Bikesh Maharjan

Puzzle game using Android MVVM Architecture

Metropolia University of Applied Sciences
Bachelor of Engineering
Information and CommunicationsTechnology
Thesis
30 April 2018

Author Title Number of Pages Date	Bikesh Maharjan Puzzle game using Android MVVM Architecture 31 pages 30 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information and CommunicationsTechnology
Professional Major	Software Engineering
Instructors	Kari Salo, Principal Lecturer and Head of Mobile Solutions

The development of Android mobile applications in application industry is growing rapidly. Application requirements change frequently and demand for code structure adjustment with addition of new features. Thus, flexibility and maintainability in a software architecture is a determining factor for an application's success. The objective of this thesis is to implement Model View ViewModel (MVVM) in a simple 8-puzzle game. The development of this application is carried out in Android Studio IDE. This application uses API level 15 or above to give access for more android devices. The main programming language used in this application is Java. This thesis also explains different architecture concepts used in Android such as Model View Controller, Model View Presenter and finally Model View ViewModel. This study also specifies the uses of best practices in software development. Also, downsides of an architecture such as Model View Controller are also pointed out and the alternatives presented.

This thesis uses a Slider puzzle application written in Java to implement MVVM architecture pattern. This was previously written in MVC architecture pattern. In conclusion, the thesis illustrates how the use of a well-designed architecture could affect the overall quality of an application in terms of flexibility and maintainability. It also encourages a best-practice minded approach in software development and further studies toward the implementation of the Model View ViewModel architecture in Android.

Keywords	Architecture, MVVM, MVC, MVP

Contents

List of Abbreviations

IDE Integrated Development Environment

UI User Interface

SDK Software Development Tools

JDK Java Development Kit

JRE Java Runtime Environment

API Application Programming Interface

AVD Android Virtual Device

MVC Model View Controller

MVP Model View Presenter

MVVM Model View ViewModel

1 Introduction

Since the proliferation of mobile devices, the demand for a mobile application development is skyrocketing, especially in Android platform because of an advantage of open source for the developers. To survive in a tough competition with iOS and other platform, an Android mobile application should be cost-efficient and of good quality.

A report published by Statista.com, on December 2017, shows there is around 3.5 million available applications at Google play store [1]. In comparison to applications published on March 2017, there was approximately 2.8 million of them, which shows that the growth rate is 25**%** within 9 months period [2]. The favorable growth of Android applications results in a competition for market shares. A report published in white paper by Intelliware pointed out that there are several elements that can make one app more successful than another [3].

For an application to be successful, contributing factors are business value delivery, branding protection are application technical sides: optimization, scalability and extensibility. The most important element is application growth, hence, it should be carefully arranged in the early stage of development. This indicates the importance of developing the application based on a testable, scalable and maintainable architecture. This thesis explores the role of software architecture in application development and different architecture patterns that can be used in Android software development.

For many years, the Model View Controller (MVC) architecture has been preferred by developers. But, on arrival of an additional data and business logic, the View Controller class will be overloaded, because View Controller cannot be separated from the View, so it will neither be reusable nor testable. Therefore, MVC is often abbreviated as Massive View Controller. As an alternative, Model View Presenter (MVP) is the preferred architectural model over MVC.

Recently, Google announced a better pattern namely Model View ViewModel (MVVM) as an alternative to MVP. MVVM pattern was introduced by Microsoft a few years ago and it has been widely used in WPF, Silverlight and JS environments. However, this pattern is a new approach in Android development. In MVVM, View Model is an independent business domain, which handles all the data transformation and operations.

The objective of this thesis is to implement MVVM architecture pattern in a puzzle game and observe the differences with MVC pattern. It also attempts for better pattern, which helps developers to write the code and test it in an easier way.

2 Theoretical Background

2.1. Software Architecture

An architecture is the combination of business plan and technical details. It is not just the result of the functional requirement of a system. It is a plan that describes a system from many perspectives, all of which are focused on business goals. This plan clearly explains overall system structure, describes its various components, indicates how the components fit together, and defines the rules and standards that govern their behavior. Architecture is influenced by system stakeholders, organizations developing it, the experience and background of an architect, and technical environment. These influences will change based on the environment where the architecture is to be used. Following factors should be kept in mind while developing an architecture: you need a reference here!

- Creating clearly defined business case for a system which explains cost, targeted market and targeted time to market the product
- Understanding requirements by creating prototype which make the system real
- Selecting or Creating appropriate architecture
- Documenting the architecture and communicating it to stakeholders, developers, testers and management department
- Analyzing or Evaluating the architecture for qualities which is appropriate for a system
- Implementing and ensuring that the architecture is communicated and represented well

Software architecture defines software elements and it is an important part in a software development process. According to Bass et al. “The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.” [9, p. 21]. In general, the purpose of software architecture is to serve a communication tool with stakeholders.

The goal of software architecture is to focus on development resources and building the right software for a situation, with a limited wastage of time and effort as possible. It helps to avoid reinventing the wheel so that solution of software problem someone has already solved can be used. In the development of software production, changes occur frequently, and flexibility is a key factor of a success while using any architectural pattern. As the architect and the development team progress through the phases of development cycle and obtain more ideas about the project, its requirements and technology are adjusted repeatedly. Architecture should be precise to show the development plan, offer a working framework, but also remain flexible so that changes can be made according to the circumstances.

As described by Hanmer (2012) in Pattern-Oriented Software architecture for dummies, a proper choice of architecture can satisfy all the requirement of the business constraints and technical constraints. In software architecture, there are several audiences, including architects, developers, programmers, configuration managers, testers and end users [10]. These audiences are interested in different things in the architecture and to achieve it, an architecture is divided into four main models.

Logical View

The logical view is connected to the functional requirements and it focuses on the parts of the system that provide the functionality and that the users of the system will see when they interact with it. It divides the system into classes and components.

Process View

Processes are groups of tasks put together making something that can execute and perform a desired function. It explains how those parts of an architecture work together and how the parts stay synchronized. It also explains how the system is mapped onto the units of computing, such as processes and threads.

The process view brings in some nonfunctional requirements that are not directly related to visible functions.

Physical View

It shows how the software that implements the system is mapped onto the computing platforms. The various components of the system, networks, processes, tasks, and objects are mapped onto the tangible parts of the system in the physical view. This view contains information related to the system's nonfunctional requirements such as availability, performance, and scalability.

Development View

It explains how the software will be managed during development. The software will be written in small pieces that individuals or small teams can work on them together. The development view highlights these pieces and shows how they are intertwined and are interdependent. The development view reflects any limitations on the organization of the software based on limitations in the programming language and development environment. (See Figure 1)

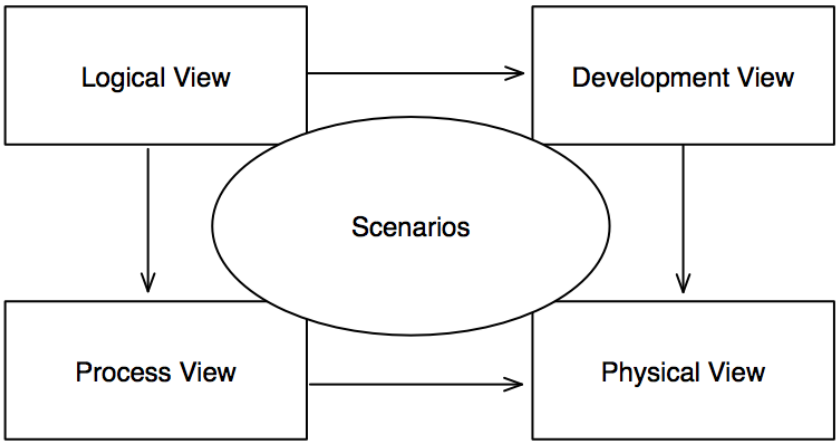

Figure 1: The 4+1 model of an architecture.

