Software Requirements Document

Project Name

Game Genie

Group Name

Hacker's Guide to the Galaxy

Date

30 April, 2021

Team Members

Ian Holder

Emily Crabtree

Suqoya Rhodes

Jesse Carter

Honor Code

We have abided by the UNCG Academic Integrity Policy on this assignment.

Table of Contents

1.3 - Purpose

The purpose of this application is to provide the user with a platform to rate games based on their preferences via a simple user interface, and use this data to provide appropriate game recommendations for the user in a visually simple but informative manner. This application will be turned in, along with the appropriate documentation, as a group project for CSC 340.

1.4 - Document Conventions

N/A

1.5 - Intended Audience

This document is intended for Software Engineering Professor Ike Quigley.

1.6 - Definitions

ESRB Rating - Entertainment Software Rating Board (ESRB) ratings are assigned to games to give guidance to consumers and advertisers on the appropriate audiences for that game. Possible ratings include EC (early childhood), E (everyone), E 10+ (ages 10 and up), T (teens, ages 13 and up), M (mature audiences, ages 17 and up), and AO (adults only, ages 18 and up).

1.7 - Project Scope

The scope of this project includes creating an application that will process user preferences to provide game recommendations. This will include an API connection, a functional user interface, and a form of persistent data storage. No maintenance is currently planned following the project deadline.

1.8 - Technical Challenges

As a team of novice developers with little to no experience working with API connections, persistent data storage, or GUI development, our progress will be hindered as we will be learning new things consistently throughout the process of creating this application.

In April, the RAWG API our team decided to use for our project was redesigned so anyone using the API would have to provide an API key or the API would not connect. This update caused a change in the way the URLs needed to be built, resulting in our team being unable to make API calls or make any updates to the API classes for a short period of time before the correct URL was determined.

1.9 - References

https://api.rawg.io/docs/

Section 2 - Project Description

2.1 - Project Features

Our application will present the user with video games, one at a time, and allow them to indicate their enjoyment level for that particular game. Alternatively, the user can indicate that they have not played the game. Based on the data collected, our application will provide recommendations for other video games that the user may like, based on the common attributes of the games they enjoyed.

2.2 - User Characteristics

Our application will be used by ourselves, as well as Ike Quigley, and anyone who enjoys video games.

2.3 - Operating Environment

This application can be used anywhere, so long as the user has connection to the internet.

2.4 - Design & Implementation Constraints

Our application will only be implemented in English. Additionally, we are limiting the scope of our design to cater only to people who have played video games before, and have a general understanding of video game jargon.

2.5 - Assumptions and Dependencies

This application assumes that users have a basic understanding of the jargon surrounding video games and the marketing thereof. The functionality of the

application is also dependent upon the RAWG Video Games Database API.

Section 3 - Functional Requirements

3.1 - Primary

The primary purpose of this application is to provide the user with a platform that will provide video game recommendations based on user-selected genres and/or previously liked/disliked games. In the application, there is a section dedicated to listing all the possible genres that the user can select to receive game recommendations from. After the user selects their genre(s), the RAWG API is called and the user is provided video games that fit under their chosen genre(s).

As the user likes or dislikes recommended games, their decisions are stored in a database. When a game is liked, its individual genres and tags are incremented. When a game is disliked, its genres and tags are decremented. As genres and tags are incremented and decremented, those with the highest values are used to recommend the user games based simply off of the games they have already rated.

3.2 - Secondary

The secondary purpose of this application is to locally save the user’s save files from the application.

Section 4 - Technical Requirements

4.1 - Operating Systems/Compatibility

Our application can be used on any computer that has access to a Java development environment.

4.2 - Interface Requirements

4.2.1 - User Interface

In order to use our application, the user will need access to a computer, as well as a monitor as the application interface requires visibility to make selections by clicking checkboxes and buttons.

4.2.2 - Hardware Interface

The user must have access to a mouse in order to make selections in the genres list, as well as like, dislike, or select “I don’t know” for a recommended game.

4.2.3 - Software Interface

In order to use our application, the user would need access to a Java development environment that supports Java FX.

4.2.4 - Communications Interface

The user will have to have access to the internet in order to use our application. Without internet, the program will be unable to make calls to the API and therefore will be unable to recommend games or store liked/disliked games.

Section 5 - Nonfunctional Requirements

5.1 - Performance Requirements

Our application will feature a simple user interface that is intuitive and user friendly. The interface is clean and does not have distracting images. The various ways to filter which games are recommended are obvious and buttons are clearly labelled.

5.2 - Safety/Recovery Requirements

When a user opens our application, the program checks to see if there is a previous save file on the computer. If there is a previous save file, that file is reopened the user will continue where they left off with games previously liked or disliked. If there is not a previous save file, a new one is created in the database and the user starts from scratch. If the application were to crash, when it is reopened, the save file will be detected and the user will pick up with the data they had prior to crashing.

5.3 - Security Requirements

Our application does not require users to create an account or login to use the platform.

5.4 - Policy Requirements

N/A

5.5 - Software Quality Attributes

5.5.1 - Availability

While our application does not feature a login system, the user's liked/disliked games are saved in both a database and locally on the user's computer, so the application will reopen to the most recent save file for the user's machine.

5.5.2 - Correctness

Because our application provides the user with checkboxes to select which genre(s) they would like their games to be recommended from, there is little room for correctness errors. The user will not have the responsibility of correctly spelling and inputting the genre(s) they would like to have games recommended from.

If a user were to accidentally like a game they wanted to dislike or dislike a game they wanted to like, the effect of the misclick would depend on how many games they have already rated that feature those same genre(s) and tag(s). Because each like is an increment of 1 and each dislike is a decrement of 1, accidentally liking an action game would have more influence on a selection of 5 rated actions than a selection of 30 rated action games.

5.5.3 - Maintainability

Since our application relies heavily on API calls, maintainability as far as how the application functions would rely heavily on the maintenance of the API. For example, while developing our application, the RAWG API we are using updated their API to require each API call have an individual API key, thus reworking the url used in the API caller class. When this happened, we had to identify where the url changed and update the class. Maintenance for the program would require redownloading updated code from the Github repo, but no maintenance has been planned.

5.5.4 - Reusability

The API caller and translators used in our program for the RAWG API could be reused in other projects that require an API call to the same software. However, the other classes and methods are specific to the RAWG API, so those would not be easily reusable.

5.5.5 - Portability

Our application is easily portable to another machine, as long as the machine has a Java development environment that supports Java FX.

5.6 - Process Requirements

5.6.1 - Development Process Used

Our team primarily used waterfall methodology to work on our application. Every Tuesday and Thursday at 5:00pm, we would meet on Discord voice chat for 1-2 hours, talking about what roles each team member would take, what our next step needed to be in the development of our project, and checking in on where each team member was in their step. However, sometimes we took a more agile approach and focused on planning and developing a specific aspect of the project in the meeting. As we worked on the project and ideas for what the end product would be changed, alterations to the documentation and diagrams were made.

5.6.2 - Time Constraints

Because this project is a class project, our allotted time was restricted to the semester with the final project due on 30 April, 2021.

5.6.3 - Cost and Delivery Date

The API we chose to use for our project, RAWG, offers a free plan, so there was no monetary cost to our project. The delivery date is 30 April, 2021, the date the final project is due.