Architecture

Overview

The software is divided in two main parts: server and client. Both parts can be accessed through the Main.java class and both parts access shared packets, such as the packet, helper and data structures packets.

Server

The server holds a list of connected clients (instance of ClientThread.java) and created lobbies (instance of Lobby.java).

The server instantiates a ping thread, which is responsible for sending a ping packet in a fixed time interval to each client and disconnecting clients, which do not respond.

The server is responsible to establish a connection to all the users that want to connect to the server.

For each client that connects to the server a new thread with the class ClientThread is started.

The ClientThread class holds information about the respective client such as the nickname, if and to which lobby the client has joined, the score and the robots. The ClientThread reads input from the input stream of the socket connection. It starts to read as soon as the “Beginning” char is read and stops until the “Ending” char is read. It reads each byte and interprets it as a character.

The lobby class holds a list of the players and the game map (instance of GameMap.java), which in it self is a datatype. The game map is made of individual cells (instance of Cell.java) oriented in a two dimensional array, and each cell stores their own position in said two dimensional array and all the gameobjects (instance of GameObject.java)that are on the cell. The gameobjects are from one of five possible datatypes. Those are “Robot” (instance of Robot.java), “Ore” (instance of Ore.java), “Radar” (instance of Radar.java), “Trap” (instance of Trap.java), “Nothing” (instance of Nothing.java). All of these are self explanatory except the datatype “Nothing”, which is used to indicate that nothing is on a given cell and to ensure that information has been transferred correctly.

Each time a user is in a lobby and ends their turn the server will validate and update all requested changes that are made by the user. Once this is finished it will update the map and then inform the users about the new map. The gamemap can only be modified on the server, and the client does not make any own decisions in regards of the displayed map.

The server handles incoming packet data and forwards it to the relevant connect client (instance of ClientThread.java) which will then proceed to decode the packet. The ClientThread themselves also store a reference of which lobby they are connected to, but the server mainly stores this information and keeps track (and updates) the lobbies. The server only updates the reference in the ClientThread and the ClientThread itself does not update it. Once the ClientThread has decoded the message it will not ask the Server for further validation as this Thread is secured anyway. The ClientThread is also responsible to write all the packet data to the outputstream of the server. The server is also responsible of creating and maintaining the Highscore list and file (file: HighScore.txt)

Client

The client is built upon the Model-View-Controller architecture design pattern.

There are two views: the start menu (startmenu.fxml), allowing to chat, change nickname, create and joining lobbies; and the lobby(lobby.fxml), allowing to play the game.

Both views are written in fxml and are loaded by their respective controllers (StartMenuController.java and LobbyController.java). The controllers link the objects created for the view with the objects of the models (Client.java and LobbyInclient.java).

The Client.java class is the main model holding all information about the server, as well as the functions creating and sending packets to the server. The Client.java class starts two threads: the first one (instance of PongThread) sends pong packets to the server and handle the response or lack thereof; and the second one (InputStreamThread) reading the input stream of the server socket, recognizing the default packet ending character and validating and decoding the packet.

The second model (LobbyInClient) is responsible for holding the information regarding the game, such as the game map, the turn counter, the players, and their scores.

The client architecture deviates of the strict Model-View-Controller pattern as the controller (LobbyController) is linked to both models, derogating of the one model per controller rule. This architecture was chosen since both views share some elements and the client-server communication was built in the client.java class. However this would not be repeated in future projects.