University of Glasgow | School of Computing Science

Level 3 Project Case Study Dissertation

An Example Project

Jordyn Anne Brown
Nathan Gordon Kirkpatrick
Muhammad Raza Ali
Scott Brown
Josep Perna Montane
Ruairi Gielty

6 April 2020

Abstract

The abstract goes here

Education Use Consent

We hereby give our permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format.

1 Introduction

Software engineering

This paper presents a case study of...

The rest of the case study is structured as follows. Section **??** presents the background of the case study discussed, describing the customer and project context, aims and objectives and project state at the time of writing. Sections **??** through Section 10 discuss issues that arose during the project...

2 Case Study Background

Include details of

- The customer organisation and background.
- The rationale and initial objectives for the project.
- The final software was delivered for the customer.

3 QR Code Implementation

On the second customer meeting, Obashi told us that we should scatter the cubes in a wider area. The research done and different approaches taken for solving this problem are explained in the Cube Scattering section.

After doing research and trying the different approaches, we decided to use a QR code that would provide our application with the size of the room so that cubes could be scattered accordingly. To do this, when starting the application we place the QR code in the centre of the room and scan it before entering the AR Activity.

This left us with the need of implementing a QR reader in our application. Being this something that is implemented in many apps we decided to use ZXing[**?**], an existing QR scanning library. Implementing it was pretty straightforward thanks to all of the documentation provided in the repository. The only blocker that we found was that we had find a way of passing the measures retrieved from the QR scanner to the AR view. We used the extra field of the intent to do this. The extra field, contains a dictionary in which we could store the measures and then they could be retrieved in the AR activity and used to scatter the cubes.

4 Cube Scattering

Initially when it came to scattering the cubes, we used a simple algorithm to calculate a random position on the plane detected by ARCore. The functions `getExtentX()` and `getExtentY()` in the `getRandomPlacement` method in the `ArUtils.java` file return the estimated lengths for each local axis centred on the plane, therefore this must be doubled for both `+` and `-`. We then later make use of a translation of the center of the plane to create an anchor node which we then assign the cube to later on. However with this basic implementation we encountered two serious blockers.

The first major blocker was that cubes were sometimes being placed inside each other, a simple fix we devised for this was to keep a list of pairs of all cubes and their anchor nodes positions and whenever we needed to place a new cube we would iterate over this list and check whether or not our new position was within a minimum distance of any other cubes and either generate a new position and check again or place it.

The second major blocker that we encountered was that not all the cubes we desired to be placed were being placed, we first assumed this was due to the distance we were trying to maintain between all the cubes but later discovered this was due to the plane not being large enough to fit all the cubes in. For the plane detected by ARCore to grow, the user needed to pan the camera and ARCore needed to detect that the plane can be expanded - this can therefore create a situation were the user is searching for the last cubes which have not been placed due to the plane not being large enough but the user is panning in such a way that ARCore is not recognising the plane and the game being stuck in a limbo state. We overcame this issue with the implementation of the QR Scanner (and later manual integer input) which lets us negate the need for checking the plane extents and simply use a `distanceToPlace` variable which receives an integer in the form of the radius of the room from the scanner or manual input to determine the dimensions of the room to scatter accordingly. This implementation also allows us to scatter all cubes instantly as we don't need to wait for ARCore to grow the size of the plane.

When it came to assigning elements from the diagram to the cubes we decided to keep a list of the elements from the diagram in a list `listToStack` which we would then use to create a stack of all the elements, we would then use `Collections.shuffle` to randomise the order of the stack and then whenever a cube was needing to be placed and the position was okay we could pop an item from the stack and assign it with the number of cubes needing to be placed coming from the difficulty selection on the main screen.

A design decision we made further into development after the 4^{th} customer day was instead of having the user collect cubes with the diagram element text being displayed directly on them, we would instead make use of what we called an `infocard` which would display the text and feature a button for the user to tap on for them to collect the element and add it to their inventory. This was done to increase the game aspect of the application and therefore we decided that these cards would only be displayed once the user had found the cube and tapped on it and the element could only be collected if the user was within a certain distance of the cube - this distance is also

determined by the difficulty selection on the main screen.

When it came to the design of the cubes we wanted to keep it simplistic and still keep it challenging for the user to find so therefore we opted to assign each cube with a random RGB value as well as a random transparency and size within a set range, this can be found within the `createCube()` method in the `ArActivity.java` file.

5 Testing

When we set out to start writing tests for our application, we researched testing frameworks for AR applications but we realized that none were available. This is because testing an augmented reality application is quite a difficult task, due to the unpredictability and volatility of AR environments. Another big factor, is that AR environments are hugely "continuous", it is very difficult to reach 100% test coverage when there are an almost infinite amount of states that an environment can be in. Moreover, AR applications are quite a new thing and a testing environment for them still hasn't had time to be developed[**?**].

We conducted two distinct types of tests: instrumented tests and unit tests. In Android, a unit test is just a regular JUnit test which runs on the hardware that an application is written on using the JVM or in a continuous integration environment. An instrumented test on the other hand, runs on a physical device or a device emulator which the app is designed for[**?**]. We tried to use unit tests in all of the cases possible, as they could be executed in the CI/CD pipeline.

In some cases such as the ArActivity, JUnit tests could not be used due to the Activity implementing a check for the version of OpenGL of the device that would close the application if it failed (we were not able to bypass this tests using mocking frameworks).

5.1 Unit Testing

After a lot of research, some frameworks were found, which helped build unit test. Mockito is a "unit based mocking framework"[**?**] which gave us an excellent alternative to simulation of an AR environment for testing purposes. Mockito was invaluable in the cases where we wanted to test methods which are called in an AR environment, and operate on data from this environment.

