Labsheet 10 – Friday 26th November, 2021

Android Architecture Patterns

Software Development for Portable Devices

Information about HD TA:

AASHITA DUTTA <h20201030130@hyderabad.bits-pilani.ac.in>

Breakout room link: https://meet.google.com/erq-uxqi-egv

Information about FD TA:

ARUNEEMA DESHMUKH <f20180568@hyderabad.bits-pilani.ac.in>

Breakout room link: https://meet.google.com/pvo-osiz-cuh

Introduction

The best practices approach for organizing Android applications into logical components has evolved over the last few years. The community has largely moved away from the monolithic Model View Controller (MVC) pattern in favor of more modular, testable patterns.

Model View Presenter (MVP) & Model View ViewModel (MVVM) are two of the most widely adopted alternatives, but developers are often divided as to which one better fits with Android. There have been numerous blog posts over the past year or so strongly advocating for one over the other, but often these turn into arguments of opinion over objective criteria. We will look at the value and potential issues with all three approaches so you can make an informed decision for yourself.

To help us see each pattern in action, we use a simple Tic-Tac-Toe game. Some of the files will be common in the three examples with some minor changes required.

Start by creating a new project with empty activity.

Create a package called **model** in **app/src/main/java/com.example.appname (com.example.tictactoe.model).** Add these files

Player.java

```
package com.example.tictactoe.model;

public enum Player { X , O }
```

Cell.java

```
package com.example.tictactoe.model;

public class Cell {

  private Player value;
```

```
  public Player getValue() {
    return value;
  }

  public void setValue(Player value) {
    this.value = value;
  }
}
```

Board.java

```
package com.example.tictactoe.model;

import static com.example.tictactoe.model.Player.O;
import static com.example.tictactoe.model.Player.X;

public class Board {

  private Cell[][] cells = new Cell[3][3];

  private Player winner;
  private GameState state;
  private Player currentTurn;

  private enum GameState { IN_PROGRESS, FINISHED };

  public Board() {
    restart();
  }

  /**
   *  Restart or start a new game, will clear the board and win status
   */
  public void restart() {
    clearCells();
    winner = null;
    currentTurn = Player.X;
    state = GameState.IN_PROGRESS;
  }

  /**
   * Mark the current row for the player who's current turn it is.
   * Will perform no-op if the arguments are out of range or if that position is already played.
   * Will also perform a no-op if the game is already over.
   *
   * @param row 0..2
   * @param col 0..2
```

```
 * @return the player that moved or null if we did not move anything.
 *
 */
public Player mark( int row, int col ) {

   Player playerThatMoved = null;

   if(isValid(row, col)) {

      cells[row][col].setValue(currentTurn);
      playerThatMoved = currentTurn;

      if(isWinningMoveByPlayer(currentTurn, row, col)) {
         state = GameState.FINISHED;
         winner = currentTurn;

      } else {
         // flip the current turn and continue
         flipCurrentTurn();
      }
   }

   return playerThatMoved;
}

public Player getWinner() {
   return winner;
}

private void clearCells() {
   for(int i = 0; i < 3; i++) {
      for(int j = 0; j < 3; j++) {
         cells[i][j] = new Cell();
      }
   }
}

private boolean isValid(int row, int col ) {
   if( state == GameState.FINISHED ) {
      return false;
   } else if( isOutOfBounds(row) || isOutOfBounds(col) ) {
      return false;
   } else if( isCellValueAlreadySet(row, col) ) {
      return false;
   } else {
      return true;
   }
```

```
  }

  private boolean isOutOfBounds(int idx) {
    return idx < 0 || idx > 2;
  }

  private boolean isCellValueAlreadySet(int row, int col) {
    return cells[row][col].getValue() != null;
  }


  /**
   * Algorithm adapted from
http://www.ntu.edu.sg/home/ehchua/programming/java/JavaGame_TicTacToe.html
   * @param player
   * @param currentRow
   * @param currentCol
   * @return true if <code>player</code> who just played the move at the <code>currentRow</code>,
<code>currentCol</code>
   *           has a tic tac toe.
   */
  private boolean isWinningMoveByPlayer(Player player, int currentRow, int currentCol) {

    return (cells[currentRow][0].getValue() == player        // 3-in-the-row
          && cells[currentRow][1].getValue() == player
          && cells[currentRow][2].getValue() == player
          || cells[0][currentCol].getValue() == player     // 3-in-the-column
          && cells[1][currentCol].getValue() == player
          && cells[2][currentCol].getValue() == player
          || currentRow == currentCol           // 3-in-the-diagonal
          && cells[0][0].getValue() == player
          && cells[1][1].getValue() == player
          && cells[2][2].getValue() == player
          || currentRow + currentCol == 2    // 3-in-the-opposite-diagonal
          && cells[0][2].getValue() == player
          && cells[1][1].getValue() == player
          && cells[2][0].getValue() == player);
  }

  private void flipCurrentTurn() {
    currentTurn = currentTurn == X ? O : X;
  }

}
```

