

# SE 3XA3: Test Plan Gifitti

Team #2,Gifitti  
Nicolai Kozel kozeln  
Riley McGee mcgeer  
Pavle Arezina arezinp

October 31, 2016

# Contents

<b>1</b>	<b>General Information</b>	<b>1</b>
1.1	Purpose . . . . .	1
1.2	Scope . . . . .	1
1.3	Acronyms, Abbreviations, and Symbols . . . . .	1
1.4	Overview of Document . . . . .	2
<b>2</b>	<b>Plan</b>	<b>2</b>
2.1	Software Description . . . . .	2
2.2	Test Team . . . . .	2
2.3	Automated Testing Approach . . . . .	3
2.4	Testing Tools . . . . .	3
2.5	Testing Schedule . . . . .	3
<b>3</b>	<b>System Test Description</b>	<b>3</b>
3.1	Tests for Functional Requirements . . . . .	3
3.1.1	Open GIF . . . . .	3
3.1.2	Save GIF . . . . .	5
3.1.3	Save GIF as Sprite Spreadsheet . . . . .	6
3.1.4	GIF Start, Stop, Modify Length . . . . .	7
3.1.5	Save all frames in a GIF to another known format . . . . .	9
3.1.6	Help Context . . . . .	10
3.1.7	GIF reset . . . . .	11
3.1.8	Frame Addition . . . . .	13
3.1.9	Frame Drawing . . . . .	14
3.2	Tests for Nonfunctional Requirements . . . . .	15
3.2.1	Look and Feel Requirements . . . . .	15
3.2.2	Performance Requirements . . . . .	16
<b>4</b>	<b>Tests for Proof of Concept</b>	<b>18</b>
4.1	Opening a GIF file for playback . . . . .	18
4.2	Saving a GIF file's frames . . . . .	18
<b>5</b>	<b>Comparison to Existing Implementation</b>	<b>19</b>
5.1	Graphics/UI . . . . .	19
5.2	Performance . . . . .	19

<b>6</b>	<b>Unit Testing Plan</b>	<b>19</b>
6.1	Unit testing of internal functions . . . . .	20
6.2	Unit testing of output files . . . . .	20
<b>7</b>	<b>Appendix</b>	<b>21</b>
7.1	Symbolic Parameters . . . . .	21
7.2	Usability Survey Questions . . . . .	21

## List of Tables

<b>1</b>	<b>Revision History</b> . . . . .	<b>ii</b>
<b>2</b>	<b>Table of Abbreviations</b> . . . . .	<b>1</b>
<b>3</b>	<b>Table of Definitions</b> . . . . .	<b>2</b>

## List of Figures

Table 1: **Revision History**

Date	Version	Notes
October 22	1.0	Adding sections 1,2, and 7
October 24	1.1	Adding functional req tests
October 25	1.2	Adding non functional req tests.
October 30	1.3	Adding Unit tests, and automated functional tests

This document describes the test plan for the Gifitti application developed for 3XA3 at McMaster University.

# 1 General Information

## 1.1 Purpose

The purpose of the testing plan is to establish a set of tests that will test the product in its entirety to ensure that it fulfills the intended purpose. This would be accomplished through verifying if Gifitti satisfies the different functional and non-functional requirements that were assigned to it. Having test plans for any product is essential to be able to understand how well the product is satisfying the clients needs and if there is room for improvement.

## 1.2 Scope

This testing plan is utilizing different testing methods, automated and user created, and various techniques, black box and white box testing, to establish if the project has any need for improvement. Two different products will be analyzed through these tests, the Proof of Concept and the first iteration of the final product. Proof of Concept will be tested to ensure that a basic representation of the product was demonstrated while the requirements should be tested when the first iteration of the final product is completed.

## 1.3 Acronyms, Abbreviations, and Symbols

Table 2: <b>Table of Abbreviations</b>	
<b>Abbreviation</b>	<b>Definition</b>
GIF	Graphics Interchange Format - a lossless format for image files that supports both animated and static images.
Abbreviation2	Definition2

Table 3: **Table of Definitions**

<b>Term</b>	<b>Definition</b>
Term1	Definition1
Term2	Definition2

## 1.4 Overview of Document

The testing plan is broken up into distinct parts. Under the heading Plan, the basic information will be given on the product and the tests. System Test Description will contain the specific tests for the functional requirements stated for the product. These tests will broke down into what type of tests they are and the results they achieve depending on their specific input. Testing for non-functional requirements will follow the same format where input will be given and the output will be measured for all of the provided non-functional requirements. Under tests for proof of concept, the same format will be utilized as the functional testing but it will not be testing the requirements for the project but for the goals of the proof of concept. Furthermore there will be tests to compare Gifitti to the original product it was based on and unit testing plans to ensure correct output is achieved through proper internal functions.