Figure 1 illustrates the categories of view, together forming an overall architecture model. It divides the view in an individual category and describes the problems, hence solving the problem in software architecture. For example, logical view describes the user functionality.

These four models are supplemented with an additional view, which defines common scenarios combining all view together by showing how they all work together. This additional view is often called 4+1 view [10, p. 11-12].

Selection of an appropriate architecture helps to produce a working software. So, architectural quality can be determined by measuring the quality of the finished software using common attributes such as [7,9].

- Availability
- Modifiability
- Performance
- Security
- Testability
- Usability
- Scalability
- Maintainability
- Reliability

Software architecture helps avoiding system failure and consequences associated with it. Software architecture also helps to avoid reinventing same thing again by using solution of software problem someone else has already solved. Without a proper architecture, the program tends to become chaotic over time thus making small changes to the application difficult to accomplish.

2.2 Architecture Patterns

In the development of apps using Android architecture, it does not require any model, so the quality of an application and architecture choice purely depends on the experience of a developer. In contrast, the development of iOS application is based on Model View Controller architecture which separates user interface (view) and business rules and data (model) using a mediator (controller) to connect the model to the view [4]. Hence, it is important to choose certain architecture pattern to ensure the quality of implementation and reduce development time. Sharing project with designers, flexibility on both design work and development work to happen simultaneously is achievable with the use of an architecture pattern.

In a small application, where there is only a few number of activity screens it does not matter which architecture pattern is used although it is good practice to follow one. However, for a large application, which utilizes a multiple number of activity and fragments it is best to organize an Android application into some logical components. Certain number of architecture patterns can be used in Android, which are discussed below.

2.2.1. MVC (Model View Controller)

Model View controller in short MVC, was first introduced in Smalltalk in 1980s became influential design pattern for developing a rich graphical user interface application [5]. This pattern separates the application in three main sets of responsibilities i.e. Model, View and Controller. These modules are separated with one another by an abstract boundary and communication occurs within the boundary as illustrated in Figure 2.

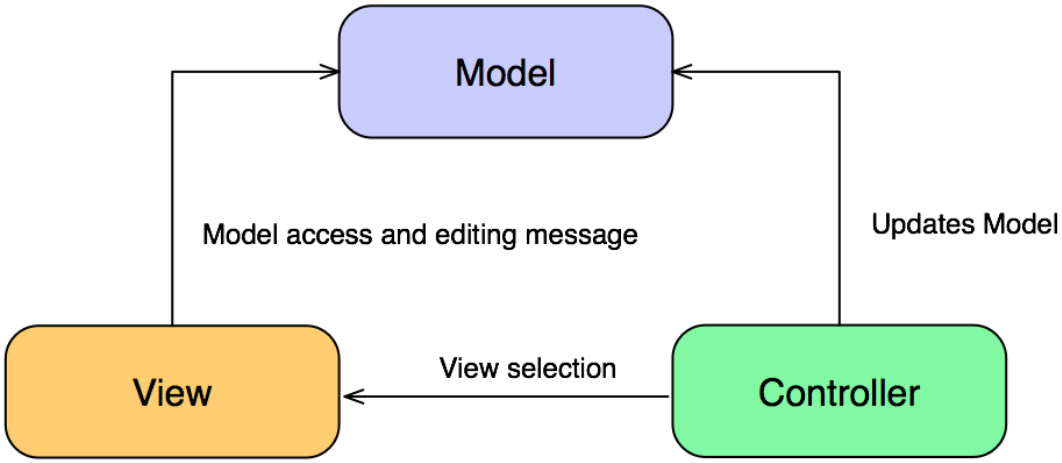

Figure 2: Model View Controller Class structure

Figure 2 shows the communication between the components in MVC pattern. The controller as a mediator has a role to update the model and update or select the view. In addition, the view is also linked with a model to access and edit the messages.

Model

Model represents the actual data or information in an application. Model can be a single object, or it can also be structure of objects. For example, a model can be contact information where name, phone number and address are its fields.

Model represents information, but it does not hold either the behavior or any services that can manipulate it. Additionally, a model is also not responsible for formatting text, display on screen and fetching a data from remote server.

View

View, as the name implies is a user interface and does the presentation of data, which is in a Model. View can also be expressed as the pictorial representation of a model.

View is the only element, which the end user interacts with and makes the data presentation easier. In MVC, View is aware of the Model and any changes that occur in Model is notified to the view. Ideally, Model and View communicate through a Controller.

Controller

Controller is the mediator between the Model and View. As described in Figure 2, it connects the user interface to the data, but it also creates a layer of separation between the Model and the View. When user interacts within a View, for example by clicking a button, the controller decides on how to interact with the model. Typically, in Android application, the controller is represented by an Activity or Fragments. [8]

Advantages

The Model-View-Controller pattern supports the separation of concerns. It separates model, controller and view. This increases the testability of the code and makes extending and allowing an easy implementation of new features easier.

The Model classes do not contain any reference to Android classes and are therefore easier to unit test. Also, the Controller does not extend or implement any Android classes and should contain a reference to an interface class of the View. In conclusion, unit testing the Controller becomes possible.

In the case when Views follows the single responsibility principle, their role is just to update the Controller for every user event and only display data from the Model, without any business logic implementation. In this situation, UI tests should be sufficient to cover the functionalities of the View.

Disadvantages

The view is connected to both controller and model. In order to minimize the logic in the view, the model should be able to provide testable methods for every element that gets to be displayed. Since, View needs both the Model and Controller, changes in the UI logic might require updates in several of classes, therefore making this pattern rigid, and changing the logic difficult. As a result, another architecture pattern will emerge to give better solution for developers.

2.2.2. MVP (Model View Presenter)

MVP architecture pattern improves testability of an application by separating the business model from view making the presenter as a mediator. The presenter is capable of communication with both model and view as shown in Figure 3.

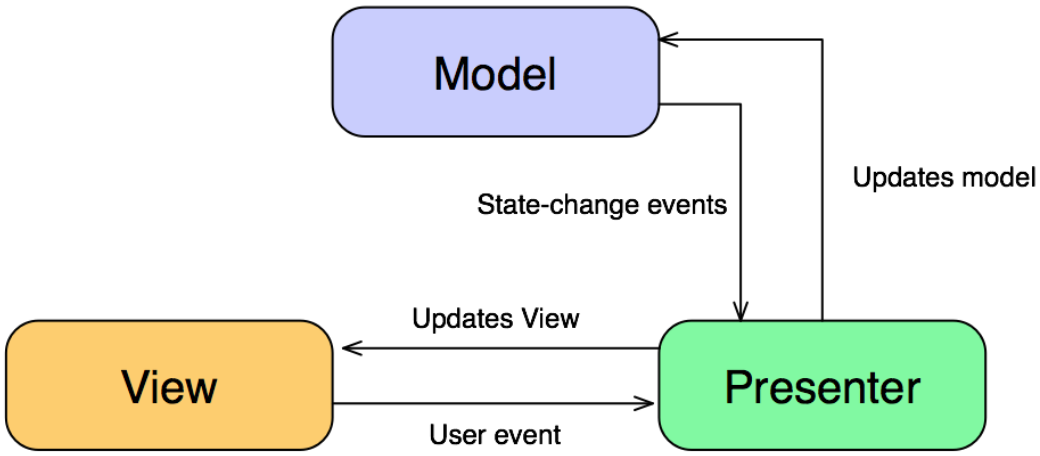

Figure 3: Model View Presenter class structure

Figure 3 illustrates the communication of presenter with model and view. Presenter update the model as well as view. Also, it handles the user events from view and state change events from the model.

