Hemuppgift 5:
Distributed Applications Using Android

Nätverksprogrammering, ID1212

Evan Saboo
saboo@kth.se

2017-12-17

https://github.com/ZpeedX/Network-Programming/tree/master/Homework_5

1 Introduktion

Målet med femte hemuppgiften är att kunna använda Android SDK och IDE utvecklings verktyg för att utveckla och köra en Android applikation.

Applikationen man ska utveckla en klient-serverapplikation för spelet Hangman. Serven körs i en dator och hanterar all data och logik medan klienten körs i en Android emulator och hanterar användargränssnittet. Serven och klienten kommunicerar också med varandra under spelets gång.

2 Litteraturstudie

Uppgiften utfördes med hjälp av dessa källor:

* Android föreläsningarna hjälpte mig förstå hur man programmerar i Android studio och hur man implementerar multitrådning i en Android applikation.

* Jag använde StackOverflow hemsidan för att få hjälp med problem som uppstod och för att få snabb information på några java syntaxer.[1]

* Android dokumentationshemsidan var också hjälpsam för att hitta syntaxförklaringar.[2]

[1] https://stackoverflow.com
[2] https://developer.android.com/reference/classes.html

3 Metod

I denna hemuppgift behövde jag mest gå igenom Android SDK och Utvecklingsverktyget Android studio.
Eftersom Android SDK använde Java så var det ganska lätt att förstå hur man utvecklar Android funktionaliteter i en applikation, men det vara inte samma för Android UI.
Den jobbiga delen med designen var implementationen av Android ”ConstraintLayout”, för att applikationens UI ska kunna anpassas till flera enheter.

Både serven och klienten (Android appen) körs i android studio.
Jag behöver inte netbeans för att köra serven eftersom android studio har java implementerad och kan köra java applikationer utan att behöva använda någon Android emulator eller enhet.

4 Resultat

När Android applikationen startas visas först startsidan (figur 1). Efter att man har tryckt på knappen ”start new game” kopplas klienten till serven genom sockets. Om klienten inte lyckas koppla till serven visas en felmeddelande till användaren i samma sida, annars går applikationen vidare till nästa sida (figur 2).
I en spelomgång kan användaren antingen gissa en bokstav genom att trycka på en bokstav i skärmen eller gissa hela ordet genom att skriva ordet i textfältet med hjälp av androids tangentbord och sedan trycka på ”guess word”. Gissningen skickas och hanteras av serven som sedan skickar tillbaka ett svar till klienten.
Om användaren lyckas gissa ordet skickas hen vidare till ”Game Over” sidan (figur 3), där kan hen spela en ny runda eller gå till startsidan.
Om användaren går tillbaka till startsidan stängs kopplingen mellan serven och klienten.

Serven
Hela serverdelen har tagits från första hemuppgiften och därför följer MVC arkitekturen. Den är också multitrådad, där en tråd lyssnar på en specifik port för att ta emot nya klienter och en tråd för varje klient som är kopplat till serven.

