

**The Commuters**

**SugarLabs**

**Automated Test Framework**

**Document Details**

**Prepared By:** Andrew Armstrong

Ian Feeley

Tan Nguyen

Justin Wooton

**Project:** Automated Test Framework for Sugar Operating System

**Version: 5**

**Date:** Nov. 21st , 2013

**Scope:** SugarLabs Operating System, ‘Physics’ Activity

**Classification:** Open Source, Non-confidential

**Contact:** Tan Nguyen, Group Leader

**Email:** dtnguyen@g.cofc.edu

**Document History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Editor(s)** | **Details** |
| 1 | Sept. 24, 2013 | A Armstrong,  I Feeley,  T Nguyen | Summary of project, report on Subversion experience |
| 2 | Oct. 08, 2013 | A Armstrong | Detailed test plan, partial test suite |
| 3 | Oct. 29, 2013 | A Armstrong | Framework design, partial test suite |
| 4 | Nov. 14, 2013 | A Armstrong,  T Nguyen,  J Wooton | Complete framework design, completed test suite |
| 5 | Nov. 21, 2013 | A Armstrong,  J Wooton | Polish document, create poster and powerpoint |

**Table of Contents**

1. **Context ……………………………………………………………………………………………………………………………………………….. 4**
2. **Project Summary ………………………………………………………………………………………………………………………………… 5**
   1. Project Summary – Sugar …………………………………………………………………………………………………………………. 5
   2. Project Installation and Build …………………………………………………………………………………………………………… 5
3. **Test Plan ………..………………..…………………………………………………………………………………………………………………. 6**
   1. Weekly Test Schedule ……………………………………………………………………………………………………………………… 6
   2. Test Strategy …………………………………………………………………………………………………………………………………… 6
   3. Requirements Traceability ………………………………………………………………………………………………………………. 6
   4. Test Artifacts …………………………………………………………………………………………………………………………………… 6
   5. Constraints ……………………………………………………………………………………………………………………………………
   6. Weekly Test Schedule …………………………………………………………………………………………………………………….
   7. Testing Artifacts ……………………
   8. Test Cases ………………………………………………..
4. **Testing Framework ……………………………………………………………………………………………..**
   1. Framework Design ……………………………………………………………………………………..
   2. Code Documentation ………………………………………………………………………………………………………………………
   3. Framework Instructions ……………………………………………………………………………………………………….............
5. **Fault Injection ………………**
6. **Self Evaluations ……………..**
7. **Suggestions ……………………**
8. **Code Appendix ……………………**

# 0. Context

**This section provides a basic understanding of why the document has been written, and what readers can expect to see in the rest of the document.**

The *Project Deliverable* document defines all of the group procedures for the development and task/time management of the Automated Test Framework for Sugars ‘Physics’ activity. These procedures range from group conference scheduling, task and role distribution, plan of action to carry out the given tasks, and reporting of status of completion of tasks. This document helps to ensure that group members stay on task, gives all other group members a sense of how overall development is progressing, and allows them to make documented updates on their own progress.

# 1. Project Summary

**This section details the chosen project to be tested along with system requirements and our experience installing and building it.**

## 1.1. Project Summary – Sugar

Sugar is an open-source operating system designed for use in inexpensive laptops that are mass produced by the non-profit organization, One Laptop per Child, to be sold to developing countries as educational devices. It has many small programs, called ‘Activities’, which range from lightweight web browsers, to small chat programs between computers, to games that encourage critical thinking. Our group has chosen to focus on the ‘Physics’ Activity, in which the some of the laws of physics are simulated and users can create shapes, connect them together in different ways, and observe how forces like gravity, friction, momentum, etc. act on their creations.

## 1.2. Project Installation and Build

Sugar is written in Python and runs on the GNU/Linux Kernel, so can be natively installed on several distributions (listed here: <http://wiki.sugarlabs.org/go/Supported_distributions>). It can also be run on a virtual machine from other operating systems simulating any of the listed distributions.

Once the ideal operating system is installed, you must have git installed to check out the code from the repository in order to build and run it. The SugarLabs community has pre-developed scripts available that make building the project very easy, so all that is needed to be running Sugar are these steps in the command line:

git clone git:*//github.com/sugarlabs/sugar-build.git*

cd sugar-build

./osbuild pull

run

Note that there are no pre-existing test cases for the ‘Physics’ Activity, so our group will be the first to develop them.

