

Assignment 1

by Adam Howells

FILE	11672_ADAM_HOWELLS_ASSIGNMENT_1_164031_389414559.DOCX		
TIME SUBMITTED	21-DEC-2018 04:04 PM (UTC+0000)	WORD COUNT	3442
SUBMISSION ID	97911874	CHARACTER COUNT	22550

Adam Howells (AH180955)
Lecturer: Daniel Davies

Design Patterns

Assignment 1 – (DUM5002) Application Development



Table of Contents

1	Introduction	3
2	Procedure.....	3
3	Main	4
3.1	Part A: Understanding	4
3.1.1	Investigation	4
3.1.2	Model-View-ViewModel (MVVM) Pattern	15
3.2	Part B: Implementation	17
3.2.1	Exercise 1	17
3.2.2	Exercise 2	23
3.2.3	Exercise 3	26
4	Conclusion	33
1.1	Investigation.....	Error! Bookmark not defined.
	Bibliography	34

Adam Howells



1 Introduction

The understanding section will cover software patterns in mobile application development, as well as a look into a specific architectural and design pattern. Model-View-ViewModel (MVVM) will also be assessed.

The final implementation section will show the results from the various exercises, along with testing and documentation.

2 Procedure

To tackle this assignment, I will be using my Android Development skills I've learned in lecture alongside the official Android Studio documentation online to aid in programming android applications. My research for the investigation will largely consist of online sources; from journals/articles to blog posts and e-books in the form of PDFs.

25th November 2018



3 Main

3.1 Part A: Understanding

3.1.1 Investigation

According to Mark Richards [1], architectural patterns “help define the basic characteristics and behaviour of an application”. There are no ‘perfect’ patterns, but rather patterns with their own advantages and disadvantages that are used for various uses. Design Patterns are like architectural patterns, but instead of defining the application’s template on a whole, they serve to address specific recurring software tasks.

Model View Controller (MVC) Architectural Pattern

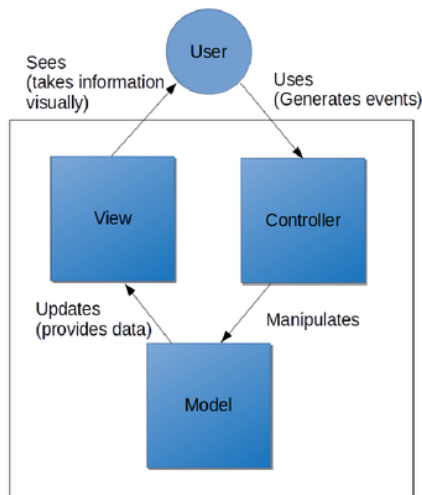


Figure 1: Illustration of MVC pattern

MVC is an architectural pattern that breaks an application’s code down into three components: functionality and data (model), rendering interface (view), and handling user input (controller). Figure 1 illustrates the components and their relationships, including the user.

According to an article written by Prathamesh Churi [2], it describes the basic functions of each component.

The Model encapsulates and provides core data and application functionality and does not rely on any input/output system. It also informs other components when any data is modified.

The View retrieves data from the Model and displays the information to the user and can be dynamic (a view is able to change in real time). A view in an MVC application is always associated with a controller component. In

Xamarin, a View is typically a *page*.



The Controller handles user inputs and can translate events, such as a button press, into requests for the Model component. A controller component is associated with one or more view components. For dynamic behaviour, it's the best practice that if a user makes changes to a controller on one specific view, all views associated with that controller should also be updated.

Simić D and Plakalović D [3] mention a concept called "change propagation mechanism". This means that when a change is made, it is distributed to all views. An observable object is used to notify the changes to all observers, which will be all registered views. According to Source Making [4], an educational website on software design patterns, the observable "defines a one-to-many relationship so that when one object changes state, the [observers] are notified and updated". This allows MVC to synchronize all views and controllers.

Example of MVC in Android Studio

In Java within Android Studio, MVC can be quite tricky to implement due to the main activity class being both a view and controller. The Activity which includes view and controller will act as the Observer, and the Model will behave as an Observer. This means that whenever a value is updated, the Model will notify the observer.

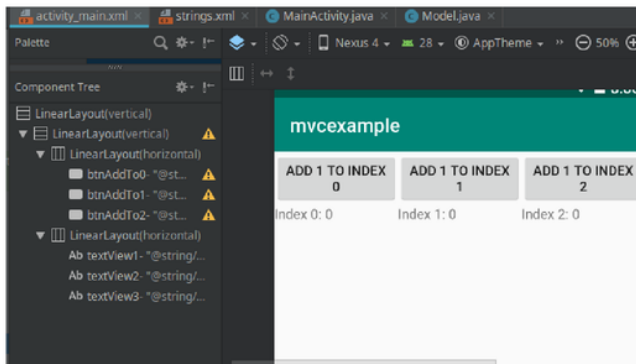


Image 1: Screenshot of activity_main.xml in design view

The model class will involve our observable object. As seen in Image 3, we extend our class to Observable. We then create a method to return value from a list, as well as setting the value to the list. The last method is important because we set the



changes and notify our observer (main activity class).

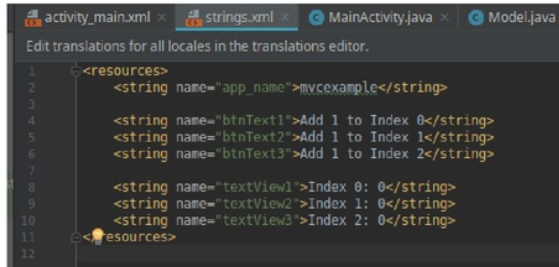


Image 2: Screenshot of strings.xml file

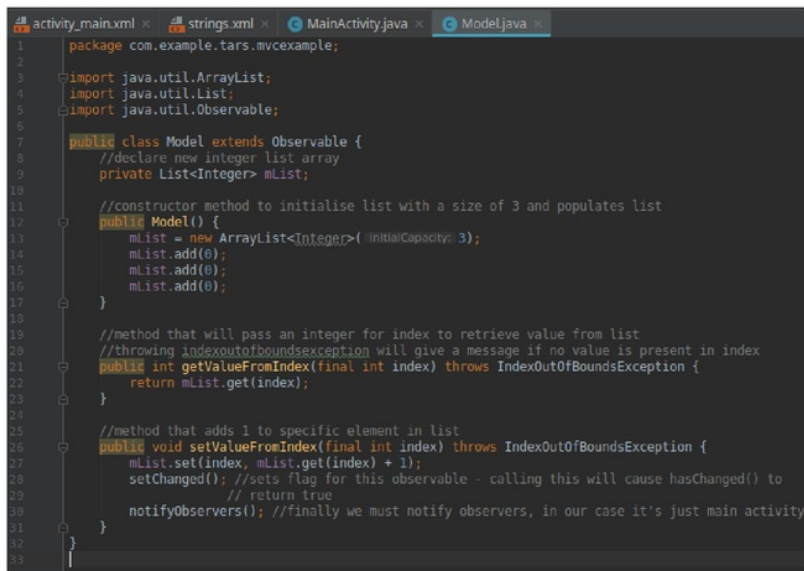


Image 3: Screenshot of Model.java class (Observable)

