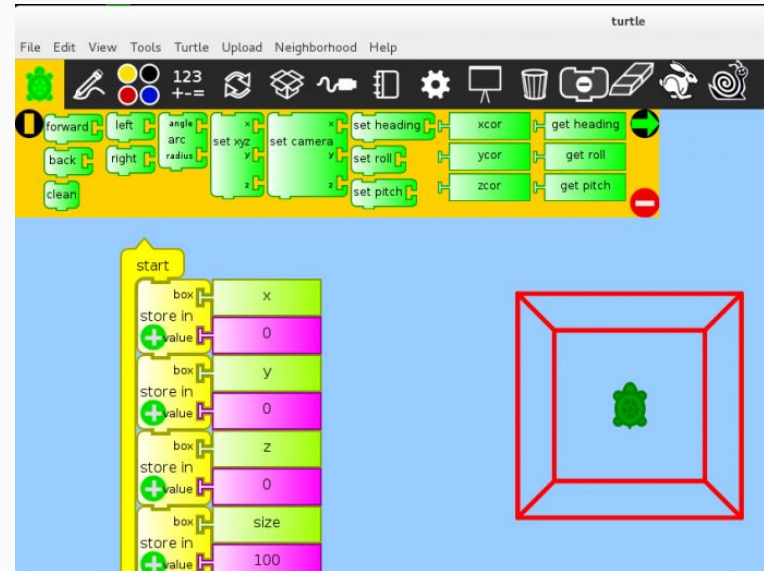


CSCI 362 Term Project: Sugar Desktop Turtle Blocks

Meagan Gould, Sam Ferguson, and Thomas Davis

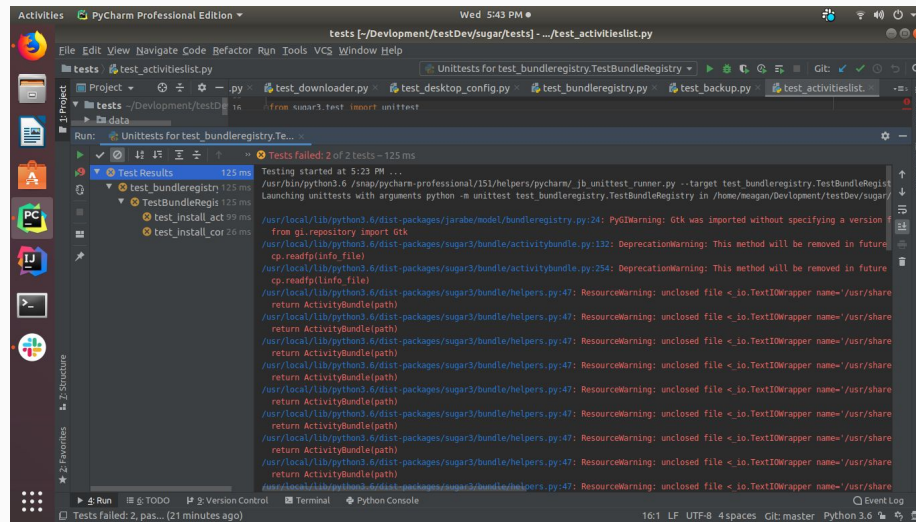
Sugar Desktop and Turtle Blocks

- Sugar Desktop is an environment for kids built in Linux
- Turtle Blocks teaches kids to program

