Songle Documentation

Stylianos Nicoletti s1516821

Contents

Introduction

This document illustrates the documentation for Songle. Everything mentioned in the design document was implemented and made available in the last version of Songle, with some improvements.

Bonus Features

As mentioned in the design document, some additional features were added to Songle game apart from the essential requirements. Songle game has many bonus features:

- User Login and Registration, which provides storage of progress for multiple users
- Different Map Styles, according to time of day (Day and Night Mode)
- Use of Hints, when player thinks that he cannot guess the song
- Use of Coins, to buy hints for each song
- YouTube Song Interaction, where the user can hear songs found
- Help Video, brief tutorial of how the game is played
- Statistical Information, including total kilometres walked, available coins, current score, current difficulty and number of songs found
- Background Music, on maps activity while the user catches words (Can be disabled)
- Accepting Similar Submitted Title to the Expected One, using Levenshtein distance, helping the user when missing tiny part of the title or making spelling mistakes

Activities Overview

Figure 1 – Splash Screen

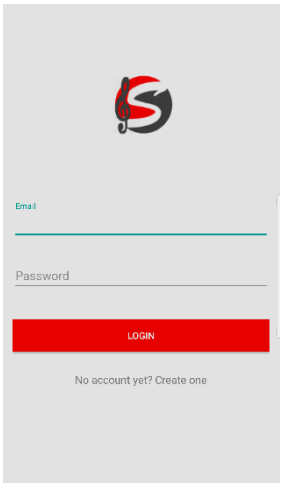

Figure 2 – Login Screen

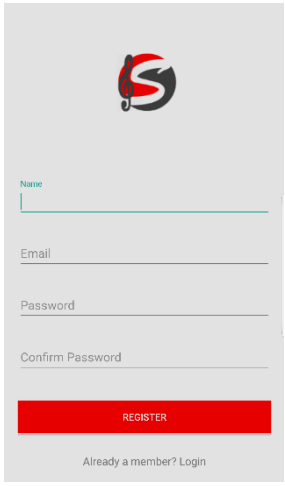

Figure 3 – Register Screen

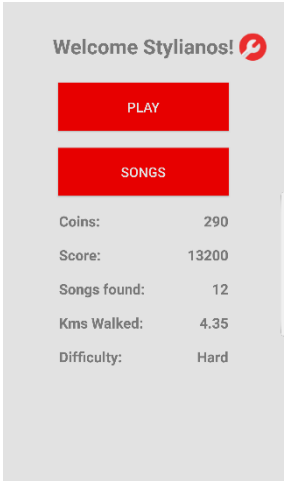

Figure 4 – Main Screen

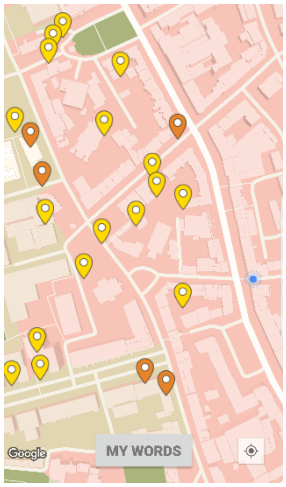

Figure 5 – Map Screen

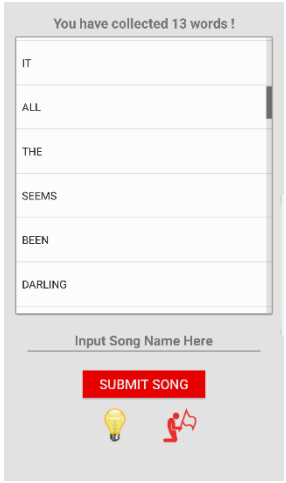

Figure 6 – Words Screen

Figure 7 – The End Screen Screen

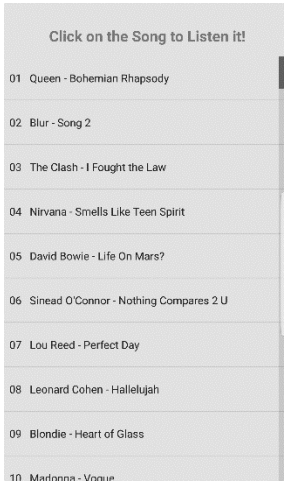

Figure 8 – Songs Screen

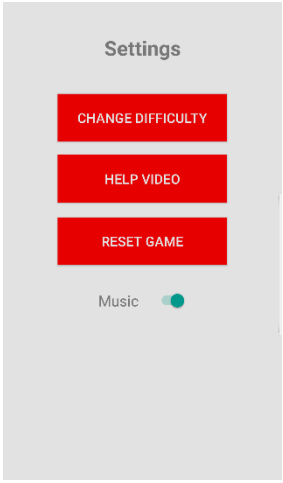

Figure 9 – Settings

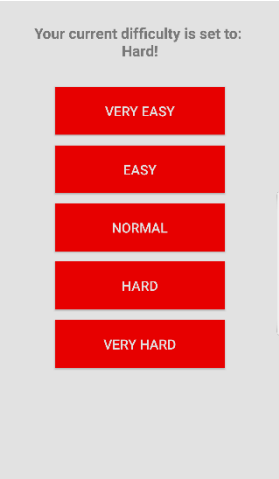

Figure 10 – Difficulty Screen

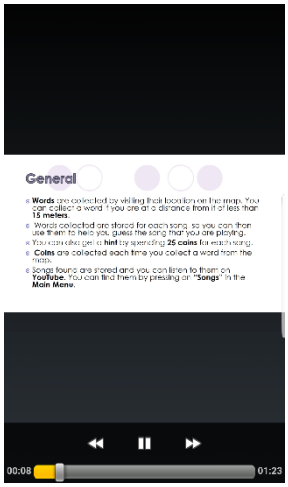

Figure 11 – Help Video Screen

Algorithms and Data Structures

Download and Parsing of the XML Songle Song List from the Server:

This was done by using an asynchronous task to download the XML file from the server. Then a custom-made XML Parser *(SongParsing.parseSongs)*, taking as argument the input stream of the file, was used to parse information needed from the XML into an array list of Song Objects. My custom parser makes use of *XMLPullParser* to get song's Number, Artist, Title and Link.

Download and Parsing of the TXT Lyrics List from the Server:

As mentioned above, an asynchronous task was used to download the TXT file for the current playing song's URL. Then a custom-made Parser *(LyricsParsing.parseLyrics)*, taking as argument the input stream of the TXT file, was used to parse Lyrics of the current playing song into a String. My custom parser makes use of *BufferedReader* and *StringBuilder* to get song's Lyrics and append the line by line into the String.

