

Assignment 1, AR198066

by Ashley Reeves

FILE	15445_ASHLEY_REEVES_ASSIGNMENT_1__AR198066_164031_1044617233.DOCX		
TIME SUBMITTED	21-DEC-2018 11:24AM (UTC+0000)	WORD COUNT	4533
SUBMISSION ID	97892069	CHARACTER COUNT	26990

Coleg Sir Gar

Trinity Saint David

University of Wales

Daniel Davies

Application Development

Design Patterns

DUM5002

Student number AR198066

Ashley G. Reeves

12-20-2018

Table of contents

Contents

Introduction	2
software patterns in the development of mobile applications	3
1.1 Future Trends.....	3
MVC (Model View Controller).....	5
Model	5
View	5
Controller	6
2.0 Facade Pattern	6
Summary	8
3.0 Investigation of MVVM pattern	9
Why use MVVM?	9
View	10
ViewModel	10
Model	11
Summary of MVVM / Recommendations	12
Exercise 1.	13
Summary	15
Exercise 2.	16
Summary	18
Exercise 3.	19
Summary	23
Total Word Count: 3382	23
Bibliography	24

Introduction

This report will be broken into three sections. Part 1.0 will be an investigation into design patterns that are used within mobile applications and future trends will be discussed. Model View Controller architecture (MVC) and the Facade design pattern will be discussed within this report and will outline the different components and their uses.

Part 2.0 of the report will outline the benefits and drawbacks of Model View ViewModel (MVVM) pattern when developing mobile applications.

Part 3.0 will demonstrate numerous exercises based on mobile application development and their testing to assess whether they meet the user requirements.

software patterns in the development of mobile applications

The aim of a mobile developer is not just to achieve functionality and usability, but to balance these issues with performance and efficiency. These issues must be balanced with what a user is expecting and in recent years' mobile applications have become faster and more reliable, [1, p. 792].

High-quality design not only stops a user from being easily uninterested with an application but can also affect crashes and information security, so it is important that design patterns are considered while at the development stage of the project, [2].

One of the main factors with application development is the fact that there are a substantial number of different types of devices which need to be catered for with their own Operating System (OS), display resolutions, and processor speeds, all which need to be taken into consideration when choosing a design pattern for an application. Another constraint with mobile technology is the limitation on the resources available within a mobile device. For example, a piece of software specifically designed for a desktop cannot be ported over to an iPhone or an Android device, because of the limitations on the available resources (Mobile device), [1].

Another constraint is the limited data allowance on a mobile device. Many modern applications often connect to the cloud during the day to perform authentication tasks or to download or upload data. This can be for many reasons such as updates or even to query the user's current location. Not only does this affect the user's data allowance, it also affects the user's battery life, [3]. Another reason for a mobile device to regularly send and receive data is to allow the collection and correlation of data which may allow a developer to monitor trends for many purposes such as targeted marketing or to produce BigData. Big Data is a term used to describe the large volume of data that can be collected to help predict human behavior, the Big Data can be used to produce analytics to help forecast many recurring or hidden patterns often used to produce targeted advertising, but Big Data has many other uses, [4].

Location-based Services (LBS) is one of the fastest growing types of apps in 2018, [5]. These allow notifications based upon your location such as information on interesting sites near you, or it gives you the ability to track other users in real time, [6].

1.1 Future Trends

One of the leading trends in 2018 has been the constant development of Artificial Intelligence (AI) and this is set to continue into 2019. The term AI is a diverse term that covers many future trends such as automated chat bots that are increasingly being used within some of the world's fastest-growing client bases such as Starbuck and MasterCard, [7].

MasterCard is allowing the users who have downloaded their apps to easily and quickly check bank accounts using voice recognition software with the help of AI software. This allows the user to generate reports on a variety of topics such as total monthly expenditure or specific outgoings, [8].

Starbucks has used chatbots to allow customers to order their favorite drinks before they enter the store, [9]. The customers can either use text messaging or voice commands to accomplish this, which has reduced the queue time in many of the outlets who are trialing this ability, [10].

Augmented Reality (AR) has been labeled as “the next big thing” in technology groups for 2019, one of the driving factors behind this is several large retailers backing the concept, [11]. Ikea recently announced they were working with Apple and using the Apple ARKit to produce their Ikea place app, this app allows customers to place furniture within their homes before they buy it, [12].

The Wearables market has had a 5.5% growth in the United States between 2017 – 2018. Forecasts suggest the market will grow 6.2% from 2018 – 2019, [13]. With the market for wearables set to increase with the emergence of more manufacturers now seeing the gains to be made in this sector, and with 71% of 16 to 24-year old now wearing wearables, this trend is set to increase. Wearables have evolved from the Bluetooth headset that was released in 2002. Since then Nike, Fitbit, and Google have had successes with their own versions of wearable technology such as the Google Glass Explorer or Fitbit fitness tracker, [14], [15].

The future of wearable technology is focusing on the transition from basic to smart wearables and this has many opportunities from healthcare with the development of wearable sensor-based systems to monitor a person’s health and help doctors reach a more accurate diagnosis [16].

MVC (Model View Controller)

Model View Controller (MVC) is an architecture pattern that allows a developer to separate an application into three components: Model, View, and the Controller. Each component is responsible for a specific set of aspects of an application, [17]. By separating these components, it allows the whole application to become more modular, lightweight and testable. This allows certain parts of the application to be used between projects, [18]. For instance, if the developer had previously designed and built a main menu class, it is possible to implement that class within another project which reduces the overall work of the developer and increases his/her work rate, which in turn increases the feasibility of the project by reducing overall cost.

Using the MVC pattern directly affects the UX (User Experience) as it restricts what the user can access and interact with. Another advantage of using MVC is that the user would notice is that it allows for asynchronous requests which have an impact on the load times, which is often a primary concern for a user, [19].

One benefit of this modular design pattern is that a developer can work on each of the components at the same time, which reduces the development process of the overall project. This has the disadvantage of needing multiple programmers which would have a direct impact on the project's budget.

Additional positives of using the MVC pattern is that any modifications on one of the components of the architecture does not affect the other components, although this can lead to the added complexity of the overall architecture, [20].

Model

The model component is responsible for “Data-Related Logic” that a user works with such as fetching and processing data. The model can interact with a database and make changes to those databases such as SELECT, UPDATE or DELETE data from the databases. The model will take instructions from the Controller on how to display data to the user [21, p. 18]. The Model is only responsible for the information held with that class structure and is not concerned with services that manipulate the information within the class, [22].

View

The View is responsible for displaying the data to the user normally in a form of user interface (UI) and how the data is displayed. It is possible that many views linked to a single controller [21]. The View in MVC is controlled directly by the Controller, [22], [23].

Controller

The Controller is in control of communication between the Model and View components, which can be in many forms such as button clicks or refreshing a HTML (Hyper Text Markup Language) page. The Controller gets data from the model [23].

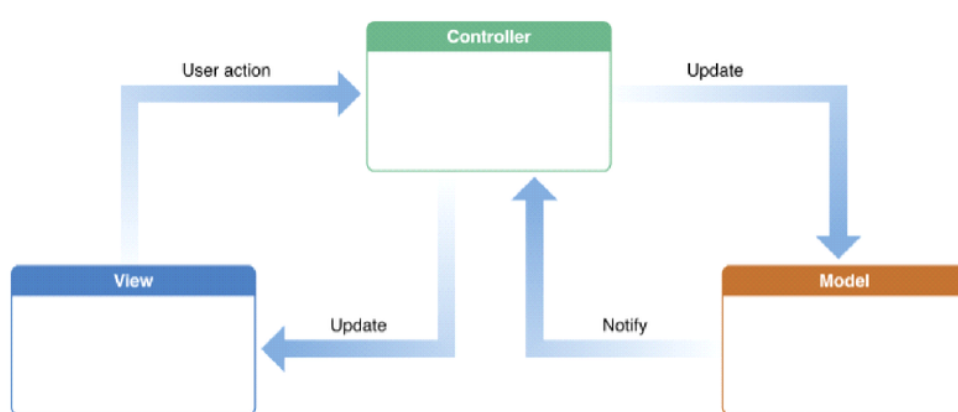


Figure 1. An example of MVC architecture, [24].

Figure 1 represents a user requesting information from a database. The flow of information from the View to the controller is a request from the user. The Controller then interacts with the Model who retrieves the requested information. The Model then replies to the Controller which then updates the View for the user.

2.0 Facade Pattern

The purpose of a Façade pattern is to provide a client program with a simple unified interface to a set of subsystems. It does this by encapsulating multiple subsystems together. The Façade pattern provides a higher-level interface which makes subsystems simpler to use, [25] “With the Facade Pattern you can take a complex subsystem and make it easier to use by implementing a Facade class that provides one, more reasonable interface,” [26].

