SE 3XA3: Software Requirements Specification
Sokoban

Team # 13, The Box Group
Gurpartap Kaler and kalerg1
Sagar Thomas and thomas12
Freddie Yan and yanz20

November 9, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
Nov 9, 2018	0.0	Revision 0 Initial Edit
Date 2	1.0	Notes

1 Introduction

1.1 Overview

The Box Group's Sokoban project is a re-implementation of an open-source video game that was originally found on GitHub. The original project was implemented in Java, and the re-implementation will be in Python. Sokoban is a logic based video game, that challenges the user making them solve a maze, by moving boxes into their designated areas in the maze.

1.2 Context

According to **?**, this document is the Module Guide (MG), which is created after the Software Requirements Specification (SRS). The purpose of the MG is to provide a modular decomposition of the system, showing the modular structure. The MG also shows how the system will meet both functional and non-functional requirements specified in the SRS.

1.3 Design Principles

The design principles that will be used throughout the MG include information hiding and encapsulation. Furthermore, we would like to include a hierarchy in our modules, that contain no cycles, have high cohesion, and low coupling.

- Information Hiding is the principle that each module should hide a design decision that is apart of the system.
- Encapsulation is the principle that the services a module provides should remain consistent, regardless of how it is implemented.
- No cycles is the principle that we will not have two modules using each other (Module 1 uses Module 2, but Module 2 should not use Module 1).
- High Cohesion is the principle that the components of a module are closely related.
- Low Coupling is the principle that a module is not strongly dependent on other modules

1.4 Document Structure

- Section **??** lists anticipated and unlikely changes of requirements.
- Section **??** summarizes the module decomposition.
- Section **??** specifies the connections between the SRS and the modules.
- Section **??** gives detailed descriptions of the modules.

- Section **??** includes two traceability matrices: One that checks the completeness of the design against the functional requirements provided in the SRS; Another that shows the relation between anticipated changes and the modules.
- Section **??** shows the uses hierarchy of the modules.

2 Anticipated and Unlikely Changes

2.1 Anticipated Changes

AC1: The format of the graphical user interface (GUI).

AC2: The theme (character, boxes, maze art) of Sokoban.

AC3: The format of the output data from modules.

AC4: The implementation of the Character, Box, and Maze objects.

AC5: The specific hardware on which the software is running.

AC6: The implementation of "undo" button

AC7: Default settings for input

2.2 Unlikely Changes

UC1: Input/Output devices (Input: Keyboard, Mouse / Output: Memory, Screen).

UC2: The amount of users using the system at one time (assume only one person uses Sokoban at a time)

UC3: The purpose and objective of Sokoban

UC4: The installation method of the video game (there is storage available on the local machine)

UC5: The library PyGame used to implement the software.

3 Module Hierarchy

M1: Hardware-Hiding Module

M2: Behaviour-Hiding Module: SceneManager Module

M3: Behaviour-Hiding Module: Scene Module

M4: Behaviour-Hiding Module: Character Module

M5: Behaviour-Hiding Module: Box Module

M6: Behaviour-Hiding Module: Game Module

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	SceneManager Module Scene Module Character Module Box Module Game Module
Software Decision Module	N/A (No generic modules)

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. The functional requirements will be satisfied throughout the modules. The Main class is the class that will connect all methods together and will be the only way the program is executed. Appearance and navigation requirements will be satisfied through the Scenes and SceneManager Module, as this is the module that deals with GUI. Usability requirements will be satisfied through the Character Module, as this is the module that converts user input into Sokoban. The Box Method is required for creating boxes in the maze, and ensuring their movement is valid. The Game Method is required for making sure the in-game buttons are available for users. The Game and Scenes modules are used to check if the user won the game, and continue the progress through the game.

Non-functional requirements will also be satisfied through the modules specified above. Look and feel requirements will be implemented in the Scenes and ScenesManager module for the main menu and Game module for the game. This will ensure that the user experiences the game as required. Usability and Humanity requirements will be implemented by the Game module. The game will be easy to install for the user because it will all be in a package. Performance requirements are key to the program, and will be fulfilled throughout all modules, ensuring that each part of the game fulfills the requirements. Operational and Environmental requirements will be implemented in the Scenes and ScenesManager module. This will make sure that the program changes states accordingly. Cultural, Political, Health and Safety, will be implemented, by not using any external images that may offend the user, and by making sure colors are not too bright.

5 Module Decomposition

5.1 Hardware Hiding Module

Secrets: The implementation of the video game.

Services: Serves as an interface between software and hardware. It allows the users system to communicate with the software through input and output devices.

