Software Engineering Large Practical 2016-17

GR**A**BBLE
Implementation Report

Designed and planned by
Radoslav Kirilchev – s1452923

Table of Contents

1. Summary

GR**A**BBLE (to be depicted with a bold "A") is the name of the Android game for this course, developed solely by me.

This is the "documentation" report, as required by the final part 3 of the coursework.

As seen in the table of contents, we'll go over all the points in the final part of the coursework requirements, plus some additional information.

Concept created by Radoslav Kirilchev
Zero Vector Studios, 2016
www.zerovector.eu

2. Core features

1. **Downloading and parsing the daily Placemark list**
   - Placemark downloading is done asynchronously upon the player's login. The entire document is written into a *Document* object, which is then parsed using DOM.
   - The Document is then iterated over, and all placemark references (wrapped in the *Placemark*) class are placed into a *SortedMap* (more specifically, using the *TreeMap* implementation) to avoid any erroneous duplicate IDs that might be in existence.
   - The minimum and maximum latitude and longitude of all points, essentially the game's playing field, is established during this process. *(See section 2.3)*
   - These placemark references, contained in the global-behaviour *Game* class, are used (by the *CityMap* fragment that implements the Google Map View) as keys to a *HashMap* that pairs them with a *Marker* object, required for the map, and a *BitmapDescriptor* for the marker icon.
2. **Efficient Grabble dictionary lookup**
   - In spite of what was perhaps implicitly suggested, the game does not allow its players to form words which are *not* present in the Grabble dictionary; in other words, a player *always* works towards a word drawn from the Grabble dictionary.
   - The provided text file with all words is included with the application. Parsing it is done asynchronously upon the player's login. The file is read line-by-line, and individual words, capitalised and wrapped in the *Word* class, are added to a *GrabbleDict* object.
   - The *GrabbleDict* maintains its list of words in a *HashSet*, effectively forbidding duplicates from appearing. (**Note:** This, coupled with the uppercase treatment of all words, reduces the total amount of entries in the dictionary by over a hundred.)
   - A beneficial side effect of using a Java *HashSet* is that the words end up unsorted despite being passed in alphabetically. This allows us to omit random selection later on.
   - Upon registering and logging in for the first time, the player is assigned the first word in this randomised set. The player may not change the current objective, ergo the only time where dictionary lookup is necessary is in the event of a player completing a word.
   - Upon completing a word, the set of all incomplete words is iterated in order (which is, due to the *HashSet*s, implicitly randomised). The *best-possible* word for the player is selected, using the *Kirilchev Coefficient* of the current word in regards to the player's current Inventory.

- The *Kirilchev Coefficient*, named after its creator (yours truly), is a simple, "kernelised" distance measure between two frequency fields in a speculative "bag-of-letters" model.

3. **Efficient per-location letter lookup**
   - To prevent the system from iterating over 1000 placemarks all the time, I've implemented a *segmentation system* (largely through the *MapSegments* class).
   - After the parsing of the daily placemark list is complete, the total play area is calculated and split into a number of rectangular segments with equal sizes. Each segment has a minimum size as declared by a global constant.
   - At any point during the gameplay, when the location of any nearby letters is requested, the segmentation system calculates the player's current segment, then calculates distances only for those placemarks that fall within segments immediately near the current one.
   - This takes into consideration the side of the player's Sight radius as well.
   - Distance calculation is also done asynchronously.

3. Cut features

1. **Multiplayer**
   - The multiplayer component, rather unlike the professional solutions I'm used to working with in the game development industry, resulted in too great of a time investment for the functionality it'd be used for. This required its removal.
   - The left-over multiplayer files (a written-from-scratch, incomplete framework based on JavaScript using NodeJS, Express, and MongoDB) can be found in an archive on the *multiplayer* branch of the BitBucket repository. *(See section 5)*

4. Bonus features

1. **Skills system (post-Design addition)**
   - To replace the cut Multiplayer component, I added a *Skills* system to further expand the progression element.
   - In short, both Alignments *(teams)* gain a unique set of four Skills, which provide them with different benefits. Skills are unlocked at different levels (at rank 5, 25, 50, 75 for both factions), and may provide passive benefits, or such which scale with the player's current level.
   - The skills for the *Closers* are focused around the collection of more letters, whereas the skills for the *Openers* suggest a more liberal and efficient use of the *Ashery* and *Crematorium*.
   - There is also a "skills screen", which allows players to monitor the skills available for their Alignment, and any current bonuses for their unlocked skills.
2. **Brief summary of all other bonus features**
   - **Currency:** Ash, collected by completing words and collecting letters; and the conversion of letters from and to it accordingly, through the *Ashery* and *Crematorium* menus.
   - **Alignments** that provide a different set of benefits to players, as well as different names for the different Ranks. (The progression culminates in the *Eternal Prophet* rank for the *Closers*, and the *Dark Messiah* rank for the *Openers*.)
   - **Experience** which increases through collecting letters, and is responsible for the player's *rank* (level). Higher levels improve the player's Sight and Grab ranges, the amount of inventory slots at their disposal, and the rate of passive Ash generation. (Note that the experience amounts required to achieve various levels were calculated mathematically by exploring an ellipsoid segment.)
   - **Persistence**, currently done locally through the local device's *SharedPrefs*.
   - The game is rather light on mechanics as is, but I did my best to add at least some form of variety, as well as a set of goals for the players.

