SE 3XA3: Software Requirements Specification

Team 10, MacLunky
Albert Zhou, zhouj103
Abeer Al-Yasiri, alyasira
Niyatha Rangarajan, rangaran

April 12, 2021

Contents

List of Tables

List of Figures

Table 1: Revision History

Date	**Developer(s)**	**Change**
March 11, 2021	Albert, Abeer, Niyatha	Version 0 made
April 12, 2021	Albert, Abeer, Niyatha	Revision 1 modifications

1 Introduction

Decomposing a system into modules is a commonly accepted approach to developing software. A module is a work assignment for a programmer or programming team (**?**). We advocate a decomposition based on the principle of information hiding (**?**). This principle supports design for change, because the "secrets" that each module hides represent likely future changes. Design for change is valuable in SC, where modifications are frequent, especially during initial development as the solution space is explored.

Our design follows the rules layed out by **?**, as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is used in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

After completing the first stage of the design, the Software Requirements Specification (SRS), the Module Guide (MG) is developed (**?**). The MG specifies the modular structure of the system and is intended to allow both designers and maintainers to easily identify the parts of the software. The potential readers of this document are as follows:

- New project members: This document can be a guide for a new project member to easily understand the overall structure and quickly find the relevant modules they are searching for.
- Maintainers: The hierarchical structure of the module guide improves the maintainers' understanding when they need to make changes to the system. It is important for a maintainer to update the relevant sections of the document after changes have been made.
- Designers: Once the module guide has been written, it can be used to check for consistency, feasibility and flexibility. Designers can verify the system in various ways, such as consistency among modules, feasibility of the decomposition, and flexibility of the design.

The rest of the document is organized as follows. Section **??** lists the anticipated and unlikely changes of the software requirements. Section **??** summarizes the module decomposition that was constructed according to the likely changes. Section **??** specifies the connections between the software requirements and the modules. Section **??** gives a detailed description of the modules. Section **??** includes two traceability matrices. One checks the completeness of the design against the requirements provided in the SRS. The other shows the relation between anticipated changes and the modules. Section **??** describes the use relation between modules.

2 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section **??**, and unlikely changes are listed in Section **??**.

2.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1: The specific hardware and OS on which the software is running.

AC2: The format of the input data the player uses for controls.

AC3: The user controls the size of the game window.

AC4: Adapting new game levels by changing the game screen as the player progress in the game.

AC5: The size and image of game objects.

AC6: The player's speed.

AC7: The speed of the interactive moving objects in the game.

AC8: The player scoring system and health points.

AC9: The player movement and attack mechanisms.

AC10: The winning criteria of the game.

AC11: The player character features such as shape, size, and theme.

2.2 Unlikely Changes

The module design should be as general as possible. However, a general system is more complex. Sometimes this complexity is not necessary. Fixing some design decisions at the system architecture stage can simplify the software design. If these decision should later need to be changed, then many parts of the design will potentially need to be modified. Hence, it is not intended that these decisions will be changed.

UC1: Input/Output devices (Input: File and/or Keyboard, Output: File, Memory, and/or Screen).

UC2: There will always be a source of input data external to the software.

UC3: The behaviour of the game's interactive objects.

UC4: The control options used by the player.

UC5: The types of screen display of the game.

UC6: The input to the scoring system saving procedure.

UC7: The number of players in one game run.

UC8: Adding player movement and game environment sounds.

3 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table **??**. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

M1: Display Module

M2: Input Controls Module

M3: Player Module

M4: Entities Module

1. Entity Module
2. Collectable Module
3. Treasure Module
4. Gold Module
5. Sapphire Module
6. Emerald Module
7. Ruby Module
8. Diamond Module
9. BombPile Module
10. RopePile Module
11. Sign Module
12. Explosion Module
13. Rope Module
14. Throwable Module

15. Chest Module
16. Bomb Module
17. EmptyHand Module
18. Enemy Module
19. Snake Module
20. Spider Module
21. Trap Module
22. ArrowTrap Module
23. Arrow Module
24. Spikes Module
25. Weapon Module
26. Whip Module

