Software Engineering Large Practical

Part 3: Documentation

Name: Ioannis Baris, Matriculation Number: 1443483

1. Android Version and SDK Version specification: The application was developed and runs without problems in Android Version 4.4.2 (Api 19) and in Android Version (5.1) (Api 22). It presents various problems in versions above that, therefore it should only be tested in the specified versions. More specifically, I ran the app on a physical device with 4.4.2 version and on emulators with 4.4.2 version and 5.1 version.

2.1. Methods/Algorithms used in the Activities:

MainActivity

In the design document, MainActivity had the function of a Login Screen. However, after receiving feedback for the design document (part 2 submission), I modified its use to the one below, which was suggested as part of the feedback.
This activity starts when the user opens the app. In other words, it's the initial screen. Its layout consists of a background image and a button which, when clicked, directs the user to the loading screen (LoadingActivity). The "Start the game" button sends the intent to LoadingActivity only if the user's device is connected to the internet. Upon clicking the button while being disconnected from the internet, the user is asked to turn mobile networks on and if they refuse then they are asked to turn wi-fi on. This has been implemented in the form of dialog boxes. There is also music playing on this activity, using the MediaPlayer service.

LoadingActivity

No changes done from the design document (just more carefully explained here). This activity starts when the button in MainActivity is clicked. It functions as a Loading/Splash screen until the actual game begins. Its layout consists of a progress bar and a background image. The progress bas loads (is filled) from 0 to 100% for two seconds, while the next activity (MapsActivity) is created. When the progress bar becomes full, MapsActivity (the game's map) starts. I have used ObjectAnimator for the implementation of the progress bar.

MapsActivity

This activity has been enhanced with additional features that were not described in the design document. MapsActivity (the game's map) starts after the progress bar in the loading screen (LoadingActivity) becomes full. If the location services (GPS) is turned off, the user is asked, in the form of a dialog box, to turn it on.
Visible content on MapsActivity: The MapsActivity layout consists of a map fragment, 1 EditText, 3 buttons and 1 toggle button. The *EditText* receives an address as an input and by clicking the "Search" button, the user is directed to the searched address. The "search" function has been implemented with the *onSearch()* method, which receives an address in user-input text and finds the location of the address on the map using Geocoder, as an alternative to scrolling. The "Bag" button sends an intent (directs user) to GameActivity, when clicked. Two more buttons called "Auto" and "Type" were implemented additionally to

what was described in the design document. “Auto” button is a toggle button with “on” and “off” states. When in “on” state, the map zooms automatically to the user’s current location when the location changes (real-time tracking). When in “off” state, the map doesn’t zoom automatically to the user’s current location and it requires manual scrolling (this is useful when the user wants to search their surroundings). The “Type” button offers the user, in the form of a dialog box with single choice items, the option to change the type of the map. When clicked, it gives the user 4 choices: satellite, terrain, hybrid and normal. The map is a Google map, layered with markers. **The data structures** used are: 1) an arraylist of markers, called *markers*, which contains all the markers that exist on the map, 2) an arraylist of markers, called *markersVib*, which contains the markers which have not been involved in a vibration yet (will explain shortly).
Files in use: There are 2 files which are created in MapsActivity for the purpose of the game: 1) A “Date” file, in which the most recent day that the app was opened is saved, 2) A “latlngpoints.txt” file, in which the current (perhaps modified) markers on the map are saved.
Download and parsing of the daily Grabble letter map from the server: On map creation, the current date is saved in a *day* variable. If *day* is not equal with the value saved in the “Date” file, then the *checkDate(day)* method is called. The *checkDate(day)* method downloads the suitable Grabble letter map (kml file) depending on the value of day, e.g.

```
case Calendar.SUNDAY:
    input = new
URL("http://www.inf.ed.ac.uk/teaching/courses/selp/coursework/sunday.kml").
openStream();
    date = day;
    break;
```

