Supplementary Specification
For

Pokémon Go Back

Prepared by:

Name	ID
Amitt Bhardwj	40037899
Gaganjot Kaur	40037933
Gunpreet Ahuja	40041311
Gurpreet Kaur Lotey	40038315
Gurpreet Raju	40013682
Harinder Kaur	40041313
Mandeep Singh	40040964
Navjot Singh	40014477
Navrang Pal Kaur Dhaliwal	40047555
Simranjit Singh Bachher	40040601

Instructor name: Stuart Thiel
Course name: Software engineering processes
Date: 24th May 2017

Table of Contents

1. Introduction

The introduction of the **Supplementary Specification** should provide an overview of the entire document. It should include the purpose, scope, definitions, acronyms, abbreviations, references, and overview of this **Supplementary Specification**.

The **Supplementary Specification** captures the system requirements that are not readily captured in the use cases of the use-case model. Such requirements include:

· Legal and regulatory requirements, including application standards.

· Quality attributes of the system to be built, including usability, reliability, performance, and supportability requirements.

· Other requirements such as operating systems and environments, compatibility requirements, and design constraints.

1.1. Purpose

This document includes the detailed functionalities of the system. Majorly this document focuses on the non-functional requirements of the system and other constraints like hardware, software, and user interface. It also includes details about the applicable standards, license and other copyright issues.

1.2. Scope

This document is related to the project which is named as PokemonGoBack. It helps everyone whosoever related to this project to get detailed functional requirements as well as information about the non-functional requirements, constraints and various other issues that are not part of the software requirement document.

1.3. Definitions, Acronyms and Abbreviations

PGB - Pokémon Go Back

COCOMO - Constructive Cost Model

GUI - Graphical User Interface

KLOC - 1,000 Lines of Code

SRS - Software Requirements Specification

Http - Hypertext transfer protocol

AI- Artificial Intelligence

1.4. Overview

This subsection should describe what the rest of the **Supplementary Specification** contains and explain how the document is organized.

It contains non-functional requirements, detailed functional requirements, various standards and license requirements.

2. Functionality

This section describes the functional requirements of the system for those requirements which are expressed in the natural language style. For many applications, this may constitute the bulk of the SRS Package and thought should be given to the organization of this section. This section is typically organized by feature, but alternative organization methods, for example organization by user or organization by subsystem, may also be appropriate. Functional requirements may include feature sets, capabilities, and security.

Where application development tools, such as requirements tools, modelling tools, etc., are employed to capture the functionality, this section document will refer to the availability of that data, indicating the location and name of the tool used to capture the data.

2.1. Details of Requirement No: 7. which is specified in Software Requirement Document

Each card type should have further sub categories.

--- For each type, there are categories available

1. type**:pokemon**

(a) cat:basic

(b) cat:stage-one

2. type:**energy**

(a) cat:colorless

(b) cat:water

(c) cat:lightning

(d) cat:psychic

(e) cat:fighting

3. type:**trainer**

(a) cat:Stadium

(b) cat:Supporter

(c) cat:Item

2.2 Details of Requirement No: 8. which is specified in Software Requirement Document

Abilities target various elements in the game, they are listed below for clarification.

1. targeting pokemon

(a) target:your-active

(b) target:opponent-active

(c) target:choice:opponent

(d) target:choice:your

(e) target:choice:opponent bench

(f) target:choice:your-bench

2. targeting players

(a) target:you

(b) target:them

The abilities themselves are:

1. null

• do nothing

2. dam

• dam:[target]:[amount]

• damages a pokemon

3. heal

• heal:[target]:[amount]

• heals a pokemon

4. deenergize

• deenergize:[target]:[amount]

• owner of target removes [amount] energy from that pokemon

5. reenergize

• reenergize:[target:source]:[target:destination]:[amount]

• move energy from one pokemon to another, [amount] times

6. re-damage

• redamage:[target:source]:[target:destination]:[amount]

• move damage from one pokemon to another, [amount] times, allowing multiple destinations at once.

7. swap

• swap:[target:source]:[target:destination]

• swap one pokemon from source with one pokemon from target. swapping an active pokemon to the bench removes all status effects.

8. destat

• destat:[target]

• removes all status effects from the target pokemon

9. applystat

• applystat:[status]:[target]

• the targeted pokemon gets the specified status

10. draw

• draw:[amount]

• draw [amount] cards

11. search

• search[target]:[source]:[filter]:[amount]

• source may be either source:deck or source:discard

• filter is a combination of type and category, e.g. type:energy—cat:water would allow searching for any water energy

• using this ability, the player searches their deck or discard pile and adds the specified [amount] of allowed cards to their hand.

12. Deck

• deck:[target]:[destination]:[choice]:[amount]

• destination may be either destination:deck or destination:discard, qualified by either top or bottom

• choice specifies whether you choose, whether the target chooses or whether the choice is random

• using this ability, the player target can place cards from their hand into either the deck or discard, as specified.

13. Shuffle

• shuffle:[target]

• using this ability, shuffles the target deck. 2 cards accepted by the filter are shown in the interface, out of order. A player may choose those cards, and they will be removed from where they were in the deck/discard, but there is no reshuffling 6

14. conditions

• cond:[condition]:[ability](—[ability])*(:else:[ability](—[ability])*)

• If [condition] is met, apply the ability listed, otherwise apply the second ability (or do nothing if there is none listed)

• **conditions may be any of the following**

(a) healed:[target] (applies if target has been healed)

(b) flip (applies 50% of the time)

(c) ability:[ability] (applies of the player chooses to use that ability)

(d) choice (the player playing the ability chooses whether the condition passes)

15. add

• add:[target]:[ability](—[ability])*

• append this ability to all other abilities on the targeted pokemon.