M5: Game State Module

M6: Game Level Module

M7: Overlap Module

M8: Gravity Module

Level 1	**Level 2**
Hardware-Hiding Module	Display Input Controls
Behaviour-Hiding Module	Player Entities Entity Collectable Treasure Gold Sapphire Emerald Ruby Diamond BombPile RopePile Gold Gem Sign Explosion Rope Throwable Chest Bomb EmptyHand Enemy Snake Spider Trap ArrowTrap Arrow Spikes Weapon Whip Game State Game Level
Software Decision Module	Overlap Gravity

Table 2: Module Hierarchy

4 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table **??**.

5 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. Also indicate if the module will be implemented specifically for the software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented. Whether or not this module is implemented depends on the programming language selected.

5.1 Hardware Hiding Modules

5.1.1 Display Module M??

Secrets: Loads the ui of the game involving the display of the map, entities placed on the map and the player that the user controls.

Services: The input takes all game entities and player images to be displayed. Output shows a visual rendition of the game after various in-game interactions. Adjusts on screen images to represent a game object in motion.

Implemented By: Original project

5.1.2 Input Controls Module M??

Secrets: The mechanism to connect user keyboard inputs with object creation/interaction and movement on the game map screen.

Services: The output translates (for a given frame rate) the game object's movement that is their game map coordinates based on the key provided. The output could also generate map entities that subsequently interact with the player. Finally the key inputs can also allow the player to have access to defense mechanism while interacting with other map objects.

Implemented By: Original project

5.2 Behaviour-Hiding Module

5.2.1 Player Module M??

Secrets: The module containing the state and characteristics of the player character.

Services: Provides movement, interactions with entities, resource tracking, and resource use for the player character.

Implemented By: Original Project

5.2.2 Entities Module M??

Secrets: The module containing the state data of all entities found in the game.

Services: Provides the data of all types of entities found in the game.

Implemented By: Original project

5.2.2.1 Entity Module M4.??

Secrets: Entity functions.

Services: Provides functions for entity entities.

Implemented By: Albert Zhou

5.2.2.2 Collectable Module M4.??

Secrets: Entity functions.

Services: Provides functions for collectable entities.

Implemented By: Albert Zhou

5.2.2.3 Treasure Module M4.??

Secrets: Entity functions.

Services: Provides functions for treasure entities.

Implemented By: Albert Zhou

5.2.2.4 Gold Module M4.??

Secrets: Gold functions.

Services: Provides functions for gold entities.

Implemented By: Albert Zhou

5.2.2.5 Sapphire Module M4.??

Secrets: Sapphire functions.

Services: Provides functions for sapphire entities.

Implemented By: Albert Zhou

5.2.2.6 Emerald Module M4.??

Secrets: Emerald functions.

Services: Provides functions for emerald entities.

Implemented By: Albert Zhou

5.2.2.7 Ruby Module M4.??

Secrets: Ruby functions.

Services: Provides functions for ruby entities.

Implemented By: Albert Zhou

5.2.2.8 Diamond Module M4.??

Secrets: Diamond functions.

Services: Provides functions for diamond entities.

Implemented By: Albert Zhou

5.2.2.9 BombPile Module M4.??

Secrets: BombPile functions.

Services: Provides functions for bombPile entities.

Implemented By: Albert Zhou

5.2.2.10 RopPile Module M4.??

Secrets: RopPile functions.

Services: Provides functions for RopPile entities.

Implemented By: Albert Zhou

5.2.2.11 Sign Module M4.??

Secrets: Sign functions.

Services: Provides functions for sign entities.

Implemented By: Original project

5.2.2.12 Explosion Module M4.??

Secrets: Explosion functions.

Services: Provides functions for explosion entities.

Implemented By: Albert Zhou

5.2.2.13 Rope Module M4.??

Secrets: Rope functions.

Services: Provides functions for rope entities.

Implemented By: Albert Zhou

5.2.2.14 Throwable Module M4.??

Secrets: Throwable functions.

