Distributed Android Application: Chatroom Simplejack

By Kristian Wang, Justin Curewitz, Pascale Queralt, Christopher Chen

https://github.com/justincurewitz/concurrentProject

1.0 ABSTRACT

Our team presents a distributed Android chatroom application that allows users to not only send messages back and forth, but also to play a simplified version of Blackjack, called Simplejack, within the chatroom. Implementation of the chatroom itself requires a chatroom server that spawns a thread for each user who joins. In addition, client(s) and the server communicate using sockets. Implementation of the game within the chatroom, Simplejack, demonstrates many distributed computing concepts, including tokens and using a master process to synchronize global state. In this paper, we present our implementation of this chatroom and Simplejack.

2.0 INTRODUCTION

As mobile devices continue to grow in popularity, distributed algorithms become more important due to the inherent distributed nature of mobile computing. The purpose of this paper is to present our team's implementation of chatroom Simplejack, including how we created a client-server chatroom using sockets and how we synchronized data that is split across many android devices to allow for multi-user gameplay. In this paper, we include the main functionality of our Android application and a tutorial on developing Android applications so that readers may have a starting point from which to follow our implementation. We provide an in-depth discussion of the system components of our solution, including details on algorithms employed, implementation, and design alternatives. Finally, we present the results and conclusion of our work.

3.0 PROJECT DESCRIPTION

A user may open the application titled “EE360P Chatroom” and enter their name at the login screen to join the chatroom. A user can even join from a web application. Once users have joined the chatroom, they are free to chat at will or begin a game of Simplejack. A user initiates a game with “/blackjack” and the server replies to all users present in the chatroom “Starting Blackjack! Type /join to enter.” Users are free to join or not. Those who join receive 2 standard poker cards from the dealer. After, each player gets a turn to either hit any number of times or stay (/hit or do nothing to stay). The purpose of the game is to beat the dealer and get as close to 21 as possible without going over. If any of the players go above 21, they “bust” or lose the game. The game is over when each player has had a turn, and whoever is closest to 21 wins the game. To simplify the game, we don’t use cards but rather just integers to represent card values from (1-13) inclusive. Intuitively, each integer represents the value that gets added to the sum for each player.

4.0 HOW TO DEVELOP AN ANDROID APP

The official android tutorial on creating your first app is a great start on developing Android applications and can be found at https://developer.android.com/training/basics/firstapp/creating-project.html. We include the first page of this tutorial in Appendix A and invite the reader to follow the provided instructions and create their first app.

Before starting the tutorial, it is important to download all the relevant components to minimize the amount of time spent remedying gradle error messages. Make sure to download: Android Studio (https://developer.android.com/studio/install.html) and the relevant SDKs from the SDK manager (after downloading Android Studio). In addition, our group recommends using a physical Android device for testing rather than the simulator which is generally very slow without special software such as HAXM.

5.0 IMPLEMENTATION

In order to implement chatroom Simplejack, we relied on external resources (tutorials) as well as our own proprietary implementation of distributed algorithms. Our team lacked Android development experience so we chose to follow a tutorial provided on Android Hive to assist us in building the base chatroom using sockets. However, we wrote Simplejack on our own by implementing a token-based algorithm and a scheduler. We chose to implement our chatroom application this way, over using third-party APIs such as PubNub or Firebase which may be more optimal than sockets, to ensure that the concurrent and distributed components of our solution were implemented by us.

5.1 External Resources

We decided to follow the tutorial found on Android Hive (http://www.androidhive.info/2014/10/android-building-group-chat-app-using-sockets-part-1/) to build our base chatroom because of our unfamiliarity with Android websockets and because the chatroom itself is not essential to the distributed concepts that we wished to demonstrate through implementation of

Simplejack. Despite this, we learned a lot by following this tutorial and creating a chatroom that consists of a socket server and the Android app client.

The socket server is responsible for handling socket client connections and passing the chat messages between clients. The socket server is a Tomcat Server, which is ideal because Tomcat implements WebSocket and provides a server environment in which Java code can run. The socket server is responsible for opening a websocket connection with any client that connects to it. If there are more than one users accessing the chatroom, the server will spawn a thread for each client to allow for concurrent connections.

The Android application allows a user to connect to the socket server and send and receive messages. The app uses the Android websockets library (https://github.com/koush/android-websockets) to implement websocket communication with the server by connecting to the server's URL (<IP address>:<port>/chatServer). Messages are formatted as JSON strings and exchanged between the server and client and are parsed on both ends to initiate the appropriate action.

5.2 Proprietary Implementation

The purpose of our game Simplejack is to demonstrate distributed concepts with a multi-player game where data across multiple devices must be synchronized. The implementation of the game uses a token-based algorithm to determine whose turn it is during each round. The socket server is the coordinator in this algorithm and is responsible for tracking which process owns the token. The client that has the token is allowed to play their turn. A scheduler is used to manage turn length and game registration. Upon starting a game, players have 15 seconds to join. Once the game starts, each player has 15 seconds to play when it is their turn, then the token is passed to the next player. This is accomplished by a thread that is scheduled every 15 seconds, which passes the token. Player actions are handled via callback functions. These functions are blocked if the player's client does not own the token. When every player has taken their turn, the final scheduled thread runs and determines a winner.