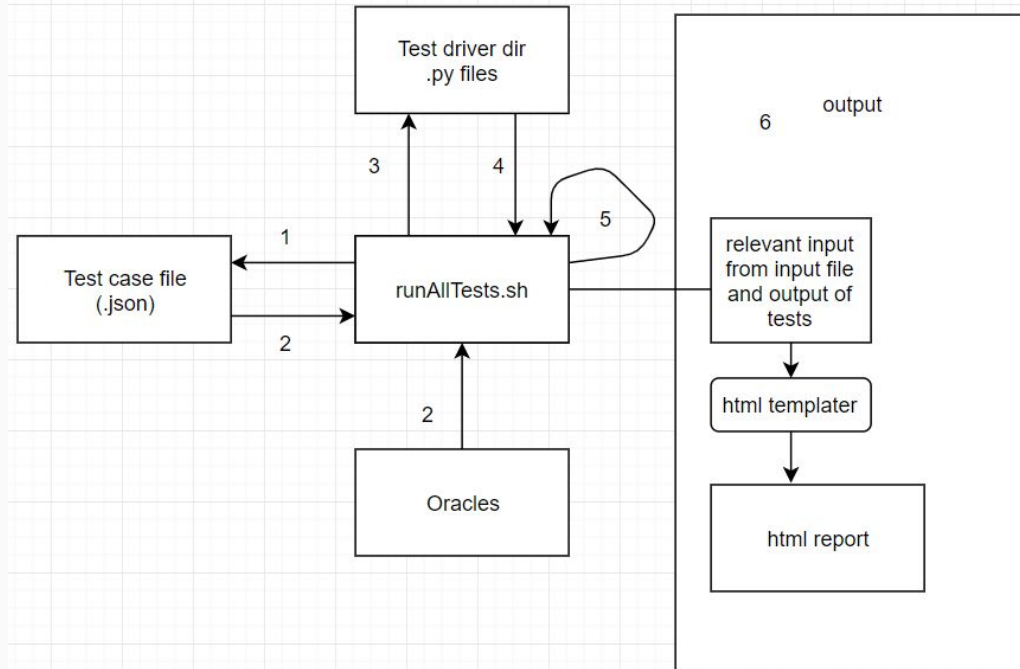


The Beginning - Struggles and Triumphs

- Requires many dependencies that are hard to download
- Was only able to run 4 test cases
- Found a sugar iso we were able to develop in



The Design of the Architecture



Important Areas

- runAllScripts.sh
 - Driver that runs the program
 - TestCase > TestCaseExecutables > reports > open in browser
- Drivers
 - Create different objects, runs the methods, and returns outputs to be tested
- HTML Template
 - Python library jinja2
 - Takes in json

runAllTests.sh libraries

Technology and structures used:

- jq
- jinja2
- xdg-utils

Test case example

Contains attributes that are pulled into the testing script

JSON objects can be made into bash variables with jq command

example --->

```
1  {
2      "test_id": 4,
3      "requirement": "Return None clicking on blank canvas",
4      "driver_name": "testCasesExecutables/testFindSprite.py",
5      "method_tested": "sprites.find_sprite()",
6      "inputs": [15,115],
7      "output": "None",
8      "extra_path":["project/TurtleBlocks/TurtleArt"]
9  }
```

```
test_id=$(jq .'test_id' $test_case)
import_dir=$(jq -r .'extra_path[]' $test_case)
requirement=$(jq .'requirement' $test_case)
inputs=$(jq .'inputs[]' $test_case)
driver_name=$(jq -r .'driver_name' $test_case)
method_tested=$(jq -r .'method_tested' $test_case)
expected_output=$(jq -r .'output' $test_case)
```

Drivers

Data necessary for testing process
sent to driver

```
{  
  "test_id": 14,  
  "requirement": "move the sprite x by 2 and move y by -3",  
  "driver_name": "testCasesExecutables/testMoveSprite.py",  
  "method_tested": "move_relative()",  
  "inputs": [15,115, 2,-3],  
  "output": "(17, 112)",  
  "extra_path":["project/TurtleBlocks/TurtleArt"]  
}  
  
#grab variables from the test case file  
test_id=$(jq -r '.test_id' $test_case)  
import_dir=$(jq -r '.extra_path[]' $test_case)  
requirement=$(jq -r '.requirement' $test_case)  
inputs=$(jq -r '.inputs[]' $test_case)  
driver_name=$(jq -r '.driver_name' $test_case)  
method_tested=$(jq -r '.method_tested' $test_case)  
expected_output=$(jq -r '.output' $test_case)
```

The diagram illustrates the data flow from a JSON test case to a driver script. Blue arrows connect the JSON fields to their corresponding shell variable assignments: "test_id" to test_id, "requirement" to requirement, "driver_name" to driver_name, "method_tested" to method_tested, "inputs" to inputs, "output" to expected_output, and "extra_path" to import_dir. A red arrow points from the test_case variable to the jq commands. A black arrow points from the expected_output variable to the output field in the driver script command.

```
x = int(sys.argv[1])  
y = int(sys.argv[2])  
move_x = int(sys.argv[3])  
move_y = int(sys.argv[4])
```

```
output=$(python $driver_name $inputs $import_dir)
```

The diagram shows the driver script command and its arguments. A black arrow points from the expected_output variable to the output field in the driver script command. A red arrow points from the test_case variable to the jq commands. A black arrow points from the driver_name variable to the driver_name argument in the driver script command. A red arrow points from the inputs variable to the inputs argument in the driver script command. A red arrow points from the import_dir variable to the import_dir argument in the driver script command.

