

Stephen Kyle Strahan 180976 - Application Development Assignment 1

by Stephen Strahan

FILE	11695_STEPHEN_STRAHAN_STEPHEN_KYLE_STRAHAN_180976_- _APPLICATION_DEVELOPMENT_ASSIGNMENT_1_164031_1503607909.D OCX		
TIME SUBMITTED	20-DEC-2018 08:19PM (UTC+0000)	WORD COUNT	3242
SUBMISSION ID	97862541	CHARACTER COUNT	20487

BSc Applied Computing
Application Development
Assessment 1

Design Patterns

Name: Stephen Kyle Strahan

Date: 01/11/2018

Table of Contents

Introduction	3
Part A (Understanding)	4
Software/Design Patterns	4
MVC.....	5
MVC Flowchart.....	5
MVC Example	5
Facade Pattern	9
Façade Example	10
MVVM Pattern	11
Benefits	11
Drawbacks.....	12
Part B (Implementation)	13
Exercise 1	13
Exercise 2	16
Exercise 3	19
Evaluation	25
References	26

Introduction

In this report, I will be showing my understanding and knowledge of the different software patterns used in the development of mobile applications. I will be creating examples of mobile applications which show the patterns. As part of this understanding, I will also be discussing the purpose of the MVVM pattern, whilst thorough evaluating its benefits and drawbacks when developing mobile applications.

To show this knowledge, I will be implementing different android applications using Android Studio and java, whilst testing the applications on an android tablet.

Part A (Understanding)

In this section, I will be investigating the use of software patterns in mobile applications, and discussing the purpose of the MVVM pattern, along with its drawbacks and benefits.

Software/Design Patterns

According to the Technology Articles Blog, design patterns are generally used and are helpful for developers to implement efficient solutions in software design, in which can help maintain complex code and avoid costly changes [1]. Protelo also says that a design pattern is a generic solution to common problems in software design, which can be repeatable in numerous solutions. They also say that a pattern can be a description or template, in which can be used to solve problems in many solutions [2].

Design patterns are not blocks of code that can be implemented into a program, but they are blueprints in which are used as a concept for solving particular problems in software engineering. Compared to an algorithm, design patterns are a higher-level descriptive solution. [3].

There are three types of patterns used in development:

1. UX Patterns – “UX design patterns are repeatable solutions to recurring design problems”. When designing applications or webpages, some of the most common features are signing up, and logging in and out of the application. UX patterns could be used for something like this to solve this common problem, to save repeating code numerous times. “When sites incorporate known patterns, users often have an easier time getting where they’re going” [4]. The same could be applied to applications, as if a similar design pattern is used, then the user will have a common idea on how to accomplish their goal in multiple applications. Similarly, Yulia and Maryna state that UX patterns help software developers and designers create workable solutions to achieve common problems with user interfaces. [5]
2. Design Patterns – “In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design”. Even though this is a broad concept in software engineering, a design pattern is different from a UX pattern, as it does not focus on just user interfaces. Design patterns are also not finished designs, so they cannot be transformed directly into code to be put to use, they help to find solutions to the common problem in software development. [6]
3. Architectural Patterns – Mallawaarachchi says that architectural patterns are reusable solutions to common problems in software architecture, which are similar to software design patterns. However, architectural patterns have a broader scope of information. [7]

MVC

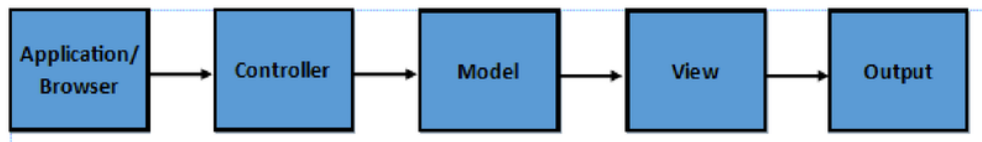
MVC, or Model View Controller, is a common design pattern that is used in application development. It is an architectural pattern in which is mainly used to develop user interfaces, where it separates the application in three parts, the model, view and controller. A benefit to this pattern is that it promotes modularity and some of the code can be reused for other applications [8].

Kevin Kononenko explains MVC through ordering drinks at a bar, where the bartender serving the drink is the controller, as once the user asks for a drink, the controller will understand and carry out the action. The toolset is the model, like the bartender's hands, mixing equipment and drinks, and the view is the finished product at the end. "The view is built out of the limited options from the model, and arranged and transmitted via the controller" [9].

According to Brainvire, the model layer defines where the data objects can be stored, and represents the data to the user, the view layer is the visual representation, or the user interface which the user can interact with, and shows the output, which comes from the model and controller. The view layer also handles user input and informs the controller of this input, so the information can be processed. The controller, as mentioned above is the brain of the MVC pattern, and creates a link from the system to the user. [10]

Codecademy say that MVC is a popular choice for software development, as it is useful for organising code, where some of the code holds the data of the software application (model), some of the code makes the app look nice (view) and the other part controls how the application functions and works (controller). They also say that MVC is useful for organising the codes core functions in their own, storage boxes, making the code cleaner and more understandable. [11]

MVC Flowchart



When the application or web browser sends a request, the controller will receive the request, process it and create a data model. This model is then passed to the view and will create an output, which will be processed by the end user [12].