Layout files

[You will have to change the tools:context as required]

tictactoe.xml in res/layout folder

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  android:id="@+id/tictactoe"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:gravity="center_horizontal"
  tools:context="com.example.tictactoe.controller.TicTacToeActivity">

  <androidx.appcompat.widget.Toolbar
  android:id="@+id/my_toolbar"
  android:layout_width="match_parent"
  android:layout_height="?attr/actionBarSize"
  android:background="?attr/colorPrimary"
  android:elevation="4dp"
  android:theme="@style/ThemeOverlay.MaterialComponents.Dark.ActionBar"
  app:popupTheme="@style/ThemeOverlay.MaterialComponents.Light"/>

  <GridLayout
    android:id="@+id/buttonGrid"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:layout_gravity="center"
    android:columnCount="3"
    android:rowCount="3">

    <Button
      android:tag="00"
      android:onClick="onCellClicked"
      style="@style/tictactoebutton"
      />

    <Button
      android:tag="01"
      android:onClick="onCellClicked"
      style="@style/tictactoebutton"
      />
```

```
    <Button
        android:tag="02"
        android:onClick="onCellClicked"
        style="@style/tictactoebutton"
        />

    <Button
        android:tag="10"
        android:onClick="onCellClicked"
        style="@style/tictactoebutton"
        />

    <Button
        android:tag="11"
        android:onClick="onCellClicked"
        style="@style/tictactoebutton"
        />

    <Button
        android:tag="12"
        android:onClick="onCellClicked"
        style="@style/tictactoebutton"
        />

    <Button
        android:tag="20"
        android:onClick="onCellClicked"
        style="@style/tictactoebutton"
        />

    <Button
        android:tag="21"
        android:onClick="onCellClicked"
        style="@style/tictactoebutton"
        />

    <Button
        android:tag="22"
        android:onClick="onCellClicked"
        style="@style/tictactoebutton"
        />

</GridLayout>

<LinearLayout
    android:id="@+id/winnerPlayerViewGroup"
```

```
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    android:visibility="gone"
    tools:visibility="visible"
    >

    <TextView
        android:id="@+id/winnerPlayerLabel"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="40sp"
        android:layout_margin="20dp"
        tools:text="X" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textSize="30sp"
        android:text="@string/winner" />

  </LinearLayout>

</LinearLayout>
```

menu_tictactoe.xml in res/menu folder

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:app="http://schemas.android.com/apk/res-auto"
  xmlns:tools="http://schemas.android.com/tools"
  tools:context=".controller.TicTacToeActivity">

  <item android:id="@+id/action_reset"
    android:title="@string/action_reset"
    android:orderInCategory="100"
    app:showAsAction="ifRoom" />

</menu>
```

dimen.xml in res/values

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <!-- Default screen margins, per the Android Design guidelines. -->
  <dimen name="activity_horizontal_margin">16dp</dimen>
```

```
  <dimen name="activity_vertical_margin">44dp</dimen>
</resources>
```

strings.xml in res/values

```
<resources>
  <string name="app_name">TicTacToe</string>
  <string name="winner">Winner</string>
  <string name="action_reset">Reset</string>
</resources>
```

themes.xml add new style

```
<resources xmlns:tools="http://schemas.android.com/tools">
  <!-- Base application theme. -->
  <style name="Theme.tictactoe" parent="Theme.AppCompat.DayNight.NoActionBar">
    <!-- Primary brand color. -->
    <item name="colorPrimary">@color/purple_500</item>
    <item name="colorPrimaryVariant">@color/purple_700</item>
    <item name="colorOnPrimary">@color/white</item>
    <!-- Secondary brand color. -->
    <item name="colorSecondary">@color/teal_200</item>
    <item name="colorSecondaryVariant">@color/teal_700</item>
    <item name="colorOnSecondary">@color/black</item>
    <!-- Status bar color. -->
    <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimaryVariant</item>
    <!-- Customize your theme here. -->
  </style>

