SE 3XA3: Test Plan
Ratava

Team 9, Makiam Group
Aidan McPheilm - mcpheima
Alexie McDonald - mcdona16
Illya Pilipenko - pilipeni

December 6, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	Version	Notes
2018-09-18	0.0	Added basic file
2018-10-26	1.0	Added sections 5, 6, 7
2018-10-26	1.1	Added sections 1, 2 and revision history
2018-10-26	1.2	Added functional and non-functional test cases
2018-12-03	1.3	Updated team name
2018-12-05	1.3	Modified test cases

1 General Information

1.1 Purpose

The purpose of this document is to have comprehensive documentation for the testing of the software that can be referred to and updated as needed.

1.2 Scope

The project is pretty simple mechanically, with the largest complexity being adding additional generation options that otherwise reuse the hashing code. Therefore, the components the testing scope can cover are: General user interface usability, robustness with varying input, and output consistency when having consistent input.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
TDD	Test Driven Development

Table 3: **Table of Definitions**

Term	**Definition**
Test Driven Development	Unit test cases are written before the corresponding function is written.
Functional Test	A blackbox test which makes sure requirements are met.
Structural Test	A whitebox test which checks for errors and issues with implementation.
Dynamic Test	Testing that involves operating the software while it is running to check for issues.
Static Test	Testing that involves analyzing source code without running the software; opposite of Dynamic Test.
Manual Test	Testing software by means of individual cases, as opposed to automating it.

1.4 Overview of Document

2 Plan

2.1 Software Description

The concept of the software is to take a user input string and generate an image file of an avatar by using the input string as an input to a hashing function, for randomization.

2.2 Test Team

All source code developers and maintainers are in charge of testing, as we are implementing TDD. This makes the test team be comprised of:

- Aidain McPheilm
- Alexie McDonald
- Illya Pilipenko

2.3 Automated Testing Approach

We will be following the TDD approach.

2.4 Testing Tools

We will be doing our testing using pytest, as it's an easy yet effective testing tool for Python.

2.5 Testing Schedule

See Gantt Chart at the following url: Gantt Chart

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Output Testing

Output Test

1. ~~type-test~~test_image_output_type

   Type: ~~Manual~~Automated functional

   Initial State: The program is running and has generated an output image

   Input: User inputs

   Output: A lossy graphics file (.jpg)

   How test will be performed: The image will be generated, the image's extension will be ~~examined, and it's ability to open with such an extension in an available image editor~~ analyzed using the imghdr library and verified to be jpeg

2. ~~hash-out~~test_generate_hash_is_consistent

   Type:~~Manual~~Automated functional

   Initial State: The program is running and the user has entered the appropriate parameters

   Input: hash encoding type and user string

   Output: The hexadecimal output to the hash function

   How test will be performed: The hash function will generate the output, then digest it. To ensure it is truly the output of a hash function, it will be passed through a third party hash analyzer in order to confirm it satisfies the three hash requirements

3. set-qual

   Type: Manual functional

   Initial State: The program is running and the user has been prompted to enter the parameters they wish to fix (if any)

Input: a set of parameters that should be fixed

Output: The hexadecimal output to the hash function

How test will be performed: The program will run and produce the output image. The iteration over the set will satisfy the following predicate: $(\forall i \in set\_of\_qualities \bullet (\exists j \in set\_of\_qualities\_output \bullet i = j))$

4. avatar-select

   Type: Manual functional

   Initial State: The program is running and the user has been prompted to enter the parameters they wish to fix (if any)

   Input: a template to follow

   Output: A lossy graphics file (.jpg)

   How test will be performed: The output image will be generated and opened. The following predicate should be satisfied:
   $(\forall i | 0 \leq i < image\_width \bullet (\forall j | 0 \leq i < image\_height \bullet template[i][j] = image[i][j]))$

5. string-hash

   Type: Manual functional

   Initial State: The program is running and the user has been prompted to enter their preferred string to hash

   Input: A user selected string

   Output: The hexadecimal output to the hash function

   How test will be performed: The hash function will be seeded and the output will be generated. The output given the exact same input string should be equal before digesting

6. hash-type

   Type: Manual functional

   Initial State: The program is running and the output to the hash function has been generated

   Input: User inputs

Output: The hexadecimal output to the hash function

How test will be performed: The hash function will be seeded and the output will be generated. It will be digested, to ensure it is truly the output of that specific encoding, it will be passed through a third party hash analyzer in order to confirm that it is of that encoding.