Our Main Activity class is where the controller and view functions of our application will reside, as seen in Image 4 and 5. An Observer object and onClickListener is implemented in the class. The update method within this class will be called when the Model makes a change and notifies our Observer.



```
activity_main.xml x strings.xml x MainActivity.java x Model.java x
12 //class implements Observer object as well as view onClickListener
13 public class MainActivity extends AppCompatActivity implements Observer, View.OnClickListener {
14     //declaring the model
15     private Model mModel;
16     //declaring the buttons
17     private Button mButton1;
18     private Button mButton2;
19     private Button mButton3;
20     //declaring the text views
21     private TextView mTextView1;
22     private TextView mTextView2;
23     private TextView mTextView3;
24
25
26
27 @Override
28 protected void onCreate(Bundle savedInstanceState) {
29     super.onCreate(savedInstanceState);
30     setContentView(R.layout.activity_main);
31
32     mModel = new Model(); //initialises constructor method from model
33     mModel.addObserver( this ); //adds this class as observer for observable in Model
34
35     //initialising text views
36     mTextView1 = (TextView) findViewById(R.id.textView1);
37     mTextView2 = (TextView) findViewById(R.id.textView2);
38     mTextView3 = (TextView) findViewById(R.id.textView3);
39
40     //initialising buttons on create
41     mButton1 = (Button) findViewById(R.id.btnAddTo0);
42     mButton2 = (Button) findViewById(R.id.btnAddTo1);
43     mButton3 = (Button) findViewById(R.id.btnAddTo2);
44     //set on click listeners for buttons - 'this' refers to our onClick method
45     mButton1.setOnClickListener(this);
46     mButton2.setOnClickListener(this);
47     mButton3.setOnClickListener(this);
48 }
```

Image 4: Screenshot of first half of MainActivity.java class



```
50 //used for view onClickListener - all buttons will use this as event handler
51 @Override
52 public void onClick(View v) {
53     //switch statement that retrieves ID of button and makes a case if it's each button
54     switch (v.getId()) {
55         case R.id.btnAddTo0:
56             mModel.setValueFromIndex(0); //calls method from model that adds 1 to value and saves it
57             break;
58         case R.id.btnAddTo1:
59             mModel.setValueFromIndex(1); //calls method from model that adds 1 to value and saves it
60             break;
61         case R.id.btnAddTo2:
62             mModel.setValueFromIndex(2); //calls method from model that adds 1 to value and saves it
63             break;
64     }
65 }
66
67 //used for observable - when change is set on model this method is called which will update
68 //the view by updating the text to display the value from each index of list
69 @Override
70 public void update(Observable o, Object arg) {
71     mTextView1.setText("Index 0: " + mModel.getValueFromIndex(0));
72     mTextView2.setText("Index 1: " + mModel.getValueFromIndex(1));
73     mTextView3.setText("Index 2: " + mModel.getValueFromIndex(2));
74 }
75 }
```

Image 5: Screenshot of second half of MainActivity.java class

Once the app has loaded, the user can then press any of the three buttons to add 1 to the individual item of the list. The activity (view/controller) can seamlessly communicate with the Model which then updates the view dynamically. Image 6, 7, and 8 show how each index updates by 1 every time one of the buttons are pressed.

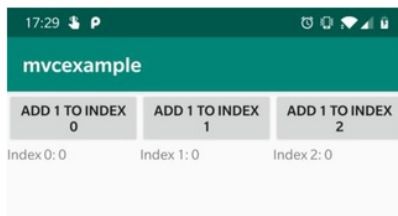


Image 6: Screenshot of application before any buttons are pressed

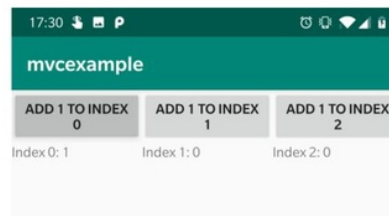


Image 7: Screenshot of application after left-most button pressed, index below is updated

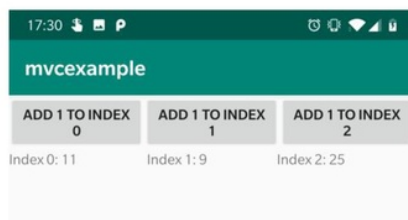


Image 8: Screenshot of application showing that each button is working



Assessment of MVC

Simić D and Plakaločić D wrote a journal article on applying MVC in mobile applications. According to the article [3], "MVC strictly separates the model from the user interface". This means that MVC allows the creation and execution of multiple views for one individual model. Multiple views can be opened and interacted with at the same time. Because of this, MVC can support "pluggable" controllers and views. A developer may want to replace and test with certain views or controllers without affecting the model. Another advantage of separating the model from the user interface is that the model can be transferred and ported to different platforms without sacrificing core functionality.

While easy to implement new features, MVC can, according to Simić D and Plakaločić D [3], "increase complexity without getting a lot of flexibility". This may be due to the nature of MVC introducing new levels of indirection. Implementing a small solution that may be simple could prove to be needlessly complicated, and could have adverse effects on application performance.

They [3] also discuss that MVC can create the "potential for an excessive number of updates". This could be caused by making frequent changes to the model which would cause all registered views to queue updates. This might lead to views such as pages taking some time to update and render.



Adapter Design Pattern

Erich Gamma and others [5] write about the intentions of adapter design pattern. The pattern allows two “classes [to] work together that couldn't otherwise because of incompatible interfaces”. This is done by using an adapter, sometimes known as a wrapper.

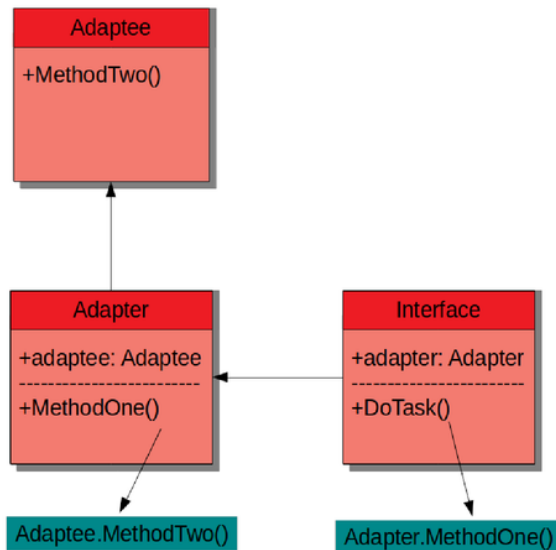


Figure 2: Illustration example of Adapter implementation

and a bridge, as according to Source Making [4], an adapter allows two classes/objects to work after they have been designed whereas a bridge is implemented before they are designed.