Pragati Singh says that MVC is advantageous, as MVC supports asynchronous techniques. This can be beneficial to developers, as the applications developed using MVC are usually fast. Another benefit to using MVC is that any modification or changes to any of the code does not affect the entire architecture (depending on the modification). For example, if any changes are made in the model file, then only the model is affected and the entire architecture of the application will stay intact. A downside to this is then increased complexity. [13]

MVC Example

```
index.php
1 <?php
2 include_once("controller/Controller.php");
3
4 $controller = new Controller();
5 $controller->invoke();
6
7 ?>
```

This is an example of MVC which is using PHP. Here is the index file which accesses the Controller.php file from the controller directory, and instantiates it as a new controller.

```
php x Controller.php x Model.php x Animals.php x viewanimals.php x animalslist.php x
<?php
//Get the Model.php file from the model directory
include_once("model/Model.php");
//Create the controller class
class Controller {
    //Create a publicly accessible variable called model
    public $model;

    //Construct method
    public function __construct()
    {
        //New instance of the model
        $this->model = new Model();
    }

    public function invoke()
    {
        //IF statement for the animal list
        if (!isset($_GET['animals']))
        {
            // no special animals are requested, we'll show a list of all available animals
            $animals = $this->model->getAnimalsList();
            include 'view/animalslist.php';
        }
        else
        {
            // show the requested animals
            $animals = $this->model->getAnimals($_GET['animals']);
            include 'view/viewanimals.php';
        }
    }
}
?>
```

```
Model.php x Animals.php x viewanimals.php x animalslist.php x
1 <?php
2 //Call in the the animals file from the model directory
3 include_once("model/Animals.php");
4
5 class Model {
6     public function getAnimalsList()
7     {
8         // here goes some hardcoded values to simulate the database
9         return array(
10             "Dog" => new Animals("Dog", "Cocker Spaniel", "Molly"),
11             "Cat" => new Animals("Cat", "Persian", "Fluffy"),
12             "Spider" => new Animals("Spider", "Tarantula", "Black")
13         );
14     }
15
16     public function getAnimals($type)
17     {
18         // we use the previous function to get all the animals and then we return the requested one.
19         // in a real life scenario this will be done through a db select command
20         $allAnimals = $this->getAnimalsList();
21         return $allAnimals[$type];
22     }
23
24 }
25
26
27 ?>
```

```
Model.php x Animals.php x viewanimals.php x animalslist.php x
1 <?php
2
3 class Animals {
4     public $type;
5     public $breed;
6     public $name;
7
8     public function __construct($type, $breed, $name)
9     {
10         $this->type= $type;
11         $this->breed= $breed;
12         $this->name= $name;
13     }
14 }
15
16 ?>
```

```
Model.php x Animals.php x viewanimals.php x animalslist.php x
1 <html>
2 <head></head>
3 <body>
4
5 <?php
6
7     echo 'Type:' . $animals->type . '<br/>';
8     echo 'Breed:' . $animals->breed . '<br/>';
9     echo 'Name:' . $animals->name . '<br/>';
10
11 ?>
12 </body>
13 </html>
```

```
Model.php x Animals.php x viewanimals.php x animalslist.php x
1 <html>
2 <head></head>
3
4 <body>
5
6 <table>
7     <tr><td>Type</td><td>Breed</td><td>Name</td></tr>
8     <?php
9
10         foreach ($animals as $type => $animals)
11         {
12             echo '<tr><td><a href="index.php?animals='.$animals->type.'">'.
13                 $animals->type.'</a></td><td>'.$animals->breed.'</td><td>'.$animals->name.'</td></tr>';
14         }
15
16     ?>
17 </table>
18 </body>
19 </html>
```

Type	Breed	Name
Dog	Cocker Spaniel	Molly
Cat	Persian	Fluffy
Spider	Tarantula	Black

Type:	Cat
Breed:	Persian
Name:	Fluffy

These two are the outputs of the previous code, in which a table is produced listing the type of animals, and when the user selects a type of animal then it shows the details of just that animal in a different view.

Facade Pattern

Facade is a structural design pattern that is used in application development which provides a unified interface for a set of other interfaces in a sub-system (small part out of the whole application) [14].

Tutorialspoint also say that the façade pattern is a structural design pattern which is used to hide the complexities of the system, and provided an interface to the user in order for them to be able to access the system. The façade pattern adds this interface to an existing system, hiding the complexities from the user. Façade uses one class, which implements simplified methods which are needed by the client and convokes methods of the classes in the existing system. [15]

Kumar compares the Façade pattern like the face of a building, as when people walk past a building, they do not see anything inside, hiding all the complexities of the internals of the building, in which the user only sees the face of the building (interface). An example of this is the start-up of the computer system, where on boot, the system components are working together to start the PC. It could be said that a Façade is used to wrap all these complexities up, and provide one interface to the user. [16]



This diagram shows the client/user accessing a system, though a façade. This façade will then access the subsystem, in which all of the user information/requests can be handled, and only inputted through the façade, where an output will be shown. [14]

Qazi says the façade pattern useful as this software pattern shields the clients from the complexities, making it easier to use the system, and promotes loose coupling (interconnecting the components) between the clients/users and the subsystems. An example of how this façade pattern is used is when a simple interface is used on top of a complex system, like login pages on organisations like google accounts. It is also used when there are several dependences in which that exist amid the users and “implementation classes of an abstraction”. [17]

Façade Example

```
1 <?php
2 class FacadePatternExample //Facade class
3 {
4     public function __construct($str) //Construct method
5     {
6         $this->str = $str;
7         $this->_AddText();
8         $this->_AddDigits();
9         $this->_AddObject(new FacadeObject);
10        echo $this->str;
11    }
12    private function _AddText() //AddText method
13    {
14        $this->str .= ' | Adding Text ' ;
15    }
16    private function _AddDigits() //AddDigits method
17    {
18        $this->str .= ' | Adding Digits: 12345 ' ;
19    }
20    private function _AddObject(FacadeObject $facadeobject) //AddObject method with the facade object
21    {
22        $this->str .= $facadeobject->WriteObjects();
23    }
24 }
25 class FacadeObject //FacadeObject class
26 {
27     //Method to write the objects
28     public function WriteObjects()
29     {
30         return __Class__ . ' Writing Object Contents ' ;
31     }
32 }
33 //New FacadePatternExample
34 new FacadePatternExample('Facade Pattern Example Website ');
35 ?>
```

This is an example of the Façade pattern, in which displays text and digits, which is written in PHP.

