LANG, Chen
18M38127

[Prichan Mobile Card Book]
Final Report

1. Introduction and Background

"Kirato Prichan" is an anime television series that is currently broadcasting on TV Tokyo [1]. At the same time, there is also a music arcade game with the same name that is now operating in many Japanese game centers [2]. Its target customer is from primary school students to middle school students. However, the actual players are from every age group since the anime is so attractive and the arcade game reproduces many scenes of the anime.

Before entering the music game, the player needs to scan his "Pri-Ticket," which is a paper card with a printed QR code and other images, to equip the character he used with clothes, dress, shoes, accessories and other promotional items. After each play, the arcade game randomly generates several items in this play, and the player needs to choose one of them to print out and make a new Pri-Ticket with a QR code that represents this item.

In this game, different items have different names, rarities, scores, brands, colors and so on. To get a higher score and unlock more scenes in the game, most players play this game hundreds and thousands of times so that they could collect a large amount of Pri-Tickets. So, managing these Pri-Tickets well and avoiding choosing the same item to print out after each play is quite desirable for all the players.

By this project, I developed an Android application that aims to help the player to manage all Pri-Tickets that he possessed and check whether a duplicated item exists when the player chooses to make a new Pri-Ticket after each play.

2. Design Problems and Alternatives

By the Requirement Specification, there are five functional requirements for this project:

(R1) Display all items of a selected series from the local database;
(R2) Update the new item information from the official website: prichan.jp/items;
(R3) Mark or unmark an item, which represents the status of "possessing" or "not”;
(R4) Search an item with its ID, such as "PCH5-01";
(R5) Recognize an item based on the printed QR code by using the camera.

Except for the 4th functional requirement, it is necessary to make a design decision before implementing each of the rest functional requirements. The following sections will identify each design problem in detail, present the possible alternatives, discuss the feasibility of each option and make the corresponding design decision finally.

2.1 R1 - How to effectively handle the item data?

As what is described in the section "Introduction and Background," this Android application must store all the item data that are valid in the arcade game. Currently, the total amount of items is more than 1000, and this amount keeps growing weekly after a new series is released. So, it is required to find an appropriate way to handle these data.

To cope with this issue, two alternatives are presented: SQLite database or self-defined plain text file.

2.2 R2 - How to acquire the item data?

Since the data of all items are published on the official website of the arcade game company, a web page crawler is developed to extract and collect all these item data.

Then, it raises an issue that when should the web page crawler be used. To solve this issue, two alternatives are proposed:

(1) Embed this web crawler in the Android application package and use the crawler to collect item data once the official website is updated;
(2) Collect item data by using this web crawler externally and embed the item data in the Android application package.

2.3 R3 - How to effectively handle the status of the player’s possessed items?

The purpose of this project is to help the player to manage all his items that are printed on the Pri-Ticket, so the application must be able to effectively read and modify the status of the player's possessed items. Also, it is evident that the amount of the status data equals the amount of the item data, that means the amount of the status data is large.

Similar to the issue of how to effectively handle all item data, two options are presented to handle the status data: SQLite database or self-defined plain text file.

2.4 R5 - How to recognize the item from the QR code?

For this application, this is a functionality that is designed to help the user to input the data from a large number of new Pri-Tickets. The straightforward method is directly decoding the QR code and finding out the relationship between the data stored in the QR code and its corresponding item.

Additionally, by manually checking the QR codes from 100 Pri-Tickets, all patterns of the QR codes are same if the items that are represented by the QR codes are same, which means the QR code only contains the item identification information. This fact provides another solution that it is possible to construct a look-up table that stores a set of key-value pairs that maps the item data and its QR code pattern without decoding the information stored in the QR code.

3. Analysis and Design Decisions

3.1 R1 - Solution to effectively handling the item data.

It is important to point out that these item data are updated only if a new series is released. At the same time, updating only including appending new item data. Based on these two reasons, using a self-defined plain text file is more efficient than the SQLite database.

In addition, it is still necessary to load all the item data into memory once the application runs. In this scenario, loading all the item data from SQLite database or the self-defined plain text file uses the almost same amount of memory, whereas the cost of creating an SQLite database access object (DAO) is much more expensive than the cost of creating a file object [3].

