City Hunter
The smart city rally app

Christof Ferreira Torres
christof.ferreira.001@student.uni.lu

Angelo Migliosi
angelo.migliosi.001@student.uni.lu

A project documentation for the course of
Principles of Software Development

Document version 1.0

Project supervisors:

Jacques Klein `jacques.klein@uni.lu`
Tegawendé F. Bissyandé `tegawende.bissyande@uni.lu`

Faculty of Science, Technology and Communication
University of Luxembourg
Academic Year 2014 - 2015

UNIVERSITÉ DU LUXEMBOURG

Table of Contents

List of Figures

Chapter 1

Introduction

The main objective of this project was to develop an Android "City Rally" application, by reusing as much as possible the concepts, methods and tools presented in the Android course of Principles of Software Development. We named our application "City Hunter". City Hunter tries to present a city in a playful and funny way. The user is able to select a city from a list of cities or simply take the closest city near him. Once the user selected a city he has to solve one or more mysteries. A mystery consists in guessing a word related to a question concerning the particular city. However, to increase the chances in solving the given mystery and to increase his score, the user has to visit several checkpoints across the city which are nice spots of the city or famous squares/places. At each of these checkpoints, the user has to complete a challenge by answering a specific question or do a certain action with his device. If the user succeeds every challenge, then he/she is granted a clue which helps him to get closer to solve the particular mystery. Each of the challenges is illustrated on a map as well as the route to take to visit a particular challenge. The user can try to guess the answer of the mystery as often as he wants. However, the potential score that he can get decreases for each try he guesses wrong, starting at a maximum score of 500 points and going down to a minimum score of 100 points. Similar for the challenges, except that each challenge has a maximum number of tries. For each wrong try, the score is decreased by 100 points. The maximum score of a challenge depends on the maximum number of tries that user has available for a particular challenge. For example, a challenge with 2 tries has a maximum score of 200 points, whereas a challenge with 3 tries is more difficult and has also therefore 300 points. Once the user has consumed all of his tries and he still didn't succeed the challenge, he loses the challenge and is not allowed to play the challenge anymore.

Chapter 2

Persistence

The persistence of our application is divided into persistent data and dynamic data. The persistent data is not modified by the user and can only be read which makes it easy to be later updated and fetched from a server. The dynamic data is frequently modified by the user and contains more user related information, such as his progress or his current total score.

2.1 Persistent data

The persistent data is stored as a JSON file inside the assets folder of our application. We decided to use JSON as data format because it's nowadays very popular, easy to read and has a short notation which minimizes the file size for transmission and storage. The JSON file is read at the launch of the application and parsed in order to create the objects needed for our application. In Figure 1 we can see the structure of our JSON file. It's a tree structure where the array cities is the root and contains several city objects. Each city object contains city related properties but also an array mysteries which contains several mystery objects. Finally, each mystery object contains mystery related properties but also an array challenges which contains several challenge objects.

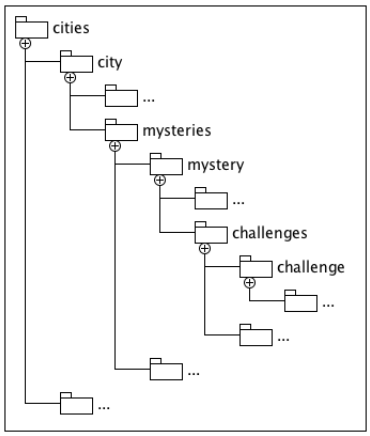


Figure 1: Diagram showing the structure of the persistent_data.json file.

Figure 2 shows the UML diagram of the different classes and relationships concerning the persistent data. The class Home contains 0 or more cities. The class City contains 0 or more mysteries. The class Mystery contains 0 or more challenges. The class Challenge is an abstract class where the classes QuestionChallenge and CompassChallenge inherit from. Finally, ChallengeType, ChallengeState and MysteryState are enumeration types used across our application.

