Vu Duc Tran 104175614

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.

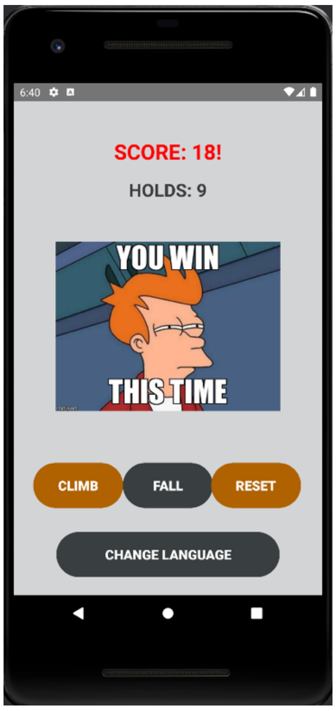
- Submission: Document a report.

6. Future Enhancements (if needed)

Tools and Resources

- StackOverflow
- Android Studio
- ImageColorPicker
- Android Developer
- Chatgpt

Functionality and Instruction

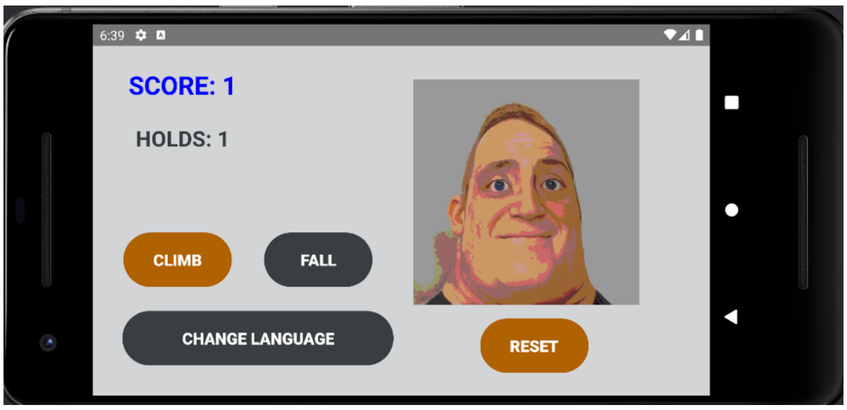


There are 4 buttons in total: Climb, Fall, Reset, and Change language.
The initial score is set to 0. The score is capped between 0 and 18, with color changes based on the zone.

- 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

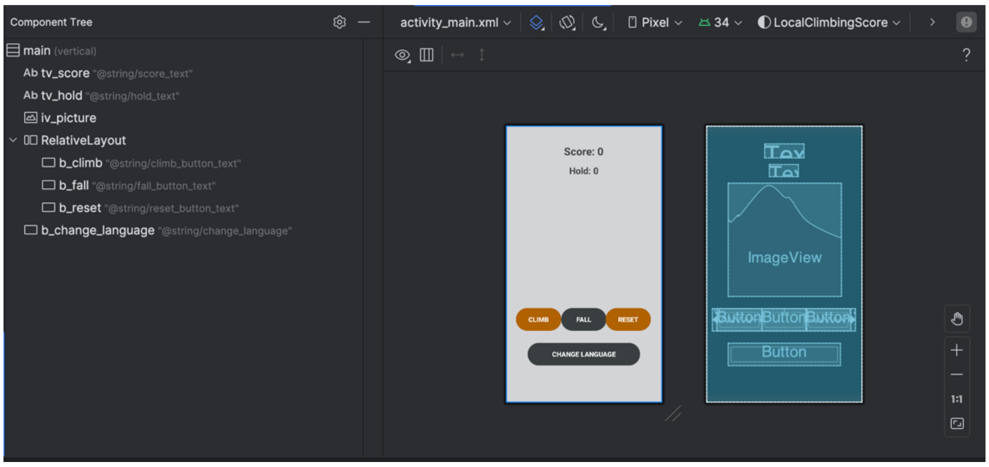


Sketch and modify the view

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).

