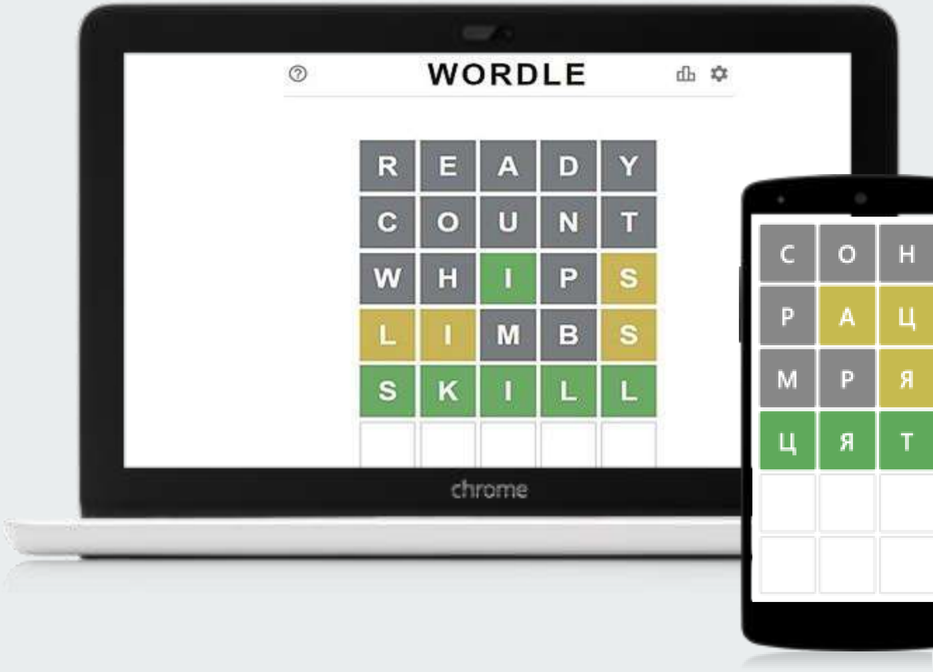
Wordle App Project

COP 4656

Rikki Villafranca, John Negrin, Gavin Antonacci



Abstract

The Wordle Game App is a mobile application that emulates the popular word-guessing game. The user is presented with a randomly selected five-letter word, and is required to guess it within six attempts. The application features a simple and intuitive interface, with a GridView displaying the user's previous guesses and the feedback associated with them. The user is even able to customize the application's appearance through settings that include a dark/light theme. The application also maintains the current game’s state along with past successful attempts and the user's preferences using data persistence mechanisms, ensuring a consistent experience across sessions.

Usability Test

i. Use of Prototype: The prototype used for testing included core functionalities such as guess validation, manual feedback display, and GridView updates. The aim was to test these features in a basic environment.

ii. Test Subjects: We used a diverse group of users, including both seasoned players and newcomers to the word-guessing genre. (Friends and Family)

iii. Modifications:

Input Validation: We improved the validation logic to handle edge cases more effectively. (We limited attempts, added a dictionary to validate attempts against.)

UI Refinements: Adjustments were made to improve the layout , visual appeal of the GridView and feedback display based on user preferences and feedback.

Activity Implementation

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    // Apply the selected theme
    applyTheme();
    super.onCreate(savedInstanceState);
    // Set the content view to activity_main.xml
    setContentView(R.layout.activity_main);
    // If no saved instance state, load GameFragment
    if (savedInstanceState == null) {
        getSupportFragmentManager().beginTransaction()
                .replace(R.id.fragment_container, new GameFragment())
                .commit();
    }
}
```

Activity Layout:

- **File:** Defined in activity_main.xml
- This layout defines a container for the fragment and other UI components such as Guess Input, Submit Guess button, and Previous Guesses GridView.

Activity Class:

- **File:** Implemented in MainActivity.java
- **Functionality:**
    - **Initialization:** Sets up UI and fragments.
    - **Theme Handling:** Manages the application theme based on user preferences in SharedPreference.
    - **Navigation Event Handling:** Manages events such as theme changes and menu button clicks.

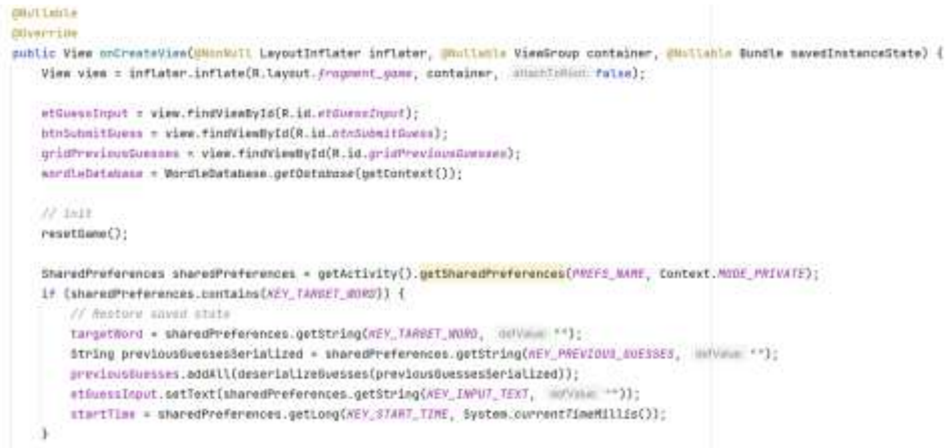
Initializes the main game screen and handles navigation between different sections of the app, such as settings and statistics.

Theme handling based on user preferences (light or dark mode).

Fragment Implementation

1. **Fragment Layout** - Defined in 'fragment_game.xml'
   a. This layout defines the UI elements that the fragment will contain
      i. Guess Input
      ii. Submit Guess
      iii. Previous Guesses
2. **Fragment Class** - Implemented in 'GameFragment.java'
   a. This handles the game logic and UI updates
      i. ValidateGuessTask
      ii. onClick

By adding the GameFragment to our app, we encapsulate the game logic and UI within the fragment, which allows for better separation of concerns and enhances the maintainability of our code. Meanwhile, the main activity handles navigation and settings.



