VU Software Engineering 1

Submitted document

Subtask 3
(Implementation of the Client)

Last Name, Name	Hasho Alisa
Matriculation number	01368897
E-Mail Address	a01368897@univie.ac.at
Date	16.01.2018

General Requirements

1. Git Repository

Git Bundle is included in the ZIP file. The client was mostly developed parallel to the server. Latest version of the client can be found under *src\ mainclient\ Client_run.java*

2. Implemented functionality for the client

Described as follows is the workflow as conceptualized, and how it is implemented by the called methods in the code:

- Main – Instantiates the class
- Client_Run – It connects to the server and creates the main **while** loop, which handles where the connection goes and the flow of control. Inside the while loop the code never leaves. Instead it bounces through a number of other methods such as :
- wait_command_play : waiting for the server to give commands after the client has connected to play
- request_to_connect: sends request to connect
- response_to_connect: waits for the connection
- ping_request : checking running time / connection
- ping_receive : checking running time / connection
- send_map_response : sends a row of the created map to the server, each time it is called
- random_char : chooses a random character in the array list , needed in generate map
- generate_map: creates the map on a 16 character string, where the amount is previously set (6 water, 6 meadow and 4 mountains)
- split_map : splits the map in multiples of 4 for easier transmission
- wait_command_idle: it waits for play command and does error handling / starts game in case server is ready and connected with 2 clients
- send_packet: sends packets to the IP address into that port via UDP
- check_play_status: checks if the client is ready to play
- parse_request_map: allows view of the received variables from the server
- command_parse : extracts the first word of an incoming message, which is its type
- create_packet_string: creates string from variables given from each request

3. Logging

In order to make logging possible for this project i have used the Java default framework (java.util.logging package), that provides all objects, methods and configuration necessary to transmit log messages. This logging API allows me to configure which message types are written. Individual classes can use this logger to write messages to the configured log files in order to to report and persist error and warning messages as well as info messages (e.g., runtime statistics) so that the messages can later be retrieved and analyzed.

Loggers are the objects that trigger log events and in this case they are created and called in the source code.

Create : Logger.getLogger(). getLogger() with a string parameter for the name of the logger. I have determined three log levels, based on their severity:

- Severe = error
- Warning
- Info = success

I have created the static method log, which takes as parameters the integer values 0, 1 and 2 for each level (0-error / 1- success / 2- warning). I call this method from different parts in the code with classes that I thought as relevant to log.

Examples:

a. Logs server sending error message :

```
Log.log(0, "Received error from server: " + variables.get("message"));
```

b. Logs socket timeout:

```
Log.log(0, "Sent Request and didn't get any response while waiting to play");
```

c. Logs created packet string:

```
Log.log(1, "Sending: " + packet_string);
```

d. Logs client not ready to play:

```
Log.log(0, "Server didn't send correct UUID, intruder in the system");
```

e. Logs successful ping received:

```
Log.log(1, "Received ping-back");
```

More implemented examples can be found in the Client_Run file.

At first i wanted to log the client-server connection status from the client side, but this log would be somehow redundant cause all the important information regarding client server connection is already logged on the server side and double checked by the send and receive ping methods, therefore i avoided this log.

Below is a short documentation of a group of logs, also stored in a pdf file inside the ZIP folder (named LogfileClient.pdf) as well as in the GIT Repository there is an extended version.

```
Jan 16, 2018 8:31:24 PM mainclient.Log log
INFO: Sending: connect:name=Test_Client:
Jan 16, 2018 8:31:25 PM mainclient.Log log
INFO: Sending: ping:UUID=20d7833d-bcc7-4520-8987-6e0f290d5999:status=1:
Jan 16, 2018 8:31:25 PM mainclient.Log log
INFO: Sent ping
Jan 16, 2018 8:31:25 PM mainclient.Log log
INFO: Received ping-back
Jan 16, 2018 8:31:25 PM mainclient.Log log
INFO: Sending: ping:UUID=20d7833d-bcc7-4520-8987-6e0f290d5999:status=2:
Jan 16, 2018 8:31:25 PM mainclient.Log log
```