Download and Parsing of the KML Place Mark List from the Server:

Asynchronous task was used to download the KML file from server, like mentioned above for the other two downloads. Then a custom-made KML (XML like) Parser *(PlaceMarkParsing.parsePlaceMarks)*, taking as argument the input stream of the KML file, was used to parse place marks information needed into an array list of Placemark Objects. Like for Song's Parsing, this parser makes use of *XMLPullParser* to get placemark's Name, Description, StyleUrl and Coordinates.

Persistent Storage of the User's Progress

Since multiple users can play the game, a database system was used to store user's progress. A custom-made helper *(DatabaseHelper)* was used to get access or modify the database

when needed. My custom helper makes use of ***SQLiteDatabase***, ***SQLiteOpenHelper*** and ***database.Cursor***, where creating tables, modifying tables and querying tables is available. More thinks are mentioned about database used in Database System Section in the document below.

Detection of the Words Which Can Be Collected at the User's Current Location

A method available by Location ***location.DistanceTo()***, was used to return the approximate distance in meters between the two locations (Location of Word Marker, Location of User). Location of the user and word marker were set using ***location.setLatitude()*** and ***location.setLongitude()***. As of getting latitude and longitude of the word marker, a method available by Marker ***marker.getPosistion().latitude*** and ***marker.getPosistion().longitude*** were used. If word marker's location to user's current location is less than 25 meters, then the user can grab the word from the map.

Distance Walked

Using the method available by Location ***location.DistanceTo()***, as in the previously mentioned, I was able to measure the distance the user travels between its last location and the current location within a second time. If the distance, measured is less than 2.78 meters, meaning that the user travels at speeds up to 10km/h, the distance measured was added to the total distance travelled stored in the database. If the distance measured is more or equal to 2.78 meters, then the distance is not recorded. This was done to avoid the user driving or being a passenger and playing, that will be both dangerous and unfair.

String Similarity When Submitting Song Title Which is Similar to the Expected One

A custom-made method, using ***Levenshtein Distance***, is practised when needed to find the similarity between two Strings. Returning a double between 0 and 1 that represents how similar Strings are (1 returned for same Strings). That is a String of user's input and a String of the expected song title. Strings which are up to 66% similar to the expected one, are considered correct.

