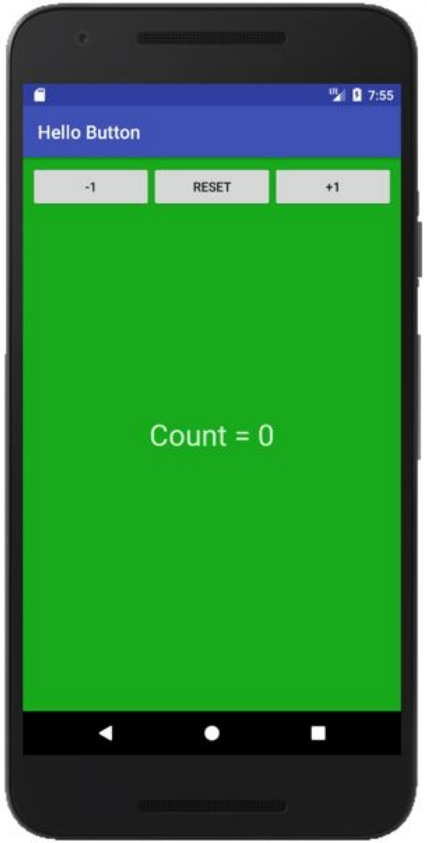
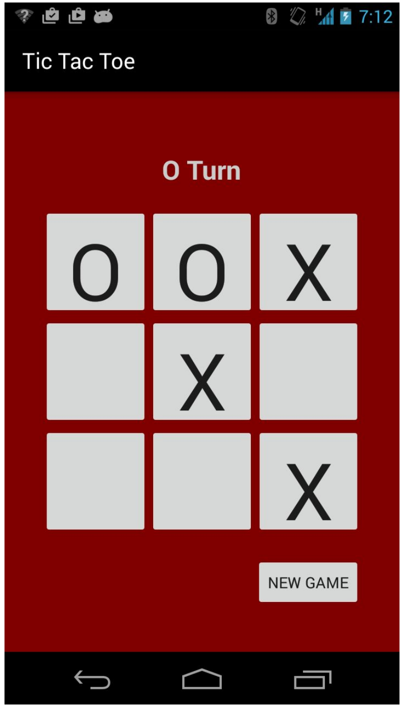
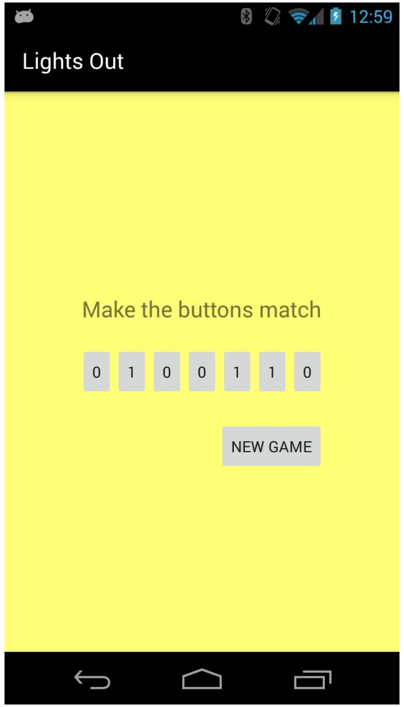
Buttons and the Model-View-Controller Pattern



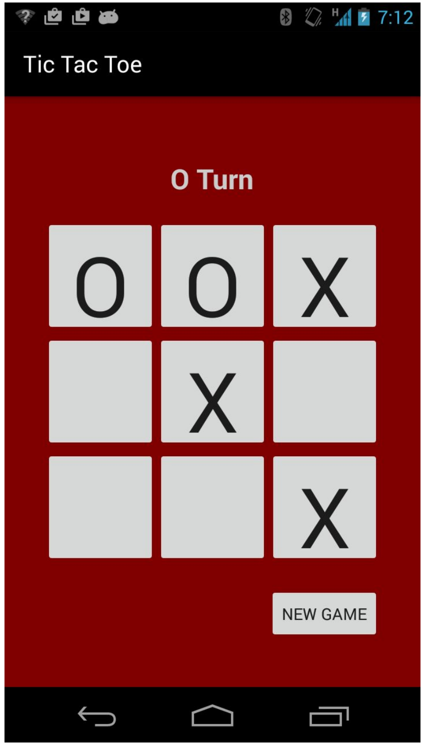




ROSE-HULMAN INSTITUTE OF TECHNOLOGY

By the end of this unit you should be able to...