The latest date is then saved to “Date” file. Moreover, a *kmlLayer* is created using the downloaded letter map, which is then parsed using the following algorithm: For each placemark in the *kmlLayer* (*kmlLayer.getPlacemarks()*), get the description and the name using the *placemark.getProperty()* method and get the coordinates by using the *placemark.getGeometry().getGeometryObject()* method together with arraylists and the *split()* method (in order to isolate the latitude and longitude values). Then, markers corresponding to the downloaded letter map’s placemarks are created and placed on the map. The markers are also added to the *markers* and *markersVib* arraylists. In case the value of *day* is the same as the value saved in the “Date” file, meaning that the user has already played the game during the same day, then instead of downloading the letter map, the “latlngpoints.txt” file is read and the saved markers are loaded to the map. On MapsActivity stopping, the markers are saved to the “latlngpoints.txt” file.
Efficient detection of the letters which can be grabbed at the user’s current location: All the markers on the map are clickable and the *onMarkerClick()* method is called when a marker is clicked. When clicked, their content (name and letter) is shown. The user is not able to collect a letter from a marker, unless they are connected to both the internet and the location services. The algorithm used for collecting a letter is the following: The clicked marker’s location is calculated by using the *marker.getPosition()* method and the *markerLocation.setLatitude()* and *markerLocation.setLongitude()* methods. Then, the user’s current location is compared against the clicked marker’s location and if their distance from the marker is within 15 meters, they can grab the letter in case the marker is not empty. Upon clicking a marker and having the capability of collecting the contained letter, the user is asked, in the form of a dialog box, if they want to do so. If they click the yes button, the user is directed to GameActivity (an intent to GameActivity is sent along with an Extra message which contains the collected letter) and a dialog box appears, stating that the user has collected the letter (will explain more in GameActivity section). Otherwise, nothing happens.

Moreover, a vibration feature has been implemented, as stated in the design document, in the *onLocationChanged()* method, which is called whenever the user's location changes. So, whenever the user's location changes, all markers contained in the *markersVib* arraylist are checked, and if any of them is within 15 meters from the user's location and it contains a letter, then the device vibrates for 500 ms (for each marker which completes these conditions) and the marker(s) involved in the vibration are removed from the arraylist (therefore the device vibrates only once for each marker so that the function doesn't become too intrusive). Music plays in this activity as well. Furthermore, if the user's location is out of the bounds where the game is playable, they receive a warning message, in the form of a dialog box. As bounds I used the coordinates that are on the coursework description.

GameActivity

This activity starts when: 1) User collects a letter, 2) User clicks the "Bag" button. When the activity is created, the *getIntent().getExtras()* method is checked. If its returning value is null, then the "Bag" button was clicked, otherwise the user collected a letter, and a message, in the form of a dialog box, stating it, is shown. The letter is also saved to the *letters* ArrayList. The activity's layout consists of 6 buttons, 1 *TextView* and 1 *EditText.* There are 4 buttons named as "Letters", "Words", "Scoreboard" and "Awards" which show the corresponding lists when clicked. There is 1 *EditText* which receives user-input text and 1 "Form word" button which initiates the word formation process. The *TextView* contains the number of the points achieved when a word is formed. Finally, the "Return" button results in GameActivity finishing and the user is directed to the previous activity, that is MapsActivity. The **Data Structures** used are 5 *ArrayLists<String>*: 1) *dictionary*, which contains the Grabble dictionary's words, 2) *letters*, which contains the collected letters, 3) *lettersTemp*, which contains the collected letters as well and is used for the process of forming words (will explain later), 4) *words*, which contains the formed words, 5) *awards*, which contains the achieved awards/titles. A *Hashmap<String,Integer>* called *letterVals* is also used for assigning values/points to the letters and the method *assignVals()* which assigns them is called when the activity is created.

Files in use: Several files are also created for the purpose of saving some values to the device's internal storage. The files are called "Letters", "Words", "Awards", "Total Letters", "Total Words", "Total Awards" and they contain the letters, words, awards, number of total letters, number of total words and number of total awards respectively. These files are read when GameActivity is created and the saved values are loaded to the ArrayLists. A method called *readDictionary()* is also called on the activity creation. This method reads the Grabble dictionary ("grabble.txt") which is in the assets folder and saves the words included in the dictionary to the *dictionary* ArrayList, after converting them to uppercase.

