SE 3XA3: Test Plan
Finite State Machine Simulator

Team #16, NonDeterministic Key
Anhao Jiao (jiaoa3)
Kehao Huang (huangk53)
Xunzhou Ye (yex33)

11 March 2022

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
11 March 2022	1.0	Revision 0
12 April 2022	2.0	Revision 1

Contents

1 General Information

1.1 Purpose

The purpose of testing this software system is to help in finalizing the software application against functional and non-functional requirements. It is also intended that testing will help find defects created by programmer while developing the software.

1.2 Scope

This document provides a detailed schedule for the testing activities of the software system Finite State Machine Simulator. It also contains a description of the intended testing approach.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
FSM	Finite State Machine
FSMS	Finite State Machine
PoC	Proof of Concept
SRS	Software Requirements Specification

1.4 Overview of Document

In this document, a general plan for the testing activities of is provides. In addition, a system test description is included to provide an overview of the desired testing area and the approach of testing against the functional requirements and non-functional requirements.

Table 3: **Table of Definitions**

Term	**Definition**
Functional Testing	Testings that validates the software system against the functional requirements
Non-Functional Testing	Testings that validates the software system against the non-functional requirements
Dynamic Testing	Testing by executing the program
Static Testing	Testings that does not execute the program
Manual Testing	Testings that execute test cases manually
Automated Testing	Testings that leverage automation tools

2 Plan

2.1 Software Description

The software system FSMS is capable of defining FSM by specifying states and transitions, invoking and simulating transitions, and rendering FSM into LaTeX snippets. It is implemented in Python.

2.2 Test Team

The test team includes all members in the development team. The test team members are:

- Anhao Jiao
- Xunzhou Ye
- Kehao Huang

2.3 Automated Testing Approach

Automated testing is planned to be performed at unit testing level.

2.4 Testing Tools

For unit testing of the software system FSMS, pytest will be used to perform automated testing.

2.5 Testing Schedule

See Gantt Chart at the following url https://gitlab.cas.mcmaster.ca/yex33/transitions_l02_grp16/-/blob/main/ProjectSchedule/Project%20Schedule.gan

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 User Input

User Input Testing

1. UIT-1

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: Initial state and relevant states

   Output: A valid FSM

   How test will be performed: The FSM simulator will be running using certain inputs. Then check if there is a valid FSM.

2. UIT-2

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: Modification of transitions between states in an existing FSM

   Output: A modified version of FSM

   How test will be performed: The FSM simulator will be running using certain inputs to form a FSM. Then it will be running modification of transitions and check if there is modified FSM.

3. UIT-3

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: Modification of states in an existing FSM

   Output: A modified version of FSM

   How test will be performed: The FSM simulator will be running using certain inputs to form a FSM. Then it will be running modification of states and check if there is modified FSM.

4. UIT-4

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: The specified transitions

   Output: The FSM will allow user to switch states following the valid transitions

   How test will be performed: The FSM simulator will be running using certain inputs to form a FSM. Then it will be running certain triggers and check if there're changes in states.

5. UIT-5

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: a non-existing trigger event

   Output: a MachineError thrown

   How test will be performed: The FSM simulator will be running using certain inputs to form a FSM. Then it will be running certain triggers and check if there're changes in states.

3.1.2 Output

Output Testing

1. OT-1

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: Require current state

   Output: The current state of the FSM

   How test will be performed: The FSM simulator will be running a valid FSM. Then check if there is current state displayed.

2. OT-2

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: A valid FSM

   Output: LaTeX snippets of the input FSM

   How test will be performed: The FSM simulator will be running using certain inputs to form a FSM. Then check if it can generate corresponding LaTeX snippets.

3. OT-3

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: A valid FSM

   Output: A .png image representing the pre-defined FSM

   How test will be performed: The FSM simulator will be running using certain inputs to form a FSM. Then check if it can generate corresponding image through visual inspection.

4. OT-4

   Type: Functional, Dynamic, Manual

   Initial State: FSM simulator that is used to define a FSM

   Input: A valid FSM with only states.

   Output: A .png image representing the pre-defined FSM with no transition.

   How test will be performed: The FSM simulator will be running using certain inputs to form a FSM. Then check if it can generate corresponding image through visual inspection.