Implemented By: Operating System

5.2 Behaviour Hiding Module

Secrets: Required Behaviours

Services: Visible behaviour of application. It is an interpreter between Hardware Hiding Module and Software Decision Module. The modules contained within the Behaviour Hiding Module are implemented according to the SRS. This ensures that the actual video game application will behave as it was described in the SRS.

Implemented By: N/A

5.2.1 SceneManager Module

Secrets: Scene Manager

Services: Changes the scene and menu of the video game.

Implemented By: Python Library

5.2.2 Scene Module

Secrets: Scenes

Services: Stores the scene and menu of the video game.

Implemented By: Python Library

5.2.3 Character Module

Secrets: Character Model

Services: Allows the user to control the characters state based off their input.

Implemented By: Python Library

5.2.4 Box Module

Secrets: Box

Services: Stores and changes the state of the box objects in the maze.

Implemented By: Python Library

5.2.5 Game Module

Secrets: Game

Services: Stores and changes the states of the game according to user input.

Implemented By: Python Library

5.3 Software Decision Module

Secrets: Data Structures

Services: The data structures that store information for the application, without user input.

Implemented By: N/A

6 Traceability Matrix

6.1 Modules and Requirements

Req.	Modules
FS-CM-1	M??
FS-CM-2	M??
FS-BM-1	M??
FS-BM-2	M??
FS-WC-1	M??, M??
FS-MM-1	M??, M??
FS-MM-2	M??, M??
FS-MM-3	M??, M??
FS-MM-4	M??, M??
FS-GB-1	M??
FS-GB-2	M??, M??, M??
FS-GB-3	M??, M??, M??

Table 3: Trace Between Requirements and Modules

6.2 Modules and Anticipated Changes

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

Table 4: Trace Between Anticipated Changes and Modules

7 Use Hierarchy Between Modules

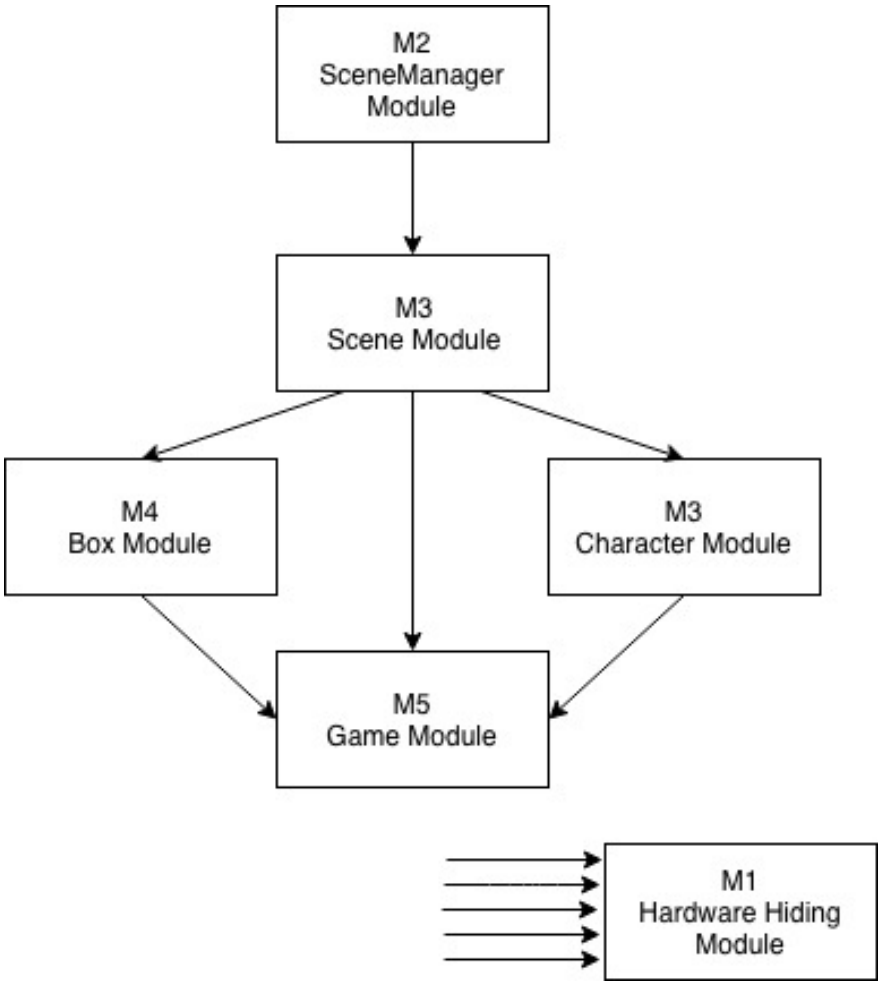

Figure 1: Use hierarchy among modules