COINZ: Clandestine Operation Imminent: Nocturnal Ziggurat

Radoslav M. Kirilchev (s1452923)

Informatics Large Practical 2018/2019

This report describes my implementation of the COINZ project. All core features, as well as a dramatic number of bonus mechanics and systems, have been implemented. No features described in the initial design document have been forgotten.

Contents

Introduction

The implementation of the COINZ project that I shall present here features a significant number of additional features over the initial specifications. This document serves as a guide to the game in general, and provides a detailed look into the various aspects of all features. Whilst all mandatory features have been implemented, a few liberties have been taken to facilitate greater compliance with bonus features, and to deepen the gameplay experience for players.

COINZ: Clandestine Operation Imminent: Nocturnal Ziggurat puts players into the shoes of an agent working to uncover the location of a missing Soviet-era nuclear warhead, codenamed *"Project Nocturnal Ziggurat"*, recently stolen whilst transported by the Kyrgyzstani government. Whilst the identity of the bomb thief is unknown, his activity has been observed in central Edinburgh, Scotland. In response, factions have begun making their moves. Players may choose to aid either the Eleventh Echelon, an elite department of the MI5 in the hopes of acquiring the bomb before anything goes wrong; or join up with the Crimson Dawn, a ruthless band of high-tech mercenaries seeking to lay claim to the nuke in order to sell it to the highest bidder on the international black market.

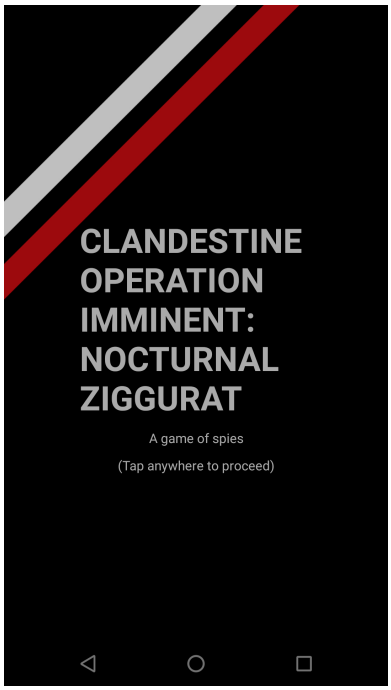


Figure 1: The splash screen of the game. Minimalistic and clean, the screen greets users every time the application is opened. The two strips of silver and red in the background represent the two factions.

To locate the bomb, players must decrypt secret messages – "cryptomessages" – found in the region. Cryptomessage decryption, however, requires computational power, and computational power – "compute" – must be purchased with money. For that reason, players are tasked with collecting the cryptocoins that spawn in the environment, whilst managing bank balances, deposit quotas, different currencies, faction-dependent providers of computational power, and so on.

Every deciphered cryptomessage brings the player's entire team a little bit closer to locating the nuclear warhead. The team that locates it first wins the game; the other loses.

Core Gameplay Loop

This section provides a more formalised look at the core gameplay loop of COINZ, and provides a brief description of every step in the process. Note that this section concerns itself with the way these features – the game mechanics – interact with each other. A deeper discussion on the features themselves, as well as application screenshots, may be found in the *Design and Features in Detail* section.

Coin collection – the main player activity, as could be expected – is tightly integrated with the various bonus features, generating a number of side activities – some established by the coursework specification, others not – that reward the player in different ways. Focus has been placed on intra-team cooperation and inter-team competition.

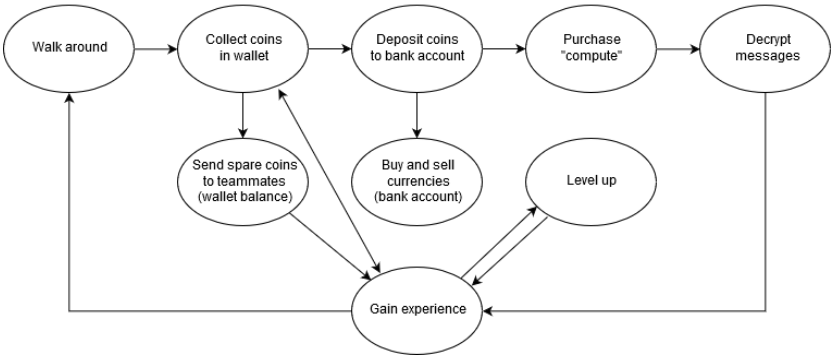


Figure 2: The main gameplay loop of COINZ. By walking around the map, players collect "spare change" coins of the four currencies in their wallets. These coins may then be deposited to the player's bank accounts (holding all currencies), subject to a fixed daily deposit quota, and can then be bought and sold for GOLD. Coins in wallets may also be sent to teammates; sending is not bound by the bank deposit quota. Currencies are then used to purchase "compute", which is spent to "decrypt messages" and propel the player's team closer to victory.

Figure 2 depicts a flow-chart representing the core gameplay loop of COINZ. Coins collected by walking around the map are stored in the player's "wallet" – this is equivalent to the "spare change" in the specification. Wallets can hold a fixed amount of every currency (with the exception of GOLD), and their capacity increases as the player gains levels.

Collected coins may either be deposited to the player's "bank account" – which holds balances for all currencies, including GOLD – subject to the fixed daily deposit quota, or sent to teammates. Coins sent to teammates are automatically converted to GOLD based on the sender's daily exchange rates. Whenever a player receives coins, a message confirming the transaction is recorded in their "mail".

From the bank account interface, players may also buy and sell the four collectable currencies with or for GOLD, again based on their daily exchange rates – which are defined as the daily "interbank rate" – the information sourced from the daily map data – plus a commission percentage, which in turn changes depending on the player's team or level.

With the currency balances in his bank account, a player may purchase "compute": computing power required to decrypt "cryptomessages". All "cryptomessages" have different costs, and contribute different amounts towards the player team's global victory counter.

Experience points are awarded for collecting coins, sending coins to teammates, and deciphering messages. Those experience points in turn increase the player's level, giving him greater team-specific bonuses and updating the player's "rank".

Design and Features in Detail

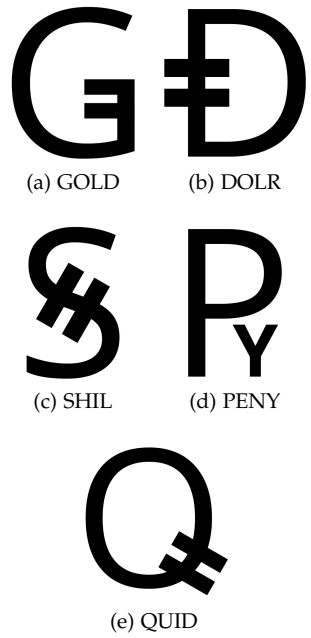
In this section we shall examine all implemented features in detail, assessing both implementation and design (UI) considerations. However, we begin with a systematisation of all features present. Individual features, and their respective game windows, shall be discussed in subsequent sections. All initially proposed features have been implemented.

Overview: Core and Bonus Features

The features of COINZ can be broadly classified into two groups: *core features* – those that were specified in the coursework document – and *bonus features* – those that were added to address the issue of underspecification and deepen the gameplay experience.