  <style name="tictactoebutton">
    <item name="android:layout_width">100dp</item>
    <item name="android:layout_height">100dp</item>
    <item name="android:textSize">30sp</item>
  </style>
</resources>
```

MVC

The model, view, controller approach separates your application at a macro level into 3 sets of responsibilities.

Model

The model is the **Data + State + Business** logic of our Tic-Tac-Toe application. It's the brains of our application so to speak. It is not tied to the view or controller, and because of this, it is reusable in many contexts.

View

The view is the Representation of the Model. The view has a responsibility to render the User Interface (UI) and communicate to the controller when the user interacts with the application. In MVC architecture, Views are generally pretty "dumb" in that they have no knowledge of the underlying model and no understanding of state or what to do when a user interacts by clicking a button, typing a value, etc. The idea is that the less they know the more loosely coupled they are to the model and therefore the more flexible they are to change.

Controller

The controller is Glue that ties the app together. It's the **master controller** for what happens in the application. When the View tells the controller that a user clicked a button, the controller decides how to interact with the model accordingly. Based on data changing in the model, the controller may decide to update the state of the view as appropriate. In the case of an Android application, the controller is almost always represented by an Activity or Fragment.

Similar to the model folder create a **controller** folder and add this Activity. Set this as the Launcher Activity in AndroidManifest.xml

TicTacToeActivity.java

```
package com.example.tictactoe.controller;

import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;


import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

import com.example.tictactoe.R;
import com.example.tictactoe.model.Board;
import com.example.tictactoe.model.Player;

public class TicTacToeActivity extends AppCompatActivity {

  private static String TAG = TicTacToeActivity.class.getName();
```

```
private Board model;

private ViewGroup buttonGrid;
private View winnerPlayerViewGroup;
private TextView winnerPlayerLabel;

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

   winnerPlayerLabel = (TextView) findViewById(R.id.winnerPlayerLabel);
   winnerPlayerViewGroup = findViewById(R.id.winnerPlayerViewGroup);
   buttonGrid = (ViewGroup) findViewById(R.id.buttonGrid);
   Toolbar myToolbar = (Toolbar) findViewById(R.id.my_toolbar);
   setSupportActionBar(myToolbar);

   model = new Board();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.menu_tictactoe, menu);
   return true;
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.action_reset:
         reset();
         return true;
      default:
         return super.onOptionsItemSelected(item);
   }
}

public void onCellClicked(View v) {

   Button button = (Button) v;

   String tag = button.getTag().toString();
   int row = Integer.valueOf(tag.substring(0,1));
   int col = Integer.valueOf(tag.substring(1,2));
   Log.i(TAG, "Click Row: [" + row + "," + col + "]");
```

```
    Player playerThatMoved = model.mark(row, col);

    if(playerThatMoved != null) {
      button.setText(playerThatMoved.toString());
      if (model.getWinner() != null) {
        winnerPlayerLabel.setText(playerThatMoved.toString());
        winnerPlayerViewGroup.setVisibility(View.VISIBLE);
      }
    }

  }

  private void reset() {
    winnerPlayerViewGroup.setVisibility(View.GONE);
    winnerPlayerLabel.setText("");

    model.restart();

    for( int i = 0; i < buttonGrid.getChildCount(); i++ ) {
      ((Button) buttonGrid.getChildAt(i)).setText("");
    }
  }

}
```

Here's what that looks like at a high level in our Tic Tac Toe app and the classes that play each part.

Evaluation

MVC does a great job of separating the model and view. Certainly the model can be easily tested because it's not tied to anything and the view has nothing much to test at a unit testing level. The Controller has a few problems however.

Controller Concerns

Testability - The controller is tied so tightly to the Android APIs that it is difficult to unit test.

Modularity & Flexibility - The controllers are tightly coupled to the views. It might as well be an extension of the view. If we change the view, we have to go back and change the controller.

Maintenance - Over time, particularly in applications with anemic models, more and more code starts getting transferred into the controllers, making them bloated and brittle.

MVP

MVP breaks the controller up so that the natural view/activity coupling can occur without tying it to the rest of the "controller" responsibilities. More on this below, but let's start again with a common definition of responsibilities as compared to MVC.

Model

Same as MVC / No change

View

The only change here is that the Activity/Fragment is now considered part of the view. We stop fighting the natural tendency for them to go hand in hand. Good practice is to have the Activity implement a view interface so that the presenter has an interface to code to. This eliminates coupling it to any specific view and allows simple unit testing with a mock implementation of the view.

So you can rename the controller folder/package as **view.**

TicTacToeActivity.java