In MVP architecture, model holds data provider and so code for updating and modifying the data remain in this part. The Model also updates the database by communicating with a webserver [20].

View component in MVP only deals with a user interface, which is a visual part of any application. Its main task is dealing with user interface but does not contain any logic and knowledge of the data, which is displayed to the user. Generally, in MVP pattern, the view components export an interface that is used by the Presenter and presenter uses these interface methods to manipulate the view [12]. Generally, view is implemented by an Activity. Example method names are: `showProgressBar`, `updateData`.

The Presenter act as a mediation between model and view. It handles the task for updating business logic i.e. model and notifying the view for update. Thus, interacting with the model then fetching and transforming data from the model to update the view. The presenter should not contain a dependency to the Android SDK whenever possible. [12]

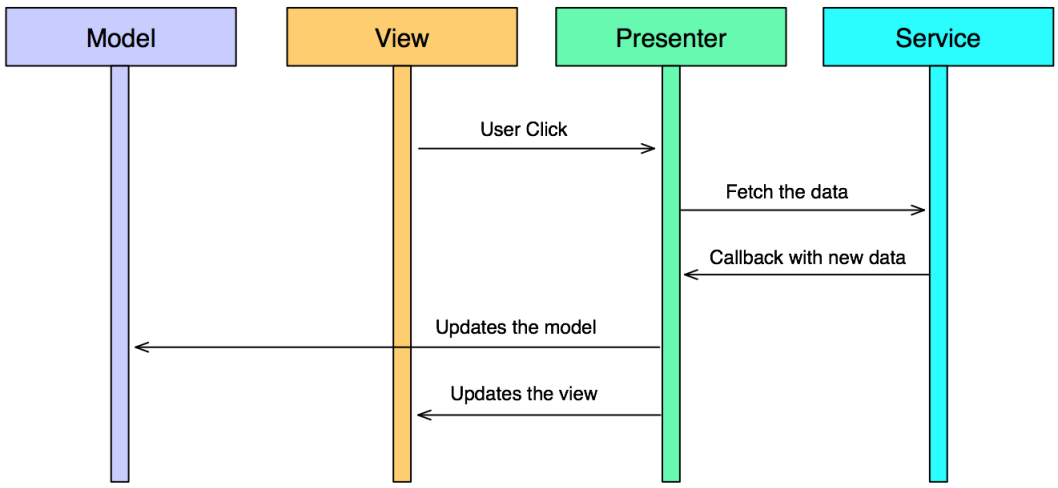

Figure 4: Data flow in MVP

Figure 4 illustrates the data flow in MVP pattern. Firstly, a user makes interaction with view by clicking a button for example. Secondly, Presenter fetches the data from web services and is updated with new data. Thirdly, Presenter updates the model with the data and finally, it also updates the view. [13]

MVP makes testing application presenter logic and to replace dependencies easier. However, using MVP also has its drawbacks, because it makes the application code longer. The standard Android templates do not use this approach now and the code structure in this template is difficult to understand for the developers.

Difference with MVC

In the Model View Presenter pattern, the view is separated from the model. The presenter is a mediator, which communicates between model and view. Presenter fetch the data from model, in addition it also knows if the view must be updated and bind a new data to a view. View does not have any idea about the model and its only purpose is capturing user input and displaying the data fetched by presenter. This makes it easier to create unit tests. Generally, there is a one-to-one connection between view and presenter, but it is also possible to use multiple presenters for complex views [6, 12].

In the Model View Controller pattern, the controllers are behavior based and can share multiple views [8]. View is notified by model in case of any change that occur on model. View can be communicated directly with the model and with controller simultaneously. Handling of the UI logic is not in a single class, but in multiple classes between the controller and the view or model making the responsibility divided between these components.

Model View Presenter pattern resolve these issues by separating the connection that make view dependent on model. MVP creates a single class called Presenter that handles task related to the presentation of the View, a class that is easy to unit test [11]. As a result, Android community prefers this architecture pattern for the development until now.

2.2.3 MVVM (Model View ViewModel)

MVVM was introduced by Microsoft and it is a natural pattern for XAML platforms, however for android platform it is a new approach. This architecture pattern is used to abstract the state and behavior of a view, which enables the development of the user interface from the model separately. By introducing ViewModel as a mediator, whose responsibility is exposing the data of model and handling applications logic while displaying view. Sometimes this architectural pattern is also called Model View Binder as described by Larv Vogel [12].

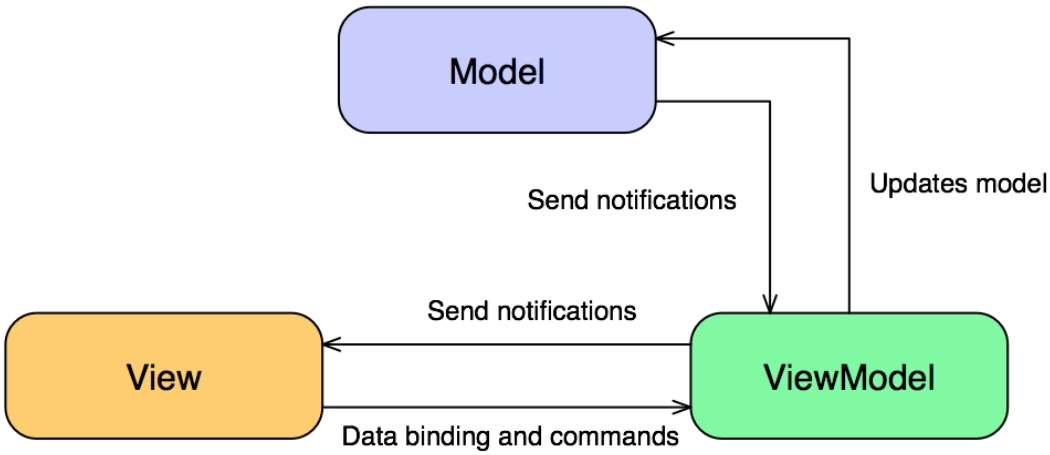

Figure 5: Model View ViewModel Class structure

As depicted in Figure 5, the role of ViewModel to update the model and send notification to its view. In return, it also receives notification from the model. Also, the view is connected to ViewModel by databinding.

In MVVM, Model and View is same as in previous two architectural patterns i.e. MVC and MVP. It is made up of three core components. Model contains the data providers and the code for fetching, updating the data from different sources such as REST API, SQLite db, Shared preference, Firebase, etc.

View is the visual part and it is responsible for starting activities and handling menus, permissions, event listeners, toast, snackbar, dialogs, etc. View binds to the observable variables exposed by viewmodel using data binding framework.