Drivers

Issue with imports to drivers in
different locations ----->

```
import sys
sys.path.insert(0, sys.argv[-1])
import sprites
from tasprite_factory import SVG, svg_from_file, svg_str_to_pixmap
```

output passed through stdout
and formatted to JSON ----->

```
print ("\" + str(test_sprite_one.get_xy()) + "\"")
```

Reporting

```
echo "\"test_id\": \"$test_id\"," >> reports/output.json
echo "\"requirement\": \"$requirement\"," >> reports/output.json
echo "\"driver_name\": \"$driver_name\"," >> reports/output.json
echo "\"method_tested\": \"$method_tested\"," >> reports/output.json
echo "\"inputs\": \"${jq .'inputs' $test_case}\"," >> reports/output.json
echo "\"expected_output\": \"$expected_output\"," >> reports/output.json
```

```
{
  "pass_color": "green",
  "fail_color": "red",
  "results": [
    {
      "test_id": "1",
      "requirement": "Return sprite when clicked on",
      "driver_name": "testCasesExecutables/testFindSprite.py",
      "method_tested": "sprites.find_sprite()",
      "inputs": [ 15, 115, "yellow", 10, 100 ],
      "expected_output": "yellow",
      "actual_output": "yellow",
      "did_pass": true
    }
  ]
}
```

Jinja2

```
import jinja2
import sys
import json
import os

def main():

    with open("reports/output.json", "r") as data_file:
        json_string = data_file.read()
        json_obj = json.loads(json_string)

    environment = jinja2.Environment(loader=jinja2.FileSystemLoader(os.getcwd()))
    template_one = environment.get_template("reports/template.html")
    rendered_html = template_one.render(data=json_obj)
    output_file = open("reports/test_results.html", "w")
    output_file.write(rendered_html)
    output_file.close()

if __name__ == "__main__":
    main()
```

```
<table>

    <tr>

        <th>Test Case</th>
        <th>Method Tested</th>
        <th>Requirement Tested</th>
        <th>Inputs</th>
        <th>Output</th>
        <th>Expected Output</th>
        <th>Status</th>

    </tr>

{% for test_case in data.results %}
    <tr>

        <td style = "text-align: center;">{{test_case.test_id}}</td>
        <td>{{test_case.method_tested}}</td>
        <td>{{test_case.requirement}}</td>
        <td style = "text-align: center;">{{test_case.inputs}}</td>
        <td>{{test_case.actual_output}}</td>
        <td>{{test_case.expected_output}}</td>
        {% if test_case.did_pass %}
        <td class="passed">Pass</td>
        {% else %}
        <td class="failed">Fail</td>
        {% endif %}

    </tr>

{% endfor %}
</table>
<table>

    <tr>

        <th>TESTS PASSED</th>
        <th>TESTS FAILED</th>

    </tr>
    <tr>

        <td style = "text-align: center;">{{data.tests_passed}}</td>
        <td style = "text-align: center;">{{data.tests_failed}}</td>

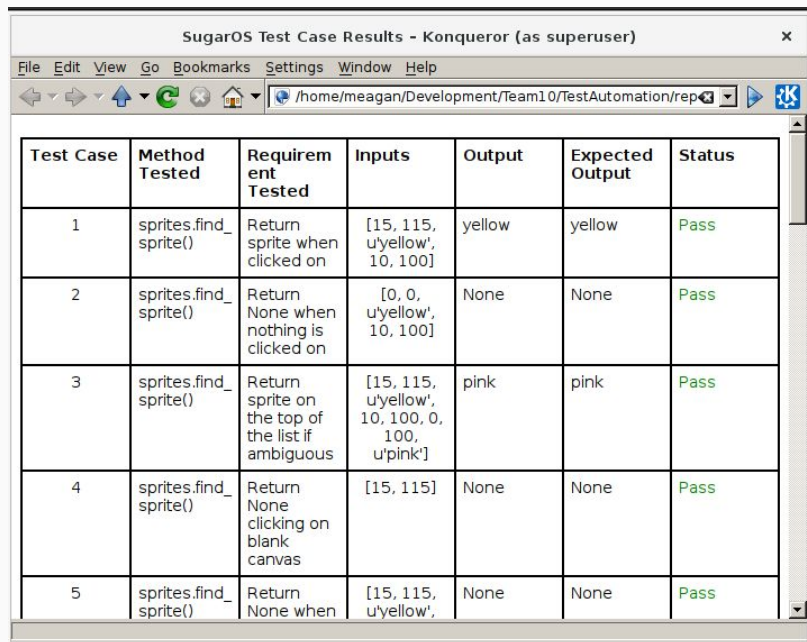
    </tr>
</table>
```