7. output-path

   Type: Manual functional

   Initial State: The program is running and the user is prompted to enter the image output location

   Input: A string representing the output file name and location

   Output: Output image in the location specified and of that name

   How test will be performed: The output image will be generated. The string representing the location and name will be compared to the location indicated by the file system

8. help-doc

   Type: Manual functional

   Initial State: The program is running

   Input: A user action (button press) to open the help documentation

   Output: Help documentation file opened

   How test will be performed: The help button will be pressed by the user. The help documentation should open and be able to be read in it's entirety.

9. image-show

   Type: Manual functional

   Initial State: The program is running and the output image has been produced

   Input: User inputs

   Output: The generated image in an image editor program

   How test will be performed: The image will be generated and will be opened in an available image editor/viewer within 0.5 seconds of image drawing completion. This metric will be timed.

3.2 Tests for Nonfunctional Requirements

3.2.1 Look and Feel Test

Accessibility Test

1. Accessible fonts test

   Type: Manual functional

   Initial State: The program is running

   Input: GUI font

   Output: Verification of accessibility

   How test will be performed: The qualities of the font will be compared with the web standards for accessibility

2. High contrast button test

   Type: Manual functional

   Initial State: The program is running

   Input: Button colour, font colour

   Output: Contrast metric

   How test will be performed: The colours will be put into WebAIM's contrast checker

3. Screen reader Test

   Type: Manual functional

   Initial State: The program is running

   Input: Text on interface

   Output: Screen reader audio

   How test will be performed: A screen reader will be run and the output will be be evaluated on understandibility

3.2.2 Usability and Humanity Tests

Ease of Use Test

1. Usability Test
   Type: Manual

   Initial State: The program has started

   Input: User input

   Output: A lossy graphics file

   How test will be performed: A sample of users will use the software without consulting the help documentation and their ease of use will be noted (85% of users should use with ease)

3.3 Performance Requirements

1. Speed Test (Random)

   Type: Manual

   Initial State: The program is running

   Input: Input string, hash function

   Output: Output image

   How test will be performed: The program shall run given a set of user inputs and the duration shall be timed. It shall not exceed 5 seconds

2. Speed Test (Fixed)

   Type: Manual

   Initial State: The program is running

   Input: Input string, hash function, set of fixed parameters

   Output: Output image

   How test will be performed: The program shall run given a set of user inputs and the duration shall be timed. It shall not exceed 5 seconds

3.3.1 Expected Technological Environment

1. Environment test Type: Manual

   Initial State: The program is running

   Input: User parameters

   Output: Output image

How test will be performed: The program shall run on a set of different machines with a python3 compiler

3.4 Traceability Between Test Cases and Requirements

In terms of traceability, many test cases and requirements have a one to one relationship.

4 Tests for Proof of Concept

4.1 Generate Hash

test_retrieve_hash_algorithm_returns_correct_hash

1. test-01

   Type: Dynamic

   Initial State:

   Input: 'sha256'

   Output: hashlib.sha256

   How test will be performed: Automatically by running test program containing all tests

2. test-02

   Type: Dynamic

   Initial State:

   Input: 'md5'

   Output: hashlib.md5

   How test will be performed: Automatically by running test program containing all tests

test_generate_hash_is_consistent

1. test-03

   Type: Dynamic

   Initial State: hash_val_1 = GenerateHash.generate_hash("test")
   hash_val_2 = GenerateHash.generate_hash("test")

   Input: hash_val_1 == hash_val_2

   Output: true

   How test will be performed: Automatically by running test program containing all tests

4.2 Use Hash

test_hex_to_rgb

1. test-04

   Type: Dynamic

   Initial State:

   Input: "ffffff"

   Output: (255,255,255)

   How test will be performed: Automatically by running test program containing all tests

2. test-05

   Type: Dynamic

   Initial State:

   Input: "abcabc"

   Output: (171, 202, 188)

   How test will be performed: Automatically by running test program containing all tests

5 Comparison to Existing Implementation

As there has been several new features added (as per the functional requirements), there is a more robust set of test cases. Tests for the GUI and for the new fixed parameters were the largest addition to the set.

6 Unit Testing Plan

6.1 Unit testing of internal functions

For the internal functions we will try, to the best of our ability, to implement TDD.
For example with the function retrieve_hash_algorithm(has_type) we would first write that header, and then write a test with an example input such as 'sha256' and then assert that for that input the output is equal to 'hashlib.sha256'.

6.2 Unit testing of output files

Unit testing of output files is tricky but there is some things that can be done.

We will;

- Check the resolution of the output file to make sure that it is what the user wanted it to be.
- Check the file type to make sure that it was exported in the correct format.
- Check the name of the file to ensure that it has the same name as the input string for the hash.

7 Appendix

7.1 Symbolic Parameters

n/a

7.2 Usability Survey Questions?

- How satisfied were you overall with using this software?
- How could the user interface be improved to make use of the software easier?
- How did you find the feedback from the software as you used it?
- Do you have any other comments?