and host is jointly made by the combined efforts of Robert and Yona.