Software Requirements Specification

Dungeon Escape

Prepared by Darence Thong, Jennifer Boutell, JP Montagnet

UW Tacoma GC-SD

3/18/22

Table of Contents

Revision History

Name	**Date**	**Reason For Changes**	**Version**
Team	1/30/22	Initial draft	1
Jennifer Boutell	2/20/2022	Updating references and characters	2
Jennifer Boutell	3/18/2022	Final Revisions	3

1. Introduction

1.1 Purpose

Dungeon Adventure is a fantasy role playing game, in which an adventurer navigates through a maze, experiences challenges, and collects items. The goal of the game is to identify the exit while in possession of the required items. Dungeon Adventure 1.0 was released in December, 2021.

This document provides a detailed description of Dungeon Adventure 2.0 for developers and stakeholders. It will describe technologies used, design and features, software and hardware requirements, interface/GUI, software constraints, and the scope of this project.

Additionally, this document has aided the software developer team in the construction of the second version of this dungeon project.

1.2 Document Conventions

Subject headings are indicated by Times New Roman, bold font, and numbered by section and subsection. Subject content is provided in Ariel font, normal typesetting.

1.3 Intended Audience and Reading Suggestions

Software developer engineers and students are the primary targeted audience. Related stakeholders can also find utility from this document.

The suggested sequence for reading this document for developers and project managers is to begin with the overview sections and proceed through all other sections in order. The suggested sequence for other stakeholders is to read through the introductory sections and select other sections as relevant to the readers' purpose, from the table of contents.

1.4 Project Scope

The purpose of Dungeon Adventure 2.0 is to provide a fun gaming experience for the user. This project will allow creators to refactor code that is well tested and exemplifies object oriented programing, add additional features that will utilize a SQLite database, add GUI, and create a more varied and interesting gaming experience than version 1.0.

1.5 References

Dungeon Adventure 1.0 assignment from TCSS502, Fall 2021.

ShapeFactory assignment from TCSS502, Fall 2021
Raycasting: https://www.youtube.com/watch?v=AjPPhx8-lXg

2. Overall Description

2.1 Product Perspective

This project is a second version. The first version was completed in December 2021. The intent of this version is to add additional features, code refactoring, and add GUI.

Features that will be added are multiple hero characters to pick from and introduction of enemy characters. Enemies will have hit points and varying attack abilities that the hero character will encounter. Additionally, the new version will feature an advanced GUI, allowing the player to navigate through the maze in a 3D-like setting while encountering animations and sounds.

The previous version that requires the player to find 4 pillars to win while encountering health potion, vision potion, and pits will be preserved.

An updated UML will outline the entire program, showing the class relationships in object oriented programming.

Finally, a video demonstration will be recorded to show how a user would play the game.

2.2 Product Features

Enemy characters will be stored in a SQLite database. Enemies are generated into this database and then retrieved during the generation of the dungeon.

The GUI will be created using Pygame, and Raycasting. Sound effects and music will be incorporated with Pygame as well.

2.3 User Classes and Characteristics

User: Game user is proficient in English language at a minimum 5th grade reading level, and familiar with use of either a keyboard and pointing device. Unlikely to explore features when not explicitly required.

Fanatic: Possesses characteristics of User class, plus advanced understanding of the fantasy genre, RPG gameplay and tolerance for terrible jokes. Likely to explore most or all features of the game.

Technical: Typically a tester or instructor. Expected to traverse the game more extensively than other users and identify potential technical issues. May potentially overlap with the Fanatic class.

2.4 Operating Environment

This software can be operated in Windows, Mac, and Linux operating systems, with Python 3.x installed. The user must be able to run Python in order to access the game. It is recommended that the user's device have at least 4 GB of RAM. A default-motherboard's graphics card should be sufficient.

2.5 Design and Implementation Constraints

A computer with the current version of any Linux, Windows, or Mac based operating system and a minimum of 4 GB of RAM should be sufficient to run Dungeon Adventure 2.0

2.6 User Documentation

User documentation will be provided in-game with use of GUI. Guidelines on how to play the game will be explained.