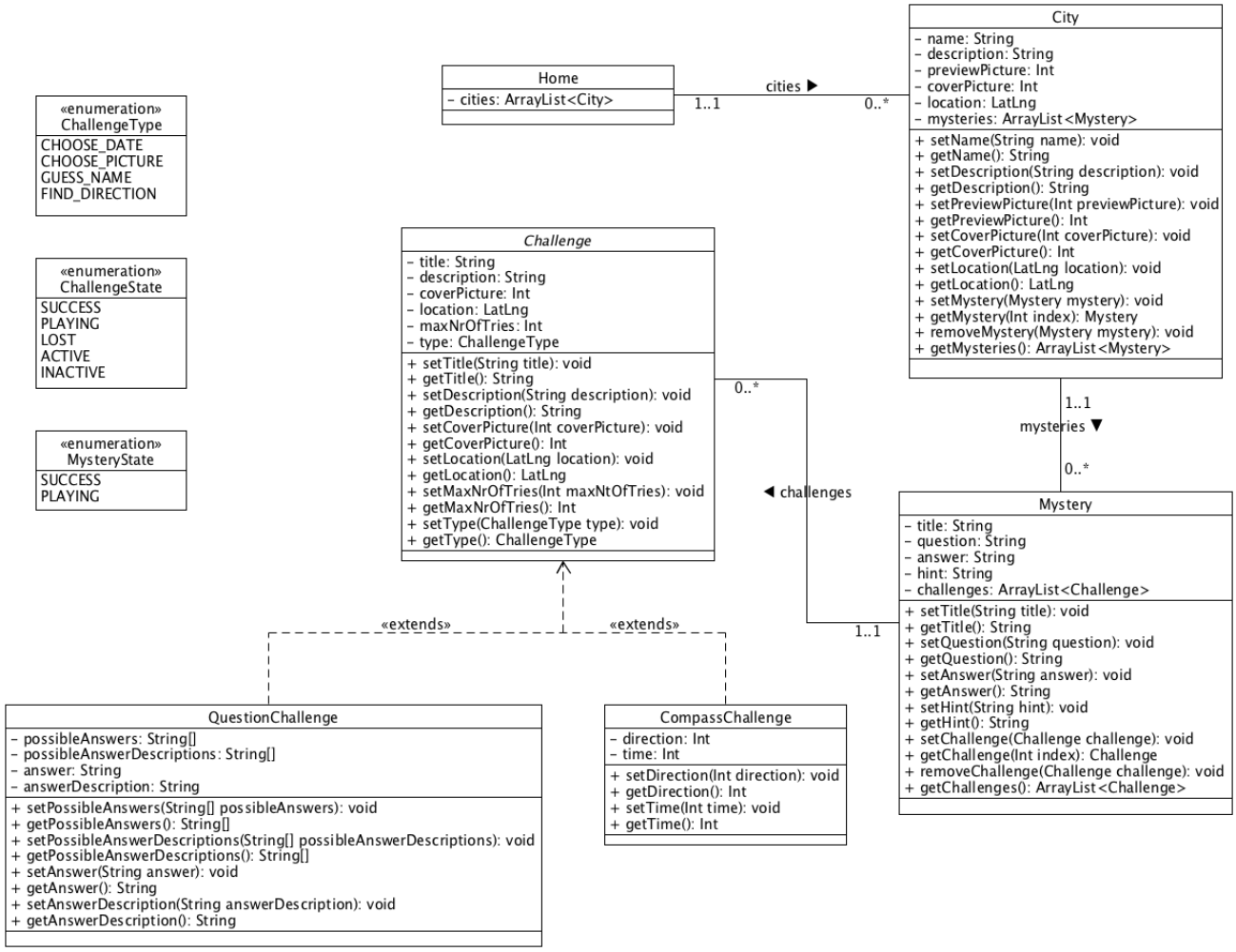


Figure 2: UML diagram showing the different class and relationships concerning the persistent data.

