Task Core 1 – Spike: [Insert Title]

Link to GitHub repository: https://github.com/SoftDevMobDevJan2023/core1-103488117.git

Goals:

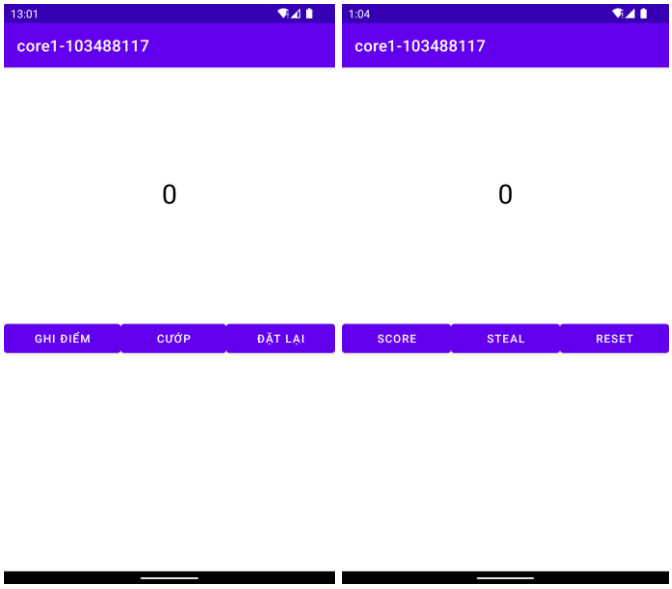
- Illustrate the personal ability to work with a single activity application.
- Understand different states of activities.
- Use logs and IDE for testing and debugging.
- Design layouts with multiple constraints.
- Apply the use of listeners successfully.
- Create an app that supports multiple languages.

Tools and Resources Used

- Android Studio IDE
- Git and GitHub
- Kotlin programming language and XML files
- Google's Sound Library: https://developers.google.com/assistant/tools/sound-library
- Android MediaPlayer API: https://developer.android.com/guide/topics/media/mediaplayer
- How to set the text colour of TextView (in our context, the "score" element): https://stackoverflow.com/questions/8472349/how-to-set-text-color-of-a-textview-programmatically
- The course's modules

Knowledge Gaps and Solutions

Gap 1: Multiple languages



The app has 2 versions with English language and Vietnamese language. To do this, we create a new value resource file, chose the locale qualifier, and chose the Vietnamese language (choosing the region is optional). The English language has already been created as the default. The two

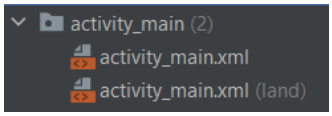
files must be identical and only the string value can be different between the string tag. String externalization supports this feature when we use the ids stored in the 2 strings.xml files. Changing the language configuration in the device will result in changing the resource file, making the usage and development of the application more convenient.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">core1-103488117</string>
    <string name="increase">GHI ĐIỂM</string>
    <string name="decrease">CƯỚP</string>
    <string name="reset">ĐẶT LẠI</string>
    <string name="default_score">0</string>
</resources>
```

