Roguelike Deckbuilder

Software Requirements Specification

1.1.1

03/01/2022

Cody Rogers

Lead Software Engineer

KyuSeung Sim

Nyah Hughes

Eden Florianne Merigold

Prepared for

CS 372—Software Engineering

Instructor: Tanim Ahsan

Spring 2020

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Description** | **Author** | **Comments** |
|  |  |  |  |
| 03/01 | Version 1 | KyuSeung Sim | File Sharing and add title, DFD, and Appendix A |
| 03/09 |  | Cody Rogers | Added information to various subsections |
|  |  | KyuSeung Sim | Fix State Transition Diagrams  Appendix A update |
| 3/11 |  | Nyah Hughes | Added information to 3.4 Classes/Objects  Added information to 2.5 |
|  |  | KyuSeung Sim | Added information to 3.3 Use case  Added information to 2.3 User characteristic |
|  |  | Cody Rogers | Added information to 1.5, 2.2, 2.4, 5 |
|  |  | Eden Florianne Merigold | Editing, error checking, review, cleanup |

# Document Approval

The following Software Requirements Specification has been accepted and approved by the following:

|  |  |  |  |
| --- | --- | --- | --- |
| **Signature** | **Printed Name** | **Title** | **Date** |
| Cody Rogers | Cody Rogers | Lead Software Eng. | 11 Mar |
|  | Tanim Ahsan | Instructor, CS 372 |  |
|  |  |  |  |

**Table of Contents**

Revision History ii

Document Approval ii

1. Introduction 1

1.1 Purpose 1

1.2 Scope 1

1.3 Definitions, Acronyms, and Abbreviations 1

1.4 References 1

1.5 Overview 1

2. General Description 2

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Characteristics 2

2.4 General Constraints 2

2.5 Assumptions and Dependencies 2

3. Specific Requirements 2

3.1 External Interface Requirements 3

3.1.1 User Interfaces 3

3.1.2 Hardware Interfaces 3

3.1.3 Software Interfaces 3

3.1.4 Communications Interfaces 3

3.2 Functional Requirements 3

3.2.1 <Functional Requirement or Feature #1> 3

3.2.2 <Functional Requirement or Feature #2> 3

3.3 Use Cases 3

3.3.1 Use Case #1 3

3.3.2 Use Case #2 3

3.4 Classes / Objects 3

3.4.1 <Class / Object #1> 3

3.4.2 <Class / Object #2> 3

3.5 Non-Functional Requirements 4

3.5.1 Performance 4

3.5.2 Reliability 4

3.5.3 Availability 4

3.5.4 Security 4

3.5.5 Maintainability 4

3.5.6 Portability 4

3.6 Inverse Requirements 4

3.7 Design Constraints 4

3.8 Logical Database Requirements 4

3.9 Other Requirements 4

4. Analysis Models 4

4.1 Sequence Diagrams 5

4.3 Data Flow Diagrams (DFD) 5

4.2 State-Transition Diagrams (STD) 5

5. Change Management Process 5

A. Appendices 5

A.1 Appendix 1 5

A.2 Appendix 2 5

# 1. Introduction

## 1.1 Purpose

To outline the intended requirements of the program to all team members and ensure all expected user (player) inputs and outputs are communicated, understood, and implemented properly. Check for and resolve inconsistencies between individual team members

## 1.2 Scope

* Roguelike Deckbuilding Game
* Users (players) will interface with a HUD that shows their controlled character’s resources (cards, health, map, etc.) via clicking and dragging scene elements. The character’s current resources are directly manipulated through the HUD to prepare for and control actions during battle sequences.
* The battle sequences ultimately dictate the outcome of the game (user’s success or failure)
* Resources will be prominently displayed on HUD and sequences of actions between map and battles should have streamlined flow that is intuitive to user through interactive graphics (buttons and cards) or simple text
* Manipulation of resources and battle sequences should be simple and fun, while still offering some challenge

