I. Climbing Mountain App Report

A. *Abstract*

The "Climbing Mountain App" is an Android application that simulates a climbing experience by tracking user progress, managing falls, and supporting multilingual functionality. This report provides a detailed analysis of the application's features, technical implementation, UI/UX considerations, and potential improvements. Additionally, it aligns the project with key Unit Learning Outcomes (ULOs) to demonstrate its educational and technical relevance.

B. *Introduction*

The development of mobile applications presents unique challenges compared to traditional personal computing or web-based environments. Factors such as screen size, memory constraints, and processor limitations significantly influence design choices. The "Climbing Mountain App" is designed to optimize user interaction while considering these hardware-imposed restrictions. This report examines the application's design, implementation, and alignment with relevant Unit Learning Outcomes.

C. *Features*

1) *Climbing Mechanism*

- Users progress incrementally up to a maximum of nine holds.
- Score increments are based on hold levels:
  - Holds 1-3: +1 point per hold (Blue zone)
  - Holds 4-6: +2 points per hold (Green zone)
  - Holds 7-9: +3 points per hold (Red zone)
- Score display color changes dynamically to indicate the zone.

2) *Falling Mechanism*

- If a user falls before reaching hold 9:
  - 3 points are deducted.
  - The fall is recorded, preventing further climbing.
- Falling is not an option upon reaching hold 9.

3) *Reset Functionality*

- Resets climbing progress and score to zero.
- Allows users to restart the climb.

4) *Localization Support*

- Supports English, Japanese, and Vietnamese.
- Language selection updates the UI dynamically.

D. *Technical Implementation*

1) *MainActivity Class*

The `MainActivity` class extends `AppCompatActivity` and implements core functionalities, including UI interactions and state management.

2) *Instance Variables*

- `currentScore`: Tracks the user's score.
- `currentHold`: Tracks the hold level.
- `hasFallen`: Boolean flag to prevent additional actions after a fall.
- `scoreText`, `climbButton`, `fallButton`, `resetButton`: UI components.

3) *Lifecycle Methods*

- `onCreate(Bundle?)`
    - Initializes UI elements.
    - Restores previous state if available.
    - Configures button click listeners.
- `onSaveInstanceState(Bundle)`
    - Saves the current score, hold level, and fall status to persist through configuration changes.

4) *Event Handlers*

- `setLocale(String)`: Dynamically updates the app language.
- `initializeViews()`: Initializes UI components.
- `setupButtons()`: Defines button actions for climbing, falling, and resetting.
- `updateScoreDisplay()`: Updates UI elements based on user progress.

E. Unit Testing

The application includes a unit test to ensure the proper functionality of the climbing and falling mechanisms.

```
package com.example.climbingmoutainapp

import android.widget.TextView
import android.widget.Button
import org.junit.Test
import org.junit.runner.RunWith
import org.robolectric.Robolectric
import org.robolectric.RobolectricTestRunner
import org.robolectric.annotation.Config
import org.junit.Assert.*

@RunWith(RobolectricTestRunner::class)
@Config(sdk = [33])
class MainActivityTest {
    @Test
    fun testScoring() {
        val activity = Robolectric.buildActivity(MainActivity::class.java)
            .create()
            .resume()
            .get()
        val scoreText = activity.findViewById<TextView>(R.id.scoreText)
        val climbButton = activity.findViewById<Button>(R.id.climbButton)
        val fallButton = activity.findViewById<Button>(R.id.fallButton)

        // Log initial state
        println("Initial score: ${scoreText.text}")
        assertNotNull("Score TextView should not be null", scoreText)
        assertNotNull("Climb Button should not be null", climbButton)
        assertNotNull("Fall Button should not be null", fallButton)

        // Test initial state
        assertEquals("Initial score should be 0", "0", scoreText.text.toString())

        // Test climbing
        climbButton.performClick()
        println("Score after climb: ${scoreText.text}")
        assertEquals("Score should be 1 after climbing", "1",
scoreText.text.toString())

        // Test falling
        fallButton.performClick()
        println("Score after fall: ${scoreText.text}")
        assertEquals("Score should be 0 after falling", "0", scoreText.text.toString())
    }
}
```

F. Debugging and Logging

- Utilizes `Log.d(TAG, message)` to log key events such as climbing progress, falls, and resets.
- Enhances debugging efficiency by tracking user interactions.

G. UI/UX Considerations

- Color-coded scoring zones for intuitive visual feedback.
- Interactive buttons enhance user engagement.
- Language selection improves accessibility.
- Score persistence across device rotations.

H. Potential Improvements

1. **Persistent Storage**: Use SharedPreferences or a database to save user progress across app restarts.
2. **Animation Effects**: Implement smooth UI animations for climbing and falling actions.
3. **Leaderboard System**: Introduce a scoring system with global or local leaderboards.
4. **Sound Effects**: Add auditory feedback for climbing, falling, and button clicks.
5. **Expanded Language Support**: Broaden localization to additional languages.

I. Unit Learning Outcomes Addressed

1) ULO1: Mobile vs. Traditional Computing

The Climbing Mountain App is tailored for mobile devices, differentiating it from desktop or web-based applications. It leverages mobile-specific features such as touch input and screen size adaptability. Unlike desktop applications that support complex rendering, this app optimizes performance for mobile hardware, ensuring an intuitive and responsive user experience.

2) *ULO2: Design Considerations for Mobile Devices*

The app design considers mobile device constraints, including screen size, memory limitations, and processing power. The UI is simplified for touch interactions, with large buttons for ease of use. Efficient state management ensures minimal memory consumption, preserving user progress through configuration changes.

3) *ULO3: Development and Debugging Using Standard Libraries*

The application utilizes standard Android libraries such as `AppCompatActivity`, `Button`, `TextView`, and `Log`. The `onSaveInstanceState` method preserves user progress during screen rotations. Debugging is facilitated through `Log.d()` statements, which track climbing progress, falls, and resets, enhancing reliability.

J. Conclusion

The Climbing Mountain App effectively demonstrates core principles of mobile application development by considering hardware limitations, user interaction, and multilingual accessibility. The project aligns with mobile computing best practices and supports Unit Learning Outcomes through its design and implementation. Future enhancements, such as persistent storage, animations, and leaderboards, can further improve user engagement and functionality.

K. References

American Psychological Association. (2020). *Publication manual of the American Psychological Association* (7th ed.). APA Publishing.

Google Developers. (n.d.). *Managing activity state changes*. Retrieved from https://developer.android.com/guide/components/activities/state-changes

Google Developers. (n.d.). *Supporting different languages and cultures*. Retrieved from https://developer.android.com/guide/topics/resources/localization