Final Project Report

for

ToDoRPG

University of Illinois at Urbana Champaign
CS429: Software Engineering II
Prof. Darko Marinov
Spring 2014

Authors:
Kevin Chen (kwchen3@illinois.edu)
Leon Chen (lchen59@illinois.edu)
Jun (Johnny) Cheng (jcheng26@illinois.edu)
Paul Lim (paulkim6@illinois.edu)
Hyeon Seok Lim (hlim10@illinois.edu)
Seong Wan Song (ssong25@illinois.edu)

Table of Contents

1. Project Description

There are numerous mobiles applications designed to help users organize their lives. These applications usually engage their users by sending reminders or by setting off alarms before tasks need to be completed. Instead of nagging users to complete their tasks, our application will give users incentives to complete their tasks by turning everyday tasks into a game. Users will get rewards for completing their tasks, which can be used to buy items for their characters or to buy rewards for themselves. Items are in-game tools that players use to defeat other players in player-vs-player battles, while rewards are real life items that users can buy to treat themselves.

2. Development Process

2.1. Iterative Development

For this project, we followed a cycle of planning, designing, developing, testing, and polishing. The length of each cycle was two weeks, which coincided with the length of each of the iterations. At the beginning of each cycle, the group met up to discuss the plan for the current cycle. At this meeting, we would discuss which components we wanted to add application and split into pairs we would be working in for the cycle. For the first week, each pair would plan out and design the components that they wanted to work on. During the second week of the cycle, the pairs would develop and test the components that they were working on. At the end of the second week, we would meet with a TA for the course, present the components we had designed and listen to his suggestions. After our meeting with the course TA, we would polish our application using his suggestions.

2.2. Refactoring

Our refactoring usually occurred between cycles. At the end of every cycle, while we were polishing our application, we would make changes to our code to make development easier during the next cycle. Most of our refactoring involved adding comments or changing our code to make it easier for the other group members to understand. However, we made two major refactoring changes over the course of the project. The first major change involved splitting up our code into the MVC framework, grouping code based on their category. The second major refactoring involved moving data objects into the application class so that they could be considered persistant storage throughout the lifecycle of the application. These two refactorings involved almost every single file in the code and made it easier for us to find and add components to the code.

2.3. Testing

We used mostly unit testing for our code. To do this, we relied on Android's built in unit testing framework and on a 3rd party library, Robotium. Android's built in unit testing framework is almost identical to the JUnit framework, which allowed us to easily write tests for our code. However, Android's unit testing framework only allowed us to test the backend parts of our code, and not the UI. To test our UI, we used Robotium, which allowed us to perform more comprehensive tests. For examble, Android's unit testing framework had no way of testing if an element was on the screen, but Robotium did. Robotium also allowed us to simulate user actions, like pressing on buttons or selecting items in a menu. By using a combination of Android's unit testing framework and Robotium, we were able to test both our backend functions and our UI functionality.

2.4. Collaborative Development

One of the major challenges of working in a large group is to make sure that pairs were not working on similar components and to make sure that there were no conflicts while merging code. To accomplish this, we used Trello to manage our project, Git as our version control, and Github as our repository. Trello allowed us to post our progress, so that each of the pairs were aware of what the others were doing and what their progress was. During our code development, each pair worked on their own individual code branch, and then merged with a master branch when they were done. By using Trello to coordinate between pairs, there were relatively few merge conflicts while merging code.

3. Requirements and Specifications

3.1. Todos and Dailies

3.1.1 User Stories

1. As a user, I want to be able to add tasks to my to-do list
2. As a user, I want to be able to add tasks that I repeat on a daily basis

3.1.2 Use Cases

1. Create ToDo task

    Primary Actor: User
    Goal in context: Add new tasks to the to-do list.
    Scope: An RPG Android application
    Level: subfunction
    Stakeholders and interests:
        User: wants a list of tasks that need to be finished
    Precondition: none
    Minimal guarantees: Created tasks will be visible on the User's profile
    Triggers: Users creates a new task in application
        1. User creates a new task in the application

2. User enters relevant information for task
3. Task details are put in database
4. User sees new task in profile

Extensions

3a. Task details fail to be uploaded in database
4a. User sees a failure message
2b. User does not put enough information for the task
3b. User sees error message. Task is not put in database
4b. User sent back to step 2. User does not see new task in database

2. Create Daily Task

Primary Actor: User
Goal in context: Add new tasks to the daily list.
Scope: An RPG Android application
Level: subfunction
Stakeholders and interests:
User: wants a list of tasks that need to be finished every day
Precondition: none
Minimal guarantees: Created daily tasks will be visible on the User's profile
Triggers: Users creates a new daily task in application
1. User creates a new daily task in the application
2. User enters relevant information for task
3. Task details are put in database
4. User sees new task in profile

Extensions

3a. Task details fail to be uploaded in database
4a. User sees a failure message
2b. User does not put enough information for the task
3b. User sees error message. Task is not put in database
4b. User sent back to step 2. User does not see new task in database

3.2. RPG Characters

3.2.1 User Stories

1. As a user, I want an in-game character that levels up as I gain experience and that I can customize with items

3.2.2 Use Cases

1. Have a RPG Character

Primary Actor: User
Goal in context: Create a user profile and stat spread to begin playing the game.
Scope: An RPG Android Application