MVVM Pattern

According to John Gossman, MVVM is based of MVC, and is a variation that is more suited towards modern user interface development platforms, in which the view is the responsibility of the designer, over a developer. The design of the application is usually done by a designer, as they are more graphical than a developer, in which they use HTML/XAML, other languages or IDEs (Integrated Development Environments). Also, MVVM also is used as a data binding mechanism. [18]

It is still classed as a structural design pattern; however, it is split into three different groups. Like MVC, the model holds the application data and the view hold the visual parts of the user interface, including the different elements and the controls (buttons). Unlike MVC, the last part of MVVM is the View-Model group, in which converts the model information into values which are then outputted to the view. In most cases, these are classes, so that they can be called in and the class can be referenced. [19]

Likness explains the MVVM patterns as parts of a pie, which consist of the model, view and view-model. The model is referred to the domain object, and represents real life data which the application deals with, like personal information about a user. The view, is what the users see on the application, and the only part they interact with. It defines how the data is presented, with associated behaviours and accepting input from the user, in which could be button clicks or mouse movements (or touch screen input for a mobile application). The view in MVVM will need to understand the model and view model, so it can coordinate the events of the design of the application. The view-model is the key part of MVVM as it can be compared to the controller segment of the MVC pattern. It contains methods and commands, along with points that are needed to maintain the view state, and manipulate the model depending on user interactions, and can trigger events that are contained in the view. The view-model acts as a link between the other two layers, and can also process input from the view (interface) and place it in the model, or it can call part of the model and output it to the view. [20]

Benefits

David Britch says MVVM is a very useful pattern which can be used to help clean the separation between business and presentation logic of applications from the user interface. From this, it can help to solve some development issues and make applications easier to test, maintain and keep up to modern standards [21]. Applications using this pattern can be more easily maintained because of the separation of code in multiple files, then any changes made in one will not affect the other, in which it is also easier to update in small parts so that current standards can be met [22]. When testing applications using the MVVM pattern, then it could be said that the application can be more easily tested, as the view layer of the pattern does not contain complex part of the system, then it can be easy to create unit tests for the view layer, meaning the application testing can be carried out faster and more efficiently [23]. If the model segment of the application needs to be tested, then UI automation not needed, which means that each part of the application can be more easily tested. As previously mentioned, the different parts of the solution can be maintained more easily, but not only this, MVVM allows for easy extension, as it has the ability to add new code into the right places without affecting the application as a whole [22]. Emad states that when creating these applications, developers can work along with the designers synchronously, so the developer can work on the model (back end processes) and the designer can work on the view (user interface), and the code can be repeated for similar applications, making it a useful and efficient pattern in software engineering. [24]

Drawbacks

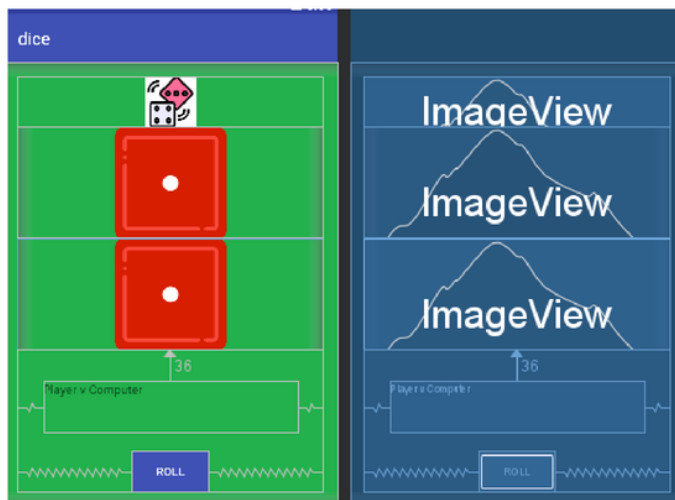
Even though there are numerous advantages, there are a few drawbacks that go along with the MVVM pattern. Due to the larger number of files, the application can be harder to debug as each piece of the application has got to be looked at [24]. For smaller applications with simple user interfaces, then MVVM can be quite “overkill” due to the complex nature that the design pattern has. [22].

Part B (Implementation)

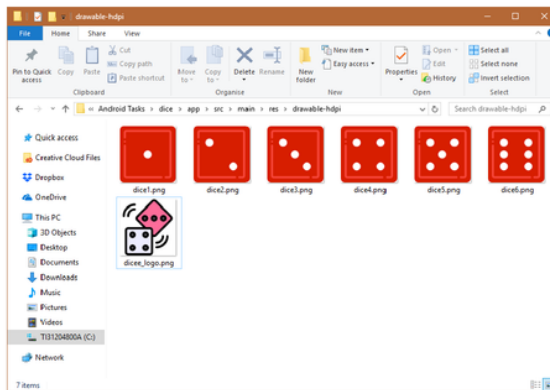
This section will contain evidence, and show understanding of android development, with the different mobile applications that have been created. All of the mobile applications created for this section will be using androids official IDE Android Studio, where java will be used to develop the functionality of the applications.

