# Task P – Spike: Dice Rolling Game

## Goals:

*The goal of the assignment is to create a dice-rolling game. This game needs to include the state-handle, creating constraint layout, localization and using the Log for debugging*

*List of knowledge gaps:*

- Design
- Button Event listener
- Display number and image view
- Game logic
- Save state when rotating the device
- Localisation
- Debugging with log
- IDE command
- Testing the app

This is the link to my GitHub https://github.com/SoftDevMobDev-2023-Classrooms/core1-bbi3mn4u69.git

## Tools and Resources Used

- Android Studio
- Build a Responsive UI with ConstraintLayout n.d., Android Developers. <https://developer.android.com/develop/ui/views/layout/constraint-layout>
- Localize the UI with Translations Editor | Android Studio n.d., Android Developers. < https://developer.android.com/studio/write/translations-editor>
- How to Translate Your Android App to Any Language (SO EASY!) - Android Studio Tutorial n.d., www.youtube.com, viewed 1 September 2023, <https://www.youtube.com/watch?v=LXbpsBtIleM&t=277s>.
- Dice image < https://www.flaticon.com/search?word=dice >

## Knowledge Gaps and Solutions

### Gap 1: Design

#### A. Portrait and Landscape design

Before come to coding the app, we have to design the overall UI of the app. We can access it in the activity_main.xml

In this app, I will create 2 layouts, the first one for portrait, the second one will be the landscape layout.

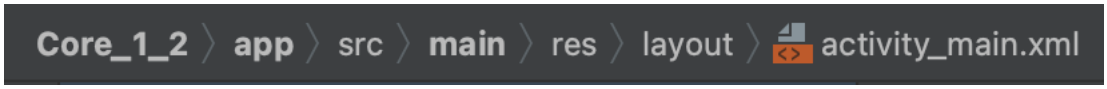


**Figure 1 layout path**

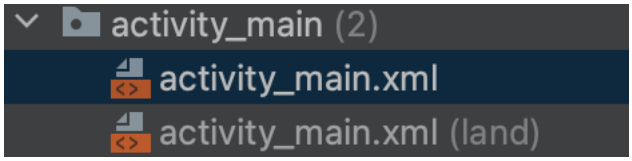


**Figure 2 two layout**

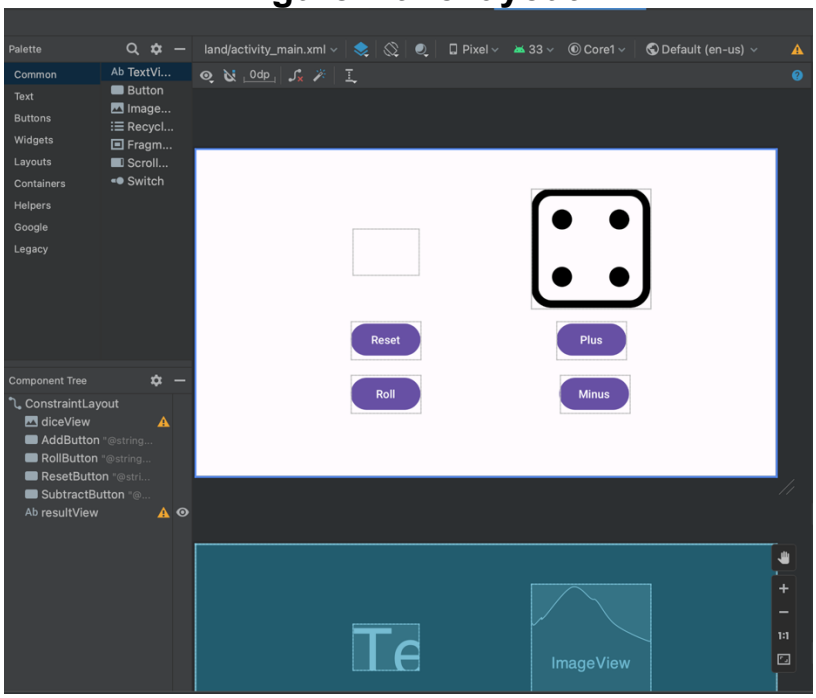


**Figure 3 landscape layout design**

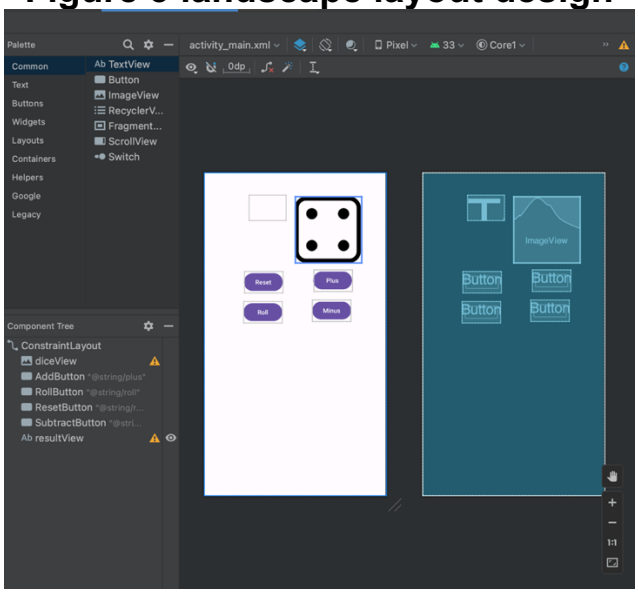


**Figure 4 portrait layout design**

## B. Constraint layout

In android, Constraint layout give us abilities to create large and complex layout with a flat view hierarchy. We can easy create layout by simply drag and drop

We will run into error if we do not specify both horizontal and vertical constraint for the object. For example, in this build, I forgot to add the horizontal constraint to the button.