INFO: Sent ping
Jan 16, 2018 8:31:25 PM mainclient.Log log
INFO: Received ping-back
Jan 16, 2018 8:31:35 PM mainclient.Log log
SEVERE: Sent Request and didn't get any response while waiting
Jan 16, 2018 8:31:35 PM mainclient.Log log
INFO: Sending: ping:UUID=20d7833d-bcc7-4520-8987-6e0f290d5999:status=2:
Jan 16, 2018 8:31:35 PM mainclient.Log log
INFO: Sent ping
Jan 16, 2018 8:31:35 PM mainclient.Log log
INFO: Received ping-back
Jan 16, 2018 8:31:45 PM mainclient.Log log
SEVERE: Sent Request and didn't get any response while waiting
Jan 16, 2018 8:31:45 PM mainclient.Log log
INFO: Sending: ping:UUID=20d7833d-bcc7-4520-8987-6e0f290d5999:status=2:
Jan 16, 2018 8:31:45 PM mainclient.Log log
INFO: Sent ping
Jan 16, 2018 8:31:45 PM mainclient.Log log
INFO: Received ping-back
Jan 16, 2018 8:31:55 PM mainclient.Log log
SEVERE: Sent Request and didn't get any response while waiting
Jan 16, 2018 8:31:55 PM mainclient.Log log
INFO: Sending: ping:UUID=20d7833d-bcc7-4520-8987-6e0f290d5999:status=2:
Jan 16, 2018 8:31:55 PM mainclient.Log log
INFO: Sent ping
Jan 16, 2018 8:31:55 PM mainclient.Log log
INFO: Received ping-back
Jan 16, 2018 8:32:05 PM mainclient.Log log
SEVERE: Sent Request and didn't get any response while waiting
Jan 16, 2018 8:32:05 PM mainclient.Log log
INFO: Sending: ping:UUID=20d7833d-bcc7-4520-8987-6e0f290d5999:status=2:
Jan 16, 2018 8:32:05 PM mainclient.Log log
INFO: Sent ping
Jan 16, 2018 8:32:05 PM mainclient.Log log
INFO: Received ping-back
Jan 16, 2018 8:32:14 PM mainclient.Log log
INFO: Sending: playakk:UUID=20d7833d-bcc7-4520-8987-6e0f290d5999:status=3:
Jan 16, 2018 8:32:14 PM mainclient.Log log
INFO: Sending: mapakk:UUID=20d7833d-bcc7-4520-8987-
6e0f290d5999:row=0:data=wgww:
Jan 16, 2018 8:32:14 PM mainclient.Log log
INFO: Sending: mapakk:UUID=20d7833d-bcc7-4520-8987-
6e0f290d5999:row=1:data=wwmm:
Jan 16, 2018 8:32:14 PM mainclient.Log log
INFO: Sending: mapakk:UUID=20d7833d-bcc7-4520-8987-
6e0f290d5999:row=2:data=mwmg:
Jan 16, 2018 8:32:14 PM mainclient.Log log
INFO: Sending: mapakk:UUID=20d7833d-bcc7-4520-8987-
6e0f290d5999:row=3:data=wggg:
Jan 16, 2018 8:32:24 PM mainclient.Log log
SEVERE: Sent Request and didn't get any response while waiting to play

4. Error Handling

All Java errors are handled as exceptions. Therefore all the possible exceptions that might occur during the running phase of the game are handled with the try and catch blocks.

Every time an exception is catched, a severe log message is added to the log file in order to report the error with relevant information. Game, human, internal and network errors are all handled with the same technique.

