Assignment 5 - Distributed Android App

Network Programming, ID1212

Bernardo Gonzalez Riede, begr@kth.se

December 17, 2017

1 Introduction

Goals:

- Design, develop and deploy an Android app using the Android SDK
- Usage of the Android SDK and IDE tools

2 Literature Study

As in previous assignments, the main source were the video lectures on the topic and the example code.

3 Method

The server side code has been reused from Assignment 1 which uses the MVC approach, having a startup, net (view), controller and model layer. The client side, an android app, has been developed with Android studio using the Android SDK 26.

4 Result

Link to public Github repository with code: https://github.com/MemBernd/ID1212-Android

4.1 Layering

The app is dived in *main* and *net* (work) layer. The network layer includes the AsyncTasks which handle the connection initialization, sending and receiving. 4 classes are inside the main layer; two activities, a global constants holding class and an *Application* extending class.

4.2 Client & Server

4.2.1 Server

Being a slightly modified version of the first assignment, the server is a Java implementation of a hangman game. It creates a new handler for every client connected, identified by the assigned port number, and is therefore multithreaded. The previously mentioned modification is an inclusion of a header in the server reply to indicate the receiving client if it's a message indicating a (new) state of the game or if its information regarding a prohibited move, e.g. an attempt to solve it with an incorrect amount of characters.

4.2.2 Client

On the client side, the android app, the implementation involves mostly GUI operations and network communication.

The *GlobalState* class extends the *Application* class, allowing other classes in the process to access this class as a context. It stores a reference to the *Socket*, *PrintWriter* and *BufferedReader*. An application context is an easy implementation of a temporary form of storage to circumvent the need of passing arguments in the creation of an *Intent* and dealing with non-serializable objects like sockets.

When trying to connect to the server (MainActivity l. 19) net.EstablishConnection uses the input from the user to try to connect to the server (l. 39 - 44). After the attempt it calls *onPostExecute* (l. 55) which either creates an intent to switch to the game activity or displays a toast with a failure indication, based on the previous connection attempt.

Net.Sender takes a String as input an sends it using the PrintWriter from the application context (l. 26).

Net.Listener is called after every sending to process the expected reply from the server. While *doInBackground* (l. 31 - 43) only receives the message, *onPostExecute* (l. 46 - 65) has the necessary logic the act based on the reply from the server. It shows error or informative messages in the form of toasts or updates the gamestate textview with the information received.

Note: The original idea was to have only one listener start when the game activity is created and let it update whenever something had been received by the server. Trying to do so resulted in the sending task to be executed after the listener finished because of not having received something and therefore timing out. It could be because the emulator uses only a max of 2 threads; one gets used by the UI thread and the other by the AsyncTasks which may get executed sequentially in this case.

4.3 GUI

Figure 1 shows the main activity when launched. The user is only taken to the gamestate server if he's able to successfully connect to the server.

Figure 2 is an example of a game after a game has been lost already. In a second iteration it would be easily possible to extract the game score and attempts left counter

Figure 1: Initial interface

and create a textview or similar to hold said information separately. Additionally, a afore mentioned toast can be seen at the bottom.

5 Discussion

Some unexpected behaviours were encountered sometimes, e.g. network operations in the UI thread, which are due to the nature of the android framework. Still, it wasn't obvious at first and delayed some of the work.

6 Comments About the Course

It took around 7 hours (netto) for the project.

- 4 hours 20 min for coding.
- 1 hour 15 min for the report.
- 1 hour 23 min watching videos, taking notes and thinking about how to tackle the problem.

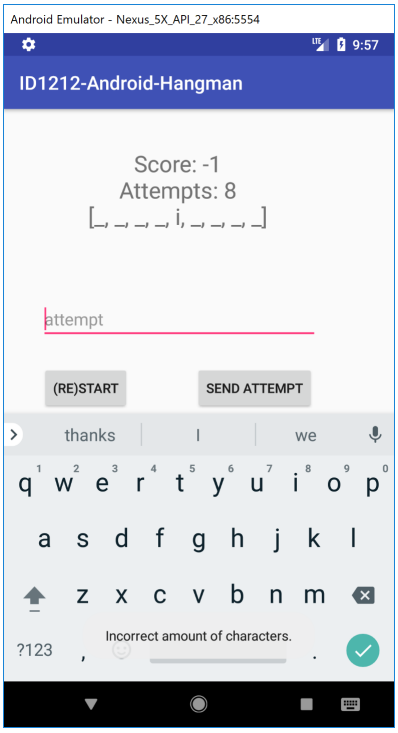

Figure 2: Game interface