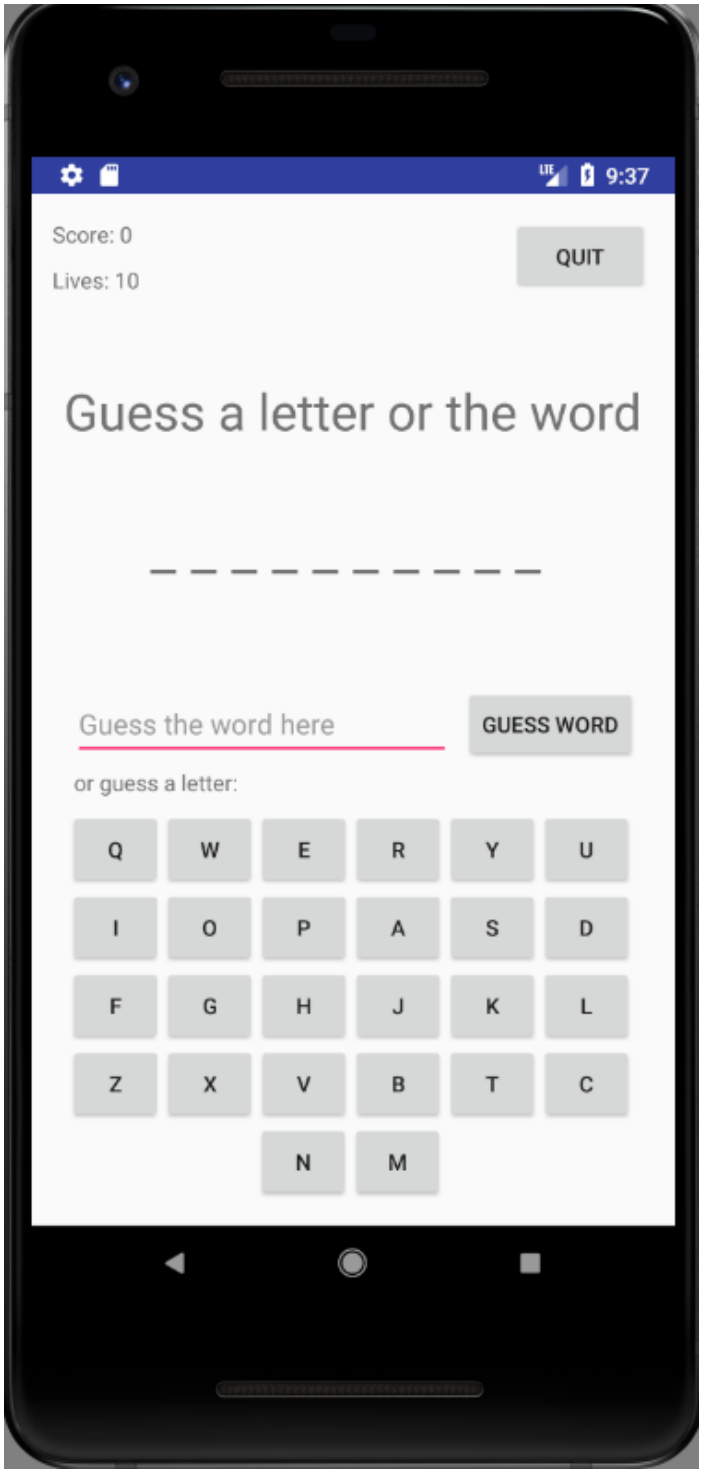
Klienten (Android applikationen)
Klienten följer också MVC arkitekturen men istället för modell paketet har den ett Net paket. Anledningen till detta är för att klientsidan är statslös, dvs. all logik och kalkylering hanteras av serven och klienten hanterar bara användargränssnittet. Net paketet används för att kommunicera med serven, View paketet tar hand om användargränssnittet och Controller hanterar alla anrop mellan View och Net.

Varje sida i applikationen sin egen en subklass som hanterar sidans funktioner. Varje subklass arvs av superklassen ”Activity”. Superklassen låter utvecklaren hantera UI i fönstret som klassen skapar.
För att hantera UI kan man skriva layouten i en XML fil eller designa den genom Android studios layoutredigerare. Där kan man snabbt skapa layouter genom att dra widgets till en visuell designredigerare istället för att skriva layout XML för hand.
Jag använde mig mest av redigeraren men jag behövde ibland lägga till attribut manuellt i XML filen.