Sample instruction text:

```
Welcome to Dungeon Escape, brave Hero!

The 4 pillars of object-oriented programming have been captured by the evil Sorcerer King
Ca-Paul, and the world has been plunged into an age of darkness and endless scripts. Ca-Pu has
placed the pillars under guard by fearsome monsters: Ogres, Skeletons, Gremlins, and worst of
all--Mean Girls.

Your task--should you dare to accept it--is to locate the four pillars and find the exit to
Ca-Paul's dungeon before the monsters drain you of all of your health points.

Take heart! For you will find that not all in Ca-Pul's realm are in league with his evil scheme.
Within the dungeon, neutral wizards have placed Vision potions and Healing potions for your use.
You have also been granted a special ability, which you may use to vanquish Ca-Pul's evil
brethren.

Use the following commands to ease your journey:

Arrow keys - find your way
R - use your special ability, most useful in a fight
H - use a healing potion
V - use a vision potion
E - draw your weapon
Tab - see a map of the maze, thus far explored

Good luck, brave hero!
```

2.7 Assumptions and Dependencies

Dependencies include Pygame, Pytest, and Python 3.x for development of this project, as well as access to Raycasting code.

3. System Features

3.1 Maze

The game shall generate a maze for transversal by the player.

Description and Priority

Maze is generated with a randomized pathway for the player. Essential objects and exit are placed after the maze pathway is created. There is built-in validation of playable pathways with obtainable required objects. In version 2.0, improvements have been made in the programming code on clarity, reliability and in testing.

This is an essential feature.

Stimulus/Response Sequences

MSR-1: Maze is auto generated after hero selection
MSR-2: Built-in maze validation automatically runs to ensure maze is playable including the ability to obtain required items to win. If a maze is determined to be not valid, another maze is generated until a valid maze is determined.
MSR-3: A new randomized maze is generated if the player loses or wins and desires another challenge.

3.2 Heroes

A new set of diverse heroes are introduced. Each character class has weaknesses and strengths.

Description and Priority

The Player will choose from 3 different heros:

H-1: Warrior – has a Crushing Blow attack which does significant damage to enemies.
H-2: Priestess – has an ability to heal.
H-3: Thief – can perform a surprise attack allowing an extra attack.

Each of the different hero classes have different attack strength and speed values, as well as health points.

This is an essential feature.

Stimulus/Response Sequences

HSR-1: Player selects 1 of 3 hero classes at the beginning of the game.
HSR-2: Upon entering a Dungeon room containing an Enemy, the Player's hero interacts with that Enemy. Player's hero may choose to fight with the Enemy, and may continue doing so, until either character's health reaches 0.

3.3 Enemies

Description and Priority

The Player can encounter and fight 3 different enemies, each with weakness and strengths:

E-1: Ogre – has a shield that makes it difficult for warriors to defeat.
E-2: Gremlin – able to move very fast, rendering thieves' surprise attack useless.
E-3: Skeleton – has a "silence" ability that nullifies healing magics.
E-4: Mean Girl – the boss, guards the pillars and the exits.

The term "Monster" may be used interchangeably with "Enemy", but the latter term recognizes that some Enemies potentially may not be Monsters (except in the metaphorical sense).

Each of the above Enemy types have different attack strength and speed values, as well as amount of health points.

This is a high priority feature.

Stimulus/Response Sequences

ESR-1: Enemies are randomly placed in dungeons during the generation of the game.
ESR-2: Enemies in a dungeon room interact with Hero, if present. enemy persists until either its health reaches 0, at which point it dies.
ESR-3: Enemy strengths and weaknesses are dependent on the Hero class that interacts with the Enemy type.

3.4 Game State Storage

Users have the ability to interrupt game play, save the game state, and resume game play.

Description and Priority

Game state includes, but may not be limited to: enemy characters, player state, inventory, game play tuning factors, and map features. During restore of a saved game, these objects are reinstantiated.

This is a high priority feature.

Stimulus/Response Sequences

This feature is auto-generated at the beginning of the game. Save and load is triggered by the player which additionally triggers changes in the database.

Functional Requirements

Players have the ability to access an inventory of items collected throughout the maze, including healing potions and vision potions. This inventory is stored when the game state is saved, and reinstated as the game is resumed.

4. External Interface Requirements

4.1 User Interfaces

Pygame has been used to develop GUI. This allows the user to move through a maze by clicking or keyboard entry of moving up, down, left, right. Users are also able to see the map of traversed maze, encounter of enemy characters, and user's hero character information.