Main difference is using ViewModel as a mediator between model and view. ViewModel have a great number benefits for testing and developing application. Firstly, ViewModel creates one-way communication channel with the View by binding itself to the corresponding view. ViewModel acts as a mediator to pass the events triggered by user in the view component to the model component. It is not tied to the view itself but only acts as model of a view. Secondly, ViewModel wraps the model and prepares observable data needed by the view. View Model receives its data from the Model. The ViewModel handles following responsibilities [12]:

- Expose the data
- Expose the state (progress, offline, empty, error, etc.)
- Handle visibility
- Input validation
- Execute calls to the model
- Execute methods in the view

The view needs to know about the application context which can start a service, bind a service, send or receive a broadcast and load a resource value. But, viewmodel cannot start an activity, inflate a layout and show a dialog.

Differences with MVC

MVVM architecture introduces two-way communication between ViewModel and its components. The controller component is replaced by ViewModel component which receives its data from the model. Data binding library in MVVM helps to connect widgets from layout file to the main activity. So declaring widget id and connecting the widgets with the use of findViewById() can be elimited. This removes numerous lines of boilerplate code which in return reduces the code size. In contrast, MVC is only able to do one-way communication between the components used. And for communication with widgets several lines of boilerplate need to be written. Furthermore, numerous widgets need unique id name hence developers have to name and keep track of all the id of widgets for updating, making the program difficult to debug.

Differences with MVP

In MVP, view knows the presenter with one to one mapping. In contrast, MVVM can map many views to one viewmodel. MVVM is event driven architecture. It combines separation of concerns provided by MVP, while gaining the advantages of data bindings. As a result, a pattern emerges where the model drives as many of the operations as possible, minimizing the logic in the view.

ViewModel in Android

The concept of MVVM has widely been used in WPF, Silverlight and JS environments. But in Android it has just been started, and Google uplift the use of MVVM architecture since the release of databinding library. (See figure 6)

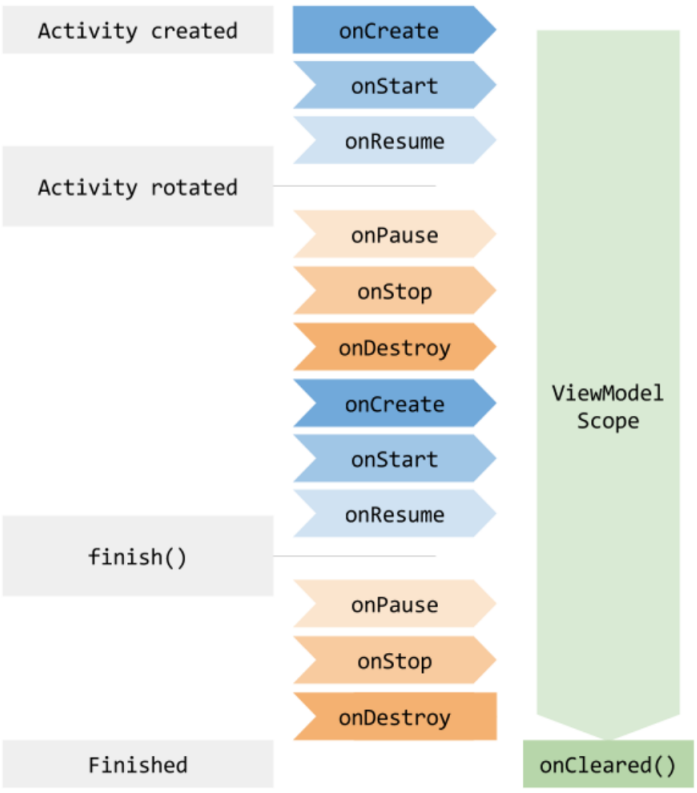

Figure 6: Lifecycle scope of viewmodel [15]

Figure 6 illustrates the lifecycle state of activity when the configuration changes. The android system call onCreate() method several times throughout the life of activity in case when configuration of screen changes.

In Android, viewmodel class stores and manages data related to UI in a lifecycle conscious way. The framework itself manages the lifecycle of UI controllers such as fragment or activity. Use of viewmodel can replace the loaders to load data. Previously, loaders were used to encapsulate the process of data loading and to prevent needless data reloading when configuration of a device changes. These are now accomplished with architecture components and handled in two separate classes LiveData and ViewModel. [15]

LiveData is an observable data holder class which is lifecycle aware and it is different compared to a regular observable. It updates observers only when the

lifecycle state is active. The state is active when the lifecycle is in Started or Resumed state. In an activity, Started state is reached in two cases, after onStart call and before onPause call. And Resumed state is reached in activity when onResume is called as shown in Figure 7.

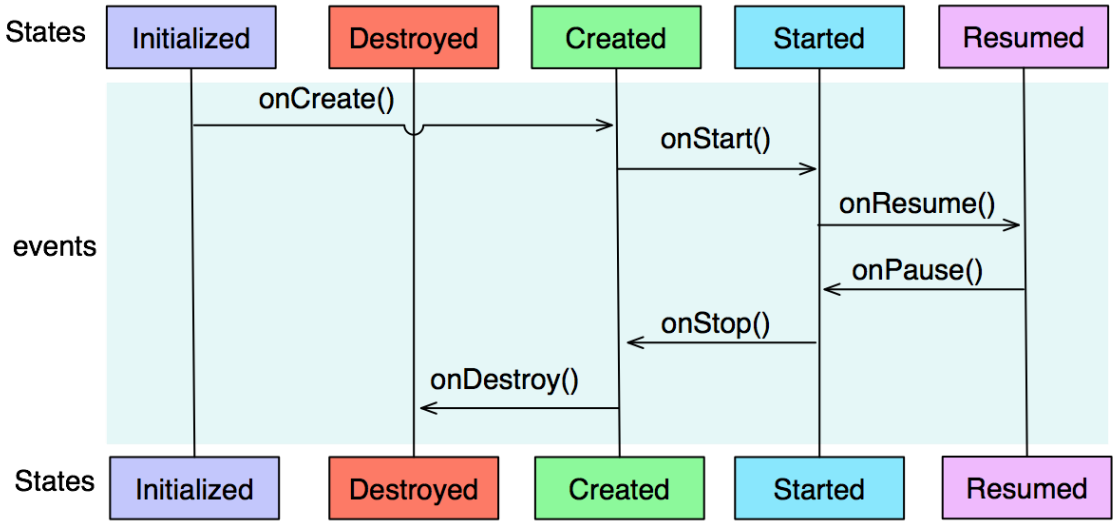

Figure 7: Lifecycle state

Figure 7 illustrates the lifecycle components. Lifecycle uses two enumerations to track it components lifecycle state and they are event and state. The framework and Lifecycle class handles the lifecycle events, and these are managed in callback events in activity or fragment. And the state is traced by lifecycle object.

Additional benefit comes with the use of LiveData. Once the lifecycle is destroyed, the observers associated to lifecycle objects clear the memory, so no memory leakage happens. And since LiveData is aware of the lifecycle, data is not updated when the observer is in inactive state. Hence, no crashes occur because of stopped activities and this may occur when the activity is in backstack.

However, after the state becomes active again the observer obtains the latest data because observer just observe the data without caring about stopped or resumed state of activity. Hence, the data is up-to-date without having to update the UI manually every time the data changes.