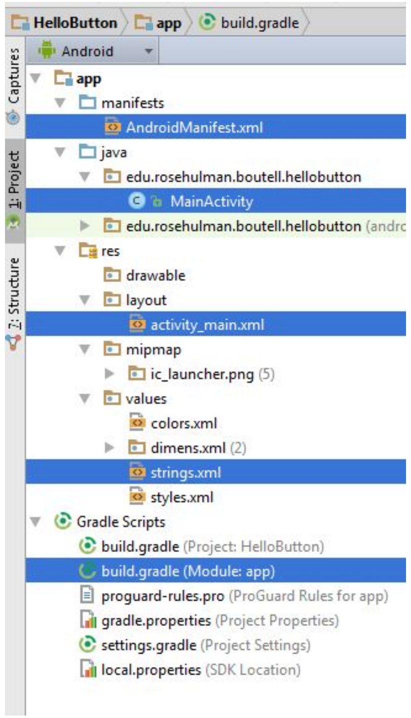
- Build apps that use buttons
- Build simple UIs to specification
  - Linear, relative, and table layouts
- Declare and use various resource values
  - Strings
  - Colors
  - Views
- Explain how Android supports the Model-View-Controller (MVC) pattern



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

New Android Application Project

In this lesson you will learn how to create a new Android project and to understand the various parts you are given



```
HelloButton > app > build.gradle
Android
▼ app
  ▼ manifests
      AndroidManifest.xml
  ▼ java
    ▼ edu.rosehulman.boutell.hellobutton
        MainActivity
    ▶ edu.rosehulman.boutell.hellobutton (andr
  ▼ res
      drawable
    ▼ layout
        activity_main.xml
    ▼ mipmap
      ▶ ic_launcher.png (5)
    ▼ values
        colors.xml
      ▶ dimens.xml (2)
        strings.xml
        styles.xml
▼ Gradle Scripts
    build.gradle (Project: HelloButton)
    build.gradle (Module: app)
    proguard-rules.pro (ProGuard Rules for app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Create a new Android Application Project

Android Studio > File > New … > New Project...

The defaults are pretty good, but follow these standards:

Use a descriptive app name: **HelloButton**

- for exams, will require your username, like **Exam1fFisher**

Company domain: use your name, like **fisher.rosehulman.edu**

Project location: anywhere you can find it.

- there is no concept of a *workspace*.

Default is good (Min SDK of 4.0.3 / API 15)

Empty Activity for now

Activity Name: MainActivity is often descriptive enough.

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

What are the key files?

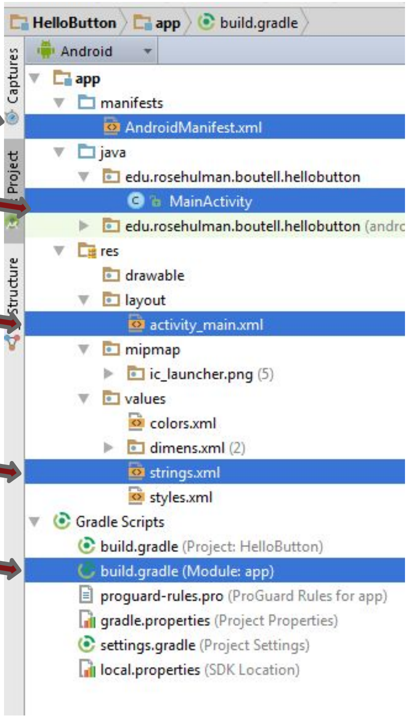
app permissions

model and controller Java code

view xml

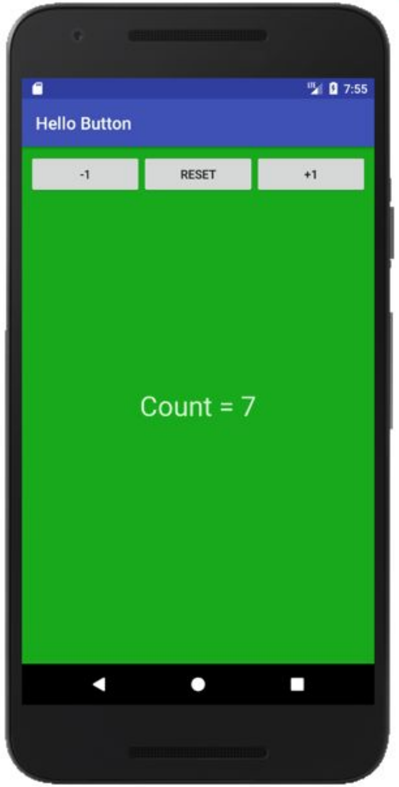
all screen text

which external libraries to use


HelloButton > app > build.gradle
Android
- app
  - manifests
    - AndroidManifest.xml
  - java
    - edu.rosehulman.boutell.hellobutton
      - MainActivity
    - edu.rosehulman.boutell.hellobutton (andro
  - res
