Core Task 1 – Spike: Noises Off

Goals:

This section is an overview highlighting what the task is aiming to teach or upskill.

This app consists of three buttons. The buttons are sneeze, blow nose, and take medication. The initial health is set at 10. The sneeze button reduces the health by 1, take medication increments by 2, and blow nose plays a sound. The range of the health is kept between 1 and 10. This task aims to teach how to use lifecycle states, basic decision making, basic UI design, string externalization, and using Media Player functions for sound.

The following list outlines the goal broken down into more specific knowledge gaps involved in the goal.

- Media player (sound)
- String externalization (second language/locale)
- Layout (Linear/Constraint)
- Save Instance state for saving score
- Logs in IDE

Github Repo : https://github.com/Eric-Sow/cos30017-coretask1-Joel-103366239

Tools and Resources Used

This section lists related software, tools, libraries, API's, and other resources used for this knowledge gap.

- Android Studio
- Stackoverflow (Range between 1-10) - Set limits for increments and decrements in Android Studio (Kotlin) - Stack Overflow
- Google developers (Media player implementation) MediaPlayer overview | Android Developers
- Sound files Sound Library | Conversational Actions | Google Developers

Knowledge Gaps and Solutions

This section presents the listed knowledge gaps and their solutions with supporting images, screenshots and captions where appropriate/required.

Gap 1: Media Player implementation

The Media Player class is part of the Android multimedia framework to control audio or video files. It can play standalone files from the local storage or from streams over a network using a set of APIs provided by the Multimedia class. In this project, a basic implementation of the Multimedia class to play the sound. The sound file named as phone_alerts_and_rings.ogg.

This gap of knowledge is filled from the link attached above under Media player implementation. The link above provides a greater detail on how to use the Media Player for advance usages. By calling the Mediaplayer.create function, it automates the nitty-gritty of playing an audio file.

```
//Media Player
var MediaPlayer = MediaPlayer.create( context: this,R.raw.phone_alerts_and_rings);
```

This code uses the Media player function to play the file. The parameters is then stored in the MediaPlayer variable.

```
//Blow Nose event
BlowNose.setOnClickListener()
{  it: View!
    MediaPlayer.start()
    Log.i( tag: "Test sound",  msg: "Sound Played")
}
```

This code shows plays the audio file when the button is clicked. The log shows whether the button has been clicked in logcat window. This is more for debugging purposes.

Gap 2: String externalization (Second language/locale)

Localization of an application is important is gaining more users for the app in various regions. Without this feature, the app can only be used by one region who understand the application.

To implement localization of an app, Android recommends using its resource framework to separate localized versions from the core functionalities of the app and prevent confusion for the programmer also. The localized translations are stored in the res/values/strings folder.

Using this string externalization, the translations for the default strings are stored in an external strings.xml file. Therefore, the text of the components are not hardcoded as attributes of the components. The ID is used to link the component text to the appropriate word in the translations in the strings.xml.

```
<resources>
    <string name="app_name">Core 01</string>
    <string name="Sneeze">Sneeze</string>
    <string name="Blow_Nose">Blow Nose</string>
    <string name="Take_Medicine">Take Medicine</string>
</resources>
```

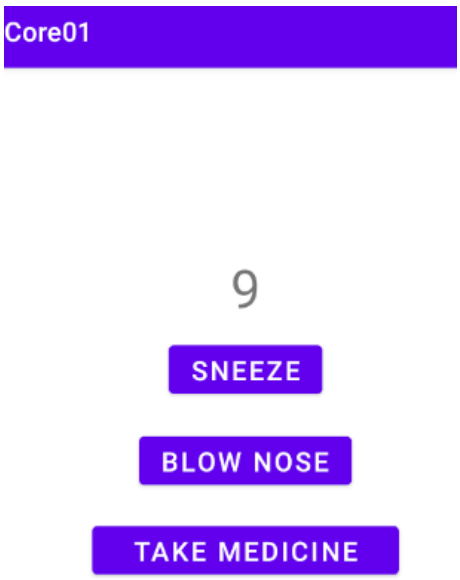
To enable localization when running the app, the language or keyboard has to be changed to that particular region. In this project, the language is changed to Malay (Malaysia) as countries like Singapore, Brunei, and Indonesia supports Malay also.