Figure 2 shows an illustration of how two classes can communicate with differing interfaces. We want to allow two classes (Interface and Adaptee) to communicate. We can implement an adapter layer between the two, acting as a 'bridge'. The end result allows the client to call a function from the Adapter instance, which in turn calls the function from the adaptee.

It's important to make a distinction between an adapter

Example of Adapter Pattern in Android Studio

In this example, a ViewPager will be used to demonstrate an Adapter class. A ViewPager allows us to have multiple views called fragments (in our case it will be imaged) and via an adapter, pass them to the ViewPager. The end result will allow the user to slide between each fragment within the ViewPager.

The main activity XML layout only consists of a constraint layout and a ViewPager widget. This has been given an ID of 'viewPager' to be called upon later, as seen in

Adam Howells



Image 9.

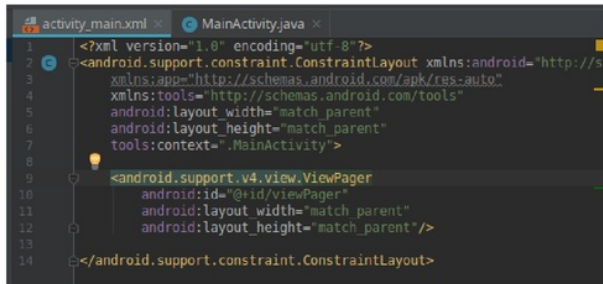


Image 9: Screenshot of activity_main.xml

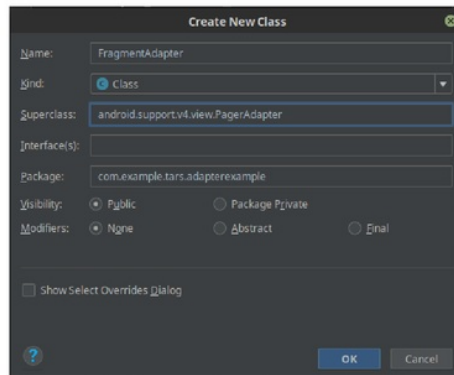


Image 10: Screenshot of creating adapter class using PagerAdapter as superclass

Next is creating the adapter for the fragments and ViewPager. Android Studio features a custom ViewPager called 'PagerAdapter' that are used specifically for our case of implementing multiple views to one layout activity. Android Studio's documentation will be used to develop the adapter [6]. Image 10 shows how this adapter class is made. Images 11 and 12 show the code of this class in Android Studio.



```
activity_main.xml x FragmentAdapter.java x MainActivity.java x
1 package com.example.tars.adapterexample;
2
3 import android.content.Context;
4 import android.support.annotation.NonNull;
5 import android.support.v4.view.PagerAdapter;
6 import android.view.View;
7 import android.view.ViewGroup;
8 import android.widget.ImageView;
9
10 public class FragmentAdapter extends PagerAdapter {
11     private Context mContext; //store context
12     //array to store images to reference them by ID
13     private int[] mImageIDs = new int[] {R.drawable.image1, R.drawable.image2, R.drawable.image3};
14
15     //method used to retrieve and store context in mContext variable
16     FragmentAdapter(Context context) {
17         mContext = context;
18     }
19
20     //returns number of views available - in this case it's number of images in array
21     @Override
22     public int getCount() {
23         return mImageIDs.length;
24     }
25
26     //works out if a page View is associated with a key object. Needed for PagerAdapter to function.
27     //Needed for viewPager to work out which view belongs to which item
28     @Override
29     public boolean isViewFromObject(@NonNull View view, @NonNull Object o) {
30         return view == o;
31     }
32 }
```

Image 11: Screenshot of first half of FragmentAdapter java class

```
32
33 //Creates the page for the given position (if user is on page 1, page 1 is loaded)
34 //In our case, an image view is declared and is set to image in array depending on user's
35 //position (which page).
36 //Ultimately, this method returns an object that is passed to isViewFromObject.
37 @NonNull
38 @Override
39 public Object instantiateItem(@NonNull ViewGroup container, int position) {
40     //creates new image view in activity context
41     ImageView imageView = new ImageView(mContext);
42
43     imageView.setScaleType(ImageView.ScaleType.CENTER_CROP); //sets image scale type
44
45     //sets an image from array to image view depending on position
46     imageView.setImageResource(mImageIDs[position]);
47
48     container.addView(imageView, index: 0); //adds imageView to contained view
49     return imageView;
50 }
51
52 //Removes a page depending on position (if user moves to page 2, page 1 is destroyed)
53 @Override
54 public void destroyItem(@NonNull ViewGroup container, int position, @NonNull Object object) {
55     container.removeView((ImageView) object);
56 }
57 }
```

Image 12: Screenshot of second half of FragmentAdapter java class

The application is able to build and can run without issues. This application allows the user to swipe between three images within the same activity. Images 13, 14, and 15 shows each fragment within the ViewPager.

Adam Howells



Image 15: Screenshot of page 1



Image 14: Screenshot of page 2



Image 13: Screenshot of page 3 and showing you cannot slide further

25th November 2018



Assessment of Adapter Pattern

The biggest benefit of using an adapter is the increased flexibility. Without this design pattern, at least one of the two classes will need to be redesigned to be compatible with the interface. According to Erich Gamma and others [5], the adapter pattern allows a class to be “reusable ... that cooperates with unrelated or unforeseen classes”. This allows a developer to not worry as much in making their code absolutely reusable, as an adapter can be implemented to allow a particular class or object to connect to any application programming interface (API).

Roshan Alexander [7], a Java developer, writes in his blog how “sometimes many adaptations are required along with an adapter”. This describes how adapter patterns can vary in complexity and how it works depending on how similar the target interface is to the adaptee’s interface. Erich Gamma and others [5] also discuss how an “adapter override[s] some of Adaptee’s behavior” due to the Adaptee becoming a subclass of the adapter. They also continue to point out how a class adapter will not work if the class *and* all corresponding subclasses need to be adapted.



3.1.2 Model-View-ViewModel (MVVM) Pattern

MVVM is an architectural pattern for implementing user interfaces. It's similar to MVC, but a controller is replaced with a ViewModel which then acts as a communicator between the View and the Model. Figure 3 shows each component

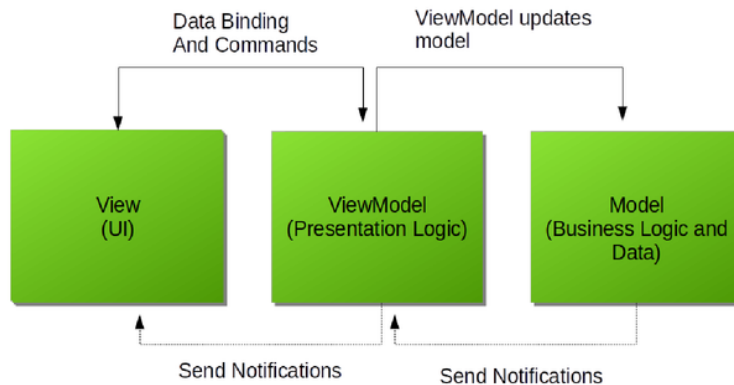


Figure 3: Illustration of MVVM pattern of the pattern along with their relationships.