## 1.3 Definitions, Acronyms, and Abbreviations

* Novice: new to roguelike and deckbuilding game genres
* Expert: experienced in roguelike deckbuilding genres – can adeptly use and be successful with scarcer resources
* Player – individuals operating the program for the purposes of entertainment or playtesting
* HUD – heads up display that presents the user with information pertinent to tactical decisions
* Battle sequence – a series of events initiated when the user selects a Battle node as indicated on the Map screen
* Soft lock – a gamestate in which the player's progress is halted indefinitely due to unintended limiting factors that the player has little to no control over. Results in a frustrating rather than enjoyable experience

## 1.4 References

* NO CURRENT REFERENCES

## 1.5 Overview

General descriptions of functions and front-end usage, with any constraints or dependencies

Any special requirements and example use cases

Diagrams to inform functional design implementation intents

# 2. General Description

# 2.1 Product Perspective

* Distinct from most game programs in that it can be completed in a single session

## 2.2 Product Functions

Player controls a character with limited resources and uses map to enter different game states determined by the map node’s type; shop, event, or battle

node selection: map UI that player may click on to select which event to initiate (shop, battle, or event room)

room checks: various checks of rooms to determine which battle room type, whether it is a boss battle room, if last floor

get resources: methods to supply player with resources (cards, coins, health, etc.) from defeating an enemy or entering an event room

check/update status and game end: updates character's status (health) and checks if game should be ended

buy: methods to purchase and sell items or cards when room is a shop

These methods should create foundation for enjoyable gameplay flow and variable factors options to balance game if expectations are not met

## 2.3 User Characteristics

* Player is expected to click the mouse button and show result from the monitor. Game system received player’s action then show result from each action code. This information also discussed in 3.3 use cases.

## 2.4 General Constraints

* User controls should be simplified, and duration of games to completion or character’s defeat should not require multiple sessions to avoid overwhelming scope

## 2.5 Assumptions and Dependencies

* Our project relies solely on JavaFX. If it were to be unavailable, we would need to turn to a different Java-based GUI builder. In the event JavaFX is unavailable, the project could turn to an Android phone game, and we could move to using Android Studio.
* However, if JavaFX is temporarily unavailable on the hardware, then one could use an online Java IDE until the issue is resolved. A link has been provided below for an online IDE that allows working with multiple files:
  + <https://www.jdoodle.com/online-java-compiler-ide/>

# 3. Specific Requirements

# 3.1 External Interface Requirements

### 3.1.1 User Interfaces

* Mouse – to click JavaFX elements

### 3.1.2 Hardware Interfaces

* Minimum to run Java programs
* Display and Mouse

### 3.1.3 Software Interfaces

* Minimum to run Java programs

### 3.1.4 Communications Interfaces

* None

## 3.2 Functional Requirements

## 3.2.1 <Functional Requirement or Feature #1>

3.2.1.1 Introduction

3.2.1.2 Inputs – player can reliably click items in inventory and map

3.2.1.3 Processing – strains on performance due to processing are not expected due to nature and scope of program. Will update as needed

3.2.1.4 Outputs – graphics and updates chance appropriately as player interacts with objects, and to reflect changes in variables pertinent to player’s input decisions

3.2.1.5 Error Handling – with any effect that the game is not crashed

## 3.3 Use Cases

### 3.3.1 Use Case #1

* Player User: A user of the system who play the game with G.U.I. User click with a mouse to select the node then received various events.

### 3.3.2 Use Case #2

* Game System: Entire system to check player user’s click event then process received events (contain Battle, Shop, Random Event).