In the example below you can see the two exceptions being handled are:

a. the Socket Timeout Exception, since the client runs into error after 10 seconds pass without receiving any response from the server
b. the IOException which is generated in case the log file is not found or other issues with the output file.

```
private void wait_command_play() {
  byte[] receiveData = new byte[512];
  DatagramPacket receive_packet = new DatagramPacket(receiveData, receiveData.length);
  try {
    client_socket.setSoTimeout(10000); // 10 seconds
    client_socket.receive(receive_packet);
    String command_type = command_parse(receive_packet);
    Map variables = parse_request_map(receive_packet);
    switch (command_type) {
    case "error":
      Log.log(0, "Received error from server: " + variables.get("message"));
      break;
    case "mapreq":
      if (!gen_map) {
        split_map();
        gen_map = true;
      }
      send_map_response(variables);
      break;
    default:
      Log.log(0, "Sent Request was malformed");
    }
  } catch (SocketTimeoutException e1) {
    // nothing happened
    Log.log(0, "Sent Request and didn't get any response while waiting to play");
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

5. Unit Tests

The tests implemented in the source code validate if that code results in the expected state (state testing) or executes the expected sequence of events (behavior testing). The following explains what is being tested by the implemented Junit tests:

```
@Test
        public void testRandom_char() {
                Client_Run client = new Client_Run();
                ArrayList<Character> tests = null;
                tests.add('a');
                tests.add('b');
                tests.add('c');
                String result = client.random_char(tests) + "";
                assertTrue(Character.isLetter(result.charAt(0)));
        }
```

Instatiating the client run will run an infinite loop of the client, so the tests were implemented and tested before the client was 100% ready to run. This first method tests if the output is a character.