Services: Provides functions for throwing and moving entities.

Implemented By: Albert Zhou

5.2.2.15 Chest Module M4.??

Secrets: Chest functions.

Services: Provides functions for chest entities.

Implemented By: Albert Zhou

5.2.2.16 Bomb Module M4.??

Secrets: Bomb functions.

Services: Provides functions for bomb entities.

Implemented By: Albert Zhou

5.2.2.17 EmptyHand Module M4.??

Secrets: EmptyHand functions.

Services: Provides functions for emptyHand entities.

Implemented By: Albert Zhou

5.2.2.18 Enemy Module M4.??

Secrets: Enemy functions.

Services: Provides functions for enemy entities.

Implemented By: Niyatha Rangarajan

5.2.2.19 Snake Module M4.??

Secrets: Snake functions.

Services: Provides functions for snake entities.

Implemented By: Niyatha Rangarajan

5.2.2.20 Spider Module M4.??

Secrets: Spider functions.

Services: Provides functions for spider entities.

Implemented By: Niyatha Rangarajan

5.2.2.21 Trap Module M4.??

Secrets: Trap functions.

Services: Provides functions for trap entities.

Implemented By: Abeer Al-Yasiri

5.2.2.22 ArrowTrap Module M4.??

Secrets: ArrowTrap functions.

Services: Provides functions for arrowTrap entities.

Implemented By: Abeer Al-Yasiri

5.2.2.23 Arrow Module M4.??

Secrets: Arrow functions.

Services: Provides functions for arrow entities.

Implemented By: Abeer Al-Yasiri

5.2.2.24 Spikes Module M4.??

Secrets: Spikes functions.

Services: Provides functions for spikes entities.

Implemented By: Abeer Al-Yasiri

5.2.2.25 Weapon Module M4.??

Secrets: Weapon functions.

Services: Provides functions for weapon entities.

Implemented By: Albert Zhou

5.2.2.26 Whip Module M4.??

Secrets: Whip functions.

Services: Provides functions for whip entities.

Implemented By: Albert Zhou

5.2.3 Game State Module M??

Secrets: The module for provoking changes in the game with respect to time..

Services: Provides game initialization and timing for the player and entities.

Implemented By: Original project

5.2.4 Game Level Module M??

Secrets: The module responsible for creating and storing game levels.

Services: Provides level generation from map files containing the types and locations of all blocks and entities.

Implemented By: Original project

5.3 Software Decision Module

5.3.1 Overlap Module M??

Secrets: The mechanism for detecting the collision between two interactive objects in the game.

Services: Detects if two objects are colliding at the same position and time of action. The module takes static inputs of objects state in the game and returns a Boolean of the collision state of the two input objects. The output of the module is used to make game state change decisions to the interacted objects.

Implemented By: Albert Zhou

5.3.2 Gravity Module M??

Secrets: The control mechanism of representing the affect of gravity in the game.

Services: Affects how interactive objects in the game descends in open space. It determines the speed at which objects fall from various heights in the game. The effects of the modules on the game's objects are used in other modules to make state change decisions for the effected objects.

Implemented By: Original project

6 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	**Modules**
R1-R2	M??
R3-R5	M??
R6-R15,R34	M??, M??, M??, M??
R16-R19	M4.M??, M4.M??, M??, M??, M4.??
R20-R24	M??, M4.M??, M??
R25-R27	M??, M4.M??, M??
R28-R30	M??, M??
R31-R33	M??
R35-R38,R49	M4.M??, M??
R39-R48, 91	M4.M?? - 7, M4.M??
R50-54	M4.M??, M??
R55-62	M4.M??, M??
R39-R48	M??, M??
R63	M4.M??
R64-R71, 90	M4.M??, M4.M??, M??
R72-R73	M4.M??
R74-R77	M??, M??
R78-R84	M??
R85-R87	M4.M??
R88	M4.M??
R89	M4.M??