The Façade pattern allows a level of abstraction from the subsystems by decoupling a client from the subsystems. Façade patterns are ideal for systems that are highly decoupled with lots of classes [21]. The façade pattern belongs to a larger group of patterns known as the Structural patterns; some of the other patterns within this group are Adapters, Proxy, and a bridge pattern, [26]. “There can be significant benefit in wrapping a complex subsystem with a simplified interface”, [26].

The Adapter and the Facade patterns can both wrap classes, although the reason that this occurs is very different between the two patterns. Façade does this to simplify the process while the Adapter pattern does this to convert the interface to something different, [26, pp. 139 - 150].

User Experience (UX) can be drastically affected by using the Façade pattern as it can simplify the users’ interaction with the chosen program.

The developer should aim to use a Façade pattern when they want a simple interface between a complex subsystem and a client. These happen naturally when a developer follows the Single

Responsibility Principle as laid out by Robert C. Martins SOLID principles [27], as this often leads to many classes each that does a single function, then a developer may need an interface between their many classes and their client, [28].

An example of a Façade pattern can be demonstrated with the modular home theater example.

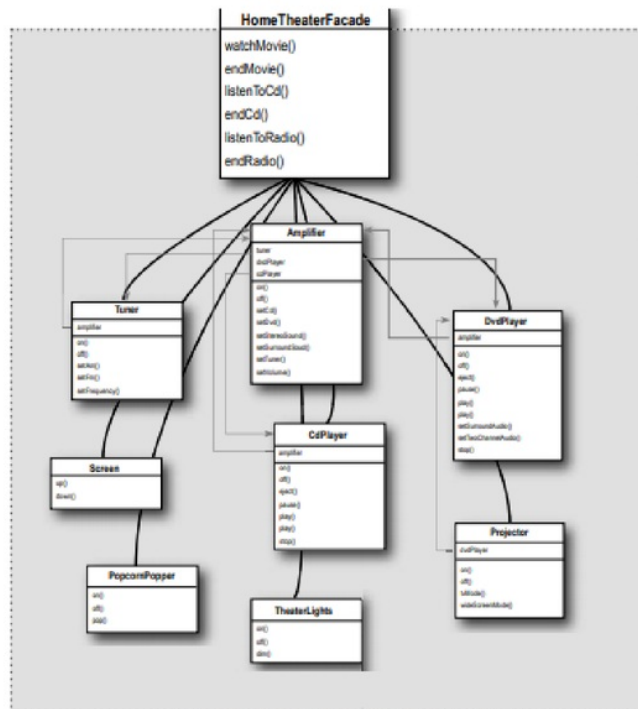


Figure 1. Home Theatre Façade. [29]

Figure 1 shows the many classes of the home theatre such as projector, screens and DVD player, all of which have been linked under a home theatre façade, which can be used to access any of the child classes which streamlines a developer's task of using a high number of classes.

An example of the Façade pattern can be demonstrated with the customer service example below.

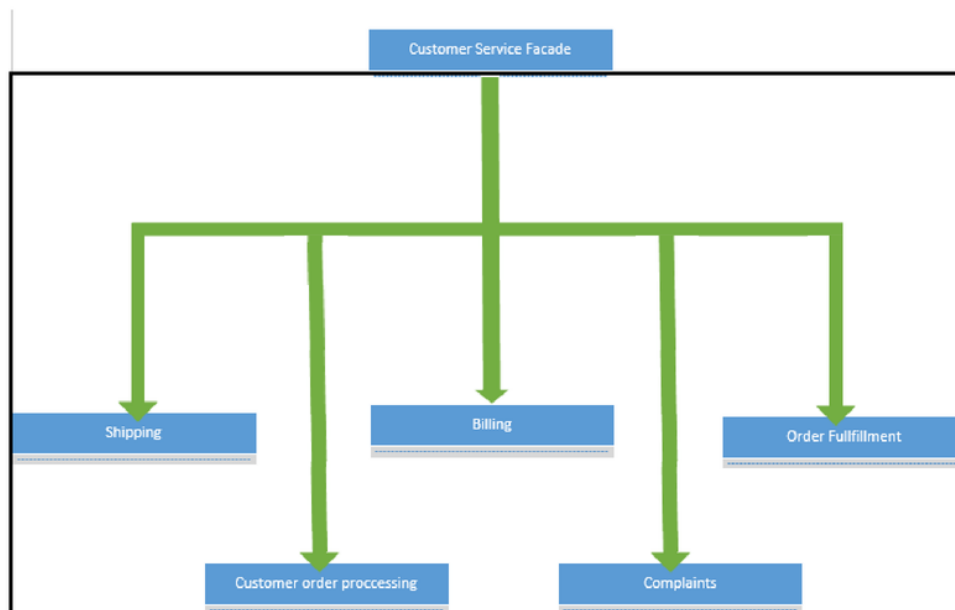


Figure 2. Façade example

Figure 2 shows the Customer Service Façade when a client contacts the customer services of a company the customer service representative acts as the façade which acts as an interface between client and customer.

The Customer Service Façade has encapsulated the entire ordering system of the company which has simplified the ordering process within the company.

Summary

The Façade patterns intend to make the life of a developer easier by allowing combined code to be used several times over multiple applications. Facades are a convenient way of providing high-level functionality that may be difficult for a developer to implement. Façade patterns also reduce the dependencies on outside code which creates extra flexibility when building applications, [28, pp. 141 -148].

The Façade pattern provides a convenient way of interface between complex tasks, it also provides a way of keeping client and subsystems loosely coupled.

One adverse effect of the Façade pattern is the often misuse of the pattern or the misunderstanding of the purpose of the pattern. There is often overuse of the Façade pattern when there is no use for its implementation, one such issue is when a developer would benefit from using an Adapter pattern and not a Façade pattern.

3.0 Investigation of MVVM pattern

Model View ViewModel (MVVM) was designed by John Grossman and released to the public in 2005 through the “John Gossman’s observations on Avalon Development” [30]. MVVM is a software architectural pattern which separates the presentation logic from the business logic. MVVM is widely used in Windows 8, WPF (Windows Presentation Logic) and Silverlight applications, [31, pp. 63 - 68].

MVVM allows a developer to change any segment at any point in the development or after release without impacting on any other part of the application. MVVM also allows the application to be extended or modified with fewer difficulties. It is also possible to have multiple developers working on different aspects of the application without changes made on one aspect affecting the others, [32].

The MVVM pattern is ideally suited for more complex applications, as it allows the developer to perform easier testing with increased modularity. MVVM promotes the use of data binding, [31].

The purpose of the MVVM is to present a clear SOC (Separation of Concern) and SRP (Single Responsibility Principle). MVVM adheres to the high cohesion and loose coupling principles which allows for easier testing and allowing the code used in the application to be reused for multiple applications, [33, p. 76].

Why use MVVM?

With traditional UI (User Interface), a developer would create a View and then this View would control the event handling, data model and the initialization code. This approach had the effect of increasing the size of the View class. This had the drawback of adding a strong dependency between the data binding logic and the UI. In this case, only a single developer would be able to on the same class. Furthermore, having the View class so tightly coupled with the model and business logic has a detrimental effect on the testability, maintainability, and extendibility.

A developer using the MVVM design pattern will avoid the tight coupling and dependency issues by following the rules of single dependency and separation of concerns to produce an application that is more maintainable, testable and easily extendable between several applications, [34].

A developer who decides to use the MVVM design pattern can affect the User Experience (UX) because MVVM has the ability of asynchronous operations which affect the load and run times of an application. In addition, testing on MVVM is modular, so its possible for a specific test or debug to be run on each of the components of the design pattern, [31, p. 11].

View

Within MVVM the view is known as an active state rather than a passive. The View may contain events behaviors and data bindings and for this, the View must have knowledge of Model and ViewModel, [35]. Databinding is a technique that binds the data source and the data consume together and synchronizes them, [36]. The View has two main responsibilities accept input from the user and to present the data the model provides, [33]. The View makes use of (Extensible Application Markup Language (XAML) to initialize structured values and objects, [37]. The View does not have any direct knowledge of the model, it is only aware of the ViewModel this is most commonly a one-to-one relationship. Best practice states that the view should not have any code-behind, although this does depend on the implementation, [38].