Thus, compared with using the SQLite database, a self-defined plain text file is used to handle all the item data in the current version.

3.2 R2 - Solution to acquiring the item data.

As what is pointed out in section 2.1, the size of item data will be huge. Embed these item data in the Android application package can significantly increase the size of the installation package. So, embedding the entire item data can lower the deliverability of the application package.

Furthermore, based on the end user license agreement (EULA) of the website, any user (or player) only has the right to use these data personally but does not have the right to re-publish these data [4]. So, the application developer actually re-publishes these data to other users and violates the EULA if he embeds these item data in the application package.

On the other hand, crawling item data from the official website consumes a significant amount of time, especially in the first run. Besides, considering this project is an Android application, crawling may also increase the data usage of the cell phone network.

To deal with this emerging issue, a compromised improvement is proposed that a reminder windows will pop up when updating is required, and an updating window with a progress bar will be shown during web page crawling. Finally, by applying this improvement, embedding the web crawler in the application becomes feasible.

Thus, the method of embedding the web crawler in the application is adopted.

3.3 R3 - Solution to effectively handling the status of the player's possessed items.

It is necessary to realize that the constraints of solving this design problem are somehow different from the constraints of solving the design problem from R1.

In this issue, updating the status of an item tends to be much more frequently. Also, instead of sequentially appending new data, an updating operation is always randomly picking an item and modify its status. So, using a self-defined plain text file to store the status of all the items is inefficient.

Even though it is able to load all the item status to the memory and update them in the memory, when should the application write the new status back to the text file becomes a new problem. Furthermore, a tough situation is that the user or the Android system may kill the application at any time. If the text file is always updated once the data in the memory is updated, there will be too much writing so that the external memory will be overwhelmed. On the other hand, the application risks losing data if the text file is updated with a longer period.

Thus, towards this design problem, the SQLite database is used to handle the status of the player's possessed items.

3.4 R5 - Solution to recognizing the item from the QR code.

For sure, decoding the QR code and directly using the relationship between the data stored in the QR code and its corresponding item data is convenient. This method is an ideal solution since building a look-up table requires a lot of work to maintain this table once new items are released.

However, the main constraint that limits the leverage of the ideal solution is that the QR code printed on the Pri-Ticket is encrypted and no published documentation is available. This fact means directly decoding the QR code is impossible currently.

So, the solution of building a look-up table is used as the current solution.

Furthermore, storing all the patterns of the existed QR code leads to a large space complexity. So, the ZXing library is used to read out the encrypted QR code data, and then the SHA-1 hash function is used to convert the encrypted QR code data to a small-size fingerprint.

Finally, this functionality is intended to be implemented by using a look-up table whose key-value pair is defined as an SHA-1 hash value and a reference to its corresponding item data.

4. Conclusions

Based on the above analysis, the final design decisions for implementing the functional requirements R1, R2, R3 and R5 for the current version are:

(R1) Using a self-defined plain text file to store and manage the data of all items.
(R2) Embedding the web page crawler in the Android application package and crawl the official web pages once new items are published.
(R3) Using SQLite database to handle the variation of the status of the user possessed items.
(R5) Building a look-up table to do item recognition from the QR code printed on the Pri-Ticket.

In the future, the following improvements may be carried out:

(1) For the current design decision of R2, it is possible to embed some indexing files to help the web page crawler to avoid redundant and unnecessary exploration. These indexing files do not contain the item data nor have a large size, but they can reduce the running time of updating the local item database.

(2) For the current design decision of R5, it is still better to use the ideal solution that is directly decoding the QR code and find the relationship between the data stored in the QR code and the item data. However, this requires published documentation that explain the encrypting mechanism of the QR code.

5. Acknowledgement

I appreciate I can attend this course that gives me a try on android application development. I received no helps nor used any references other than what I mentioned in section 6 when I wrote this report.

By surveying my friends and some players, they expect to use this android application to help them manage their possessed Pri-Tickets and save their time. So, I am going to publish this application on Google Play store later.

6. References

[1] https://www.tv-tokyo.co.jp/anime/prichan/onair/
[2] https://en.wikipedia.org/wiki/Kiratto_Pri_Chan
[3] https://developer.android.com/training/data-storage/room/
[4] https://www.takaratomy-arts.co.jp/sitepolicy.html