Exercise 1

In this exercise, the task was to create an application that allowed the user to roll two dice from a given application event, in which this case a button will be used to roll the dice. As an extension, one dice belongs to the player, and one belongs to the computer, and a scoring system will be developed in which will show what player has had the highest score.



This is the design view of the app, where relative and linear layouts have been used to place the images, buttons and textviews.



These are the images of the dice that are used to implement the application. These were placed in the drawable directory of the project so that they could be used in android studio. These images were sourced from FlatIcon (<https://www.flaticon.com/packs/casino-5>).

```

package com.example.ss180976.dice;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;
/*Importing from the java library the random function*/
import java.util.Random;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        /*Calling in the dice*/
        final ImageView Dice1 = findViewById(R.id.dicePlayer);
        final ImageView Dice2 = findViewById(R.id.diceComputer);

        /*Calling in the button*/
        Button myButton = findViewById(R.id.rollbutton);

        final TextView score = findViewById(R.id.Score);

        /*First dice array*/
        final int[] diceArray = {
            R.drawable.dice1,
            R.drawable.dice2,
            R.drawable.dice3,
            R.drawable.dice4,
            R.drawable.dice5,
            R.drawable.dice6
        };
        /*Second dice array*/
        final int[] dice2Array = {
            R.drawable.dice1,
            R.drawable.dice2,
            R.drawable.dice3,
            R.drawable.dice4,
            R.drawable.dice5,
            R.drawable.dice6
        };
    };
}

```

In order to carry out the different functions of the application, I had to import the different android widgets so that I could call in the different image and text views so that they could be used within the code.

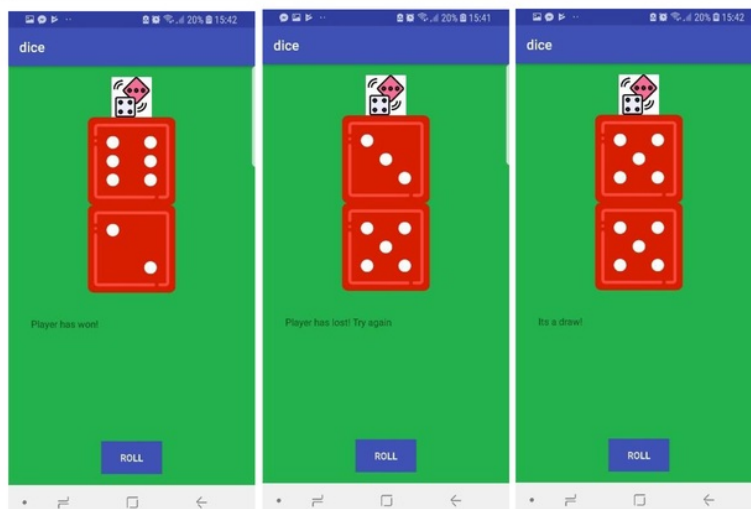
```

/*When the button is pressed, this code is carried out*/
myButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        /*This is the code for the first dice, in which a random number is
        generated.
        This random number is then stored in an integer variable.
        Using this number, the number of the integer is called and uses the
        dice array to set the image.
        */
        Random randomNumberGenerator = new Random();
        int intNumber = randomNumberGenerator.nextInt(6);
        Dice1.setImageResource(diceArray[intNumber]);

        /*This is the code for the second dice, in which another random
        number is generated.
        This random number is then stored in an integer variable.
        Using this number, the number of the integer is called and uses the
        dice array to
        set the image.
        */
        int intNumber2 = randomNumberGenerator.nextInt(6);
        Dice2.setImageResource(dice2Array[intNumber2]);

        /*These IF Statements are the handlers for who has the higher score
        IF the first dice is higher than the second dice, then the player
        has won, IF the first dice is lower than the second dice then the
        computer has won. Else it must be a draw as there are no other
        possibilities */
        if (intNumber > intNumber2){
            score.setText("Player has won!");
        }
        else if (intNumber < intNumber2){
            score.setText("Player has lost! Try again");
        }
        else{
            score.setText("It's a draw!");
        }
    }
});
}
}

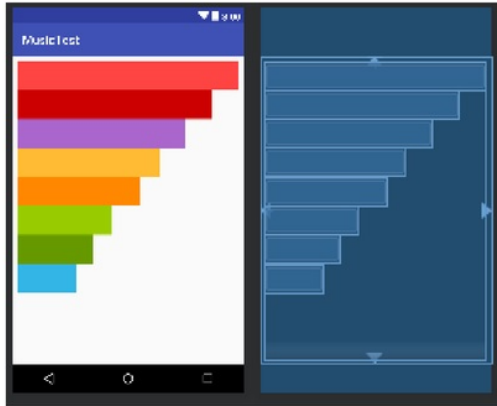
```



After testing the application on a mobile device, when the button is pressed, the dices roll and shows if the player has won, drawn or lost.

Exercise 2

In this exercise, I created a simple music player application, that allows the user to play sounds from a button press. It contains a grid with eight different buttons, in which samples are preloaded into the grid so that on button press, a different sound will play via the device's speakers. This application is a basic xylophone.