The View is the component which handles the rendering of the user interface (UI) as well as any immediate user interaction, for example, an onClick method or other event listeners. Lars Vogel outlines additional responsibilities [8], such as starting an activity, working with Android widgets, and permissions.

According to Microsoft's documentation of MVVM in Xamarin [9], the ViewModel stores "properties and commands to which the view can data bind to". The ViewModel will consist of the UI logic that the View will call upon. The ViewModel can implement an Observable object, which allows for multiple Views to be Observers to the ViewModel. Any changes made to a ViewModel will then notify all Views in direct reference. Lars Vogel again specifies responsibilities of the ViewModel [8], such as calling to Model, executing methods in View, input validation, and exposing data and state.

The Model component in MVVM is very similar to the Model in MVC. The Model stores and encapsulates the data and behaviour of the application domain. For example, if the app was a train booking service, the model domain would consist of



classes for journeys and time or a class that connects to the business database.

Benefits and Drawbacks of MVVM in Android

MVVM in Android is relatively new, with Google releasing databinding libraries, as well as a ViewModel superclass. There are a few reasons why a developer may make use of MVVM over MVC or MVP (Model-View-Presenter).

MVVM holds the same benefit as MVC in regards to allowing “pluggable” Views. The ViewModel holds a one-to-many relationship to all registered Views, because of this, a developer has the flexibility to ‘swap’ out Views independent of the ViewModel or Model.

Bikesh Maharjan [10] performed a study where he developed a puzzle game in Android Studio using both MVC and MVVM to compare. In his findings, he identified that MVVM reduced “code size ... dramatically”. This has implications of having the code becomes easier to maintain. While it was a relatively small project, the effect will be most notable with large-scale applications.

The largest benefit of MVVM is the usage of isolated unit testing in regards to presentation logic. Unit testing is a style of application testing where the application is segmented, typically as separate applications, to test individual units of code such as methods.

According to John Gossman [11], an architect at Microsoft’s Azure Core Team wrote on his blog about the purpose and downfalls of MVVM. He states that “M-V-V-M can be overkill”. For smaller applications with simple UI’s, MVVM may make it harder to develop and maintain for. The ViewModel can also be difficult to initially design. While data-binding is one of the core features of MVVM, it can be tricky to debug and cannot rely on breakpoints.

Adam Howells



3.2 Part B: Implementation

3.2.1 Exercise 1

The Code

The code of the application consists of the main activity layout (as seen in Image 16), as well as the main activity class (as seen in Image 17-20). Screenshots are shown in chronological order of the code.

25th November 2018

Adam Howells

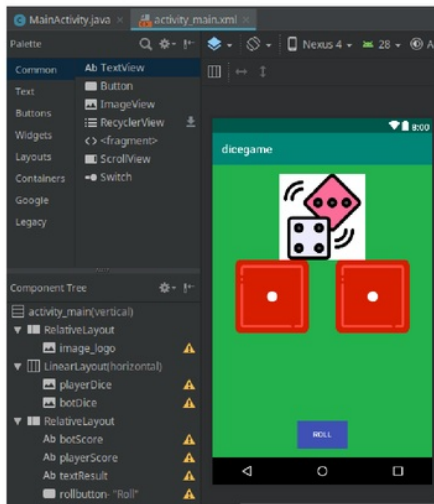


Image 16: Screenshot of `activity_main.xml` in design view

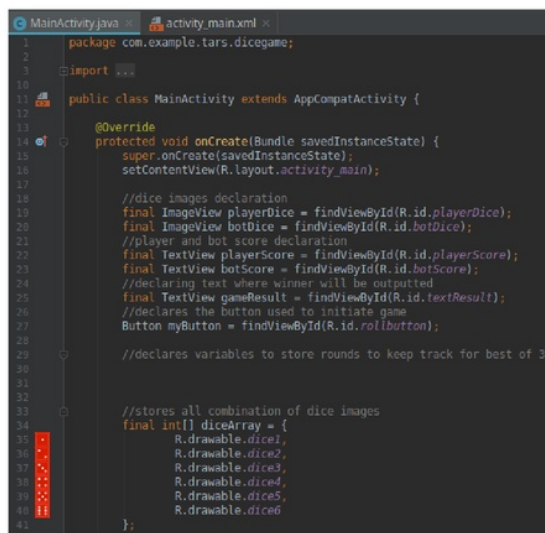


Image 17: Screenshot of beginning of `MainActivity.java`

25th November 2018



```

13 MainActivity.java x activity_main.xml x
14 myButton.setOnClickListener(new View.OnClickListener() {
15     int playerRoundsWon = initialiseNewMatch();
16     int botRoundsWon = initialiseNewMatch();
17     @Override
18     public void onClick(View view) {
19         //clicking the button generates a random number from 1-6 for both the player and the pc.
20         * The number is stored and the image of dice is updated to appropriate number.
21         * An if statement compares the random number and determines the winner.
22         * */
23
24         Random randomNumberGenerator = new Random(); //used to randomize a number
25
26         //PLAYER DICE NUMBER
27         final int intPlayerNumber = randomNumberGenerator.nextInt( bound: 6); //now number generated
28         playerDice.setImageResource(diceArray(intPlayerNumber)); //dice image is updated
29
30         //BOT DICE NUMBER
31         int intBotNumber = randomNumberGenerator.nextInt( bound: 6); //now number generated
32         botDice.setImageResource(diceArray(intBotNumber)); //dice image is updated
33
34         //switch statement to handle who won, whoever won their round storage is updated.
35         switch (winnerOfRound(intPlayerNumber, intBotNumber)) { //passes player and bot dice numbers to test winner
36             case 0:
37                 playerRoundsWon++;
38                 playerScore.setText(Integer.toString(playerRoundsWon)); //current rounds won displayed
39                 break;
40             case 1:
41                 botRoundsWon++;
42                 botScore.setText(Integer.toString(botRoundsWon)); //current rounds won displayed
43                 break;
44             case 2:
45                 //draw - no points awarded
46                 break;
47         }
48     }
49 }