Whenever the app runs in a certain locale when the developers does not provide locale-specific translation, Android loads the default strings from the res/values/strings folder. If

this default file is missing, or if it's missing a string that your app needs, then the app does not run and display a certain error.

The app in Malay (Malaysia)

The app in English (default)



Gap 3: Layout (Linear/Constraint)

Layouts are an important part of any application, It forms the core of how the user experiences the app and whether they will continue using it. A good layout will ensure good experience but a bad layout will frustrate users.

As defined in Android Developers, a layout determines the structure for a user interface and how its components are arranged in the activity. A layout is composed both of a view and ViewGroup. A view determines something a user can see and interact with. A

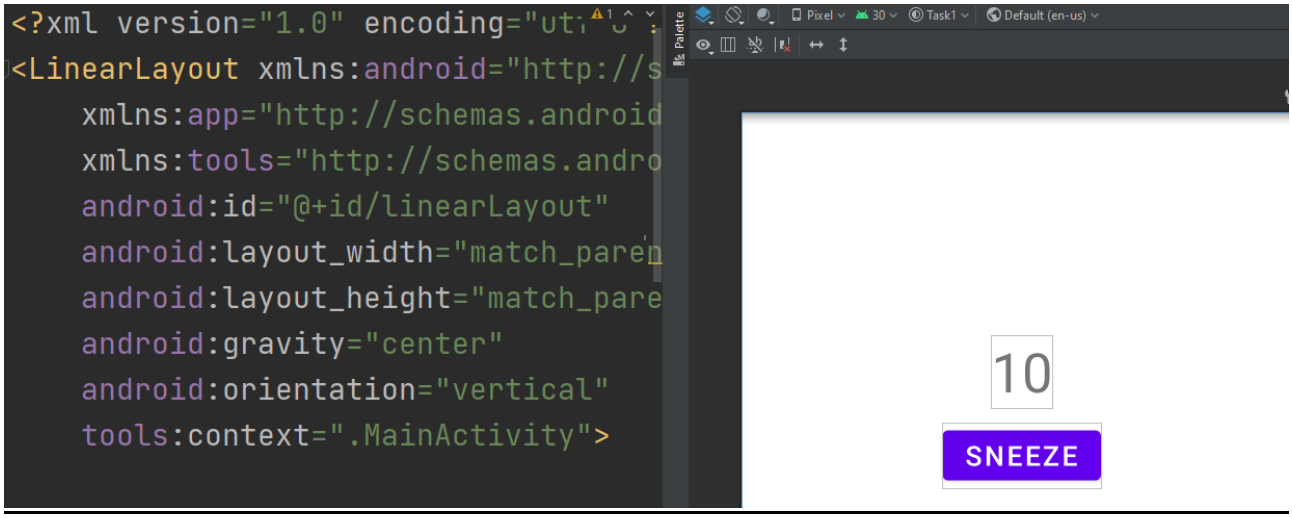
ViewGroup is an invisible container that defines the structure of both the View and ViewGroup.

Layouts in Android can be defined in XML format, use the inbuilt layout editor, or instantiated at runtime which means it is created and manipulated programmatically

In this project, the layout editor will be used for efficiency and the XML format is used to tweak certain settings to fit the project. Additionally, the projects requirements need a Linear and Constraint Layout to be used.

Portrait mode: Linear Layout

Linear layout is a view group that aligns all children in a single direction, vertically or horizontally. To modify the layout, the orientation(horizontal/vertical) attribute is manipulated with to suit the project. All child elements are stacked on top of each other. In addition, margins and gravity (left, right, or center) are enforced throughout all elements.