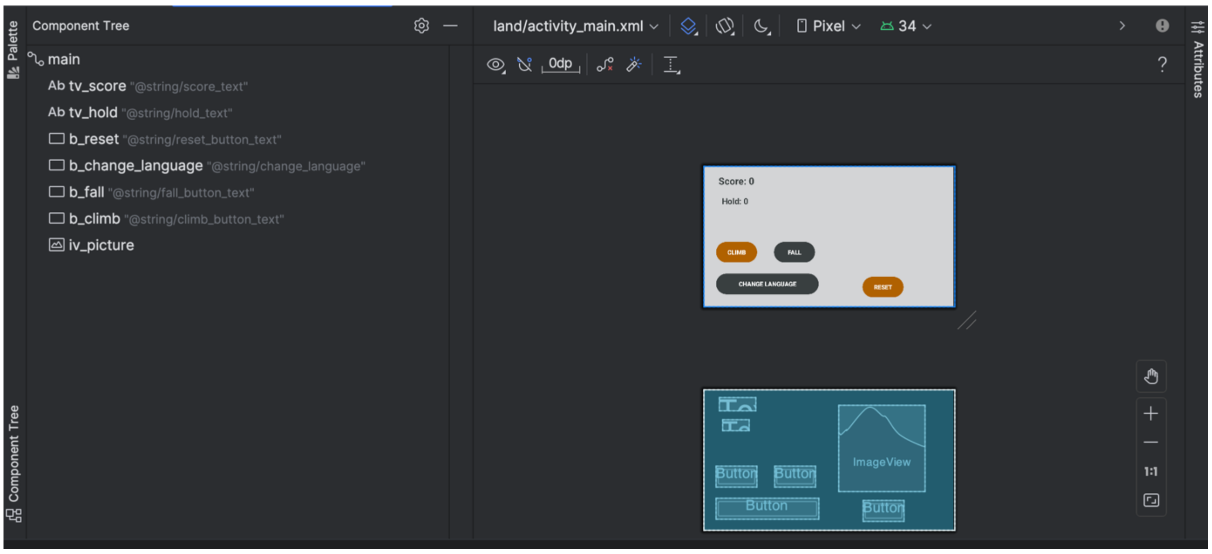
Layout design

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)



```
Component Tree
main (vertical)
  Ab tv_score "@string/score_text"
  Ab tv_hold "@string/hold_text"
  iv_picture
  RelativeLayout
    b_climb "@string/climb_button_text"
    b_fall "@string/fall_button_text"
    b_reset "@string/reset_button_text"
  b_change_language "@string/change_language"
```

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



```
main
  Ab tv_score "@string/score_text"
  Ab tv_hold "@string/hold_text"
  b_reset "@string/reset_button_text"
  b_change_language "@string/change_language"
  b_fall "@string/fall_button_text"
  b_climb "@string/climb_button_text"
  iv_picture
```

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:

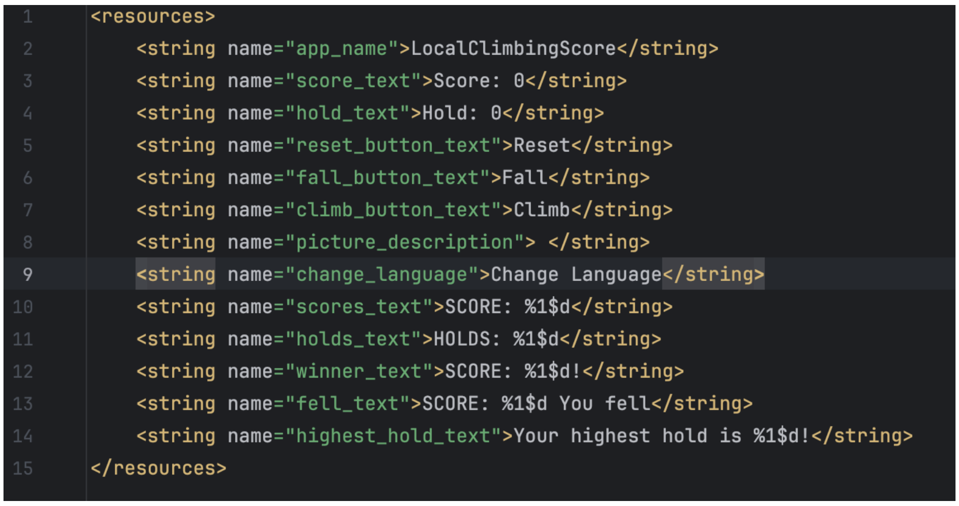
- Decision: Implement support for English and Spanish languages, allowing users to switch between languages during app usage.
  ⇨ the app is accessible to a more audience.

