The Commuters Present: DELIVERABLE # 2

The Chosen Project: SugarLabs

sugarlabs

http://www.sugarlabs.org/

The Commuter's Test Plan:

Overview Schedule:

Week 00 - Oct 06 - Write requirements, Make test cases for helper methods

Week 01 - Oct 13 - Start auto test framework.

Week 02 - Oct 20 - Implement tests. Refine tests based on results.

Week 03 - Oct 27 - Deliverable 3: The Automated Test Framework

Week 04 - Nov 03 - Begin functional implementation of ATF

Week 05 - Nov 10 - Deliverable 4: Completed Automated Test Framework

Week 06 - Nov 17 - Book

Week 08 - Nov 24 - Deliverable 5

The testing process

We will begin by testing each of the helper methods for the Class PhysicsGame. From there, progress allowing we will aim to test interactions between the Sugar OS and the PhysicsGame.

For each test we will have predicted results based on our understanding of the code and we will compare these results with the results of the test run. These pass/fail results will be recorded and further testing will be developed from their results.

Requirements traceability

Each individual test will correspond to a requirement. Be it making sure a method only accepts certain types or making sure the algorithms in the methods function as intended. These will be explicitly specified at a later date once testing has begun.

Tested items
We are beginning our testing with aspects of the Physics activity in SugarLabs. We are starting with the helper methods since they are frequently used and then expanding from there.

Testing schedule
See overview schedule.
Once test cases are concrete, explicit scheduling will be made. As of now, consider the test cases fluid, and thereby a schedule not explicit.

Test recording procedures
We will implement a script that logs all tests. This includes the data that is passed in, the method being tested, and the output, be it a pass or a fail and any errors associated with the fail. These results will be passed into an HTML document for ease of access and storage purposes. After a bought of testing, we will analyze the results and determine whether or not the tests need to be revised or otherwise augmented.

Hardware and software requirements
SugarLabs is a required Operating System. Python is a required programming language. The Activities are written in Python, so we will be using Python to write the tests. We may implement some Jquery to make the test results more easily legible.

Constraints
Our group suffers from sporadic work schedules, so group meetings will take meticulous planning and compromises. This is the most visible obstacle in our way.
Some members of our group do not have much scripting or testing experience. This learning curve may have effects on the scheduling.

List of available test subjects with tentative cases:
Helper Methods
def distance(pt1, pt2):
-Expects 2 ordered pairs (n,n),(n,n)
-Negative Negative, Negative Positive, Positive Positive, Zero

def getAngle(pt1, pt2):
-Expects 2 ordered pairs (n,n),(n,n)
-Negative Negative, Negative Positive, Positive Positive, Zero

def constructTriangleFromLine(p1, p2):
Returns list of ordered pairs describing equilateral triangle around segment pt1 -> pt2
-Negative Negative, Negative Positive, Positive Positive, Zero

Cont. list of available test subjects:
def polyArea(vertices):
def insideTriangle(pt, triangle):
def polySnip(vertices, u, v, w, n):
def decomposePoly(vertices):
def tuple_to_int(tuple_input):

The Big One:
```
class PhysicsGame:
        def __init__(self, screen):
        def switch_off_fake_pygame_cursor_cb(self, panel, event):
        def switch_on_fake_pygame_cursor_cb(self, panel, event):
        def run(self):
        def setTool(self, tool):
        def main():
```

Individual tools:
```
class Tool(object):
        def __init__(self, gameInstance):
        def handleEvents(self, event):
        def handleToolEvent(self, event):
        def draw(self):
        def cancel(self):
class CircleTool(Tool):
        def __init__(self, gameInstance):
```

```
        def handleToolEvent(self, event):
        def draw(self):
        def cancel(self):
class BoxTool(Tool):
        def __init__(self, gameInstance):
        def handleToolEvent(self, event):
        def draw(self):
        def cancel(self):
class TriangleTool(Tool):
        def __init__(self, gameInstance):
        def handleToolEvent(self, event):
        def draw(self):
        def cancel(self):
class PolygonTool(Tool):
        def __init__(self, gameInstance):
        def handleToolEvent(self, event):
        def draw(self):
        def cancel(self):
class MagicPenTool(Tool):
        def __init__(self, gameInstance):
        def handleToolEvent(self, event):
        def draw(self):
        def cancel(self):
class GrabTool(Tool):
        def __init__(self, gameInstance):
        def handleToolEvent(self, event):
        def cancel(self):
class JointTool(Tool):
        def __init__(self, gameInstance):
        def handleToolEvent(self, event):
        def draw(self):
        def cancel(self):
class PinTool(Tool):
        def __init__(self, gameInstance):
        def handleToolEvent(self, event):
        def cancel(self):
class MotorTool(Tool):
        def __init__(self, gameInstance):
        def handleToolEvent(self, event):
```

```
    def cancel(self):
class RollTool(Tool):
    def __init__(self, gameInstance):
    def handleToolEvent(self, event):
    def cancel(self):
class DestroyTool(Tool):
    def __init__(self, gameInstance):
    def handleToolEvent(self, event):
    def draw(self):
    def cancel(self):
    def getAllTools():
```

***Note: all methods, tests and schedules are, as of Oct 08, 2013, tentative and subject to change.**