Here we shall list all such features, in order to provide a comprehensive yet abridged summary of the work done on the project:

- *Coin collection*:* Walking within a 25-metre radius of a coin collects it into the player's wallet, provided the wallet has enough free space of the respective currency to accommodate the coin – all coins "contain" different amounts of their respective currency.
- *Currencies*:* Collectable coins belong to one of four currencies: DOLR, SHIL, PENY, QUID. A fifth currency, GOLD, exists, but it is only used as an intermediary for financial transactions. or to purchase compute.
- *Coin deposits*:* Coins collected can be deposited to the player's bank accounts, subject to a deposit quota of 25 "coin units" a day. Coins deposited in such a way retain their respective currency.
- *Coin sending*:* Coins collected, but not deposited to a bank account, can be sent to players of the same team. Sent coins are converted to GOLD and deposited to the recipient's bank account.
- *Cloud storage*:* Using Google Firebase, players log in to a unique account using an email and a password, and players' data, including unique usernames, is stored persistently. This allows for progress tracking through different sessions and devices.
- *Currency icons:* All currencies have been given unique and distinctive icons inspired by real-world currency symbols, as can be seen in figure 3.



(a) GOLD (b) DOLR

(c) SHIL (d) PENY

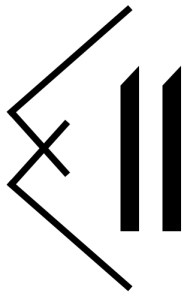
(e) QUID

Figure 3: The icons of all currencies in the game, shown in black for clarity. On the map, icons are given different colours that better match the colour scheme.

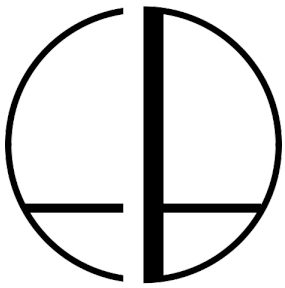
- *Non-unique coins:* The specification document seemed to imply that the individual collectable coins be treated as unique coins; in COINZ this is not the case. Coins are simply an amount of some currency and nothing more.
- *Automatic coin collection:* Whilst clicking on a coin will display information about its value and currency, no specific actions are needed to collect coins. Players may collect coins whilst observing other menus in the application.
- *Teams:* Upon registering for an account, players join one of two teams. Teams have distinct logos, identities, and ideological differences, as well as different team-specific benefits they grant to their members. See figure 4.
- *Narrative:* The game features a rudimentary "plot" which explains the necessity of coin-collection, gives and theoretically helps immerse players in the game.
- *Progression:* Actions beneficial to the team and the player are rewarded with experience points, allowing players to improve their standing with their team, unlocking better team-specific bonuses and titles.
- *Cryptomessages:* Progress towards winning the game is made by "decrypting" secret messages, tying into the game's narrative.
- *Compute:* (i.e. computational power) Another resource used for cryptomessage decryption, and purchased from "providers" using bank balances in the different currencies. Every team has a set of six providers, three of which are common and three of which are unique to the respective team.
- *War:* The game is won or lost by teams as a whole, not by individual players, creating a natural sense of competition between members of opposing teams, and underlining the importance of cooperation between teammates.
- *Tutorial:* Upon logging in for the first time, or for the first time on a new device, users will be presented with an informative tutorial that exposes some information about the game's mechanics and the specifics of the player's team.

Features marked with an asterisk *(*)* are core features; all others are bonus features.

In the following sections, instead of discussing each feature in a vacuum, we'll examine all features by doing a structured "tour" of the application.[1] All features initially proposed have been implemented, most with improvements.



(a) Insignia of the Eleventh Echelon, the playable fictional subdivision of the MI5. The icon evokes the image of a traditional longbow, as well as a pair of blades, or two candles that illuminate the dark.



(b) Insignia of the Crimson Dawn, the playable fictional group of high-tech mercenaries. It reminds of a sun almost risen above the horizon, a sundered all-encompassing circle, and of the blade of a razor.

Figure 4: The insignia logos of the two teams, along with some of the symbolism they embody.

[1] The interdependence of the game's systems, as well as the large number of bonus features, renders this approach more sensible and logical.

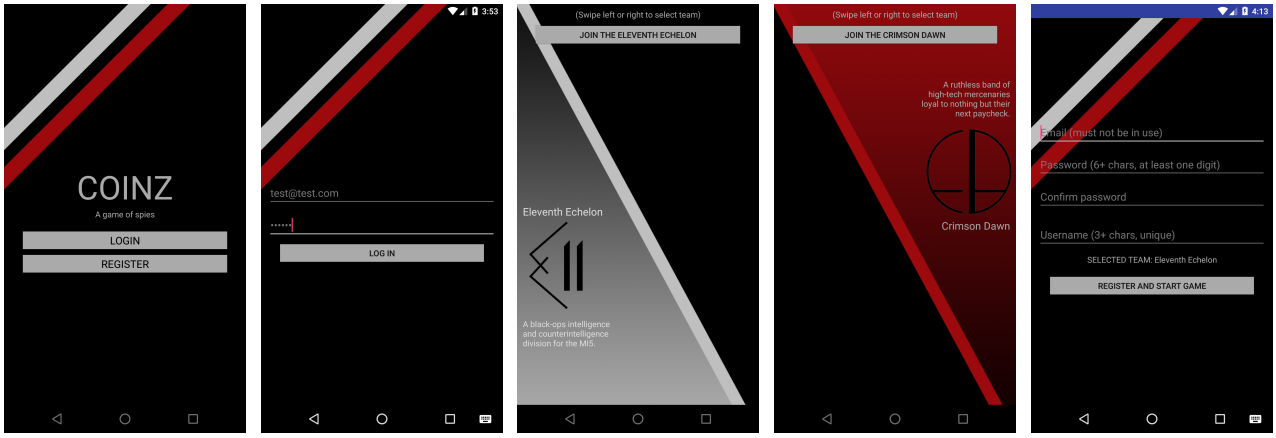
Accounts: Data and Storage

Code of interest:

- `MainMenuActivity.kt`
- `LoginActivity.kt`
- `RegisterActivity.kt`
- `AccountData.kt`
- `MapDataClasses.kt`

Accounts in COINZ are handled using *Cloud Firestore*, as required in the coursework specification. Users register using Firestore Authentication, and their data is stored in a Firestore database.

Let us now observe the menus dealing with user registration and login. After clicking on the splash screen (figure 1), the screen fades gently into the main menu: figure 5a.



(a) Main menu (b) Login screen (c) Registration: E11 (d) Registration: CD (e) Registration form

Figure 5: Login and registration menus (zoom in for details).

From the main menu, the player may opt to login or register. The login form (figure 5b) is rather straight-forward; upon clicking "Login", a player with a valid username and password will be allowed to access the game itself.