# 2 Plan

## 2.1 Software Description

Gifitti is a software product that allows the common user to be able to manipulate for their entertainment or commercial needs. With an intuitive design, it allows a person who has never done any kind of graphical editing to be able to manipulate a GIF to achieve the person's particular design. This product can also serve a commercial purpose in allowing the creation of sprite spreadsheets and a quick production of GIFs.

## 2.2 Test Team

The team to implement the test plan for the project will be Pavle Arezina, Riley McGee, Nicolai Kozel.

## **2.3 Automated Testing Approach**

This test plan will utilize automated testing for verification that the GIF manipulation functionality changes the GIF properly. Such that the exported GIF image will match what is created by the user of our product.

## **2.4 Testing Tools**

The only tool to be utilized to test this product will be the Microsoft Unit Test Framework thatt is native with Visual Studio.

## **2.5 Testing Schedule**

See Gantt Chart [here](#).

# **3 System Test Description**

Testing allows the developers to detect errors that the test cases cover. With the functional and non-functional requirements the developer knows the expected results and can evaluate the results of tests accordingly. Pass or fail values are assigned to executed tests discreetly unless the test is to have an error tolerance for passing.

## **3.1 Tests for Functional Requirements**

Functional requirements prescribe what services the software should provide. They capture the intended software effects on the environment and applicability conditions. These tests ensure that the functional requirements of Gifitti are fulfilled or the tests will help to discover any functional requirements not satisfied by Gifitti.

### **3.1.1 Open GIF**

**The User is able to open a GIF from a specified location**

1. Select proper formatted gif- id1

Type: Manual Functional. Initial State: Program loaded; no GIF Loaded. Input: File name. Output: System loads gif into memory, displays it to the user in the gif view.

How test will be performed:

- (a) Launch the program
- (b) Select Open
- (c) From the Open dialog specify a path to a known gif image
- (d) After the image is loaded verify that it is being displayed, and resides in system memory

## 2. No File Selected in File Dialog-id2

Type: Manual Functional. Initial State: Program loaded; no GIF Loaded. Input: No file path. Output: No image loaded.

How test will be performed:

- (a) Launch the program
- (b) Select Open
- (c) Select open option with no file path specified
- (d) Verify program remains open, and no image is loaded

## 3. Close File Dialog-id3

Type: Manual Functional. Initial State: Program loaded; no GIF Loaded. Input: None. Output: No image loaded.

How test will be performed:

- (a) Launch the program
- (b) Select Open
- (c) Close the file dialog
- (d) Verify program remains open, and no image is loaded

#### 4. Open random non gif file-id4

Type: Manual Functional. Initial State: Program loaded; no GIF Loaded. Input: File that is not a GIF. Output: No image loaded.

How test will be performed:

- (a) Launch the program
- (b) Select Open
- (c) Try and select a file that is not a GIF or specify a file path to a known file
- (d) Verify program remains open, and no image is loaded

### 3.1.2 Save GIF

**The user is able to save a GIF to a specified location**

#### 1. Save GIF to known location-id1

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: File path, GIF file.

Output: GIF file saved to specified location.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Select save as
- (d) Specify a known system file path and a saved image name
- (e) Verify the loaded and saved GIFs are identical

#### 2. Save GIF to no existant location- id2



Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: File path, GIF file.

Output: GIF file not saved to specified location.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Select save as
- (d) Specify a system file path known not to exist and a saved image name
- (e) Verify the user is informed that the file path does not work

### 3. Save GIF to Opened Location- id3

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: File path, GIF file.

Output: GIF file saved to specified location.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Modify the GIF
- (d) Save image
- (e) Verify new GIF is saved over the originally opened GIF

#### 3.1.3 Save GIF as Sprite Spreadsheet

**The user is able to save a GIF as a sprite spreadsheet in a specified location**

1. Save Sprite Spreadsheet to known location-id1

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: File path, GIF file.