```
@Test
        public void testGenerate_map() {
                Client_Run client = new Client_Run();
                String test = client.generate_map();
                assertEquals(test.length(), 16);
        }
```

Tests if the generated map is 16 characters long as desired.

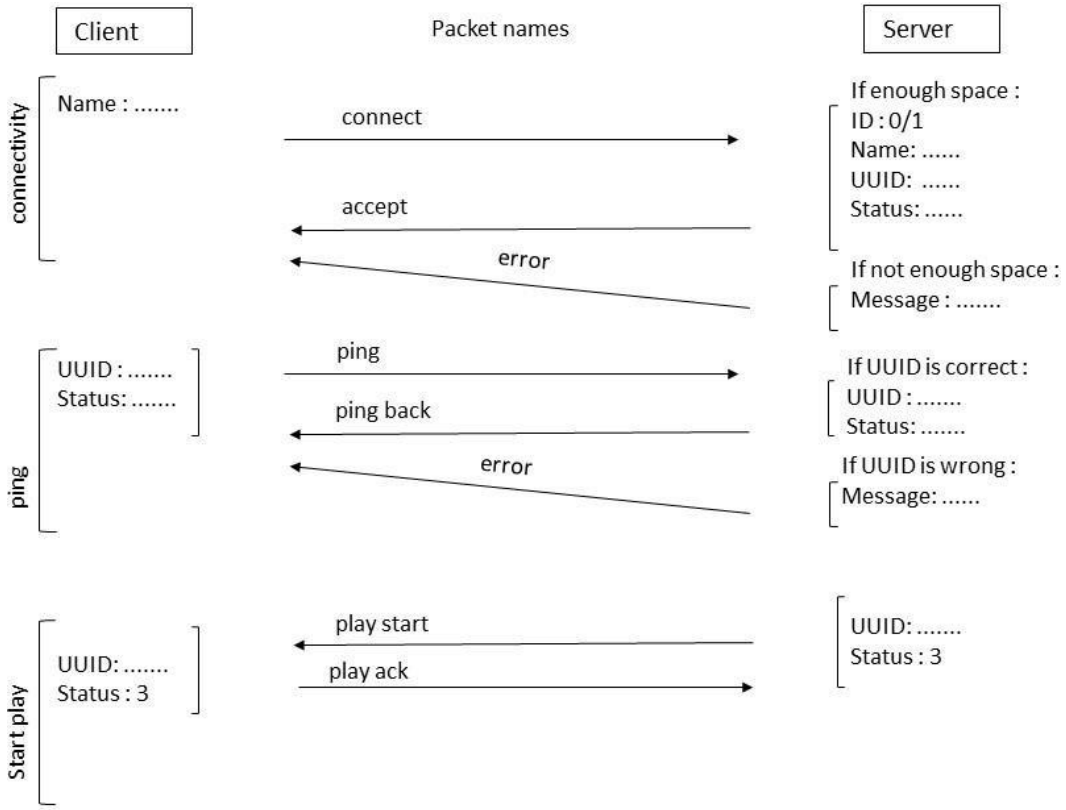
```
@Test
        public void testSplit_map() {
                Client_Run client = new Client_Run();
                String[] test = client.split_map();
                assertEquals(test.length, 4);
        }
```

Tests that the map is correctly split into 4 pieces for easier transmission.

Network communication

I saved the same exchange message scheme as designed for Subtask 2 and implemented the client able to connect with the server and use it as an intermediary for the exchange of messages during the game. The connection is tested by both the client and the server and with the help of ping messages it is possible to save the status of the game.

It is a UDP connection therefore pings are also needed to test that the clients are still connected to the server. The ping goes back and forth every 10 seconds and if the answer does not occur within the first 15 seconds of the ping sent, it means the client and the server are disconnected. Right after the Universally Unique IDentifier of the client changes and the IP is not registered anymore on the server.



Graphical User Interface & Map Generation