Registration is more interesting. Before filling in any other details, a new player is presented with the choice of a team using an interactive selector (figure 5c and d). Swiping left or right stylishly animates between the two teams.[2] Upon confirming team selection, the player is shown the registration form (figure 5e), where he may complete details. The form validates data before registering a new user; the following conditions must be met:

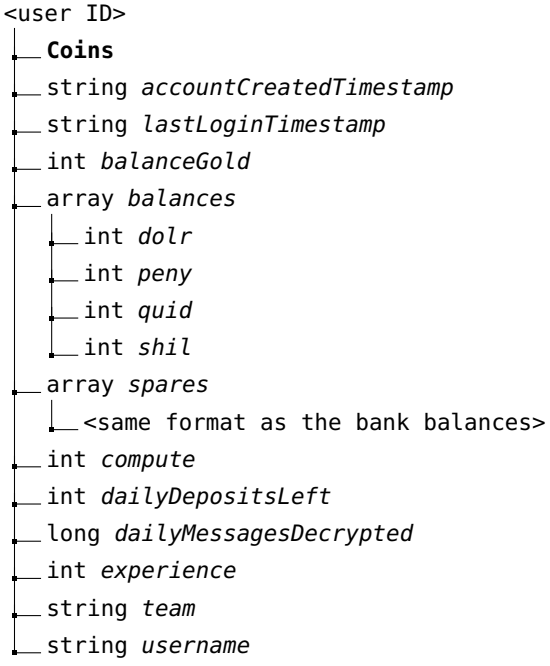
- Email and username must not be in use.
- The password must be at least 6 characters long, at least one of which must be a digit. In addition, the two passwords entered (original and confirmation) must be identical.

If the data is valid, the new user is registered with Firebase Authentication, and a new entry for his user ID is created in Firestore. The structure of the Firestore database itself is depicted in figure 6.

[2] Henceforth in this text, the abbreviations *E11* and *CD* shall be used for the purposes of brevity. "E11" stands for the Eleventh Echelon, and "CD" stands for the Crimson Dawn.

In Firestore, every user is a document in the "Users" collection. Every such user document, in turn, holds a set of data fields (that we shall address shortly), as well as a collection named "Coins", which contains a list of documents named after the daily timestamp retrieved from the map's JSON document, which, in turn, hold a set of boolean fields with the IDs of every coin taken on that day (figure 15). This allows us to quickly poll the respective document when the user logs in and see which coins have already been taken.

The data fields that the database holds per user are as follows:



Note that data bundled with the map – currency rates, coins and locations – is *not* included in the database otherwise.

The stored data should be rather self-explanatory, bar for a few points. All balances of any coin are stored as integers and multiplied by 100 in order to prevent rounding errors. "12.34 GOLD" in the game will be stored as "`balanceGold = 1234`" in the database. The same applies for `dailyDepositsLeft`, as I allow for the depositing of "pence" as well (e.g. a player may deposit 2.37 DOLR first thing in the morning, decrementing his daily deposits remaining from 25 to 22.63).

NB: "Daily Messages Encrypted" is a bit-mask of the decrypted message indices for the day of last login. This will be discussed in detail in section *"Compute": Messages, Providers, and Winning*.

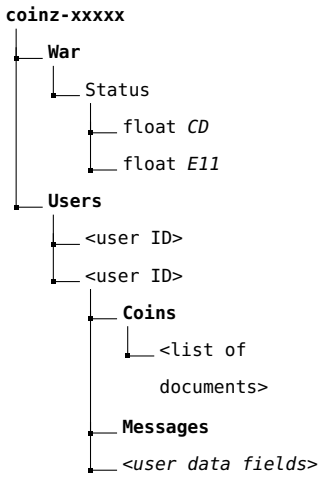


Figure 6: Layout of the Firestore database. Items in **bold** are collections, items in *italic* are individual fields in a document. Other items are documents. The specifics of the "War" document, the per-user "Coins" and "Messages" collections, and the list of user data fields shall be discussed later in this document.

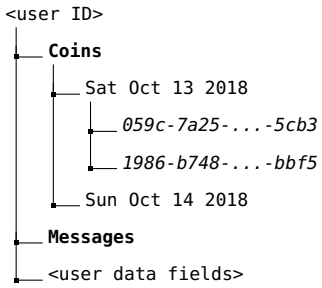


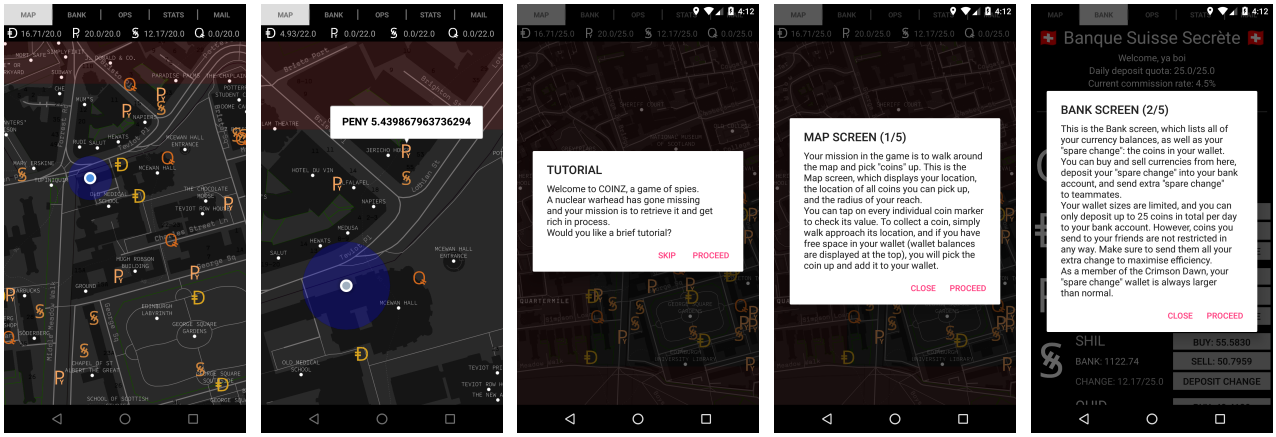
Figure 7: Example content of a user's "Coins" collection. Every document in the collection is a specific day; every day contains a boolean field (set to true) for every coin collected on that day, allowing the database to efficiently track progress across devices.

Map: Coin Collection

Code of interest:

- `GameActivity.kt`
- `MapFragment.kt`
- `DataManager.kt`

After a new player registers successfully, or after a recurring players logs in, the game switches to the map screen (figure 8a).



(a) Map screen (b) Coin information (c) Tutorial notification (d) Tutorial page 1 (e) Tutorial page 2

Figure 8: Map screen and tutorials (zoom in for details).

The map screen, powered by MapBox as per the coursework specification, utilises a customised "coding style" visual theme to fit the tonal theme of the game. The player is the dot; his collection radius – the blue circle.[3] Different coins are identified by their currency icons in different colours. Clicking on a coin will show the coin's amount (figure 8b). The map can be zoomed out to an extent, but is constrained within reasonable limits, given the limited gameplay area. The edges of said area are also denoted using a faint red overlay.

At the top of the screen, above the map area, we find the five main "tabs" of the game, in which different sections are hosted. "Map", "Bank", "Ops" (Operations), "Stats", and "Mail" (all these windows will be examined in detail in the following sections). Beneath the tab buttons. The player may also scroll left or right to change screens (when not on the map window).