```
package com.example.tictactoe.view;

import android.os.Bundle;

import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;

import com.example.tictactoe.R;
import com.example.tictactoe.presenter.TicTacToePresenter;

import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;

public class TicTacToeActivity extends AppCompatActivity implements TicTacToeView {

  private static String TAG = TicTacToeActivity.class.getName();

  private ViewGroup buttonGrid;
```

```
private View winnerPlayerViewGroup;
private TextView winnerPlayerLabel;

TicTacToePresenter presenter = new TicTacToePresenter(this);

@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.tictactoe);
   winnerPlayerLabel = (TextView) findViewById(R.id.winnerPlayerLabel);
   winnerPlayerViewGroup = findViewById(R.id.winnerPlayerViewGroup);
   buttonGrid = (ViewGroup) findViewById(R.id.buttonGrid);
   Toolbar myToolbar = (Toolbar) findViewById(R.id.my_toolbar);
   setSupportActionBar(myToolbar);
   presenter.onCreate();
}

@Override
protected void onPause() {
   super.onPause();
   presenter.onPause();
}

@Override
protected void onResume() {
   super.onResume();
   presenter.onResume();
}

@Override
protected void onDestroy() {
   super.onDestroy();
   presenter.onDestroy();
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.menu_tictactoe, menu);
   return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.action_reset:
         presenter.onResetSelected();
```

```
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
  }

  public void onCellClicked(View v) {

    Button button = (Button) v;
    String tag = button.getTag().toString();
    int row = Integer.valueOf(tag.substring(0,1));
    int col = Integer.valueOf(tag.substring(1,2));
    Log.i(TAG, "Click Row: [" + row + "," + col + "]");

    presenter.onButtonSelected(row, col);

  }

  @Override
  public void setButtonText(int row, int col, String text) {
    Button btn = (Button) buttonGrid.findViewWithTag("" + row + col);
    if(btn != null) {
      btn.setText(text);
    }
  }

  public void clearButtons() {
    for( int i = 0; i < buttonGrid.getChildCount(); i++ ) {
      ((Button) buttonGrid.getChildAt(i)).setText("");
    }
  }

  public void showWinner(String winningPlayerDisplayLabel) {
    winnerPlayerLabel.setText(winningPlayerDisplayLabel);
    winnerPlayerViewGroup.setVisibility(View.VISIBLE);
  }

  public void clearWinnerDisplay() {
    winnerPlayerViewGroup.setVisibility(View.GONE);
    winnerPlayerLabel.setText("");
  }
}
```

TicTacToeView.java

```
package com.example.tictactoe.view;
```

```
public interface TicTacToeView {
  void showWinner(String winningPlayerDisplayLabel);
  void clearWinnerDisplay();
  void clearButtons();
  void setButtonText(int row, int col, String text);
}
```

Presenter

This is essentially the controller from MVC except that it is not at all tied to the View, just an interface. This addresses the testability concerns as well as the modularity/flexibility concerns we had with MVC. In fact, MVP purists would argue that the presenter should never have any references to any Android APIs or code.

Create a new package **presenter**

Presenter.java

```
package com.example.tictactoe.presenter;


public interface Presenter {

  void onCreate();
  void onPause();
  void onResume();
  void onDestroy();

}
```

TicTacToePresenter.java

```
package com.example.tictactoe.presenter;

import com.example.tictactoe.model.Board;
import com.example.tictactoe.model.Player;
import com.example.tictactoe.view.TicTacToeView;

public class TicTacToePresenter implements Presenter {

  private TicTacToeView view;
  private Board model;

  public TicTacToePresenter(TicTacToeView view) {
    this.view = view;
    this.model = new Board();
```

```
  }

  @Override
  public void onCreate() {
    model = new Board();
  }

  @Override
  public void onPause() {

  }

  @Override
  public void onResume() {

  }

  @Override
  public void onDestroy() {

  }

  public void onButtonSelected(int row, int col) {
    Player playerThatMoved = model.mark(row, col);

    if(playerThatMoved != null) {
      view.setButtonText(row, col, playerThatMoved.toString());

      if (model.getWinner() != null) {
        view.showWinner(playerThatMoved.toString());
      }
    }
  }

  public void onResetSelected() {
    view.clearWinnerDisplay();
    view.clearButtons();
    model.restart();
  }