Output

data bound to template html

opened automatically with xdg-open

```
#Call the render_engine on the output.json to turn to html
python reports/render_engine.py
#open report with xdg
xdg-open reports/test_results.html &
```



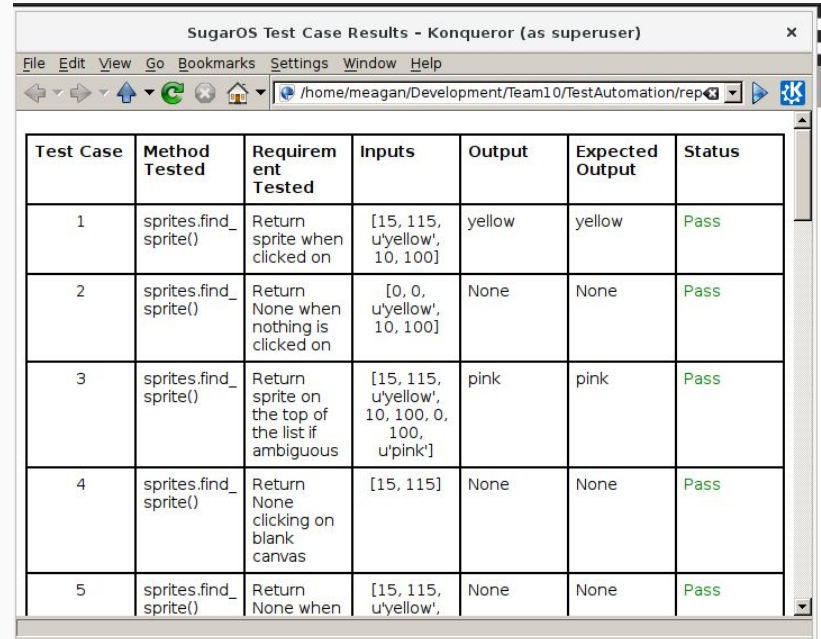
Test Case	Method Tested	Requirement Tested	Inputs	Output	Expected Output	Status
1	sprites.find_sprite()	Return sprite when clicked on	[15, 115, u'yellow', 10, 100]	yellow	yellow	Pass
2	sprites.find_sprite()	Return None when nothing is clicked on	[0, 0, u'yellow', 10, 100]	None	None	Pass
3	sprites.find_sprite()	Return sprite on the top of the list if ambiguous	[15, 115, 10, 100, 0, 100, u'pink']	pink	pink	Pass
4	sprites.find_sprite()	Return None clicking on blank canvas	[15, 115]	None	None	Pass
5	sprites.find_sprite()	Return None when	[15, 115, u'yellow',	None	None	Pass

Methods Tested

- `find_sprite()`
 - Sprite method
- `move_relative()`
 - Sprite method
- `spr_to_turtle()`
 - Turtle method

