**Assignment 1**

**Local Climbing Score**

Github repository link: <https://github.com/SoftDevMobDev-2024-Classrooms/assignment01-TranVuDuc04>

**Introduction**

This is a simple app called Local Climbing Score; it counts climbs and scores process. Users can use the application in portrait mode or landscape mode in English or Spanish.

**Development plan**

1. Requirements Analysis: rules, how to play, zone-based score, language switching,

log demonstration

2. Design

* UI Layout: Score display, Hold display
* Buttons: Climb, Fall, Reset, Change Language
* ImageView
* App Logic: Score calculation, state management (orientation change or language switch)
* Change the score text color based on the current zone (Blue, Green, Red).

3. Implementation

* Create a basic UI layout (XML)
* Implement buttons with a zone-based colour change mechanism.
* Handle orientation changes and language switching.

4: Testing and Debugging:

* Test all functionalities on different devices and orientations.
* Debug.

5. Finalization

* User Testing: Get feedback from peers or users.
* Submission: Document a report.

6. Future Enhancements (if needed)

**Tools and Resources**

* StackOverflow
* Android Studio
* [ImageColorPicker](https://imagecolorpicker.com/)
* Android Developer
* Chatgpt

**Functionality and Instruction**

**A screenshot of a cartoon

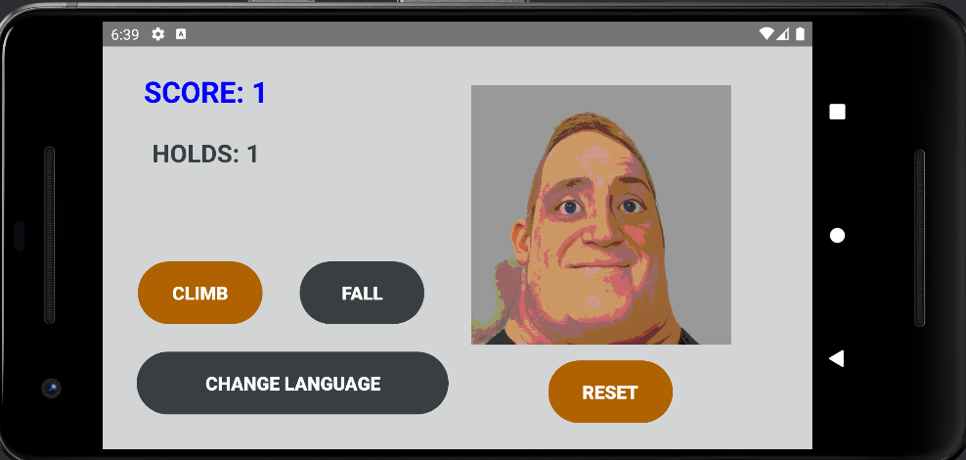
Description automatically generated**

There are 4 buttons in total: Climb, Fall, Reset, and Change language.

The initial score is set to 0. The score is capped between 0 and 18, with color changes based on the zone.

* The "Climb" button increases the score based on the hold's zone: 1 point (blue, holds 1-3), 2 points (green, holds 4-6), and 3 points (red, holds 7-9).
* The "Fall" button decreases the score by 3, but only if the climber has reached hold 1.
* "Reset" returns the score to 0.
* “Change language” changes the application from English to Spanish and the opposite.
* There is a picture indicating whether the user has won or fallen.

**UI/UX**



**Sketch and modify the view**

XML files form the foundation of mobile applications. However, it can be challenging to visualize them using only code, particularly for individuals with limited experience creating mobile apps. Hence, I use implemented plugins from Android Studio to assist the development process (Editor and Preview).