Any listed [amount] may be replaced by a count[target](—[filter])*, e.g. count[target:your-bench] would use the amount of pokemon on your bench instead of an amount. Further, amount should support basic arithmetic with +/- /* and the use of parentheses. A more complex example would be count(target:your active—damage) which would count the damage on your active pokemon**.**

2.3 Details of Requirement No: 9. which is specified in Software Requirement Document

Status : Pokémon may be affected by the status effects, which affect how they may be used during a player's turn. Status effects are always removed when retreating or evolving. Each status effect may only be applied once per Pokémon (you cannot be poisoned twice, taking two damage per turn). Only these effects are considered:

1.Status: paralyzed: a paralyzed Pokémon may neither attack nor retreat. At the end of a turn, a player removes status: paralyzed from their active Pokémon.

2.Status: asleep: a sleeping Pokémon may neither attack nor retreat. At the end of a turn, a player has a 50% chance that sleep will be removed from their Pokémon.

3.Status: stuck: a stuck Pokémon may not retreat. At the end of a turn, a player removes status: stuck from their active Pokémon.

4.Status: poisoned: a poisoned Pokémon takes one damage at the end of their turn.

2.4 Details of Requirement No: 10. Which is specified in Software Requirement Document

Mulligans:

FULL DETAILS OF TAKING A MULLIGAN

If either player has no Basic Pokémon in his or her opening hand, that player must take a mulligan. Here's how the timing works: If both players have no Basic Pokémon in their opening hands: Both players reveal their hands and then just start over as normal. If only one player has no Basic Pokémon in his or her opening hand: That player announces that he or she has a mulligan, then waits until the other player has placed his or her Active Pokémon, Benched Pokémon, and Prize cards. Then, the player with no Basic Pokémon reveals his or her hand,

and then shuffles it back into his or her deck. The player does this until he or she gets an opening hand with a Basic Pokémon, then proceeds as normal. Then, the player who did not have to start over may draw a card for each additional time his or her opponent took a mulligan. For example, if both players took 2 mulligans, and then Player A took 3 additional mulligans, Player B may draw up to 3 cards. If any of those cards are Basic Pokémon, they may be put onto the Bench. Then, reveal all Pokémon, and begin the game

3. Usability

The Pokémon Go Back game is designed to be played any age group with minimum learning curve. The user should be able to play without any lag and each action takes maximum time of 15 seconds.

4. Reliability

The system must be reliable without frequent failures and should throw error messages when a mistake is made. If a system failure happens then the Time to Recovery must be very minimal.

5. Performance

The users must be able to use the system easily without any complexity. The UI and the work flow process should be made simple (avoiding any technical jargons) to avoid complexity and to increase system efficiency.

6. Design Constraints

The Pokémon Go Back game is developed in Java using Eclipse software. For GUI development Javafx library is used. Model View Architecture is used for the whole development process. Other tools used in the development are Scenebuilder, StarUML. For version control, Git is used.

7. Purchased Components

There are no purchased components being used in the game.

8. Interfaces

As we have mentioned earlier this is a standalone application so there is not any specific interface with which application need to communicate.

8.1. User Interfaces

We have designed an interface for this project in Javafx with the help of Scene Builder visual layout tool which helps us to make an efficient interface for the end user.

8.2. Hardware Interfaces

The protocols used for interaction between the computers standalone game and the video and sound card, are standard in a working operating system environment, therefore there are no hardware interfaces in special need of explanation.

8.3. Software Interfaces

Pokémon Go Back is designed to run on Windows operating systems. This project is basically standalone application that does not need any third-party assistance to run so that is why there is not any interaction with other systems.

8.4. Communications Interfaces

There is no need for any further explanation of the standard protocols used by the software and its connecting interfaces.

9. Licensing Requirements

The project is released under the GNU General Public License. The philosophy of this license implies some basic principles which apply to the project. The GPL is a free software license, and therefore it permits people to use and even redistribute the software without being required to pay anyone a fee for doing so.

- The freedom to run the program, for any purpose.
- The freedom to study how the program works, and adapt it to your needs.

Access to the source code is a precondition for this.

- The freedom to redistribute copies so you can help your neighbour.
- The freedom to improve the program, and release your improvements to the public, so that the whole community benefits. Access to the source code is a precondition for this.

(Source: http://www.gnu.org/copyleft/gpl.html)

10. Legal, Copyright and Other Notices

The Pokémon Go Back game is built under GPL and any notices included in the license can also be applied to this game.

11. Applicable Standards

This section describes by reference any applicable standards and the specific sections of any such standards that apply to the system being described. For example, this could include legal, quality and regulatory standards, industry standards for usability, interoperability, internationalization, operating system compliance, etc.

1. We have used Jakob Nielsen's heuristics for user interface design

2. To visualize the design of the system we have used Unified Modelling language standard by the Object Management Group

3. http://www.javaworld.com/article/2102551/java-language/googles-javacoding-standards.html

Google Java Style guide:

- No wildcard imports.
- Overloads appear sequentially.
- Braces are used even when the body is empty or contains a single statement.
- Two space indentation.
- Column limit can be 80 or 100 characters.
- No C-style array declarations.
- Default statements required in switch statements.
- Modifiers appear in the order recommended by the Java Language Specification.