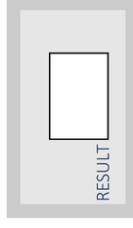


Daniel Wyn Davies Design Patterns

by Daniel Davies

FILE	11754_DANIEL_DAVIES_DANIEL_WYN_DAVIES_DESIGN_PATTERNS_164031_796469560.DOCX		
TIME SUBMITTED	19-DEC-2018 10:37PM (UTC+0000)	WORD COUNT	5655
SUBMISSION ID	97788205	CHARACTER COUNT	30601



School of Applied Computing ASSIGNMENT SPECIFICATION

Student name:	Daniel Wyn Davies	Student number:	1701700
Programme:	Applied Computing- Application Development- DUM5002		
Lecturer:	DANIEL DAVIES	Internal Verifier:	Denise Hudson
Assignment Title:	Design Patterns		
Hand Out Date:		Submission deadline:	21 st December 2018
Referencing:	In the main body of your submission you must give credit to authors on whose research your work is based. Append to your submission a reference list that indicates the books, articles, etc. that you have read or quoted in order to complete this assignment (e.g. for books: surname of author and initials, year of publication, title of book, edition, publisher: place of publication).		
Disclosure:	I declare that this assignment is all my own work and that I have acknowledged all materials used from the published or unpublished works of other people. All references have been duly cited.		
Student's Signature:	<i>Hardcopy assignments will not be accepted without a signature here</i> D.W.Davies	Date:	19/12/18
Late submission will result in a late penalty mark, as follows: Up to one week late, maximum mark of 40% for first attempts. No late submission for resit work. (Academic Quality Handbook 2016/17 7.5 (5)) Only the Extenuating Circumstances Committee may grant an extension.			

BLANK PAGE

MODULE CODE: DUM5002

Application Development- Design Patterns

MODULE TUTOR: DANIEL DAVIES

CREATED BY: DANIEL WYN DAVIES
STUDENT P NUMBER: 1701700
DECEMBER 2018

Contents

Introduction	5
Design Patterns	6
The Façade Pattern	6
MVC- Model View Controller	8
MVVM - Model-View-ViewModel	10
Conclusion	11
Evaluation	11
Exercises	13
Exercise 1-Dice Game	13
Layout View	13
Dice Game Code	15
Exercise 2 – Soundboard/Launch Pad	17
Design View	17
Launchpad code	18
Exercise 3- Text Adventure	21
Design view	21
Strings.xml	22
clsStoryProgress	23
Application	30
References	31
Appendix	32
Testing	32
Dice Application	32
Launchpad/Soundboard	41
Adventure Game	51

Introduction

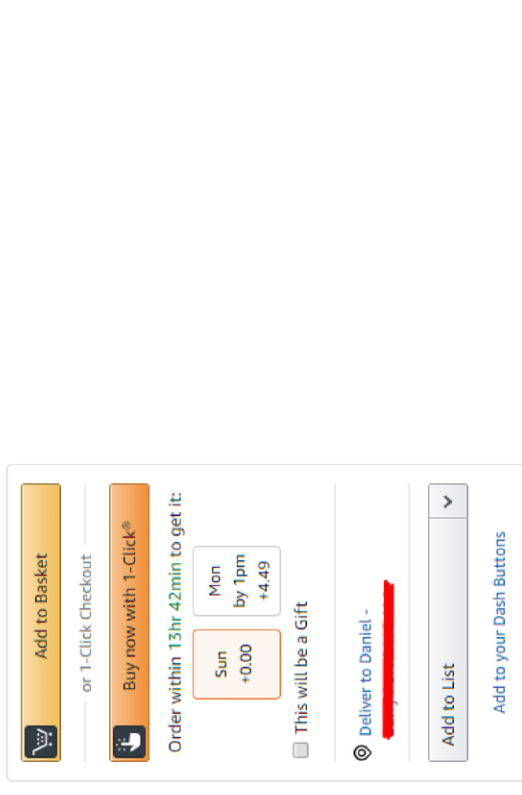
This assignment is split into two parts, one which covers and tests my understanding of software design patterns used during the development of mobile applications, including two architectural patterns or design patterns using examples to demonstrate their use. Also discussing UX design patterns, Architectural design patterns and software design patterns within my explanation. Considering how the MVVM pattern has become an important pattern within application development I have been asked to consider the benefits and drawbacks of using this approach when developing mobile applications. The second part of the assignment is testing my ability to create applications in order to satisfy the condition stated in the brief, indicating how I created them and providing testing evidence for the code.

Design Patterns

To fully understand the various design and architectural patterns I have been asked to investigate I first need an explanation of what a design pattern is. According to Christopher G Lasater [1] Design patterns are tools that are used to improve existing code, allowing for written code to be easier to implement, build and maintain. So, a design pattern in this instance would allow for the developer to become more efficient whilst developing their code as they would standardize programming tasks into recognizable forms, improving the developers design skills. Design patterns tend to be depended on various aspects of object-oriented languages such as encapsulation and polymorphism. Kyle Mew [2] indicates that whilst patterns can be used to improve efficiency whilst developing code, it is not a guarantee but there are many advantages to applying patterns to code, such as providing software structures that can be reused to solve other similar problems, and can add layers of abstraction that can allow for code that is being developed to be modified to be changed with comparative ease. Dinsh Rajput [3] states that design patterns are split into separate categories. Creational Design Pattern, programs based on this pattern are considered more flexible in deciding object creation, Behavioural Design Pattern, which characterize the way that object and classes interact, and Structural Design Patterns, which deal with the composition of classes or objects. The Facade Pattern is an example of a Structural Design Pattern.

The Façade Pattern

The role of the Façade pattern is to provide high-level views of systems and subsystems that have details that are hidden from users or as Chetan Girdhar [4] indicates the main responsibility of a Façade is to wrap up a complex group of subsystems so that it can provide a pleasing look for users to interact with, an example of this in use would be a supermarket shopping cart or a login page, so lets use amazons one click shopping as an illustration.



When ordering from amazon you need to individually go through each of the systems they have in place on their website so you would sign in using their login systems, verifying you have an account, then press add to basket on the item you desire, which is another separate method/system on their site and finally confirm the checkout and enter the delivery address/billing information required. One click checkout in this instance would allow the user to circumvent seeing all those individual steps and end up with just the payment method confirmation and the address confirmation. What does this mean exactly? Considering the main point of a façade is to hide the full system underneath this would mean that the one click shopping options will perform each of the individual tasks I listed before whilst only showing the viewer the payment method and delivery address systems as they are required, whilst it still performs the add to basket and confirm checkout the user does not necessarily see these features as the one click shop hides them as they are not necessary for the user to see in that instance. So from this we can determine that hiding detail from users is one of the key details that the façade pattern covers, however according to Judith Bishop [5] What makes the façade pattern different from the Decorator or Adapter patterns is that the interface it builds can be entirely new, and not coupled to existing requirements, meaning whilst it can use already existing interfaces in most cases it will be using subsystem whilst making an interface that is entirely independent from the original subsystem, it just uses methods from those systems to fulfil it's role within the entire system. Eric Freeman [6] states that a façade does not encapsulate the subsystem classes and merely provide an interface to their functionality, the subsystems are still available outside of the façade, meaning that a façade can be considered outside of the main system meaning that whilst the façade needs access to the desired subsystems the subsystems can function without the need of the façade, this is useful as it allows for the façade to decouple a client from the subsystem, so there is potentially less critical systems that the client can potentially interact with, meaning that it can add to the security of a page as the user sees and interacts with less of the systems that the application has within it. So to put that in context with amazon, the user no longer has any need to see the basket as the one click option decides that whilst the task needs to be completed the user does not need to see the individual process and can just show the user the billing and shipping system, simplifying all those separate tasks under one button/façade making the usage of the system by the client much easier as there are less errors that could potentially be made by the user.

MVC- Model View Controller

MVC(Model view controller) pattern is a software architecture pattern, to gain an Understanding of MVC we must split the pattern into its individual components, the Model, the View and the Controller, [7] F.E Suhabudin states that MVC is a pattern which is utilised to separate the business logic from the user interface, the model represents the information of the application and the rules used to manipulate data, the view is the elements of user interface on the application such as text and or images and the controller is responsible for the communication between the user and the application, so button presses and other inputs are handled by the model. To see an example of how an MVC pattern application works I have used an online IDE in order to create a simple application that will display student names and their roll numbers.

```
1 class Student
2 {
3     private String rollNo;
4     private String name;
5
6     public String getRollNo() //this string will get the roll number and set is as the value contained withi
7     {
8         return rollNo;
9     }
10
11     public void setRollNo(String rollNo)
12     {
13         this.rollNo = rollNo; //sets the current rollNo to the one held in rollNo currently
14     }
15
16     public String getName() //gets the students name
17     {
18         return name;
19     }
20
21     public void setName(String name) //sets the student name to the one it got from before
22     {
23         this.name = name;
24     }
25 }
26
```

This student class will Get and Set all the values needed for the application to work correctly.

```
27 class StudentView
28 {
29     public void printStudentDetails(String studentName, String studentRollNo) //prints Student Details
30     {
31         System.out.println("Student: ");
32         System.out.println("Name: " + studentName);
33         System.out.println("Roll No: " + studentRollNo);
34     }
35 }
36
```

This Class represents the “View” and will display the output to the user.

```

37 class StudentController
38 {
39     private Student model; //these are set to private as they are only used in this class
40     private StudentView view;
41
42     public StudentController(Student model, StudentView view)
43     {
44         this.model = model;
45         this.view = view;
46     }
47
48     public void setStudentName(String name)
49     {
50         model.setName(name); //setting the student name
51     }
52
53     public String getStudentName()
54     {
55         return model.getName();
56     }
57
58     public void setStudentRollNo(String rollNo)
59     {
60         model.setRollNo(rollNo);
61     }
62
63     public String getStudentRollNo()
64     {
65         return model.getRollNo();
66     }
67
68     public void updateView()
69     {
70         view.printStudentDetails(model.getName(), model.getRollNo()); //updates the view with the information
71     }
72 }
73

```

This class is the Controller and will handle getting and setting the values within each variable.

```

class MVCPattern
{
    public static void main(String[] args)
    {
        Student model = retrieveStudentFromDatabase(); //Retrieves the student data
        StudentView view = new StudentView();
        StudentController controller = new StudentController(model, view);
        //creating a new controller using the model and view
        controller.updateView(); //Controller updates the view
        controller.setStudentName("Pulsee Work");
        controller.updateView();
    }
    private static Student retrieveStudentFromDatabase()
    {
        Student student = new Student();
        student.setName("Tryna Getit");
        student.setRollNo("3370107");
        return student;
    }
}

```

And finally, this works as the pattern itself and will run through the program, retrieving the individual data needed and then following the procedure of MVC. Outputting the following:

Student:

Name: Tryna Getit

Roll No: 337U167

Student:

Name: Pulese Work

Roll No: 337U167

So, to put what happens during this example into words, the user in this scenario indicates they are a new student to the controller, which notifies the model about this new student, and their name and roll number. The model then passes the changes on to the View and displays them to the user. Meaning that the application I created would be a general and reusable solution allowing for the architecture of the application to be modified and potentially used in another similar application.