Output: GIF file saved as a Sprite Spreadsheet to specified location.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Choose to export the image as a sprite spreadsheet
- (d) Verify that the GIF is a single image representation of the GIF via frames

#### **3.1.4 GIF Start, Stop, Modify Length**

**The user is able to start (play), start and stop GIFs, testing of GIF shortening and playback speed occurs here as well**

1. Play Stop GIF-id1

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: GIF

Output: None.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Select Play Option
- (d) Verify the Gif is iterating over frames as expected
- (e) Select Play option again, verify no change occurs

- (f) Select Stop
- (g) Verify GIF stops playing
- (h) Select Stop again, verify no change occurs

## 2. GIF shorten-id2

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: GIF

Output: Shortend GIF.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Shorten gif frames in use
- (d) Play the GIF to verify it is correct
- (e) Export the GIF
- (f) Verify that the exported GIF represents the one playing, and is shorter than the original

## 3. Automated GIF shorten-id3

Type: Automated Functional.

Initial State: Image reading module driver

Input: GIF; Expected output GIF;

Output: Expected output GIF

How test will be performed:

- (a) Test will load GIF from a know loacation
- (b) Test will subset the GIF to a known frame set
- (c) The GIF will then be compared to an expected GIF in a frame-by-frame manner

- (d) Each frame is to be cast to a bitmap then compared in a pixel-by-pixel manner
- (e) Pass if 100 percent of all pixels match

#### 4. GIF speed-id4

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: GIF

Output: GIF inputted with different frame rate.

How test will be performed:

- (a) Launch the program
- (b) Open a GIF
- (c) Increase frame hold time
- (d) Play the GIF to verify it is slower than original
- (e) Export the GIF
- (f) Verify that the exported GIF represents the one playing, and is slower, with respect to frame rate, than the original

#### **3.1.5 Save all frames in a GIF to another known format**

**The user is able to save a GIF frame by frame to PNG, JPEG, BMP and TIFF formats**

##### 1. Save PNG frames to known location-id1

Type: Manual Functional.

Initial State: Program loaded; GIF Loaded.

Input: Folder path, GIF file.

Output: PNG Frames in specified folder named corresponding to frame.

How test will be performed:

- (a) Launch the program
  - (b) Open a GIF
  - (c) Choose to export the image as a PNG set
  - (d) Select the folder where the PNGs will be dumped
  - (e) Verify the folder fills with frames as PNG
  - (f) Ensure a 3rd party program can load the PNGs. Software such as Photoshop or MS paint is satisfactory
2. Save JPEG frames to known location-id2  
This test is identical to id1 with PNG replaced with JPEG
  3. Save BMP frames to known location-id3  
This test is identical to id1 with PNG replaced with BMP
  4. Save TIFF frames to known location-id4  
This test is identical to id1 with PNG replaced with TIFF

### **3.1.6 Help Context**

#### **Verifies and validates HELP context**

1. Help Context-id1

Type: Manual Functional.  
 Initial State: Program loaded  
 Input: None  
 Output: None

How test will be performed:

- (a) Launch the program
- (b) Select Help Context
- (c) Recurse through all Help Context options, verify spelling for professionalism
- (d) Close help context, verify program remains in same state as launch put it in

### 3.1.7 GIF reset

**Verifies and validates GIF can be reset to loaded state**

#### 1. Modify Attributes-id1

Type: Manual Functional.

Initial State: Program loaded; GIF loaded

Input: GIF

Output: Inputted GIF

How test will be performed:

- (a) Launch the program
- (b) Load a GIF
- (c) Modify GIF attributes such as frame length, coloration etc
- (d) Select reset
- (e) Verify GIF in is the same as current GIF shown

#### 2. Modify via adding a frame-id2

Type: Manual Functional.

Initial State: Program loaded; GIF loaded

Input: GIF, inputted frame

Output: Inputted GIF

How test will be performed:

- (a) Launch the program
- (b) Load a GIF
- (c) Modify GIF by adding a frame
- (d) Select reset
- (e) Verify GIF loaded has added frame removed

3. Modify via frame subset selected-id3

Type: Manual Functional.

Initial State: Program loaded; GIF loaded

Input: GIF

Output: Inputted GIF

How test will be performed:

- (a) Launch the program
- (b) Load a GIF
- (c) Modify GIF by subsetting what frames are used
- (d) Select reset
- (e) Verify GIF loaded returns to full GIF state

4. Automated modify via adding a frame-id4

Type: Automated Functional.

Initial State: Image reading module driver

Input: GIF; BMP image;

Output: Inputted GIF

How test will be performed:

- (a) Test will load GIF from a know loaction
- (b) Test will load the frame being added from a known loaction
- (c) The frame will be added to a specific spot of the GIF
- (d) The GIF will then be reset
- (e) The stored GIF is to be frame by frame compared to the original GIF
- (f) Each frame is to be cast to a bitmap then compared in a pixel-by-pixel manner
- (g) Pass if 100 percent of all pixels match

### 3.1.8 Frame Addition

#### Verifies frame addition

##### 1. Add GIFs-id1

Type: Manual Functional.

