SE 3XA3: Test Plan
Sokoban

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

October 26, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
October 5th	Revision 0	Software Requirements Specification
October 16th	Revision 0	Tests for Proof of Concept
October 26th	Revision 0	Test Plan

1 General Information

1.1 Purpose

The purpose of the Test Plan is to make sure that our game, Sokoban, is tested correctly and thoroughly according to the requirements. Testing will also make sure that the project is implemented correctly from the specification.

1.2 Scope

The test plan focuses on testing the behaviours of our software, which ensures all requirements are met. Also, by introducing test plan, members in test team can schedule their testing activities in a logical way.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
PoC	Proof of concept
SRS	Software Requirements Specification
CI	Continuous Integration

Table 3: **Table of Definitions**

Term	**Definition**
Sokoban	Name given to our software

1.4 Overview of Document

The Sokoban game is a Python version of the existing implementation which is in Java. With considering the previous SRS and PoC, this document will specify what testing method will be used and what functionality of Sokoban will be tested, in order to make our software satisfactory.

2 Plan

2.1 Software Description

Sokoban is a maze game created by using the PyGame package in Python. In this game, players control a character's movement for the purpose of pushing boxes into their designated cells in the maze. The implementation will be completed in Python.

2.2 Test Team

The test team consists of members from group 13 of SFWRENG 3XA3 (Fall 2018): Gurpartap Kaler, Sagar Thomas and Freddie Yan.

2.3 Automated Testing Approach

We will be making use of Gitlab's CI service to automatically run our tests for every commit or merge into the master branch. If the test fails, the commit or merge will be rejected. This will be really helpful, ensuring that tests are run regularly and throughout our whole development process.

2.4 Testing Tools

Our unit tests and mocking will be implemented using the PyUnit testing framework.

2.5 Testing Schedule

Table 4: **Testing Schedule**

Date	Task	Team Member
October 26	Character's Movement	Gurpartap Kaler
October 26	Box's Movement	Sagar Thomas
November 2	Game Winning Condition	Freddie Yan
November 2	Main Menu	Freddie Yan
November 9	In-Game Buttons	Gurpartap Kaler
November 16	Non-Functional Requirements	Sagar Thomas

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 User Input

Character's Movement

1. FS-CM-1

   Type: Functional, Dynamic, Manual

   Initial State: The character is located in an empty maze without any obstructions around the character.

   Input: Keys {UP, DOWN, LEFT, RIGHT}

   Output: The character moves {UP, DOWN, LEFT, RIGHT} by one unit cell.

   How test will be performed: We will place the character into a maze without any blocks and boxes, then we will move the character by pressing all possible keys {UP, DOWN, LEFT, RIGHT} in that order. An observation will be made on if the characters movement corresponds to the correct input given, and if each movement was only one cell.

2. FS-CM-2

   Type: Functional, Dynamic, Manual

   Initial State: A maze block is placed {ABOVE, BELOW, LEFT, RIGHT} the character.

   Input: Keys {UP, DOWN, LEFT, RIGHT}

   Output: The character gets stuck and performs no changes of positions.

   How test will be performed: We will surround the character with maze blocks, so only one cell is available. We will test if the character can move in any of the following directions {UP, DOWN, LEFT, RIGHT}, and observe the results.

Box's Movement

1. FS-BM-1

   Type: Functional, Dynamic, Manual

   Initial State: A box is placed {ABOVE, BELOW, LEFT, RIGHT} the character without any obstructions around the character.

   Input: Keys {UP, DOWN, LEFT, RIGHT}

   Output: Character moves in the specified location with the box {UP, DOWN, LEFT, RIGHT} by one unit cell.

   How test will be performed: We will place the box first above the character and press UP key, see if the character and box both move up by one cell. Then we try to place the box below the character and press DOWN key. Later we place the box left to the character and press LEFT key, finally place the box right to the character and press RIGHT key, including observation after each key pressed.

2. FS-BM-2

   Type: Functional, Dynamic, Manual

   Initial State: A box is placed {ABOVE, BELOW, LEFT, RIGHT} the character, and a maze block is placed {ABOVE, BELOW, LEFT, RIGHT} the box.

   Input: Keys {UP, DOWN, LEFT, RIGHT}

   Output: The character and box get stuck and perform no changes of positions.

   How test will be performed: We will surround the character with boxes. After this, we will surround the boxes by maze blocks. We will test if the character and box can move in any of the following directions {UP, DOWN, LEFT, RIGHT}, and observe the results.

Game Winning Condition

1. FS-WC-1

Type: Functional, Dynamic, Manual

Initial State: Four boxes are placed between character and a designated cell. The boxes surround the character.

Input: Keys {UP, DOWN, LEFT, RIGHT}

Output: The game should trigger a winning condition, and cause the user to go to the next level or displays a win screen.

