Software Requirements Specification

for

Dungeon Adventure 2.0

Final Draft Document

Prepared by Halt Catch Fire

University of Washington – Tacoma Campus

March 14, 2022

Revision History

Name	Date	Reason For Changes	Version
Kevin Chung	1/25/22	Initial Draft Software Requirements and Specifications (SRS)	1.00
Stephanie Liu	1/27/22	See above. 1.4, 2.1, 2.3, 2.7, 3 through appendix adjusted.	1.01
Kevin Chung	1/28/22	Added 3.8-3.10	1.02
Stephanie Liu	1/29/22	Adjusted 3.8-3.10 since 3.9 & 3.10 were duplicates.	1.03
Stephanie Liu	3/14/22	Final Draft to match final state of project.	1.04

1. Introduction

1.1 Purpose

<Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.>

This SRS describes the software functional and nonfunctional requirements for release 1.0 of Dungeon Adventure – The Spoony Bard Returns. This document is intended to be used by the members of the Halt Catch Fire team that will implement and verify the correct functioning of the software. Unless otherwise noted, all requirements specified here is a final draft.

1.2 Document Conventions

<Describe any standards or typographical conventions that were followed when writing this SRS, such as fonts or highlighting that have special significance. For example, state whether priorities for higher-level requirements are assumed to be inherited by detailed requirements, or whether every requirement statement is to have its own priority.>

Player: User of the software. May be a professor.
Developer: Software engineering student.
Demo: Developer using software as a demonstration of functionality for an audience.

1.3 Intended Audience and Reading Suggestions

<Describe the several types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections, and proceeding through the sections that are most pertinent to each reader type.>

This SRS document is intended for software engineering students and faculty enrolled in the Graduate Certificate program at University of Washington – Tacoma. The contents of this SRS contain implementation assessments for project deliverables. Read this document sequentially, unless otherwise noted. Previous course lectures may be referenced throughout this document, noted with course name and lecture title.

1.4 Project Scope

<Provide a brief description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here. An SRS that specifies the next release of an evolving product should contain its own scope statement as a subset of the long-term strategic product vision.>

Our goal for this project is to extend our previous course project, Dungeon Adventure – The Spoony Bard. See 2.2 for additional information about features and project scope. Will be a game

where a player can win and lose, select multiple classes, save and load a game, select difficulty, and play through using a GUI.

1.5 References

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader can access a copy of each reference, including title, author, version number, date, and source or location.>
OneDrive/Halt Catch Fire/TCSS 504/
dungeon_adventure_project_details.docx
dungeon_adventure_srs_initial.docx

GitHub/The Spoony Bard/
Original code base – tcss504 branch

Toggl time tracker
Pivotal project tracker

2. Overall Description

2.1 Product Perspective

<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.>

This project is a continuation of the previous end of the quarter project from TCSS 502 called Dungeon Adventure – The Spoony Bard. The new project should be considered a replacement for the previous iteration.
The software will use Tkinter for a graphical user interface (GUI).
The software will use SQLite databases that will store Hero/Monster creation data, difficulty data, and saved states.
Saved states will be stored in the SQLite database as JSON objects due to being human read-able.

2.2 Product Features

<Summarize the major features the product contains or the significant functions that it performs or lets the user perform. Details will be provided in Section 3, so only a high-level summary is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top-level data flow diagram or a class diagram, is often effective.>

Proposed features:

- New dungeon classes:
    - Hero

        - Warrior
        - Priestess
        - Thief
        - Bard – not yet implemented
    - Monster
        - Ogre – bosses in pillar rooms
        - Gremlin
        - Skeleton
- Game Play Mechanics
    - Combat
        - Attack speed
        - Damage inflicted
        - Battle rounds
        - Chance to hit
        - Attack/Defense mechanics
        - Special abilities to heal for priestess, chance of more damage for thief and warrior.
    - Monsters must be defeated throughout the dungeon
    - Stronger monsters are in rooms with pillars (Ogres)
    - Each hero class has different abilities
    - Ability to save and load existing game states
- SQLite Database Support
    - Monster table (name and statistics)
    - Adventurer table
    - Difficulty table (data for dungeon creation)
    - Saves table to save current state and load game
    - Used during the creation of the dungeon
- Additional features as deemed by Halt Catch Fire
    - Flee from an encounter (Kevin) (for demo only)

2.3 User Classes and Characteristics

<Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the favored user classes from those who are less important to satisfy.>

Player user class:
A player will want to play a game and save the current state of progress within the game. A save/reload feature will be required for the desired behavior.
A player will want to win the game.

Tester user class:
Like the player user class but provides access to cheat codes to view map immediately, have unlimited potions, instantly beat a monster, lose, have more health, have more vision potions, etc...
Developer would use this to test functionality throughout the development process.

2.4 Operating Environment

<Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.>

Requirements:

- Windows, Mac/Linux operating systems
- PyCharm Community Edition 2021.3.1+ or VSCode 1.63+
- TKinter
- Python 3
- SQLite3
- PIP for managing packages (command: pip install -e .)