Instead of actually simulating an environment for our app to interact with, Mockito gave us the functionality to create "mock objects". A mock object is a dummy implementation of a class, where the output of its methods can be defined by us, the testers.

One example of our use of Mockito was when testing some methods that used the plane's dimensions, to scatter cubes. According to Google's ARCore developer ref-

erences, a plane object "describes the current best knowledge of a real-world planar surface"[**?**]. Thus, a plane is an object which describes the plane that our app operates on. As it was unfeasible to create a plane using the provided AR Core API, we used Mockito to create a mock plane object that yielded the characteristics we needed for the tests[**?**].

This mock plane hugely simplified the testing process for any method which operated on a plane object, and also meant that tests were more consistent as we had the ability to define what is returned by Google's ARCore methods; methods which are otherwise volatile.

We also used Robolectric, which is a powerful alternative to conducting tests instrumentally (on an android device or emulator)[**?**]. We used it to test application interactions such as intent passing (the construct used to move between activities and pass information between them). These require the android environment to be configured in a sandbox in order have the appropriate context for the methods to operate on.

Using Robolectric allowed us to add these tests to the CI/CD pipeline and remove the manual work required to run instrumented tests directly on the device or the emulator.

5.2 Instrumented Testing

We had to test two activities using instrumented testing, due to the constraints stated above. For it we used Espresso, an Android testing framework specially designed for UI testing. With it we were able to capture intents and analyze their correctness[**?**].

6 The Overlay

The Overlay was an essential part of the application. It was needed to display various computational elements in their respective categories on the Obashi diagram. This was used when a user collected some cubes in the AR environment and then placed each cube in the correct category on the diagram.

This required thorough research and gathering of ideas from the whole team, in order to decide on the implementation. The implementation of the Overlay required many different approaches and subsequently testing each approach in Android Studio. This was necessary in order to find the most practical and visual appealing result.

The initial idea was displaying a pop up window after clicking a button [**?**]. Once clicked, a new layout would appear displaying the Obashi diagram, with the different categories as a dropdown list[**?**]. This approach was simple and straightforward, but lacked visual appeal along with some functionality. The issue here was once the cubes are placed in the correct category, the user would not be able to see them. To be able

to visualize the collected elements in each category another layout would be needed, or the cubes would have to be stored internally in some data structure. Storing them internally would not allow the user to see the cubes on the screen after they have placed them. This initial idea was the same idea as presented in the wireframes.

To combat this problem, another approach was used, which was based on moving from the AR Activity to a new Overlay Activity[**?**] and displaying the Obashi diagram inside the Overlay Activity. The full diagram was to be shown and the user, would be able to scroll in any direction and zoom in and out of the diagram [**?**]. After implementing this idea, the team soon realized that, again this was not the most appealing design, since the full diagram on the screen was too small and the user would have to zoom in into each category to visualize the cubes. The transitioning from Overlay activity back to AR Activity was also not pleasant, as it would not save the cubes in the Overlay activity once exited. During the client meeting day 4, the customer made it clear that they preferred the overlay to be in the AR environment.

To overcome this issue, the team had to research more about, how to implement an Overlay inside the AR environment. The customers wanted to see the Overlay completed and working for the customer day 5. The solution found was fragments in android, which is an independent component that can be used by an activity. Multiple fragments can be combined in one activity, to build a multi-pane User Interface and reuse a fragment in multiple activities[**?**]. This allowed an overlay to pop up, after clicking a button inside the AR environment, with its own lifecycle and functionality. Customer day 5 approached and the overlay was not completed. Most of the time was spent getting familiar with how fragments work in android and fixing small bugs. For example, the overlay fragment was not displaying properly in AR or the overlay was not closing once clicked. The clients were still very happy with the progress, as the other functionalities were completed and working. After explaining the situation to the clients regarding the overlay, they were very understanding and wanted the Overlay completed for the final presentation.

After the client meeting the main focus was the Overlay. The Overlay fragment has seven different categories from the Obashi diagram and each category has its own recyclerview[**?**]. These recyclerviews are created in the method inititateRecyclerViews in the file OverlayFragment.java, the ids[**?**] are defined in the overlay fragment xml file for each recyclerview. Each recycler view had their set of Cardviews displaying the text of the cubes for the diagram. Different colours were used for each category in the diagram. The layout of the overlay was now completed, but the biggest hurdle was the linking of the Inventory and Overlay, in particular the placing of cubes from the inventory to the diagram and keeping track of the cubes. This problem took quite some time and effort by the team. The problem was overcome eventually with the following approach. Once a player chose a level, the elements which were scattered, appeared as an EMPTY box in the diagram. The other elements remained visible to the user. The scattered elements must first be collected in the inventory and then the player must select an element to place in the diagram. Once the player places the element in the correct position the EMPTY box is filled with the cube selected. This was possible due to the tag[**?**] every element has, whether its scattered or not. The element in the inventory is also removed upon placing each cube in the diagram. All of

this was achieved in the onInterceptTouchEvent method in the OverlayFragment.java file and the onBindViewHolder method in the OverlayRecyclerViewAdapter.java.

7 Choice of Colours

The following diagrams (especially figure **??**) illustrate the process...

8 Managing Dress Sense

In this chapter, we describe how the implemented the system.

9 Kangaroo Practices

10 Knots and Bundles

11 Conclusions

Explain the wider lessons that you learned about software engineering, based on the specific issues discussed in previous sections. Reflect on the extent to which these lessons could be generalised to other types of software project. Relate the wider lessons to others reported in case studies in the software engineering literature.