How test will be performed: We will control the character to push the boxes into their designated cells. After all boxes go into their designated cells, Sokoban will go to the next level automatically or display the win screen if levels end.

3.1.2 Navigation

Main Menu

1. FS-MM-1

   Type: Functional, Dynamic, Manual

   Initial State: Main Menu screen.

   Input: Users click on Start.

   Output: Sokoban goes to the first level.

   How test will be performed: The software will be opened, then we click on the Start which is displayed in the menu, check if it turns to the first level of the game.

2. FS-MM-2

   Type: Functional, Dynamic, Manual

   Initial State: Main Menu screen.

   Input: Users click on Tutorial

   Output: Sokoban goes to the Tutorial menu.

   How test will be performed: The software will be opened, then we click on the Tutorial which is displayed in the menu. We will then observe if Sokoban takes the user to the Tutorial menu.

3. FS-MM-3

   Type: Functional, Dynamic, Manual

   Initial State: Main Menu screen.

   Input: Users click on Options

   Output: Sokoban goes to the Options menu.

   How test will be performed: The software will be opened, then we click on the Options which is displayed in the menu. We will then observe if Sokoban takes the user to the Options menu.

4. FS-MM-4

   Type: Functional, Dynamic, Manual

   Initial State: Main Menu screen.

   Input: Users click on Exit Game

   Output: Sokoban terminates.

   How test will be performed: The software will be opened, then we click on the Exit Game which is displayed in the menu. We will then observe if Sokoban terminates.

In-Game Buttons

1. FS-GB-1

   Type: Functional, Dynamic, Manual

   Initial State: After the character moves at least one cell.

   Input: Users click on Undo.

   Output: Character returns to the previous cell.

   How test will be performed: We will start a new level, move the character randomly, then click on the Undo, check if the character return to the previous position.

2. FS-GB-2

   Type: Functional, Dynamic, Manual

   Initial State: The user has clicked Start on the main menu, and is placed in the first level of Sokoban.

   Input: Users click on Exit.

   Output: The Main Menu Screen

   How test will be performed: The user will be exited from the game, and return to the Main Menu.

3. FS-GB-3

   Type: Functional, Dynamic, Manual

   Initial State: The user has clicked Start on the main menu, and is placed in the first level of Sokoban.

   Input: Users click on Mute music.

   Output: The music for Sokoban is muted.

   How test will be performed: We will click on the mute menu when in a maze on Sokoban. An observation will be made to check if the music is muted.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel

1. NF-L1

   Type: Static, Manual

   Initial State: When the game enters the splash screen

   Input/Condition: Opening the game

   Output/Result: A splash screen will show with the game title, developer logo and name, as well as the license

   How test will be performed: The test will be performed visually by making sure the splash screen shows when the game is started up.

2. NF-L2

   Type: Dynamic, Manual

   Initial State: After the game starts up, on the main menu screen

   Input: Clicking the ”Start” button

   Output: The game will load up the first level.

   How test will be performed: We will click the start button and observe whether the game will start up the first level

3. NF-L3

   Type: Dynamic, Manual

   Initial State: Any state after the game's initial start

   Input: User drags any edge of the game window or the maximize button is clicked

   Output: The game window will resize according to the final coordinates of the mouse drag or the game will expand to full screen upon clicking the maximize button.

   How test will be performed: The test will be performed visually by confirming that the window resizes by mouse drag and that the window maximizes when the button is clicked. This should be the case for all supported platforms. (Windows, Linux, MacOS)

4. NF-L4

   Type: Dynamic, Manual

   Initial State: Any state after the game's initial start state

   Input: Completion of game levels

   Output: The game's color scheme will change to darker themes as the levels progress.

   How test will be performed: The test will be performed visually by observing that the colors do get darker as the game progresses.

5. NF-L5

   Type: Dynamic, Manual

   Initial State: Any state after the game's initial start state

   Input: Completion of game levels

   Output: the character's clothing color will change as the color theme of the game changes.

   How test will be performed: The test will be performed visually by observing that the the character's clothing color changes alongside the color theme of the game.

3.2.2 Usability and Humanity

1. NF-U1

   Type: Dynamic, Manual

   Initial State: Any state after game's initial start.

   Input: Any mouse input, any input from arrow keys/hotkeys on the keyboard.

   Output: Any response from the game using the input methods

   How test will be performed: The test will be performed by manually checking to make sure that the mouse and keyboard are all what the user needs to play the game.

2. NF-U2

   Type: Manual, Static

   Initial State: Any state of the game

   Input: N/A - any operation of the game

   Output: English text that displays on the screen

   How test will be performed: The test will be performed by visually checking to make sure that there is minimal use of words in the game.