# 2. Test Plan

**This section details our weekly test plan schedule, our strategy for developing test cases, the artifacts to be tested, as well as constraints.**

## 2.1. Hardware and Software Requirements

Running Sugar OS is not very demanding. As stated previously, it is based on the linux kernel, which suggests meeting the following hardware requirements:

-          A 1GHz or faster processor

-          1GB of system memory

The testing framework is written in Python (v2.7), and therefore, the system for running the framework also needs to meet the minimum requirements for Python. Windows 9x, ME, NT4 and earlier are not supported, and MacOS9 and earlier are not supported. Python comes pre-installed on most linux/unix packages.

## 2.2. Test Strategy

We will begin by testing commonly used methods within the PhysicsGame Class. For example, the method *distance (point1, point2)* is used in many other methods of the PhysicsGame Class. From there, we will branch out and test the more complex methods. For each test we will have predicted results based on our understanding of the code and compare the theoretical results with the actual results. These pass/fail results will be recorded and exported to a web browser for easy visualization.

## 2.3. Test Recording Procedures

The testing framework will automatically record the results of the test and call a Python datetime method to record the time it was run on the local machine. This information is then passed to the web browser.

## 2.4. Requirements Traceability

Each test case will correspond to a single requirement. We will test index out of bounds errors, type mismatch errors, input overflow errors, and parameter acceptance errors.

## 2.5. Constraints

The testing framework should automatically find all of the test cases and execute them. If a user wants to add another test case, they should simply be able to create another .txt file in the appropriate directory with the desired parameters of the test case, and the framework should be able to discover it, execute it, and pass the results to the output screen.

## 2.6. Weekly Test Schedule

Here we list the overall schedule for our test plan:

Week 1 - Oct 06 - Write requirements, Make test cases for helper methods

Week 2 - Oct 13 - Start auto test framework.

Week 3 - Oct 20 - Implement tests. Refine tests based on results.

Week 4 - Oct 27 - Deliverable 3: The Automated Test Framework

Week 5 - Nov 03 - Begin functional implementation of ATF

Week 6 - Nov 10 - Deliverable 4: Completed Automated Test Framework

Week 7 - Nov 17 - Book

Week 8 - Nov 21 - Deliverable 5 and Presentation

## 2.7. Testing Artifacts

The following methods are the methods we selected as our artifacts to write test cases for:

Method 1: distance(pt1, pt2)

This method takes in two ordered pairs (or, 2-element arrays) and returns the distance between them.

Method 2: getAngle(pt1, pt2)

This method returns the angle that is formed between the intersecting line created by pt1 and pt2 and the x-axis. Returns values between -pi and pi.

Method 3: constructTriangleFromLine(p1, p2)

This method returns a list of ordered pairs that would make an equilateral triangle with pt1 and pt2.

Method 4: polyArea(vertices)

This method returns the area of the polygon with the given list of vertices.

Method 5: insideTriangle(pt, triangle)

This method returns true if the specified point is in the specified triangle.

## 2.8. Test Cases

Below are the 25 test cases we have composed:

Test Name: Test1

Test id: 1

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input: [0,0],[0,0]

expectedOutcome: 0.0

Test Name: Test2

Test id: 2

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input: [-1,0],[0,0]

expectedOutcome: 1.0

Test Name: Test3

Test id: 3

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input: [-9999999999999999,0],[0,0]

expectedOutcome: -9999999999999999.0

Test Name: Test4

Test id: 4

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input: [-0,0],[0,0]

expectedOutcome: 0.0

Test Name: Test5

Test id: 5

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input: [9999999999999999999999999999,0],[0,0]

expectedOutcome: 9999999999999999999999999999.0

Test Name: Test6

Test id: 6

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input: [q,0],[0,0]

expectedOutcome: NameError

Test Name: Test7

Test id: 7

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input: [1.8,0],[0,0]

expectedOutcome: 1.8

Test Name: Test8

Test id: 8

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input: [1.8.8,0],[0,0]

expectedOutcome: SyntaxError

Test Name: Test9

Test id: 9

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input: pie

expectedOutcome: NameError

Test Name: Test10

Test id: 10

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input: [0+32,0],[0,0]

expectedOutcome: 32.0

Test Name: Test11

Test id: 11

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input: [1.8,0][0,0][7,1][4,3]

expectedOutcome: TypeError

Test Name: Test12

Test id: 12

requirement: ?

component: helpers.py

method: distance(pt1, pt2)

input:

expectedOutcome: TypeError

Test Name: Test13

Test id: 13

requirement: ?

component: helpers.py

method: getAngle(pt1, pt2)

input: [0,0],[0,0]

expectedOutcome: 0.0

Test Name: Test14

Test id: 14

requirement: ?