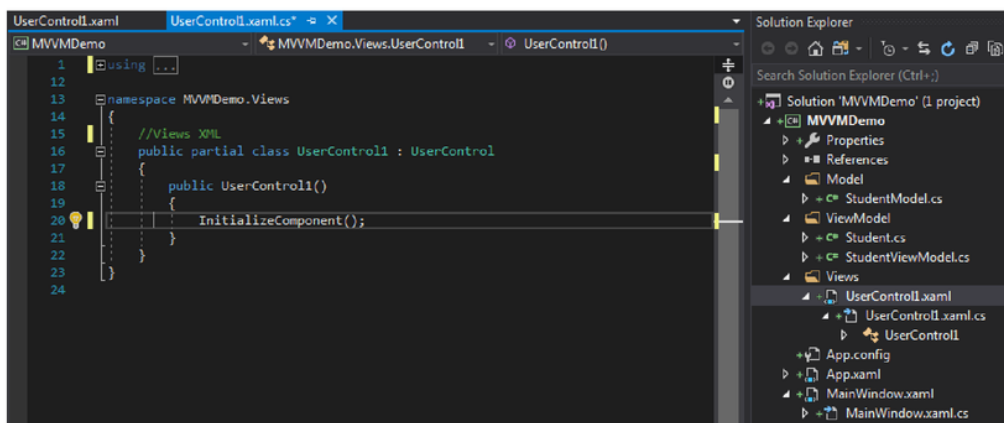


Figure 2. Example of a View XML used for a MVVM Demo

ViewModel

The ViewModel acts as a mediator between the View and the Model, [33, p. 110]. ViewModel is responsible for preparing and the managing of the data for an activity. When a user performs an action on the view, ViewModel can modify the data held on the Model, [35], [39]. The ViewModel can either be one-to-one or a one-to-many relationship also the ViewModel should directly depend on the Model for its business logic, [38].

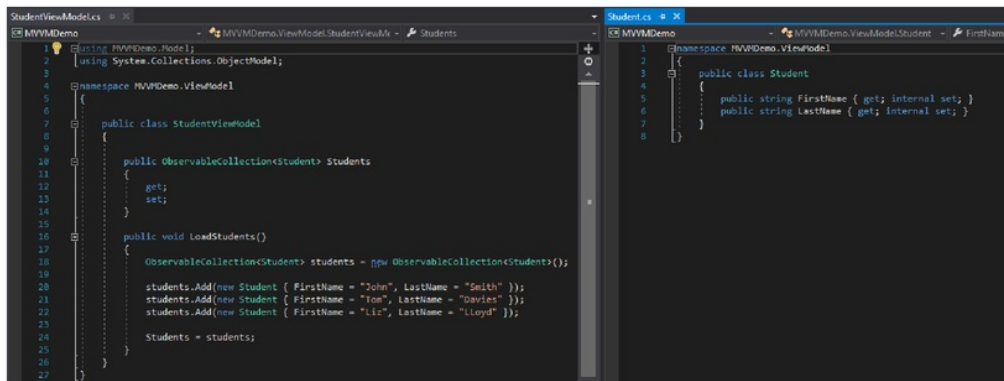


Figure 3. Example of The Student ViewModel classes

Model

The model is responsible for data access, and core business logic. The information that gets retrieved from the ViewModel is held and updated in the Model if the ViewModel gets a save or load action from the View the information is saved or sent from the Model, [35]. Best practice states that the Model should not directly or indirectly reference either the ViewModel or the View, [38].

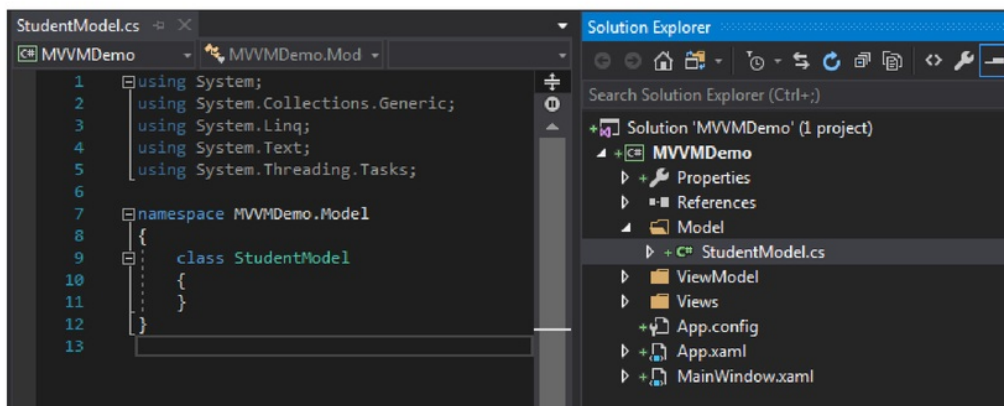


Figure 4. Example of the Student Model

Summary of MVVM

MVVM offers an excellent level of maintainability, testability and extensibility for larger project sizes, although the use of MVVM is reliant on the developers knowledge of the design pattern to be extensive, as a developer being unfamiliar with the design pattern can lead to corner cutting which can lead to the principles of the pattern being broken.

MVVM allows the development team to work independently of each other while still working on the same project with a high level of modularity, this has the benefit of core components from projects being interchangeable saving time and improving efficiency.

On a small-scale project, MVVM can add a level of complexity that is unnecessary and can add to a projects overall development time and decreasing efficiency.

All testing and building were done with Android studios build 3.2.1 and on API 28 unless stated otherwise and testing was done on either a Galaxy Tab S3 or Nexus 4 mobile Device. A white box testing approach was used in the three exercises.

Exercise 1.

Develop a dice roll application, allow a user to roll two dice, one dice should belong to the AI player and the other to the user, the outcome of the dice rolls must be randomly allocated to each dice roll.