3 **Application**

3.1 Introduction

The 8-puzzle illustrated in Figure 8, is a simple puzzle game which contain eight sliding tiles, placed in a 3x3 square block of nine tiles. This game is invented and popularized by Noyes Palmer Chapman in 1870s [16]. The digits inside the cells are numbered from 1 to 8 and one cell is always empty to leave the room for other tiles to move horizontally or vertically. But, moving the tiles diagonally is not allowed. The user moves the tile to arrange it in order that one space remaining at the bottom right of the board. The goal of this game is to start from an initial configuration and change its configuration so that the tiles are placed in ascending order as in the last block. The game acts as a challenge for user's brain.

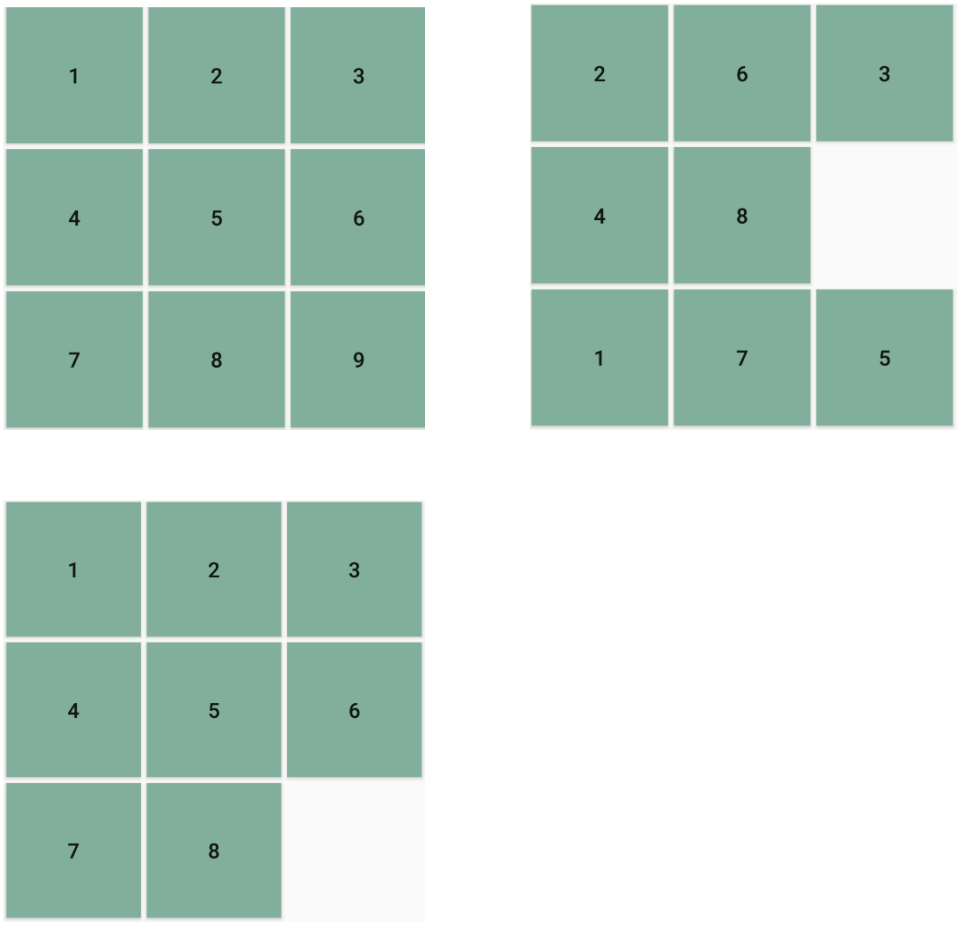

Figure 8: Puzzle block of 3x3 containing 9 squares

The first block is the initial condition of the puzzle. It contains all the tiles making the concept clear about how the tiles should be arranged to win or finish the game. The game starts by clicking the button. On each time starting a new game, user face a tile arranged randomly where one tile becomes empty to make a room for other tiles to move. The second block located on top right position shows game state on progress. And the last block located below is the final condition of the puzzle game when it is finished state.

According to Norvig and Russell, in 8-puzzle sliding game there are 9!/2 possible states where 9! is the total number of configuration but only 9!/2 configurations can be solved [23]. So, after the tile are configured randomly, the user just moves the tiles to arrange the tiles in specific order. Function for handling the tile is carried out in viewmodel class.

3.2 Implementation

The implementation phase of the project involves installation and developing process for creating application. The project was carried out on Android studio 3.0.1. because official IDE for Android app development is Android Studio and it is based on IntelliJ IDEA. However, Eclipse can also be used to do the same. Java programming was used to test the architecture pattern but kotlin is now an official language in Android [22].

3.2.1 Development Environment Setup

Android studio can be downloaded and installed from official page of Android developer [21]. Android studio comprises fastest tools to develop application for android devices. It consists of tools for code editing, debugging, and building and deploying applications. It is flexible to use and easy to handle the code in this IDE and certain things should be included before developing an application.

Android Software Development Kit (SDK)

Android Software Development Kit short for SDK, illustrated in Figure 9, is a collection of software tools used for application development in Android platform. It supports developers with external libraries, APIs, emulator, debugger and other functionalities [21]. Android SDK can also be downloaded manually, or it can be updated automatically. SDK is supplied with IDE (Integrated development environment).

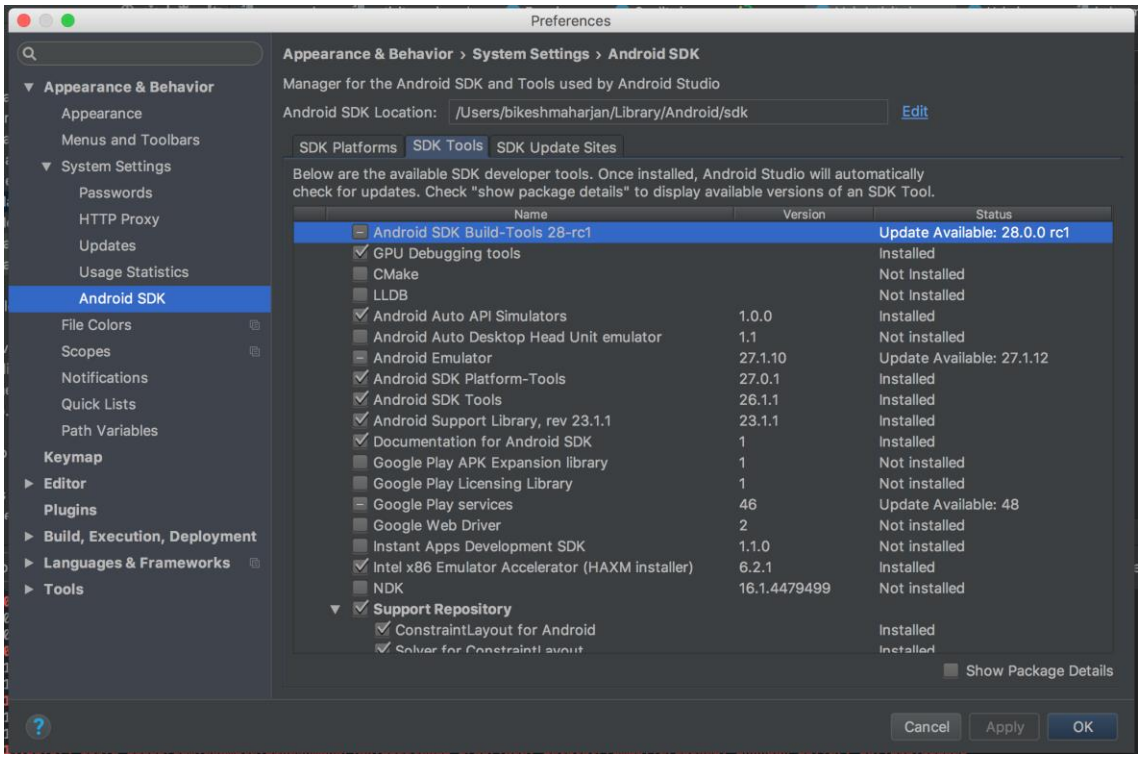

Figure 9: Android SDK snippet

Figure 9 is a snapshot of preference and different versions of API can be downloaded from SDK manager. It also shows the installed library support and API level available updates for emulator.