This is the design layout of the xylophone, I which there are eight different colour buttons on the application, where each one plays a different tone.

```
package com.example.ss180976.musictest;

/*Importing the appropriate libraries, where the audio manager and SoundPool is
imported*/
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.media.AudioManager;
import android.media.SoundPool;
import android.view.View;

public class MainActivity extends AppCompatActivity {
    /*Declaring the SoundPool and naming it mySoundPool*/
    SoundPool mySoundPool;

    /*Declaring integer variables for the sounds*/
    private int intSoundID1;
    private int intSoundID2;
    private int intSoundID3;
    private int intSoundID4;
    private int intSoundID5;
    private int intSoundID6;
    private int intSoundID7;
    private int intSoundID8;

    /*Variables for the sound properties, where there are eight sounds, left and
    right volume is equal, sounds do not loop and initialising the priority at
    zero*/
    private final int NR_OF_SIMULTANEOUS_SOUNDS = 8;
    private final float LEFT_VOLUME = 1.0f;
    private final float RIGHT_VOLUME = 1.0f;
    private final int NO_LOOP = 0;
    private final int PRIORITY = 0;
    private final float NORMAL_PLAY_RATE = 1.0f;
```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    /*Creating a new instance of SoundPool*/
    mySoundPool = new SoundPool(NR_OF_SIMULTANEOUS_SOUNDS,
    AudioManager.STREAM_MUSIC, 0);

    /*Initialising the integer variables created above to the different sound
    notes with an equal priority of 1. These notes are stored in the raw
    directory. */
    intSoundID1 = mySoundPool.load(getApplicationContext(), R.raw.note1, 1);
    intSoundID2 = mySoundPool.load(getApplicationContext(), R.raw.note2, 1);
    intSoundID3 = mySoundPool.load(getApplicationContext(), R.raw.note3, 1);
    intSoundID4 = mySoundPool.load(getApplicationContext(), R.raw.note4, 1);
    intSoundID5 = mySoundPool.load(getApplicationContext(), R.raw.note5, 1);
    intSoundID6 = mySoundPool.load(getApplicationContext(), R.raw.note6, 1);
    intSoundID7 = mySoundPool.load(getApplicationContext(), R.raw.note7, 1);
    intSoundID8 = mySoundPool.load(getApplicationContext(), R.raw.note8, 1);

}

```

```

/*Creating methods for each sound note, where all the variables are called in
for the sound number, volume, priority and the play rate. These then play when
they are linked to the buttons.*/
public void play1(View v)
{
    mySoundPool.play(intSoundID1, LEFT_VOLUME, RIGHT_VOLUME, PRIORITY, NO_LOOP,
    NORMAL_PLAY_RATE);
}

public void play2(View v)
{
    mySoundPool.play(intSoundID2, LEFT_VOLUME, RIGHT_VOLUME, PRIORITY, NO_LOOP,
    NORMAL_PLAY_RATE);
}

public void play3(View v)
{
    mySoundPool.play(intSoundID3, LEFT_VOLUME, RIGHT_VOLUME, PRIORITY, NO_LOOP,
    NORMAL_PLAY_RATE);
}

public void play4(View v)
{
    mySoundPool.play(intSoundID4, LEFT_VOLUME, RIGHT_VOLUME, PRIORITY, NO_LOOP,
    NORMAL_PLAY_RATE);
}

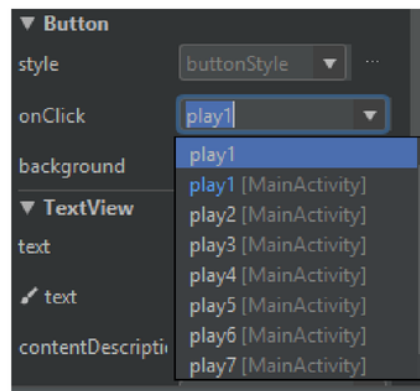
public void play5(View v)
{
    mySoundPool.play(intSoundID5, LEFT_VOLUME, RIGHT_VOLUME, PRIORITY, NO_LOOP,
    NORMAL_PLAY_RATE);
}

public void play6(View v)
{
    mySoundPool.play(intSoundID6, LEFT_VOLUME, RIGHT_VOLUME, PRIORITY, NO_LOOP,
    NORMAL_PLAY_RATE);
}

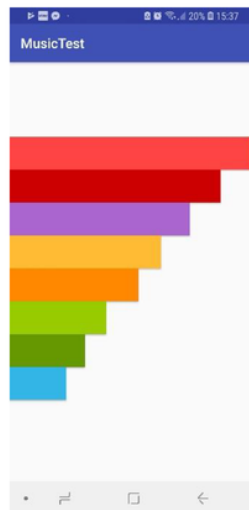
public void play7(View v)
{
    mySoundPool.play(intSoundID7, LEFT_VOLUME, RIGHT_VOLUME, PRIORITY, NO_LOOP,
    NORMAL_PLAY_RATE);
}