component: helpers.py

method: getAngle(pt1, pt2)

input: pie

expectedOutcome: NameError

Test Name: Test15

Test id: 15

requirement: ?

component: helpers.py

method: getAngle(pt1, pt2)

input: [q,0],[0,0]

expectedOutcome: NameError

Test Name: Test16

Test id: 16

requirement: ?

component: helpers.py

method: getAngle(pt1, pt2)

input: [33,0],[0,0]

expectedOutcome: 3.14159265359

Test Name: Test17

Test id: 17

requirement: ?

component: helpers.py

method: getAngle(pt1, pt2)

input:

expectedOutcome: TypeError

Test Name: Test18

Test id: 18

requirement: ?

component: helpers.py

method: getAngle(pt1, pt2)

input: [1.8.8,0],[0,0]

expectedOutcome: SyntaxError

Test Name: Test19

Test id: 19

requirement: ?

component: helpers.py

method: getAngle(pt1, pt2)

input: [1.8,0][0,0][7,1][4,3]

expectedOutcome: TypeError

Test Name: Test20

Test id: 20

requirement: ?

component: helpers.py

method: getAngle(pt1, pt2)

input: [9999999999,9999999999999],[-999999999,-99999999999]

expectedOutcome: 1.57188543527

Test Name: Test21

Test id: 21

requirement: ?

component: helpers.py

method: getAngle(pt1, pt2)

input: [0,0],[999999999,99999999999]

expectedOutcome: -1.56079666012

Test Name: Test22

Test id: 22

requirement: ?

component: helpers.py

method: constructTriangleFromLine(p1, p2)

input: [10,0],[0,0]

expectedOutcome: [[0, 0], [10.0, 5.7735], [10.0, -5.7735]]

Test Name: Test23

Test id: 23

requirement: ?

component: helpers.py

method: polyArea(vertices)

input: [[10,0],[0,0],[0,10],[10,10]]

expectedOutcome: 100

Test Name: Test24

Test id: 24

requirement: ?

component: helpers.py

method: insideTriangle(pt, triangle)

input: [100,100],[[10,0],[0,0],[10,10]]

expectedOutcome: False

Test Name: Test25

Test id: 25

requirement: ?

component: helpers.py

method: polyArea(vertices)

input: [[10,10],[-10,-10],[10,-10],[-10,10]]

expectedOutcome: 400

# 3. Testing Framework

**This section details the design of our automated test framework, provides code documentation, step by step instructions on how to run the framework and the template for creating new test cases.**

## 3.1. Framework Design

The automated test framework is composed of several files and file types. There is the Python file for finding the test cases in the host system directory, executing them, and creating a new JSON file that contains all of the test output. Also included are all of the javascript files that read the newly created JSON file and format the webpage to show the results using css, jQuery, and all of its dependencies.

## 3.2. Code Documentation

## 3.3. Framework Instructions

Here are step by step instructions to be able to run our test framework:

## 3.4. Test Case Template

Here is the template you should follow if you wish to create more test cases to use with our framework:

Test Name: <name of test>

Test ID: <test number>

# 4. Fault Injection

**Injecting faults into the source code is a good way to ensure that the written test cases are working as intended. If the test results differ from before and after the injection, the test was successful, whether the outcome was a pass or fail.**

## 4.1. Proposed Faults

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Line Number** | **Original Code** | **Modified Code** | **Affected Test Case(s)** | **Expected Outcome** |
| 29 | return math.sqrt((pt1[0] - pt2[0]) \*\* 2 + (pt1[1] - pt2[1]) \*\* 2) | return math.sqrt((pt1[0] - pt2[0]) \*\* 2 + (pt1[1] - pt2[1]) \* 2) | Test 26 | Fail |
| 34 | xcomp = pt2[0] - pt1[0] | xcomp = pt2[0] + pt1[0] | Test 16, 20, 27 | Fail |
| 41 | halfHeightVector = (0.57735 \* (p2[1] - p1[1]), 0.57735 \* (p2[0] - p1[0])) | halfHeightVector = (0.57735 + (p2[1] - p1[1]), 0.57735 \* (p2[0] - p1[0])) | Test 22 | Fail |
| 54 | A += vertices[p][0] \* vertices[q][1] + vertices[q][0] \* vertices[p][1] | A += vertices[p][0] \* vertices[q][1] - vertices[q][0] \* vertices[p][1] | Test 23 | Fail |
| 68 | by = triangle[0][1] - triangle[2][1] | by = triangle[0][1] + triangle[2][1] | Test 28 | Fail |

# 5. Self Evaluations

# 6. Suggestions

# 7. Code Appendix