CMPT 370 D1

MAINETENCE MANUAL

Dec 3, 2016

Contents

1 INTRODUCTION

This maintenance manual has been created to help keep the Robot Game system manageable. It will contain a high-level overview of the as-built architecture of the system, a description of external libraries used, and details about complex pieces of the system.

1.1 TEAM MEMBERS

Resource Name	Role
Dimaano, Nico	Developer / Tester
Neijmeijer, Niklaas	Developer/ Tester
Seidenthal, Kyle	Developer / Tester
Sterma, Brendon	Developer / Tester
Zang, Jiawei	Developer / Tester

2 AS-BUILT ARCHITECTURE

The Robot Game uses the Model-View-Controller Architecture. The classes for the program are broken up into six packages: controller, json, model, tableRow, test, and view. These will be described in more detail below.

2.1 CONTROLLER

The controller package consists of the Game class, the Client class, the Server class, and the LobbyMessage class. The game class acts as the main controller for the game and contains the main method to run the game. The Client, Server, and LobbyMessage classes make up the interfaces for multiplayer networked gameplay.

2.1.1 GAME

The game class is the controller and the centre of the system. It ties all the classes from all packages together by allowing them to communicate with each other. The main method is located here, as well as all functions that control the game. All view classes call methods in the controller to change the state of the game. Anytime a user interacts with a view, the view sends the appropriate information to the controller to update the classes in the model which then the takes in the information and changes what is displayed in the view. The controller then talks to classes in the model to update the game.

2.1.2 MULTIPLAYER NETWORKING

The multiplayer networking classes are also contained in the controller package. They communicate with the controller to determine when and what they should send across the network. There are two main classes used for networking. The Server class contains the functions required to start a server for the game. To play with other users, a server must be running which choosing the Host Game Option in the Main Menu can do. All other users connect to this server using join game and entering their name and the hosts IP address. When a user connects to a server, a client is created to handle communication with the server.

The server works by receiving messages from connected clients and then relaying those messages to every client connected. These messages are of type LobbyMessage when the players are in the lobby, and are of type board when the players are in an active game. The board contains all variables required to reconstruct the game state on any machine. This allows the controller to send the current state of the game to other users by sending the board to the server for redistribution. When players receive the board, it is sent to their game view which updates their screen as necessary for display.

2.2 JSON

The json package contains all classes and methods to handle reading and writing json files. These files contain stats about robot AI programs and the AI code for specific robots. At this point in time, the json package does not connect to the controller, and is not actively used in the system.

2.3 MODEL

The model contains all entities used in the game. This includes the classes: Board, User, Robot, Tile, and all subclasses of these. These classes are used to keep track of the state of the game, such as robot positions and user scores and turns.

2.4 TABLEROW

This package contains the classes for storing rows and tables. These are used in the views to display tables of user and robot stats.

2.5 TEST

The test package contains all Junit test modules. It includes tests for the Board, Player, Tile and User classes. These Tests are there to check the basic functions of the classes and to see if what is being displayed. These can be run using Junit test.

2.6 VIEW

The view package contains all classes that display the graphical user interface for the system. The game calls these classes when they are to be shown, and updated when the state of the game is changed in the model.

3 AN OVERVIEW OF COMPLEX PARTS

The most complex part of the system is the controller. It handles all communication between the view and model, as well as communication with other users across the network.

The server class is quite complicated. It contains a list of all connections, which are each their own class. Each connection has a thread that sends to its client, and a thread that receives from its client. There are many loops and checks to make sure that the connection is still active before sending a message. The server sends all messages to all clients. When a client sends a message to the server, the message gets put on a queue. The server then takes messages off the queue one by one and sends them to every connected client.

4 EXTERNAL LIBRARIES

4.1 GSON

We used Google's GSON library for handling JSON operations. It was very good at handling JSON files and was easy to understand.

4.2 JUNIT TEST

We used the JUNIT TEST library to create and run all our module tests. This allows us to create and run module tests separately from the main program in a simple and intuitive way.

5 COMPILATION AND RUN INSTRUCTIONS

Open and Run the project in the java editor of your choice.

Version History:

- **12/04/2016 – Initial Deployment Version – DEPLOY-V1.0**