5. VCS

1. **BitBucket**
   - This project utilised a BitBucket repository for its version control needs.
   - Being the sole developer, I largely stuck to the master branch, with deviation done only to facilitate the ill-fated multiplayer component.
   - The repository may be found at https://bitbucket.org/lumosx/uoe_selp_grabble
   - Professor Gilmore has been granted Read access as required in the coursework.

6. Testing

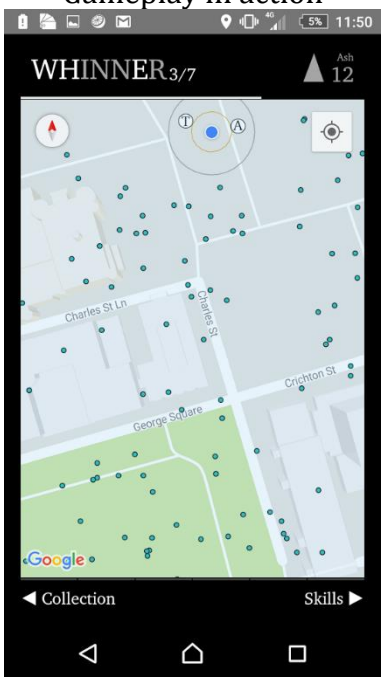
1. **Emulators**
   - Due to the curious inability of the default Android Studio emulator to handle GPS input, most of the testing was done on a Genymotion emulator running **Android 4.4** (API 19).
   - Any initial testing on the default emulator was also done on **Android 4.4** (API 19).
2. **Devices**
   - Perhaps unsurprisingly, my personal device also used **Android 4.4** (API 19).
   - User feedback and bug-finding has also been done, albeit to a much lesser extent, on a phone running **Android 6.0** (API 23).
   - Whilst the app should run fine, on any version from 19 onwards, proper testing has been conducted on API 19 only.
3. **Instrumented and unit tests**
   - To be completely honest, I've never had much use for unit tests, nor have I liked using them.
   - However, I did utilise them, albeit sparingly. Instrumented tests include a thorough verification of the behaviour of the "Registration" function when given bad data, as well as a basic test to verify the correctness of the algorithmically-generated experience amounts required for the progression system.
   - Regular unit tests include a basic test of the fitness of the segmentation system, and that's about it.

7. Design changes

1. **Assets used in the design**
   - Naturally, changes to the UI arose during the development process. The current product uses only two images as menu backgrounds – one made by me, the other CC0 (author unknown). All icons were made by me.
   - The fonts used are *"Khartiya Regular"* by Andrey Panov, and "*BukyVede Regular"* by Sebastian Kempgen.
2. **New screenshots** (from different accounts at different levels)

Registration screen

Gameplay in action



Inventory ("Collection")

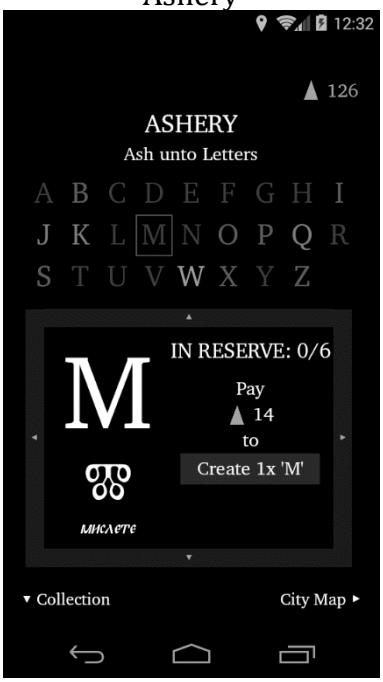
"The app design is very sleek and easy to use as well as being somewhat intuitive. The gameplay is simple but engaging enough to keep me occupied. The theme also appeals. I recommend this app."

David Wood
PhD student

"It keeps me occupied between lectures… It's even better than Yu-Gi-Oh! I like the minimalistic design. Managed to get over 1300 Ash already! I get to be an Opener just like Mara!"

Petar Vasilev
2nd year BEng student

Ashery



Skills overview