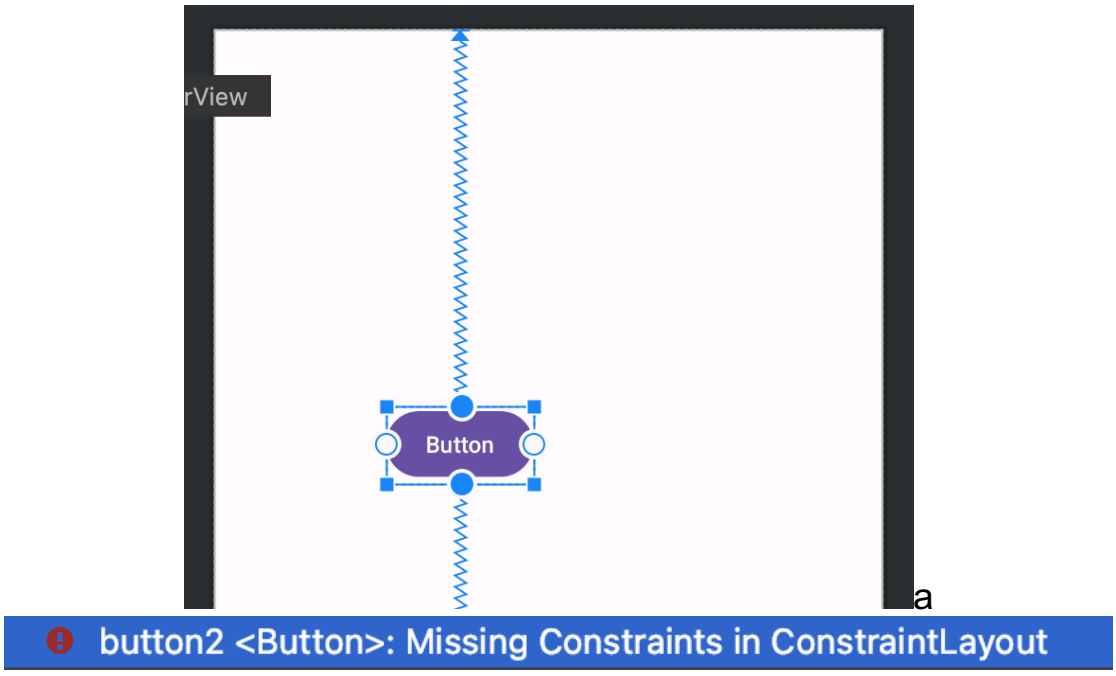


**Figure 5 missing horizontal constraint**

In this game I will using constraint layout to build the portrait layout as well as the landscape layout. Down below is how I used the constraint layout for the text view, image and button.