State Management:

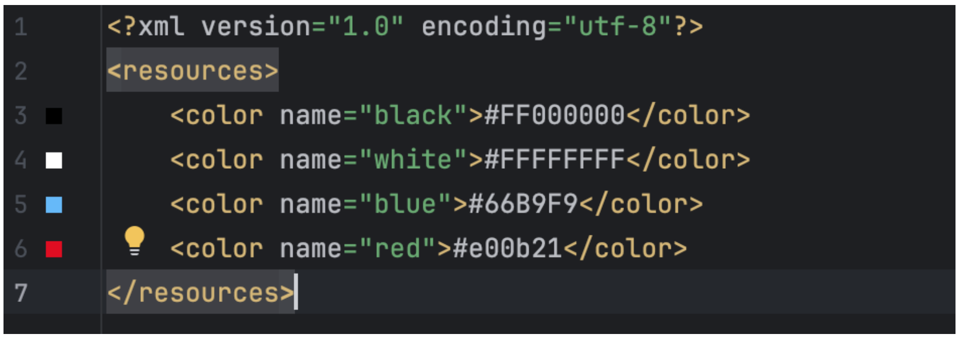
- Decision: Use onSaveInstanceState and onCreate to ensure that crucial variables like score, currentHold, and currentPictureIndex are restored after configuration changes.
  ⇨ maintain consistent user experience

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.



```xml
<resources>
    <string name="app_name">LocalClimbingScore</string>
    <string name="score_text">Score: 0</string>
    <string name="hold_text">Hold: 0</string>
    <string name="reset_button_text">Reset</string>
    <string name="fall_button_text">Fall</string>
    <string name="climb_button_text">Climb</string>
    <string name="picture_description"> </string>
    <string name="change_language">Change Language</string>
    <string name="scores_text">SCORE: %1$d</string>
    <string name="holds_text">HOLDS: %1$d</string>
    <string name="winner_text">SCORE: %1$d!</string>
    <string name="fell_text">SCORE: %1$d You fell</string>
    <string name="highest_hold_text">Your highest hold is %1$d!</string>
</resources>
```



```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="blue">#66B9F9</color>
    <color name="red">#e00b21</color>
</resources>
```

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

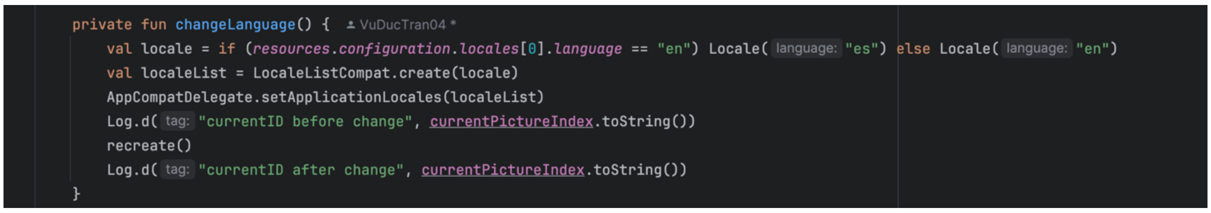
Issues and Solutions

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.