Java Development Kit(JDK)

JDK is an environment for developing applications, applets and components using Java programming language. JDK contains tools used for developing and testing and debugging programs written for Java platform using Java programming language. The collection of tools in JDK include JRE in addition with tools for developers for developing and testing Java applications. JRE is Java Runtime Environment used only for running Java programs not for development purpose and it contain libraries and Java Virtual Machine (JVM).

3.2.2 Separation of Class structure

In traditional android application development data, logic and presentation are not in a separate class. All codes are in fragment or activity. To give solution for complex UI development in Android applications, several lines of boilerplate code should be written. By taking an example, when user enters and receives data, parameters in some View might change. This method of programming is not a convenient way because as the time and complexity increases, the logic of the program gets mixed up with the presentation which makes more bugs and makes debugging difficult. The solution for this problem is to implement MVVM pattern using Data Binding which gives an option for separate data, logic and presentation.

The concept can be made clear by developing an 8-puzzle game. As explained in Figure 10, separate classes for Model, View and ViewModel was created to implement the concept.

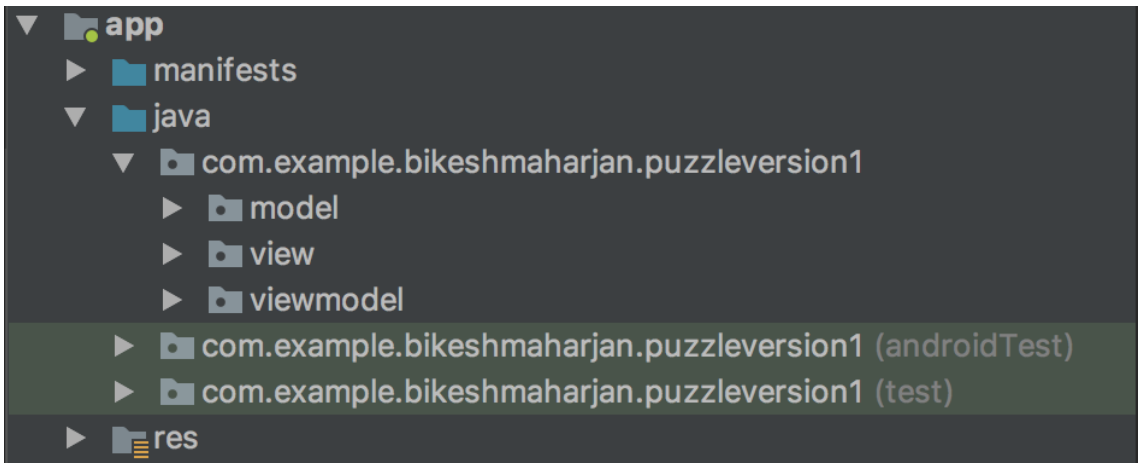

Figure 10: Separate classes in package

As depicted in figure 10, separate package was created for handling the separation of class. The model contains all the data and logic of the puzzle application. View is to be handled in layout xml file and through the main activity. ViewModel class handles and store UI-related data in lifecycle conscious way. Additionally, this class allows data to survive configuration changes such as screen rotations.

It is necessary to import ViewModel into the project for its use [19]. Google Maven repository should be added because by default, Android studio projects are not pre-configured to access this repository.

```
allprojects {
    repositories {
        jcenter()
        google()
    }
}
```

The highlighted code google() should be added in build.gradle file of the project. This gradle file in not the ones for the app or the module. Additionally, as the project demands, adding the dependencies in build.gradle file for the app or module in needed.

```
dependencies {
    // ViewModel and LiveData
    implementation "android.arch.lifecycle:extensions:1.1.1"
    // alternatively, just ViewModel
    implementation "android.arch.lifecycle:viewmodel:1.1.1"
    // alternatively, just LiveData
    implementation "android.arch.lifecycle:livedata:1.1.1"

    annotationProcessor "android.arch.lifecycle:compiler:1.1.1"

    // Room (use 1.1.0-beta2 for latest beta)
    implementation "android.arch.persistence.room:runtime:1.0.0"
    annotationProcessor "android.arch.persistence.room:compiler:1.0.0"

    // Paging
    implementation "android.arch.paging:runtime:1.0.0-beta1"

    // Test helpers for LiveData
    testImplementation "android.arch.core:core-testing:1.1.1"

    // Test helpers for Room
    testImplementation "android.arch.persistence.room:testing:1.0.0"
}
```

Figure 11: Dependencies for ViewModel, LiveData, Room [19]

The dependencies are added in build.gradle file of the project as per need. All the dependencies are not needed but added based on the use of the components as shown above in figure 11.

3.2.3 Data Binding

Data Binding creates a link between data model and the UI layer, where the data model holds information for display [17]. In previous model of Android application, finding view and updating the content was necessary. At the time of change in data, the UI widget such as TextView and ImageView bound to UI need to be updated. Writing code and updating the view takes a great deal of time and occupy large spaces in Activity.

After using data binding library, the code of application logic with the UI view reduces significantly. With its use, the methods call such as ‘findViewById’ and ‘setText’ can be eliminated.

Moreover, the real strength of data binding is when updating a value in an application code occurring at multiple points. This reduces developers time in development phase and makes updating the values easier.

Updating Gradle File

The first step in this project is adding the data binding by changing the module's build.gradle file. Recently, the Android data binding library has made data binding easier by adding data binding closure to android closure and it is already included in Google's Application and Library plugins, hence there is no need now adding a dependency. The feature of data binding in android is data connection and it can be used by adding tiny piece of its closure as shown in below.

```
1.  android {
2.      dataBinding {
3.          enabled true
4.      }
5.  }
```

Compatible version of Android Studio 1.3 or later is necessary to support data binding. And Android plugin for Gradle 2.1(API 7) and newer have data binding library available. For layouts, Android data binding generates binding classes at compile time [18].

Preparing layout file

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">

    <data>
        <import type="android.view.View" />
        <variable
            name="gameViewModel"
            type="com.example.bikeshmaharjan.puzzleversion1.viewmodel.ViewModel"/>

        <variable
            name="handler"
            type="com.example.bikeshmaharjan.puzzleversion1.viewmodel.ViewModel"/>
    </data>

    <RelativeLayout xmlns:tools="http://schemas.android.com/tools"...>
</layout>
```

There is a slight variation using default layout and using binding layout in xml file. All layout files using data binding technique must contain a layout<> root tag. And to bind objects it is important to add data<> tag within the layout tag, before UI view root and data<> element can contain multiple variable<> tag within the layout which describes a property to be used. The variable<> tag contains name and type to be used in the layout. The tag <import> allows easy reference to classes inside the layout file same as in java.

View can be used in the binding expression. In the layout, attributes properties are written using “@{}” syntax.