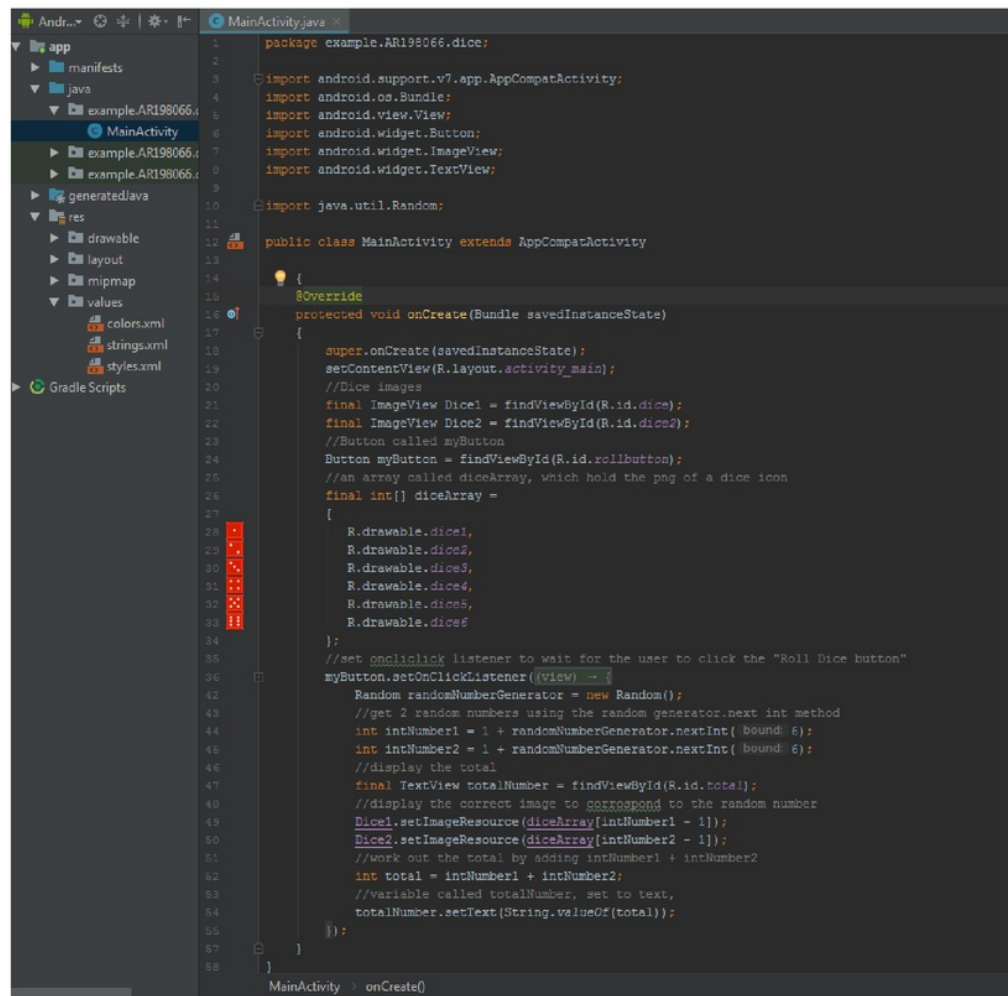


Figure 5. illustrates the Java code behind the exercise

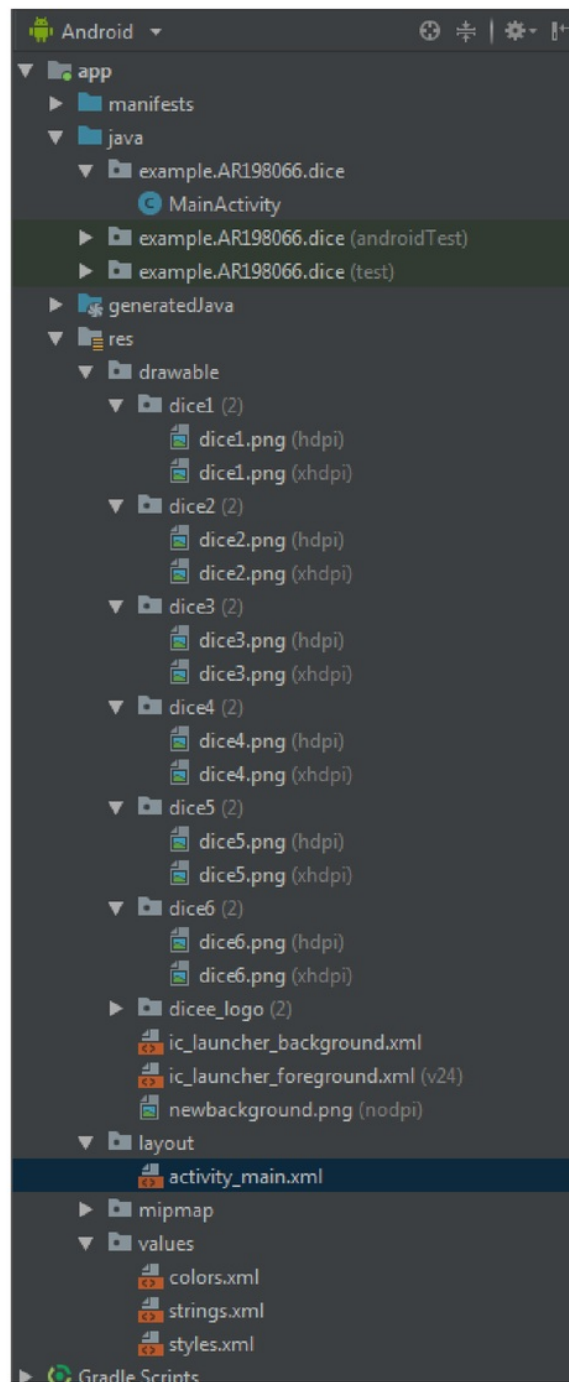


Figure 3. shows the locations of the dice logos

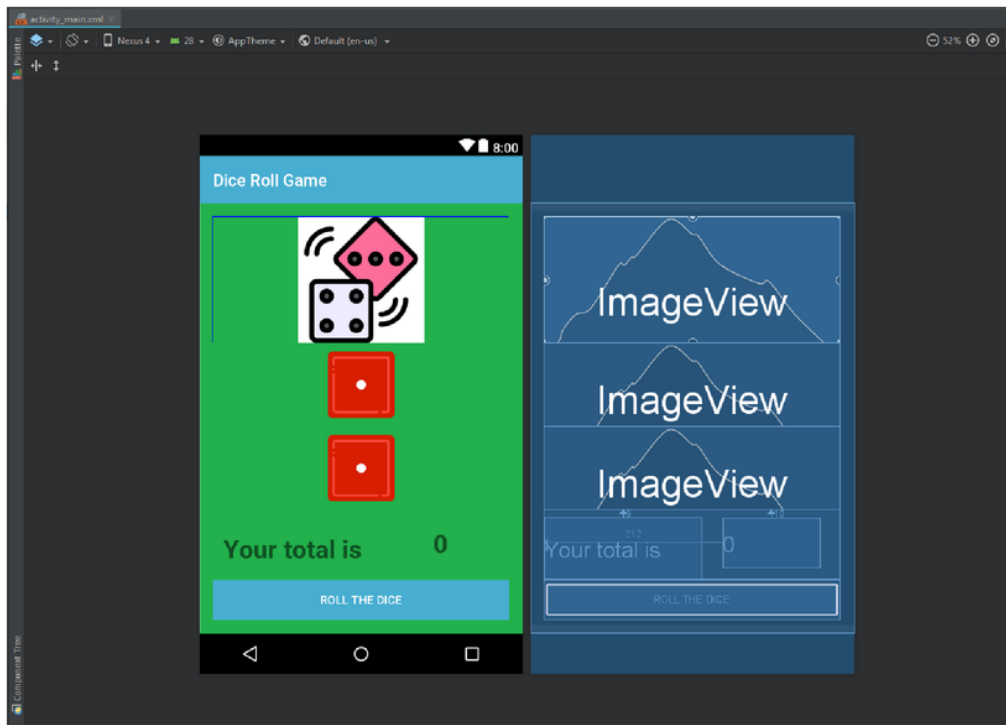


Figure 6. demonstrates the XML layout of the application

All images were sourced from royalty free sites, [40].

Pre-conditions.				
Android Build No. 3.2.1				
API No. 28				
testing carried out on Galaxy Tab S3				
Exercise Number 1				
DExercise Description: Dice Roll				
Step No.	Test Case	Test Data	Expected result Pass/Fail	Result (Pass/Fail)
1	Does Application Load	Application loaded successfully	Load successfully	Yes
2	Does Application Allow Random number generation	Yes Number generation is random	Random number generation	Yes
3	Does Dice picture match number result	Dice images match total	Dice total to match dice	Yes
4	Unexpected Crashes	none	No errors expected	Yes
5	Dice total work	Yes dice adds up to the total	Dice total to match dice	Yes
6	portrait and landscape layout acceptable	Both layouts keep format	both layouts should keep restraints	Yes

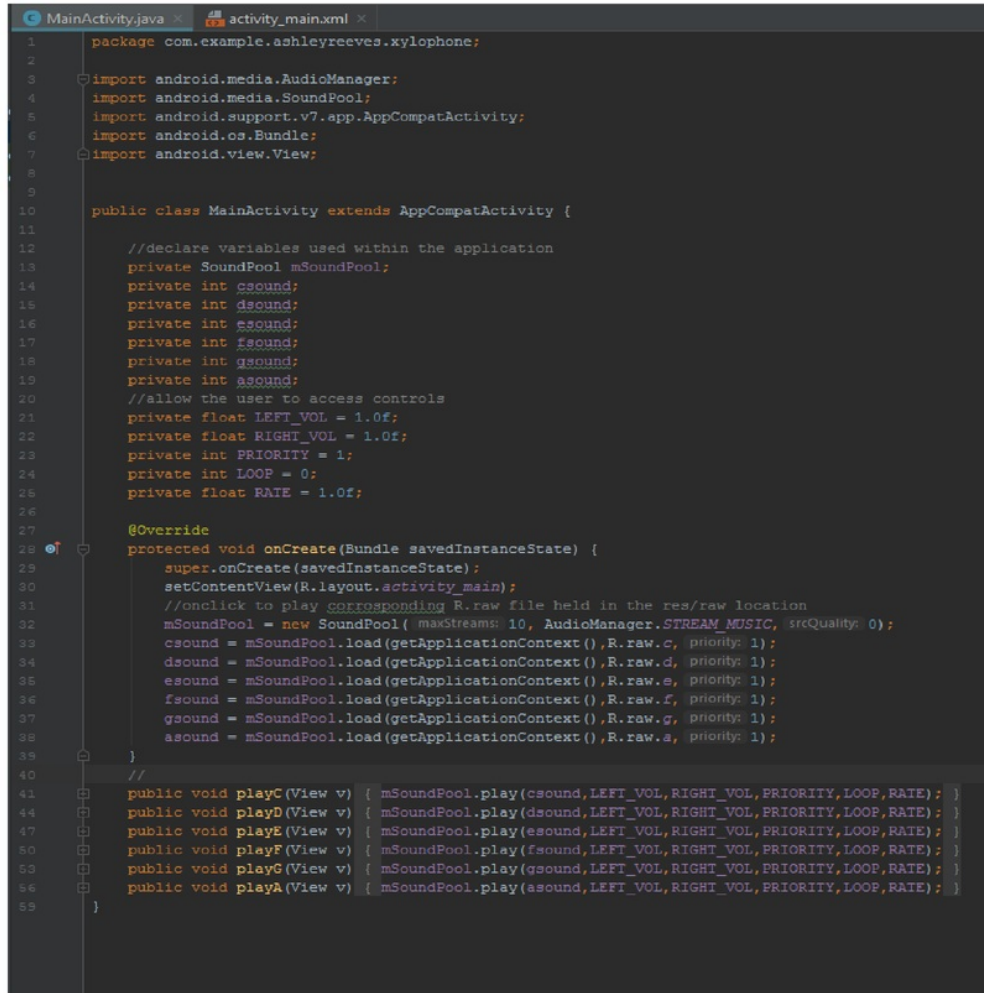
Figure 7. Testing method

Summary

The application meets the criteria of allowing a user to roll two dice, and display the total to the user, the numbers generated for each dice is random. Future development would be to label each dice as either player dice or opponents dice and display whether the player or opponents dice is the highest and the winner of that round.