```

Image 18: Screenshot of first half of button on click listener

```

50 MainActivity.java x activity_main.xml x
51         botRoundsWon++;
52         botScore.setText(Integer.toString(botRoundsWon)); //current rounds won displayed
53     case 2:
54         //draw - no points awarded
55         break;
56     }
57
58     //tests to see if win conditions are met (either player or bot reaches 2 rounds)
59     switch (checkWinnerOfMatch(playerRoundsWon, botRoundsWon)) {
60         case 0:
61             gameResult.setText("You've Won!"); //displayed when player wins
62             //Resets rounds
63             playerRoundsWon = initialiseNewMatch();
64             botRoundsWon = initialiseNewMatch();
65             //updates textView after round is reset
66             playerScore.setText(Integer.toString(playerRoundsWon));
67             botScore.setText(Integer.toString(botRoundsWon));
68             break;
69         case 1:
70             gameResult.setText("You've Lost!"); //displayed when player loses
71             //Resets rounds
72             playerRoundsWon = initialiseNewMatch();
73             botRoundsWon = initialiseNewMatch();
74             //updates textView after round is reset
75             playerScore.setText(Integer.toString(playerRoundsWon));
76             botScore.setText(Integer.toString(botRoundsWon));
77             break;
78         case 2: //if win conditions aren't met, do nothing
79             break;
80     }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }
101 }
102 }

```

Image 19: Screenshot of final portion of button on click listener



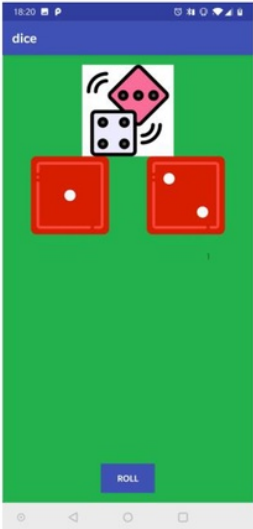
```
194 public int initialiseNewMatch() {
195     return 0;
196 }
197
198 public int winnerOfRound(int playerScore, int botScore) {
199     if (playerScore > botScore) {
200         return 0;
201     }
202     else if (botScore > playerScore) {
203         return 1;
204     }
205     else {
206         return 2;
207     }
208 }
209
210 public int checkWinnerOfMatch(int playerRoundsWon, int botRoundsWon) {
211     //Determines winner of game
212     if (playerRoundsWon == 2) {
213         return 0;
214     }
215     else if (botRoundsWon == 2) {
216         return 1;
217     }
218     else {
219         return 2;
220     }
221 }
222 }
```

Image 20: Screenshot of the rest of main activity class, includes methods used in on click listener

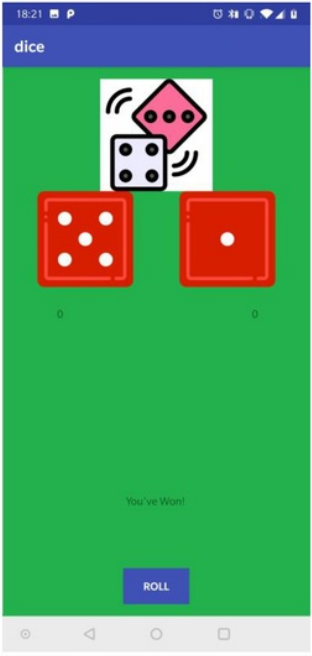
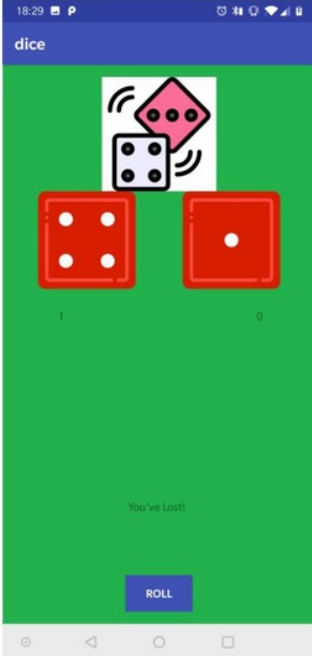
Testing

Test	Expectations	Result	Evidence
Launching Application	I expect the application to launch the main activity without any crashes.	The application loaded the main activity very quickly. All layout elements are present.	



Pressing Button (Round 1)	By pressing the button, I expect the first round to begin. If the player's or bot's dice is larger than the other, they will gain a point.	The bot's dice rolled a 2 whereas the player rolled a 1. As expected in this case, the bot gains one point.	
Draw Situation	I expect neither the player nor the bot to gain a point if both dice number are equal.	As expected, when both dice numbers are equal, neither opponent is awarded a point.	



Pressing Button (Final Round)	I expect whoever reaches 2 points first will get receive a text view correctly congratulating the player or the bot. The final scores on both sides should also be displayed. The next button press will then reset the match.	Before the final round, both the player and the bot had 1 point. In this round, the player scores a point and the correct text display of "You've Won!" is correct. However, both scores were reset to 0 as soon as a win condition was met, not when the player presses the button again for a new match.	
Pressing Button (Resetting Match)	I expect pressing the button once the final round is over will reset the match and both opponents' points are reset to 0.	The TextView that states who have won do not get updated to invisible once the next match starts. The points unexpectedly reset during the final round.	

Adam Howells



There are also no labels defining which dice is the player and which is the bots. The TextView that displays how many rounds each opponent has received could also be more clearly labeled. In the future, I'll take these considerations into account.