Upon a first-time login, or afterwards if so desired, the player may be greeted by the tutorial screen (figure 8c-e). The optional tutorial is team-dependent and drives the player through the five tabs, explaining the basics and the goal of the game, as well as listing some team-dependent perks. A flag marking whether the tutorial was shown is the only piece of information persistently stored locally on the device.

[3] Permissions for location are required upon starting the game. Denying them closes the application.

The map data is downloaded immediately after a successful

login, and only stored in memory temporarily, whilst the user is logged in. The HTTP request to download the map is performed using the Fuel library[4] and the JSON file is parsed using Moshi.[5]

After the JSON file is parsed, the database entry for the current user is retrieved. If this login is taking place on a day different than the recorded ones, daily deposit quotas, decrypted messages, and the last login timestamp are updated.

In addition, all IDs of coins already collected on this day are obtained, and removed from the set of coins supplied by the JSON file. This is to prevent players from collecting the same coin twice. Collectable coins are added to a `HashMap` with a key type of `string` – the coin's unique ID – and a value type of `CoinInfo` – a simple class that holds the coin's currency, its value, and its latitude and longitude. All other coin-specific information from the JSON file is discarded.

[4] `https://github.com/kittinunf/Fuel`

[5] `https://github.com/square/moshi`

Map markers are drawn naïvely, and so are distance comparisons made. Given the small number of coins available at any one time – up to 50 at most – there is no need for special optimisations, as performance is not an issue. As the player moves around the map, the distance from his location to all coins is measured. Any coins falling within a 25-metre radius – as per the specification – of him are collected, if his wallet is not currently full of the respective currency. Collected coins are removed from the `HashMap`, experience is awarded.[6] the database is notified of the changes, and all markers – including the player's position – are updated. The various fragments can register themselves as `"UIListeners"` to this event in a simple implementation of the Observer design pattern, and thus all such listeners receive callbacks and update the interface.

[6] The amount of experience awarded for the collection of a coin is equal to the value of the coin, rounded down.

Coin collection happens automatically, without a specific need for confirmation from the user. This decision was made to streamline gameplay, given that coin collection is the "backbone" activity of the game that is bound to be repeated numerous times during a single play-session, and to allow for the collection of coins whilst the player is browsing through other menus.

Addendum: Though of no significant importance, the structure of the interface is as follows. There exist a few activities: `MainMenuActivity`, `RegisterActivity`, `LoginActivity`, and `GameActivity`. All of these inherit from a `BaseFullscreenActivity`.[7] The `GameActivity` hosts a `SmartTabLayout` component.[8] That component – forming the five "tab buttons" at the top of the screen seen in figure 8a and b – complemented by a custom-made "`SmartViewPager`" that hosts all five tabs as separate fragments: `MapFragment`, `BankFragment`, `WarFragment`, `StatsFragment`, and `MailFragment`.

[7] The base activity serves as a centralised location to set the various system flags for full-screen utilisation.

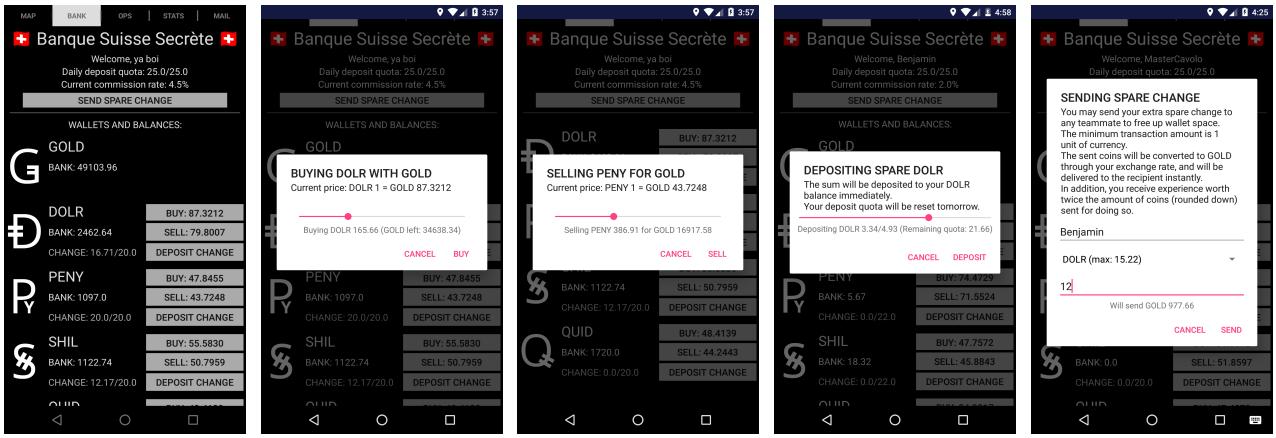
[8] `https://github.com/ogaclejapan/SmartTabLayout`

Bank: Sending and Exchanging

Code of interest:

- `BankFragment.kt`
- `DataManager.kt`
- `CurrencyView.kt`
- `Currency.kt`

CLICKING ON THE "BANK" TAB scrolls the screen to the Bank window, a focal point for the player's personal strategic decision-making during gameplay.



(a) Bank screen (b) Buy menu (DOLR) (c) Sell menu (PENY) (d) Depositing "spares" (e) Sending "spares"

Figure 9: Bank screen and child dialogs (zoom in for details).

THE BANK WINDOW shows players the amount of currency they possess, both in their wallets as well as their bank balances, displays the player's username, daily deposit quota, and commission rate, and facilitates currency exchanges and the sending of spare change to teammates (figure 8a).[9]

Currencies are listed in a scrollable window using a generic UI component called the `CurrencyView`. The presence of the currency's unique icon helps players recognise the map coins with ease.

Each of the collectable currencies present in the player's bank balances can be bought with GOLD (figure 8b) or sold for GOLD (figure 8c). The daily rates are obtained by applying the player's personal commission rate[10] to the "pure rates" listed in the daily map's JSON file.

The player's personal commission rate is determined by his team and level. Members of the Crimson Dawn have a fixed 4.5% commission rate, whereas members of the Eleventh Echelon have a variable rate ranging from 4.0% at level 1 to 0.2% at level 10. (Team bonuses are discussed in detail in section *Teams: Narrative, Experience, and Progression*.)

The presence of this commission rate naturally decreases the value of exchanged currencies – as currency sold will produce less GOLD than the price to buy the same amount of currency – and drives play-

[9] Freshly-registered players are started off with a bank balance of 300 GOLD and 200 compute.

[10] The buying and selling prices of a currency are obtained as per the following formulae:

$$P_B(x) = R(x)\dot{(}1 + \frac{C}{100}\dot{)}$$

$$P_S(x) = R(x)\dot{(}1 - \frac{C}{100}\dot{)}$$

where x is the given currency, $R(x)$ is the currency's "pure rate" to GOLD as taken from the map file, and C is the player's commission rate in percent. In other words, percentages are simply being added or subtracted.

ers to collect a diverse amount of coins, even ones that are "cheap" on this particular day. Bank accounts have no limits to the amount of currency they can hold.