2.2 Dynamic data

The dynamic data is stored as shared preferences. Shared preferences provide an easy interface to retrieve and modify primitive type values. Android saves shared preferences internally as files and takes care of their management. They are the perfect choice for storing settings and user related information. Our application uses the following shared preferences:

- GELLE_FRA_PREFERENCES
- CATHEDRAL_PREFERENCES
- PALAIS_PREFERENCES
- PLACE_GUILLAUME_PREFERENCES
- PLACE_CLAIREFONTAINE_PREFERENCES
- MYSTERY_1_PREFERENCES
- SCORE_PREFERENCES

We store as shared preferences the progress of each challenge and mystery, but also the number of tries of each challenge and mystery. Depending on the challenge, we also store specific information about the current state of already selected buttons, pictures or words entered. The current total score of the user is also stored as a shared preference. The 'Settings' activity allows a user to delete all shared preferences and reset the state of the application.

Chapter 3

Activities

In this chapter we describe the different activities that are used in our Android application. Figure 3 shows the life cycle of all our activities and provides a global view of the structure of our application. The 'Home' activity is our main activity. This is the first activity that is started when our application is launched. The user can then start further activities from this activity, such as the 'City' activity or the 'All Cities' activity. Notice that the 'City' activity can be started from the 'Home' activity but also from the 'All Cities' activity. Afterwards, the user can start from the 'City' activity the 'Mystery Tab' activity which includes the 'Mystery Map' activity and the 'Mystery Info' activity. The 'Mystery Map' activity allows the user to start different 'Challenge' activities, such as the 'Choose Date' activity, the 'Choose Picture' activity, the 'Guess Name' activity or the 'Find Direction' activity. Finally each activity, except of the individual challenge activities, has access to a menu which includes the 'Settings' activity, 'Help' activity and the 'About' activity.

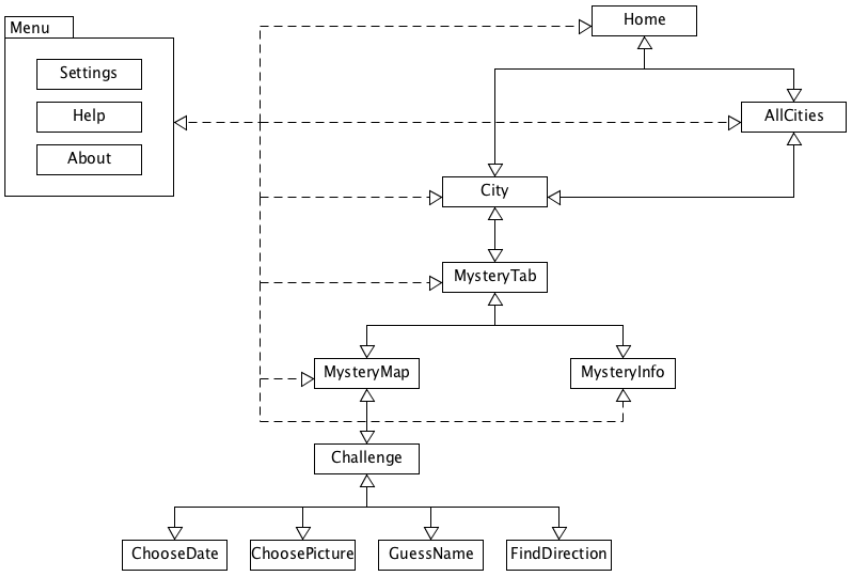


Figure 3: Life cycle of all activities.

3.1 Home Activity

The 'Home' activity is our main activity. This is the first activity that the user sees and that is started at the launch of our application. It loads the JSON persistent data and creates the necessary data structure. It also displays the total score that the user has achieved so far. It provides an 'All Cities' button, which allows a user to start the 'All Cities' activity. However, based on the user's current location it also able to suggest to the user the closest city near by.