3.2.2 Exercise 2

The Code

25th November 2018

Adam Howells

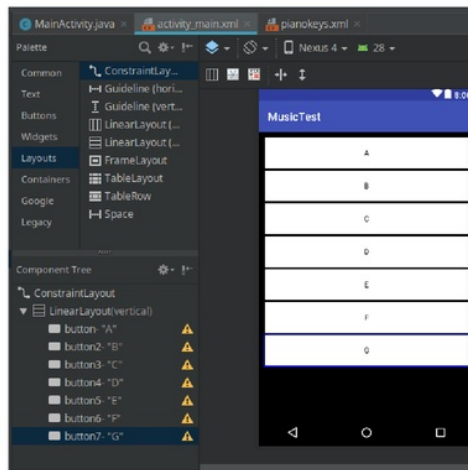


Image 21: Screenshot of `activity_main.xml` in design view

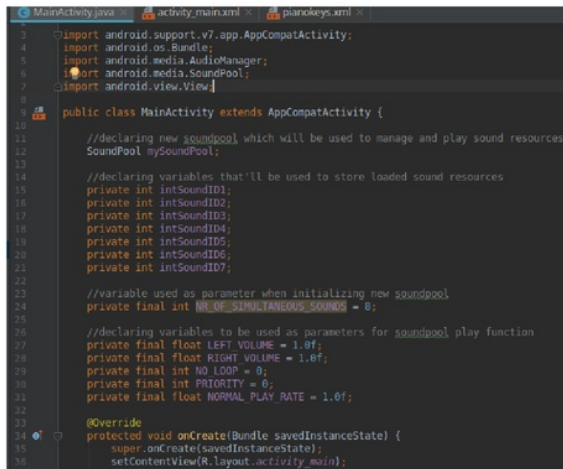


Image 22: Screenshot of first half of `MainActivity.java` class

25th November 2018



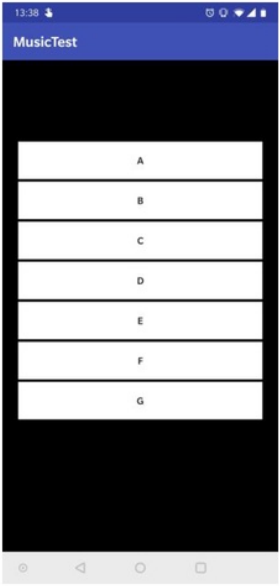
```

38 //initializes soundpool
39 mySoundPool = new SoundPool(NR_OF_SIMULTANEOUS_SOUNDS, AudioManager.STREAM_MUSIC, "frequency: 0");
40
41 //loads sound files to their respective ID's
42 intSoundID1 = mySoundPool.load(getApplicationContext(), R.raw.piano_a, priority: 1);
43 intSoundID2 = mySoundPool.load(getApplicationContext(), R.raw.piano_b, priority: 1);
44 intSoundID3 = mySoundPool.load(getApplicationContext(), R.raw.piano_c, priority: 1);
45 intSoundID4 = mySoundPool.load(getApplicationContext(), R.raw.piano_d, priority: 1);
46 intSoundID5 = mySoundPool.load(getApplicationContext(), R.raw.piano_e, priority: 1);
47 intSoundID6 = mySoundPool.load(getApplicationContext(), R.raw.piano_f, priority: 1);
48 intSoundID7 = mySoundPool.load(getApplicationContext(), R.raw.piano_g, priority: 1);
49
50 }
51
52
53
54 /* Methods are triggered when a button is pressed. Each method is used as an on click event which
55 is linked in activity_main.xml. Each method executes soundpool play function for each sound
56 file.
57 */
58
59 public void playA(View v)
60 {
61     mySoundPool.play(intSoundID1, LEFT_VOLUME, RIGHT_VOLUME, PRIORITY, NO_LOOP, NORMAL_PLAY_RATE);
62 }
63
64 public void playB(View v)
65 {
66     mySoundPool.play(intSoundID2, LEFT_VOLUME, RIGHT_VOLUME, PRIORITY, NO_LOOP, NORMAL_PLAY_RATE);
67 }
68
69 public void playC(View v)
70 {
71     mySoundPool.play(intSoundID3, LEFT_VOLUME, RIGHT_VOLUME, PRIORITY, NO_LOOP, NORMAL_PLAY_RATE);
72 }

```

Image 23: Screenshot of last portion of main class (play methods continue from playA to playG)

Testing

Test	Expectations	Result	Evidence
Launching the Activity	I expect the application to launch with no errors, loading in the main activity layout.	The application loaded and launched main activity with no errors.	
Pressing All	I expect each	All 7 buttons	A screen capture was performed to



7 Buttons	button to play their corresponding note. For example, A button plays A note.	play their correct notes with no issues. The sound quality is low, but that may have to do with the sound files themselves.	record the app and audio. The video file is 'pianoapp.mp4'.
Holding Down a Button	I expect the note to only play once instead of queue up many sound files to play.	Holding down the button doesn't play anything, releasing the button is what triggers the onClick event to occur.	A screen capture was performed to record the app and audio. The video file is 'pianoapp.mp4'.

3.2.3 Exercise 3

The Code

The layout is first created using a simple combination of a text view and two buttons as seen in Image 24.

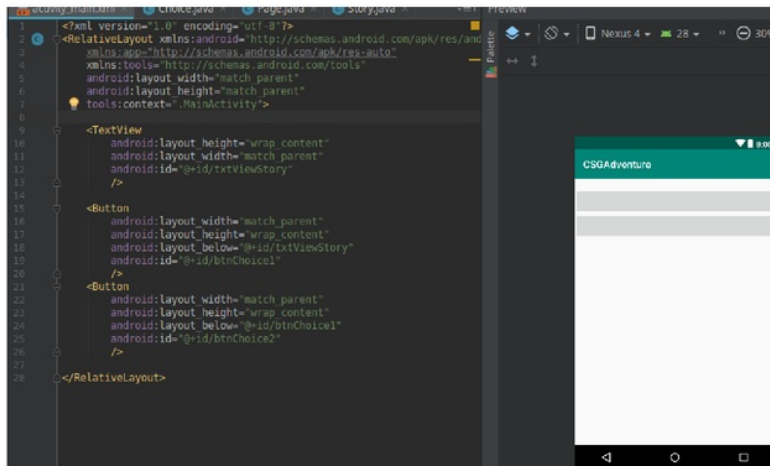


Image 24: Screenshot of activity_main.xml in text adventure game application

Adam Howells



This application will heavily feature examples of encapsulation of objects. Firstly, an object representing choice was made as seen in Image 25. This holds the properties of what the choice is as a string, as well as an integer value of which page to load next. Getter methods were declared to be able to pass the values of these properties.

```
activity_main.xml | Choice.java | Page.java | Story.java | MainActivity.java
package com.example.fairs.csqadventure;

//
// Choice is an object that'll contain the properties of what the text of the choice options are,
// as well as what the next page is as a result of the choice. There will be two options per page,
// so two choice objects will be initialised as part of the properties of the page object.
//
public class Choice {
    private String mDisplayText; //To store the text that'll display on the button
    private int mNextPage; //To store the ID of next page to determine what will load when user clicks button

    //this method is a constructor that initializes this object
    public Choice(String text, int nextPage) {
        mDisplayText = text;
        mNextPage = nextPage;
    }

    //getters and setters to store and retrieve from class
    public String getDisplayText() {return mDisplayText;}
    public int getNextPage() {return mNextPage;}

    public void setDisplayText(String mDisplayText) {this.mDisplayText = mDisplayText;}
    public void setNextPage(int mNextPage) {this.mNextPage = mNextPage;}
}
```

Image 25: Screenshot of Choice.java class

Next, a page object was made as seen in Image 26. This object will store the properties of the strings of the story text as well as both choice dialogues. This object will have many initializations contained in an array. Getters were also used to retrieve values of properties outside the class.

```
activity_main.xml | Choice.java | Page.java | Story.java | MainActivity.java
package com.example.fairs.csqadventure;

//
// Page is an object that'll be used to create multiple story descriptions and scenarios. This will
// be used as an array.
//
public class Page {
    private String mDisplayText; //used to store the story text for each page
    //declaring two choice objects for the user to choose from
    private Choice mChoice1;
    private Choice mChoice2;

    //object constructor: three properties.
    public Page(String text, Choice choice1, Choice choice2) {
        mDisplayText = text;
        mChoice1 = choice1;
        mChoice2 = choice2;
    }

    //getters to be able to retrieve values from this class
    public String getDisplayText() {return mDisplayText;}
    public Choice getChoice1() {return mChoice1;}
    public Choice getChoice2() {return mChoice2;}

    //setters to be able to store values to this class
    public void setDisplayText(String mDisplayText) {this.mDisplayText = mDisplayText;}
    public void setChoice1(Choice mChoice1) {this.mChoice1 = mChoice1;}
    public void setChoice2(Choice mChoice2) {this.mChoice2 = mChoice2;}
}
```