WALLET BALANCES ARE PRESENT HERE under the "change" listings for each currency, as is the wallet balance. Similarly to the commission rate, wallet balances are dependent on team and level. Members of the Eleventh Echelon have wallet sizes ranging from 15 at level 1 to 35 at level 10, whereas members of the Crimson Dawn have a wallet capacity of 18 at level 1, improving gradually up to 45 at level 10.[11]

"Spare change" – wallet balances – can be deposited to the player's bank account (figure 8d), limited to the fixed daily quota of 25 coins, as per the coursework specification. This amount is agnostic to the currency being deposited. However, the minimum deposit amount is one "penny" of any currency, and this is also the granularity of the depositing amount, allowing the player a finer control over his deposits.

Alternatively, "spare change" can be sent to teammates, requiring knowledge of another user's username (figure 8e). This is not subject to any deposit quotas, but the sent coins are automatically converted to GOLD as per the *sender*'s daily rates. In addition, the minimum amount of spare change that can be sent is one full coin, no matter the currency or its current GOLD value. However, even with those restrictions, this remains the only way remedy a full wallet once the daily 25 coins are deposited. To further encourage cooperation within the team, users are rewarded with experience equal to twice the amount of the coins sent, rounded down.

The recipient's database entry is updated simultaneously with the sender's, after the sender commits a transaction.

Furthermore, players that receive a transaction by a teammate are notified of this in their "Mail" tab. The message is team-dependent and displays the sender's name and team-and-level-specific title, as well as the amount of GOLD received.

Addendum: The bank is named *"Banque Suisse Secrète"* ("Secret Swiss Bank") as a tongue-in-cheek reference to the allegedly popular usage of Swiss banks by shady organisations, governments, and other entities of similar calibres.

[11] Again, whilst the wallet size is just one, this capacity applies to coins of different currencies individually: with a wallet size of 15, one can hold up to 15 coins of every currency, not 15 coins in total.

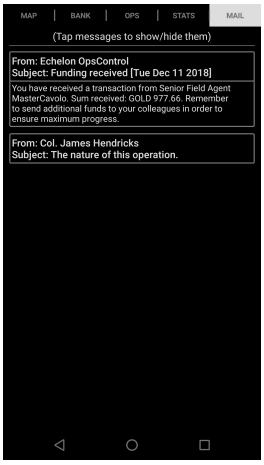


Figure 10: An example of the notification received by the player "Benjamin" after the transaction sent by the player "MasterCavolo" in figure 9e. The message, available through the "Mail" tab, shows off the sender's level, and provides a timestamp of the event (zoom in for details).

"Compute": Messages, Providers, and Winning

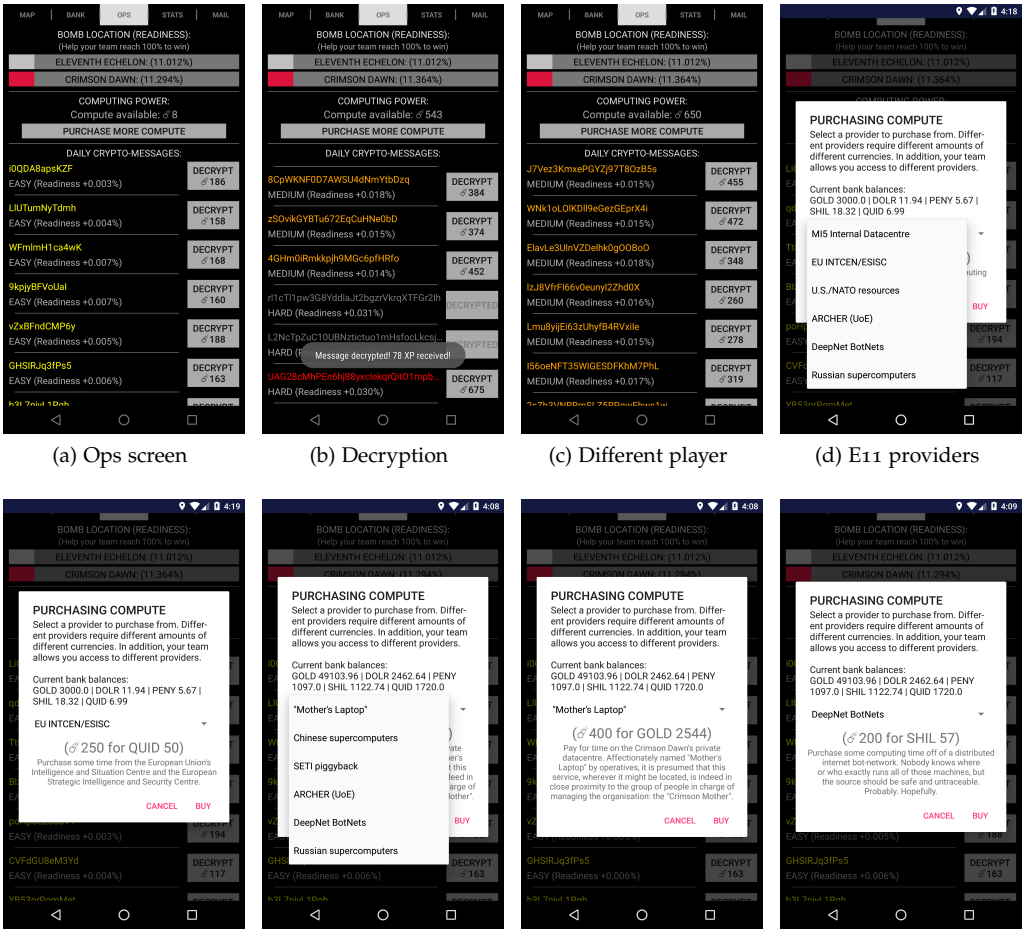
Code of interest:

- `WarFragment.kt`
- `DataManager.kt`
- `Messages.kt`
- `MessageProviders.kt`
- `CryptomessageView.kt`

The accumulation and spending of "compute", or computational power, is the direct goal of the game. Collected coins contribute to bank balances, which in turn may be used to purchase compute. Compute is used to "decrypt" "cryptomessages", which brings the player's team victory percentage.

To examine compute, we need to also examine the "Ops" (Operations) screen.[12]

[12] Initially planned as the "War" screen, explaining names like `WarFragment` found in the code.



(a) Ops screen (b) Decryption (c) Different player (d) E11 providers

(e) E11 provider selected (f) CD providers (g) CD provider selected (h) A common provider

Figure 11: Ops screen, cryptomessages, and compute providers (zoom in for details).

The Ops screen (figure 11a) displays the current "Readiness" standing of both teams, which is simply a narrative term for "victory percentage". The screen also lists the player's current amount of compute, and opens the dialogs for the purchase of extra compute (figure 11d-h). Below these, the list of the player's daily cryptomes-

sages is displayed, along with the option to decrypt each of them. The purpose of compute and cryptomessages is to establish a narrative connection to the global victory condition (the "war" between the two teams), and to add another layer of complexity to the gameplay experience.