3. NF-U3

   Type: Manual, Static

   Initial State: Any state after game's initial start

   Input: N/A

   Output: Graphics and pictures found in any state of the game.

   How test will be performed: The test will be performed visually by checking that there is a good amount of graphics and pictures in each state of the game.

4. NF-U4

   Type: Dynamic, Manual

   Initial State: Main menu, user has the option to start a tutorial

   Input: Mouse and Keyboard interaction to get through the tutorial

   Output: Completion of tutorial and return to main menu

   How test will be performed: The test will be performed manually by going through the tutorial and making sure that everything is functional.

3.2.3 Performance

1. NF-P1

   Type: Dynamic, Automated

   Initial State: Any game level state

   Input: Keyboard input for movement.

   Output: Character moves on the screen.

   How test will be performed: The game will start a timer and emulate key strokes. If the character does not move to its respective location within a specified time, the test fails.

2. NF-P2

   Type: Dynamic, Automated

   Initial State: Any state of the game

   Input: Any input

   Output: Any output by the game

   How test will be performed: The test will be performed by having a watcher keep track of the main game thread to make sure it is always responding. If the game becomes unresponsive, the watcher will time out and the test fails.

3. NF-P3

   Type: Dynamic, Automated

   Initial State: Any game state

   Input: Key strokes

   Output: Various responses based on the input

   How test will be performed: the test will be performed by emulating every combination of valid key strokes that are defined in the game and making sure the result is as expected.

4. NF-P4

   Type: Dynamic, Automated

   Initial State: Any state of the game

   Input: Key stokes

   Output: Various responses based on the input

   How test will be performed: The test will be performed by emulating key strokes that are not considered valid strokes by the game. If the game responds to any of these key strokes, the test will fail.

5. NF-P5

Type: Dynamic, Manual

Initial State: N/A

Input: N/A

Output: Uninstalling and installing the game

How test will be performed: The test will be performed manually by attempting to uninstall and re-install the game on the same device and make sure it is still functioning. The test will be run on all supported platforms. (Windows, Linux, MacOS)

3.2.4 Operational and Environmental

1. NF-O1

   Type:Manual, Static

   Initial State: Any state of the game.

   Input: Any input.

   Output: Any output respective to the input.

   How test will be performed: The test will be performed by manually running the game on different types of computers (Laptops, Desktops, Hybrids) and making sure the game runs as intended.

2. NF-O2

   Type:Manual, Static

   Initial State: Any state of the game.

   Input: Any input.

   Output: Any output respective to the input.

   How test will be performed: The test will be performed by manually running the game on all supported operating systems (Windows, Linux, MacOS) and making sure it runs as expected.

3. NF-O3

   Type:Manual, Static

Initial State: Any state of the game.

Input: Any input.

Output: Any output respective to the input.

How test will be performed: The test will be performed by manually attempting to install this game on all popular internet browsers and ensure that the game is functional after install.

4. NF-O4

   Type:Manual, Static

   Initial State: Any state of the game.

   Input: Any input.

   Output: Any output respective to the input.

   How test will be performed: The test will be performed by manually installing the game from the package we create and make sure the game is functional after install.

5. NF-O5

   Type:Manual, Static

   Initial State: Any state of the game.

   Input: Any input.

   Output: Any output respective to the input.

   How test will be performed: The test will be performed by manually making sure that newer updates do not break the existing functionality by ensuring the other tests that we have created (Unit tests etc.) pass as expected.

3.2.5 Cultural and Political

1. NF-CP1

   Type: Manual, Static

   Initial State: Any game state.

Input: N/A

Output: N/A

How test will be performed: This test will be performed by manually checking each state of the game for any offensive text, images or media.

2. NF-CP2

   Type: Manual, Static

   Initial State: Any game state.

   Input: N/A

   Output: N/A

   How test will be performed: This test will be performed by manually making sure that there is a disclaimer found in the game if it does use any possibly political/cultural media.

3.2.6 Health and Safety

1. NF-HS1

   Type: Manual, Static

   Initial State: Any game state.

   Input: N/A

   Output: N/A

   How test will be performed: The test will be performed by manually making sure that the color theme in the game uses colors that are easy on the eyes.

2. NF-HS2

   Type: Manual, Static

   Initial State: Any game state.

   Input: N/A

   Output: N/A

How test will be performed: The test will be performed by manually making sure that there is no violence in the game.

3.3 Traceability Between Test Cases and Requirements

A traceability matrix will be used to record the relations between test cases and requirements.According to the requirements indicated in SRS, we will use the table below to record in order to ensure every requirement is tested.