2.5 Design and Implementation Constraints

<Describe any items or issues that will limit the options available to the developers. These might include corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer's organization will be responsible for maintaining the delivered software).>

Time constraint as most peers will be able to work on this project for at most 7-9 hours a week
Limited knowledge of working with a team to build a complex program
Size constraint – submitted through Canvas
Limited knowledge of SQL and TKinter
Security considerations aren't a major issue as game is designed to be run and played on a single machine.
Graphics are not intense, and will run on most modern laptops.

2.6 User Documentation

<List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software. Identify any known user documentation delivery formats or standards.>

Docstring comments will be adherent to PEP8 standards. Sphinx preferred.
User documentation on how to use the software may be found within the Help menu.

2.7 Assumptions and Dependencies

<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project, unless they are already documented elsewhere (for example, in the vision and scope document or the project plan).>

We are dependent on external services listed in 2.4 and should any of those be unavailable for any reason or pose any unknown threats. Similarly, vulnerabilities in SQLite leave us more vulnerable, however due to the project being only available on a local machine, a lot of these considerations are less problematic.

3. System Features

<This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>

3.1 Start Game

3.1.1 Description and Priority
<Provide a brief description of the feature and indicate whether it is of High, Medium, or Low priority. You could also include specific priority component ratings, such as benefit, penalty, cost, and risk (each rated on a relative scale from a low of 1 to a high of 9).>

Open game - 9

3.1.2 Stimulus/Response Sequences
<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>

User runs game from main.py. Game window opens for start screen. - 9

3.1.3 Functional Requirements
<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use "TBD" as a placeholder to indicate, when necessary, information is not yet available.>

<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>

REQ-1: main.py file created with connected controller, DB, and game view.
REQ-2: Must display start screen to player.

3.2 New Game

3.3.1 Description and Priority
Able to start a new game. Second highest of priorities.

3.2.2 Stimulus/Response Sequence:
Player clicks "New Game." View asks for difficulty, hero name, and class. Controller sends difficulty setting to Model, which generates a dungeon containing monsters to match the current difficulty level.

3.2.3 Functional Requirements:

REQ-1: View implemented so user can click on New Game which takes to new view to ask data.
REQ-2: View able to take input for difficulty, hero name, class.
REQ-3: View able to send that information to controller on button press.
REQ-4: Controller able to modify info into Model friendly language and send it difficulty level and hero information to generate the game.
REQ-5: Controller can take generated information from Model for hero and dungeon and provide it to view.
REQ-6: View can display that information.

3.3 Monsters

3.3.1 Description and Priority
Monsters with different abilities exist within the game and are in the dungeon.

3.3.2 Stimulus / Response Sequence
When loaded, a dungeon is filled with monsters in available slots. Monsters display in dungeon room when player enters room with monsters. Combat initiated.

3.3.3 Functional Requirements
REQ-1: Monsters have the following attributes:
- Names
- Stats (hp, mp, defense, attack, initiative?)
- Special abilities

REQ-2: Monsters require way to be displayed in view.
REQ-3: Boss monsters in pillar rooms are always ogres.
REQ-4: Monsters differentiated by difficulty by the frequency they occur.
REQ-5: Dungeon needs a way to know a monster is in the room.
REQ-6: Monsters need to be created when difficulty is set using dungeon builder pattern.

3.4 Heroes

3.4.1 Description and Priority
Heroes with different classes (and matching abilities) exist within the game and are in the dungeon. Identify location of hero.

3.4.2 Stimulus / Response Sequence
Created, hero name and class are selected by player. Players see their hero, and when engaging in combat, have a specific skill for their hero.

3.4.3 Functional Requirements
REQ-1: Heroes have the following attributes:
- Names
- Stats (hp, block chance, attack, attack speed)
- Special abilities
- Inventory of pillars and potions

REQ-2: Heroes require way to be displayed in combat.
REQ-3: Dungeon needs a way to display hero on a map.
REQ-4: Heroes have inventory

3.5 Combat

3.5.1 Description and priority

Combat occurs when a hero enters a room with a monster. It ends when one of them reaches 0 HP.

3.5.2 Stimulus / Response Sequence:
Hero enters room with monster. Combat initiated. Based on attack speed, player selects skill or attack, does so, and then monster goes or monster goes first. May have multiple attacks in a single round. Once all have gone, starts over until one has died / drop to 0 HP. Once they have, exit combat either to game over screen or resuming from map.

3.5.3 Functional Requirements
REQ-1: Initiative / attack speed compared
REQ-2: Whoever has higher attack speed or initiative goes first.
REQ-3: Need interface for player to select what they want to do then have that selection sent to controller which processes the outcome and updates local model. View will be updated by controller with updated stats to display for both.
REQ-4: Need combat interface and menu.
REQ-5: Need to display outcomes of each round as they happen or a log to show monster actions.
REQ-6: Display status of bother parties.
REQ-7: Enable potion usage during battle.
REQ-8: Displays victory of defeat at the end of combat.

3.6 Save Game

3.6.1 Description and priority
Ability for players to save games.

3.6.2 Stimulus / Response Sequence
Player clicks save or reaches a save point (should we add these?) and saves the game so it can be reloaded later.