```xml
<resources>
    <string name="app_name">core1-103488117</string>
    <string name="increase">SCORE</string>
    <string name="decrease">STEAL</string>
    <string name="reset">RESET</string>
    <string name="default_score">0</string>
</resources>
```

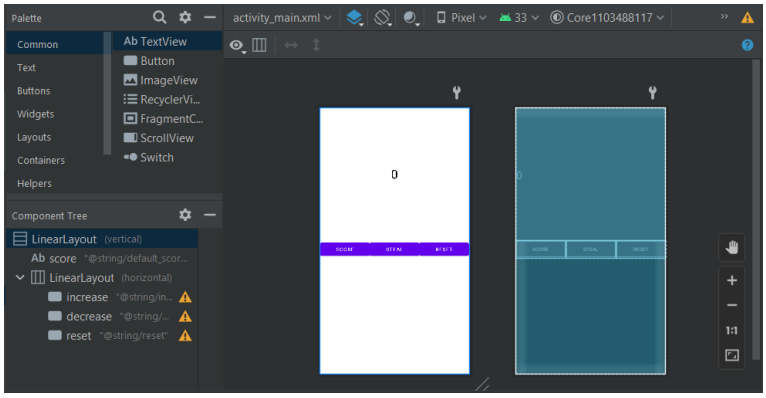
Gap 2: Multiple layouts for landscape and portrait

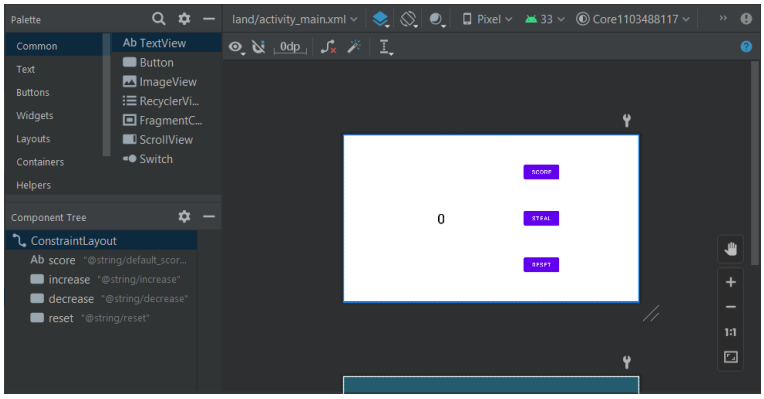
The app came with a default orientation of portrait. To create a landscape layout, we need to create a new layout resource file in the `activity_main` folder.



Gap 3: Implementation of different layouts (linear and constraints)

After speculating the demo video, I came to conclude that the portrait screen should use a linear layout while the landscape screen should use a constraint layout.

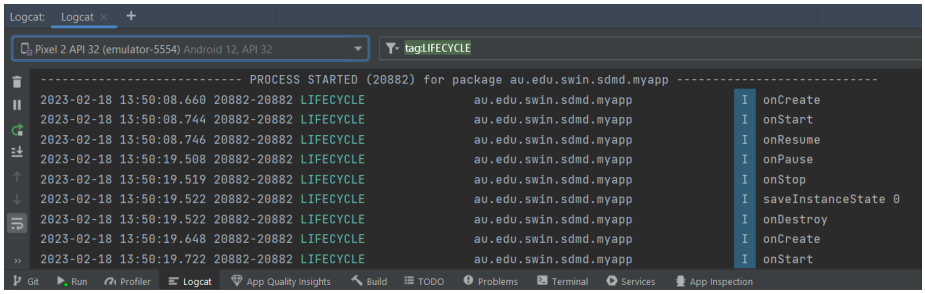




Gap 4: Logging usage

To view the logs, in this case, in each case the program enters in its lifecycle, we override the function dedicated to that cycle state and put a log message with a tag. After the program's execution, we can open Logcat to view these changes. The code and screenshot are as follows:

```
override fun onStart() {
    super.onStart()
    Log.i("LIFECYCLE", "onStart")
}
```



This log terminal helped integrate the `saveInstanceState`, as we can the current number logged out after the screen rotates and changes its state. To do this we override the `onSaveInstanceState()` function:

```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
    outState.putInt("ANSWER", currentScore)
    Log.i("LIFECYCLE", "saveInstanceState $currentScore")
}
```

Gap 5: Using listeners

The listeners were used as the backbone of the program's logic and functions. I have used the listeners to increment/decrement/reset the score, set the colour of the score, and play the sound when the score reached 15. The listener for the SCORE button is shown in the code snippet below