3.2 Tests for Nonfunctional Requirements

3.2.1 Usability Testing

1. UT-1

   Type: Non-Functional, Dynamic, Manual

   Initial State: a well-defined FSM with states, initial state, transitions

   Input/Condition: user asked the program to export the defined FSM into LaTeXsnippets

   Output/Result: the image rendered using the exported snippet is visually appealing, with minimal overlapping of arrows, circles, or other visualization elements.

   How test will be performed: feed a fixed list of states and transition into the program. Developers in the team would be asked to compare the rendered image against their hand-drawn representation of the FSM.

2. UT-2

   Type: Non-Functional, Dynamic, Manual

   Initial State: a computer with only Python preinstalled

   Input: user asked to install the program without assistance

Output: the majority of users are able to install the program independently within 5 minutes

How test will be performed: a group of McMaster engineering students will be asked to pull our repository from gitlab, follow the installation guide, and have the program running for FSM related work.

3. UT-3

   Type: Non-Functional, Dynamic, Manual

   Initial State: a computer with the program installed

   Input: user asked to incorporate the program in their academic work flow

   Output: the majority of users are benefited from the use of our tool

   How test will be performed: a survey will be conducted to ask the user to rate the program.

3.2.2 Maintainability and Support Testing

1. MS-1

   Type: Non-Functional, Dynamic, Manual

   Initial State: a computer with the program installed

   Input: update the software with new features

   Output: The overall maintaining duration shall less than 4 hours

   How test will be performed: manually update the software and time the duration

2. MS-2

   Type: Non-Functional, Dynamic, Manual

   Initial State: a computer with the program installed

   Input: Install and use the software

   Output: 98% of students are able to run the system on their pcs or laptops

   How test will be performed: a survey will be conducted to determine the success rate of using the software

3.3 Traceability Between Test Cases and Requirements

Test Case	Requirement
UIT-1	FR1
UIT-2	FR2
UIT-3	FR3
UIT-4	FR4
UIT-5	FR4
OT-1	FR5
OT-2	FR6
OT-3	FR5
OT-4	FR5
UT-1	NF1–2
UT-2	NF3, NF10, NF13
UT-3	NF2, NF4–7, NF9, NF11–12

4 Tests for Proof of Concept

1. POC-1

   Type: Functional, Dynamic, Manual

   Initial State: user have defined a FSM

   Input: FSM with 5 states and 4 transitions

   Output: a LaTeXsnippet that is ready to be compiled into pdf

   How test will be performed: the snippet will be copied into a .tex file and compiled. The tester will visually inspect the output image to verify its correctness and appearance.

2. POC-2

   Type: Functional, Dynamic, Manual

   Initial State: user have defined a FSM

   Input: FSM with 5 states and 0 transitions

Output: a LaTeXsnippet that is ready to be compiled into pdf

How test will be performed: the snippet will be copied into a .tex file and compiled. The tester will visually inspect the output image to verify its correctness and appearance.

3. POC-3

   Type: Functional, Dynamic, Manual

   Initial State: user have defined a FSM

   Input: FSM with 0 states and 0 transitions

   Output: a LaTeXsnippet that is ready to be compiled into pdf

   How test will be performed: the snippet will be copied into a .tex file and compiled. The tester will visually inspect the output image to verify its correctness and appearance.

4. POC-4

   Type: Functional, Dynamic, Manual

   Initial State: user have defined a FSM

   Input: FSM with 100 states and 400 transitions

   Output: a LaTeXsnippet that is ready to be compiled into pdf

   How test will be performed: the snippet will be copied into a .tex file and compiled. The tester will visually inspect the output image to verify its correctness and appearance.

5 Comparison to Existing Implementation

N/A

6 Unit Testing Plan

6.1 Unit testing of internal functions

We will use pytest to perform unit testing of internal functions. In this project, a method is considered to be a unit. For each method, there is

going to be at least one test case that execute the method with a given input and verify if the output is the same as expected. The unit testing will be conducted on a coverage base, that is, we aim to achieve 80% statement coverage of the entire program.

6.2 Unit testing of output files

The only format of output file of the software system FSMS is LaTeX snippets. In order to validate that the output file generated by the program is correct, tester has to copy and paste the snippet into a .tex file and compile the file. As a result, an image will be generated in the compiled .pdf file. Testings is done by human inspection. In specific, the tester is going to visually validate if the generated image of FSM is correct.

7 Appendix

7.1 Symbolic Parameters

N.A.

7.2 Usability Survey Questions?

N.A.