Table 3: Trace Between Requirements and Modules

Req.	**Modules**
NFR1	All modules
NFR2	All modules
NFR3	M??
NFR4	M??
NFR5	M??
NFR6	M??
NFR7	M??
NFR8	M??
NFR9	M??
NFR10	All modules
NFR11	All modules
NFR12	All modules
NFR13	All modules
NFR14	All modules
NFR15	All modules
NFR16	All modules
NFR17	All modules
NFR18	All modules
NFR19	M??
NFR20	M??
NFR21	All modules
NFR22	All modules
NFR23	M??
NFR24	M??

Table 4: Trace Between Non Functional Requirements and Modules

AC	Modules
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??
AC??	M??

Table 5: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure **??** illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

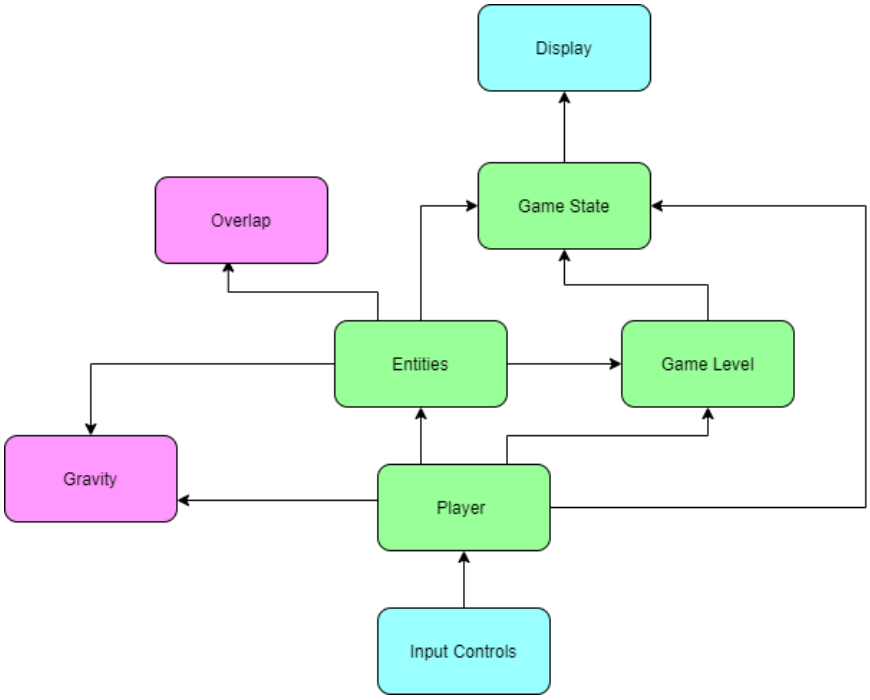

Figure 1: Use hierarchy among modules

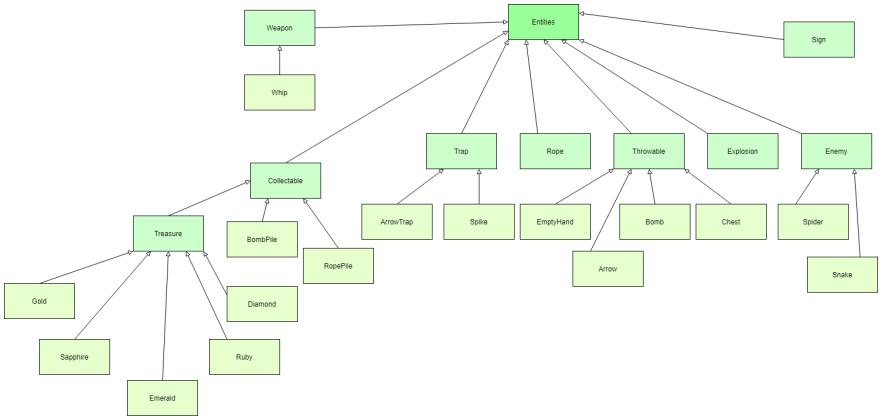

Figure 2: Use hierarchy among entity modules