4.2 Hardware Interfaces

Users input is via a mouse and keyboard and output, or touch-based equivalents. Output through a full-sized graphical display at least 11 inches diagonal. It is not intended for use on smaller mobile devices.

4.3 Software Interfaces

This is a self-contained application and does not interface with software other than its host operating system.

4.4 Communications Interfaces

No communication protocol is required for this single-player, offline game.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

Performance requirements are minimal for this single player lightweight game. The visual display should update within one second of any user action.

5.2 Safety Requirements

The user may encounter bad jokes. Although not required, access to anti-nausea medication is recommended as a preventive measure.

5.3 Security Requirements

There are no security requirements.

5.4 Software Quality Attributes

Interoperability-1: The game should behave roughly the same on all supported operating systems.

Testability-1: A decoupled approach in programming should allow unit-testing of all game internals. Automated unit-testing of graphical features is explicitly excluded.

Reliability-1: The game maze generation shall guarantee that any maze presented to the user can be completed.

Robustness-1: The game should not crash. Game termination should only be by completion of the game via win or loss, or saving of game state for later resumption.

Appendix A: Glossary

Dungeon - Representation of a physical space consisting of a Maze and all contents therein. The Maze itself is not dangerous to the Hero; its contents often are.

Enemy - Opposing non-player characters that the Hero encounters and must defeat.

Entrance - Starting point for the Hero in the Maze. Abuts the perimeter of the Maze.

Exit - It’s your freedom from the Maze. Find this location to escape but only after you have found all 4 pillars. Abuts the perimeter of the Maze.

Health points - Bounded set of units representing a character’s vitality. When a character’s Health points are reduced to zero, the character “dies”.

Health potion - Type of Potion that restores some portion of lost Hero health points.

Hero - Character that represents a user playing the game, traversing a dungeon maze. The character has health points that the user must maintain greater than zero or fails in the game.

Load game - Resume a saved game and continue where you left off, before you were so rudely interrupted.

Maze - A set of interconnected passages, with a single entrance and a single exit. Hero traverses this as part of gameplay.

Monster - Roughly synonymous with Enemy, but typically limited to beasts, phantasms and politicians.

Pillar - One of four pillars that the player must collect before exiting the dungeon to win the game.

Trap - Stationary peril that causes Hero to fall and lose health points.

Potion - Object that Hero picks up. It can be a Health potion or Vision potion.

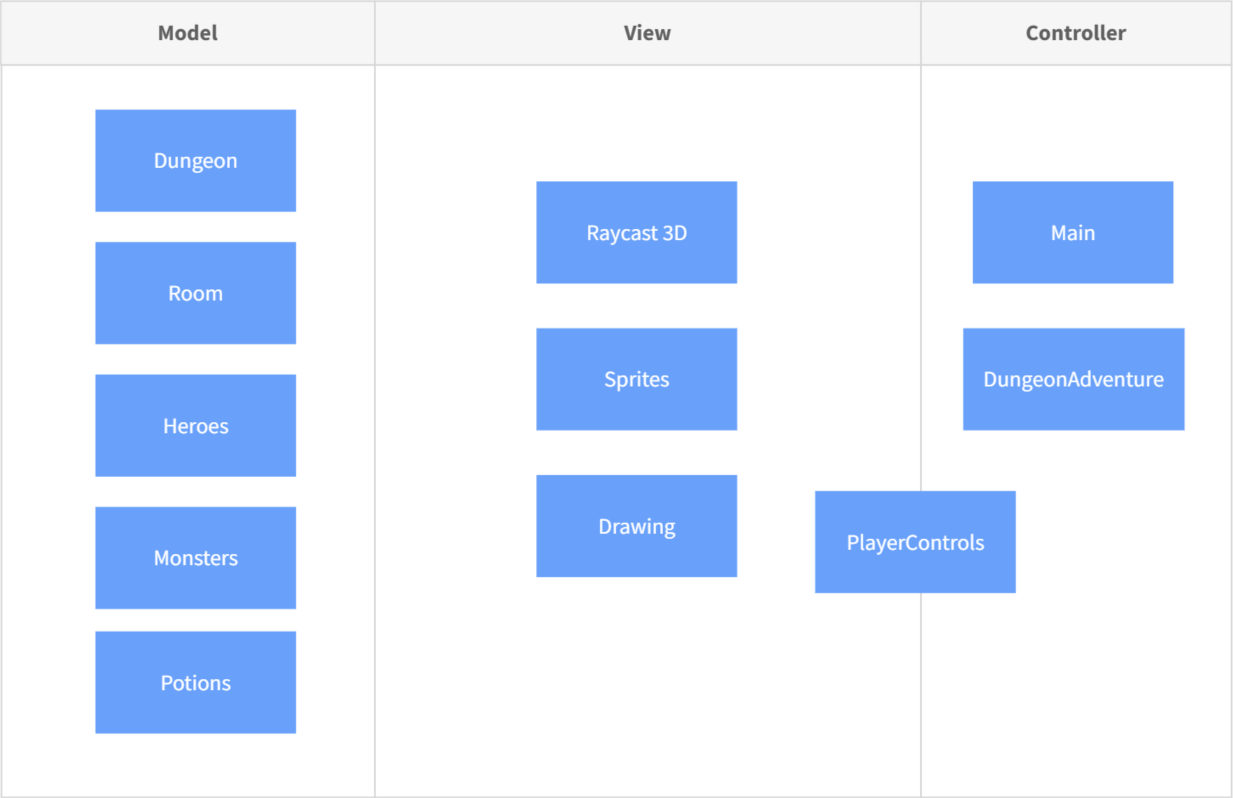
Save game - Save your current location and current progress in the game to resume later.