Database System

As mentioned above a SQLite database was used to store and access user's progress. The game's database contains a table called "user". This table includes:

- ***User_id (Integer Primary Key)***
  Id of the user in table that is automatically increment when new user added

- ***User_name (Text)***
  Name of the user as registered

- ***User_email (Text)***
  Email address of the user as registered, which can also uniquely identify the user since each user is forced to have an email address that is not already registered

- ***User_password (Text)***
  Password of the user as set in the register activity

- ***User_difficulty (Integer)***
  Current playing difficulty, which is a number between 1 to 5 (5 for Very Easy)

- ***User_score (Integer)***
  Current user score

- ***User_song (Integer)***
  Current playing song's number

- ***User_json (Text)***
  Words collected for current song are stored on JSON format, converted to String and added to user's table

- ***User_distance (Real)***
  Total distance walked

- ***User_coins (Integer)***
  Available coins

- ***User_music (Integer)***
  Whether background music is enabled (1) or disabled (0)

- ***User_hint (Integer)***
  Whether hint for current song was used (1) or not used (0)

  Every cell in the table can be accessed or modified using ***DatabaseHelper' s*** change or get methods. Change methods take as arguments the user's email address and the value to be changed. For example, if needed to change user' s current coins, ***changeCoins(String email, Integer coins)*** method was used. However, get methods take only as arguments user's email address. For example, if needed to get user's current score, ***getScoreByEmail(String email)*** method was used.

Parts Not Mentioned in Design Document

Everything mentioned in the design document was implemented and well tested. The only think that was not mentioned in the design document, was the media controller applied to Songle when the user wishes to watch the Help Video, as seen in Figure 12. By using the media controller, the user can skip some part of the video instead of watching it all from the beginning.

Apart from that, in the design document my plan was to collect markers within a range of 15 meters. I have changed the range to 25 meters.

Figure 12 – Video Media Controller

Testing

Automated Testing

Several automated tests were written using both Espresso for user interface testing and Junit for some method Testing. Some of them are:

(**ATTENTION**: In order to run some of the automated tests, Wi-Fi should be enabled, and also a manual click is needed when the device asks for Accessing Location Services, the first time you run Songle. Apart from that, tests should be run only once. If you want to rerun a specific test, you must first uninstall the app from the device.

- User that tries to register with the same email address as an already registered user
- Trying to buy a hint with no coins
- Changing difficulty
- Buying a hint for 25 coins
- Successful user login
- Successful registration
- Invalid email address when logging in
- User default settings when registering for first time
- Trying to login without inserting an email address or password
- Trying to login with an incorrect email or password
- Trying to view songs list when no songs found yet
- Resetting users progress
- Using the surrender option to skip a song
- Wrong song submission
- Levenshtein distance test with threshold 0.66

Physical Testing

Apart from automated testing, a beta version of the game was installed on three different devices of well-known persons and tested for functionality and how friendly is to the user. Anything that they did not like was updated in the last version. I also myself, tested Songle in both my personal phone and the emulator provided by Android Studio. Some of the tests I made, which were not automatically tested were:

- Collecting placemarks within 25 meters of my current location
- Trying to collect placemarks that were far away (more than 25 meters from my current location)
- Distance walked when both walking and running
- Distance walked when being passenger of a vehicle (should not record that distance as distance walked)
- Previewing songs from Songs Activity on YouTube
- Finding all the songs, ending the game
- View the help video

- Trying to play game with no Internet Connection (toasts were displayed informing me about that, as implemented)
- Trying to play game with no Location Services Enabled (toasts were displayed informing me about that, as implemented)

Acknowledgments

Login and Register Activities

Some part of the code implemented for Login and Register Activities come from:

https://sourcey.com/beautiful-android-login-and-signup-screens-with-material-design/

It was used as a template and reedited with my own features

String Similarity Class

Using the Levenshtein Distance in String Similarity Class, comes from:

https://stackoverflow.com/questions/36472793/levenshtein-distance-algorithm/

Version Control

Git was used for management of changes in my code for Songle application. It helped when needed to revert to earlier version of the code and backing up my whole project in case anything was lost from my local machine. Bitbucket repository was used.

My Bitbucket repository link for Songle is:

https://bitbucket.org/s1516821/songle

Conclusion

This documentation presented material regarding algorithms and data structures used for the implementation of Songle. Information concerning testing, acknowledgments and version control were also offered. Everything promised in the design document was implemented and verified on the final version of the game.