MVC is useful as it allows for various bits of a program to be separated from each other. Erich Gamma [8] indicates that MVC decouples Views and Models by establishing a subscribe protocol between them. Meaning that a view must reflect the model, so if the model changes data it notifies the view and will have the view change as a result, so in my example it would be fairly simple to change students to cars or animals, or change the student data, as I would only need to change variable names and the inputs themselves the rest of the code will remain either the same or with small changes. Jose Rolando Guay Paz [9] has some interesting insights into the benefits of utilising MVC, the pattern itself makes application complexity as it clearly separates the functionality of the application into the three core parts discussed before. Considering ASP.NET MVC web applications that do not use View state or server-based forms, it makes MVC ideal as it can allow developers to have full control over the application. This could potentially mean that multiple programmers could be needed as the developer might need knowledge of both Client side and HTML code meaning the complexity of the application could increase as a result.

MVVM - Model-View-ViewModel

The MVVM pattern is very similar to that of the MVC model as both are considered Architectural patterns and have model and views, however the MVVM instead is based on the Model, View, and ViewModel. According to Godfrey Nolan [10] the main difference between MVC and MVVM is that the model should contain no logic that is specific to the view, only logic that's necessary to provide minimal API to the ViewModel, so as an example the model in this scenario would only need to add/delete information whereas the ViewModel handles the needs of the view meaning the ViewModel is more akin to that of a specialized Controller which acts as a data converter. Ryan Vice [11] states that under MVVM the view is still responsible for displaying data, but is also responsible for collecting user input, and passing said information onto the ViewModel, the view can also be responsible for mapping views to View Models. Gary Hall [12] summarises that the model is neither dependant on either the view or ViewModel, it is alone and isolated from both of the other aspects of MVVM, however the ViewModel is dependant on the Model but not the view, and the view is dependant on the ViewModel and not the model. This is useful as the View is abstracted as a result of this interaction meaning that the quantity of business logic required in the code is lessened to an extent. According to Addy Osmani [13] MVVM can be overkill when dealing with an application that has a simpler UI as it can add extra unnecessary strain on developers to integrate MVVM effectively, however the ViewModel component of applications made with MVVM pattern can be considered easier to test than event driven code and can potentially be tested without concerning UI automation or interaction. Considering that a Binder is used in most uses of MVVM, whilst it can

free up a developer from having to write code that links the Model View and the View. John Gossman [14] the data-binding performance can be considered good, the issue comes with how much book-keeping data these binders tend to create, considering this is in reference to Android development most phones have a very limited amount of RAM to work with so if it's too strenuous on that front the "good performance" that Gossman spoke about might be changed to a lesser quality due to the data load of the binders. Gossman also reiterates that MVVM can be overkill for simple UI designs and that data binding can be much harder to debug than the ViewModel increasing the overall load on the developer so it's not necessarily suited to smaller developer teams creating a UI for a smaller scale application as the added advantages of using MVVM are outweighed by the complexity of developing a UI to adhere to MVVM and the disadvantages associated with it.

Conclusion

To conclude this report the Façade pattern is a design pattern which is used to create facades within programs in order to simplify a large body of code into one simplified interface, turning a multi stage process such as a checkout on a supermarket into a single button press. Using facades can allow for complex systems to have simple interfaces and can allow for the complexities of the code to be hidden and reduce the dependencies of outside code on the innerworkings of the system meaning the façade can allow for a much more flexible system. The MVC pattern is utilised to separate the concerns of the application into the Model, which handles the logic of the application which will update the controller on data changes, the View, which will visualise the data that the model contains, and the controller which will control data flow of the model and update the view with said data, separating these functions can allow for systems to be flexible enough that you can potentially re-use and adjust MVC application to suit the need of a similar application, so a student registry using MVC could potentially be modified into a animal registry MVC program or a car manufacturer register. MVVM can be considered useful as it once again separates the concerns of an application meaning the Model, View and View Model are responsible for different processes within an application potentially reducing the strain on developers who need to debug and test their code when compared with event driven patterns, however the MVVM pattern has limited success when dealing with a simpler UI structure and can introduce higher memory usage when compared to application that are not using MVVM as the binders can use and require large amounts of memory in order to utilise their binders successfully.

Evaluation

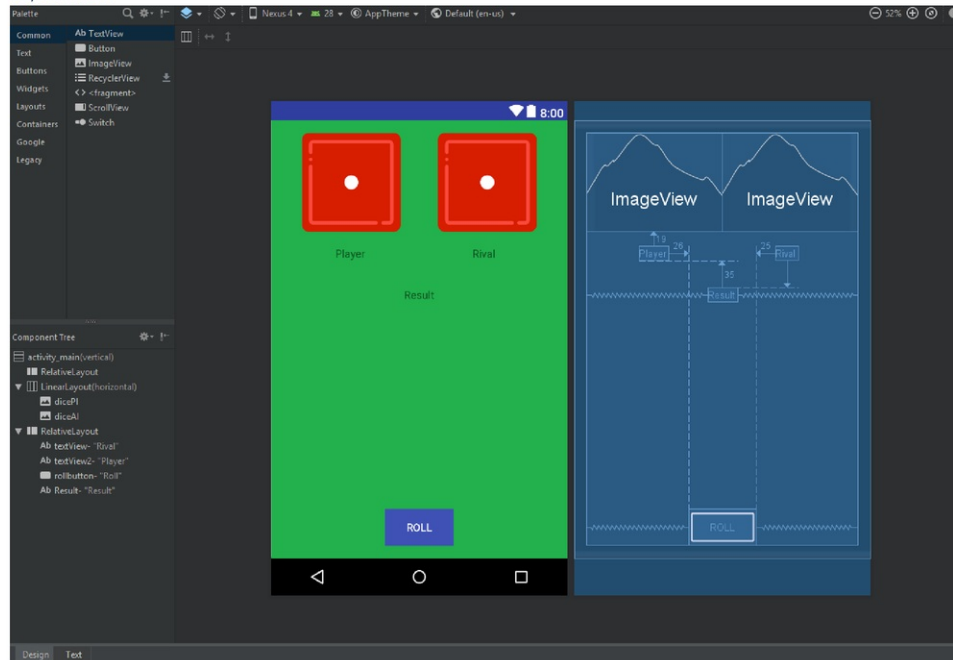
Whilst each of the programs I have created are functional there are a myriad of ways in which they could be improved with more work, for instance I could have implemented a Tally system for the dice game rather than just Winner, Draw and Loser, to make each subsequent win or loss have meaning to the user. Whilst I was creating the Launch Pad application I did attempt to create a program that was based on the user uploading the music themselves but after failing to get it working to a degree I found acceptable I settled for it just playing sound samples when the button pressed, I do acknowledge that the design is a tad broken at the moment but I am not too familiar with getting layouts to work properly and that is a consideration I would have if I were to re-complete this task, as the design is fine in the layout view but when deployed the design messes up and goes slightly astray from the intended layout. The text game in its current form is very inefficient with its code, if I were to complete this task or offer improvements, I would use case statements to prevent the large chain of if statements and nested if statements the current iteration has. With regards to the testing I feel as though I should have used an emulator to test on an older device than my own, but due to issues with android studio and my PC I was unable to test the program using

something other than my own mobile device as the emulators would encounter errors when installing themselves.

Exercises

Exercise 1-Dice Game

Layout View



Here is the layout of the application, it utilises 2 image views in order to place the dice images (which I acquired from the college Moodle) onto the application. 3 Text views which will indicate the player's dice, the Rival dice and finally the result of who won the dice roll.

Dice Game Code

```
package com.example.od1976.dice;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    //Creating a new public class called main Activity

    @Override//This becomes the existing method
    protected void onCreate(Bundle savedInstanceState) {
        //Creates an instance of the dice game
        super.onCreate(savedInstanceState); //if you close and re-open the activity this will ensure it is set to default
        setContentView(R.layout.activity_main); //Displays the application

        final ImageView Dice1 = findViewById(R.id.diceP1); //Creates a new instance of the players dice
        final ImageView Dice2 = findViewById(R.id.diceA1); //creates a new instance of the AI dice
        final TextView Result = (TextView) findViewById(R.id.Result); //This creates a new Result TextView

        Button myButton = findViewById(R.id.rollbutton); //creates a new mybutton variable

        final int[] diceArray = {
            R.drawable.dice1,
            R.drawable.dice2,
            R.drawable.dice3,
            R.drawable.dice4,
            R.drawable.dice5,
            R.drawable.dice6
        };

        //Creates a new dice array which will be used to determine which dice image is displayed on the application
        myButton.setOnClickListener(new View.OnClickListener() {
            //Creates the on click method
            Random randomNumberGenerator = new Random(); //Creates a new random number Variable

            int intNumberP1 = randomNumberGenerator.nextInt( 6);
            //This is a new integer for the player that is based on the randomNumberGenerator function, and is bound by only 1-6
            Dice1.setImageResource(diceArray[intNumberP1]);
            //This set the dice 1 as the picture contained within the array based on the number gained from the random number generation.

            int intNumberA1 = randomNumberGenerator.nextInt( 6);
            //This is the same as intNumberP1 but has a new number as .nextInt will ensure it is a different number it then sets the second dice image to the correlating array number
            Dice2.setImageResource(diceArray[intNumberA1]);

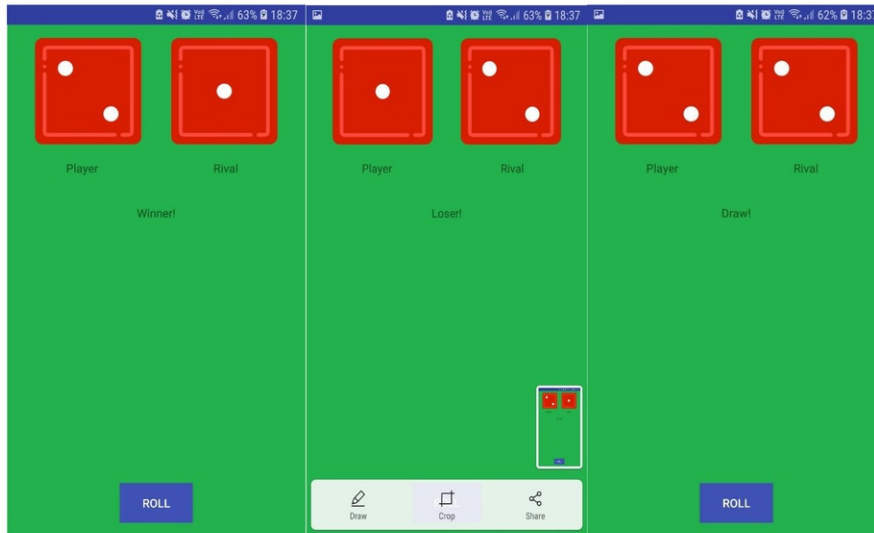
            if (intNumberP1 > intNumberA1) {
                Result.setText("Winner!");
            } //if the players number is Greater than the AI then the text will display winner
            if (intNumberA1 > intNumberP1){
                Result.setText("Loser!");
            } //if the AI number is greater than the player then it will display Loser
            if (intNumberA1 == intNumberP1){
                Result.setText("Draw!"); //if both values are equal then it will display Draw!
            }
        });
    }
}
```

```

    }
    intNumberP1 = randomNumberGenerator.nextInt( bound: 6); //This creates the new number
    intNumberA1 = randomNumberGenerator.nextInt( bound: 6); //after the code is run so the roll button will always use a new random number
  }
}

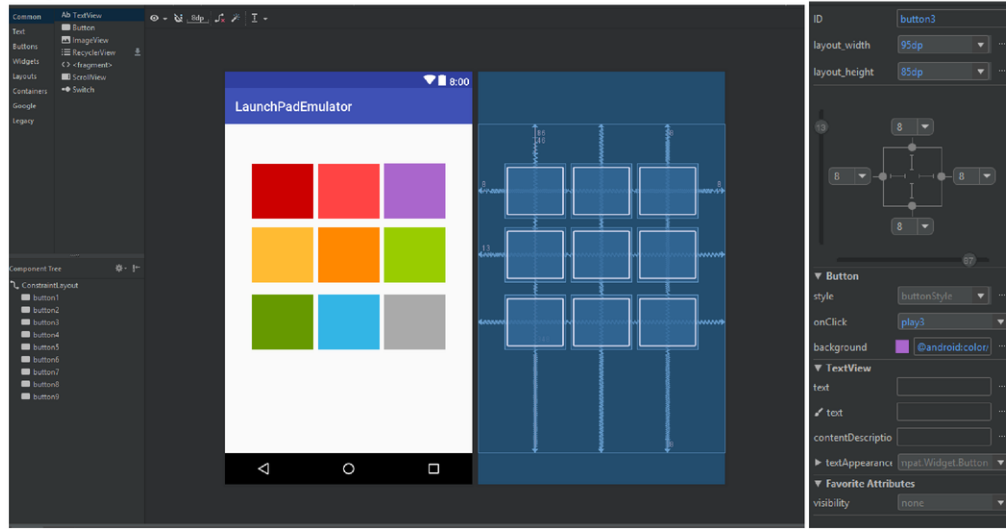
```

This game utilises an array in order to store the dice numbers and use them in if statements. The game then generates an on click listener event which will indicate when the button is pressed and will then create a new random number for both the player and the Rival and store them as integers. These values are then compared using three if statements and if the player has a higher number than the rival then you win, if the opposite is true you lose, if both numbers are the same then it is a draw. Here are three examples winning, losing and drawing in action.