SRS - Software Requirement Specification.

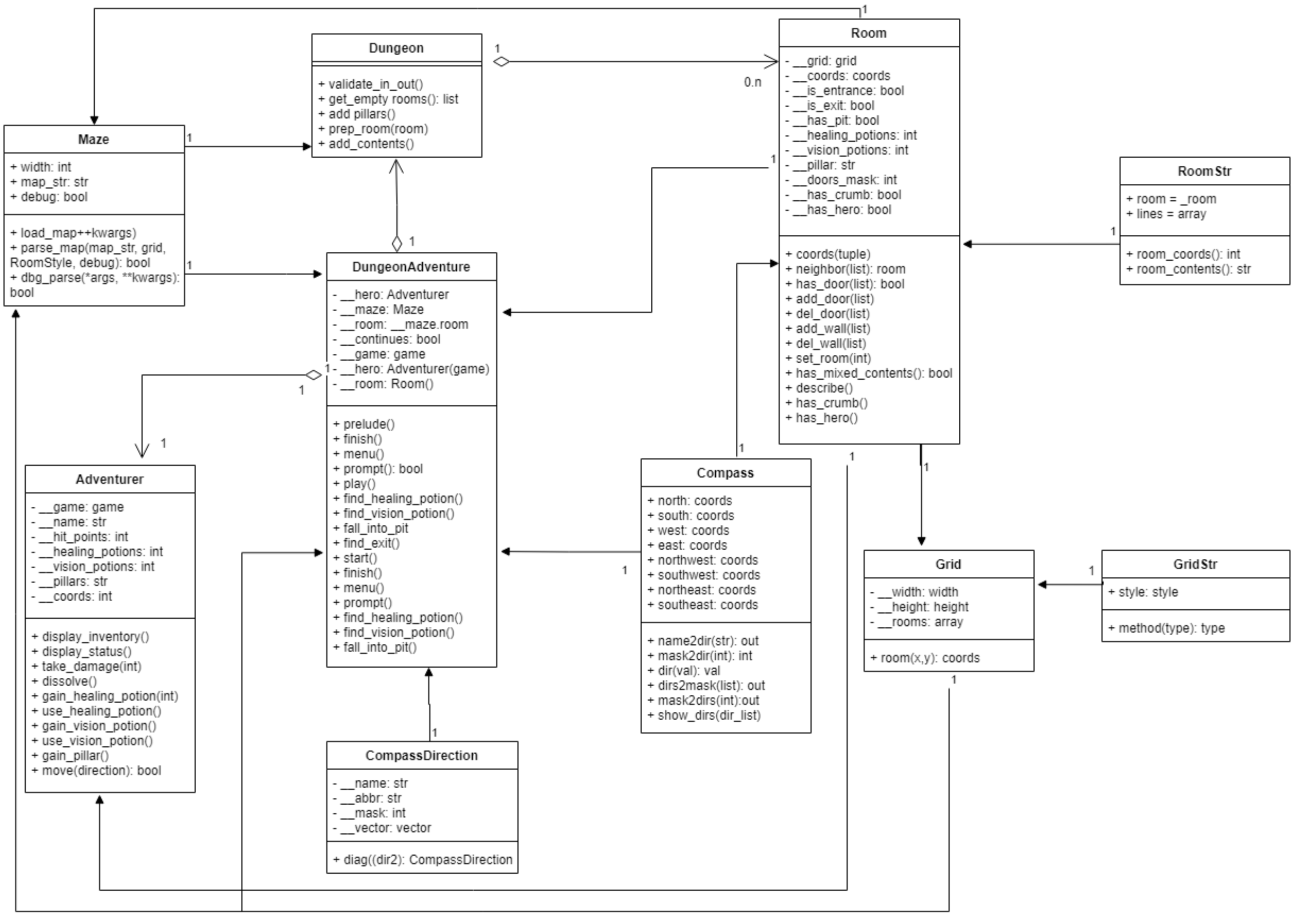
Vision potion - Type of Potion that reveals nearby rooms. This lets the user briefly see the 8 rooms surrounding the current room.