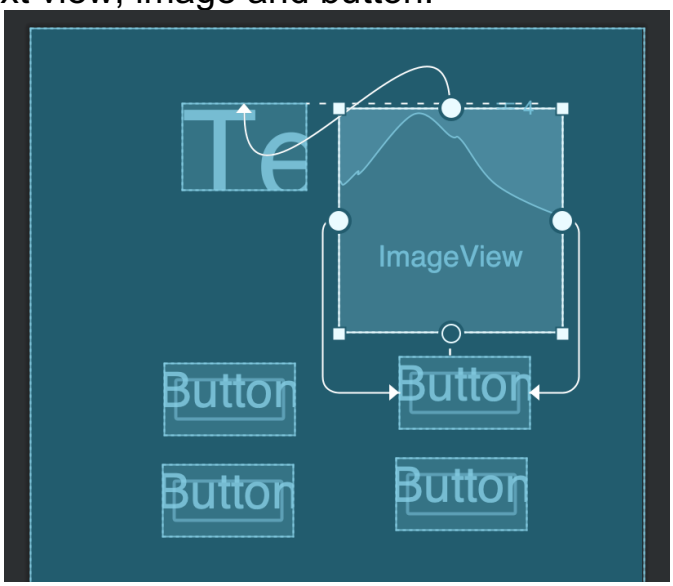


**Figure 6 image view**

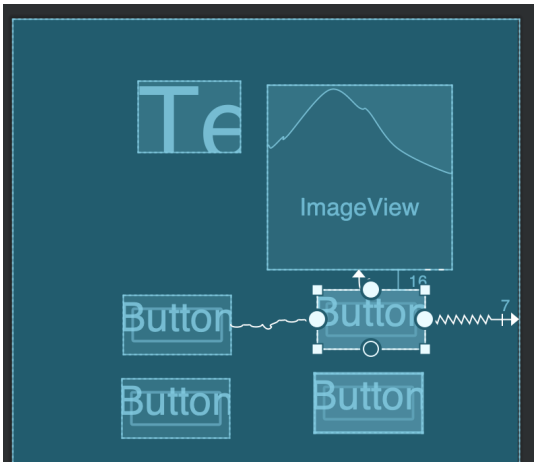


**Figure 7 The add button**

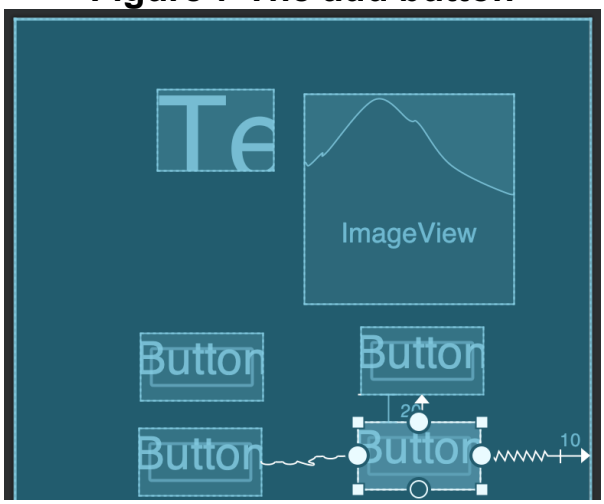


**Figure 8 the minus button**

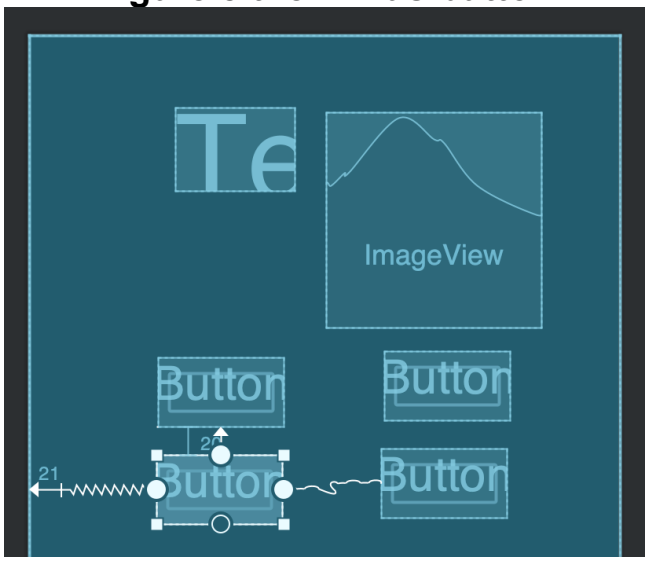


**Figure 9 the roll button**

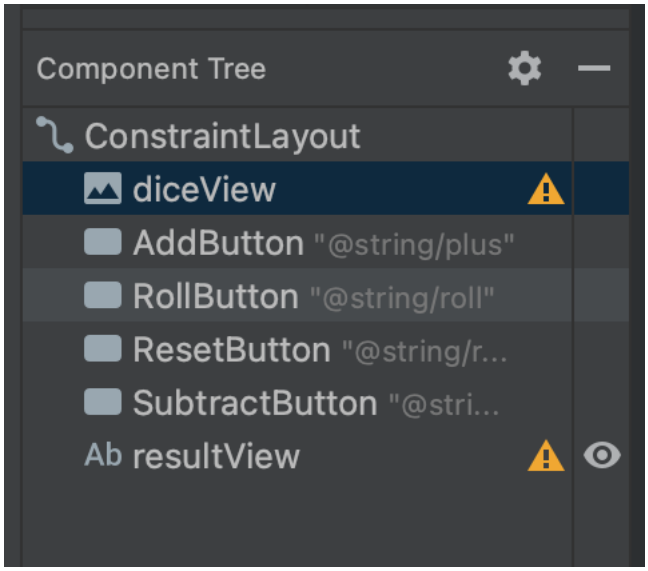


**Figure 10 constraint layout**

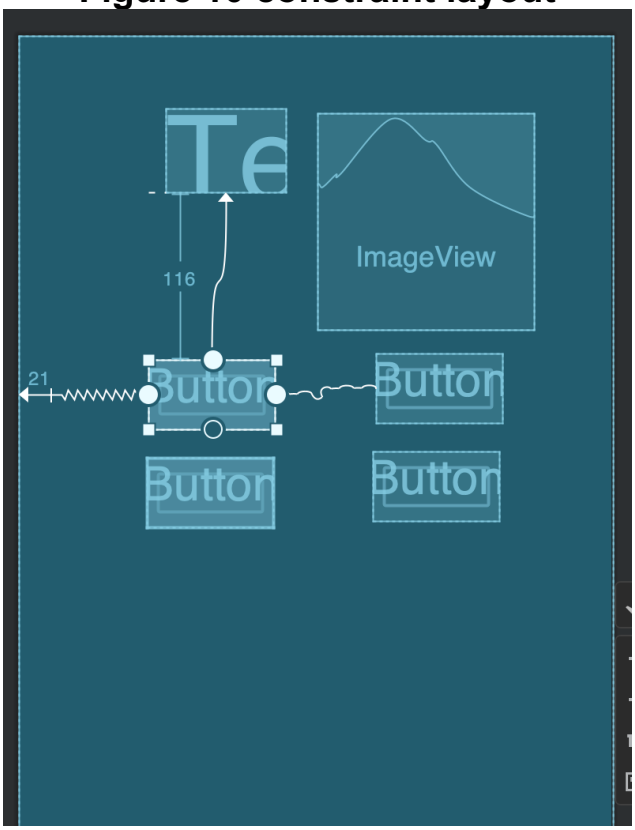


**Figure 11 blueprint**

Similarly, I also create a constraint layout for the landscape orientation:

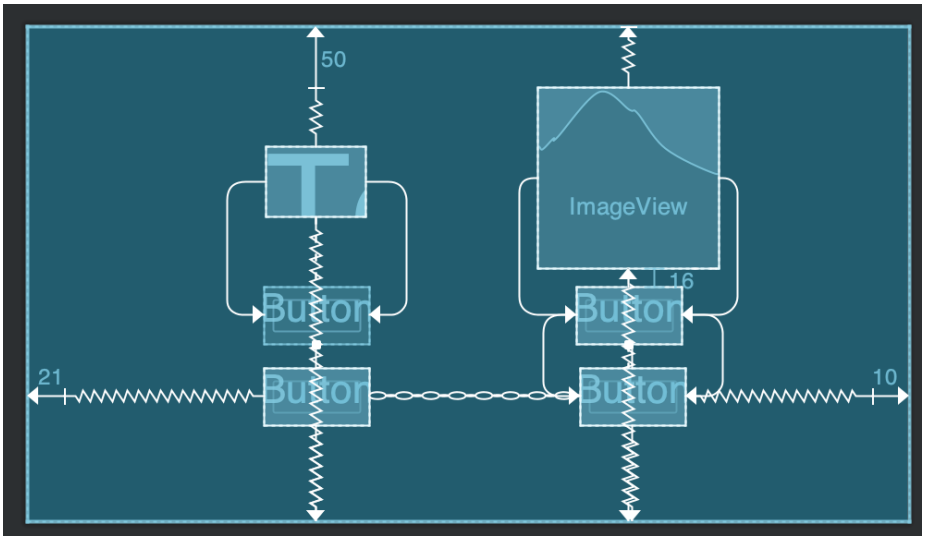


**Figure 12 constraint layout for landscape**

**Gap 2: Button event listener**

To make the button work, the button event listener needs to be created in the MainActivity.kt
First, we need to define the button using the findViewById:

```
val rollButton = findViewById<Button>(R.id.RollButton)
val addButton = findViewById<Button>(R.id.AddButton)
val subtractButton = findViewById<Button>(R.id.SubtractButton)
val resetButton = findViewById<Button>(R.id.ResetButton)
```