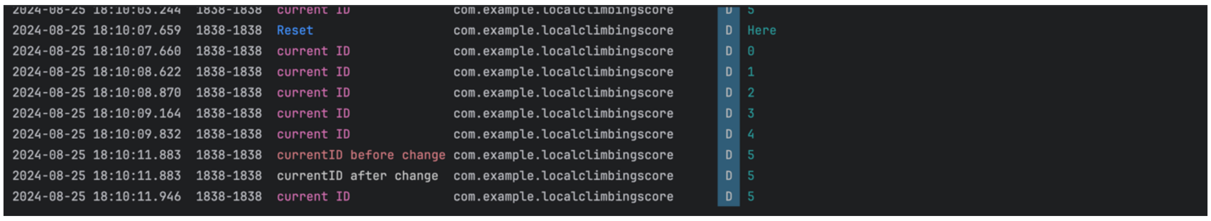
1. Issue: Inconsistencies in the score and hold states during button interactions or after configuration changes.
   - ⇨ Solution: Implemented onSaveInstanceState to save and restore critical state variables automatically. (as suggested in the assignment description)
   - ⇨ Outcome: Ensured consistent state management across different activity.

2. 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.

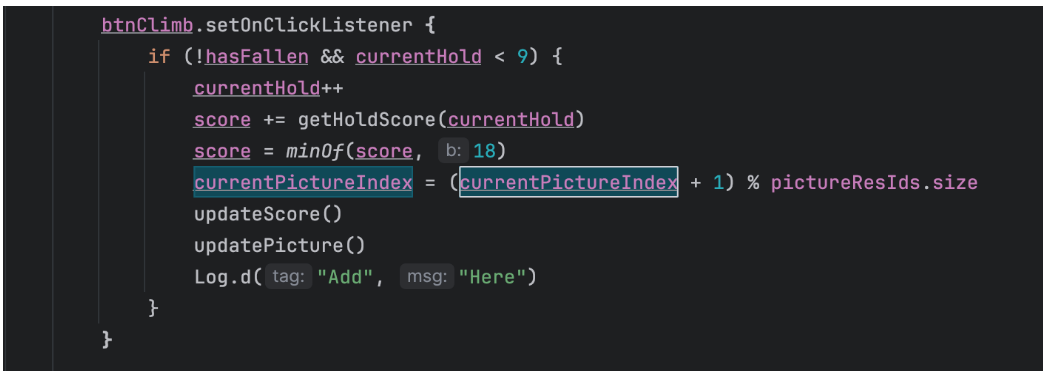


```
private fun changeLanguage() {  VuDucTran04 *
    val locale = if (resources.configuration.locales[0].language == "en") Locale( language: "es") else Locale( language: "en")
    val localeList = LocaleListCompat.create(locale)
    AppCompatDelegate.setApplicationLocales(localeList)
    Log.d( tag: "currentID before change", currentPictureIndex.toString())
    recreate()
    Log.d( tag: "currentID after change", currentPictureIndex.toString())
}
```



```
2024-08-25 18:10:03.244  1838-1838  current ID              com.example.localclimbingscore      D  5
2024-08-25 18:10:07.659  1838-1838  Reset                   com.example.localclimbingscore      D  Here
2024-08-25 18:10:07.660  1838-1838  current ID              com.example.localclimbingscore      D  0
2024-08-25 18:10:08.622  1838-1838  current ID              com.example.localclimbingscore      D  1
2024-08-25 18:10:08.870  1838-1838  current ID              com.example.localclimbingscore      D  2
2024-08-25 18:10:09.164  1838-1838  current ID              com.example.localclimbingscore      D  3
2024-08-25 18:10:09.832  1838-1838  current ID              com.example.localclimbingscore      D  4
2024-08-25 18:10:11.883  1838-1838  currentID before change com.example.localclimbingscore      D  5
2024-08-25 18:10:11.883  1838-1838  currentID after change  com.example.localclimbingscore      D  5
2024-08-25 18:10:11.946  1838-1838  current ID              com.example.localclimbingscore      D  5
```

- 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.



```
btnClimb.setOnClickListener {
    if (!hasFallen && currentHold < 9) {
        currentHold++
        score += getHoldScore(currentHold)
        score = minOf(score,  b: 18)
        currentPictureIndex = (currentPictureIndex + 1) % pictureResIds.size
        updateScore()
        updatePicture()
        Log.d( tag: "Add",  msg: "Here")
    }
}
```

⇨ 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.