3.6.3 Functional Requirements
REQ-1: View option to save the game
REQ-2: When save is clicked, generates a file, or saves data to a DB to store current game state.
Game state data:
Hero (hp, name, class, difficulty)
- Hero location
- Current inventory
- Current viewed map
- Whole dungeon (including pillar, monster location with reference IDs)

3.7 Load Game

3.7.1 Description and priority
Ability for players to load game from a saved game data.

3.7.2 Stimulus / Response Sequence
At the main menu, the player clicks load game. Game gives menu of saved games. Player selects from saved games and continues from previous game state.

3.6.3 Functional Requirements
REQ-1: View option to load the game.

REQ-2: View displays saved games when load is clicked.
REQ-3: When load is clicked, then loads up the saved game to continue.
REQ-4: Need to be able to process saved game state into active running game.
REQ-5: Need saves functional, with saves games stored locally in a database.

3.8 Win Game

3.8.1 Description and priority
Ability for users to achieve a victory state in the game:
Escape the dungeon after collecting all the pillars and reaching the exit

3.8.2 Stimulus / Response Sequence
Once the player reaches the exit of the dungeon, if they have all pillars, they win the game and exit.
A final screen will be presented with information about the dungeon session and a congratulatory message upon completion.

3.8.3 Functional Requirements
REQ-0: Check on game state to see if victory conditions have been met
REQ-1: Congratulatory message at the end
REQ-2: Provides options to start a new game, load game or quit program

3.9 Lose Game

3.9.1 Description and priority
Ability for users to lose the game.
Users can be defeated by monsters that overpower the hero character.

3.9.2 Stimulus / Response Sequence
Once a player is defeated by monsters by going below 0 hit points the game should end.

3.9.3 Functional Requirements
REQ-0: Check on game state to see if lose conditions have been met
REQ-1: Game over message at the end
REQ-2: Provides options to start a new game, load game or quit program

4. External Interface Requirements

4.1 User Interfaces

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons, and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>

The software will require the following views:

View	Description

Start screen	Start a new game or continue previous game and load from saved game.
Dungeon crawler	Move character around map, indicates any item found in text to the side as a log. Menu to save game, quit, load, help, check inventory and character status.
Inventory menu	Shows character status and player inventory. Allows item use.
Dungeon Brawler	Log of battle actions taken by monsters and by players
Tutorial / Help	Explain how to play the game, key commands.
Win Game	Screen for victory. Should provide same options as start screen but also display some details of winning the game.
Lose Game	Screen for failure. Similar to above but for losing.

Start Screen

New Game creation.

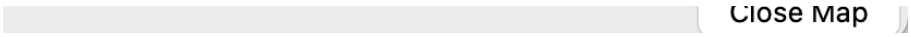
New Hero Creation.

Dungeon Crawler

Dungeon Brawler

Load Game

Inventory



Map

4.2 Hardware Interfaces

<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>

Supports computer screen resolutions 800 x 800 pixels or larger.

4.3 Software Interfaces

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be

implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>
Dependent on SQL using a QueryHelper interface to manage access to the database and a SaveManager to modify data going in and out of the database as well.

4.4 Communications Interfaces

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>

Run on a local machine with Python 3 interpreter. Saved on local and loaded as well.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>
Game is expected to save and load reasonably quickly as it is an individual playing on their own machine and is not reliant on unpredictable factors such as querying a remote database. The data models are expected to be consistent for saves, as thinking you saved and finding out that you had not is the stuff of geeky nightmares. It should be highly reliable. The game should not crash on a local system for previously aforementioned reasons. Any lack of reliability on the game and software end outside of a user's computer having individual issues is unacceptable.

Some design choices to help with performance are not to create potions until they are used to decrease memory dependency. This is not the case with monsters, and potentially a future refractor here would be beneficial for improved performance.

5.2 Safety Requirements

<Specify those requirements that are concerned with loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product's design or use. Define any safety certifications that must be satisfied.>

None required at this time as the game plays locally on a user's machine only.

5.3 Security Requirements

<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication

requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>

Software will be using a SQL database; therefore, connections must be secured.
Serialization of game states will be, therefore use serialized objects for secure/trusted sources only. As we are using human readable JSON objects and the dictionaries are checked for having the required keys and load in a way that should be relatively safe, and is only played locally on a user's computer, this isn't a massive concern. Don't download other people's saved states off the internet, folks!

5.4 Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>

Unit testing will be vital to verify the individual modules other components depend on are comprehensible to their users. Modular design will also be vital due to how much larger this program will be. Modular design for model well implemented using OOP practices. This lead to the model code being very well tested, reusable, and flexible as features were expanded (to include things like saves).

6. Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Avoid copyright infringement.

6.1 Appendix A: Glossary

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>

6.2 Appendix B: Analysis Models

<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams.>

6.3 Appendix C: Issues List

< This is a dynamic list of the open requirements issues that remain to be resolved, including TBDs, pending decisions, information that is needed, conflicts awaiting resolution, and the like.>

No bards yet implemented. Refactoring to make View and Controller more modular would be ideal.