Exercise 2 – Soundboard/Launch Pad

Design View



This is the overall design on the main_activity.xml. this application uses buttons and resources in the res folder in order to function. Each button is set with an on click listener event in order to play a specific sound, so in the example above the purple button is set to use play3 as its onclick event, so it should play music sample 3 when pressed. The music used is Yes-Roundabout (<https://www.youtube.com/watch?v=cPCLFtxpadE>) and whilst it includes none of the vocals of the song, I must give credit and state that it is not my own music I simply utilised it for my example.

Launchpad code

```

1 package com.example.d1797643.musictest;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 public class MainActivity extends AppCompatActivity {
6
7     //this creates a new soundpool that the application will draw from in order to function
8
9     private int intSoundID1;
10    private int intSoundID2;
11    private int intSoundID3;
12    private int intSoundID4;
13    private int intSoundID5;
14    private int intSoundID6;
15    private int intSoundID7;
16    private int intSoundID8;
17    private int intSoundID9;
18
19    //here we create 9 integers, that will all be designated as sounds and stored to be used later
20
21    private final int NR_OF_SIMULTANEOUS_SOUNDS = 8; //This sets the number of simultaneous sounds to 8
22    private final float LEFT_VOLUME = 1.0f; //sets the left volume to 1.0
23    private final float RIGHT_VOLUME = 1.0f; //sets the right volume to 1.0 in order to enhance the users ability to hear
24    private final int NO_LOOP = 0; //this sets the looping to 0 so a sound does not loop infinitely
25    private final int PRIORITY = 0; //this sets the priority of the button press to 0 so it will play when pressed
26    private final float NORMAL_PLAY_RATE = 1.0f; //this sets the speed of the playback so that it is not sped up or slowed down.
27
28
29
30
31
32
33
34
35
36    @Override
37    protected void onCreate(Bundle savedInstanceState) {
38        super.onCreate(savedInstanceState);
39        setContentView(R.layout.activity_main); //This loads the applications content
40
41        mySoundPool = new SoundPool(NR_OF_SIMULTANEOUS_SOUNDS, AudioManager.STREAM_MUSIC, 1);
42        //this creates a new instance of soundpool which will be used to set the sound to be played
43
44        intSoundID1 = mySoundPool.load(getApplicationContext(), R.raw.part1, 1);
45        intSoundID2 = mySoundPool.load(getApplicationContext(), R.raw.part2, 1);
46        intSoundID3 = mySoundPool.load(getApplicationContext(), R.raw.part3, 1);
47        intSoundID4 = mySoundPool.load(getApplicationContext(), R.raw.part4, 1);
48        intSoundID5 = mySoundPool.load(getApplicationContext(), R.raw.part5, 1);
49        intSoundID6 = mySoundPool.load(getApplicationContext(), R.raw.part6, 1);
50        intSoundID7 = mySoundPool.load(getApplicationContext(), R.raw.part7, 1);
51        intSoundID8 = mySoundPool.load(getApplicationContext(), R.raw.part8, 1);
52        intSoundID9 = mySoundPool.load(getApplicationContext(), R.raw.part9, 1);
53
54        //this is all the sounds that the application will use, whilst this could be done in an array for efficiency
55        //using just integers works just as well. It gets the sound to be played from the res folder
56
57    }
58 }

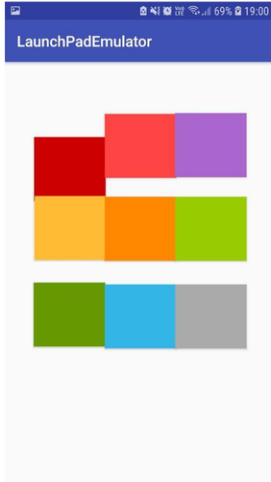
```

```

57     }
58     //
59     public void play1(View v)//this creates a new variable which will play the first sound
60     {
61         mySoundPool.play(intSoundID1, LEFT_VOLUME, RIGHT_VOLUME,PRIORITY,NO_LOOP,NORMAL_PLAY_RATE);//this gets all the settings we instantiated at the beginning such as volume/looping
62         //the sound is designated by the intSoundID so intSoundID1 is the first sound to be played.
63         //in order to play the sound with the correct settings
64         //on each button they have an on click event and that is set to the corresponding play number, so number 1 is assigned play1, button 2 is assigned play2 etc.
65         //This is used for each of the 9 buttons so there are a total of 9 play methods which will be utilised in order to play the sounds.
66     }
67
68     public void play2(View v)
69     {
70         mySoundPool.play(intSoundID2, LEFT_VOLUME, RIGHT_VOLUME,PRIORITY,NO_LOOP,NORMAL_PLAY_RATE);
71     }
72
73     public void play3(View v)
74     {
75         mySoundPool.play(intSoundID3, LEFT_VOLUME, RIGHT_VOLUME,PRIORITY,NO_LOOP,NORMAL_PLAY_RATE);
76     }
77
78     public void play4(View v)
79     {
80         mySoundPool.play(intSoundID4, LEFT_VOLUME, RIGHT_VOLUME,PRIORITY,NO_LOOP,NORMAL_PLAY_RATE);
81     }
82
83     public void play5(View v)
84     {
85         mySoundPool.play(intSoundID5, LEFT_VOLUME, RIGHT_VOLUME,PRIORITY,NO_LOOP,NORMAL_PLAY_RATE);
86     }
87     public void play6(View v)
88     {
89         mySoundPool.play(intSoundID6, LEFT_VOLUME, RIGHT_VOLUME,PRIORITY,NO_LOOP,NORMAL_PLAY_RATE);
90     }
91
92     public void play7(View v)
93     {
94         mySoundPool.play(intSoundID7, LEFT_VOLUME, RIGHT_VOLUME,PRIORITY,NO_LOOP,NORMAL_PLAY_RATE);
95     }
96
97     public void play8(View v)
98     {
99         mySoundPool.play(intSoundID8, LEFT_VOLUME, RIGHT_VOLUME,PRIORITY,NO_LOOP,NORMAL_PLAY_RATE);
100     }
101     public void play9(View v)
102     {
103         mySoundPool.play(intSoundID9, LEFT_VOLUME, RIGHT_VOLUME,PRIORITY,NO_LOOP,NORMAL_PLAY_RATE);
104     }
105
106 }
107
MainActivity

```

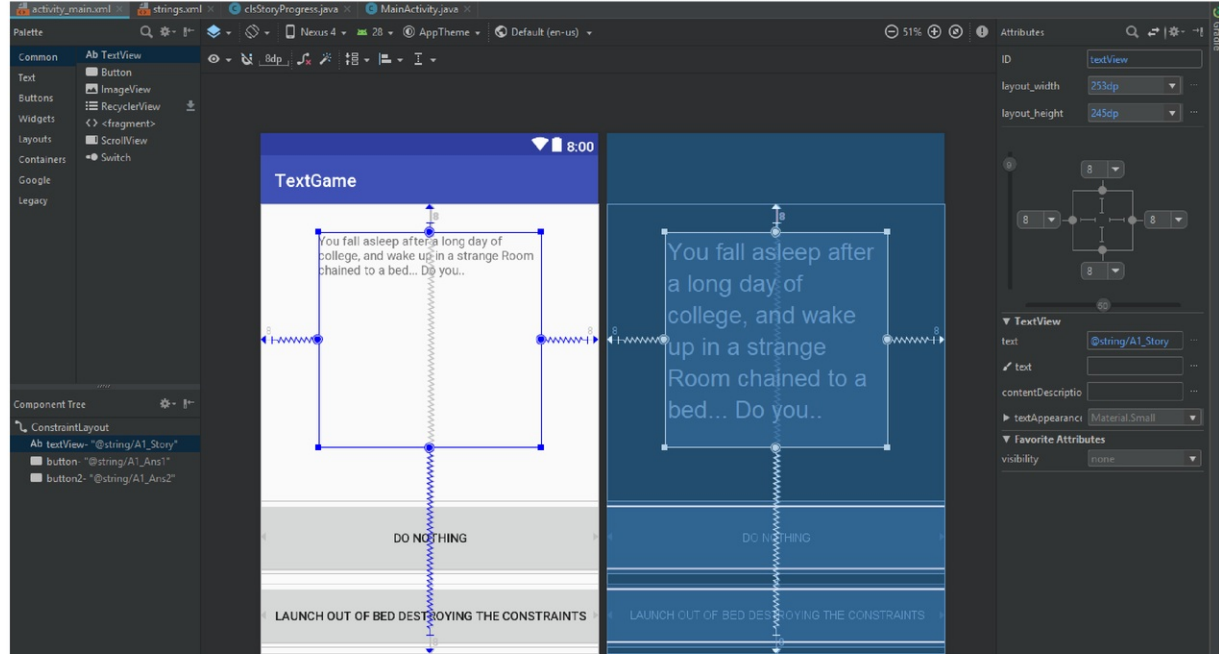
In order to complete the task given, a sound pool is created in order to store all the music files and prepare them for use within the application, this is then followed by 9 integers which will hold the part of the song that they are designated to play when pressed. This is then followed by several variables which store the properties of the sounds that will be played, such as the volume in the right and left ear, the pitch of the song, the length of the song etc. this is so the user does not have an infinite looping song that plays at a horrible volume and pitch. Then after initialising the applications layout, we create the ID for all the sounds to be played, from 1 to 9 and load them using R.raw.part1 so this will load part 1 into intSoundID1. Once this has been done the application creates the onclick events of play1 up to play9, which will play the designated sound when the onclick event on the buttons is triggered.



Unfortunately, despite the design layout looking much more streamlined when the application is run on a phone the buttons do not line up correctly, after spending some time trying to get the layout working, I'm uncertain as to why it is so skewed when I run the application although each button is functional and plays the sound correctly.

Exercise 3- Text Adventure

Design view



This application utilises a text view, which has been set to the string A1_story so that it loads the initial story of the application, the buttons have a similar text setting and have the string A1_ans1 for the top button and A1_Ans2 is used for the bottom button.