**Figure 13 define the button**

Each button is assigned to an immutable variable, then the setOnClickListener is implemented. When the button is pressed, some function will be performed. These functions we will revisit in the game logic later. We will do the same for the other button.

```
rollButton.setOnClickListener { it: View!
    Roll()
    Log.d( tag: "MainActivity",  msg: "Roll Number: $rolledNumber")
    Update()
    UpdateButtonState( roll: false,  add: true,  substract: true)
    if (TotalNumber == 20) {
        rollButton.isEnabled = isRollEnable
    }
}
```

**Figure 14 Button event listener**

**Gap 3: Display number and Image view**

For the number view and image view, first, we also need to define it with the immute variable. The number view here will be used to display the score. Meanwhile, the image here is used to display the dice.

```
// Initialize the total number display
val resultViewText = findViewById<TextView>(R.id.resultView)
resultViewText.text = "0"
```

**Figure 15 define number view**

```
// Initialize the result dice view image
val resultDiceViewImage = findViewById<ImageView>(R.id.diceView)
resultDiceViewImage.setImageResource(R.drawable.dice1) // You can
```

**Figure 16 define image view**

In here, we are using the TextView to display number. So that number are being treated as a string. To adapt, the number need to be converted to string to display it on the screen. For the image view, the image need be input in the resource file.

**Gap 4: Game Logic**

1. **the button:**

Based on the requirement, there will be four buttons including add, subtract, reset and roll. As the requirement, button should be enable when they can be used. As the reset button can be pressed any time so we will not worries about the condition for the reset button to enable or disable. Initially, there will be:

```
// Initially, disable Add, Subtract button
addButton.isEnabled = false
subtractButton.isEnabled = false
rollButton.isEnabled = true
```

**Figure 17 State of button initially**

The add button will be disable, as well as the subtract button. In the very beginning of the of the game, the roll button is being enable to roll the dice.

Add button:

```
addButton.setOnClickListener { it: View!
    Add()
    Log.d( tag: "MainActivity", msg: "Total Number: $TotalNumber")
    Update()
    UpdateButtonState( roll: true, add: false, substract: false)
    if (TotalNumber == 20) {
        isRollEnable = false
        rollButton.isEnabled = isRollEnable
    }
}
```

**Figure 18 add button onclick**

The add button when on clicked will perform the add function:

```
private fun Add() {

    TotalNumber = currentDiceValue + TotalNumber
}
```

**Figure 19 add function**

The add function will calculate the score buy adding the current dice number being rolled with the current score. I initialize a variable called TotalNumber which represent the score.

Subtract button:
Similar to add button, the subtract button is also perform a calculation which in the subtract function:

```
//        subtract button on click
        subtractButton.setOnClickListener { it: View!
            Subtract()
            Log.d( tag: "MainActivity",  msg: "Total Number: $TotalNumber")
            Update()
            UpdateButtonState( roll: true,  add: false,  substract: false)
            if (TotalNumber == 20) {
                isRollEnable = false
                rollButton.isEnabled = isRollEnable
            }
        }
```

**Figure 20 subtract button on click**

The subtract button contains a logic which a bit different from the add button, As the requirement, the score can not be a negative number so if the TotalNumber(score) is greater than the current dice values, the subtract will happen. Meanwhile, if the TotalNumber(score) is in a negative state, the score will remain 0

```
//    function for subtract button
private fun Subtract() {
    if (TotalNumber >= 0) {

        TotalNumber = TotalNumber - currentDiceValue
    }
    if (TotalNumber <= 0){
        TotalNumber = 0
    }
}
```

**Figure 21 subtract button logic**

Roll Button:

The roll button will be used to roll the dice to create a random value. But we need to check for a condtion that if the TotalNumber(score) reach 20, the roll button need to be disable:

```
//      button roll onclick
        rollButton.setOnClickListener { it: View!
            Roll()
            Log.d( tag: "MainActivity",  msg: "Roll Number: $rolledNumber")
            Update()
            UpdateButtonState( roll: false,  add: true,  substract: true)
            if (TotalNumber == 20) {
                rollButton.isEnabled = isRollEnable
            }
        }
```

**Figure 22 Roll button on click**

The assignment required we using a random seed 1, the list of number being randomed will be fixed with the number in the the defined position, we will talk more about this in the random logic

```
private fun Roll(){
    if (currentNumberIndex < rolledNumber.size) {

        currentDiceValue = rolledNumber[currentNumberIndex]
        Log.d( tag: "MainActivity",  msg: "current Number: $currentDiceValue")
        currentNumberIndex++
        DiceRolled()
        Update()
        if (currentNumberIndex == 10 ){
            currentNumberIndex = 0
        }
    }
}
```

**Figure 23 Roll button Logic**

The roll function will access the number in rolled number list of integer, which we already have a fixed list of number. Each time we press the roll button, the current dice values will be assigned to the next number in that list if the current index reach out of 10 integer in that list, it need to be set back to the beginning of the list.

Reset button:

The reset button is used to reset the score when it reaches 20. As I mentioned earlier, the roll button will be disabled when the score reaches 20, so the reset button will need to enable the roll button. Meanwhile, it will reset the score back to 0.

```
resetButton.setOnClickListener { it: View!
    isRollEnable = true
    rollButton.isEnabled = isRollEnable
    TotalNumber = 0
    Log.d( tag: "MainActivity",  msg: "Total Number: $TotalNumber")
    Update()
}
```

**Figure 24 Reset button on click**

2. **The Rolling Dice logic:**

First, we will need a variable for the random number list being created by the function RollTheDice. The random values will use the seed of 1 which generate a list of 10 integers from 1 to 7 in a fixed random position (figure 15)

```
private var rolledNumber = RollTheDice(Random( seed: 1))
```

**Figure 25 list of random number**

```
//   function to roll the dice
private fun RollTheDice(random: Random): List<Int> = List( size: 10) { it: Int
    random.nextInt( from: 1, until: 7)
}
```

**Figure 26 function to create a random number with the seed of 1**

```
Roll Number: [4, 1, 5, 1, 4, 6, 2, 4, 3, 4]
```

**Figure 27 List of random number**

If the roll button is pressed, the number will be taken from that list with the increase of the index number

Check if reach 20:

The values of TotalNumber need to be checked every time if it reaches 20. So the function named update is being created to check it every time when the button is pressed:

```
private fun Update() {
    val resultViewText = findViewById<TextView>(R.id.resultView)
    val rollButton = findViewById<Button>(R.id.RollButton)
    if (TotalNumber < 20) {
        resultViewText.text = TotalNumber.toString()
        resultViewText.setTextColor(resources.getColor(R.color.black))
    } else if (TotalNumber > 20) {
        resultViewText.text = MaxNumber.toString()
        resultViewText.setTextColor(resources.getColor(R.color.Red))
    }else if(TotalNumber == 20) {
        resultViewText.text = TotalNumber.toString()
        resultViewText.setTextColor(resources.getColor(R.color.Green))
        rollButton.isEnabled = false
    }
}
```

**Figure 28 check if reach 20**

As the requirement, the TotalNumber(score) will remain black when it is under 20. I have created a variable named MaxNumber which has the values of 20. If the score is more than 20, the MaxNumber will be displayed with the red colour, if the score is exactly equal to 20, the colour will be green.