Initial State: Program loaded

Input: GIF x 2

Output: Concatinated GIFs

How test will be performed:

- (a) Launch the program
- (b) Load a Gif
- (c) Select a frame and import a gif
- (d) Gif should subset from frame to end of the imported gif
- (e) Verify the concatination holds on export

##### 2. Add PNGs-id2

Type: Manual Functional.

Initial State: Program loaded

Input: GIF; PNG image

Output: GIF with PNG input added as a frame

How test will be performed:

- (a) Launch the program
- (b) Load a Gif
- (c) Select a frame and import a PNG
- (d) Verify PNG is added after current frame in GIF
- (e) Verify the the new GIF is exported the same way



3. JPEG frames to GIF-id3  
This test is identical to id2 with PNG replaced with JPEG
4. TIFF frames to GIF-id4  
This test is identical to id2 with PNG replaced with TIFF
5. BMP frames to GIF-id5  
This test is identical to id2 with PNG replaced with BMP
6. AutomatedTesting of BMP frams to GIF-id6 Type: Automated Functional.  
Initial State: Image reading module driver  
Input: GIF; BMP image; Expected output GIF  
Output: GIF with BMP input added as a frame

How test will be performed:

- (a) Test will load GIF from a know loaction
- (b) Test will load the frame being added from a known loaction
- (c) The frame will be added to a specific spot of the GIF
- (d) The new GIF is to be compared to expected through breaking each down into frames
- (e) Cast each frame and pixel-by-pixel compare the output to the expected
- (f) Pass if 100 percent of all pixels match

### **3.1.9 Frame Drawing**

**Verifies and validates ability to draw on frames**

1. Single Frame Drawing-id1

Type: Manual Functional.

Initial State: Program loaded; GIF loaded

Input: GIF

Output: Inputted Gif with image overlay on a frame

How test will be performed:

- (a) Launch the program
- (b) Load a GIF
- (c) Make an obvious edit to a frame via drawing
- (d) Export the GIF and verify obvious change remains

## 2. Multi Frame Drawing-id1

Type: Manual Functional.

Initial State: Program loaded; GIF loaded

Input: GIF

Output: Inputted Gif with image overlay many frames

How test will be performed:

- (a) Launch the program
- (b) Load a GIF
- (c) Make an obvious edit to a frame via drawing
- (d) Make the edit extend over a set of frames
- (e) Export the GIF and verify obvious change remains on all selected frames

## 3.2 Tests for Nonfunctional Requirements

Non-functional requirements constrain how such services should be provided. They are more involved in the quality of the product such as the safety of utilizing the software or how accurate it is. These tests ensure that the non-functional requirements of Gifitti are fulfilled or the tests will allow the discovery of any requirements not satisfied by Gifitti.

### 3.2.1 Look and Feel Requirements

#### Audio Test

1. audio-01

Type: Manual Structural

Initial State: Program loaded;

Input: A file type not supported by Gifitti.

Output: Error sound played through speakers.

How test will be performed: When loading a file into the program, select a file type other than a .gif.

2. audio-02

Type: Manual Structural

Initial State: Program loaded; GIF loaded;

Input: GIF

Output: Affirmative ding sound played through speakers.

How test will be performed: Output a subset of frames from the GIF.

### 3.2.2 Performance Requirements

#### Speed

1. speed-01

Type: Manual Structural

Initial State: Program loaded; GIF loaded.

Input: GIF with number of frames on the magnitude of 100's of frames.

Output: Subsection of frames chosen to be exported.

How test will be performed: Create a timer inside the program to output the time it takes to output the files to a debug.txt file. The time should be less than MAX\_EXPORT\_TIME.

2. speed-02

Type: Manual Structural

Initial State: Program loaded;

Input: GIF.

Output: NA

How test will be performed: Load a GIF into the program. The time to load the GIF and start playback should be less than MAX\_UI\_LOAD.

## **Precision**

### **1. precision-01**

Type: Manual Structural

Initial State: Program loaded; GIF loaded.

Input: GIF

Output: Subsection of frames chosen to be exported.

How test will be performed: Export the first 5 frames of the GIF. Navigate to the save location and verify that the number of frames saved is equal to 5.