Exercise 2.

Develop an application that allows a user to create an audio sample from a user action.

A screenshot of an IDE showing the MainActivity.java file. The code is for an Android application named 'xylophone'. It imports AudioManager, SoundPool, AppCompatActivity, Bundle, and View. The MainActivity class extends AppCompatActivity and declares several private variables for sound IDs, volumes, priority, loop, and rate. It overrides onCreate to initialize a SoundPool and load eight raw sound files (c, d, e, f, g, a, s, and another a). It also defines play methods for each sound ID, each calling mSoundPool.play with the respective sound ID and volume/rate parameters.

```
1 package com.example.ashleyreeves.xylophone;
2
3 import android.media.AudioManager;
4 import android.media.SoundPool;
5 import android.support.v7.app.AppCompatActivity;
6 import android.os.Bundle;
7 import android.view.View;
8
9
10 public class MainActivity extends AppCompatActivity {
11
12     //declare variables used within the application
13     private SoundPool mSoundPool;
14     private int csound;
15     private int dsound;
16     private int esound;
17     private int fsound;
18     private int gsound;
19     private int asound;
20     //allow the user to access controls
21     private float LEFT_VOL = 1.0f;
22     private float RIGHT_VOL = 1.0f;
23     private int PRIORITY = 1;
24     private int LOOP = 0;
25     private float RATE = 1.0f;
26
27     @Override
28     protected void onCreate(Bundle savedInstanceState) {
29         super.onCreate(savedInstanceState);
30         setContentView(R.layout.activity_main);
31         //onclick to play corresponding R.raw file held in the res/raw location
32         mSoundPool = new SoundPool( maxStreams: 10, AudioManager.STREAM_MUSIC, srcQuality: 0);
33         csound = mSoundPool.load(getApplicationContext(), R.raw.c, priority: 1);
34         dsound = mSoundPool.load(getApplicationContext(), R.raw.d, priority: 1);
35         esound = mSoundPool.load(getApplicationContext(), R.raw.e, priority: 1);
36         fsound = mSoundPool.load(getApplicationContext(), R.raw.f, priority: 1);
37         gsound = mSoundPool.load(getApplicationContext(), R.raw.g, priority: 1);
38         asound = mSoundPool.load(getApplicationContext(), R.raw.a, priority: 1);
39     }
40
41     //
42     public void playC(View v) { mSoundPool.play(csound, LEFT_VOL, RIGHT_VOL, PRIORITY, LOOP, RATE); }
43     public void playD(View v) { mSoundPool.play(dsound, LEFT_VOL, RIGHT_VOL, PRIORITY, LOOP, RATE); }
44     public void playE(View v) { mSoundPool.play(esound, LEFT_VOL, RIGHT_VOL, PRIORITY, LOOP, RATE); }
45     public void playF(View v) { mSoundPool.play(fsound, LEFT_VOL, RIGHT_VOL, PRIORITY, LOOP, RATE); }
46     public void playG(View v) { mSoundPool.play(gsound, LEFT_VOL, RIGHT_VOL, PRIORITY, LOOP, RATE); }
47     public void playA(View v) { mSoundPool.play(asound, LEFT_VOL, RIGHT_VOL, PRIORITY, LOOP, RATE); }
48 }
```

Figure 8. Xylophone Java

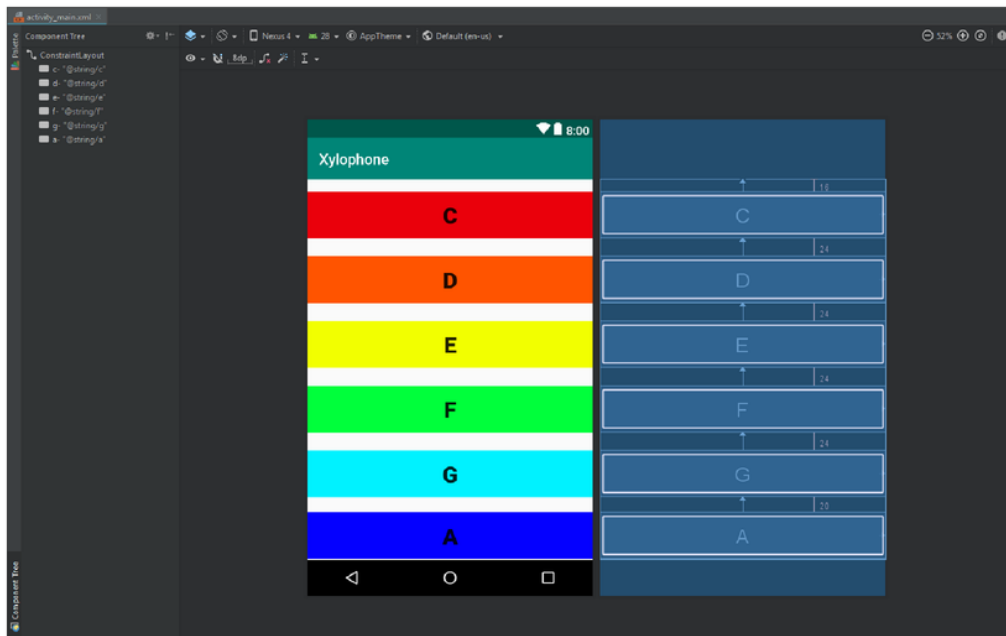


Figure 9. demonstrates the exercise layout

The layout is consistent in both landscape and portrait views.

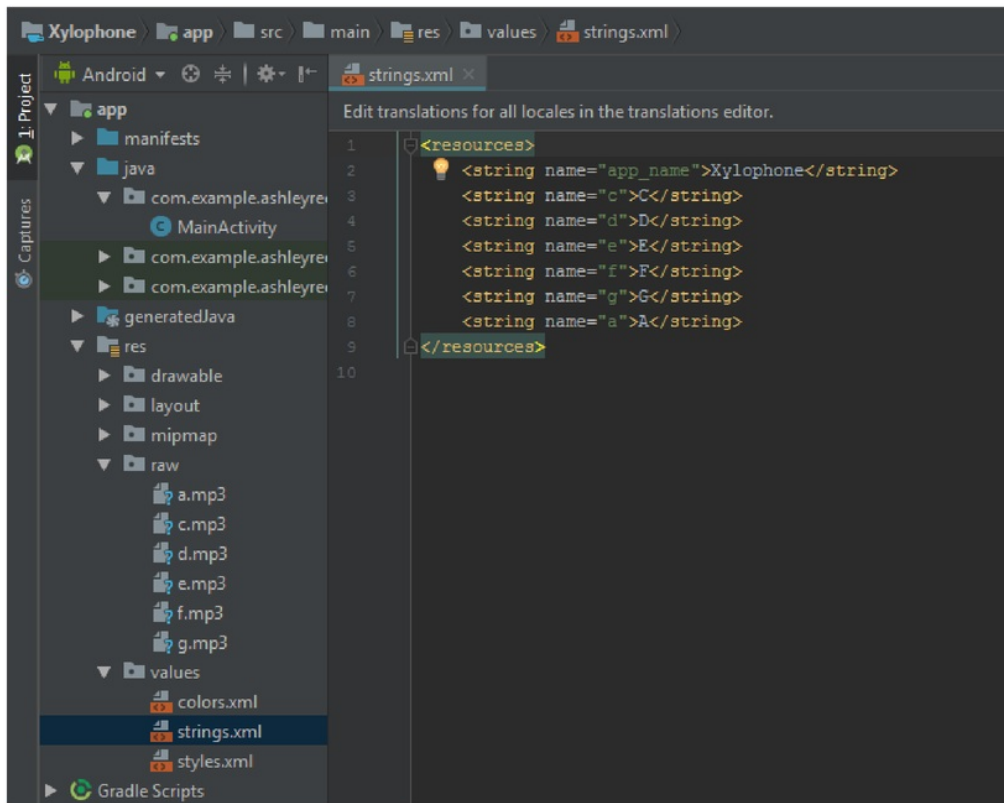


Figure 10. Resource file, and hardcoded String names

All the mp3 files were sourced from a free to use and modify, [41].