The map generation consists on a partially intelligent, partially random algorithm since i was unable to complete the AI on time. The initial intention was to create an AI that would follow one of these two strategies: aggressive or passive.
An aggressive strategy would be one with very little obstruction of movement, mountains and water pieces placed mostly on the sides to allow the AI to get to the opposite treasure faster. A passive strategy on the other hand would include more obstacles in the middle of the board, putting more focus on slowing down the opponent.

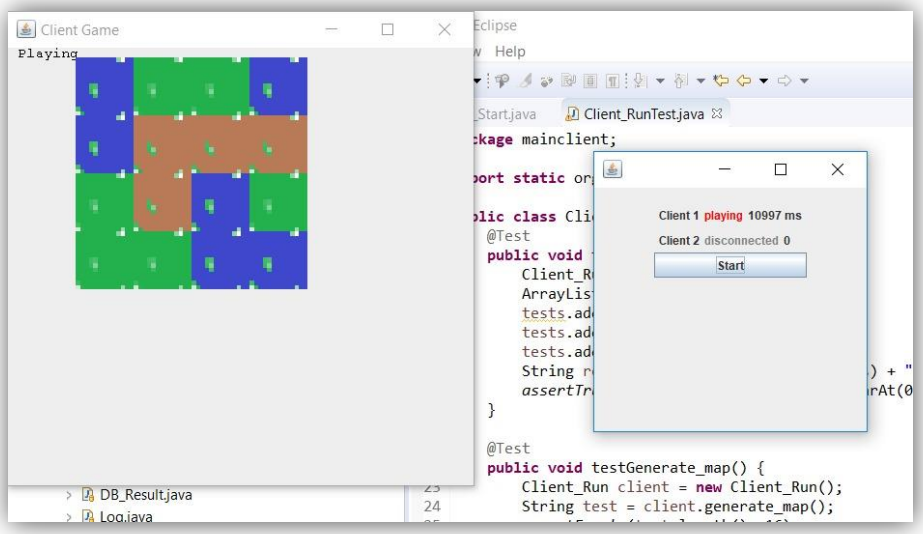
The GUI is implemented with JApplet, a java swing public class that allows an interactive view of the generated map every time the client is running in connection with the server. Based on the variables in Client_Run under the name of Client_map, it transmits / prints them to screen, according to the chosen icons. It is a very basic form of interface but it is functional to the algorithm.

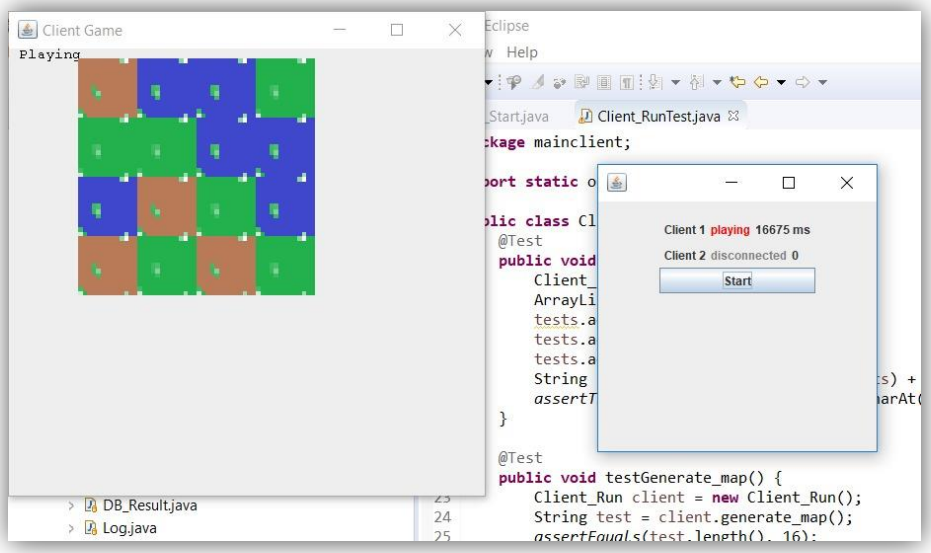
Below you will find screenshots of generated maps next to the current game status.

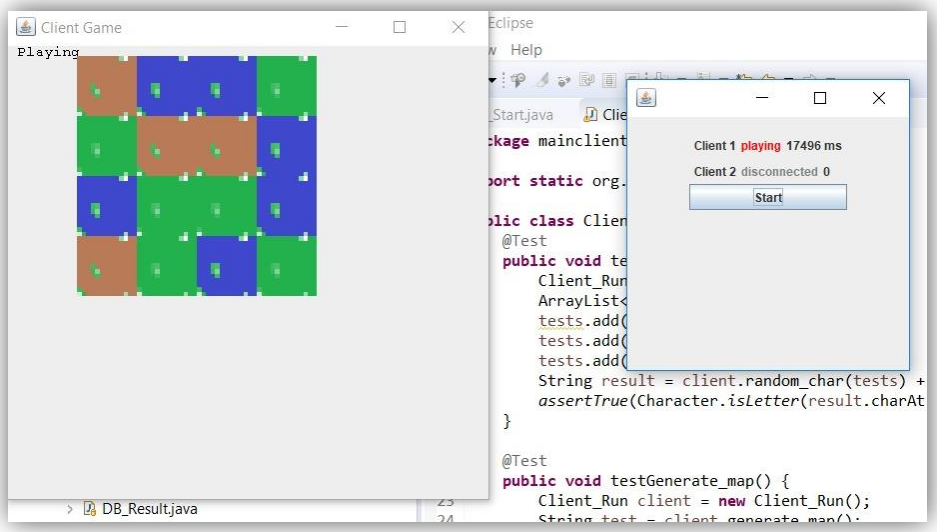
*Legend: Meadow

Water

Mountain


Client Game
Playing
Client 1 playing 10997 ms
Client 2 disconnected 0
Start
Client_RunTest.java
DB_Result.java
Log.java


Client Game
Playing
Client 1 playing 16675 ms
Client 2 disconnected 0
Start
Client_RunTest.java
DB_Result.java
Log.java


Client Game
Playing
Client 1 playing 17496 ms
Client 2 disconnected 0
Start
DB_Result.java