The Test Results Without Faults

- All tests pass when no faults are injected



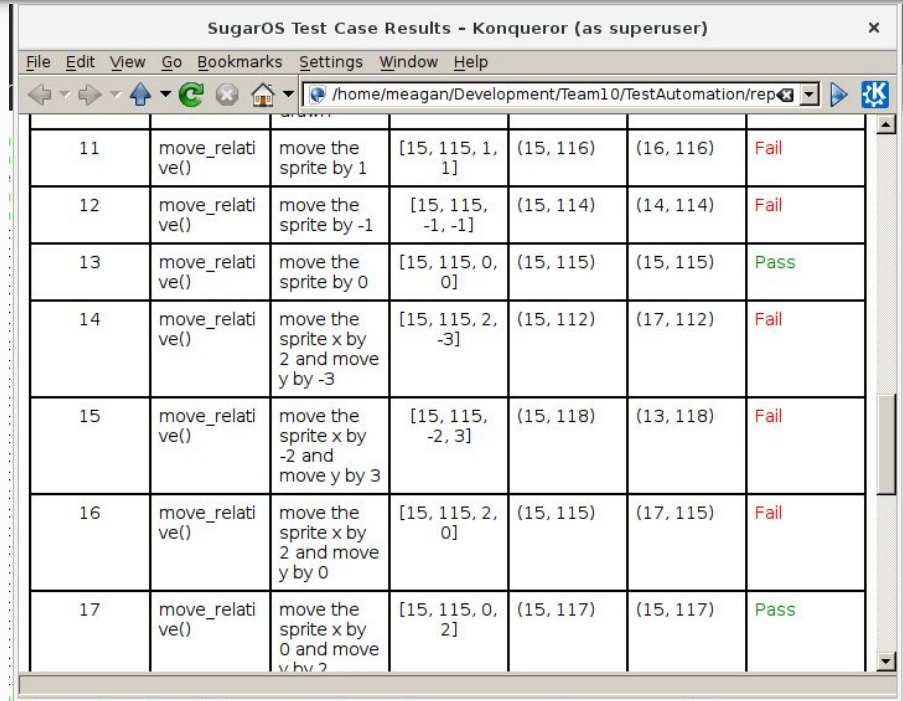

Test Case	Method Tested	Requirement Tested	Inputs	Output	Expected Output	Status
1	sprites.find_sprite()	Return sprite when clicked on	[15, 115, u'yellow', 10, 100]	yellow	yellow	Pass
2	sprites.find_sprite()	Return None when nothing is clicked on	[0, 0, u'yellow', 10, 100]	None	None	Pass
3	sprites.find_sprite()	Return sprite on the top of the list if ambiguous	[15, 115, u'yellow', 10, 100, 0, 100, u'pink']	pink	pink	Pass
4	sprites.find_sprite()	Return None clicking on blank canvas	[15, 115]	None	None	Pass
5	sprites.find_sprite()	Return None when	[15, 115, u'yellow',	None	None	Pass

The Test Results With Faults

- Quite a few of the test cases fail with the faults injected
- Took away certain lines of code
 - Addition sign
 - Conversion
 - If statement checks

SugarOS Test Case Results - Konqueror (as superuser)

File Edit View Go Bookmarks Settings Window Help

 /home/meagan/Development/Team10/TestAutomation/rep 

11	move_relative()	move the sprite by 1	[15, 115, 1, 1]	(15, 116)	(16, 116)	Fail
12	move_relative()	move the sprite by -1	[15, 115, -1, -1]	(15, 114)	(14, 114)	Fail
13	move_relative()	move the sprite by 0	[15, 115, 0, 0]	(15, 115)	(15, 115)	Pass
14	move_relative()	move the sprite x by 2 and move y by -3	[15, 115, 2, -3]	(15, 112)	(17, 112)	Fail
15	move_relative()	move the sprite x by -2 and move y by 3	[15, 115, -2, 3]	(15, 118)	(13, 118)	Fail
16	move_relative()	move the sprite x by 2 and move y by 0	[15, 115, 2, 0]	(15, 115)	(17, 115)	Fail
17	move_relative()	move the sprite x by 0 and move y by 2	[15, 115, 0, 2]	(15, 117)	(15, 117)	Pass

Reflections on the Project as a Whole

- Time management is definitely needed
- Documentation helps
- Glad it was python because you do not need to compile things
- Just tons of dependencies and figuring out how to run things as root

Questions?