    - drawable
    - layout
      - activity_main.xml
    - mipmap
      - ic_launcher.png (5)
    - values
      - colors.xml
      - dimens.xml (2)
      - strings.xml
      - styles.xml
- Gradle Scripts
  - build.gradle (Project: HelloButton)
  - build.gradle (Module: app)
  - proguard-rules.pro (ProGuard Rules for app)
  - gradle.properties (Project Properties)
  - settings.gradle (Project Settings)
  - local.properties (SDK Location)

Hello Button View

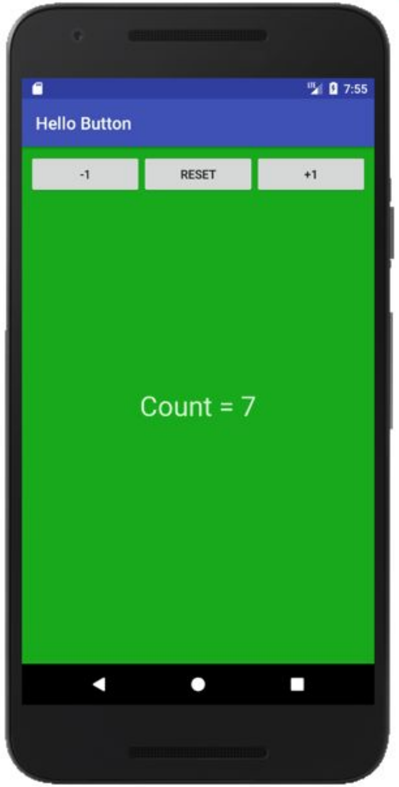
In this lesson you will learn how to implement a UI from a specification



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Implementing a UI from a specification

Start by planning resources: strings, colors



Plan by adding strings to strings.xml to make it easy to update later. Like translating to a new language!

Plan your resources if possible:

```
<resources>
   <string name="app_name">Hello Button</string>
   <string name="message_start">Count = 0</string>
   <string name="message_format">Count = %d</string>
   <string name="button_decrement">-1</string>
   <string name="button_reset">Reset</string>
   <string name="button_increment">+1</string>
</resources>
```

Note: There is also a thing called ‘plurals’ that make this more elegant, but more complex. Given that this is our first app, I won’t bother with ‘plurals’ today. Read more here: http://developer.android.com/guide/topics/resources/string-resource.html

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Create color resources: colors.xml

File > New > Other > Android xml file if it isn't there.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="colorPrimary">#3F51B5</color>
   <color name="colorPrimaryDark">#303F9F</color>
   <color name="colorAccent">#FF4081</color>
   <color name="background">#ff00aa00</color>
   <color name="text">#dfd</color>
</resources>
```

#<alpha><red><green><blue>

Colors given in RGB or ARGB format (A = alpha = transparency)
Background is bright green: (red, green, blue) = (0, 170, 0)
Text is very light green: (238, 255, 238)
Each value can be **1 or 2 hex digits, alpha is optional**