```
increase.setOnClickListener {
    currentScore = increase(score.text.toString())
    score.text = currentScore.toString()
    setScoreColor()
    if (currentScore == 15) {
        mediaPlayer.start()
    }
}
```

Gap 6: Apply new concepts to the application

Another requirement of the assignment was to incorporate sound. When the score reaches 15, the program should play a cheering sound (to the winning team). The link to the Google library of sound and MediaPlayer API that I have listed in the Tools section was helpful, and this part was completed with just several lines of code. The requirement also told us to change the text colour

under certain conditions, and the link to the StackOverflow discussion contained detailed examples from different API versions of Android Studio.

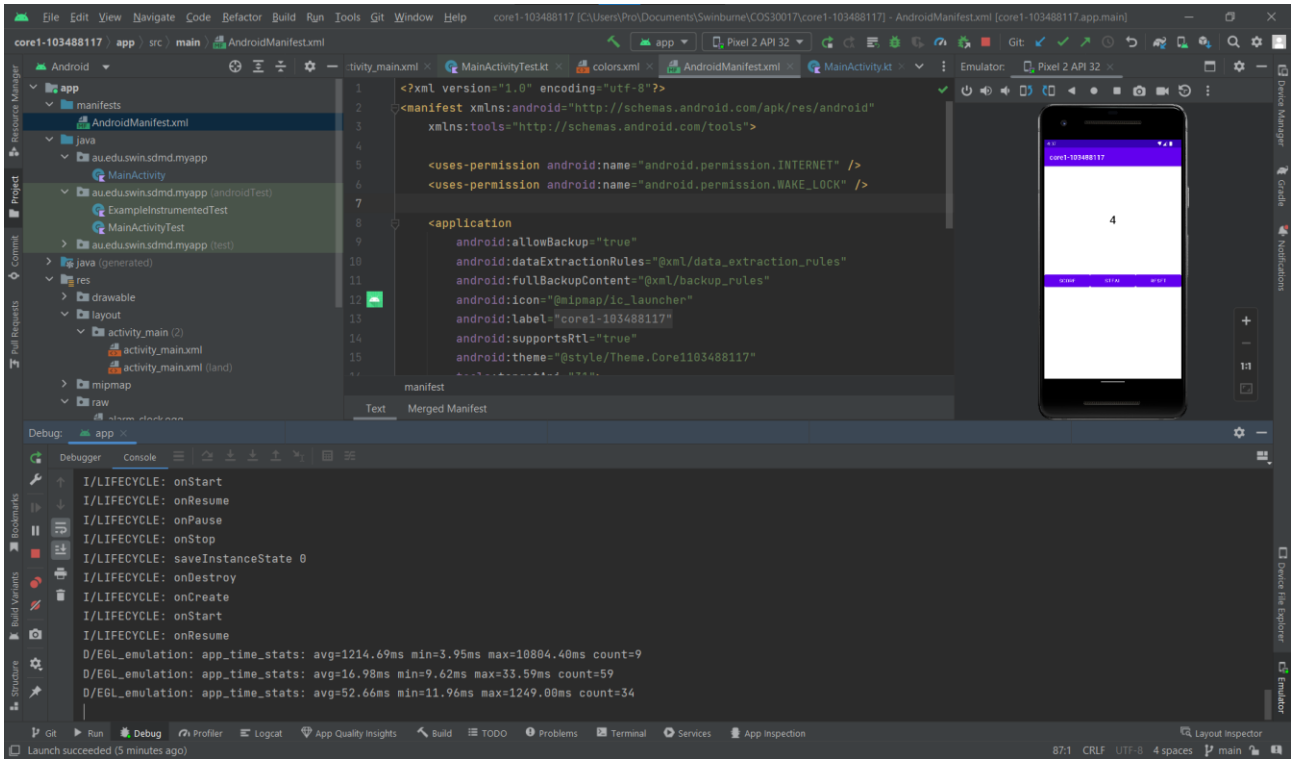
Gap 7: Saving the state upon program rotation

For this part, I used the `saveInstanceState` to save the `currentScore`. Which is described in the code snippet below. The score TextView was the element that needed to be saved upon screen rotation.

```
savedInstanceState?.let {
    currentScore = savedInstanceState.getInt("ANSWER")
    score.text = currentScore.toString()
    setScoreColor()
}
```

Gap 8: Using the IDE

The tools that I used in the IDE were run, debug, test, and log, which helped find and fix errors.



Open Issues and Recommendations

An issue that I encountered while developing the app was in the testing phase. As the notes have stated in the core 1 requirement, the final test case given was not stable when working on specific setups, and my setup was one of them. Even though when doing the listed task manually, the app functioned well and did not introduce any new error, the test failed and broke the app. After some inspections, I found that the app checked for the score too fast after the screen had rotated, and broke the app, A solution I came up with was to delay the `check()` function, letting the screen rotate and load properly. The code I used is as below:

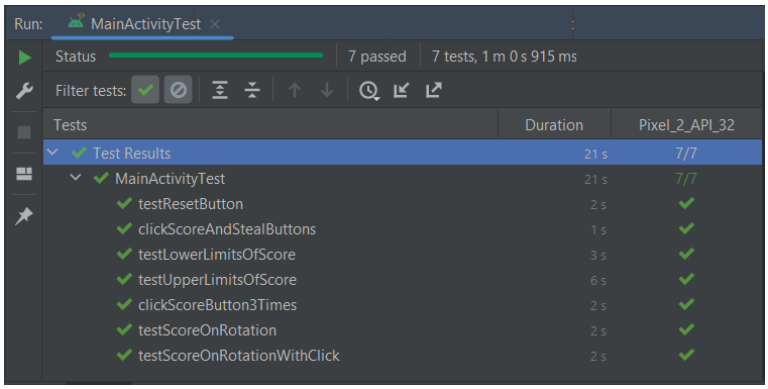
- Original code:

```
scoreButton.perform(click())
textView.check(matches(withText("4")))
```

- Adjusted code:

```
…
Import java.util.Timer
Import kotlin.concurrent.schedule
…
scoreButton.perform(click())
Timer("SettingUp", false).schedule(500) {
   textView.check(matches(withText("4")))
}
```

With the adjusted code, we use the Timer library to add a timer delay to the `check()` function and use the schedule library to set the time that we want the timer to delay, in this case, 500 milliseconds or 0.5 seconds. The test returned passed as the screenshot below.




```
activityRule.scenario.onActivity { activity ->
    activity.requestedOrientation =
ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE
}
```

Another way to prove this inspection was by setting the device in landscape mode at the beginning of the test. By doing this, when the unit test jumped to test 7, the device did not need to go through its lifecycle and reload the screen, resulting in a passed test.