Efficient lookup of words in the Grabble Dictionary: When the user types a word in the *EditText* field and clicks the "Form word" button, the process begins. The algorithm works as following: The input text is converted to uppercase and 2 conditions are checked, one specifying that the word's length is exactly 7 (consists of 7 letters) and the other one that the word is contained in the *dictionary* ArrayList (checks words against dictionary).

```
if(word.length()==7 && dictionary.contains(word))
```

If both of the conditions are true, then a further condition is checked for each letter in the typed word to see if the letter has been collected by the user. In case all letters have been obtained, the word is formed and added to the *words* ArrayList, the current points deriving from the formed word are calculated and added to the total points as well as shown in the

TextView, the letters are removed from the *letters* ArrayList, that is from the list of collected letters, and the message, in the form of a dialog box, that a word was formed is shown. In case any of the letters in the typed word has not yet been collected by the user or in case the checked conditions are false, then a message indicating an invalid word is shown. The *lettersTemp* ArrayList is used as a supporting list, in the case that some of the letters included in the typed word have not been collected by the user (the letters are removed from *lettersTemp* until there is a letter not collected by the user; then the letters are added back on unless the word formation is successful). The *getLetterAward*() and *getWordAward()* methods are used for determining the awards applicable to the user. The first method is called when the user collects a letter, and in case the collected letter is the 10th, 25th, 50th, 100th, 250th, 500th or 1000th, then the user obtains a corresponding award/title which is saved to the *awards* ArrayList. The second method is called when the user forms a word, and in case the formed word is the 1st, 5th, 10th, 20th, 50th or 100th, then the user obtains a corresponding award/title which is also saved to the *awards* ArrayList. In this activity, too, there is music playing.

2.2. Parts of the Design which have not been realised in the Implementation:

- MainActivity with the function of a Login Screen. After receiving feedback for the design document (part 2 submission), I modified its use, as it was suggested as part of the feedback.

2.3. Additional features of the Implementation which were not described in the Design:

- "Auto" toggle button and "Type" button in MapsActivity.
- Warning message when user is out of bounds.
- Provided specifications about the GameActivity features (e.g. word formation), which were omitted in the Design document for no good reason.

3. Use of Version Control System to manage project source code:

I archived every part of my project, including all activities, classes and tests as well as layouts, in my version control system using Git. I have been managing a Git repository in BitBucket (https://bitbucket.org/JohnBaris/grabble/src , private-only readable by me and user stephengilmore) in which I pushed the commited changes of my source code in Android Studio. I used Version Control to keep log of the changes and be able to revert back in case I broke something.

4. Types of testing that I applied to my Implementation:

I created five instrumented tests for my application, which I placed under the androidTest folder. Three of them test the existence of *EditText* fields and buttons in the MainActivity, MapsActivity and GameActivity as well as the correct start of the next activity when a button is clicked. The corresponding tests are the MainActivityTest, MapActivityTest and GameActivityTest. The fourth one is an instrumented unit test called GameActivityUnitTest, which tests the following: 1) the method of checking if a word has exactly seven letters, 2) the method of checking if a word is contained in the dictionary, 3) the method of calculating the points. The last one is an interface automation test called MainActivityUITest, which simulates a scenario:

1) The app opens

2) (In MainActivity) The “Start game” button is clicked
3) LoadingActivity starts and then MapsActivity starts
4) (In MapsActivity) A marker is clicked.
5) (In GameActivity) Every button is clicked in sequence to show the existing lists.
6) (In GameActivity) Click the “Return” button.
7) (In MapsActivity) Click the “Bag” button. (In GameActivity) Click the “Return” button.
8) (In MapsActivity) Click the “Auto” button and then “Type” button and choose a different map type (terrain).
9) The end.

Attention:

- The interface test works only if the device or emulator is connected to the internet and the location services from the beginning (before the app starts). Also, if the device or emulator, where the test runs, is very slow, the test may not work as intended (because I used sleep methods).
- Running the MapsActivityTest interferes with the markers on the map (in the installed app), therefore run it in the end, only after you have looked into everything else, otherwise there will be a problem when you run the application (markers non-existent because of null value).

I tested all the tests on a physical device and on an emulator.