Implementing a UI from a specification

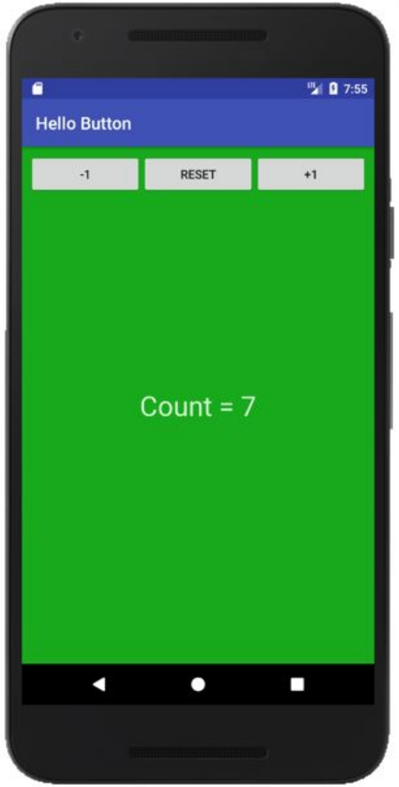
Observe:

- Text is centered in both directions, 32 sp, light green
- LinearLayout has a 8 dp margin on all sides
- Buttons equally fill the Linear Layout
- Green background

When elements are centered in screen or defined in relation to each other (aligned, above, below, etc), a **RelativeLayout** often works well. Even spacing in a row or column a **LinearLayout** often works well. Nesting layouts deeply is bad, this one is ok though. :)

Implement in layout > activity_main.xml now!

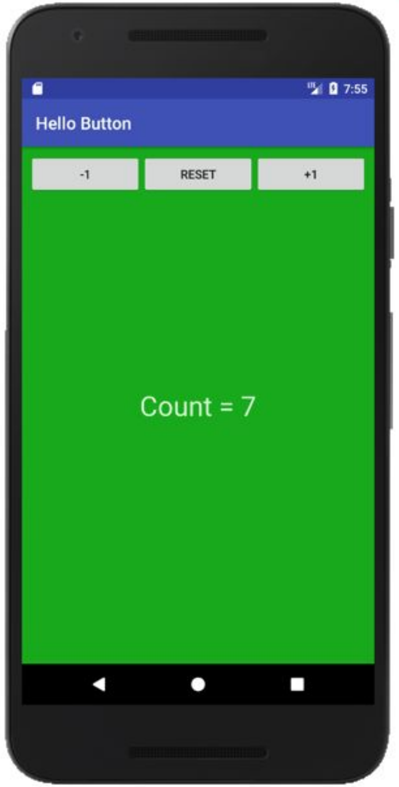
dp = density-independent pixels. Like px, but works with multiple resolutions. **Use for layouts.**
sp = scale-independent pixels. Like dp, but scaled by user's font size preference. **Use for text size.**
http://developer.android.com/guide/topics/resources/more-resources.html#Dimension



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

A Solution

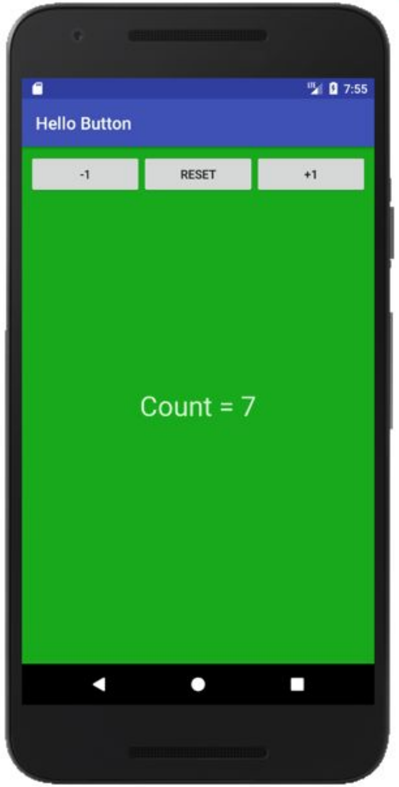
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"   android:layout_height="match_parent"
  android:background="@color/background"
  tools:context="edu.rosehulman.fisher.hellobutton.MainActivity">

