Informatics Large Practical

Plan - Coursework 1

Qais Patankar - s1620208

October 12, 2018

1 Introduction

Coinz is an Android app. Prior to the release of Android Studio I had attempted Android development. It was an *interesting* experience and everything was running excruciatingly slow on my old machine. So, after discovering that iOS development (with Swift) had a “better” ecosystem, I gave up on Android and have not touched the SDK since.

Since then I’ve shifted my focus towards developing webapps and I’d like to take this opportunity to delve back into app development, in particular, app development for Android.

One of the primary development goals I hope to achieve from this project is to gain some more practical experience with developing for systems with limited resources (i.e mobile phones).

Eventually, I hope that all the insight gained from this project will lead me into a hobby of professional app development.

2 Bonus features

2.1 Training

It should be possible for players to go back and ”train” on previous maps. This is especially useful if the player forgets to play the game one day and would like to explore the coins.

Players can pick up coins but they will not be added to the bank. It will be ”picked up” for the duration of the training session.

Leaving a training session will allow the player to return to the regular playing field. Returning a training session will give the player the option to *Pick up where you left off* or *Start over.*

It would also be useful to download a map for the user to train whilst not connected to the Internet.

2.2 Create & Share

It should be possible for players to create custom maps (in the regular area of play, but not on the regular map).

Custom maps can be configured to behave like time trials. This requires players to take an optimal route (that they have to figure out themselves) and get the most coins.

The coins picked up will count towards a *points* attribute that is displayed on a *track scoreboard*. This scoreboard is local to the specific track.

2.3 Scoreboard

A global scoreboard showing:

1. Richest players (players sorted by top bank coins only)
2. Collectors (all-time coins collected, regardless of what's currently in the bank)
3. Philanthropists (or if coins are sold for value, *Traders*)

2.4 Dark mode

Every application should have a dark mode. What if you want to play Coinz at night?!

2.5 Easter eggs

There should be sufficient easter eggs in the application to satiate the needs of power users. Power users should be rewarded for their hard work with valuable coins.

There should be special themes during Halloween, Christmas & Easter.

2.6 Friends

It should be possible to add friends and see a custom scoreboard with progress that day amongst your friends.

This feature would also make it easier to send coins.

Friends can opt-in for their location to be shared to their circle whilst there is a running game session.

3 Language of choice

3.1 Why *Java*?

- Java is a C-style programming language and doesn't stray too far from the norm, so a developer who knows C, C++ or JavaScript will easily be able to pick up the project and understand the code.
- Java has been in use for many years. It is clearly stable and well-supported (by Oracle, Google, and the rest of the internet community.)
- A lot of examples are available in Java.

3.2 Why *Kotlin*?

- Kotlin has a lot of shorthands that facilitates rapid prototyping. And this is *not* at the expense of readability![2]
- Kotlin solves a lot of issues that developers have had with Java... spend less time worrying about at *NullPointerException*—it's clear where and when such an exception can be thrown.
- Documentation is widely available. Communities exist online[1] to ask questions. Officially supported by Google for Android development.
- Anko *just makes a lot of things easier.* Developers should spend less time reinventing the wheel, and just getting things done.
- Ultimately, keeping on top of technology (and not sticking with the same-old legacy code) is important. Developers should always be up to date and shouldn't get too comfortable with one language. A developer learning Kotlin for the first time will gain an educational benefit from the experience.

3.3 What I've chosen

I've chosen *Kotlin* because I feel that keeping up to date and having code being free of cruft (i.e, easy to understand and visualise) is of the utmost importance for developers.

Many developers spend a lot of time reinventing the wheel and writing the same chunks of code (in a way that cannot be made DRY), and Kotlin either fixes the underlying problem, introduces shorthands, or has a library available to resolve these situations.

Kotlin is interopable with Java[4] and so a developer can still introduce some code from their own library. All Java code can be "converted" to Kotlin[5], and this is an educational tool to see how certain chunks of code can be implemented (albeit not idiomatically).

4 Timetable

Week 2 Create the project

1. Initialise Git repository
2. Set up Android project
3. Create LaTeX environment

Week 3 Getting things running

1. Make sure app works on phone (with *adb* use)
2. Make sure app works on emulator on DICE
3. Make sure app works on emulator on personal device

Week 4 A Mapbox full of Coinz

1. Get Mapbox working
2. User must update on map
3. Load the GeoJSON based on the current date
4. Coin pickups
5. Implement basic banking system

Week 5 It's hot. It's firey. It's Firebase.

1. Network the basic banking system
2. Add user system
3. Make sure each user has their own bank
4. Explore testing options

Week 6 Hail Corporate

1. Enhance banking system with Firebase
2. Make sure multi-user system is spick and span
3. Start working on promised bonus features

Week 7 Bonus

1. Continue working on promised bonus features
2. Explore where more unit tests can be added to the project

Week 8 It's time to paint

1. Polish everything
2. Design onboarding flow
3. Make the entire experience pretty

Week 9 Hack on the project

1. Hackathon to complete as many new features as possible
2. Refactor and improve messy classes/functions
3. Begin documenting parts of design that have not been realised in the implementation
4. Add tests for these new features

Week 10 Feature-freeze project

1. Finish documenting parts of design that have not been realised in the implementation
2. Remove features that are incomplete
3. Fix bugs left over from the hackathon
4. Remove features that are still way too buggy

Week 10 Security review & stress test

1. Ensure that the code is "safe" (in terms of *NullPointerException* and race conditions)
2. Make sure the application does not ask for too many permissions
3. Evaluate whether or not other users can cheat the system
4. Ensure any vulnerabilities have tests written to prevent regressions
5. Perform a stress test with many users

Week 12 Finishing touches

1. Annotate report with screenshots
2. Clean up anything you're unhappy about

Week 13 Submit

1. Finalise the report
2. Ensure everything is well documented
3. Submit the report
4. Review code submission information

References

[1] JetBrains and open source contributors, *Community - Kotlin Programming Language*. https://kotlinlang.org/community/

[2] Magnus Vinther, *Why you should totally switch to Kotlin*. Medium, https://medium.com/@magnus.chatt/why-you-should-totally-switch-to-kotlin-c7bbde9e10d5

[3] Anthony Awuzie, *Kotlin and Java: Where Do They Fit In?*. DZone, https://dzone.com/articles/kotlin-and-java-where-do-they-fit-in

[4] JetBrains and open source contributors, *Calling Java code from Kotlin*. https://kotlinlang.org/docs/reference/java-interop.html

[5] JetBrains s.r.o. *Converting a Java File to a Kotlin File* https://www.jetbrains.com/help/idea/converting-a-java-file-to-kotlin-file.html