Strings.xml

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

  <string name="A1_Story">You fall asleep after a long day of college, and wake up in a strange Room chained to a bed... Do you..</string>
  <string name="A1_Ans1">Do nothing</string>
  <string name="A1_Ans2">Launch out of bed destroying the constraints</string>

  <string name="A2_Story">After launching out of bed you spiral out of control and land in some glass, you are bleeding.</string>
  <string name="A2_Ans1">You rip you clothes and attempt to bandage the wound</string>
  <string name="A2_Ans2">Do nothing again(why are you like this?)</string>

  <string name="A3_Story">You notice a stranger has entered the room via a door.</string>
  <string name="A3_Ans1">Ask if he knows what's going on here</string>
  <string name="A3_Ans2">Ignore the stranger</string>

  <string name="A4_Story">The stranger is very nonchalant and states that you were captured by a rogue serial killer</string>
  <string name="A4_Ans1">Given the situation you ask the stranger for help</string>
  <string name="A4_Ans2">You ignore the stranger</string>

  <string name="A5_End">You succumb to your wounds and pass out, your inaction caused your own demise Congratulations.</string>
  <string name="A6_End">The stranger is very displeased that you ignored him and shoots you.</string>
  <string name="A7_End">The stranger was actually a law enforcement officer and rescues you..</string>
</resources>
```

This is where all the strings for the story are kept, this is used to populate an array within the main activity.

clsStoryProgress

```
package com.example.danielD.textgame;

/**
 * Created by DanielD on 23/10/2018.
 */

public class clsStoryProgress { //this creates a new public class which will be called during the main activity in order to check the story progress.

    private int mStoryID; //Creating a new integer called mStoryID which will be used to store the story ID that the user is currently on.
    private int mAnswer; //Creates a new integer called mAnswer which will be used to record the users chosen answer during the story.

    public clsStoryProgress(int StoryResourceID, int intAnswer) {
        //here a new instance of the clsStoryProgress class is made using the story resource ID and intAnswer as the values it will be using
        mStoryID = StoryResourceID; //mStoryID becomes equal to the value contained within StoryResourceID
        mAnswer = intAnswer; //mAnswer becomes equal to the value contained within intAnswer
    }

    public int getStoryID() {return mStoryID;} //Here we create a new integer called getStoryID which will be used to get the current story ID

    public void setmStoryID(int StoryID) {mStoryID = StoryID;} //This will then set the gotten story ID and store it in setmStory

    public int isAnswer() {return mAnswer;} //this returns the mAnswer value.

    public void setmAnswer(int answer) {mAnswer = answer;} //setmAnswer then sets the answer to the users selection.

}
```

This is the class that is used to track the story progress by getting and setting the users choices and the current story step so that the program does not lose where the user is during the story.

Main_Activity

```
1 package com.example.danielid.textgame;
2
3 import ...
4
5
6
7
8
9
10
11 public class MainActivity extends AppCompatActivity { //here we create a new class called main activity
12
13     private Button mTopButton; //creates a new private instance of Button to be used in the application as the button the user presses for their choices
14     private Button mBottomButton; //This functions the same as the previous Button instance but this is the bottom button choice
15     private TextView mTextView; //here we create a new instance of the text view that the user will be using to read the story of the adventure as it progresses
16
17     private int mStoryIndex; //Here we create a new integer which will be used to determine which story step the user is currently on
18     private int mStory; //this will be used to indicate the text needed for the story
19     private int mTopButtonIndex; //this sets the index number for the button so the text in the button should display the new story step
20     private int mBottomButtonIndex; //this is exactly the same as the previous however this sets the bottom button
21
22     private clsStoryProgress[] mStoryBank = new clsStoryProgress[] { //creates a new array called story bank using clsStoryProgress as the value
23         new clsStoryProgress("You fall asleep after a long day of college, and wake u...", intAnswer 1),
24         new clsStoryProgress("Do nothing", intAnswer 2),
25         new clsStoryProgress("Launch out of bed destroying the constraints", intAnswer 3),
26         new clsStoryProgress("After launching out of bed you spiral out of control an...", intAnswer 4),
27         new clsStoryProgress("You rip you clothes and attempt to bandage the wound", intAnswer 5),
28         new clsStoryProgress("Do nothing again(why are you like this?)", intAnswer 6),
29         new clsStoryProgress("You notice a stranger has entered the room via a door.", intAnswer 7),
30         new clsStoryProgress("That is he knows what's going on here", intAnswer 8),
31         new clsStoryProgress("Ignore the stranger", intAnswer 9),
32         new clsStoryProgress("The stranger is very nonchalant and states that you wer...", intAnswer 10),
33         new clsStoryProgress("Given the situation you ask the stranger for help", intAnswer 11),
34         new clsStoryProgress("You ignore the stranger", intAnswer 12),
35         new clsStoryProgress("You succumb to your wounds and pass out, your inaction ...", intAnswer 13),
36         new clsStoryProgress("The stranger is very displeased that you ignored him an...", intAnswer 14),
37         new clsStoryProgress("The stranger was actually a law enforcement officer and...", intAnswer 15),
38     }; //each new clsStoryProgress is assigned a string to determine which text to display, the index position will be used to call the specific story instance later in the program
39
40     };
41
42     @Override
43     protected void onCreate(Bundle savedInstanceState) {
44         super.onCreate(savedInstanceState);
45         setContentView(R.layout.activity_main); //here we set the view and layout of the main activity
46
47         if (savedInstanceState != null) {
48             mStoryIndex = savedInstanceState.getInt( key: "StoryIndex");
49         } //this will save the state of the application when the user uses the back button or the home button
50         else //if this is not the case then the story is reset to it's default values
51         {
52             mStoryIndex = 0;
53             mTopButtonIndex = 1;
54             mBottomButtonIndex = 2;
55             //initialising each of the values to their default state
56         }
57     }
58 }
```

The application starts by initialising all the buttons and the text view to be used in the code so that the text of the story can be loaded into the buttons and the text view on the application.

Various story related variables are then created in order to set the index of the story, the buttons and to dictate which text is to be loaded. Then an array is populated in order to be used in the program, starting with intanswer 1 and ending on intanswer 15. This contains the string that the application will be using to load into the buttons and the text view.

There is then a new onCreate protected void made in order to save the users progress so that if the home button is pressed during the application it will not reset and will retain their progress, however if the instance state is currently 0 then it sets the story index to 0, and loads the defaults of the indexes for both the buttons.

```

66 mTopButton.setOnClickListener({view} - {
67     checkStoryPath( userSelection: 1); //if the button pressed is the top button it sets the user selection to 1 meaning the first button.
68     UpdateStory(); //This will then run the UpdateStory method
69 });
70
71
72
73
74 mBottomButton.setOnClickListener({view} - {
75     checkStoryPath( userSelection: 2);
76     UpdateStory(); //this functions the same as the top button, but instead of setting the user selection to 1, it sets it to 2
77 });
78
79
80
81
82
83 }
84
85 private void checkStoryPath(int userSelection) //creates a new method to check the story path, retaining the user selection
86 {
87     int ChosenAnswer = mStoryBank[mStoryIndex].isAnswer(); //this then sets the ChosenAnswer integer to the button to the answer picked,
88
89     if (userSelection == 1) //if the user answer is equal to 1 then this if statement chain will run
90     {
91         if (mStoryIndex == 0) //so if the story index is equal to 0 (default)
92         {
93             mStoryIndex = 5; //then the Story index is set to 5 to progress the story
94             mTopButtonIndex = 7; //this sets the top button index to 7
95             mBottomButtonIndex = 8; //this sets the bottom index to 8
96         }
97         if (mStoryIndex == 3) //if the story index is equal to 3 then this if statement will run
98         {
99             mStoryIndex = 5; //this sets the Story index to 5
100             mTopButtonIndex = 7; //this sets the top index is 7
101             mBottomButtonIndex = 8; //this sets the bottom index to 8
102         }
103         if (mStoryIndex == 6) //if the story index is equal to 6 it will run this statement
104         {
105
106             mStoryIndex = 8; //sets the story index to 8
107             mTopButtonIndex = 10; //sets the Top button index to 10
108             mBottomButtonIndex = 11; //sets the bottom button index to 11
109
110         }
111         else if (mStoryIndex == 9) //this will cover if the story index is equal to 9
112         {
113             mStoryIndex = 13; //this then sets the story text to be that of index position 13
114             End_Of_story(); //this then runs the end_of_story() method
115         }
116     }
117     if (userSelection == 2) //if the user presses option two however these if statements will run
118     {
119         if (mStoryIndex == 0) //default story instance value
120         {
121             mStoryIndex = 2; //sets the story text to that of index 2
122             mTopButtonIndex = 4; //sets the top button text to that of index 4

```

MainActivity

This part of the code begins with creating the onclicklisteners which will indicate if the top or bottom buttons are pressed by the user, indicating that the top button is user selection 1 and that the bottom button is user selection 2, after determining the button input the story path is then checked, and then updated to the values determined by the users button choice.

Check story path is a private void that utilises the user selection to perform an if statement which will check the status of the user's current story. So, if the user presses the top button, the if statement will check if they are currently on story indexes 0, 3, 6 and 9. If any of those requirements are fulfilled (which there should always be 1 of the statements that are currently fulfilled) it will change the story and button indexes to that of the next story step. This is then repeated for the bottom button.

```

121         mStoryIndex = 2;//sets the story text to that of index 2
122         mTopButtonIndex = 4;//sets the button text to that of index 4
123         mBottomButtonIndex = 5;//sets the button text to that of index 5
124     }
125     if (mStoryIndex == 3)//if the story index is equal to 3 then this will run
126     {
127         mStoryIndex = 11;//this sets the story text to that of index 11
128         End_of_story();//runs the end of story method
129     } if (mStoryIndex == 6)//if the story index is equal to 6
130     {
131         mStoryIndex = 12;//index 12's text is displayed
132         End_of_story();//the end story method runs
133     }
134     else if (mStoryIndex == 9)//if the index is equal to 9
135     {
136         mStoryIndex = 12;//the index 12 text is loaded
137         End_of_story();//resulting in the end of story method running
138     }
139 }
140
141
142
143 @Override
144 public void onSaveInstanceState(Bundle outState){//this saves the instance of the application
145     outState.putInt("Story Index", mStoryIndex);
146     super.onSaveInstanceState(outState);
147 }
148
149 private void End_of_story(){//this method will end the application when it is called
150     mTopButton.setVisibility(View.GONE);//this will hide the top button
151     mBottomButton.setVisibility(View.GONE);//this will hide the bottom button
152
153     AlertDialog.Builder alert = new AlertDialog.Builder( context, this);//this creates a new alert dialogue box to be displayed
154     alert.setTitle("End of Story");//this sets the title to End of Story
155     alert.setCancelable(false);//this makes it so the user cannot cancel the alert or dismiss it.
156     alert.setPositiveButton( new DialogInterface.OnClickListener() { //this then sets the positive button to display close application
157         finish();
158     });
159     alert.show();//this will then show the created alert.
160
161 }

```

The key difference with the bottom button is how there are more endings located on the bottom route, so if the user presses the bottom button when they are at story index 3, it will load the index 11 text and will then run the End_of_story method.

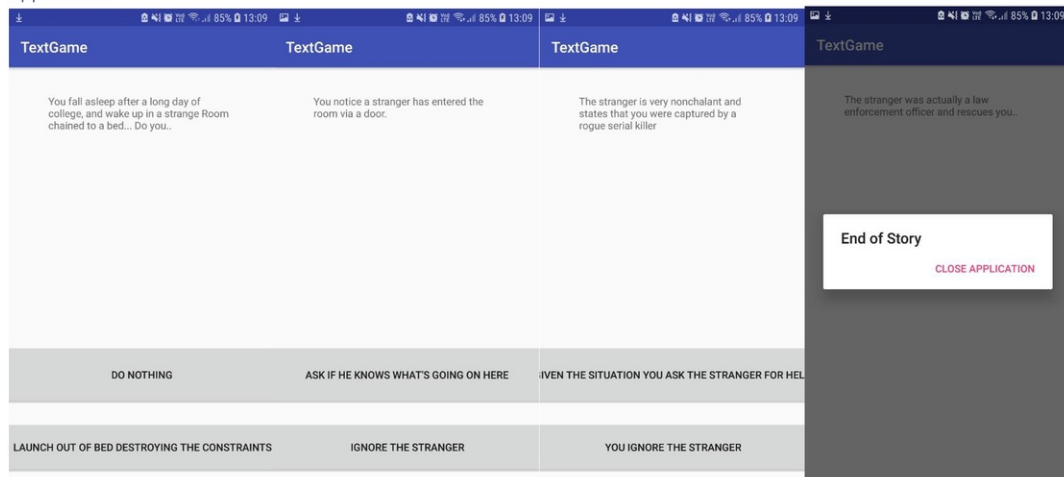
The onSaveInstanceState will save the users current instance of the application so the buttons and text view will display the correct story when the application is closed and re-opened by pressing the home button.

End_of_Story is the method used in order to close the application when the story reaches one of the many potential climaxes the user can trigger, when it is called it will hide both buttons using setVisibility and making the view obscured so the buttons do not show up for the user during an ending. It also displays a dialogue notification which only has one button which is to close the application.

Then finally the update story method is created in order to update the current story to the newly set values. So first is will check if the story index is currently a valid number (as it cannot be 0 based on the check story function) as the program will start on 0 and then will be changed to another number depending on the user's input, so 0 cannot be the result of the calculation (mStoryIndex+1) % m_storybank.length, but if it were to be true then the application would end as the end_of_story method would be called to lose the application if the result of the calculation is 0. Otherwise the mStory is set to the new storyID, the textView has new text assigned to it based on the mStory and the StoryID. This is repeated for both buttons.