```
<TextView
    android:id="@+id/winmessage"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/button_newGame"
    android:layout_centerHorizontal="true"
    android:paddingBottom="10dp"
    android:text="@={gameViewModel.winmessage}"
    tools:text="@{gameViewModel.winmessage}"
    android:textAppearance="?android:attr/textAppearanceMedium"
/>
```

```
<Button
    android:id="@+id/button1"
    android:layout_width="90dp"
    android:layout_height="90dp"
    android:layout_above="@+id/button5"
    android:layout_margin="2dp"
    android:layout_toLeftOf="@+id/button5"
    android:background="@color/buttonColor"
    android:clickable='@{gameViewModel.win=="true" ? false : true}'
    android:visibility='@{gameViewModel.tiles["0"]=="0" ? View.INVISIBLE : View.VISIBLE}'
    android:onClick="@{(v)->handler.tileClicked(v)}"
    android:text='@{gameViewModel.tiles["0"]}'
     />
```

The button tiles are arranged with a relative layout hence button id was used for adjusting the layout. Binding expression was used for handling the visibility of an empty tile and for button clickable function as well as text property. The android:onClick property handles the function for moving the tile in empty location based on the valid condition. And if the tiles can be moved the visibility of the tile is managed through the binding expression.

Binding on Main Activity

```
public class MainActivity extends AppCompatActivity {

    ActivityMainBinding activityMainBinding;
    ViewModel viewModel ;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        viewModel=new ViewModel();
        activityMainBinding = DataBindingUtil.setContentView( activity: this, R.layout.activity_main);
        activityMainBinding.setGameViewModel(viewModel);
        activityMainBinding.setHandler(viewModel);
    }
}
```

Binding class is generated based on the layout file name by default. The layout file used in the project was activity_main.xml so generated class was ActivityMainBinding. This class holds all the bindings from the layout properties to the layout's views and know how to assign values for the binding expressions. For creating bindings, the easiest way is to do it while inflating.

Event Handling

Data binding allows to write expression handling events that are dispatched from the views. For example, View.OnClickListener has method onClick(), hence the attribute for this event is android:onClick. Likewise, it also has method onLongClick(), and attribute for this event is android:onLongClick. Mainly, there are two ways to handle events in binding i.e. Method References and Listener Bindings [14].

Method References

Implementation of events in method reference is created when the data is bound. Events are bound to the handler method in a similar way as android:onClick. These events are processed in compile time, hence compile time error occurs either in the case that method does not exist, or the signature of the method is incorrectly declared. In addition, the parameters of the event listener should also match for the method to work properly. Example of method reference:

```
1. android:onClick = “@{handlers::onClickFriend}”
```

in layout file

```
1.  public void onClickFriend(View view) {
2.  ...
3.  }
```

in java class

Listener Bindings

In contrast to method reference, events in listener binding is created when the event is triggered. And the parameters of the method must match the parameter of the event listener unless the method type is void. Example of listener binding:

```
1.  android: onClick = “@ {(v) - > gameViewModel.tileClicked(v)}”
```

in layout file

```
1.  public void tileClicked(View view) {...
2.  }
```

in java class

Few number of operators such as this, super and new cannot be used in the expression syntax, which can be used in java. However, the benefit of data binding is that whenever the data changes it is notified. Three different notification mechanisms are present when the data changes and they are observable objects, observable fields and observable collection. This project uses observable field and observable collection for tiles and message to show to the user.