Pre-conditions.				
Android Build No. 3.2.1				
API No. 28				
Testing carried out on Galaxy Tab S3				
Exercise Number 2				
Diverse Description: Sounds Board				
Step No.	Test Case	Test Data	Expected result	Result (Pass/Fail)
1	Does Application Load	Application loaded successfully	Load successfully	Yes
2	Unexpected crashes	None	any unexpected crashes when app runs	Yes
3	Orientation	Re-orientate device	Orientation should stay the same through both landscape and portrait	Yes
4	Volume controls work	Try the volume controls	Sound to increase or decrease	Yes
5	audio play from user interaction	Press multiple buttons	Multiple sounds to play	Yes
6	Application match design criteria	Design Brief	Test against design brief	Yes

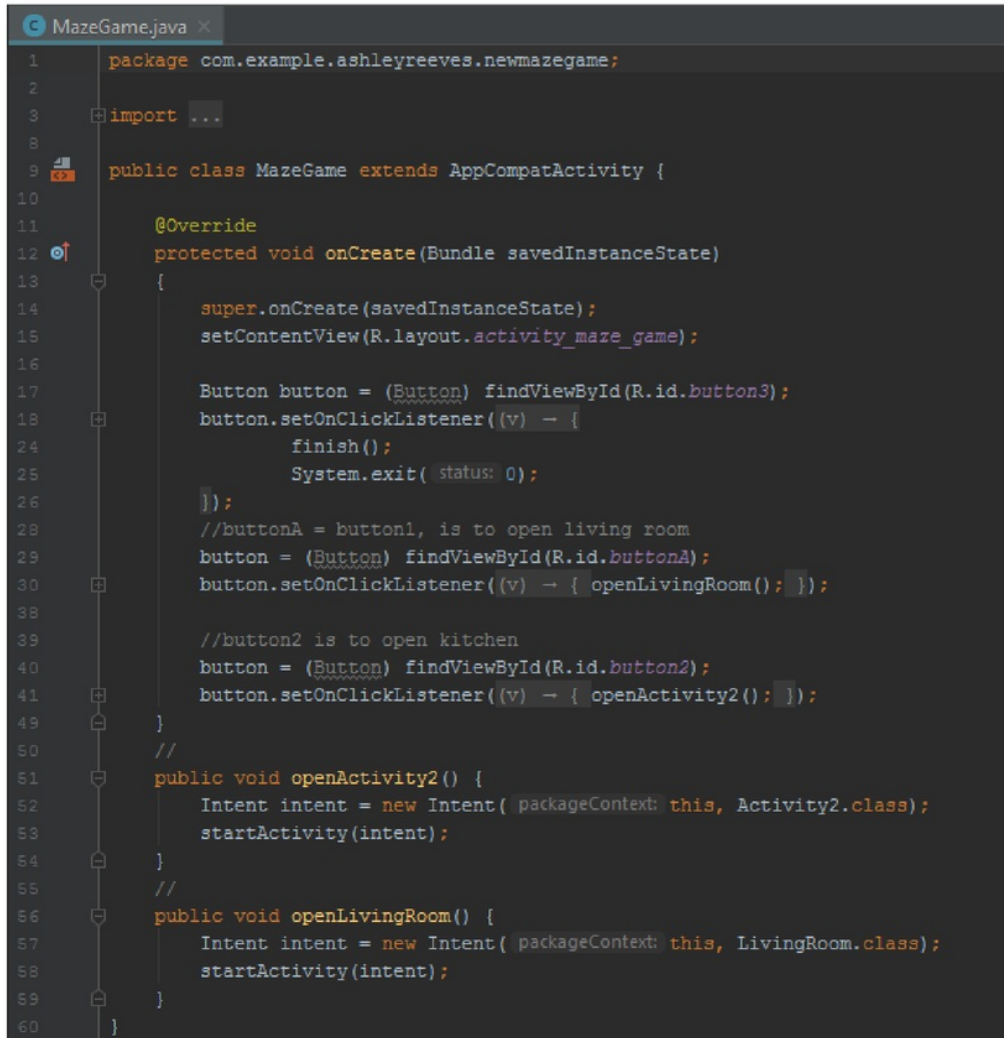
Figure 11. Testing Method

Summary

For future Development allow a user to record, and then play audio that the user has created and allow the user to store and share audio on multiple platforms.

Exercise 3.

Develop a Text based game that allows the user to choose his path through a series of actions that have been predetermined.



```
1 package com.example.ashleyreeves.newmazegame;
2
3 import ...
4
5 public class MazeGame extends AppCompatActivity {
6
7     @Override
8     protected void onCreate(Bundle savedInstanceState)
9     {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.activity_maze_game);
12
13         Button button = (Button) findViewById(R.id.button3);
14         button.setOnClickListener((v) -> {
15             finish();
16             System.exit( status: 0);
17         });
18         //buttonA = button1, is to open living room
19         button = (Button) findViewById(R.id.buttonA);
20         button.setOnClickListener((v) -> { openLivingRoom(); });
21
22         //button2 is to open kitchen
23         button = (Button) findViewById(R.id.button2);
24         button.setOnClickListener((v) -> { openActivity2(); });
25     }
26
27     //
28     public void openActivity2() {
29         Intent intent = new Intent( packageContext: this, Activity2.class);
30         startActivity(intent);
31     }
32
33     //
34     public void openLivingRoom() {
35         Intent intent = new Intent( packageContext: this, LivingRoom.class);
36         startActivity(intent);
37     }
38 }
39
40 }
```

Figure 12. Main Game Java.

Figure 12 displays the Java code to produce the main page of the exercise. Here we have two buttons that allow the user to progress to either a living room activity or a kitchen activity “Activity2” with a simple click. This will allow the user to progress through the text-based game. Lines 17 -26 allows the user to exit the application.

```

1 package com.example.ashleyreeves.newmazegame;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.TextView;
8
9
10 public class LivingRoom extends AppCompatActivity {
11
12     //variables used within displaying message to user
13     TextView txt;
14     Button btn;
15
16     @Override
17     protected void onCreate(Bundle savedInstanceState) {
18         super.onCreate(savedInstanceState);
19         setContentView(R.layout.activity_living_room);
20
21         //used to change message upon user clicking "Turn off button"
22         txt = (TextView) findViewById(R.id.textView3);
23         btn = (Button) findViewById(R.id.button7);
24
25         //display updated message to user
26         btn.setOnClickListener((v) -> {
27             txt.setText("as you turn off the TV, you can hear a voice behind you saying your name, but as you turn you see that no one is there! " +
28                 "a cold shiver runs down your back. You must get out of the house fast!");
29         });
30
31         //used to change message upon user clicking "Try to open the window"
32         txt = (TextView) findViewById(R.id.textView3);
33         btn = (Button) findViewById(R.id.button8);
34
35         btn.setOnClickListener((v) -> {
36             txt.setText("The window has been called shut and boarded up on the outside, you must find another way out of the house!");
37         });
38
39         Button button = (Button) findViewById(R.id.button3);
40         button.setOnClickListener((v) -> {
41             finish();
42             System.exit(1);
43         });
44     }
45 }

```

Figure 13. Living Room Java

Figure 13 shows the code behind the living room activity a message setting the scene is displayed to the user which is held in the "Intro2" string found within figure 15. The user can select one of the three options available: Option 1 will display a new message to the user to inform them that they have turned off the TV. Option 2 allows the user to try and complete the text-based game by trying to open the window to escape. Option 3 allows the user to exit the game from this point.

```

1 package com.example.ashleyreeves.newmazegame;
2
3 import android.support.v7.app.AppCompatActivity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Button;
7 import android.widget.TextView;
8
9
10 public class Activity2 extends AppCompatActivity {
11
12     TextView txt;
13     Button btn;
14
15     @Override
16     protected void onCreate(Bundle savedInstanceState) {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.activity_2);
19
20         //used to change message upon user clicking "try the door"
21         txt = (TextView) findViewById(R.id.textView3);
22         btn = (Button) findViewById(R.id.button4);
23
24         btn.setOnClickListener((v) -> {
25             txt.setText("you try the door handle, the door seems locked, you push harder, the door begins to open, you hear your name being shouted at you from behind, but decide not to " +
26                 "move rather you force the door open. The door opens fast enough for you to squeeze through, the voice behind continues to shout your name as he approaches you. " +
27                 "he grabs your arm as you push your way through the door, but you manage to shake him off, and run into the kitchen as the door flies open, and a cold shiver runs down your back!");
28         });
29
30         //used to change message upon user clicking "pick up a weapon"
31         txt = (TextView) findViewById(R.id.textView3);
32         btn = (Button) findViewById(R.id.button5);
33
34         btn.setOnClickListener((v) -> {
35             txt.setText("you look for one of the knives from the kitchen working and hold it in both hands");
36         });
37     }
38 }

```

Figure 14. Kitchen Java.