```java
@Nullable
@Override
public View onCreateView(@NonNull LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_game, container, attachToRoot: false);

    etGuessInput = view.findViewById(R.id.etGuessInput);
    btnSubmitGuess = view.findViewById(R.id.btnSubmitGuess);
    gridPreviousGuesses = view.findViewById(R.id.gridPreviousGuesses);
    wordleDatabase = WordleDatabase.getDatabase(getContext());

    // init
    resetGame();

    SharedPreferences sharedPreferences = getActivity().getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    if (sharedPreferences.contains(KEY_TARGET_WORD)) {
        // Restore saved state
        targetWord = sharedPreferences.getString(KEY_TARGET_WORD, defValue: "");
        String previousGuessesSerialized = sharedPreferences.getString(KEY_PREVIOUS_GUESSES, defValue: "");
        previousGuesses.addAll(deserializeGuesses(previousGuessesSerialized));
        etGuessInput.setText(sharedPreferences.getString(KEY_INPUT_TEXT, defValue: ""));
        startTime = sharedPreferences.getLong(KEY_START_TIME, System.currentTimeMillis());
    }
```

Background Thread

The purpose of our background thread in **'GameFragment.java'** is to validate the user's guess asynchronously.

By using **'AsyncTask'** we validate the guess in the background while our **'ValidateGuessTask'** class performs the guess validation in the **'doInBackground'** method.

This method runs on a **background thread** that takes the user's guess, checks its length, and validates it against the dictionary, and constructs the result string based on the comparison with the target word.

After this the **'onPostExecute'** method updates the UI with the result and adds the guess to the **'previousGuesses'** list.

By implementing the background thread in this way, it ensures the UI remains responsive while performing the validation logic for each guess.

```java
import android.os.AsyncTask;

private class ValidateGuessTask extends AsyncTask<String,
Void, String[]> {
    @Override
    protected String[] doInBackground(String... params) {
        String userGuess = params[0];
        // Background task to validate
        return new String[] { "result" };
    }
    @Override
    protected void onPostExecute(String[] result) {
        // Updates the UI with the result
    }
}
```

Data Persistence

The data we persist in the app involves the preferences, current game state and the previous successful guess data. Our game state includes the previous guesses, the inputted guess, the start time and the word the user is guessing for.

To save our preferences and the game state we chose to just use the shared preferences method. We use this method in the **OnPause** and the **OnCreate** function. Also we specifically had to create our own serialization and deserialization methods to save and load the previous guesses, shown here.

For the previous successful guess data, we ended up using the **Room** database persistence method, similar to the one shown in the book to specifically save the word they correctly guessed, the time it took and amount of attempts.

```java
private String serializeGuesses(ArrayList<String> guesses) {
    StringBuilder serialized = new StringBuilder();
    for (String guess : guesses) {
        serialized.append(guess);
        serialized.append(",");
    }
    return serialized.toString();
}

private ArrayList<String> deserializeGuesses(String serialized) {
    ArrayList<String> guesses = new ArrayList<>();
    String[] parts = serialized.split(",");
    for (String part : parts) {
        if (!part.isEmpty()) {
            guesses.add(part);
        }
    }
    return guesses;
}
```

Input Data Validation

```java
private class ValidateGuessTask extends AsyncTask<String, Void, String[]>
{
    @Override
    protected String[] doInBackground(String... params) {
            String userGuess = params[0];
            String message = null;
            // Check if guess is a valid 5-letter word
             if (userGuess.length() != 5) {
                 message = "Please enter a 5-letter word.";
                 return null;
             } else if (!dict.isValidWord(userGuess)) {
                 message = "Invalid word.";
                 return null;
             }
    }
}
```

Input Data Validation:

- **File:** Defined in GameFragment.java
- It ensures user input meets the required criteria before processing.

Criteria:

- **Length Check:**
  - It ensures the input is exactly 5 characters long.
  - It displays an error message if the length is incorrect.
- **Character Check:**
  - It ensures the input contains only alphabetic characters.
  - It displays an error message if non-alphabetic characters are entered.
- **Dictionary Check:**
  - It ensures the input is a valid possible word contained in our dictionary. Our dictionary is not complete but it has about 2000+ plus words.

The validation both provides immediate feedback to the user and ensures only valid data is processed, maintaining the integrity of the game's logic.

Code Base Analysis

Percentage of Original Code: ~70% (~1,200/1,700 Lines)

Percentage of Copied Code: ~30% (DAO, Database, SettingsActivity) (~500 Lines)

References

1. Source 1: Setting Implementation - COP 4656: Mobile Programming 6.9
2. Source 2: Fragment Implementation - COP 4656: Mobile Programming 5.2
3. Source 3: DAO Implementation - COP 4656: Mobile Programming 6.5

Questions?