```
1.  public ObservableArrayMap < String, String > tiles = new ObservableArrayMap <> ();
2.  public ObservableField < String > winmessage = new ObservableField <> ();
```

On completion, the program can be tested either on a real android device or on emulator. The screen shot is taken from the emulator. The screen on left is taken when the game is being played. The message to move the tile is shown until the game is finished. When the game is finished by solving the puzzle, diffrerent message is shown to the user which can be seen in the right corner of the Figure 11. The user can either choose to play again by clicking the new game. All the tiles including the empty tile is positioned randomly again. The code can be obtained form github repository link: https://github.com/bksha8/8-puzzle

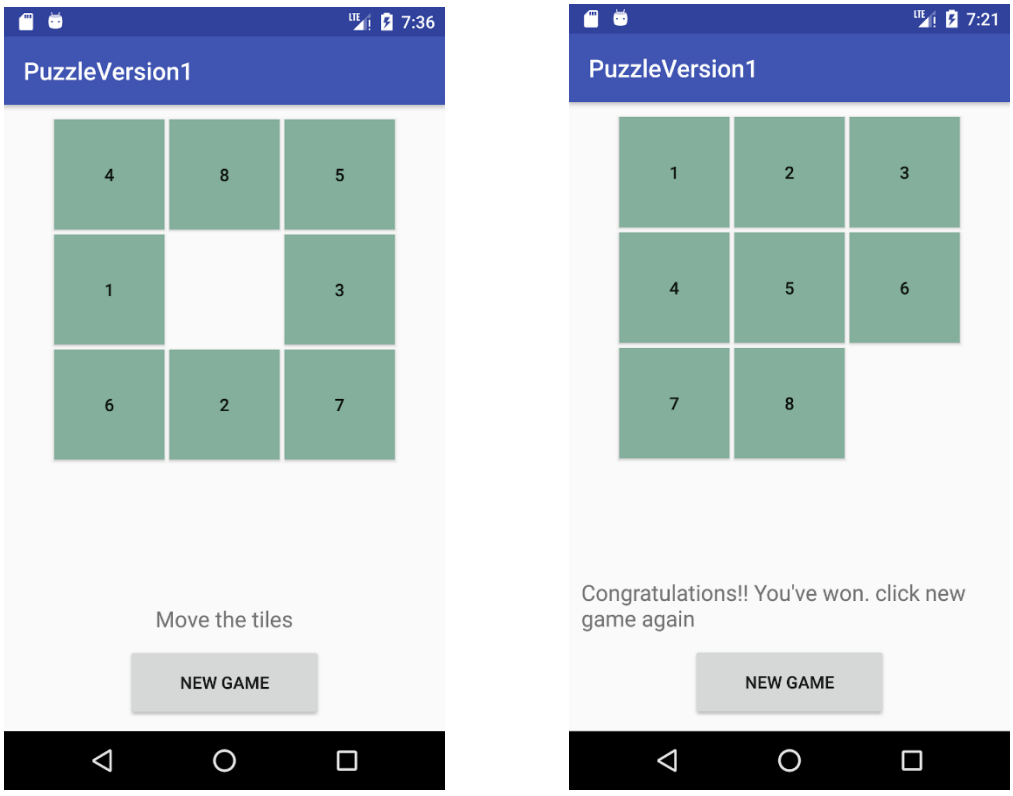

Figure 11: Screen shot of the puzzle

4 Result and Discussion

The purpose of the thesis was to implement the concept of MVVM architecture. To implement this concept a simple puzzle game was developed using Java programming language. The game consists of 8 tiles and one empty tile for the user to move the tile. This game was written in Android studio version 3.0.1 and it can be tested in a physical device or on emulator. This game gives user's brain a challenge for the solution.

Firstly, MVC architecture pattern was used to develop the game. In MVC architecture pattern, the model class handles all the logic and data of the game, and the controller is the main activity which handles the user's commands. Likewise, the view in MVC is the layout file and it is handled in the main activity. All the widgets are connected with activity by using findViewById() command. While findViewById() can be easily implemented for certain number of widgets, but on implementing numerous widgets such as Button, TextView, ImageView, the code to connect it with the activity becomes immense. Also, making track of the id and setting the command becomes very difficult.

Finally, MVVM architecture was used to develop the same game. In this architecture pattern, the model is the same as in MVC. However, the viewmodel class is the replacement for controller. ViewModel handles the command from the user. It uses data binding library to bind view, in other words the layout file to the viewmodel class. Comparing the implementation of MVC and MVVM, it was found that in MVVM architecture, the code becomes a lot easier to maintain. As a result, the code size reduced dramatically. The difference in the code might be miniscule; for example, in this game, only a limited number of widgets are used. When there are numerous widgets, the difference in the code size can be observed clearly.

A simple 8-puzzle game was used to test the architecture model. Furthermore, the game can be made difficult by increasing the number of tiles to 15 of 4x4 board and 24 tiles of 5x5 board. Additional functionality such as counting number

of moves, time consumed, and undo function can be added later. Finally, UI design can be designed more attractive to appeal the user to play the game.

Utilizing data binding library, the project accomplishes separation of view and model with the help of viewmodel class as mediator. Furthermore, because ViewModel is not attached to any other layers, unit testing is easily integrated. Although, MVVM pattern comes with many advantages, Drawbacks are also present in this approach. In MVVM, ViewModel state needs to be saved. Also, separating logic from presentation is not always possible.

It is difficult to determine which approach is the correct way to develop the application, since MVVM is in its initial stage and sufficient number of sources are not available. It seems that the use of data binding library MVVM can be applied in the future development. Business application can be an area where MVVM architecture pattern would be suitable. It is better to use MVVM pattern in case of possible data binding with data context because it is easier to maintain the code. However, if the data context is not bindable, then MVP might be an alternative for MVVM architecture pattern.

5 Conclusion

The purpose of this thesis was to implement the MVVM architecture pattern, know the concept and obtain the ideas related to this pattern. All these goals were achieved. A simple puzzle game was developed and MVVM pattern was implemented. This project utilizes separation of concern for model, view and viewmodel. With separate classes created in the project, it became easier to unit test the code and easier to redesign the user interface. In addition, the size of code reduced in comparison with MVC pattern. However, it is difficult to debug the code because of additional number of files which might also effect the performance.

The success of any software project is determined mainly by using proper software architecture. Despite the time consumed in the initial design phase of an architecture, it is good practice to choose a proper architecture in the beginning. Refactoring and modifying the code with bad architecture becomes problematic in the later phases of development.

The Android framework for Data Binding is still in beta phase, internal support from Android Studio is still partial, and there is room for improvement. However, it is very well designed and developed, and will change the way Android applications are written. The possibility to define custom attributes is quite powerful. For complicated logic and UI screen it is a convenient method. MVVM benefit comes with Data binding library, such as Observable, so need to use findViewById or applying libraries such as Butterknife can be omitted. Binding used in the code dramatically reduces code size and makes the code more compact.

References

1. Statista GmbH. Number of available applications in the Google Play Store from December 2009 to December 2017. Available from: https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ [Accessed 19th March 2018].

2. Statista GmbH. Mobile App Usage-Statistics and Facts. Available from: https://www.statista.com/topics/1002/mobile-app-usage/ [Accessed 19th March 2018].

3. Intelliware software development. Next-Generation Mobile Apps - 7 Critical Success Factors. Available from: http://i-proving.com/wp-content/uploads/2010/05/White-Paper-Next-Generation-Mobile+Apps-May2010.pdf [Accessed April 2018].

4. Sampaio J. Android MVC: Creating a Model-View-Controller Framework for Android [online]. Available from : http://mrbool.com/android-mvc-creating-a-model-view-controller-framework-for-android/34463 [Accessed 19th March 2018].

5. Krasner G. and Pope S. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk-80 System. Available from : http://www.global-webnet.com/Adventures/Files/DescriptionOfMvcUi-KrasnerPope.pdf [Accessed 19th March 2018].

6. Zukanov V. MVP and MVC Architectures in Android. (2015, July 12) [online]. Available from: https://www.techyourchance.com/mvp-mvc-android-1/ [Accessed 18th March 2018].

7. Architech Solutions. The Importance of Software Architecture. Available from: http://static.architech.ca/wp-content/uploads/2010/06/The-Importance-of-Software-Architecture.pdf [Accessed 18th March 2018].

8. Reenskaug T. (1979). The original MVC reports. Dept. of Informatics University of Oslo [online]. Available from: http://heim.ifi.uio.no/~trygver/2007/MVC_Originals.pdf [Accessed 19th March 2018].

9. Bass L., Clements P., Kazman R. (2005). *Software Architecture in Practice*, 2nd edition. Boston, MA: Addison Wesley.

10. Hanmer R. (2012). *Pattern-Oriented Software Architecture for Dummies*. John Wiley & Sons, Incorporated

11. Muntenescu F. (2016, October 08). Android Architecture Pattern [online]. Available from: https://upday.github.io/blog/model-view-presenter/ [Accessed 26 March 2018]

12. Vogel L. (2017, April 18). Android Architecture with MVP or MVVM, vogella GmbH - Version 0.3 [online]. Available from:http://www.vogella.com/tutorials/AndroidArchitecture/article.html [Accessed April 2018]

13. Dorfmann H. (2015, Mar 25). Ted Mosby – Software Architect [online]. Available from: http://hannesdorfmann.com/android/mosby [Accessed April 2018]

14. Data binding Library (2018) [online]. Available from: https://developer.android.com/topic/libraries/data-binding/index.html#studio_support

15. Architecture component (2018) [online]. Available from: https://developer.android.com/topic/libraries/architecture/viewmodel.html

16. 8-puzzle, COS 226 Programming Assignment [online]. Available from:https://www.cs.princeton.edu/courses/archive/spr10/cos226/assignments/8puzzle.html [Accessed 12 March 2018]

17. Ivanov V. (2015, December 1). Going with MVVM on Android via Data Binding [online]. Available from: https://www.azoft.com/blog/mvvm-android-data-binding/ [cited 14 April 2018]

18. Sinhal A. (2017, January 19). Faster Android Development with Data Binding [online] Available from: https://android.jlelse.eu/faster-android-development-with-data-binding-eeef7cc0c4b [cited April, 2018]

19. Adding component to project (2018) [online]. Available from: https://developer.android.com/topic/libraries/architecture/adding-components.html

20. Megali T. Model View Presenter in Android [online]. Available from: http://www.tinmegali.com/en/2016/03/04/model-view-presenter-android-part-1/ [Accessed April 2018]

21. Android Studio. Available from: https://developer.android.com/studio/index.html

22. Kotlin and Android. [online]. Available from: https://developer.android.com/kotlin/index.html

23. Russell S. and Norvig P. (2003). *Artificial Intelligence A Modern Approach*, 2nd edition, Pearson Education, Inc. Available from: http://www.eng.uerj.br/~fariasol/disciplinas/Topicos_B/AGENTS/books/Stuart%20Russell,%20Peter%20Norvig-Artificial%20Intelligence_%20A%20Modern%20Approach-Prentice%20Hall%20(2002)-2nd-ed.pdf