Image 26: Screenshot of Page.java class

25th November 2018

Adam Howells



As seen in Images 27 and 28, our story class will store the declared array that will store multiple pages. Each page in the array will contain the story text to display, as well as two choice objects which both store the choice dialogue and the index of next page. A getter method is used to retrieve and load pages.

```
activity_main.xml x Choice.java x Page.java x Story.java x MainActivity.java x
4 A class that will store my story for each page and allow navigating through the story.
5
6
7 class Story {
8     private Page[] mPages; //declare new page object of an array type to create multiple pages
9
10    public Story() {
11        mPages = new Page[14]; //initializes page array with a length of 14 pages
12
13        /*ALL FOLLOWING CODE IN THIS METHOD IS CREATING MULTIPLE PAGE OBJECTS
14        THAT WILL STORE THE STORY TEXT, TEXT OF THE CHOICE, AND WHICH INDEX IN PAGE ARRAY
15        THAT OPTION SHOULD LOAD NEXT.*/
16
17        //beginning page - deaths, restarts, and first time loading will be directed here
18        mPages[0] = new Page(
19            text: "Story 0",
20            new Choice(text: "To Story A1", nextPage: 1),
21            new Choice(text: "To Story B1", nextPage: 7));
22
23
24        //A Path
25        mPages[1] = new Page(
26            text: "Story A1",
27            new Choice(text: "To Story A2", nextPage: 3),
28            new Choice(text: "To Story A1: Death", nextPage: 2));
29
30        mPages[2] = new Page(
31            text: "Story A1: Death",
32            new Choice(text: "Restart", nextPage: 0),
33            new Choice(text: "Go Back", nextPage: 1));
34
35        mPages[3] = new Page(
36            text: "Story A2",
37            new Choice(text: "To Story A3", nextPage: 4),
38            new Choice(text: "To Story B3", nextPage: 9));
39
40        mPages[4] = new Page(
41            text: "Story A3",
42            new Choice(text: "To Story A Finish", nextPage: 6),
43            new Choice(text: "To Story B3", nextPage: 9));
44    }
```

Image 27: Screenshot of first half off Story.java class

```
activity_main.xml x Choice.java x Page.java x Story.java x MainActivity.java x
52 //B Path
53 mPages[7] = new Page(
54     text: "Story B1",
55     new Choice(text: "To Story B2", nextPage: 8),
56     new Choice(text: "To Story B3", nextPage: 9));
57
58 mPages[8] = new Page(
59     text: "Story B2",
60     new Choice(text: "To Story A3", nextPage: 4),
61     new Choice(text: "To Story B4", nextPage: 11));
62
63 mPages[9] = new Page(
64     text: "Story B3",
65     new Choice(text: "To Story B3: Death", nextPage: 10),
66     new Choice(text: "To Story B4", nextPage: 11));
67
68 mPages[10] = new Page(
69     text: "Story B3: Death",
70     new Choice(text: "Restart", nextPage: 0),
71     new Choice(text: "Go Back", nextPage: 10));
72
73 mPages[11] = new Page(
74     text: "Story B4",
75     new Choice(text: "To Story B4: Death", nextPage: 12),
76     new Choice(text: "To Story B Finish", nextPage: 13));
77
78 mPages[12] = new Page(
79     text: "Story B4: Death",
80     new Choice(text: "Restart", nextPage: 0),
81     new Choice(text: "Go Back", nextPage: 11));
82
83 mPages[13] = new Page(
84     text: "Story B Finish",
85     new Choice(text: "Restart", nextPage: 0),
86     new Choice(text: "Restart Harder", nextPage: 0));
87
88 }
89
90 //method will be called in main activity class to get page object depending on index
91 public Page getPage(int pageNumber) {
92     return mPages[pageNumber];
93 }
94 }
```

Image 28: Screenshot of second half of Story.java class

Main Activity (Image 29 and 30) is where we will link the elements of the layout to

25th November 2018

Adam Howells



variables. We will set these variables in the load page method that calls the get methods created to retrieve story text and choice text. Two on click listeners for both buttons are then used to retrieve which page to load next, to which the load page method will call upon itself, passing the new index for the new page.

```
1 package com.example.tars.csgadventure;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 public class MainActivity extends AppCompatActivity {
6
7     private Story mStory = new Story(); //creates new story instance
8
9     //declaring variables that will be used for the elements on layout xml file
10    private TextView mTextView;
11    private Button mChoice1;
12    private Button mChoice2;
13
14    //used to keep track of current page
15    private Page mCurrentPage;
16
17    @Override
18    protected void onCreate(Bundle savedInstanceState) {
19        super.onCreate(savedInstanceState);
20        setContentView(R.layout.activity_main);
21
22        //Initialises text view for story text, and buttons for two choices
23        mTextView = (TextView) findViewById(R.id.txtViewStory);
24        mChoice1 = (Button) findViewById(R.id.btnChoice1);
25        mChoice2 = (Button) findViewById(R.id.btnChoice2);
26
27        loadPage(mChoice1); //on create, the beginning page will load
28    }
29
30    //method that loads and updates to page corresponding to user's choice
31    private void loadPage(int choice) {
32        //gets user's choice and sets it as current page - default is 0 which is beginning page
33        mCurrentPage = mStory.getPage(choice);
34
35        String pageText = mCurrentPage.getDisplayText(); //variable that loads text on current page
36        mTextView.setText(pageText); //sets retrieved text to story text view
37    }
38}
```

Image 29: Screenshot of first half off MainActivity.java class

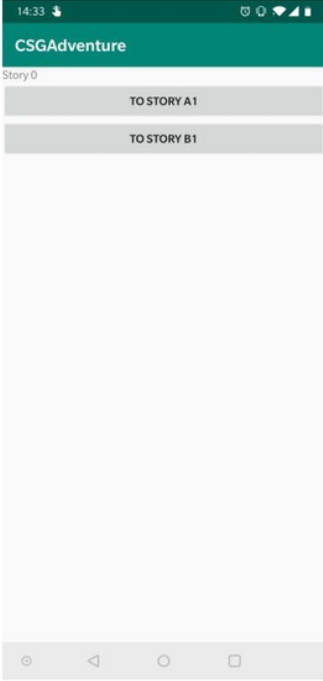
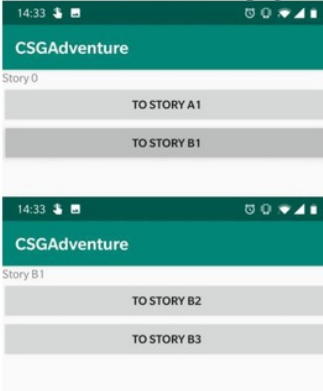
```
34
35 //method that loads and updates to page corresponding to user's choice
36 private void loadPage(int choice) {
37     //gets user's choice and sets it as current page - default is 0 which is beginning page
38     mCurrentPage = mStory.getPage(choice);
39
40     String pageText = mCurrentPage.getDisplayText(); //variable that loads text on current page
41     mTextView.setText(pageText); //sets retrieved text to story text view
42
43     //sets display text for each option for both buttons
44     mChoice1.setText(mCurrentPage.getChoice1().getDisplayText());
45     mChoice2.setText(mCurrentPage.getChoice2().getDisplayText());
46
47     //on click listeners that reloads the loadpage method depending on user's choice
48
49     //choice listener 1
50     mChoice1.setOnClickListener(new View.OnClickListener() {
51         @Override
52         public void onClick(View v) {
53             int nextPage = mCurrentPage.getChoice1().getNextPage(); //fetches which page to load next
54             loadPage(nextPage); //updates current page with new page
55         }
56     });
57
58     //choice listener 2
59     mChoice2.setOnClickListener(new View.OnClickListener() {
60         @Override
61         public void onClick(View v) {
62             int nextPage = mCurrentPage.getChoice2().getNextPage(); //fetches which page to load next
63             loadPage(nextPage); //updates current page with new page
64         }
65     });
66 }
67 }
```