Level: subfunction
Stakeholders and interest:
User: wants a character that levels up as they complete tasks and that they can use to battle other users
Precondition: none
Minimal guarantees: User will see icons that represent what their character has equipped
Triggers: User open application
1. User opens app
5. User sees their character with equipped items
Extensions: none

3.3. Battle System

3.3.1 User Stories

1. As a user, I want to be able to battle with my friends' RPG characters using bluetooth.

3.3.2 Use Cases

1. Battle another Users' Character

Primary Actor: User
Goal in context: Add new tasks to the daily list.
Scope: An RPG Android application
Level: subfunction
Stakeholders and interests:
User: wants to battle their friends' character
Precondition: User has already paired bluetooth devices with their friend
Minimal guarantees: User will be able to battle their friends
Triggers: User sends a battle request to their friend
1. User sends a battle request to their friend
2. Other User accepts battle request
3. Users battle with their characters
Extensions
3a. Bluetooth connection fail
4a. Users return to battle screen
2b. Other user rejects battle request
3b. User returns to battle screen
1c. Bluetooth fails to send request
2c. User sees error message

3.4. Items and Rewards

3.4.1 User Stories

1. As a user, I want to be able to buy items for my in-game character.
2. As a user I want to be able to set and buy rewards for myself.

3.4.2 Use Cases

2. Buy items for RPG characters

Primary Actor: User
Goal in context: Buy Item and equip for RPG characters
Scope: An RPG Android application
Level: subfunction
Stakeholders and interests:
    User: User wants to be able to buy items for RPG characters
Precondition: User has enough gold to pay for the items
Minimal guarantees: User can equip items that they bought
Triggers: Users buys item from store
    1. Users buys item from store
    2. User Equips item on inventory screen
Extensions
    1a. User does not have enough gold
    4a. User sees an error message
    2b. User already has an item in the slot
    3b. Current item is unequipped
    4b. New item is equipped

2. Set and Buy Rewards

Primary Actor: User
Goal in context: Set and buy rewards for self
Scope: An RPG Android application
Level: subfunction
Stakeholders and interests:
    User: wants to be able to reward themselves
Precondition: User has already created rewards and has enough gold to pay for the reward
Minimal guarantees: User can create and buy rewards for self
Triggers: Users creates a new reward for themselves
    1. Users creates a new reward for themselves
    2. User enters relevant information for reward
    3. User buys reward
Extensions
    2a. User does not put enough information for the task
    3a. User sees error message. Task is not put in database
    4a. User sent back to step 2. User does not see new task in database
    3b. User does not have enough gold for reward

4b. User sees error message

3.5. Player Stats and Log

3.5.1 User Stories

1. User wants to see a record of everything they've done

3.5.2 Use Cases

1. See a record of everything user has done
   Primary Actor: User
   Goal in context: User wants to see a record of all their actions
   Scope: An RPG Android application
   Level: subfunction
   Stakeholders and interests:
   User: wants to be able to remember all their actions
   Precondition: none
   Minimal guarantees: User can quantify their actions and see a log of their actions
   Triggers: Users performs an action
   1. Action is logged
   Extensions: none

4. Architecture and Designs

4.1. Architecture

4.1.1 UML Diagrams

Class Diagram
Use Case Diagram

4.2. Design

For this project, we used the MVC framework during development.

4.2.1 Model

For the model, we used Java classes to represent the data after it is pulled from the database. After data is pulled from the database, it is stored in Java objects, which are stored in the GameApplication class, which extends the Application class. The Game Application class is created when the application first starts up and is accesssible by all of the other activities in the application. This class is static in the context of the application, so data is consistant throughout the application. For our application, all we had to do was design the layout for the lists and the rows of the list. All of the data classes for the model are in the model folder.

4.2.2 View

For the View, we mainly used Android's ListViews. Most of the application consists of presenting data in the form of lists, so the way we presented our data needed to be consistent throughout the application and needed to be done so in a way that was aesthetically pleasing to the user. Android's ListView allowed dynamically displayed our data in the form of a list, so we did not need to hard code any of the lists in our application. All of the Activities that use the ListViews are in the view folder.

4.2.3 Controller

For the controller, we created subclasses of Android's BaseAdapter class. Our subclasses were used to bridge together the data classes we used as models and the ListViews we used as views. In each of the BaseAdapter subclasses, all we had to do was specify the data source, and then specify which fields in the list correspond to the different parts of the data. All of the classes that subclass the BaseAdapter class are in the controller folder.

4.2.4 Influence on Design

Using the MVC framework allowed us to easily add and modify components for the application. After finishing the first component, our progress increased substantially as we were able to reuse most of the code for subsequent components. Using the MVC framework also had the unexpected benefit of making the code easier to understand. Although two of the six members of our team made the initial components, because they used the MVC framework, the other group members were able to easily add components with minimal help, just by replicating the existing code.

5. Future Plans

After the semester is over, the logical next step would be to continue polishing the app and then release it on the Google Play store. However, for most of the group, this was the first Android application they have ever developed. As a result we are not that confident that the application would be well received on the appstore. Until, we are more confident in our abilities, our code will be open source on Github.