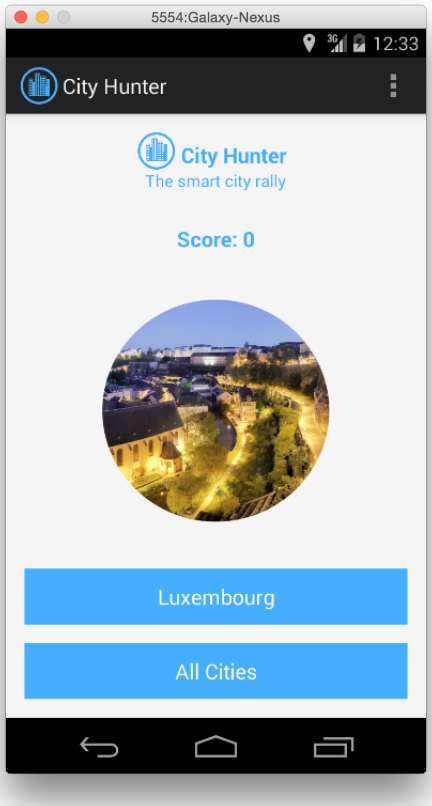


Figure 4: A snapshot of the Home Activity.

3.2 All Cities Activity

The 'All Cities' activity shows all available cities in a grid list. The user can select an individual city out of this list to start the 'City' activity which displays more details about the selected city.

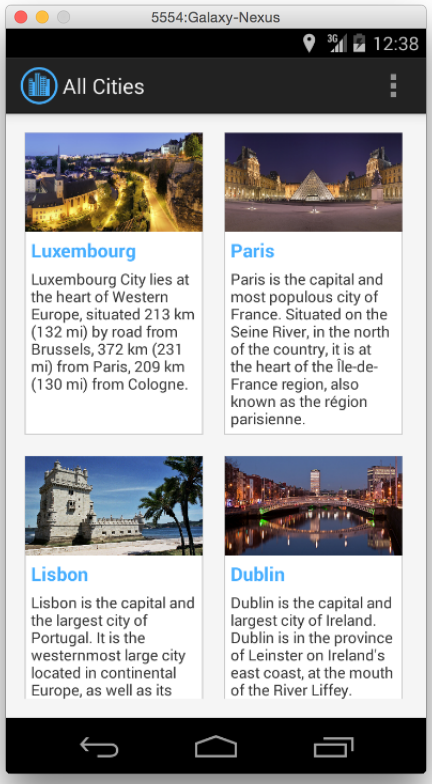


Figure 5: A snapshot of the All Cities Activity.

3.3 City Activity

The 'City' activity shows all available mysteries for a particular city in a grid list. The user can select an individual mystery out of this list to start the 'Mystery Tab' activity.

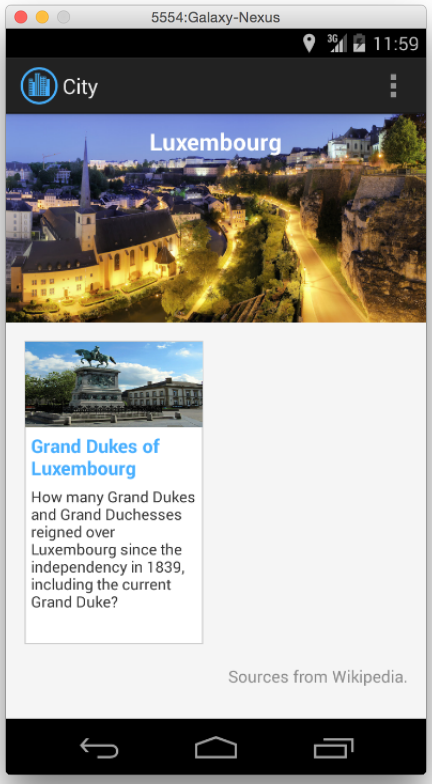


Figure 6: A snapshot of the City Activity.