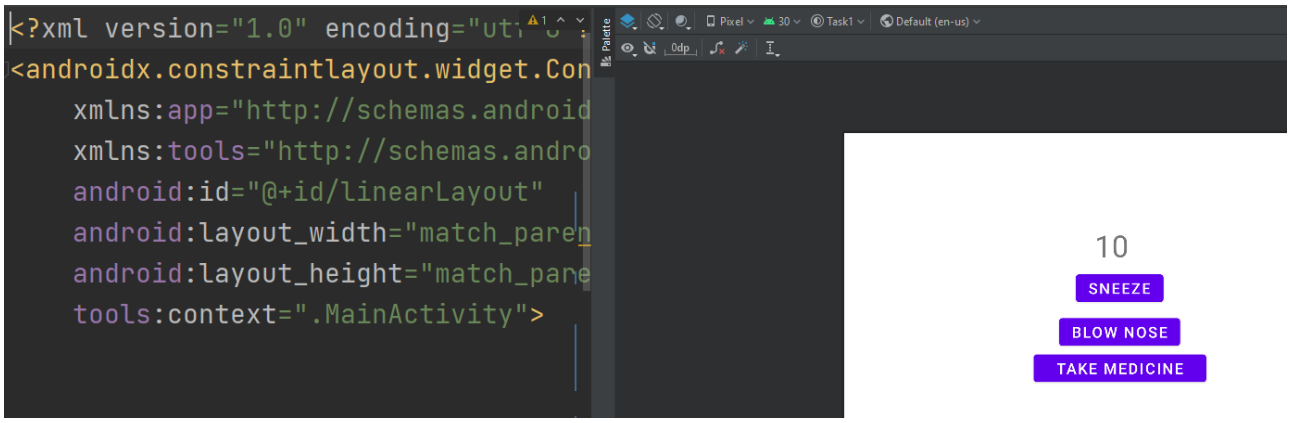


Landscape mode: Constraint Layout

Constraint layouts allows the developer to create various types of complex layouts based on a flat hierarchy. It is much more flexible than a linear layout by allowing child elements to be moved anywhere in the layout structure The layout structure can be easily manipulated via the layout editor. This is an advantage for the layout design if the developer does not want to modify XML code and prefers drag and drop.

In terms of constraints, the layout editor provides a “magic button” to automate the constraint settings of the layout structure. Constraints are needed so that the elements stay in their current position and not move out of position, for example, when rotation is

involved or the activity state changes. Without constraints, the elements may shift out of position.



Gap 4: Save instance state for saving score

In ensuring a good user experience, the activity's UI state must be preserved and restored when the system changes state. From the users part, they expect the UI state to remain same. Without saving the activity state, the current UI state will be destroyed and restored to default.

In this project, the changing activity state is rotation horizontally and the state that has to be saved is the health value. To save the instance state, the SaveInstanceState function will be used to handle all state changes. This reduces the developer time to write code to handle state changes.