Figure 14 illustrates the code used within the kitchen activity. Again, the user will be shown a message that has been held in the Strings XML. The user will be able to choose from the 3 options available. Option 1 in the kitchen activity allows the user to try and escape the house and win the game by exiting through the kitchen door. Option 2 allows the user to pick up a weapon to try and defend himself. Option 3 allows the user to exit the game from this point in the application.

```

<resources>
    <string name="app_name">Super Fun Maze Game</string>
    <string name="Intro">You wake up from the night before, where you drank to much and cannot remember what happened, you slowly wake, to realise your alone and slowly get to your feet, from down the hallway you can hear screaming and raised voices, "You must escape the house without being captured!"</string>
    //story1 progress into hallway
    <string name="Story1">Enter the living room</string>
    //story2
    <string name="Story2">Enter the kitchen</string>
    //
    <string name="Intro">You enter the kitchen, and you see the backdoor of the house, just to the side of the door you see an array of sharp knives </string>
    //Story3
    <string name="Story3">try the door</string>
    //Story4
    <string name="Story4">search for a weapon</string>
    //
    <string name="Intro">you enter the living room and you see the TV is showing a film where the main characters are arguing and screaming at each other "you think to yourself was this the screaming you
    //Story5
    <string name="Story5">Turn off the TV</string>
    //Story6
    <string name="Story6">try to open the window</string>
    // ExitGame button to close App
    <string name="ExitGame">Exit the game</string>
</resources>

```

Figure 15.Strings XML.

Figure 15 displays the pre-populated messages that are held in the strings XML format and are accessed via the @ String method which can be seen in Figure 6.

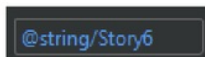


Figure 16. @ String method.

Figure 16 shows the @ string method of referencing a string from an XML string.

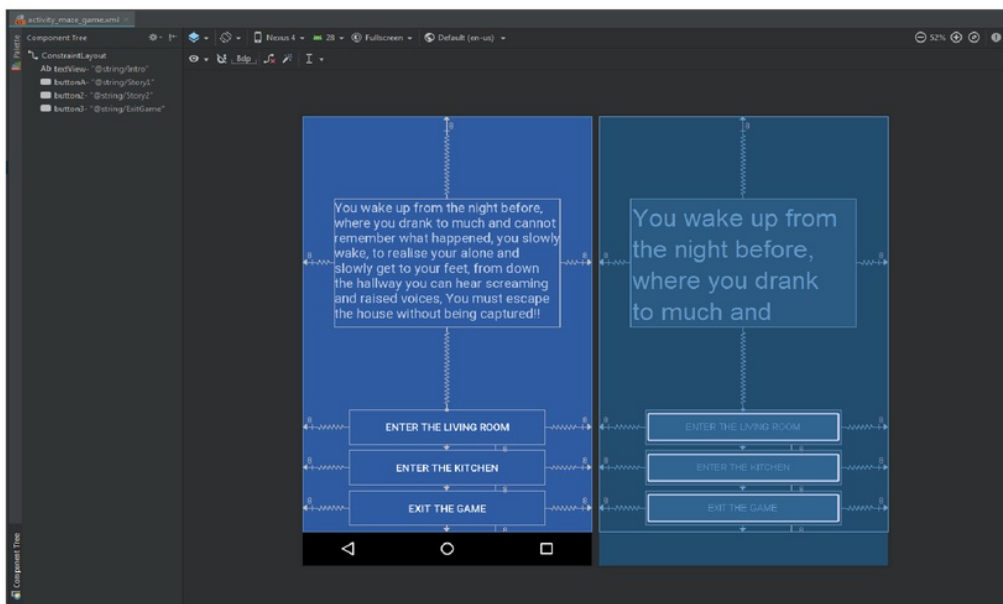


Figure 17. Demonstration of the main design theme

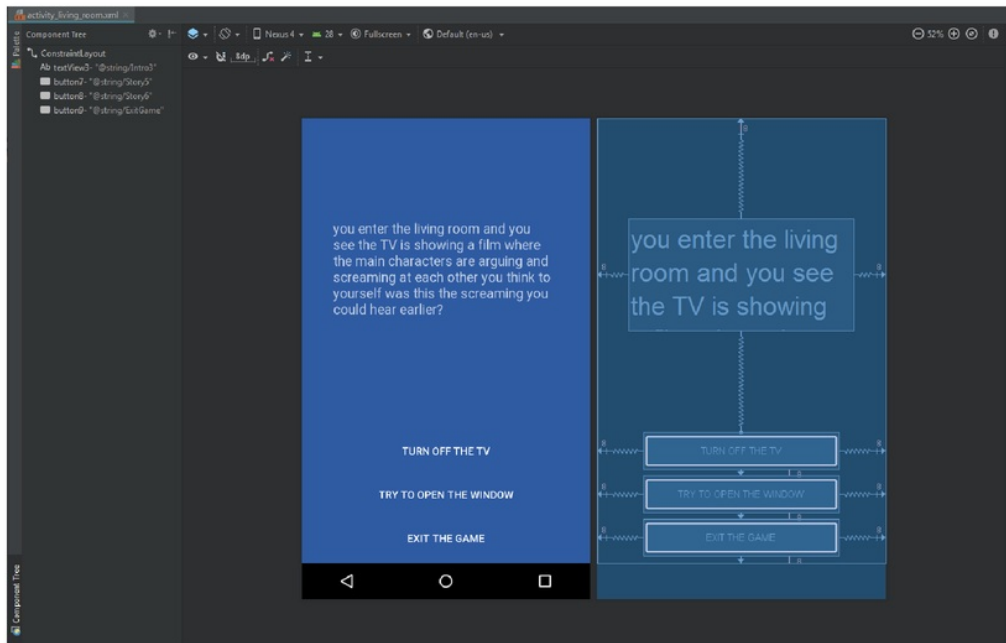


Figure 18. Demonstration of the Living room design.

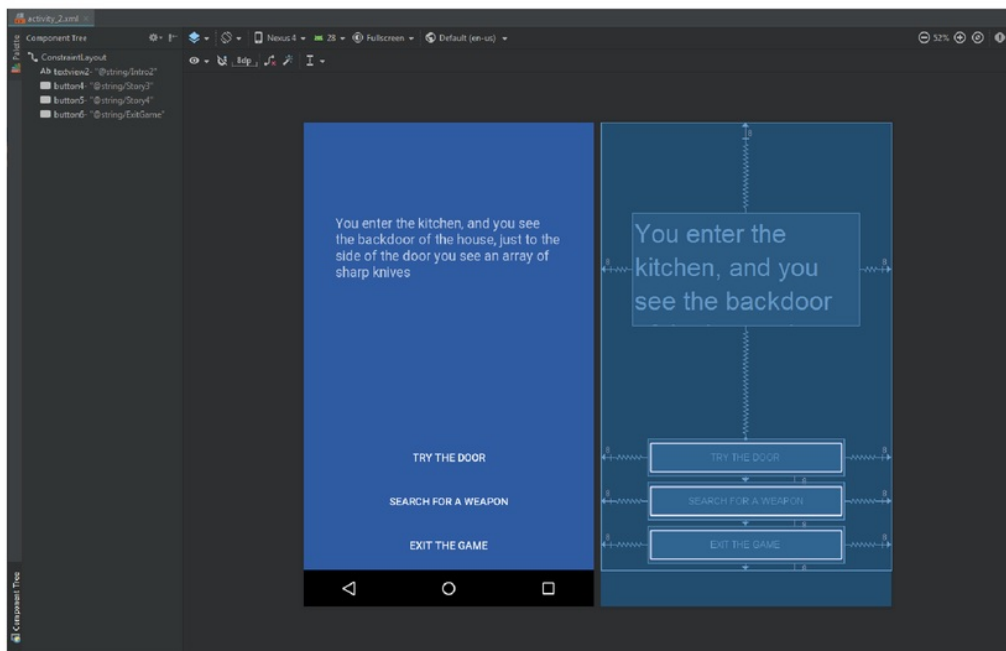


Figure 19. Demonstration of the kitchen design

Figures 17, 18 and 19 illustrate the design patterns used with the application. The exercise uses a full-screen theme to hide the taskbar from the user which will hide the time and notifications from the user. The background color scheme was changed from blue to a lighter shade of blue to allow the user to better read the text on the screen. Facebooks Colour scheme of blue and white was chosen as a template for the text game, [42].

Pre-conditions.