```
166     private void UpdateStory()//this creates a new method called Update Story
167     {
168         mStoryIndex = (mStoryIndex + 1) % mStoryBank.length;//this makes it so the story index is equal to the story index +1 uses a remainder function along with the length of story
169         //calculate if the story index is equal to 0.
170
171         if (mStoryIndex == 0){//if the story index is equal to 0 then it will end the story
172             End_of_story();//this runs the end of story method
173         }
174
175         mStory = mStoryBank[mStoryIndex].getStoryID();//this gets the current story ID
176         mTextView.setText(mStory);//this then displays the text in the text view by setting the value contained
177         mStory = mStoryBank[mTopButtonIndex].getStoryID();//this then gets the ID of the top button
178         mTopButton.setText(mStory);//and sets the text of the button to that which was gotten
179         mStory = mStoryBank[mBottomButtonIndex].getStoryID();//this gets the ID of the bottom button
180         mBottomButton.setText(mStory);//This then sets the text of the bottom button.
181     }
182 }
183
184 }
185
```

Application



Here is a run-through of the application by clicking only the top button, it indicates the correct options on each of the buttons and displays the correct text when used.

References

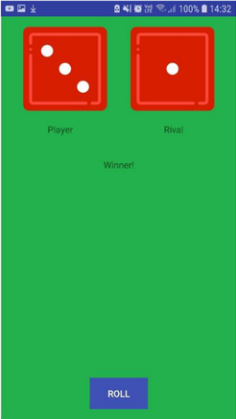
- [1] C. G. Lasater, Design Patterns, Wordware Publishing, Inc, 2010.
- [2] K. Mew, Android Design Patterns and Best Practice, Packt Publishing Ltd, 2016.
- [3] D. Rajput, Spring 5 Design Patterns: Master efficient application development with patterns such as proxy, singleton, the template method, and more, Packt Publishing Ltd,, 2017.
- [4] C. Giridhar, Learning Python Design Patterns, Packt Publishing, 2016.
- [5] J. Bishop, C# 3.0 Design Patterns: Use the Power of C# 3.0 to Solve Real-World Problems, O'Reilly Media, Inc, 2007.
- [6] E. F. E. R. K. S. B. B. Eric Freeman, Head First Design Patterns, O'Reilly Media, Inc, 2004.
- [7] C. F. Shahbudin FE, "Design Patterns for Developing High Efficiency Mobile Application.," *Journal of Information Technology & Software Engineering*, vol. 3, no. 3, pp. 1-9, 2013.
- [8] R. H. R. J. J. V. Erich Gamma, Design Patterns: Elements of Reusable Object-Oriented Software (Adobe Reader), Pearson Education, 1994.
- [9] J. R. G. Paz, Beginning ASP.NET MVC 4, Apress, 2013.
- [10] D. T. R. S. O. C. Godfrey Nolan, Android Best Practices, Apress, 2014.
- [11] M. S. S. Ryan Vice, MVVM Survival Guide for Enterprise Architectures in Silverlight and WPF, Packt Publishing, 2012.
- [12] G. Hall, Pro WPF and Silverlight MVVM: Effective Application Development with Model-View-ViewModel, Apress, 2011.