5.3 Design Alternatives

There were many ways we could have designed both the chatroom and our application. For example, instead of using a token based system for mutual exclusion during turns, we could have used a quorum or any number of other mutual exclusion algorithms. Additionally, the chat room itself could be distributed. Instead of using a centralized server to ensure mutual exclusion of access to the message buffer, the devices themselves could maintain global state of messages through the use of other distributed methods such as Lamport's mutual exclusion algorithm.

Had we been building a product which we intended to bring to market, we would have made use of Pubnub or Firebase, two services which offer complete end-to-end APIs for building distributed mobile applications. These services abstract the work of synchronizing messages from the user by allowing an app to both listen and publish to data streams, allowing developers to focus on building out features of an application rather than spending time implementing server side synchronization.

6.0 PERFORMANCE RESULTS

As the purpose of our project was to demonstrate distributed concepts by creating an android app, we cannot gather many quantitative results. However, we tested the functionality of our application's chatroom and Simplejack game with 3 players on a mixture of Android and web app clients. We hope that one of the most important results of our project is aiding future EE 330P students who attempt to make distributed android applications for their final projects. One of the biggest challenges for our team was simply familiarizing ourselves with Android development and how sockets are implemented on the Android platform. With the tutorial in Appendix A and discussion of our implementation in Section 5.0, we hope that this project becomes a viable starting point for future students and that they may take our results from chatroom Simpleajack and expand upon them for increased functionality and performance.

7.0 CONCLUSION

In this report, we have discussed the functionality and implementation of our distributed Android application Chatroom Simplejack. We applied many concepts learned in EE360P while implementing this project, including spawning threads for each client connected to the socket server and applying a distributed computing token-based algorithm to Simplejack. We expanded our knowledge by becoming familiar with an Android implementation of websockets and distributed Android applications in general. This expanded knowledge of distributed computing and Android development will serve us in the future when developing other more complex multi-user distributed applications.

Appendix A

1. In Android Studio, create a new project:
    - If you don't have a project opened, in the **Welcome to Android Studio** window, click **Start a new Android Studio project**.
    - If you have a project opened, select **File > New Project**.
2. In the **New Project** screen, enter the following values:
    - **Application Name**: "My First App"
    - **Company Domain**: "example.com"
3. You might want to change the project location, but leave the other options as they are.
4. Click **Next**.
5. In the **Target Android Devices** screen, keep the default values and click **Next**.
6. If you're curious about how these SDK versions affect your app, read Supporting Different Platform Versions.
7. In the **Add an Activity to Mobile** screen, select **Empty Activity** and click **Next**.
8. In the **Customize the Activity** screen, keep the default values and click **Finish**.

After some processing, Android Studio opens the IDE. Now take a moment to review the most important files.

First, be sure the **Project** window is open (select **View > Tool Windows > Project**) and the **Android** view is selected from the drop-down list at the top of that window. You can then see the following files:

app > java > com.example.myfirstapp > MainActivity.java

This is the main activity (the entry point for your app). When you build and run the app, the system launches an instance of this Activity and loads its layout.

app > res > layout > activity_main.xml

This XML file defines the layout for the activity's UI. It contains a TextView element with the text "Hello world!".

app > manifests > AndroidManifest.xml

The manifest file describes the fundamental characteristics of the app and defines each of its components.

Gradle Scripts > build.gradle

You'll see two files with this name: one for the project and one for the "app" module. Each module has its own build.gradle file, but this project currently has just one module. You'll mostly work with the module's build.gradle file to configure how the Gradle tools compile and build your app. For more information about this file, see Configure Your Build.

References

"Create an Android Project." Android Developers. N.p., n.d. Web. 15 Apr. 2017.
Tamada/, Ravi. "Android Building Group Chat App using Sockets - Part 1." Androidhive. N.p., 30 Oct. 2014. Web. 18 Apr. 2017.

"Install Android Studio." Android Developers. N.p., n.d. Web. 15 Apr. 2017.

"Configure Your Build." Android Developers. N.p., n.d. Web. 15 Apr. 2017.

"App Manifest." Android Developers. N.p., n.d. Web. 15 Apr. 2017.

"TextView." Android Developers. N.p., n.d. Web. 16 Apr. 2017.

"Activity." Android Developers. N.p., n.d. Web. 16 Apr. 2017.

"Supporting Different Platform Versions." Android Developers. N.p., n.d. Web. 17 Apr. 2017.

"Overview of Realtime Data Streams on Android." PubNub. N.p., 15 July 2016. Web. Apr. 2017.