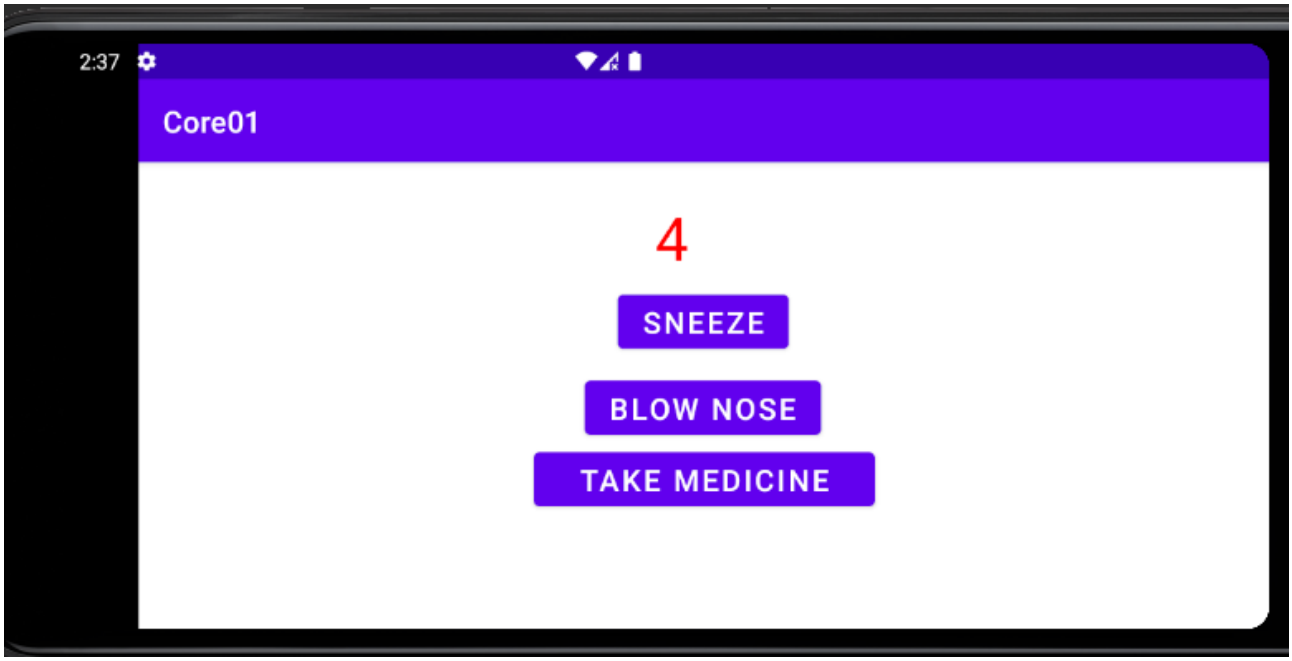
Two main code handlers consisting of decision making and a function will be used together with the savedInstanceState.

```
if(savedInstanceState != null)
    {
        InitialHealth = savedInstanceState.getInt( key: "HEALTH")
        textView.setText("$InitialHealth" )
    }
```

This code shows that if the saved instance state is not null or empty, it will store the state and then reproduce it in the changed state.

```
//Saving instance state
override fun onSaveInstanceState(outState: Bundle)
{ // Here You have to save count value
super.onSaveInstanceState(outState)
outState.putInt("HEALTH",InitialHealth)
Log.i( tag: "LifeCycle",  msg: "onSaveInstanceState")
}
```

This code shows that the saved instance state function that saves the state and shows the log that the instance has been saved.



This shows that the saved instance state is working as the health status is saved successfully.

Gap 5: Logs in IDE

Debugging and log activity is key in error/logic identification. In the debugging process, log activity can tell the developer whether UI components are working properly or whether there are issues. To view the logs, the logcat window is used.

Types of log activities

- Log.v() - verbose
- Log.d() - debug
- Log.i() - info
- Log.w() - warning
- Log.e() – error

In this project, the Log.i() function will be used because we need to test whether the UI components are working or not. Below are examples of the output from this functions.

```
2021-09-23 14:49:51.622 18952-18952/com.example.task1 I/Increment test: Increment by 2
```

```
2021-09-23 14:49:45.969 18952-18952/com.example.task1 I/Decrement test: Decrement by 1
```

This is the logs being captured when the components are clicked.

Open Issues and Recommendations

Currently, the app is running fine however, certain elements relating to layout types are difficult to implement especially linear where everything is constraint to row forms. I had to modify the attributes to fit the components well into the layout. For the media player, there were different implementation types based on the android versions, so it was difficult to implement at first. It took some time to find the implementation that worked for this current android version. The following issue is the emulator, it can be clunky at time like crashing sometimes while debugging code. For larger projects, an actual device should be used.