- [13] A. Osmari, "Understanding MVVM - A Guide For JavaScript Developers," 10 April 2012. [Online]. Available: <https://addyosmani.com/blog/understanding-mvvm-a-guide-for-javascript-developers/>. [Accessed 15 December 2018].
- [14] J. Gossman, "Advantages and disadvantages of M-V-VM," Microsoft, 4 March 2006. [Online]. Available: <https://blogs.msdn.microsoft.com/johngossman/2006/03/04/advantages-and-disadvantages-of-m-v-vm/>. [Accessed 15 December 2018].
- [15] C. Biligin, Mastering Cross-Platform Development with Xamarin, Packt Publishing Ltd, 2016.

Appendix

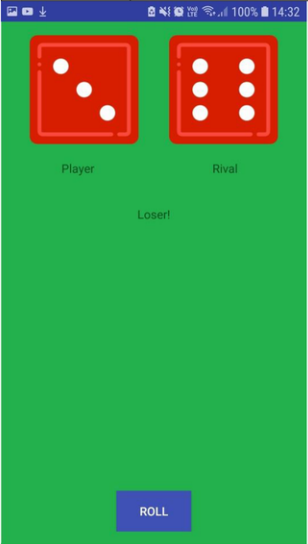
Testing

Dice Application

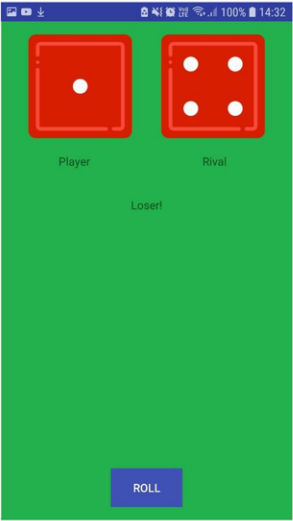
Test Number	Test Purpose	Test Data	Expected Result	Actual Result
01	To test if the random numbers generated work and will determine who the winner is.	Pressing the roll button	If the player number is larger than the Rival then the player wins, if the Player has a lower number then they lose, if both numbers are the same then it is a draw.	The numbers 3 and 1 were displayed for the player and rival, resulting in the player being the winner.



Test Number	Test Purpose	Test Data	Expected Result	Actual Result
02	To test if the random numbers generated work and will determine who the winner is.	Pressing the roll button	If the player number is larger than the Rival then the player wins, if the Player has a lower number then they lose, if both numbers are the same then it is a draw.	The numbers 3 and 6 are displayed as die, the player is considered the loser.



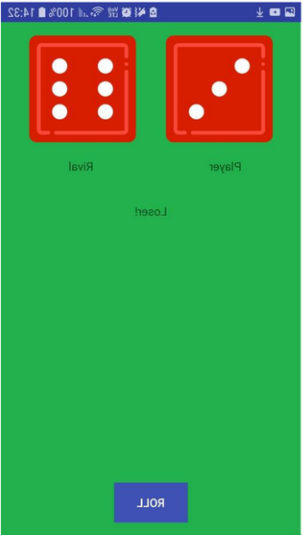
Test Number	Test Purpose	Test Data	Expected Result	Actual Result
03	To test if the random numbers generated work and will determine who the winner is.	Pressing the roll button	If the player number is larger than the Rival then the player wins, if the Player has a lower number then they lose, if both numbers are the same then it is a draw.	The numbers 1 and 4 were displayed for the player and rival, resulting in the player being the loser.



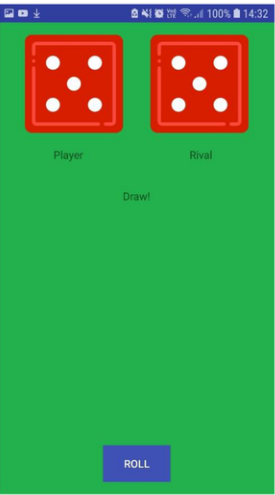
Test Number	Test Purpose	Test Data	Expected Result	Actual Result
04	To test if the random numbers generated work and will determine who the winner is.	Pressing the roll button	If the player number is larger than the Rival then the player wins, if the Player has a lower number then they lose, if both numbers are the same then it is a draw.	The numbers 6 and 5 were displayed for the player and rival, resulting in the player being the winner.



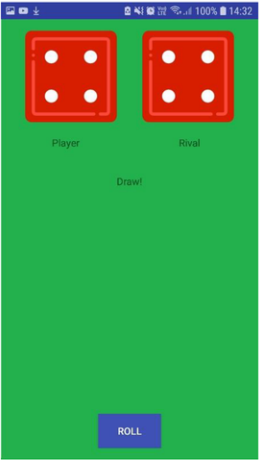
Test Number	Test Purpose	Test Data	Expected Result	Actual Result
05	To test if the random numbers generated work and will determine who the winner is.	Pressing the roll button	If the player number is larger than the Rival then the player wins, if the Player has a lower number then they lose, if both numbers are the same then it is a draw.	The numbers 6 and 3 were displayed for the player and rival, resulting in the player being the winner.