From a narrative perspective, "cryptomessages" are supposed to be secret communication between the party or parties currently in possession of the nuclear bomb (see section *Teams: Narrative, Experience, and Progression* for a more detailed explanation). From a technical perspective, however, they are a procedurally-generated set of options, unique to every player, yet still persistent for the day.

After the player logs in and the map data is downloaded, the daily "cryptosettings" are also generated. This is done by obtaining a *base seed* value exclusive to the player[13], which is used to instantiate a random number generator ("Seeding" a random generator with a specific value makes the generated sequence the same every time the same seed is used). Affixing the seed generation to be exclusive to the player and the current day means that the same user will receive the same set of cryptomessages for any one day, no matter the device used.

This random generator is then used to produce a random amount of cryptomessages of three difficulties: easy, medium, and hard. The number of messages generated is randomly selected per each difficulty, ranging from 0 up to the maximum number of messages of that difficulty. Every generated message also has a random readiness bonus value, as well as a random compute price, again chosen randomly within bounds set by the difficulty. Table 1 lists these bounds.

[13] The base seed is generated using the following formula:

$$\text{seed} = H(\text{uid} \| \text{dailyTimestamp})$$

where "uid" is the user's unique Firestore ID, "dailyTimestamp" is the timestamp value obtained from the map file, $\|$ denotes string concatenation, and H is the native hash code function available in Kotlin. The resulting value is cast into a "long" (64-bit integer).

Difficulty	Min bonus	Max bonus	Min price	Max price	Max messages	String length
Easy	0.003%	0.007%	100	200	12	12
Medium	0.012%	0.018%	250	500	8	24
Hard	0.030%	0.040%	500	1000	6	36

Table 1: A summary of the bounds for a cryptomessage's readiness bonus, compute price, and total number of messages per difficulty. "String length" is how long the randomly-generated string that represent the message is.

As seen in figure 11a-c, every cryptomessage is represented by a randomly-generated nonsense string of a certain length. The length of the string is also dependent on the difficulty of the message, and generated strings are also "fixed", meaning that the same cryptomessage will always produce the same string, even on subsequent logins.

Messages can only be decrypted once per day. In the interface, they are created using a custom-built `CryptomessageView` component, which is responsible for the creation of the aforementioned nonsense string.

In Firestore, the decrypted messages for the day are stored as a

bit-mask in the `dailyMessagesDecrypted` field. The field is a 64-bit integer, of which 20 bits are reserved for each message difficulty. The "decrypted" status of one message is treated like a single bit flag in this masked field.[14]

On average, an easy message contributes 0.005% readiness for an average price of 150 compute, a medium message provides 0.015% for 375 compute, and a hard message provides 0.035% compute for 750 compute. Whilst the cost-efficient "value" of messages decreases as difficulty increases, medium messages provide a narrow band of valuese, and hard messages provide a large amount of readiness. It should be evident that the amounts provided are this small in order to accommodate multiple players playing the game at the same time – larger individual contributions would propel a team to victory much faster, and diminish the influence of a large team.

Experience is awarded for message decryption, given that it is the main gameplay actvity that contributes to the team's victory. The amount of experience points awarded for decrypting one message is equal to twice the message's bonus multiplied by 1000.[15]

As stated earlier, compute is necessary to decrypt messages. Compute is a secondary resource obtained through "providers" – narratively, organisations that can provide computational support to paying customers. Mechanically, providers are merely different ways to obtain compute, taking different currencies and providing different value for money.

[14] As messages are generated in order, each has an index from 0 to the max number of messages per difficulty (table 1). The set/check operations for messages (i.e. flagging a message as "decrypted" or checking whether it is decrypted) is done in the following manner:

$$\text{Set}(i, d) = M \vee (2 \ll i + B \cdot d)$$

$$\text{Check}(i, d) = M \wedge (2 \ll i + B \cdot d)$$

where i is the message index, d is the difficulty index (easy = 0, medium = 1, hard = 2), B is the number of bits reserved per field (20), and M is the total value of the bit-mask field.

[15] Example: if a medium-difficulty message contributes 0.016% to the global victory counter, the experience points awarded will be equal to 32.

Name	Compute	Base price	Currency	Price multipliers		Price per 1 compute
				E11	CD	
MI5 Internal Datacentre	500	3000	GOLD	1.0x	N/A	6
EU INTCEN/ESISC	250	50	QUID	1.0x	N/A	0.2
U.S./NATO resources	600	120	DOLR	1.0x	N/A	0.2
"Mother's Laptop"	400	2650	GOLD	N/A	1.0x	6.625
Chinese supercomputers	200	60	PENY	N/A	1.0x	0.3
SETI piggyback	100	500	QUID	N/A	1.0x	5
ARCHER (UoE)	1000	100	PENY	1.1x	1.0x	0.1
DeepNet BotNets	200	60	SHIL	1.0x	1.0x	0.3
Russian supercomputers	400	40	DOLR	1.0x	1.15x	0.1

Table 2: A summary of all cryptomessage providers. Providers unavailable for a specific team are listed as "N/A" under the team's price multiplier column. The "Mother's Laptop" provider, a mirror of the "MI5 Internal Datacentre", is more expensive to compensate for the Crimson Dawn team's perk that decreases prices of compute.

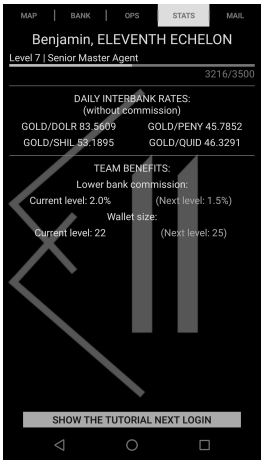
Each team has access to three exclusive providers, as well as three common providers. This makes for a total of nine different providers (figure 11d and f). Team-specific providers are flavoured to better match the specific team's personality. Providers take fixed amounts of different currencies and provide fixed amounts of compute in return. In addition, every provider is associated with a tongue-in-cheek

remark in the dialog screens (figure 11e, g, and h). All providers are listed in table 2.

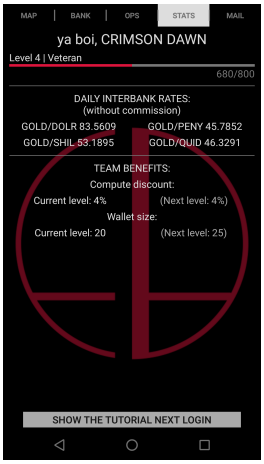
Both teams have access to providers for all currencies. In addition, there is only provider accepting GOLD per team, and doing so requires considerably larger amounts of GOLD than otherwise. This is because the "pure" exchange rates of the other currencies to gold appear to drift in the 20:1 to 80:1 range, rendering GOLD a rather weak currency, necessitating higher prices for its usage.

Members of the Crimson Dawn benefit from a compute cost discount from all providers, from 1% at level 1, to 12% at level 10. The "Mother's Laptop" provider, intended to be a mirror match for the Eleventh Echelon's "MI5 Internal Datacentre", is thus about 10% more expensive in order to offset this benefit. The effects of this discount can be seen in figure 11g and h.