  <LinearLayout
    android:layout_width="match_parent"      android:layout_height="wrap_content"
    android:layout_margin="8dp"      android:orientation="horizontal">
    <Button
      android:layout_width="0dp"        android:layout_height="wrap_content"
      android:layout_weight="1"        android:text="@string/button_decrement"  />
    <Button
      android:layout_width="0dp"        android:layout_height="wrap_content"
      android:layout_weight="1"        android:text="@string/button_reset"  />
    <Button
      android:layout_width="0dp"        android:layout_height="wrap_content"
      android:layout_weight="1"        android:text="@string/button_increment"  />
  </LinearLayout>

  <TextView
    android:id="@+id/message_text_view"      android:layout_width="wrap_content"
    android:layout_height="wrap_content"      android:text="@string/message_start"
    android:layout_centerHorizontal="true"      android:layout_centerVertical="true"
    android:textColor="@color/text"      android:textSize="32sp"/>

</RelativeLayout>
```



ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Hello Button Controller

In this lesson you will learn how to refer to resources in code and to create the button logic



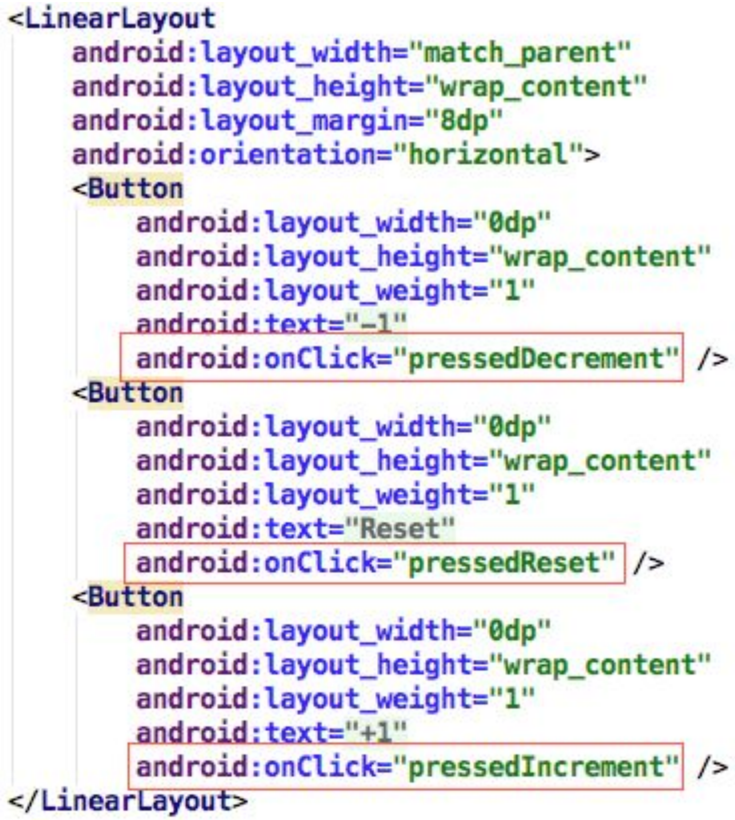
Making an instance variable (field) to track the state

```
public class MainActivity extends AppCompatActivity {

    private int count = 0;

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

    }
}
```

Capture references to Views using findViewById()

```
public class MainActivity extends AppCompatActivity {
        private int count = 0;
        private TextView textView;

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

                textView = (TextView) findViewById(R.id.message_text_view);
        }
}
```

Set a callback for button clicks



```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="8dp"
    android:orientation="horizontal">
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="-1"
        android:onClick="pressedDecrement" />
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Reset"
        android:onClick="pressedReset" />
    <Button
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="+1"
        android:onClick="pressedIncrement" />
</LinearLayout>
```

Set a callback for button clicks