}
```

Let's again examine what that looks like in our app.

Evaluation

This is much cleaner. We can easily unit test the presenter logic because it's not tied to any Android specific views and APIs and that also allows us to work with any other view as long as the view implements the TicTacToeView interface.

Presenter Concerns

Maintenance - Presenters, just like Controllers, are prone to collecting additional business logic, sprinkled in, over time. At some point, developers often find themselves with large unwieldy presenters that are difficult to break apart.

MVVM

MVVM with Data Binding on Android has the benefits of easier testing and modularity, while also reducing the amount of glue code that we have to write to connect the view + model. To configure your app to use data binding, enable the **dataBinding** build option in your *build.gradle* file in the app module.

```
android {
...
   dataBinding {
     enabled = true
   }
}
```

More about data binding at -
https://developer.android.com/topic/libraries/data-binding/start

Let's examine the parts of MVVM.

Model

Same as MVC / No change

View

The view binds to observable variables and actions exposed by the viewModel in a flexible way. We dont need the TicTacToeView interface.

TicTacToeActivity.java

```java
package com.example.tictactoe.view;

import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;

import androidx.appcompat.app.AppCompatActivity;
import androidx.appcompat.widget.Toolbar;
import androidx.databinding.DataBindingUtil;

import com.example.tictactoe.R;
import com.example.tictactoe.databinding.TictactoeBinding;
import com.example.tictactoe.viewmodel.TicTacToeViewModel;

public class TicTacToeActivity extends AppCompatActivity {

  TicTacToeViewModel viewModel = new TicTacToeViewModel();

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    TictactoeBinding binding = DataBindingUtil.setContentView(this, R.layout.tictactoe);
    binding.setViewModel(viewModel);
    Toolbar myToolbar = (Toolbar) findViewById(R.id.my_toolbar);
    setSupportActionBar(myToolbar);
    viewModel.onCreate();
  }

  @Override
  protected void onPause() {
    super.onPause();
    viewModel.onPause();
  }

  @Override
  protected void onResume() {
    super.onResume();
    viewModel.onResume();
  }

  @Override
  protected void onDestroy() {
    super.onDestroy();
    viewModel.onDestroy();
  }
```

```
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.menu_tictactoe, menu);
        return true;
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.action_reset:
                viewModel.onResetSelected();
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }

}
```

ViewModel

The ViewModel is responsible for wrapping the model and preparing observable data needed by the view. It also provides hooks for the view to pass events to the model. The ViewModel is not tied to the view however.

Create a folder called **viewmodel** or (refactor presenter)

ViewModel.java the interface code is similar to Presenter.java

```
package com.example.tictactoe.viewmodel;

public interface ViewModel {

    void onCreate();
    void onPause();
    void onResume();
    void onDestroy();

}
```

TicTacToeViewModel.java

```
package com.example.tictactoe.viewmodel;

import androidx.databinding.ObservableArrayMap;
```

```java
import androidx.databinding.ObservableField;

import com.example.tictactoe.model.Board;
import com.example.tictactoe.model.Player;

public class TicTacToeViewModel implements ViewModel {

  private Board model;

  public final ObservableArrayMap<String, String> cells = new ObservableArrayMap<>();
  public final ObservableField<String> winner = new ObservableField<>();

  public TicTacToeViewModel() {
    model = new Board();
  }

  @Override
  public void onCreate() {

  }

  @Override
  public void onPause() {

  }

  @Override
  public void onResume() {

  }

  @Override
  public void onDestroy() {

  }

  public void onResetSelected() {
    model.restart();
    winner.set(null);
    cells.clear();
  }

  public void onClickedCellAt(int row, int col) {
    Player playerThatMoved = model.mark(row, col);
    cells.put("" + row + col, playerThatMoved == null ? null : playerThatMoved.toString());
    winner.set(model.getWinner() == null ? null : model.getWinner().toString());
  }
```