One more factor comes into play with common providers: the price multiplier. Two of the three common providers apply a slight price increase to one of the two teams, simulating a greater difficulty of the respective team to acquire access to the service.



(a) Eleventh Echelon



(b) Crimson Dawn

Figure 12: The Stats screen, as seen by members of the two teams at various levels. The user's name, title, team, and level, as well as the daily interbank rates and level-specific perks for the current and next level are listed.

WINNING THE GAME is done by the first team that reaches 100% "readiness", representing that the team has located the nuclear device and may now retrieve it. In the event of a victory, the database doesn't reset itself automatically – the readiness values must be reset manually in order for a new game to start. The recommended starting values for team readiness are 11%, to encourage players to contribute to the "war effort".

It's additionally worth noting that the global readiness counter seen in the Ops screen (figure 11a-c) updates in real time whenever *any* player decrypts a message.

Teams: Narrative, Experience, and Progression

EXPERIENCE AWARDS HAVE BEEN MENTIONED throughout this document. Experience points, awarded for collecting coins, sending spare change to teammates, and decrypting messages, are used to progress the player through levels.

There are ten levels in total, requiring fixed amounts of experience to reach. Advancing to a always level requires more experience than advancing to the previous one. Both teams gain unique titles at every level, and have their wallet sizes gradually increased (with an advantage for members of the Crimson Dawn). Members of the Eleventh Echelon also gain improved bank commission rates, and members of the Crimson Dawn gain discounts when purchasing compute. All

information relevant to experience is summarised in table 3.

A player may check his current status by accessing the Stats screen (figure 12). This window reports the daily "pure" currency rates, obtained from the map file, as well as information about the play: name, title, team, current experience and experience left to the next level, as well as current team benefits and how they'll change at the next level. Furthermore, the team's logo is present in the background with its respective colour; the progress bar also reflects the player's team's preferred colour.

Code of interest:

- `StatsFragment.kt`
- `Experience.kt`
- `MailFragment.kt`
- `ExpandableMessageView.kt`

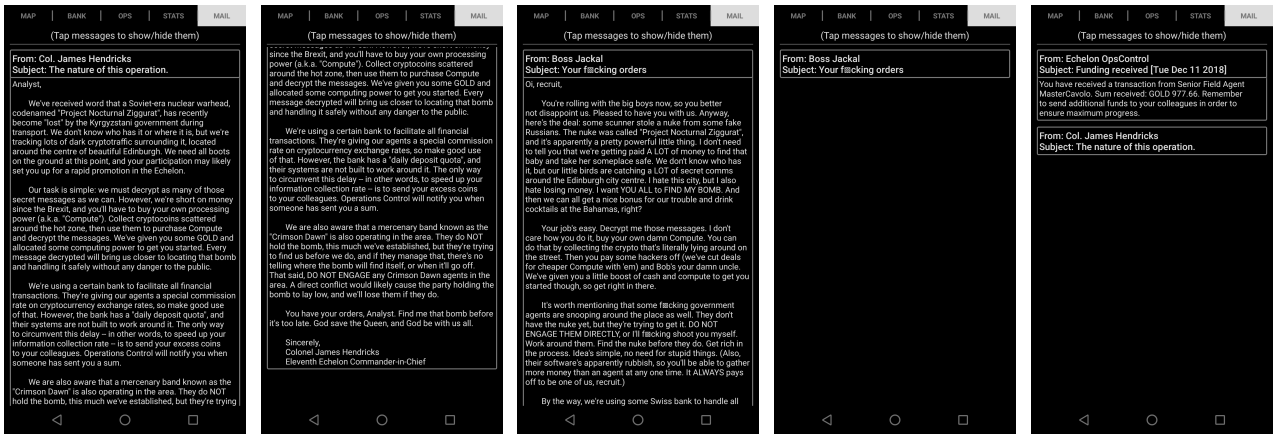
Level	XP needed	XP delta	Player title		Wallet size		Commission rate (E11)	Compute discount (CD)
			E11	CD	E11	CD		
1	0	-	Analyst	Recruit	15	18	4.0%	1%
2	100	100	Field Analyst	Grunt	15	18	3.5%	1%
3	250	150	Junior Agent	Enforcer	18	20	3.0%	3%
4	500	250	Field Agent	Veteran	20	20	3.0%	4%
5	800	300	Senior Field Agent	Specialist	20	25	2.5%	4%
6	1500	700	Master Agent	Elite	22	30	2.5%	4%
7	2200	700	Senior Master Agent	Expert	22	30	2.0%	7%
8	3500	1300	Special Agent	Master at Arms	25	35	1.5%	7%
9	5000	1500	Agent Double-Zero	Operative	25	35	1.0%	9%
10	10000	5000	Agent One	Ops. Commander	35	45	0.2%	12%

Table 3: A summary of all ten levels. "XP" stands for "Experience"; "XP delta" is how much experience must be acquired to progress from the previous level to the next. "Ops." is short for "Operations". Members of the Crimson Dawn have a fixed commission rate of 4.5% at all levels; members of the Eleventh Echelon gain no compute discount.

The narrative of the game largely serves to differentiate the two teams. The stolen Kyrgyzstani nuke, codenamed "Project Nocturnal Ziggurat", has gone missing and appears to be possessed by someone in the vicinity of the city centre in Edinburgh. It is in this conflict that the differences between the two teams can be allowed to shine.

The Eleventh Echelon, as a subsection of the MI5, is an honourable and moral organisation that seeks to recover the nuke before it is hopelessly engulfed by the black market. The Crimson Dawn, on the other hand, being a band of scrupleless high-tech mercenaries, is largely concerned with making a lot of money off the nuke, rather than whom it goes to.

The differences between the teams are made painfully obvious in the "introductory messages" that new players receive. The aforementioned Mail screen, whose main purpose it to document incoming financial transactions (figure 13e), is also used to "send" the player a message originating from the leader of their specific team – an upstanding Colonel James Hendricks for the Eleventh Echelon, and a menacing, likely unhinged psychopath known as "Boss Jackal" for the Crimson Dawn. (figure 13a-c)



(a) E11 intro message (b) E11 message, cont. (c) CD intro message (d) Message "closed" (e) Transaction note, E11

Figure 13: Mail screen and team-specific introductory messages (zoom in for details).

In these messages, it is easy to portray the ideology of the respective team, and the goals this faction wishes to accomplish by retrieving the nuke. This helps immerse players into the game world, and helps introduce players to the game mechanics and the reasoning for their existence.

The aforementioned compute providers, as well as the transaction messages, are also flavoured to match the two teams – e.g. containing information about the existence of the "Crimson Mother" (the "Mother's Laptop" provider), a nebulous entity that rules the Crimson Dawn. Compare the transaction message received by a member of Eleventh Echelon (figure 13e) to a message with the same purpose, received by a member of the Crimson Dawn (figure 14).

These little hints and details do a great job of giving distinct identities to the two teams, and hopefully immersing players into the game.