Image 30: Screenshot of second half of MainActivity.java class

25th November 2018



Testing

Test	Expectations	Result	Evidence
Launching Activity	I expect the application to launch the main layout with no errors.	The application launched and the activity was loaded.	
Checking if Choice buttons behave correctly	I expect each button to load the next correct page. Once reaching the end, I expect the restart buttons to restart to the first page. I also expect deaths to cause the buttons to either allow restarting or going back a page.	By pressing each choice button, the next page is loaded correctly. When on death, both the 'restart' and 'go back' button behave as expected. Endings only feature a restart option, which also behaves as expected.	Pressing the choice to go to story B1 will take me to the B1 page.  Pressing restart of ending will load the first page.



			14:34 CSGAdventure Story A Finish RESTART RESTART HARDER 14:33 CSGAdventure Story 0 TO STORY A 1 TO STORY B 1 Pressing restart on death will load the first page. 14:33 CSGAdventure Story B3: Death RESTART GO BACK 14:33 CSGAdventure Story 0 TO STORY A 1 TO STORY B 1
Checking if Story Text is correct	I expect the first story text to read 'Story 0', with each corresponding button linking to the correct page. E.g. pressing choice 'A1' will lead me to 'A1' page with correct story text.	By pressing the choice button 'Story A1', the next page is loaded with the story text 'A1'.	By pressing the choice to go to B1 page, the story text correctly reads 'B1'. 14:33 CSGAdventure Story 0 TO STORY A 1 TO STORY B 1 14:33 CSGAdventure Story B1 TO STORY B 2 TO STORY B 3

Adam Howells



Exploring all choice flows	I expect to find at least one page where the story text doesn't match the choice dialogue, or a choice button leading to an incorrect page.	Surprisingly, all pages with corresponding story text and choice buttons are correct.	N/A
----------------------------	---	---	-----

25th November 2018



4 Conclusion

From my findings, I can conclude that MVC is most appropriate when implementing UI solutions. Implementing MVC across the entire application may make matters more complicated than is necessary. Due to its confusing application flow, I would not recommend using MVC in small level applications.

An adapter is a useful tool for developers to allow two incompatible interfaces to communicate. It also allows developers to develop solutions that can be integrated with any API in the future. However, the amount of work and the number of adapters can vary, in which case the adaptee may need to be redesigned for smaller projects if the cost outweighs the benefit.

MVVM is like MVC but has a few distinct advantages such as the ability to unit test. It's highly dynamic and easy to maintain, however for small applications it may make it harder to maintain the code. The initial planning time also needs a large amount of time due to the specification of the ViewModel.

The exercises undertaken allowed me to demonstrate skills that were developed during this assignment. There are many areas I can see for improvement in the user interface for all exercises.



Bibliography

- [1] M. Richards, 'Software Architecture Patterns', p. 54. [Online]. Available: <https://www.oreilly.com/programming/free/files/software-architecture-patterns.pdf>. [Accessed: 30-Nov-2018].
- [2] M. Kalelkar, P. Churi, and D. Kalelkar, 'Implementation of Model-View-Controller Architecture Pattern for Business Intelligence Architecture', *Int. J. Comput. Appl.*, vol. 102, no. 12, pp. 16–21, Sep. 2014.
- [3] D. Plakalovic and D. Simic, 'Applying MVC and PAC patterns in mobile applications', *ArXiv10013489 Cs*, Jan. 2010. [Online]. Available: <http://arxiv.org/abs/1001.3489>. [Accessed: 30-Nov-2018].
- [4] 'Design Patterns and Refactoring'. [Online]. Available: <https://sourcemaking.com/>. [Accessed: 30-Nov-2018].
- [5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. [Online]. Available: [https://sophia.javeriana.edu.co/~cbustaca/docencia/DSBP-2018-01/recursos/Erich%20Gamma,%20Richard%20Helm,%20Ralph%20Johnson,%20John%20M.%20Vlissides-Design%20Patterns_%20Elements%20of%20Reusable%20Object-Oriented%20Software%20%20-Addison-Wesley%20Professional%20\(1994\).pdf](https://sophia.javeriana.edu.co/~cbustaca/docencia/DSBP-2018-01/recursos/Erich%20Gamma,%20Richard%20Helm,%20Ralph%20Johnson,%20John%20M.%20Vlissides-Design%20Patterns_%20Elements%20of%20Reusable%20Object-Oriented%20Software%20%20-Addison-Wesley%20Professional%20(1994).pdf). [Accessed: 30-Nov-2018].
- [6] 'PagerAdapter', *Android Developers*. [Online]. Available: <https://developer.android.com/reference/android/support/v4/view/PagerAdapter>. [Accessed: 18-Dec-2018].
- [7] R. Alexander, 'Adapter pattern | Java Revealed', *Adapter Design Pattern*, 12-Sep-2013. [Online]. Available: <https://javarevealed.wordpress.com/tag/adapter-pattern/>. [Accessed: 30-Nov-2018].
- [8] 'Android Architecture with MVP or MVVM - Tutorial'. [Online]. Available: <http://www.vogella.com/tutorials/AndroidArchitecture/article.html>. [Accessed: 20-Dec-2018].
- [9] David Britch, 'The Model-View-ViewModel Pattern - Xamarin'. [Online]. Available: <https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm>. [Accessed: 30-Nov-2018].
- [10] B. Maharjan, 'Puzzle game using Android MVVM Architecture', p. 38. [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/145871/Maharjan_Bikesh.pdf?sequence=1. [Accessed: 30-Nov-2018].
- [11] J. Gossman, 'Advantages and disadvantages of M-V-VM – Tales from the Smart Client', 04-Mar-2006. [Online]. Available: <https://blogs.msdn.microsoft.com/johngossman/2006/03/04/advantages-and-disadvantages-of-m-v-vm/>. [Accessed: 30-Nov-2018].