public void play8(View v)
{
    mySoundPool.play(intSoundID8, LEFT_VOLUME, RIGHT_VOLUME, PRIORITY, NO_LOOP,
    NORMAL_PLAY_RATE);
}
}

```



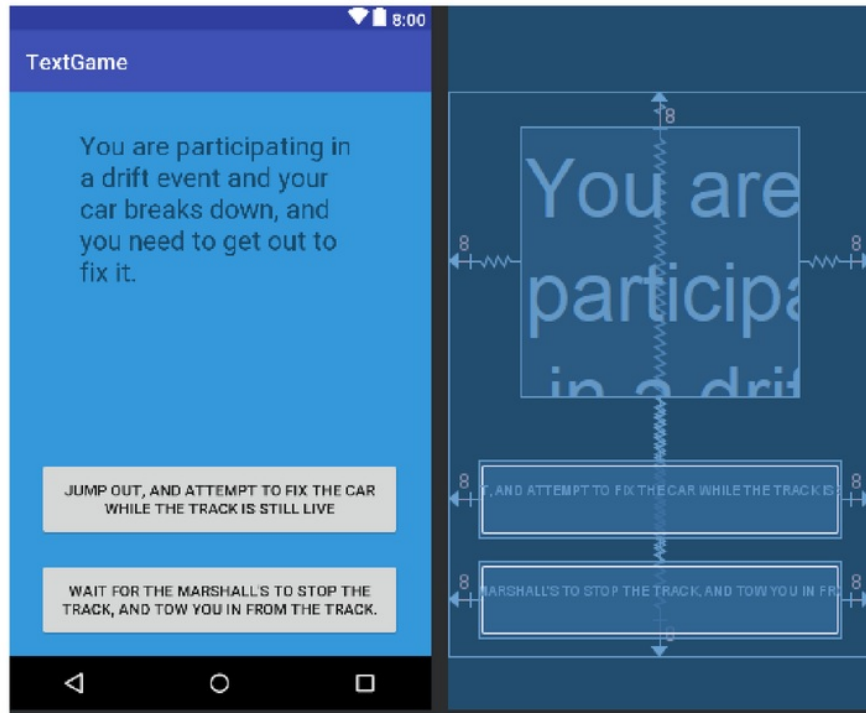
To link the sounds to the buttons, I then used the attributes menu in android studio where I could then link each sound to the onClick event on each button. Each of the buttons and sounds were labelled, so that each button could be linked to the appropriate note.



This is the application on the mobile device, where you can see the buttons. On testing, each of the sounds played on the device.

Exercise 3

This exercise is based on a text-based role-playing game, which has been created in android studio. The user has a choice of events in which will lead them through the game, and end when the story is over.



This is the design view for the game, where the story is shown in a text view, and two buttons which display the predetermined options for the game.

```
<resources>
  <string name="app_name">TextGame</string>

  <string name="A1_Story">You are participating in a drift event and your car breaks down, and you need to get out to fix it.</string>
  <string name="A1_Ans1">Jump out, and attempt to fix the car while the track is still live</string>
  <string name="A1_Ans2">Wait for the marshall's to stop the track, and tow you in from the track.</string>

  <string name="A2_Story">Whilst waiting for the marshall, a car hits you.</string>
  <string name="A2_Ans1">You jump out, and open the bonnet to check the damage.</string>
  <string name="A2_Ans2">You step off the track and wait.</string>

  <string name="A3_Story">While looking in the engine, you can hear cars coming around the corner.</string>
  <string name="A3_Ans1">Move out of the way and leave the car.</string>
  <string name="A3_Ans2">The car comes around the corner, and hits you. Dead!</string>

  <string name="A4_End">The marshall's come over and tow the car in.</string>
  <string name="A5_End">You died, and the car has been donated to the scrap yard.</string>
  <string name="A6_End">The car is not fixable, and have to leave the event early.</string>
</resources>
```

In the string's xml file, I created strings which contained the story lines and answer options.

```

/**
 * Created by SS180976 on 13/11/2018.
 */

public class clsStoryProgress {
    /*Variables for the ID and answer*/
    private int mStoryID;
    private int mAnswer;

    /*Calling in the above variables and storing them in the story resource ID and answer*/
    public clsStoryProgress(int StoryResourceID, int intAnswer){
        mStoryID = StoryResourceID;
        mAnswer = intAnswer;
    }

    /*Method to return the story ID and store in the getStoryID variable*/
    public int getStoryID() {return mStoryID;}

    public void setmStoryID(int StoryID) {mStoryID = StoryID;}
    /*Method to return the answer and store in isAnswer variable*/
    public int isAnswer() {return mAnswer;}

    public void setmAnswer(int answer) {mAnswer = answer;}
}

```

```

package com.example.SS180976.textgame;

import android.app.AlertDialog;
import android.content.DialogInterface;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    /*Variables for the buttons and textview*/
    private Button mTopButton;
    private Button mBottomButton;
    private TextView mTextView;
    /*Variables for the story and button indexes*/
    private int mStoryIndex;
    private int mStory;
    private int mTopButtonIndex;
    private int mBottomButtonIndex;
    /*This uses the story progress class that I created, in which uses the strings
    created for the story, and is given an answer ID*/
    private clsStoryProgress[] mStoryBank = new clsStoryProgress[]{
        new clsStoryProgress(R.string.A1_Story, 1),
        new clsStoryProgress(R.string.A1_Ans1, 2),
        new clsStoryProgress(R.string.A1_Ans2, 3),
        new clsStoryProgress(R.string.A2_Story, 4),
        new clsStoryProgress(R.string.A2_Ans1, 5),
        new clsStoryProgress(R.string.A2_Ans2, 6),
        new clsStoryProgress(R.string.A3_Story, 7),
        new clsStoryProgress(R.string.A3_Ans1, 8),
        new clsStoryProgress(R.string.A3_Ans2, 9),
        new clsStoryProgress(R.string.A4_End, 10),
        new clsStoryProgress(R.string.A5_End, 11),
        new clsStoryProgress(R.string.A6_End, 12)
    };
}

```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    /*If the saved instance state is not equal to null, then get the store
    index*/
    if (savedInstanceState != null){
        mStoryIndex = savedInstanceState.getInt("StoryIndex");
    }
    else
    {
        /*Else initialise the story index and button indexes*/
        mStoryIndex = 0;
        mTopButtonIndex = 1;
        mBottomButtonIndex = 2;
    }
    /*Call in the buttons and text view*/
    mTopButton = (Button) findViewById(R.id.button);
    mBottomButton = (Button) findViewById(R.id.button2);
    mTextView = (TextView) findViewById(R.id.textView);

    mStory = mStoryBank[mStoryIndex].getStoryID();
    mTextView.setText(mStory);

    /*When the user presses the top button, check the story path and update*/
    mTopButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            checkStoryPath(1);
            UpdateStory();
        }
    });
    /*When the user presses the bottom button then check the story path and
    update story*/
    mBottomButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            checkStoryPath(2);
            UpdateStory();
        }
    });
}