Requirement	PW	UC1	UC2	UC3	...
FR1					
FR2					
...					
FR15					
DR1					
NF1					
NF2					
...					
NF41					

4 Tests for Proof of Concept

PoC testing will be focused on whether we are able to perform simple logic of the game. This will include the character's basic movement and the box's basic movement. Furthermore, we hope to ensure that character and box, will collide with the walls, and are unable to escape the indicated area. Lastly, we would like to create a winning condition for the game, by creating designated areas for boxes.

4.1 Game Logic

4.1.1 Character's Movement

1. FS-GL-CM-1

   Type: Functional, Dynamic, Manual

Initial State: The character is located in a maze.

Input: Keys {UP, DOWN, LEFT, RIGHT}

Output: The character moves {UP, DOWN, LEFT, RIGHT} by one unit cell.

How test will be performed: We will place the character into a maze without any blocks and boxes, then we will move the character by pressing all possible keys {UP, DOWN, LEFT, RIGHT} in that order. We will observe if the character moved only one cell, and if they move in the correct direction.

4.1.2 Box's Movement

1. FS-GL-BM-1

   Type: Functional, Dynamic, Manual

   Initial State: A box is placed {ABOVE, BELOW, LEFT, RIGHT} the character.

   Input: Keys {UP, DOWN, LEFT, RIGHT}

   Output: Character moves in the specified location with the box {UP, DOWN, LEFT, RIGHT} by one unit cell.

   How test will be performed: We will surround the character with boxes. We will move the character in the {UP, DOWN, LEFT, RIGHT} by one unit cell. We will observe if the box is also moved in the specified direction by one unit cell.

4.1.3 Collision Logic

1. FS-GL-CL-1

   Type: Functional, Dynamic, Manual

   Initial State: A maze block is placed {ABOVE, BELOW, LEFT, RIGHT} the character.

   Input: Keys {UP, DOWN, LEFT, RIGHT}

   Output: The character gets stuck and performs no changes of positions.

How test will be performed: We will surround the character with maze blocks, so only one cell is available. We will test if the character can move in any of the following directions {UP, DOWN, LEFT, RIGHT}, and observe if the character moves any cells.

2. FS-GL-CL-2

   Type: Functional, Dynamic, Manual

   Initial State: A box is placed {ABOVE, BELOW, LEFT, RIGHT} the character, and a maze block is placed {ABOVE, BELOW, LEFT, RIGHT} the box.

   Input: Keys {UP, DOWN, LEFT, RIGHT}

   Output: The character and box get stuck and perform no changes of positions.

   How test will be performed: We will surround the character with boxes. After this, we will surround the boxes by maze blocks. We will test if the character and box can move in any of the following directions {UP, DOWN, LEFT, RIGHT}, and observe if the character and/or box move any cells.

4.1.4 Winning Condition

1. FS-GL-WC-1

   Type: Functional, Dynamic, Manual

   Initial State: One box is placed between character and a designated cell

   Input: Keys {UP, DOWN, LEFT, RIGHT}

   Output: The game should trigger a winning condition and display a win prompt.

   How test will be performed: We will control the character to push the boxes into their designated cells. After all boxes go into their designated cells, Sokoban will exit, and display a win prompt in the terminal.

5 Comparison to Existing Implementation

There are 8 tests that compare Sokoban to the existing implementation of the game. Refer to the following tests:

- test FS-GL-CM-1 in Tests for Proof of Concept, which tests basic character movement
- test FS-GL-BM-1 in Tests for Proof of Concept, which tests basic block movement
- test FS-GL-CL-1 in Tests for Proof of Concept, which tests collision mechanics between the character and maze block
- test FS-GL-CL-2 in Tests for Proof of Concept, which tests collision mechanics between the box and maze block
- test FS-GL-WC-1 in Tests for Proof of Concept, which tests the game winning condition
- test NF-P1 in Tests for Non-Functional Requirements, which tests latency for key strokes
- test NF-P4 in Tests for Non-Functional Requirements, which tests to make sure unregistered key strokes have no effect on the game
- test NF-O1 in Tests for Non-Functional Requirements, which tests to make sure that the game can run on all supported operating systems (Windows, Linux, MacOS)

6 Unit Testing Plan

The PyUnit testing framework will be used to perform unit testing for Sokoban.

6.1 Unit testing of internal functions

In order to test internal functions of Sokoban, we can test all methods in the implementation that have return values. This will involve taking all methods from Sokoban, providing them with input, and measuring/checking their output values. We will create a series of unit tests to check the output of these methods, and see if they give the correct values. All unit tests will provide internal functions with correct inputs, and provide inputs that generate exceptions. For example, passing a key other than {UP, DOWN, LEFT, RIGHT}. Sokoban will not need any stubs or drivers to be uni tested. Coverage metrics will be used to determine how much of the code is covered by unit tests. The goal is to cover at least 75% of the program, to ensure all functions are tested correctly.

6.2 Unit testing of output files

N/A, Sokoban does not create or produce any output files.