3.4 Mystery Map Activity

The 'Mystery Map' activity is the first tab of the 'Mystery Tab' activity. It shows a map with all the available challenges. The map also contains a button to switch to fullscreen and a button to route the user from his current location to the closest challenge. Below the map is a list of all available challenges. Challenges are normally inactive and only become active if the user's current position is close to a challenge. Each inactive challenge has a button which when pressed shows the route from the user's current location to the particular challenge. Active challenges can be clicked on to start the particular 'Challenge' activity.

Figure 7: A snapshot of the Mystery Map Activity.

3.5 Mystery Info Activity

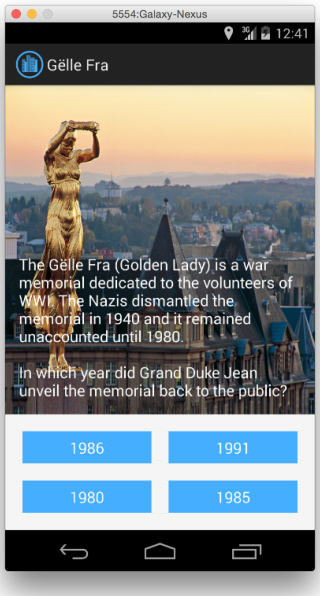
The 'Mystery Info' activity is the second tab of the 'Mystery Tab' activity. It shows the title and the question of the mystery that the user has to solve. If the user was able to successfully complete all the challenges, then also a hint is displayed to help the user solving the mystery. Finally, the activity also provides to the user a way to enter his answer for the mystery.

Figure 8: A snapshot of the Mystery Info Activity.

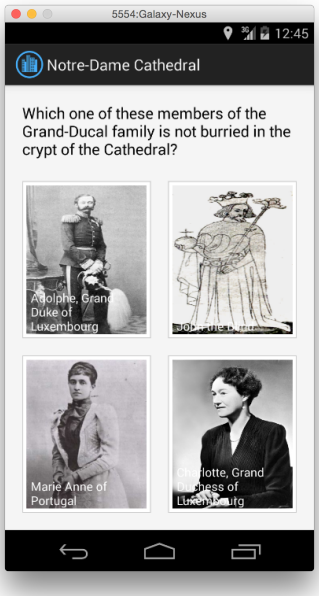
3.6 Challenge Activity

The 'Challenge' activity loads individual challenge activities relative to the selected type of challenge in the 'Mystery Map' activity. Possible challenge activities are:

- Choose Date Activity
- Choose Picture Activity
- Guess Name Activity
- Find Direction Activity