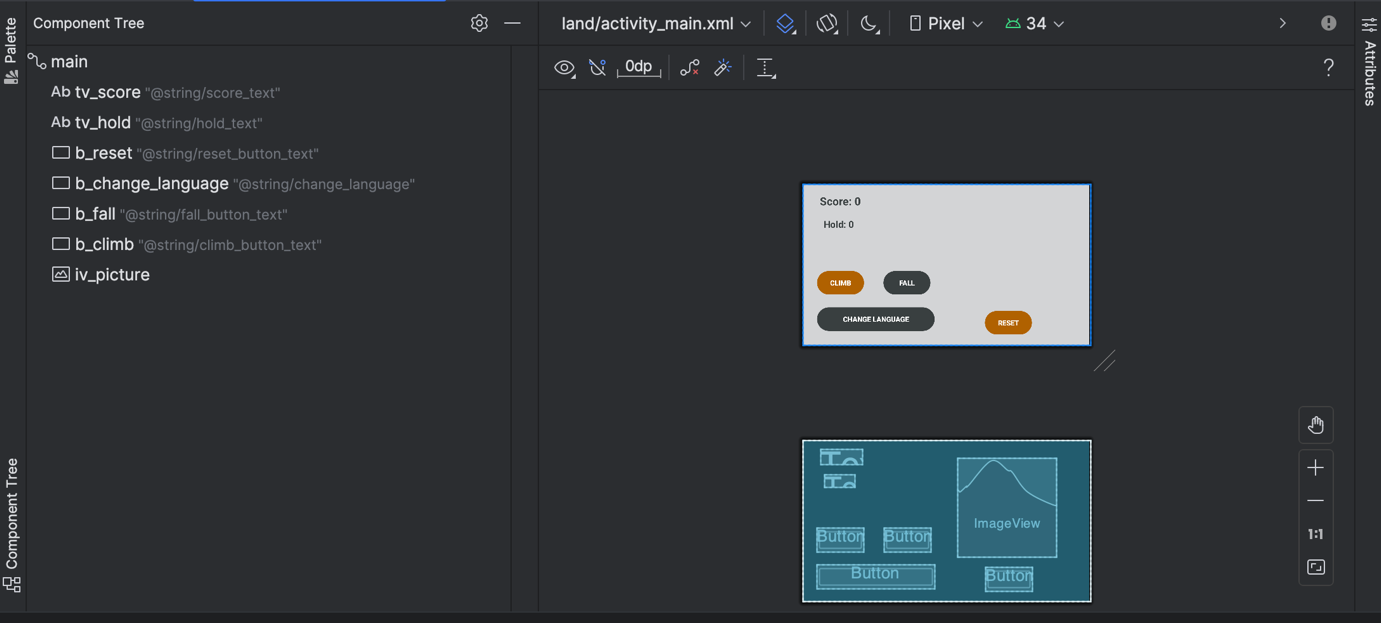
**Layout design**

For portrait mode, I used a LinearLayout, which is preferred due to its straightforward stacking mechanism. I combine relative layout as well, apply for 3 buttons (creating a similar effect to using a ConstraintLayout)

A screenshot of a computer

Description automatically generated

I add one more view for the horizontal version. With landscape mode, ConstraintLayout would be the most suitable.



To utilize the user’s experience, I use the ImageColorPicker website to find best match colour text and backgrounds, and make sure the contrast ratio doesn’t lower the user’s satisfaction.

**Key design**  
Language Support:

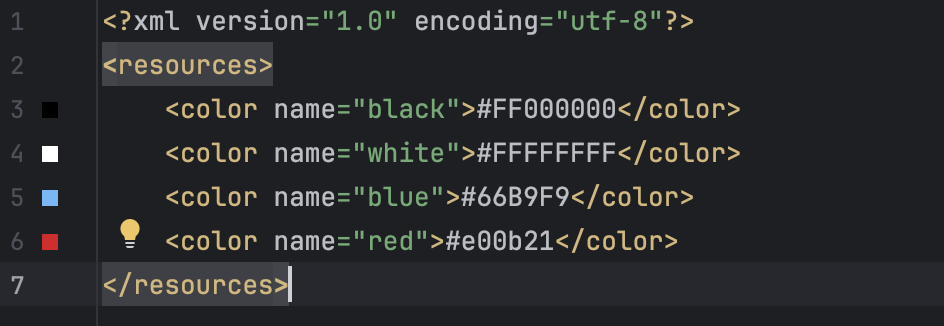
* Decision: Implement support for English and Spanish languages, allowing users to switch between languages during app usage.
* the app is accessible to a more audience.

State Management:

* Decision: Use onSaveInstanceState and onCreate to ensure that crucial variables like score, currentHold, and currentPictureIndex are restored after configuration changes.
* maintain consistent user experience

Dynamic UI Updates:

* Decision: Use dynamic string resources and color updates based on the current state (e.g., score, hold position). This also helps flexible display, not hardcoded.



* Provide visual feedback to users, making the app more interactive and intuitive.

**Issues and Solutions**

Logging is a crucial part that I have implanted into my MainAcitivy.kt files to test the functions of the application. Log command added in the listener of any button will print a log line under Logcat window. I will demonstrate further with the following issues.

1. Issue: Inconsistencies in the score and hold states during button interactions or after configuration changes.

* Solution: Implemented onSaveInstanceState to save and restore critical state variables automatically. (as suggested in the assignment description)
* Outcome: Ensured consistent state management across different activity.

1. Issue: When switching languages, the currentPictureIndex unexpectedly increments due to the activity being recreated.

* Solutions: I spend hours scrolling on StackOverFlow and ask ChatGPT for a suggestion as I cannot think it myself. I was suggested to keep track of an index of the picture through logcat for debug purposes and create a separated the logic for updating the picture and incrementing the index.

A screen shot of a computer program

Description automatically generated

A black background with white text

Description automatically generated

* Turn out, at first I increase plus 1 index in updatePicture(), hence, every time I recreate the application (including change the orientation, change language), the picture changes without “Climb” button.
* With the help of logging, I notice the problem. I simply remove index increment in updatePicture() and move it to climb. Therefore, an index is incremented only whenthe “Climb” button is clicked.A screenshot of a computer program

  Description automatically generated
* Outcome: The solution successfully maintained the correct picture display, preventing unexpected index increments.

**Reflection**

* Clear and consistent use of Kotlin-specific features( for example, lateinit for view initialization and safe type checks with when statements.)
* Well-documented with comments that clarify crucial functions, making the code easier to understand and maintain.
* The use of onSaveInstanceState was effective in maintaining app state during configuration changes, ensuring a smooth user experience.
* Updating UI: based on the current score and hold, providing required feedback to users.
* Logging: crucial in spotting and tackling the picture index issue.