## 3.4 Classes / Objects

### 3.4.1 <Class / EventRoom>

3.4.1.1 Attributes

3.4.1.2 Functions

<Reference to functional requirements and/or use cases>

* getRandomItem()
* getRandomCard()
* getRandomLife()
* loseRandomLife()

### 3.4.2 <Class / Room>

* BattleRoom()
* ShopRoom()
* EventRoom()

### 3.4.3 <Class / Shop Room>

* BuyCard()
* BuyItem()
* DeleteCard()

### 3.4.4 <Class / Battle Room>

* Normal
* Elite
* Boss

### 3.4.5 <Class / Battle>

* Normal
* Elite
* Boss

### 3.4.6 <Class / Boss>

* Floor

## 3.5 Non-Functional Requirements

### 3.5.1 Performance

* Program runs smoothly – transition between input screens and other events have insignificant to no downtime

### 3.5.2 Reliability

* Program does not crash
* Player is unable to put game in softlock during standard gameplay

### 3.5.3 Availability

* Product available to individuals with source files

### 3.5.4 Security

* No Concerns

### 3.5.5 Maintainability

* After product release, playtest (consider finding players). Use feedback to determine whether user experience was enjoyable. Adjust properties of in-game character resources and other mechanics for game balancing to improve experience

### 3.5.6 Portability

* Run on PC devices and laptops

## 3.6 Inverse Requirements

* None

## 3.7 Design Constraints

* Graphics must be used with JavaFX
* No third party integration

## 3.8 Logical Database Requirements

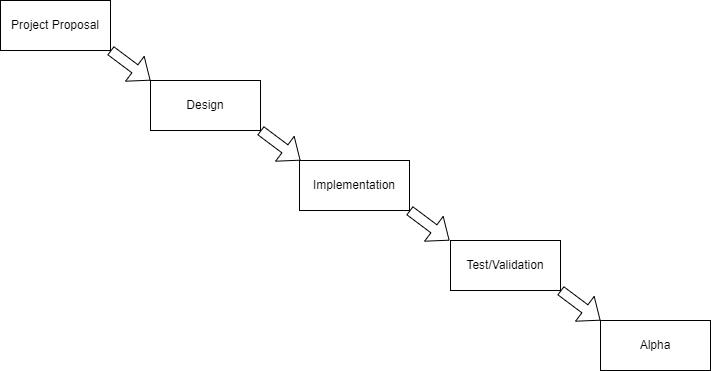