Test Number	Test Purpose	Test Data	Expected Result	Actual Result
06	To test if the random numbers generated work and will determine who the winner is.	Pressing the roll button	If the player number is larger than the Rival then the player wins, if the Player has a lower number then they lose, if both numbers are the same then it is a draw.	The number 5 was displayed for the player and rival, resulting in the round being a draw



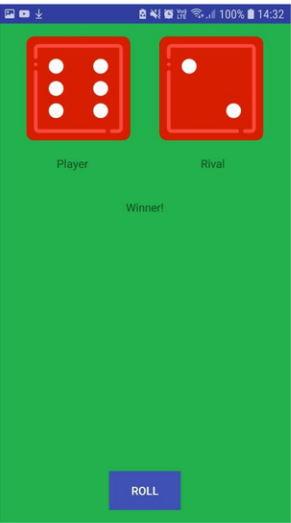
Test Number	Test Purpose	Test Data	Expected Result	Actual Result
07	To test if the random numbers generated work and will determine who the winner is.	Pressing the roll button	If the player number is larger than the Rival then the player wins, if the Player has a lower number then they lose, if both numbers are the same then it is a draw.	The number 4 was displayed for the player and rival, resulting in the round being a draw



Test Number	Test Purpose	Test Data	Expected Result	Actual Result
08	To test if the random numbers generated work and will determine who the winner is.	Pressing the roll button	If the player number is larger than the Rival then the player wins, if the Player has a lower number then they lose, if both numbers are the same then it is a draw.	The number 6 was displayed for the player and rival, resulting in the round being a draw



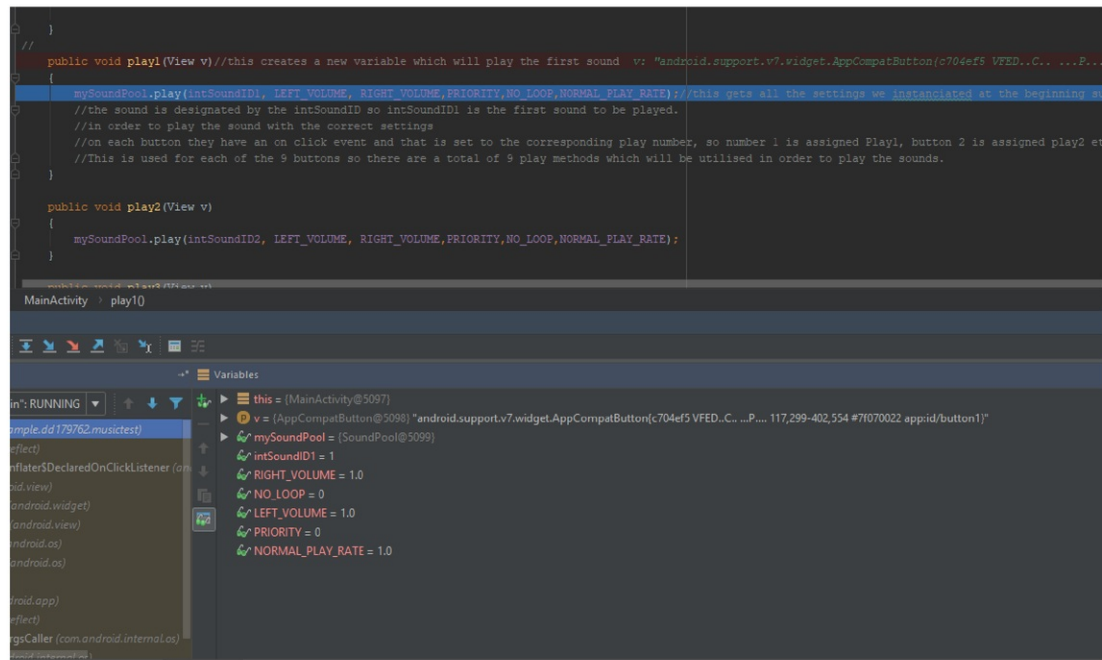
Test Number	Test Purpose	Test Data	Expected Result	Actual Result
09	To test if the random numbers generated work and will determine who the winner is.	Pressing the roll button	If the player number is larger than the Rival then the player wins, if the Player has a lower number then they lose, if both numbers are the same then it is a draw.	The numbers 6 and 2 were displayed for the player and rival, resulting in the round being considered the players victory



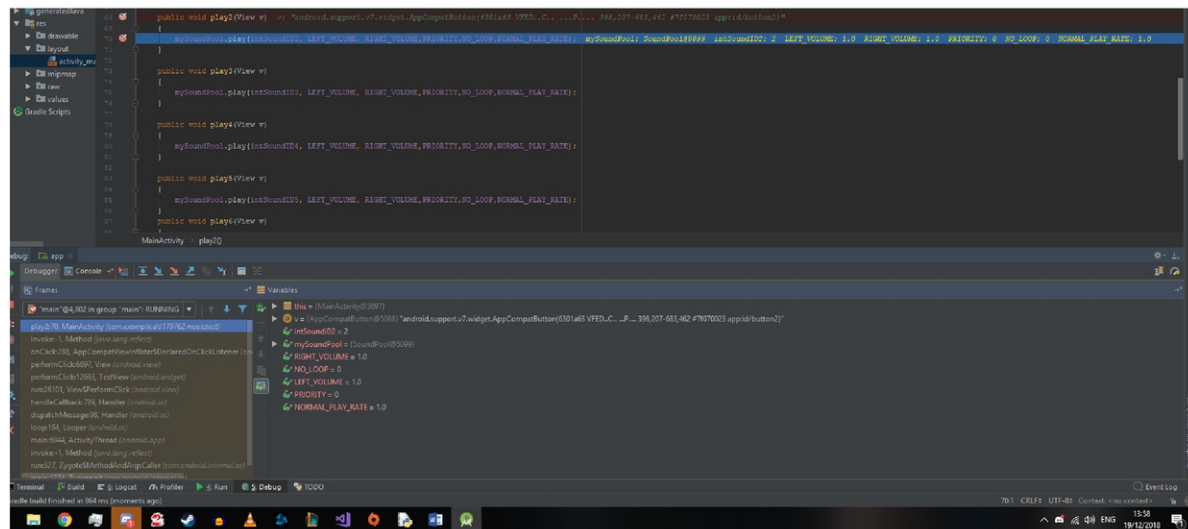
Launchpad/Soundboard

Test Number	Test Purpose	Test Data	Expected Result	Actual Result
01-09	To test if each button plays the correct sound when pressed	Pressing each button to determine if they are functioning correctly	Each press makes the correct part of Yes Roundabout play	The correct sound did play on each button press

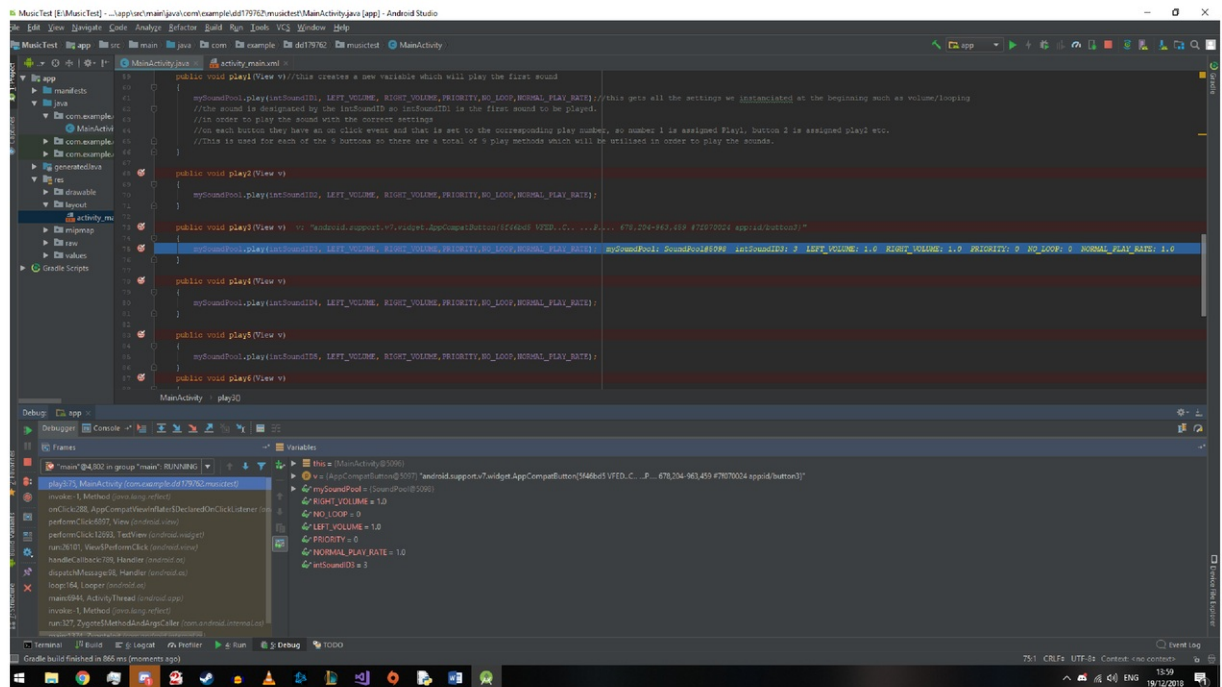
Red button



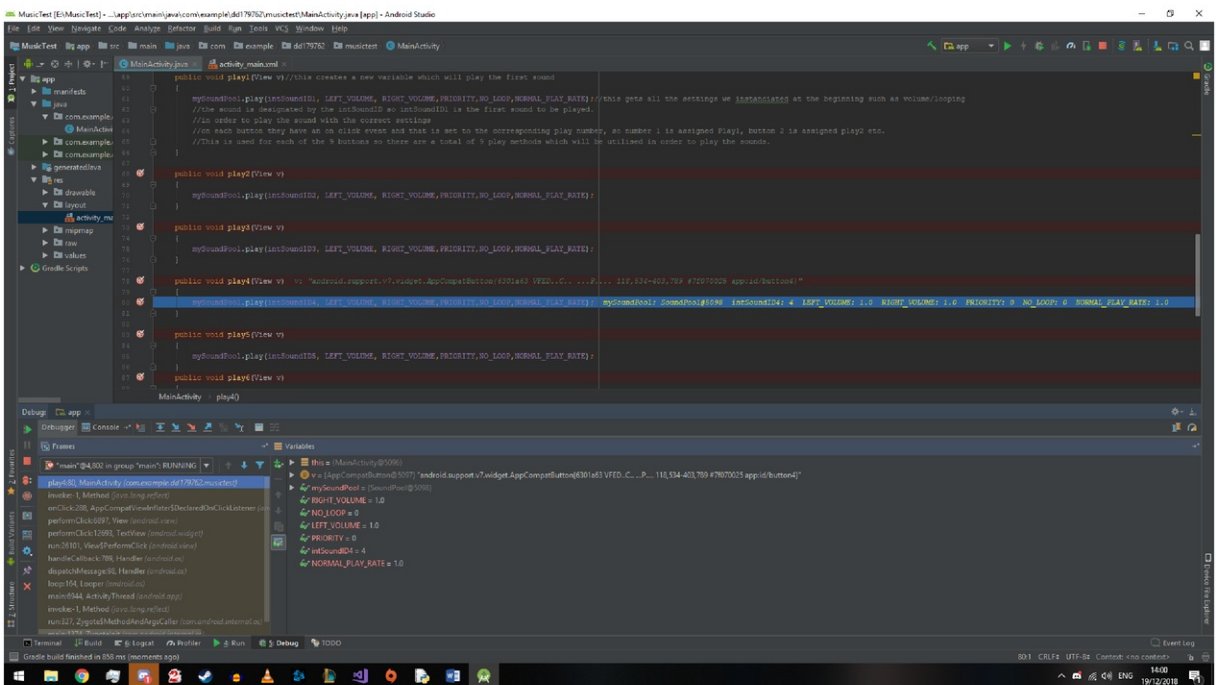
Orange button



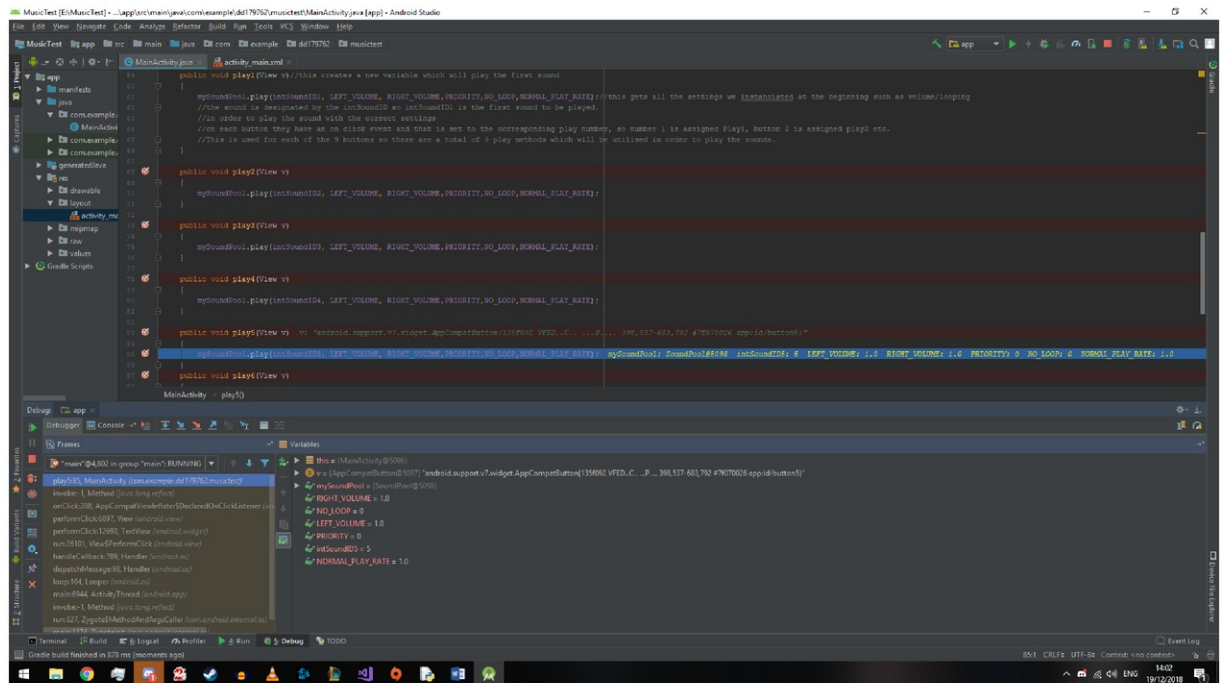
Purple button



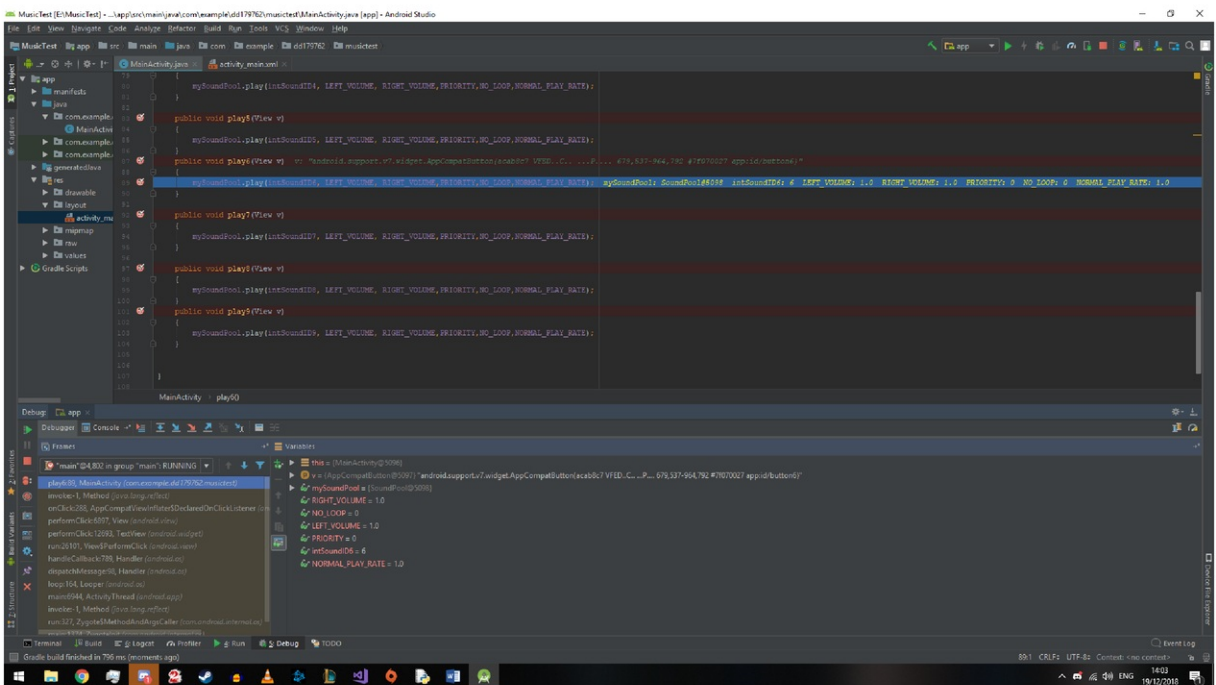
Yellow button



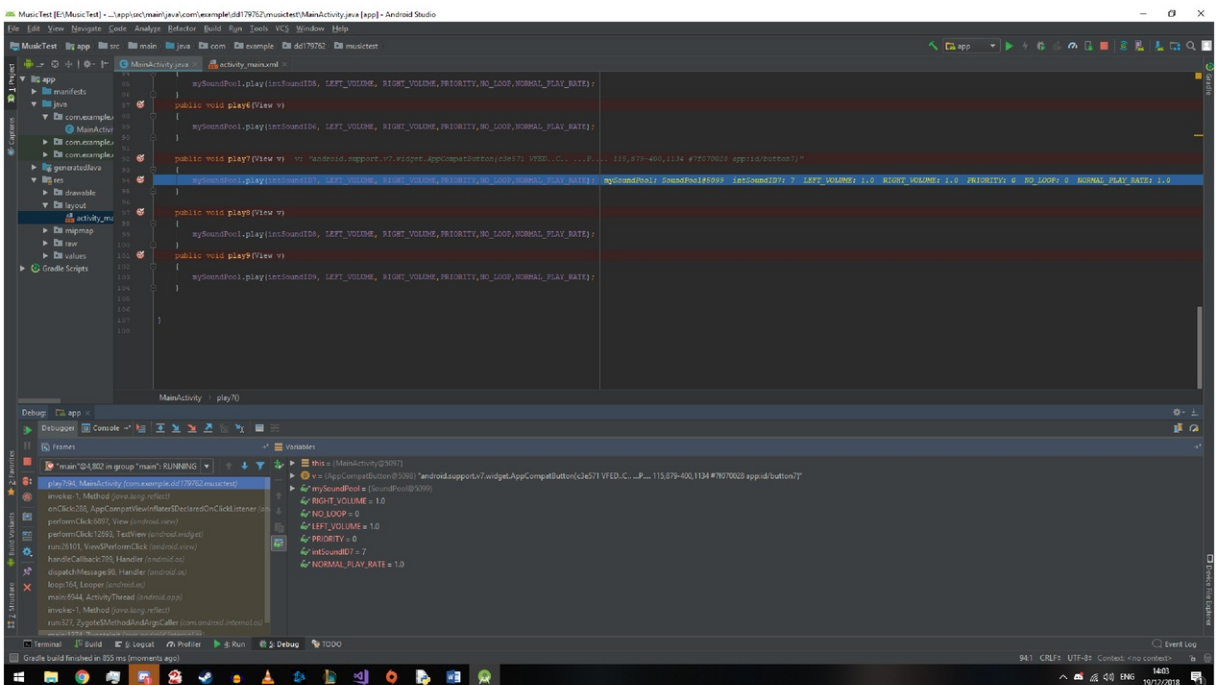
Brighter Orange button



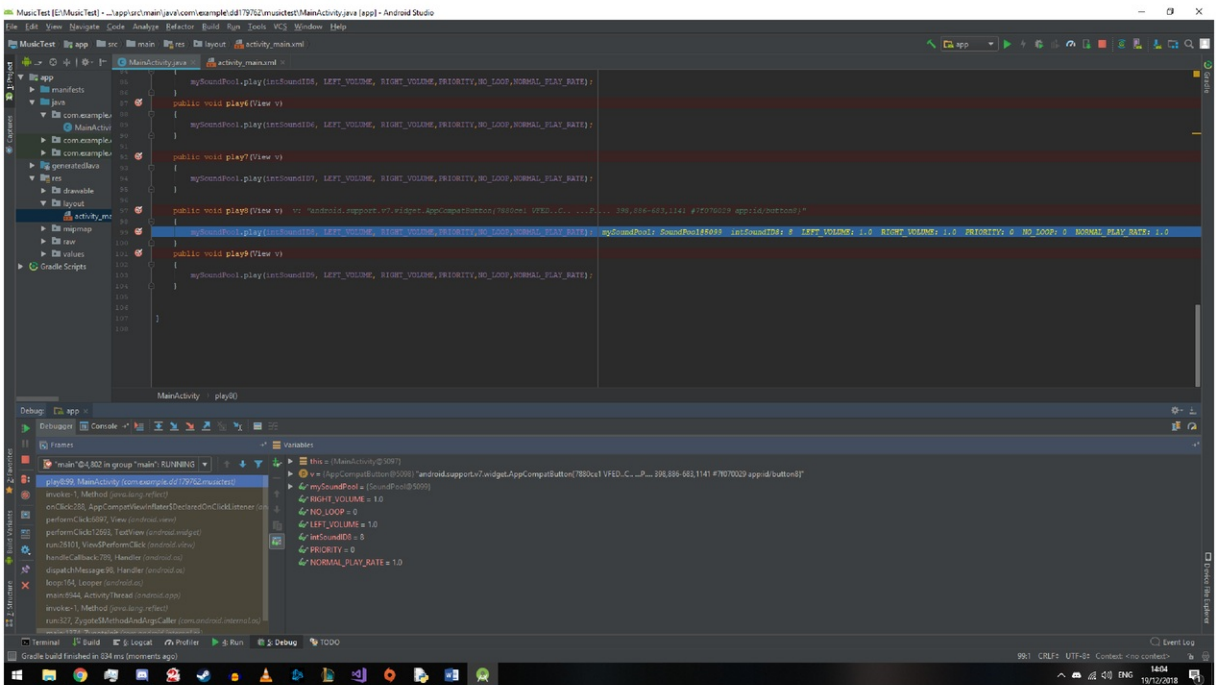
Green Button



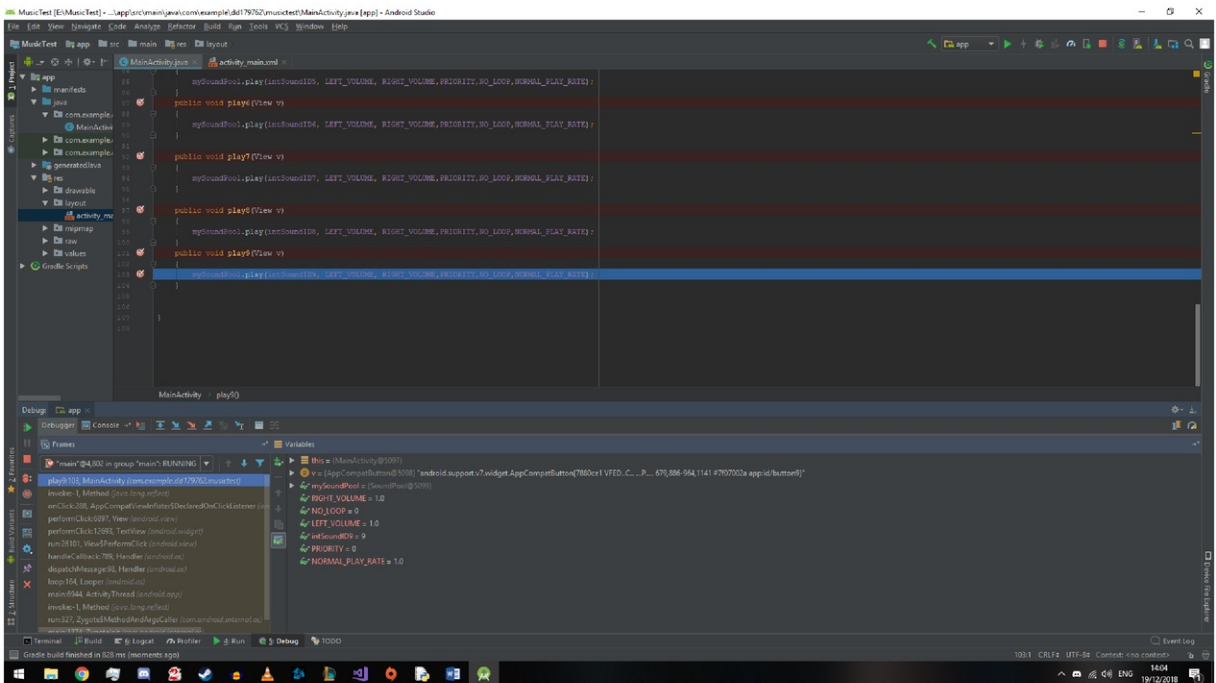