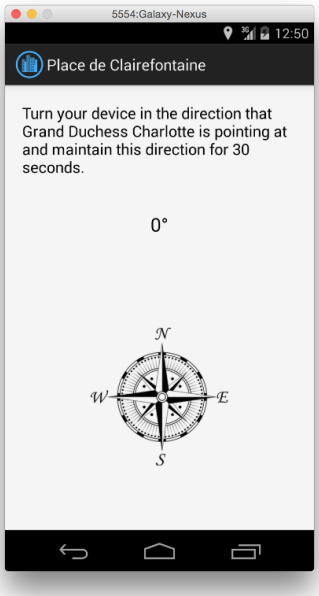


(a) Choose Date Activity



(b) Choose Picture Activity

(c) Guess Name Activity



(d) Find Direction Activity

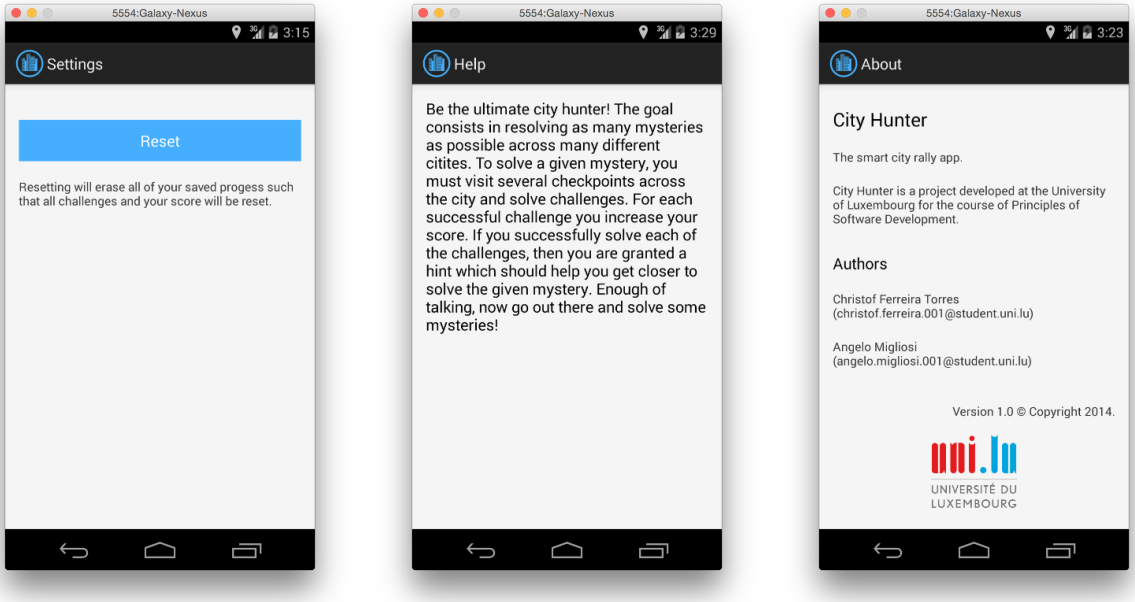
Figure 9: Snapshots of the possible challenge activities.

3.7 Menu Activity

Each activity, except of the challenge activities, has access to a menu which provides a selection of the following three activities:

- Settings Activity
- Help Activity
- About Activity

The 'Settings' activity provides a button to reset all saved data, including the user's progress and his current score. The 'Help' activity provides a description of the application in order to help the user better understand how the application works. The 'About' activity includes a small description and information about the copyright and the authors of the application.



(a) Settings Activity (b) Help Activity (c) About Activity

Figure 10: Snapshots of the three menu activities.

Chapter 4

Challenges

As already mentioned in the previous chapters, there are two different challenge classes in our application, namely 'QuestionChallenge' and 'CompassChallenge'. The 'QuestionChallenge' class represents a challenge where a user needs to find an answer to a specific question. The 'CompassChallenge' class represents a challenge where a user needs to interact with the compass of the device. Each of these challenge classes inherits from the abstract class 'Challenge'. This means, each challenge has a title, a description, a location, a maximum number of tries and a challenge type. Possible challenge types are:

- Choose Date
- Choose Picture
- Guess Name
- Find Direction

4.1 Choose Date

The 'Choose Date' challenge is a question challenge. The user gets four possible dates as answer. Depending on the description of the challenge, the user has to select the right date or the wrong date out of these four dates. The user has normally two tries for this challenge.

4.2 Choose Picture

The 'Choose Picture' challenge is a question challenge. The user sees four different pictures. Depending on the description of the challenge, the user has to select the right picture or the wrong picture out of the four pictures. The user has normally two tries for this challenge.

4.3 Guess Name

The 'Guess Name' challenge is a question challenge. The user sees a picture and depending on the description of the challenge, the user needs to enter the name of the place or person on the picture. Since this is a more difficult challenge, the user has normally three tries for this challenge.

4.4 Find Direction

The 'Find Direction' challenge is a compass challenge. The user needs to hold his device in a certain direction for a certain amount of time. The direction depends on the description of the challenge. Since this is a very difficult challenge, the user has an unlimited number of tries for this challenge.

Chapter 5

Test Cases

In this chapter we will provide 2 test cases describing the steps to:

1. Run successfully through all challenges and get the hint to the mystery.
2. Run unsuccessfully through several challenges and see the different states of the challenges.