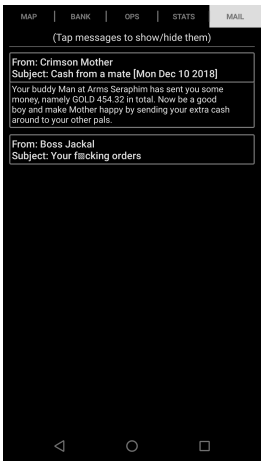


Figure 14: A transaction message as received by a member of the Crimson Dawn.

Addendum: Mail messages can also be opened and closed with a smooth animation. This is done using a special `ExpandableMessageView` component written for that express purpose.

In Firestore, transaction messages are also stored indefinitely, as fields of a document named "Transactions" in the "Messages" collection.

At present, messages may not be deleted from the user's Mail window.

The system, as it is created, can be easily extended to support player-to-player "emails". However, I decided against doing that during the planning stage, as that would simply be icing on the already-overloaded cake of bonus features.

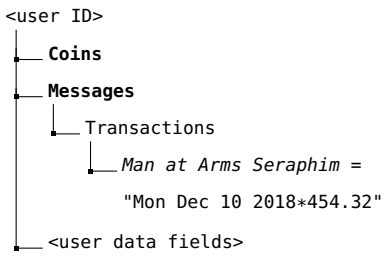


Figure 15: Example content of a user's "Messages" collection. Messages are fields of the "Transactions" document; the ID of the field is the full title and name of the sender, and its value – the timestamp of the transaction and the GOLD amount, separated by an asterisk.

VCS & Testing

As per the coursework requirements, the project uses a private Git[16] repository on github. The repo may be accessed at

```
https://github.com/radoMK/coinz
```

by authorised accounts.

Do not look at the number of commits. I'm more of a "no more than two to three commits a day" kind of guy, because I focus on a single feature and work it to completion.

Unit testing has been limited. I have never *not* despised unit tests, and attempting to write instrumented tests in Kotlin was nothing short of torturous. As a result, I've included some rather thorough tests of the behaviour of my custom systems, and a few tests that make sure the main menu activity is in order.

The application has been tested using the free version of the brilliant **Genymotion emulator**, with a **Google Pixel** configuration **(API 26, 1080x1920, 420 dpi)** (Android 8.0.0). The minimum tested API level is 26. All screenshots have been produced using the Genymotion emulator.

Unfortunately, testing the application on other platforms was met with problems that were out of my hands. Attempting to test on the default Google emulator (same configuration, but **480 dpi** by default), or on my own device **(Sony Xperia X, API 26, 1080x1920, 441 dpi)** was met with problems. Consider figure 16.

[16] Although I prefer Mercurial due to its proper Windows support...

Testing accounts: You may use these accounts to test the game more easily, as they come with some currency in both wallets and banks, and are of different levels.

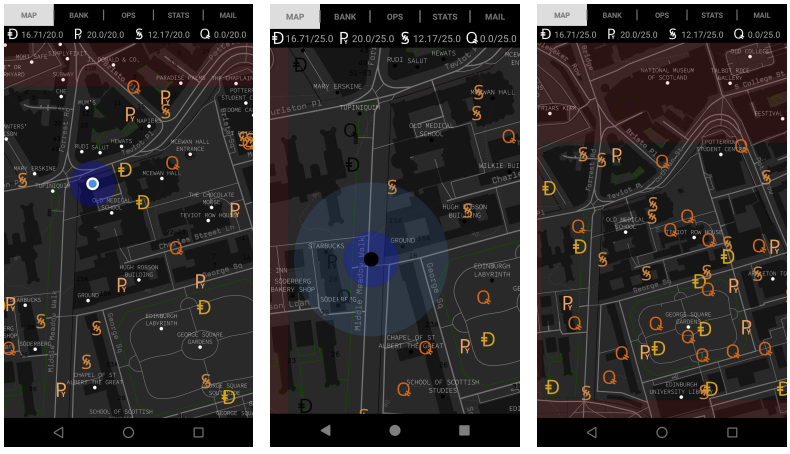
The password for all test accounts is "123456".

Eleventh Echelon:

- `test3@test.com` | "MegaloManiac", level 3, small amounts of resources;
- `test4@test.com` | "Benjamin", level 7, moderate resources;

Crimson Dawn:

- `test@test.com` | "ya boi", level 3, scant resources;
- `jackal@crimsondawn.com` | "Boss-Jackal", level 10, *lots* of resources;



(a) Genymotion (b) Default emulator (c) Real Xperia X

Figure 16: A comparison of the map screen under different emulation conditions, showing problems with MapBox:
(a): The Genymotion emulator, working perfectly. The player's location is reported correctly, and all markers are working well.
(b): The default emulator bundled with Android Studio. In this case, half the markers on the map are all black for no discernible reason. The issues vary as the map redraws itself, with most frequently *all* markers being fully black.
(c): My personal Android device. The location service fails to register at all, and this is an error with MapBox that is impossible for me to fix.

The issues with the markers and the emulator have been reported by others on Piazza numerous times.[17]

The problem with the location services has been tracked down to a hard to reproduce bug in MapBox that some people report fix and others – don't.[18] It appears to be device-specific. The application seems to work on my phone if I launch Google Maps first and wait for the "location dot" to show up, then re-open COINZ, but I can't confirm whether this is 100% reproducible.

In any case, my implementation is correct and any failure of the markers to be drawn correctly, or of the location services to launch, is a fault with MapBox that I shall not be held responsible for.

In the interest of avoiding such issues with a proprietary library, perhaps a reasonable suggestion for future years of this course would be to allow students to pick their own mapping libraries.[19]

[17] @77, @140, and @305; as well as @100 and @248 (in the follow-ups).

[18] `https://github.com/mapbox/mapbox-navigation-android/issues/1121`

[19] Perhaps on the natural condition that they're free.

Acknowledgements

This final section serves to document important links and sources used in the creation of this implementation.

First and foremost is the Tufte-LaTeX[20] package, which was used to create this document.

The main libraries and frameworks used, aside from the mandated MapBox and Firebase, are:

- `SmartTabLayout`, a better tab layout component;[21]
- `Fuel`, an HTTP networking library;[22]
- `Moshi`. a brilliant JSON parser;[23]

An honourable mention goes to `jsonschema2pojo`[24], which was used to generate Java objects with Moshi annotations from the JSON file; those were then converted to Kotlin by hand.

Finally, links with useful pieces of information, chiefly from StackOverflow:

- A utility function to generate MapBox icons from drawables,[25] used for the coin markers;
- A helper for animating ProgressBars;[26]
- A clever way to detect an application's first run;[27]
- A method for motion interception[28] that was partially used in my `SmartViewPager`.

Thanks for reading!

[20] `https://github.com/Tufte-LaTeX/tufte-latex`: A style inspired by the works of Edward Tufte!

[21] `https://github.com/ogaclejapan/SmartTabLayout`

[22] `https://github.com/kittinunf/Fuel`

[23] `https://github.com/square/moshi`

[24] `http://www.jsonschema2pojo.org/`

[25] `https://github.com/mapbox/mapbox-gl-native/issues/8185`

[26] `https://stackoverflow.com/a/18015071`

[27] `https://stackoverflow.com/a/30274315/668143`

[28] `https://stackoverflow.com/a/9650884/668143`