Android Build No. 3.2.1
API No. 28
testing carried out on Galaxy Tab S3
Exercise Number 3
DExercise Description: TextBased Adventure Game

Step No.	Test Case	Test Data	Expected result Pass/Fail	Result (Pass/Fail)
1	Does Application Load	Application loaded successfully	Load successfully	Yes
2	Unexpected crashes	None	any unexpected crashes when app runs	Yes
3	Orientation	Re_orientate device	Orientation should stay the same through both landscape and portrait	Yes
4	Exit Application	press the exit game button	Application to close	Yes
5	Button Navigation	Press the new story buttons	New story path to load	Yes
6	Return Button	Press the return button	Character should return to the previous story	Yes
			Result	Pass

Figure 20. Testing Method

Summary

For future modifications, a return button should be implemented on screen rather than having the user to use the prebuilt return button that is available. Additionally, all colors used within the application should be loaded into the colors XML and then referenced rather than declaring them in the activities.

Total Word Count: 3382

Bibliography

- [1 R. B'Far, "Mobile Computing Principles," in *Designing and Developing Mobile Applications with UML and XML*, Cambridge, England, Cambridge, 2005, pp. 4 - 9.
- [2 I. Global, "Application Development and Design," in *Concepts, Methodologies, Tools, and Applications*, Washinton DC, ICI Global, 2017, pp. 90 - 101.
- [3 F. Shearer, "Power Management in Mobile Devices," New York, Newespresss, 2007, pp. 152 - 180.
- [4 S. C. &. J. Lyward, *Man Vs Big Data*, London: Aurum Press, 2017.
- [5 "op 10 Best GPS Tracking Apps For Android in 2018," 13th June 2018. [Online]. Available: <https://www.redbytes.in/top-10-best-gps-tracking-apps-for-android/>. [Accessed 17th November 2018].
- [6 S. A. A. &. M. Llyas, *Location Based services handbook*, Boca Raton, Florida: CRC Press, 2017.
- [7 K. Warwick, "Artificial Intelligence," in *The Basics*, Reading, Routlegde Publishing, 2012, pp. 13 - 17.
- [8 S. Eisen, "Mastercard Enhances Artificial Intelligence Capability," 17th July 2017. [Online]. Available: <https://newsroom.mastercard.com/press-releases/mastercard-enhances-artificial-intelligence-capability-with-the-acquisition-of-brighterion-inc-2/>. [Accessed 14th November 2018].
- [9 StarBucks, "StarBucks," StarBucks Coffee Company, https://play.google.com/store/apps/details?hl=gb&id=com.starbucks.mobilecard&referrer=mat_click_id%3D85fd24ecdf9b5048b45018372ec7e721-20181220-19034%26utm_campaign%3DInternal%26utm_content%3DInternal%26utm_medium%3DInternal%26utm_source%3Dstarbucks.com%26, 2018.
- [1 K. Taylor, "Starbucks online ordering," 25th August 2017. [Online]. Available: <https://www.businessinsider.com/starbucks-attempts-to-fix-mobile-ordering-review-2017-8?r=US&IR=T>. [Accessed 7th Novermber 2018].
- [1 J. L. &. K. Babilinski, "Augmented Reality for Developers," in *Build practical augmented reality applications with Unity, ARCore, ARKit, and Vuforia*, Birmingham England, Packt Publishings, 2017, pp. 1 - 26.
- [1 I. Store, "Ikea," Inter Ikea Systems , 21st September 2017. [Online]. Available: https://play.google.com/store/apps/details?id=com.ikea.kompis&hl=en_GB. [Accessed 28th October 2018].
- [1 V. Bhargava, "The Growth Of Mobile Apps," Medium, 2017.

- [1 R. K. Tong, Wearable technology in Medicine and Health Care, Cambridge, Massachusetts:
4] Academic Press, 2018.
- [1 J. M. & D. Bryson, Smart Clothes and Wearable Technology, Sawston, Cambridge: Woodhead
5] Publising, 2009.
- [1 A. Pantelopoulous and N. G. Bourbakis, "A Survey on Wearable Sensor-Based Systems for Health
6] Monitoring and Prognosis," *A Survey on Wearable Sensor-Based Systems for Health Monitoring and Prognosis*, vol. 40, no. 1, pp. 1 - 12, 2010.
- [1 A. F. & S. Sanderson, "Pro ASP.NET MVC 3 Framework," New York, Apress, 2011, pp. 10 - 19.
7]
- [1 J. Ciliberti, "ASP.NET Core Recipes,,," in *A problem_Solving Approach*, New York, Apress, 2017, p.
8] 27.
- [1 J. C. & T. Snyder, "Programming ASP.NET MVC 4: Developing Real-World Web Applications with
9] ASP.NET MVC," California, O'Reilly, 2012, pp. 295 - 298.
- [2 J. Kurtz, "ASP.NET MVC 4 and the Web API," in *Building a REST Service from Start to Finish*, New
0] York, Apress, 2013, pp. 1 - 3.
- [2 R. J. J. V. Richard Helm, "Design Patterns," in *Elements of Reusable Object_oriented Software*,
1] Cambridge, Pearson Education, 1994, pp. 175 - 176.
- [2 E. G. & R. Helm, "Design Patterns," in *Elements of Reusable Object-Oriented Software (Addison-
2] Wesley Professional Computing Series)*, Boston, Addison - Wesley , 1994, pp. 10 - 14.
- [2 E. F. E. R. K. S. B. B. Eric Freeman, "Head First Design Patterns," in *Head First Design Patterns*,
3] Surrey, O'Reilly, 2004, pp. 542 - 543.
- [2 A. Inc., "Model-View-Controller," Apple Inc., [Online]. Available:
4] <https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html>. [Accessed 23rd November 2018].
- [2 W. C. W. Steven John Metsker, "Design patterns in Java," in *Software patterns series*, Boston
5] MA, Pearsons Education Inc, 2006, pp. 33 -46.
- [2 R. H. R. J. & J. V. Erica Gamma, "Design Patterns," in *Elements of reusable Object Orientated
6] software*, Boston, Addison - Wesley, 1994, pp. 137 - 141.
- [2 R. C. Martin, "Clean Code," in *A handbook of agile software craftsmanship*, New Jersey, Prentice
7] Hall, 2008, pp. 150 - 153.
- [2 R. H. Dustin Diaz, Pro JavaScript Design Patterns, New York: Apress, 2008, pp. 147 - .
8]
- [2 E. F. B. B. K. S. Elisabeth Freeman, "Head First Design patterns," in *Head First Design patterns*,
9] vol. 1, New Jersey, O' Reilly & Associates, Inc. ©2004, 2004, p. 258.

- [3 J. Gossman, "observations on Avalon development," *Introduction to Model/View/ViewModel*
0] *pattern for building WPF apps*, vol. 1, no. 1, p. 1, 2005.
- [3 A. F. & S. Sanderson, "Pro ASP.NET MVC 3 Framework," New York, Apress, 2011, p. 11.
1]
- [3 M. S. S. Ryan Vice, "MVVM Survival Guide," in *for Enterprise Architectures in Silverlight and WPF*,
2] Birminham England, Packt, 2012, p. Chapter 2.
- [3 G. M. Hall, "Pro WPF and Silverlight MVVM," in *Effective Application Developement with Model-
3] View-ViewModel*, Apress Publishings, 2010, p. 73.
- [3 A. S. & J. Greene, "Architecting apps with the MVVM pattern," in *Head first C#*, USA, O'Reilly,
4] 2013, p. 749.
- [3 A. S. a. J. Greene, "Head first C#," in *Head first C#*, USA, O'Reilly, 2013, p. 749.
5]
- [3 M. MacDonald, "Pro WPF 4.5 in VB," in *Windows Presentation Foundation in .NET 4.5*, New York,
6] Apress, 2012, pp. 555 - 597.
- [3 Microsoft, "XAML overview (WPF)," 30 March 2017. [Online]. Available:
7] <https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-overview-wpf#what-is-xaml>. [Accessed 29th November 2018].
- [3 J. Bytes, "byte-sized chunks of .NET," *byte-sized chunks of .NET*, vol. 1, no. 1, pp. 1 - 9, 2012.
8]
- [3 G. Hall, "Pro WPF and Siliver Light MVVM," New York, Apress, 2010, p. 81.
9]
- [4 "Public Domain Pictures," Public Domain Pictures, [Online]. Available:
0] <https://www.publicdomainpictures.net/en/view-image.php?image=51222&picture=red-dice>.
[Accessed 19th November 2018].
- [4 "Orange Free Sounds," Orange Free Sounds, [Online]. Available:
1] <http://www.orangefreesounds.com/?s=piano>. [Accessed 3rd November 2018].
- [4 Color_hex, "Facebook Color Palette," [Online]. Available: [https://www.color-hex.com/color-](https://www.color-hex.com/color-palette/185)
2] [palette/185](https://www.color-hex.com/color-palette/185). [Accessed 16th December 2018].