The correct answers and the number of tries will be shown between brackets in the test case descriptions.

We used "mock locations" in order to simulate a run through the city. Hence it is important to tick the "*Allow Mock Locations*" in the developer options of your device.

Furthermore, we only use mock locations in the Mystery Map Activity. A click on a route button, as shown in Figure 11, will automatically launch a thread which will feed the location service of the device with the location points of the direction. In order to avoid inconsistencies on the map, no new thread will be started when another one is still running.

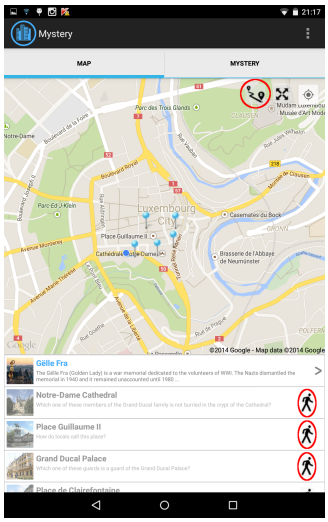


Figure 11: Route buttons in Mystery Map Activity

After the installation of the City Hunter App the following test cases can be performed:

5.1 Test Case 1: A Successful Run

1. Start the City Hunter App.
2. Click on the "*All Cities*" button.
3. Choose "*Luxembourg*".
4. Choose the "*Grand Dukes of Luxembourg*" mystery.
5. Wait until the current-location-marker (CLM) arrived at the first challenge "*Gëlle Fra*". In the list view, the challenge changed the state to active. Click on it.
6. Choose the right answer. (*1985, 2 tries*)
7. Click on the "*Next Challenge*" button as shown in Figure 12, and wait until the CLM arrived.

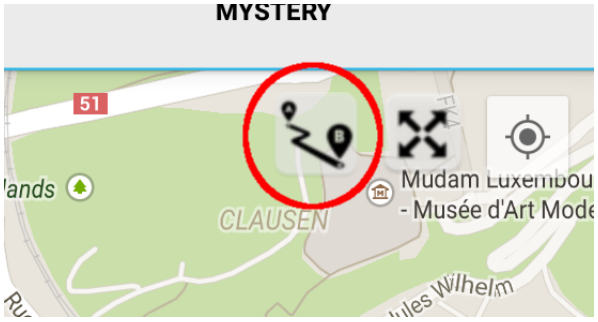


Figure 12: Route to the next challenge in Mystery Map Activity

8. Click on the "*Notre-Dame Cathedral*" challenge and choose the correct answer. (*Adolphe, 2 tries*)
9. In the list view click on the "*Place de Clairefontaine*" challenge and wait until the CLM arrived.
10. Click on the "*Place de Clairefontaine*" challenge and do the challenge. (*Hold your device between 100 ° and 120 °, infinite tries*)
11. In the list view, click on the "*Grand Ducal Palace*" challenge and wait until the CLM arrived.
12. Click on the "*Grand Ducal Palace*" challenge and choose the correct picture. (*Guard in the sun, 2 tries*)
13. In the list view, click on the "*Place Guillaume II*" challenge and wait until the CLM arrived.
14. Click on the "*Place Guillaume II*" challenge and enter the correct answer. (*Knuedler, 3 tries*)
15. Click on the "*Mystery*" Tab and solve the mystery. (*9, infinite tries*)