## **Safety Critical**

### **1. safety-01**

Type: Manual Structural

Initial State: Program loaded; GIF loaded.

Input: GIF

Output: NA

How test will be performed: Export a subsection of frames the GIF to a folder. Try to export to the same folder again. The program should ask if you are okay with overwriting the existing files.

### **2. safety-02**

Type: Manual Structural

Initial State: Program loaded; GIF loaded.

Input: GIF

Output: NA

How test will be performed: Create a small partition of the HDD and fill it so there is no available space. Try to export frames to this location. The program should not continue the operation and inform the user that disk space needs to be cleared before continuing.

## **4 Tests for Proof of Concept**

The proof of concept is a demonstration of the feasibility of the goal of the software project. Ensuring that the Proof on Concept is producing the intended results can only be achieved through tests specifically designed to root out any of the errors present in the build of the proof of concept. By successfully testing the proof of concept and achieving the intended goals it is shown that the software product is a possible product.

### **4.1 Opening a GIF file for playback**

#### **Open GIF**

1. OpenGif-01

Type: Manual Functional

Initial State: Program must be in normal state (form window is open and playback window is blank).

Input: File

Output: GIF is shown in playback window of the form.

How test will be performed: Click open button, select a file of type .gif, and verify that the GIF loads and begins to playback within the form window.

### **4.2 Saving a GIF file's frames**

#### **Save GIF**

1. SaveGif-01

Type: Manual Functional

Initial State: Program must be in playback state (form window is open and playback window is playing GIF).

Input: GIF File

Output: GIF's frames are saved as .bmp in folder specified.

How test will be performed: Click save frames button, select a folder, and verify that the GIF's frames are saved to the specified folder as .bmp.

## **5 Comparison to Existing Implementation**

### **5.1 Graphics/UI**

1. GIF playback resolution is the same or better than Gif Viewer.
2. Program has the same button scheme as Gif Viewer (Open button to open file, Extract Frames button to save frames).
3. Program has a help menu available that is similiar to Gif Viewer, but is available at all times.
4. Program's color scheme and design resembles Gif Viewer.

### **5.2 Performance**

1. GIF playback is at the same smoothness/framerate or better than Gif Viewer.
2. Opening and saving a file takes the same amount of time or less than Gif Viewer.

## **6 Unit Testing Plan**

The unit testing plan, is the devised set of tests code and system modules must undergo regardless of the system integration level. Unit testing allows

the project to have white box testing on the individual system components, each unit test can be executed via testing frameworks in C Sharp.

The testing framework being utilized is the native Visual Studio unit testing framework: `Microsoft.VisualStudio.TestTools.UnitTesting`.

## **6.1 Unit testing of internal functions**

Unit testing will be done on system methods that return a value, or modify known objects in an expected way. A metric for passing is to be used on all methods with an error tolerance, such as manipulation of a GIF (where slight variations can exist in files that present the same). A method with no expected error tolerance should be implemented as discreetly pass or fail. All unit test inputs must fall in a known domain where the expected output can be assumed. No stubs will be required for unit testing of functions. Image reading is needed as a driver for any method using GIFs or other image types. Any written unit test must utilize the C Sharp attributes for Unit Testing provided by the framework. This ensures the system does not build the testing package into the consumer version of the software. We plan to have unit testing cover approximately 50 percent of our major methods due to project time constraints.

## **6.2 Unit testing of output files**

To test the validity of outputted GIF images the testing system will hold image resources of input frames, input GIFs, as well as the expected GIF outputted. A known GIF should be manipulated then compared to the expected GIF in a frame-by-frame matter. To exclude meta-data issues associated with GIF images, each frame should be cast to a known image format such as a bitmap, then compared pixel-by-pixel. Frames should have a 100 percent pixel match for the expected GIF to pass. A method for casting Frames to bitmaps then comparing the pixels should be added to the testing package, prior to testing GIF manipulation. Simply verify this function by sending the same image as parameters.

## **References**

## 7 Appendix

This section contains symbolic parameters for this document and the usability survey that will be delivered to a focus group upon initial completion of the application.

### 7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC\_CONSTANTS. Their values are defined in this section for easy maintenance.

### 7.2 Usability Survey Questions

The survey will be delivered in the same format as the Questionnaire for User Interface Satisfaction. This questionnaire is composed of various questions pertaining to several sub categories on a 0-9 scale. This includes the screen, terminology and system information, learning, and system capabilities. It also allows the user to list the most positive and negative aspects of the program. The questionnaire can be found at [garyperلمان.com](http://garyperلمان.com)