```

```

/*Method which checks the story path using the user selection variable*/
private void checkStoryPath(int userSelection)
{
    int ChosenAnswer = mStoryBank[mStoryIndex].isAnswer();
    /*If the user selects option one*/
    if (userSelection == 1)
    {
        /*If the story index is equal to 0 or 3 */
        if (mStoryIndex == 0 || mStoryIndex == 3)
        {
            /*Put the story index to 5
            * Top button to 7
            * Bottom button to 8
            * Which advances the story*/
            mStoryIndex = 5;
            mTopButtonIndex = 7;
            mBottomButtonIndex = 8;
        }
        /*IF the story index equals to six, then put the story index to ten and
        end the story*/
        else if (mStoryIndex == 6)
        {
            mStoryIndex = 10;
            End_Of_story();
        }
    }
    else
    {
        /*IF the story index equals zero, put the story index up to two
        * top button index to four
        * and the bottom button to five*/
        if (mStoryIndex == 0)
        {
            mStoryIndex = 2;
            mTopButtonIndex = 4;
            mBottomButtonIndex = 5;

            /*else if the story index equals three, then put the story index to
            * eight and end the story*/
        } else if (mStoryIndex == 3)
        {
            mStoryIndex = 8;
            End_Of_story();
        }
        /*else if the story index equals six, then put the story index to
        * nine and end the story*/
        else if (mStoryIndex == 6)
        {
            mStoryIndex = 9;
            End_Of_story();
        }
    }
}
}

```

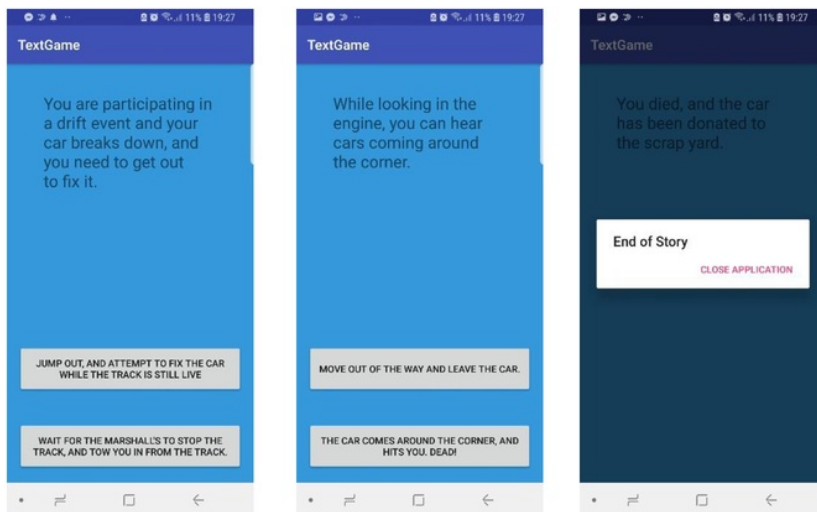
```

@Override
public void onSaveInstanceState(Bundle outState){
    outState.putInt("Story Index", mStoryIndex);
    super.onSaveInstanceState(outState);
}
/*End story method*/
private void End_Of_story(){
    mTopButton.setVisibility(View.GONE);
    mBottomButton.setVisibility(View.GONE);
    /*Create an alert dialog which gives the user the option to close the
    application*/
    AlertDialog.Builder alert = new AlertDialog.Builder(this);
    alert.setTitle("End of Story");
    alert.setCancelable(false);
    alert.setPositiveButton("Close Application", new
DialogInterface.OnClickListener() {
        @Override /*When the user presses close application, the application
        finishes*/
        public void onClick(DialogInterface dialogInterface, int i) {
            finish();
        }
    });
    alert.show(); /*Show the dialog when the story ends*/
}
/*Method to update the story*/
private void UpdateStory()
{
    /*Add one to the story index and get the length of the story bank*/
    mStoryIndex = (mStoryIndex + 1) % mStoryBank.length;
    /*If the story index is equal to zero then end story*/
    if (mStoryIndex == 0){
        End_Of_story();
    }

    /*Set the text view to the current part of the story*/
    mStory = mStoryBank[mStoryIndex].getStoryID();
    mTextView.setText(mStory);

    /*Set the top button to the one of the options*/
    mStory = mStoryBank[mTopButtonIndex].getStoryID();
    mTopButton.setText(mStory);
    /*Set the bottom button to the one of the options*/
    mStory = mStoryBank[mBottomButtonIndex].getStoryID();
    mBottomButton.setText(mStory);
}
}

```



After testing the application, the game navigates through the different storylines, in which all of the buttons achieve the correct result, and ends the application once it reaches the end point.

Evaluation

During this report there has been an investigation into the use of MVC and façade software patterns, along with a discussion on MVVM, with the uses, benefits and drawbacks of the pattern when developing mobile applications. In the future, I would investigate further into the different software patterns, and put into use by designing applications that use these underlying concepts of software engineering.

