Scramble Word Game - Android App Documentation

Overview

The **Scramble Word Game** is an interactive Android application that allows users to drag and drop scrambled letters into designated boxes to form words. Users are challenged to unscramble a set of random letters to form valid words from a predefined dictionary. The app includes feedback on correct or incorrect attempts and maintains a score for correct guesses.

Features

1. **Drag and Drop Mechanics**: Users can drag scrambled letters and drop them into boxes.
2. **Word Validation**: The app checks if the formed word is valid by referencing a dictionary stored in the app's assets.
3. **Random Word Generation**: Words are picked randomly from the dictionary, and the letters are shuffled for the player.
4. **Score Tracking**: A score is maintained and displayed based on how many correct words the user forms.
5. **Feedback System**: The app provides real-time feedback on whether the guessed word is correct or incorrect.

Project Structure

- **MainActivity.kt**: This is the main activity that handles the game logic, including drag and drop operations, word validation, and score updates.
- **activity_main.xml**: The layout file contains UI elements such as TextViews for the scrambled letters, EditTexts for the word boxes, a check button, and feedback and score displays.
- **dictionary.txt**: A text file located in the app's assets folder that contains a list of valid words for the game.

Code Walkthrough

MainActivity

Variables

- **scoreTextView**: Displays the user's current score.
- **feedbackTextView**: Provides feedback on whether the user formed a correct word.
- **checkButton**: A button that triggers the word validation logic when clicked.
- **wordSet**: A `Set<String>` that stores all valid words loaded from `dictionary.txt`.
- **currentWord**: Holds the current word that the user is trying to guess.
- **score**: An integer to track the player's score.

Methods

1. onCreate(Bundle?):
    - Initializes the UI elements and sets up the game.
    - Calls `loadDictionary()` to load the word list from `dictionary.txt`.
    - Randomly selects a word and scrambles its letters to display them.
    - Calls `setupCharacterDrag()` to handle the dragging of the scrambled letters.
    - Sets up drop targets using `setupDropTargets()` for the boxes where users drop characters.
2. loadDictionary():
    - Reads `dictionary.txt` from the assets folder and loads valid words into `wordSet`.
3. setupCharacterDrag(String):
    - Associates each scrambled letter in the TextViews with a drag event listener, allowing the user to drag letters.
4. setupDropTargets():
    - Attaches a drag listener to the 7 EditText boxes, enabling users to drop the characters.
5. handleDrop(View, DragEvent):
    - Handles the drop event when a user drags a character into a box and sets the dropped character into the target box.

6. checkWord():
    - Collects the characters from the EditText boxes, forms the guessed word, and checks if the word exists in the `wordSet`.
    - If the word is valid, it increments the score and displays a success message. Otherwise, it prompts the user to try again.

XML Layout (activity_main.xml)

1. 7 Drop Boxes (`EditText`):
    - Displayed in a horizontal row, each box is designed to hold one character of the guessed word.
    - The boxes are non-editable and act as drop targets for the scrambled letters.
2. 7 Scrambled Letters (`TextView`):
    - Displayed in a horizontal row below the drop boxes, these TextViews hold the scrambled characters and are draggable.
3. Check Button:
    - Allows the user to submit their guess after placing letters in the boxes.
4. Feedback TextView:
    - Provides feedback to the user about whether their guess is correct or not.
5. Score TextView:
    - Displays the current score based on the number of correct guesses.

Assets

- **dictionary.txt**: This file is located in the `assets` folder and contains a list of valid words that the app uses for word generation and validation. Each word is placed on a new line.

How to Play

1. **Start the Game**: The app displays 7 scrambled letters and 7 empty boxes.

2. **Drag and Drop Letters**: The player can drag letters from the scrambled list and drop them into the boxes to form a word.
3. **Check the Word**: After arranging the letters, the player clicks the “Check Word” button.
4. **Get Feedback**: The app checks if the word formed is correct. If so, the player's score increases by 1, and new scrambled letters are displayed. If not, the player is prompted to try again.
5. **Continue**: The player can continue playing, with the goal of forming as many correct words as possible.

UI Components

1. **TextViews (Scrambled Letters)**: These display the scrambled characters and allow dragging.
2. **EditTexts (Drop Boxes)**: Non-editable text fields that serve as drop targets for the scrambled characters.
3. **Button (Check Word)**: A button that triggers the word validation process.
4. **TextView (Feedback)**: Displays whether the guessed word is correct.
5. **TextView (Score)**: Displays the player's current score.

Future Improvements

1. **Multiple Word Lengths**: Currently, the game is set up for 7-letter words. Future versions could include different word lengths for added variety.
2. **Timer**: Adding a timer to increase the challenge by limiting the time players have to guess a word.
3. **Levels or Difficulty**: Introducing levels with increasing difficulty, such as more complex words or fewer hints.
4. **Sound and Animation**: Adding sound effects and animations for a more engaging user experience.
5. **Word Hint System**: A hint system where users can get a clue or the first letter of the word for a small score penalty.