* No extensive database required

## 3.9 Other Requirements

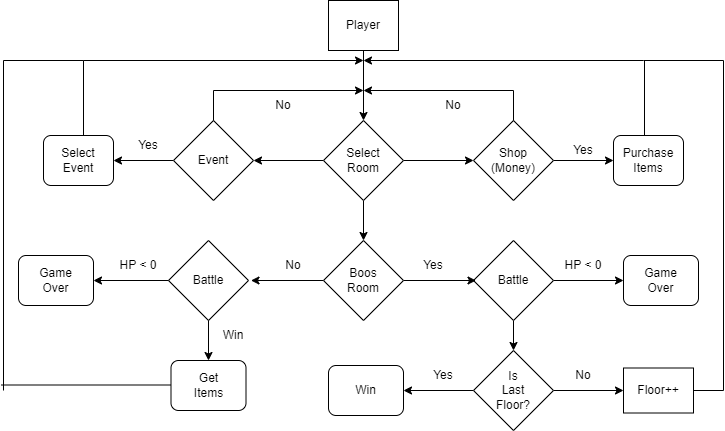
* No additional data requirements

# 4. Analysis Models

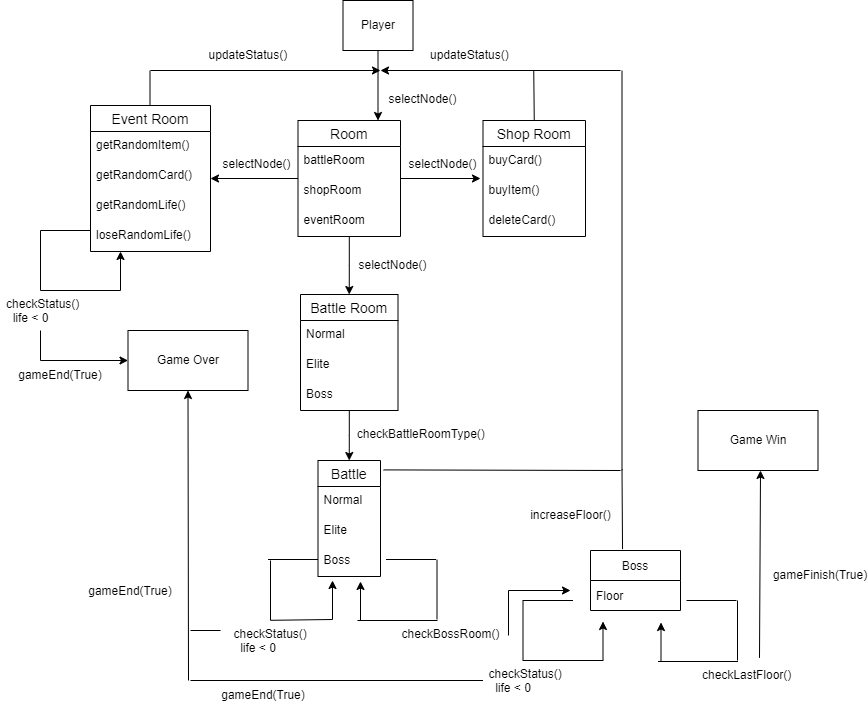
# 4.1 Sequence Diagrams



## 4.3 Data Flow Diagrams (DFD)



## 4.2 State-Transition Diagrams (STD)

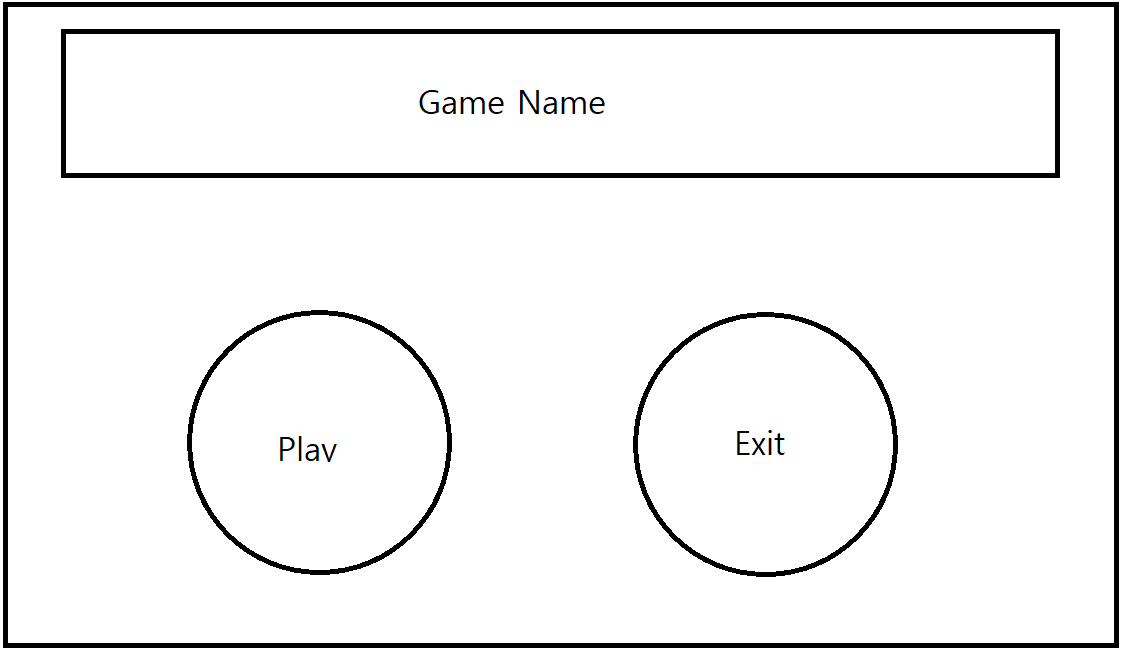


# 5. Change Management Process

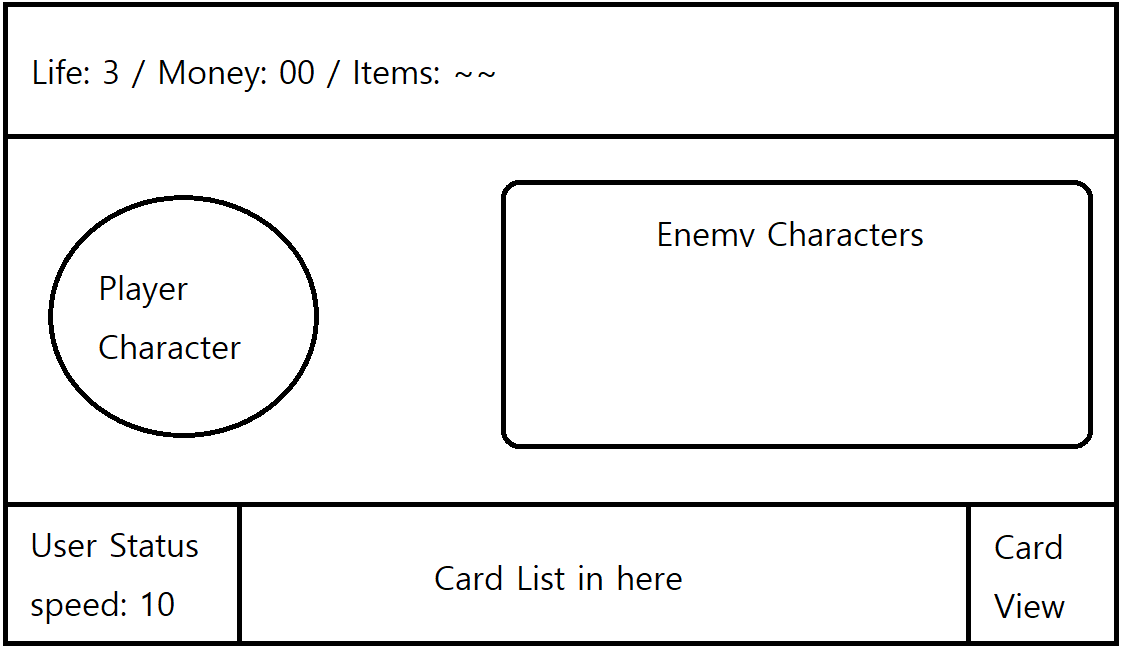
Any changes or additions made directly to document should be recorded on revision history table. Modifications past the SRS final date due to requirement changes will be communicated directly to the team. Approval of these changes must come from unanimous decision of whole team