```
}
```

To convert a regular layout to Data Binding layout:

1. Wrap your layout with a <layout> tag
2. Add layout variables (optional)
3. Add layout expressions (optional)

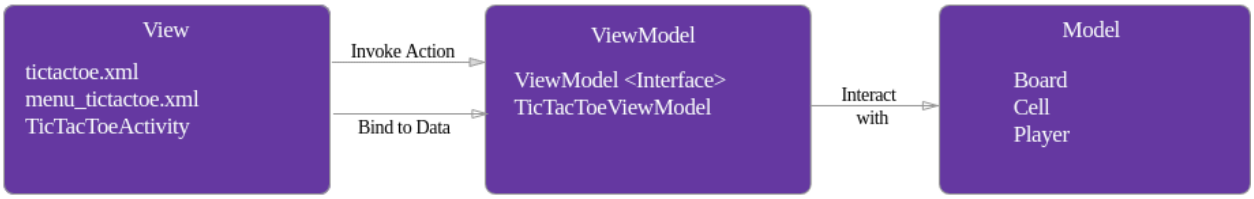
tictactoe.xml in res/layout folder

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

  <data>
    <import type="android.view.View" />
    <variable name="viewModel" type="com.example.tictactoe.viewmodel.TicTacToeViewModel" />
  </data>

  <LinearLayout
    android:id="@+id/tictactoe"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center_horizontal"
    android:orientation="vertical"
    tools:context="com.example.tictactoe.view.TicTacToeActivity">

    <androidx.appcompat.widget.Toolbar
      android:id="@+id/my_toolbar"
      android:layout_width="match_parent"
      android:layout_height="?attr/actionBarSize"
      android:background="?attr/colorPrimary"
      android:elevation="4dp"
      android:theme="@style/ThemeOverlay.MaterialComponents.Dark.ActionBar"
      app:popupTheme="@style/ThemeOverlay.MaterialComponents.Light"/>

    <GridLayout
      android:id="@+id/buttonGrid"
      android:paddingBottom="@dimen/activity_vertical_margin"
      android:paddingLeft="@dimen/activity_horizontal_margin"
      android:paddingRight="@dimen/activity_horizontal_margin"
      android:paddingTop="@dimen/activity_vertical_margin"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
android:columnCount="3"
android:rowCount="3">

<Button
    style="@style/tictactoebutton"
    android:onClick="@{() -> viewModel.onClickedCellAt(0,0)}"
    android:text='@{viewModel.cells["00"]}' />

<Button
    style="@style/tictactoebutton"
    android:onClick="@{() -> viewModel.onClickedCellAt(0,1)}"
    android:text='@{viewModel.cells["01"]}' />

<Button
    style="@style/tictactoebutton"
    android:onClick="@{() -> viewModel.onClickedCellAt(0,2)}"
    android:text='@{viewModel.cells["02"]}' />

<Button
    style="@style/tictactoebutton"
    android:onClick="@{() -> viewModel.onClickedCellAt(1,0)}"
    android:text='@{viewModel.cells["10"]}' />

<Button
    style="@style/tictactoebutton"
    android:onClick="@{() -> viewModel.onClickedCellAt(1,1)}"
    android:text='@{viewModel.cells["11"]}' />

<Button
    style="@style/tictactoebutton"
    android:onClick="@{() -> viewModel.onClickedCellAt(1,2)}"
    android:text='@{viewModel.cells["12"]}' />

<Button
    style="@style/tictactoebutton"
    android:onClick="@{() -> viewModel.onClickedCellAt(2,0)}"
    android:text='@{viewModel.cells["20"]}' />

<Button
    style="@style/tictactoebutton"
    android:onClick="@{() -> viewModel.onClickedCellAt(2,1)}"
    android:text='@{viewModel.cells["21"]}' />

<Button
    style="@style/tictactoebutton"
    android:onClick="@{() -> viewModel.onClickedCellAt(2,2)}"
    android:text='@{viewModel.cells["22"]}' />
```

```
    </GridLayout>

    <LinearLayout
        android:id="@+id/winnerPlayerViewGroup"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:gravity="center"
        android:orientation="vertical"
        android:visibility="@{viewModel.winner != null ? View.VISIBLE : View.GONE}"
        tools:visibility="visible">

        <TextView
            android:id="@+id/winnerPlayerLabel"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_margin="20dp"
            android:textSize="40sp"
            android:text="@{viewModel.winner}"
            tools:text="X" />

        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/winner"
            android:textSize="30sp" />

    </LinearLayout>

  </LinearLayout>
</layout>
```

High level breakdown for Tic Tac Toe.



Homework

Try to implement the Book Search app discussed in lecture using all three patterns - MVC/MVP/MVVM and post your work in Google classroom.