Dark Green button



Blue Button

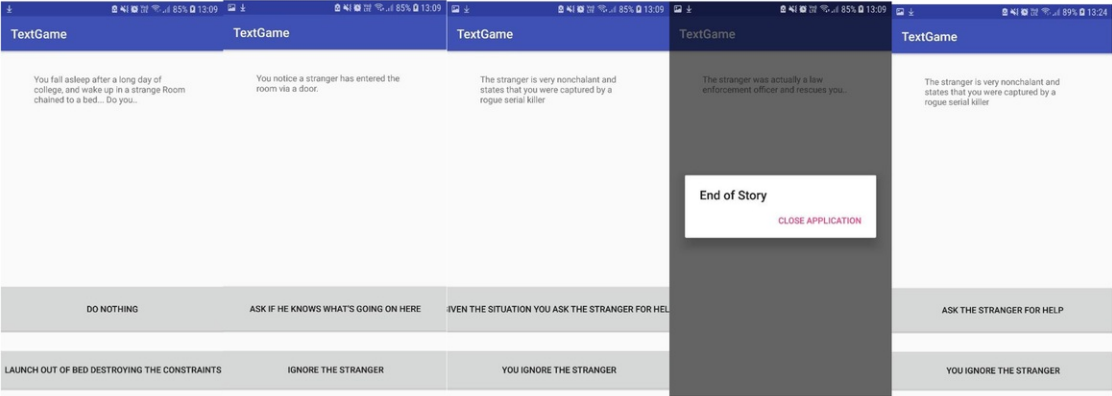


Grey button

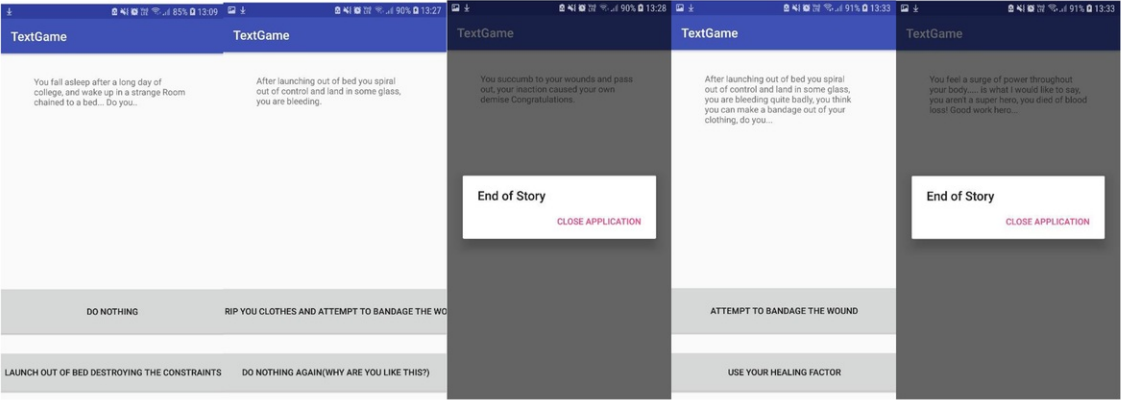


Adventure Game

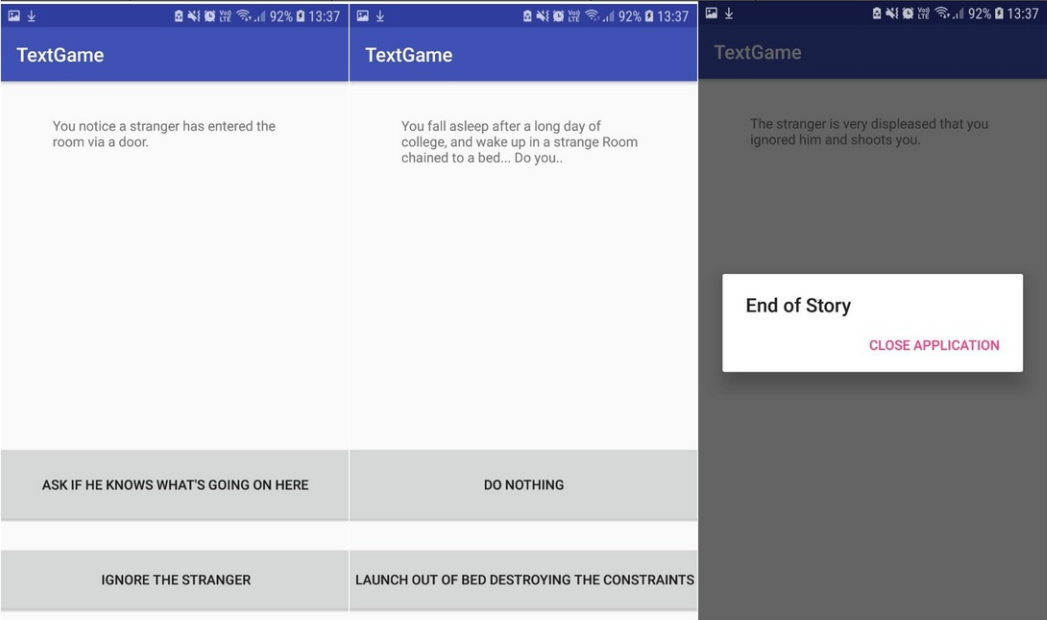
Test Number	Test Purpose	Test Data	Expected Result	Actual Result
01	To Complete a run-through of the application only using the top button	Pressing the top button at each dialogue option	The application should run through properly and end after the last choice	Whilst the run through was successful one of the button options was too long, resulting in the true option being hidden as the full line was obscured. (this has now been remedied by changing the string)



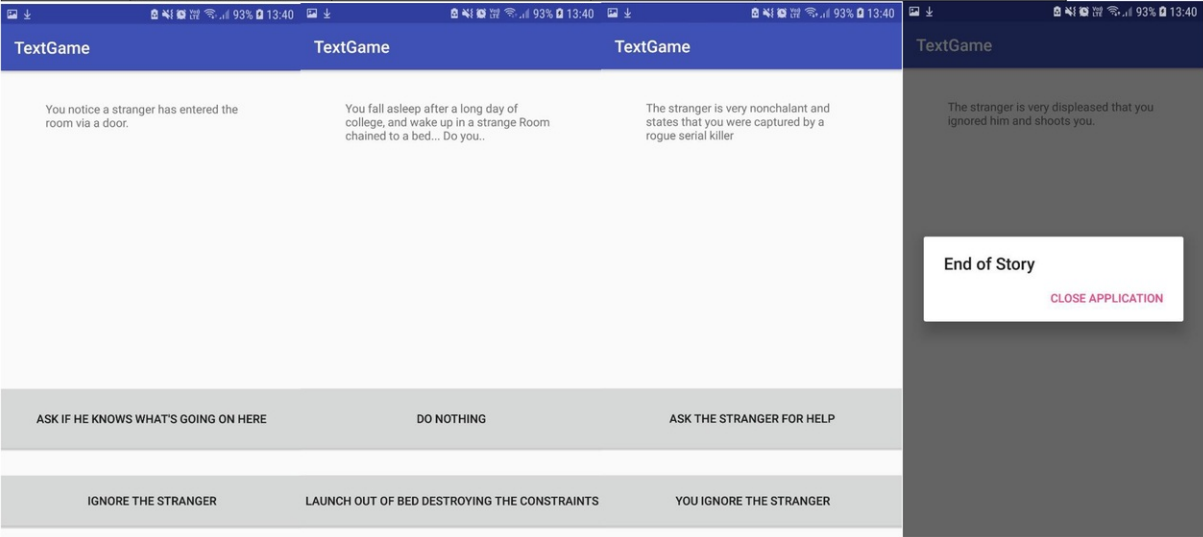
Test Number	Test Purpose	Test Data	Expected Result	Actual Result
02	To Complete a run-through of the application only using the bottom button	Pressing the bottom button at each dialogue option	The application should run through properly and end after the second	Whilst the code ran through properly, some of the dialogue is either cut off or makes no sense so this has now been rectified. (images on the left indicate the initial run through, images on the right indicate the revisions)



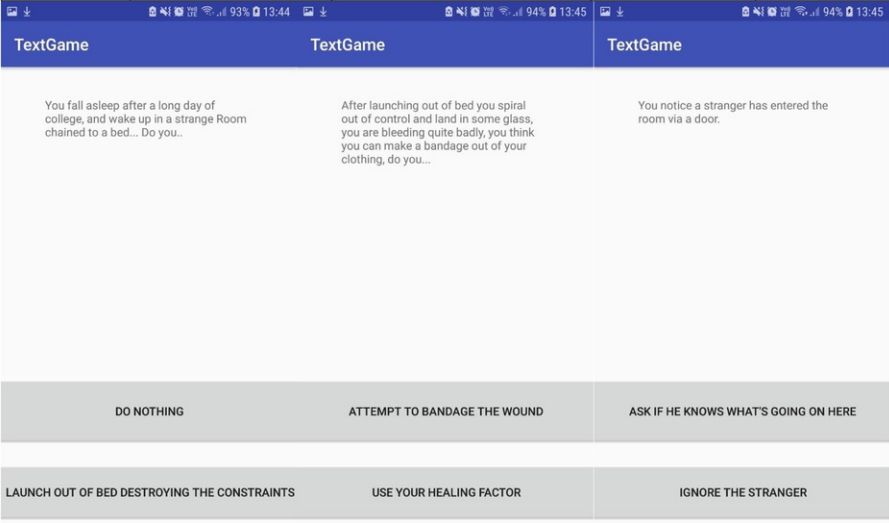
Test Number	Test Purpose	Test Data	Expected Result	Actual Result
03	Choosing to ignore the stranger	Pressing the bottom button at the second story step after pressing the top button.	The application should end once the option is selected.	The run through is successful and the story ends correctly, displaying the correct end text.



Test Number	Test Purpose	Test Data	Expected Result	Actual Result
04	To test if the story flows properly if I press the top button for all but the last option.	Pressing the top button for the first 3 options then pressing the bottom option.	The ending where the player is shot should trigger	The story progresses as intended, ending with the correct ending once the ignore option is selected.



Test Number	Test Purpose	Test Data	Expected Result	Actual Result
05	To test if the bottom option loops back into the correct story sequence when you press the bottom option then the top option.	Beginning the story with option 2 and then pressing option 1 after	The story should loop into the story entering the room path.	The program functioned correctly considering the path has already been tested it works all the way to the ending.



Test Number	Test Purpose	Test Data	Expected Result	Actual Result
06	To test if the application saves the story steps taken when closed and re-opened when pressing the home button	Beginning the story with option 2 and then pressing option 1 after	The story should open back into the story step the user was in.	The program functioned correctly and re-opened on the correct story instance.