# A. AppendicesA.1 Appendix 1

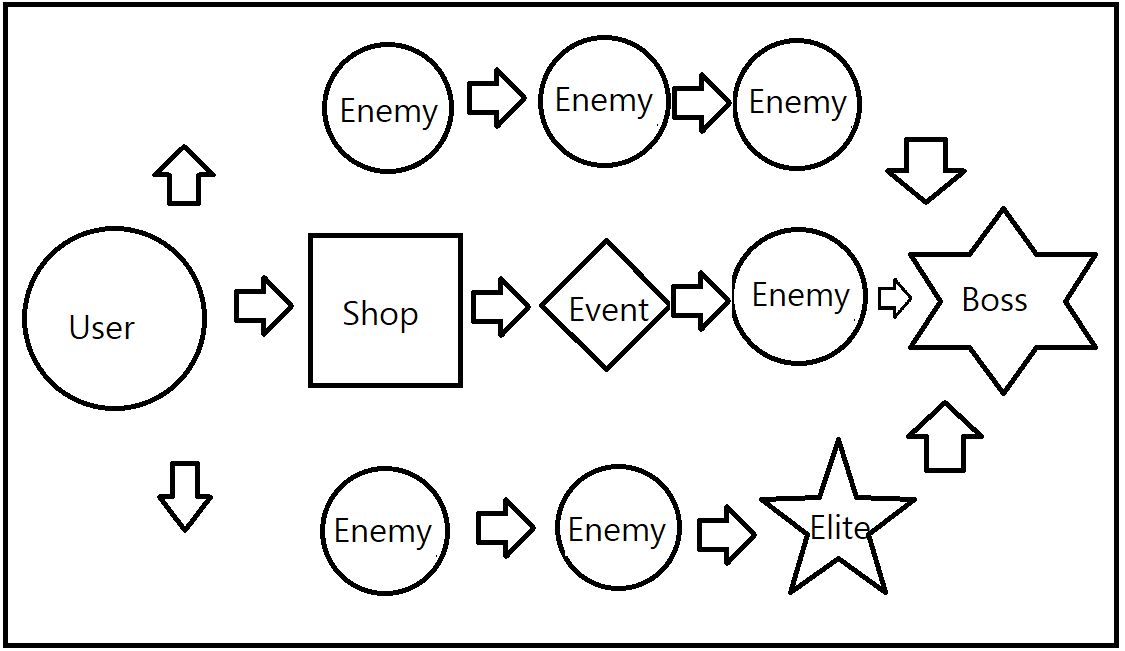
Start UI



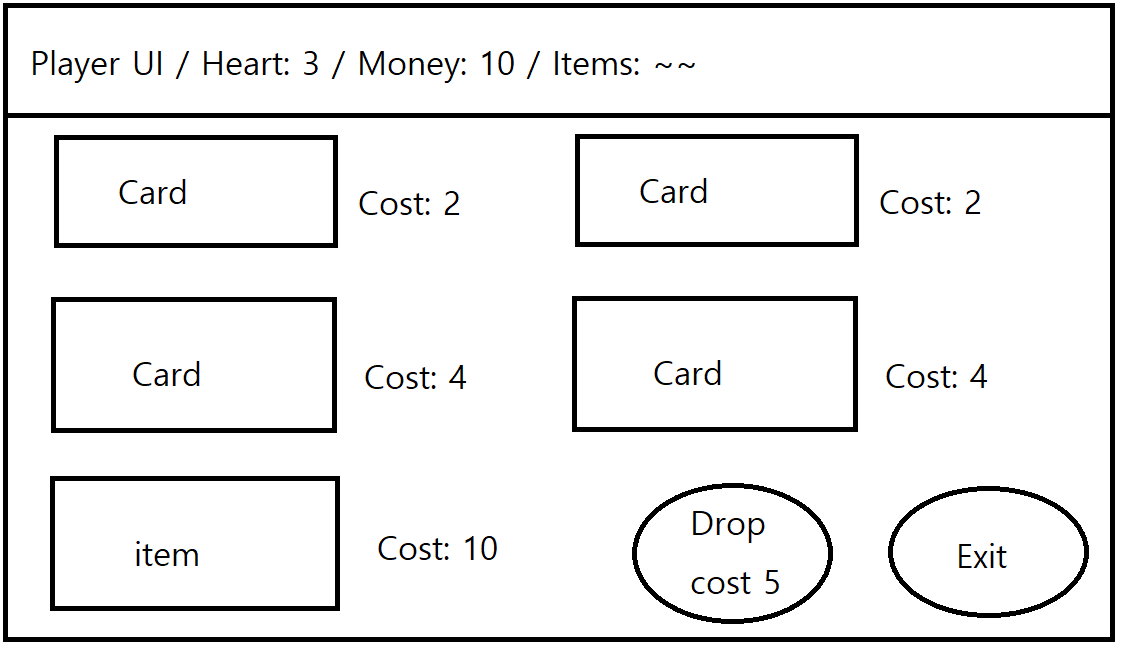
Battle UI



Map UI



Shop UI



Event UI