5.2 Test Case 2: An Unsuccessful Run

1. Start the City Hunter App.
2. Click on the "*Option*" button as shown in Figure 13 and choose "*Settings*".

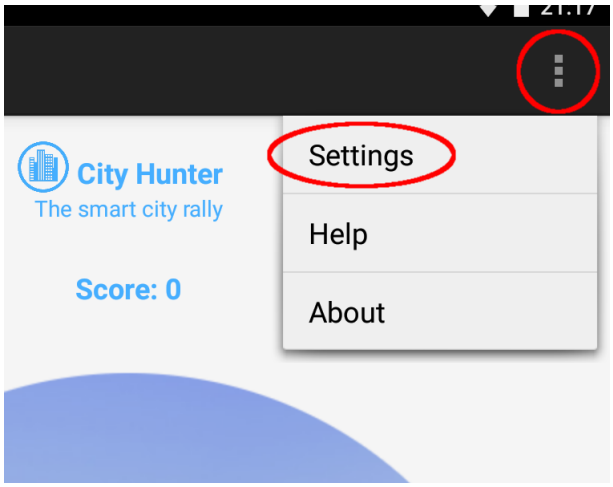


Figure 13: Option Button

3. Reset the game progress.
4. Go back to the "*Home*" Activity
5. Click on the "*All Cities*" button.
6. Choose "*Luxembourg*".
7. Choose the "*Grand Dukes of Luxembourg*" mystery.
8. Wait until the current-location-marker (CLM) arrived at the first challenge "*Gëlle Fra*".
9. Click on the "*Gëlle Fra*" challenge and choose 2 wrong answers. (*1980, 1991*)
10. In the list view, click on the "*Place Guillaume II*" challenge and wait until the CLM arrived.
11. Click on the "*Place Guillaume II*" challenge and enter 3 times a wrong answer.
12. In the list view click on the "*Place de Clairefontaine*" challenge and wait until the CLM arrived.
13. Click on the "*Place de Clairefontaine*" challenge and succeed the challenge. (*Hold your device between 100 ° and 120 °, infinite tries*)
14. In the list view click on the "*Notre-Dame Cathedral*" challenge and wait until the CLM arrived.

15. Click on the "*Notre-Dame Cathedral*" challenge and choose the correct answer. (*Adolphe, 2 tries*)
16. In the list view, click on the "*Grand Ducal Palace*" challenge and wait until the CLM arrived.
17. Click on the "*Grand Ducal Palace*" challenge and choose 2 wrong pictures. (*Swiss guard, British guard*)
18. Click on the "*Mystery*" Tab and solve the mystery. (*9, infinite tries*)

Chapter 6

Conclusion

6.1 Future Work

There are clearly some limitations inside of our application and aspects which could be improved. Creating static shared preferences for each challenge and mystery is not a long term solution. We like the idea of saving the user related data as shared preferences. However, it is not so trivial to create shared preferences on the run and creating shared preferences for each challenge and mystery also seems a bit of an overhead, but due to the timing constraints we couldn't elaborate a better solution than just creating static shared preferences. Furthermore, we decided to use JSON as data format for the persistent data, because we wanted to have the future possibility to update the cities, challenges and mysteries easily over the network from a server. A final improvement would also be creating more different types of challenges, so not only question challenges or compass challenges, but also challenges which maybe make use of the camera of the device.

6.2 Final Words

The City Hunter app provides a fun way to learn about cities and their history. We designed the code to be flexible and easily extendible. The challenges, mysteries and cities can be extended by simply modifying the persistent JSON file. Furthermore, the hierarchical structure of challenges enables developers to introduce new challenge types without changing too much code. We introduced a lot of Android concepts, such as intents and intent services, location services, shared preferences, sensors, animations, ... and mock locations in order to test the application.

Altogether, the project was a lot of work and gave us a deep insight into Android. During the development we encountered several issues, which showed us that there is still a lot of work to be done in order to improve the developer experience in this domain. For example, we used the official Android Developer Tools (ADT) and there is a big issue with the Android emulator, which runs very slowly or, after installing some software to speed it up, does not really support every feature, e.g. Google Play Services, simulation of sensors, etc. Some code of Android is not clearly marked as deprecated and often you only find tutorials with deprecated and sometimes even wrong code in the official Android documentation, e.g. *Geofences*. Furthermore, some code, supposed to replace deprecated code, does not behave as described in the official documentation,

see *GeofencingEvent*.

All in all, we struggled a lot, since our code rarely worked right away. Nevertheless, we learnt a lot about the possibilities of Android and the development of a mobile application.