```
public class MainActivity extends AppCompatActivity {

 private int count = 0;
 private TextView textView;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.activity_main);
  textView = (TextView) findViewById(R.id.message_text_view);
 }

 public void pressedIncrement(View view) {
  count++;
  updateView();
 }

 public void pressedDecrement(View view) {
  count--;
  updateView();
 }

 public void pressedReset(View view) {
  count = 0;
  updateView();
 }
```

Set a callback for button clicks

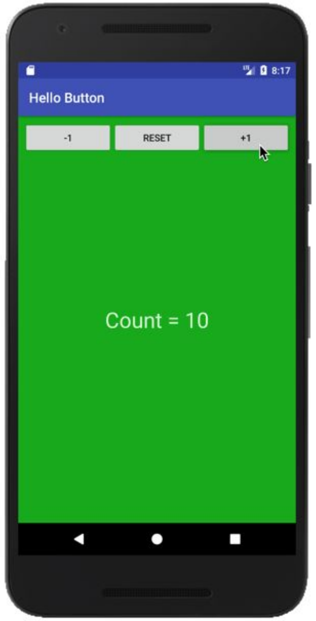
Demonstrating log (very useful) and UI updates

```
public void pressedIncrement(View view) {
  count++;
  updateView();
}

public void pressedDecrement(View view) {
  count--;
  updateView();
}

public void pressedReset(View view) {
  count = 0;
  updateView();
}

public void updateView() {
  Log.d("HelloButton", "Count updated to " + count);
  textView.setText(getString(R.string.message_format, count));
}
```

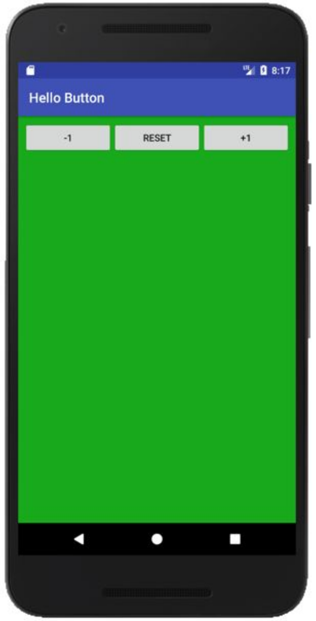
ROSE-HULMAN INSTITUTE OF TECHNOLOGY

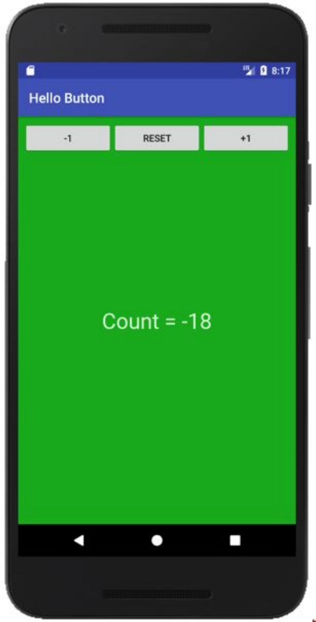
Challenge

See if you can make the text **visible** when count <= 10, but **invisible when the count > 10**
Hint: .setVisibility(View.*INVISIBLE*)



Count = 10





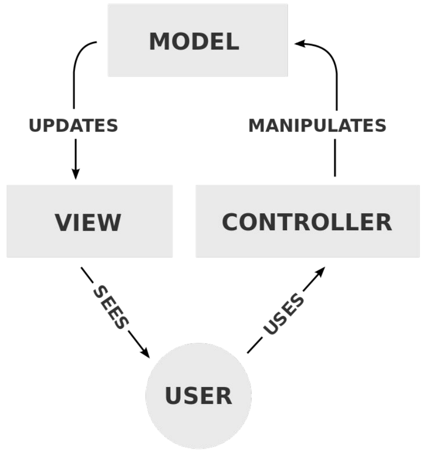
Count = -18

Note, it should disappear >10, but come back if count is ever <=10

ROSE-HULMAN INSTITUTE OF TECHNOLOGY

Model View Controller (MVC) and Tic Tac Toe Model

In this lesson you will learn how Android helps you follow the MVC paradigm


MODEL
UPDATES
MANIPULATES
VIEW
CONTROLLER
SEES
USES
USER