After creating the android applications, I would also develop my experience and knowledge in android development and java. This is because during these exercises, only basic features were used when developing the applications, and if I was more knowledgeable during these, then the applications would have had more features and would have been more effective in reaching today's standard of applications.

References

- [1] Technology Articles Blog, "Importance of Design Patterns," 20 June 2016. [Online]. Available: <http://www.breakingnewsroll.com/bn/2016/06/20/importance-of-design-patterns/>. [Accessed 05 November 2018].
- [2] Protelo, "Why Design Patterns are Important in Software Engineering," [Online]. Available: <https://www.proteloinc.com/blog/design-patterns-important-software-engineering/>. [Accessed 05 November 2018].
- [3] Refactoring Guru, "What's a Design Pattern," [Online]. Available: <https://refactoring.guru/design-patterns/what-is-pattern>. [Accessed 05 November 2018].
- [4] D. Schlecht, "A Step by Step Guide to User Experience (UX) Patterns," 1 October 2018. [Online]. Available: <https://careerfoundry.com/en/blog/ux-design/ux-patterns-why-use-them/>. [Accessed 05 November 2018].
- [5] Y. Z and M. Z, "Devising Interfaces with UX Design Patterns," [Online]. Available: <https://rubygarage.org/blog/ux-design-patterns>. [Accessed 05 November 2018].
- [6] Source Making, "Design Patterns," [Online]. Available: https://sourcemaking.com/design_patterns. [Accessed 05 November 2018].
- [7] V. Mallawaarachchi, "10 Common Software Architectural Patterns in a nutshell," 04 September 2017. [Online]. Available: <https://towardsdatascience.com/10-common-software-architectural-patterns-in-a-nutshell-a0b47a1e9013>. [Accessed 05 November 2018].
- [8] MDN Web Docs, "MVC Architecture," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/Apps/Fundamentals/Modern_web_app_architecture/MVC_architecture. [Accessed 05 November 2018].
- [9] K. Kononenko, "freeCodeCamp," 2 May 2016. [Online]. Available: <https://medium.freecodecamp.org/model-view-controller-mvc-explained-through-ordering-drinks-at-the-bar-efcba6255053>. [Accessed 05 November 2018].
- [10] Brainvire, "Six Benefits of Using MVC Model for Effective Web Application Development," 28 April 2016. [Online]. Available: <https://www.brainvire.com/six-benefits-of-using-mvc-model-for-effective-web-application-development/>. [Accessed 05 November 2018].
- [11] Codecademy, "MVC: Model, View, Controller," [Online]. Available: <https://www.codecademy.com/articles/mvc>. [Accessed 05 November 2018].
- [12] Tutorialspoint, "MVC Framework - Architecture," [Online]. Available: https://www.tutorialspoint.com/mvc_framework/mvc_framework_architecture.htm. [Accessed 05 November 2018].
- [13] P. Singh, "Difference between MVC and MVVM," 28 July 2017. [Online]. Available: <https://medium.com/@erpragatisingh/difference-between-mvc-and-mvvm-456ec67181f6>. [Accessed 17 November 2018].

- [14] SourceMaking, "Facade Design Pattern," [Online]. Available:
https://sourcemaking.com/design_patterns/facade. [Accessed 12 November 2018].
- [15] Tutorialspoint, "Design Patterns - Facade Pattern," [Online]. Available:
https://www.tutorialspoint.com/design_pattern/facade_pattern.htm. [Accessed 12 November 2018].
- [16] S. Kumar, "Facade Design Pattern | Introduction," [Online]. Available:
<https://www.geeksforgeeks.org/facade-design-pattern-introduction/>. [Accessed 12 November 2018].
- [17] M. Qazi, "Design Pattern - Facade Pattern," 30 April 2015. [Online]. Available:
<https://www.slideshare.net/mudasirqazi00/design-patterns-facade-pattern-47619071>.
[Accessed 12 November 2018].
- [18] J. Gossman, "Introduction to Model/View/ViewModel pattern for building WPF apps," 08 October 2005. [Online]. Available:
<https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps/>. [Accessed 12 November 2018].
- [19] J. Strawn, "Design Patterns by Tutorials: MVVM," 25 April 2018. [Online]. Available:
<https://www.raywenderlich.com/34-design-patterns-by-tutorials-mvvm>. [Accessed 12 November 2018].
- [20] J. Likness, "Model-View-ViewModel (MVVM) Explained," 24 April 2014. [Online]. Available:
<https://www.wintellect.com/model-view-viewmodel-mvvm-explained/>. [Accessed 12 November 2018].
- [21] D. Britch, "Enterprise Application Patterns using Xamarin.Forms," in *Enterprise Application Patterns using Xamarin.Forms*, Washington, Microsoft Corporation, 2017, p. 24.
- [22] Tutorialspoint, "MVVM Advantages," [Online]. Available:
https://www.tutorialspoint.com/mvvm/mvvm_advantages.htm. [Accessed 12 November 2018].
- [23] B. Jacobs, "Benefits of Model-View-ViewModel," 24 November 2016. [Online]. Available:
<https://cocoacasts.com/what-are-the-benefits-of-model-view-viewmodel>. [Accessed 12 November 2018].
- [24] A. Emad, "MVVM," 25 September 2014. [Online]. Available:
<https://www.slideshare.net/AhmedEmad13/mvvm-model-view-view-model>. [Accessed 12 November 2018].