Appendix B: Analysis Models

Model - View - Controller Design



Original UML from dungeon adventure 1.0


Dungeon
+ validate_in_out()
+ get_empty rooms(): list
+ add pillars()
+ prep_room(room)
+ add_contents()
1
0.n
Room
- __grid: grid
- __coords: coords
- __is_entrance: bool
- __is_exit: bool
- __has_pit: bool
- __healing_potions: int
- __vision_potions: int
- __pillar: str
- __doors_mask: int
- __has_crumb: bool
- __has_hero: bool
+ coords(tuple)
+ neighbor(list): room
+ has_door(list): bool
+ add_door(list)
+ del_door(list)
+ add_wall(list)
+ del_wall(list)
+ set_room(int)
+ has_mixed_contents(): bool
+ describe()
+ has_crumb()
+ has_hero()
1
1
Maze
+ width: int
+ map_str: str
+ debug: bool
+ load_map++kwargs)
+ parse_map(map_str, grid, RoomStyle, debug): bool
+ dbg_parse(*args, **kwargs): bool
1
1
RoomStr
+ room = _room
+ lines = array
+ room_coords(): int
+ room_contents(): str
1
DungeonAdventure
- __hero: Adventurer
- __maze: Maze
- __room: __maze.room
- __continues: bool
- __game: game
- __hero: Adventurer(game)
- __room: Room()
+ prelude()
+ finish()
+ menu()
+ prompt(): bool
+ play()
+ find_healing_potion()
+ find_vision_potion()
+ fall_into_pit
+ find_exit()
+ start()
+ finish()
+ menu()
+ prompt()
+ find_healing_potion()
+ find_vision_potion()
+ fall_into_pit()
1
1
1
1
Adventurer
- __game: game
- __name: str
- __hit_points: int
- __healing_potions: int
- __vision_potions: int
- __pillars: str
- __coords: int
+ display_inventory()
+ display_status()
+ take_damage(int)
+ dissolve()
+ gain_healing_potion(int)
+ use_healing_potion()
+ gain_vision_potion()
+ use_vision_potion()
+ gain_pillar()
+ move(direction): bool
1
Compass
+ north: coords
+ south: coords
+ west: coords
+ east: coords
+ northwest: coords
+ southwest: coords
+ northeast: coords
+ southeast: coords
+ name2dir(str): out
+ mask2dir(int): int
+ dir(val): val
+ dirs2mask(list): out
+ mask2dirs(int):out
+ show_dirs(dir_list)
1
1
1
Grid
- __width: width
- __height: height
- __rooms: array
+ room(x,y): coords
1
1
GridStr
+ style: style
+ method(type): type
1
CompassDirection
- __name: str
- __abbr: str
- __mask: int
- __vector: vector
+ diag((dir2): CompassDirection