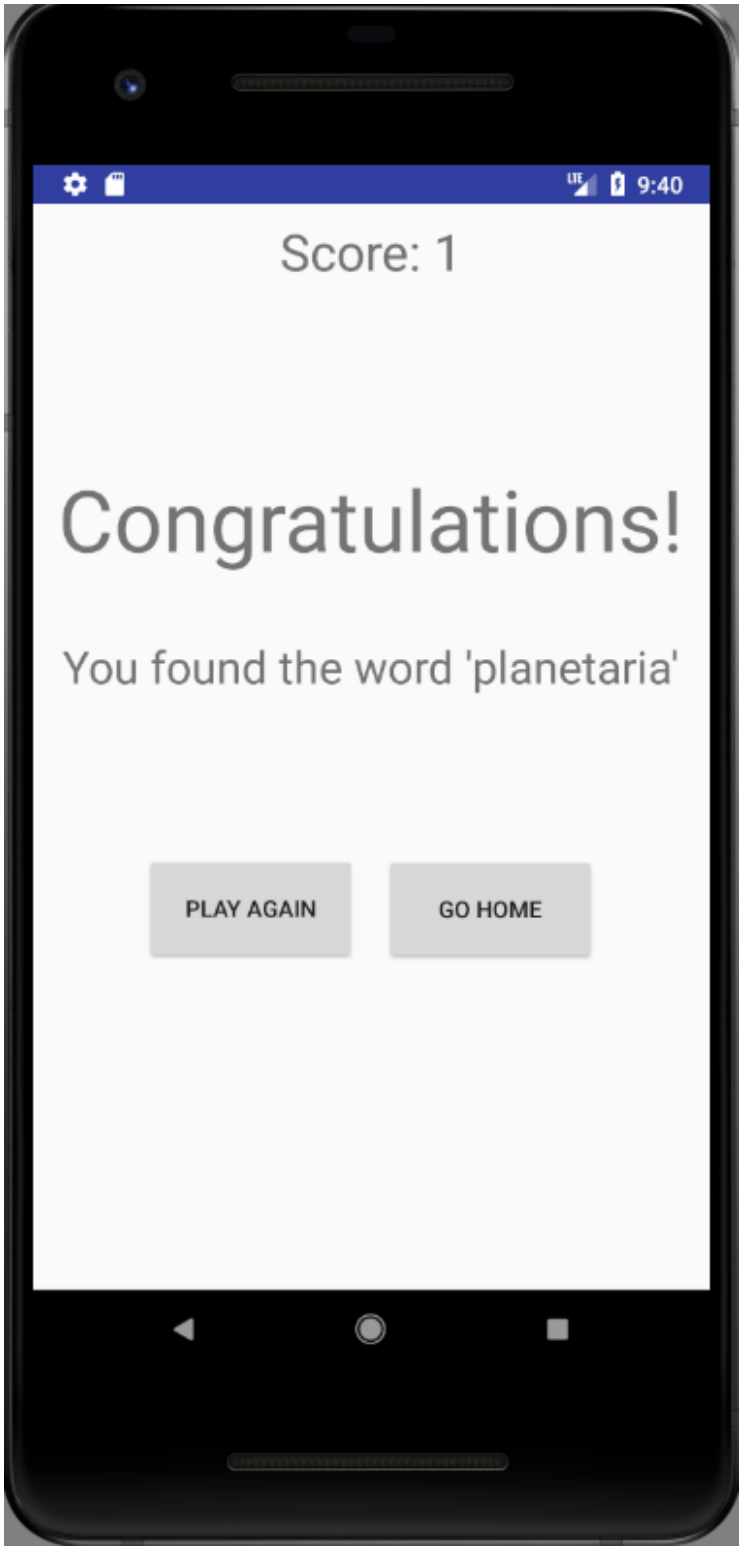
För att implementera multitrådning används abstrakta klassen AsyncTask för att låta en annan tråd (worker) än UI tråden hantera all kommunikation med serven. Tråden gör bakgrunds operationer medan UI tråden kan hantera användaren så att användargränssnittet blir responsiv.

För att klienten ska kunna koppla och kommunicera med serven används Java socket. Implementation gjordes genom att skapa en klass som har metoder för att skicka och ta emot data från serven. Klassen har statiska metoder och variabler för att andra klasser ska kunna använda samma socket för kommunikation, då det inte gick att skicka över socket objekt till en annan aktivitet (nästa sida).

Figur 1: Applikationens startsida



Figur 2: UI under spelet gång.



Figur 3: UI för när en spelrunda har avslutat.

5 Diskussion

Kraven nedan uppfylldes:

- Serversidan följer MVC arkitekturen för bättre struktur över logiken och användargränssnittet.
- Jag använde TCP-sockets för att serven och klienten ska kunna kommunicera med varandra under spelets gång.
- Serven tar hand om all data och logik medan klienten hanterar användargränssnittet.
- Android applikationen är multitrådad för att ha en responsiv UI.
- Servern är också multitrådad för at kunna hantera flera klienter.

Denna hemuppgift var inte så jobbig som jag trodde. Den ända jobbiga delen var designen.
Eftersom server delen var redan klar behövde jag bara göra Android applikationen.

Att implementera sockets var inte heller svår då hanteringen gjordes på samma sätt som i första hemuppgiften.
Det ända problemet som jag stötte på var socket kopplingen via IP-adressen. Då Android emulator inte använde samma lokala IP-adress som Windows behövde jag använda en annan, 10.0.2.2 istället för 127.0.0.1.

6 Kommentar om kursen

Jag har spenderad 2–3 timmar åt att gå igenom föreläsningarna innan jag började med hemuppgiften. Tiden det tog att utföra hemuppgiften blev 15–16 timmar. Rapporten tog ungefär 2 timmar att skriva.