### 3. Check button state:

One more thing across the program is to update the button state every time one button is pressed so the function called UpdateButtonState will handle that. Also in each of the buttons, this function is being called.

```
private fun UpdateButtonState(roll: Boolean, add: Boolean, substract: Boolean) {
    isRollEnable = roll
    isAddEnable = add
    isSubtractEnable = substract
    val rollButton = findViewById<Button>(R.id.RollButton)
    val addButton = findViewById<Button>(R.id.AddButton)
    val subtractButton = findViewById<Button>(R.id.SubtractButton)
    rollButton.isEnabled = isRollEnable
    addButton.isEnabled = isAddEnable
    subtractButton.isEnabled = isSubtractEnable
}
```

**Figure 29 check the button state**

Display the dice:

in order to display the dice, a function called DiceRolled will be created:

```
private fun DiceRolled() {
    // condition for image rendering
    val resultDiceViewImage = findViewById<ImageView>(R.id.diceView)
    val diceResource = listOf( R.drawable.dice1,
        R.drawable.dice2,
        R.drawable.dice3,
        R.drawable.dice4,
        R.drawable.dice5,
        R.drawable.dice6)

    if (currentDiceValue in 1 ≤ .. ≤ 6) {
        resultDiceViewImage.setImageResource(diceResource[currentDiceValue - 1])
    }
}
```

**Figure 30 display the dice**

the image to be displayed will depend on the current dice value. The image resource will be put in a list, when displaying the image, the current dice number has a range from 1 to 6 but in the list will be the range from 0 – 5 so it needs to be minus 1 to correctly display the image.

**Gap 5: Save state when rotating**

If we rotate the device, the program will onDestroy() and call onCreate() again. To
recover the values in the variable, we need to implement savedInstanceState. I will save the necessary variables which are the button state, the current dice value, and the total number (score):

```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    outState.putInt("Total_Value", TotalNumber)
    outState.putInt("Dice_Value", currentDiceValue)
    outState.putBoolean("RollState", isRollEnable)
    outState.putBoolean("AddState", isAddEnable)
    outState.putInt("CurrentIndex", currentNumberIndex)
    outState.putBoolean("SubtractState", isSubtractEnable)
}
```

**Figure 31 save instance state**

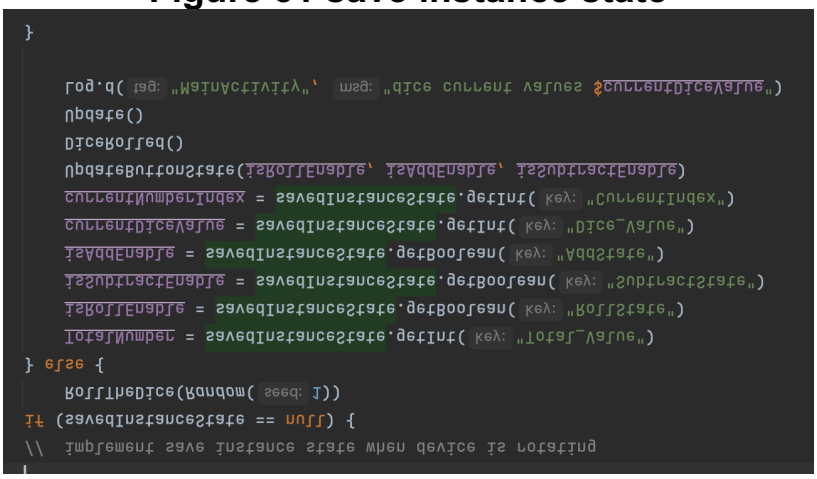

**Figure 32 save instance state**

Result:

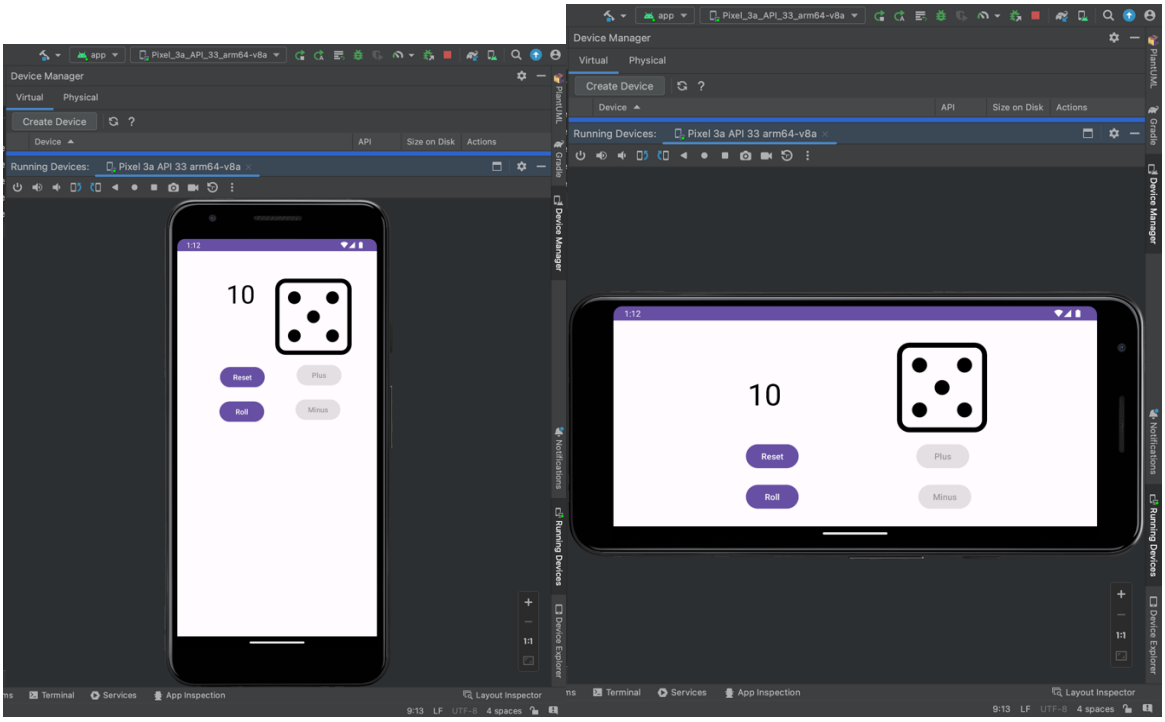


**Figure 33 result for save instance state**

**Gap 6: Localizing the app**

In android development, the string externalization is the practical of extract and manage string separately from our application code and store it in a separate source file. In here, I'm create and using the String Resources (XML Files) locate in app/src/main/res/.

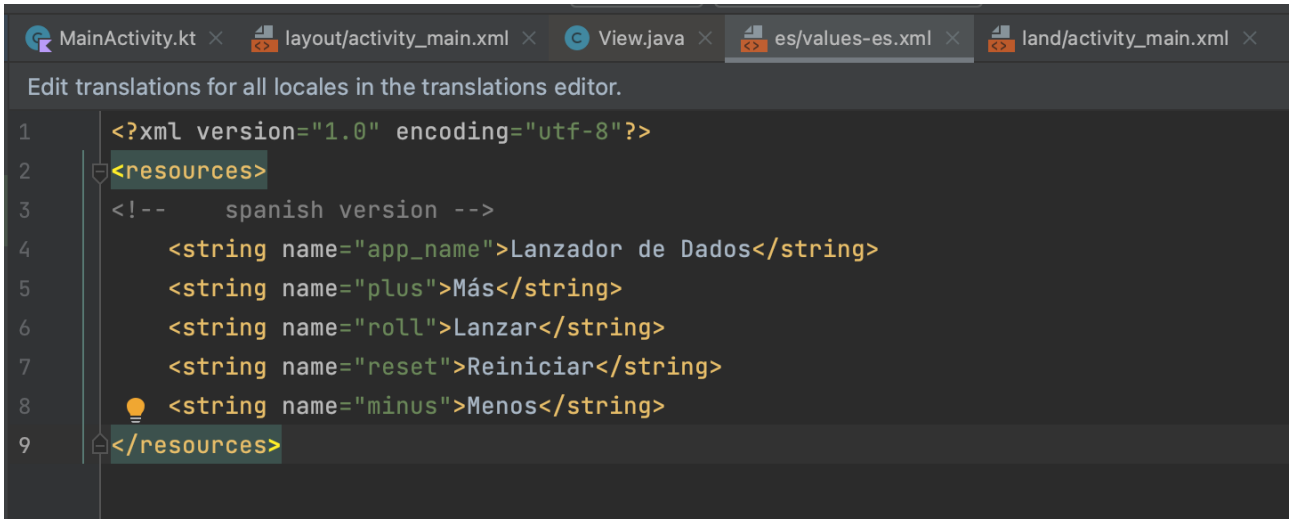


**Figure 34 localizing**

This can assist the localization in a various way by separating the text content from the source code and organize it in the resource file. These resource file can be easily accessed by the developer team and convenient to translate the text based on their location language. This comes with a range of advantage such as: eases maintenance, scalability, etc.

To make the app support different languages, I make the app adapt to 2 languages, the first one is Spanish, and the second one is Vietnamese. When changing the language in the system setting, the text in the app also being changed.

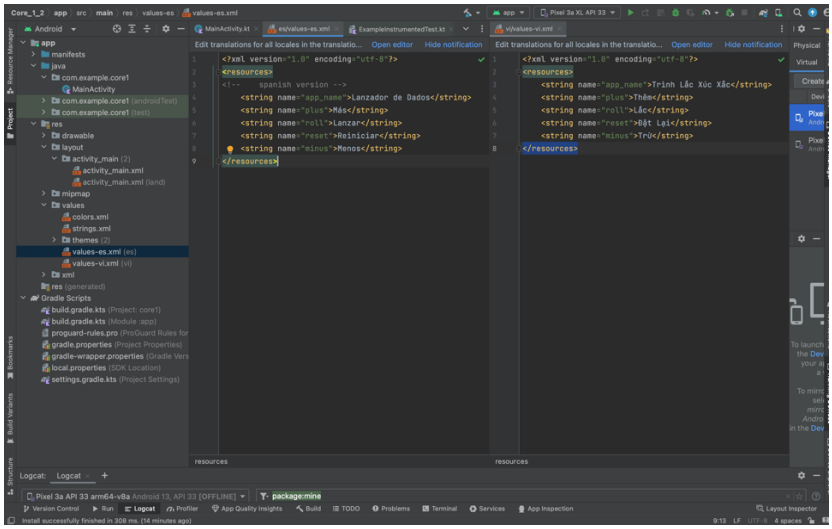

**Figure 35 localizing**

Result:

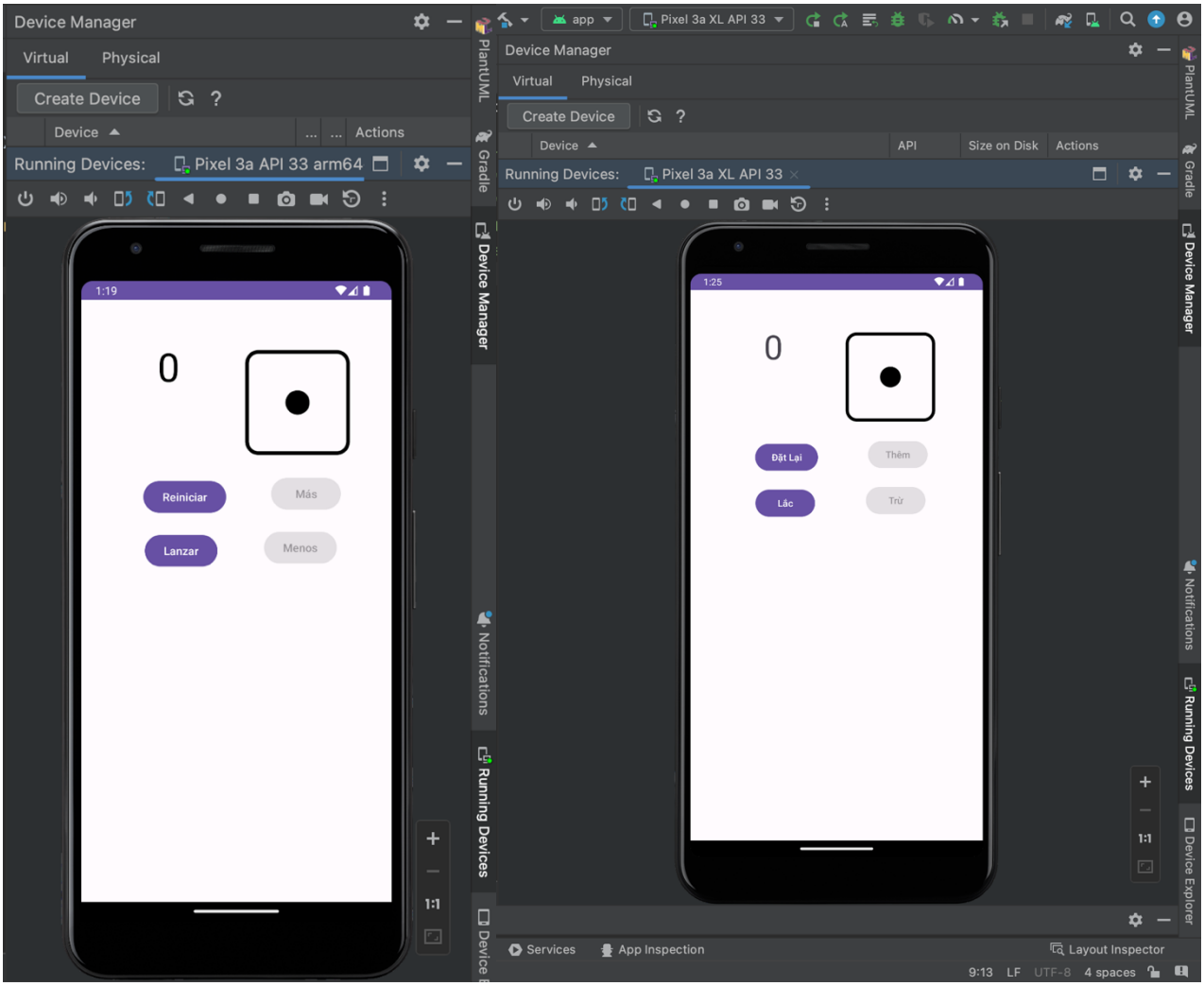


**Figure 36 result**

**Gap 7: IDE Command:**

We need to set up our emulator to run the app, We can select the one that already exists or we can create a new one in the device manager.

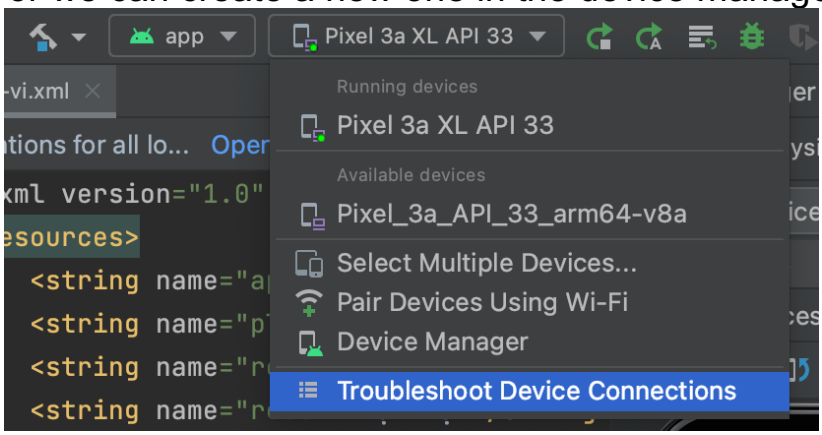


**Figure 37 select emulator**

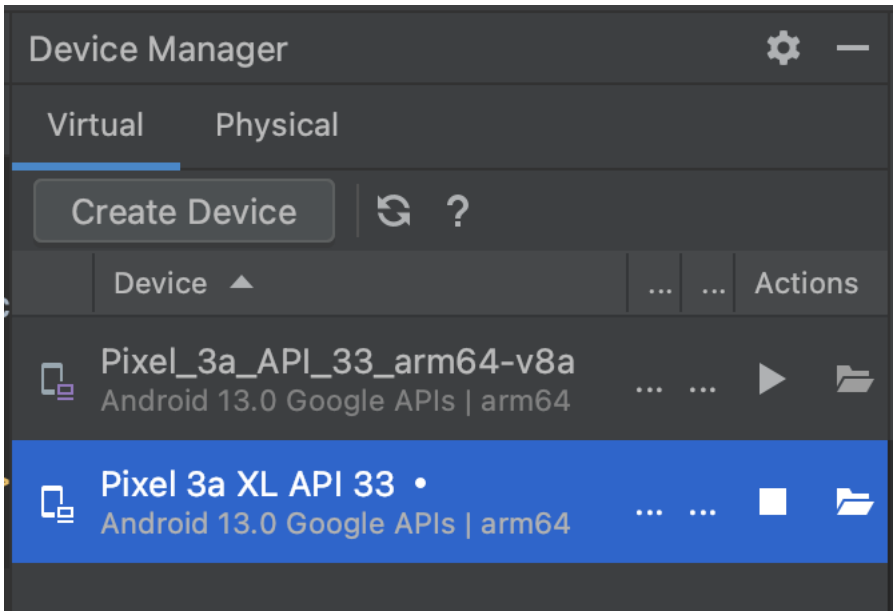


**Figure 38 device manager**

After creating the device, we can run our program by clicking the green play button, in this panel, we can also select debug the app by the bug icon:

**Figure 39 control panel**

We can debug the program using the breakpoint, the program will stop at the break point and we can see where the program goes by clicking the up or down button in the debug panel at the bottom.

```
rollButton.setOnClickListener { it: View!
    Roll()
    Log.d( tag: "MainActivity", msg: "Roll Number: $rolledNumber")
```

**Figure 40 Example of breakpoint**

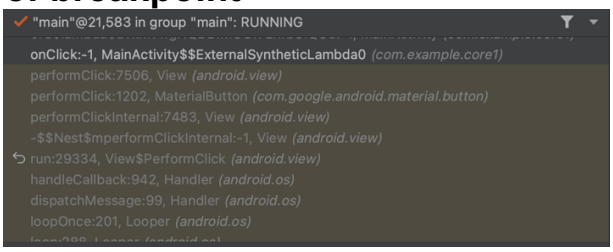


**Figure 41 list of interaction**

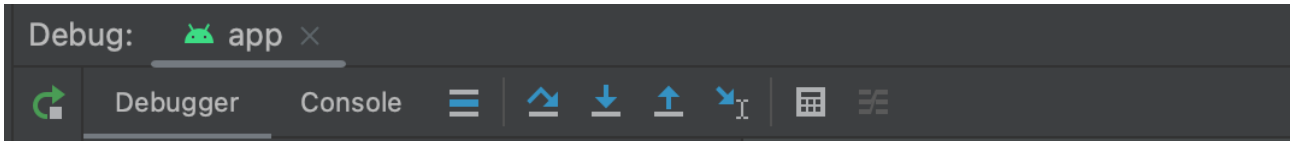


**Figure 42 debug panel**

**Gap 8: Debug with log**

When coding an app encountering errors or bugs is inevitable. Therefore, the log is a useful tool for debugging. In this app, some of the values for example list of random numbers is happen under the hood or when the score is over 20, the score will be frozen at 20  which not displayed in the interface so we can't access that. To view that variable values, I'm using log:


**Figure 43 log example**

For example, I log the current dice value, and at the time of coding the save instance state, I run into a bug in that the dice values are changed when rotating the device. After viewing that variable on the logcat, I can debug it. For another example, I want to see the list of random numbers when click the roll button, I log that list out (that list is being assigned to the rolled number var)


**Figure 44 log example**

Then I will be able to see the values of the variable that I log in the log cat (figure 36):

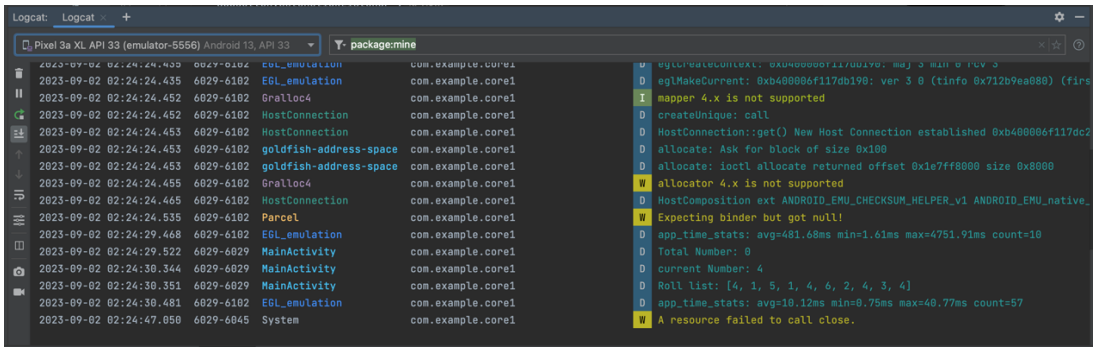


**Figure 45 log cat**

**Gap 8: Testing the app**

With the test file provided in GitHub, I take that down and test the program. The test file needs to put in the android test.

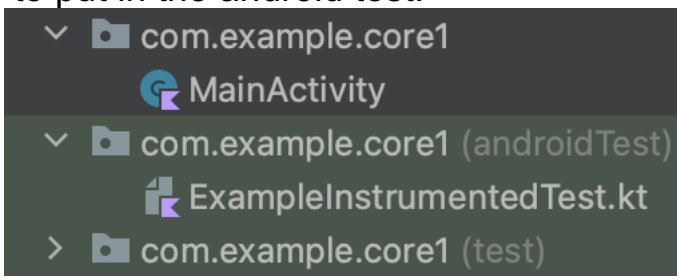


**Figure 46 test file**

After running the test, all six tests are passed without any issue:

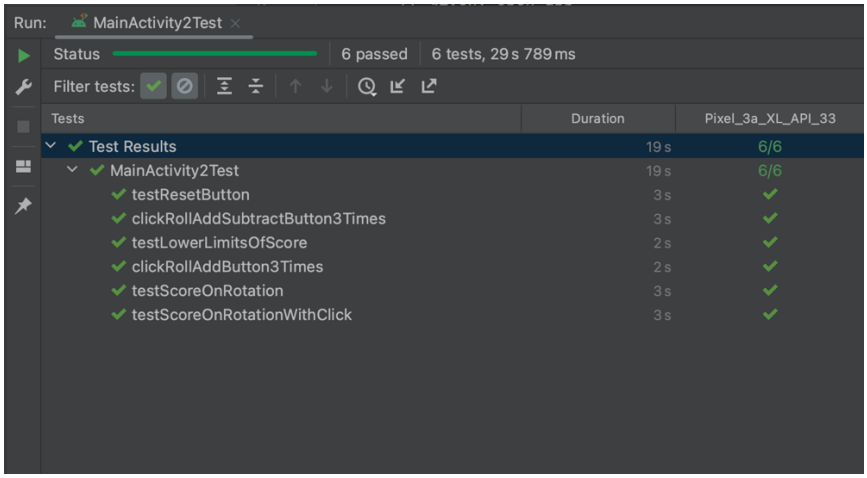


**Figure 47 passed test**