Navid Hoque

CMPSC 470 Section 001

Nelson

29 April 2024

**Final Test Plan and Final Scripts to Implement the Test Suits and or Unit Tests**

**Test Plan**

**Objective:** Ensure that the car control functions and the script interpreter work as expected and handle all edge cases, including script errors and loop control.

**Scope:**

* Unit testing each car control function (**drive**, **stop**, **reverse**, **horn**).
* Testing the sensor reading functionality.
* Integration testing the script interpreter to handle Car Interpreter code correctly.

**Test Environment:**

* Node.js environment where the current JavaScript can execute.
* Use of a testing framework like Jest or Mocha for structured testing.

**Test Data:**

* Various scripts representing typical, boundary, and erroneous inputs for the interpreter.

**Test Cases:**

1. **Unit Tests:**
   * **Test Drive Function:**
     + *Description:* Ensure the **drive** function outputs the correct console message.
     + *Test:* Call **drive()** and expect "Car is driving forward."
   * **Test Stop Function:**
     + *Description:* Check if the **stop** function outputs the correct console message.
     + *Test:* Call **stop()** and expect "Car has stopped."
   * **Test Reverse Function:**
     + *Description:* Validate that the **reverse** function outputs the correct console message.
     + *Test:* Call **reverse()** and expect "Car is reversing."
   * **Test Horn Function:**
     + *Description:* Confirm that the **horn** function outputs the correct console sound.
     + *Test:* Call **horn()** and expect "Honk! Honk!"
   * **Test Sensor Read Function:**
     + *Description:* Ensure **read\_sensor** reads and outputs the correct sensor status.
     + *Test:* Call **read\_sensor()** and expect "obstacle."
2. **Integration Tests:**
   * **Basic Command Execution:**
     + *Description:* Test if basic commands (**drive();**, **stop();**, etc.) are executed correctly.
     + *Input:* "drive();\nstop();\nEND"
     + *Expected Output:* Should sequentially print "Car is driving forward." and "Car has stopped."
   * **Control Structure Handling (if, else, while):**
     + *Description:* Ensure control structures like **if**, **else**, and **while** are interpreted correctly.
     + *Input:* Scripts with conditional and loop constructs involving sensor checks and other commands.
     + *Expected Output:* Depending on conditions, appropriate sequences of actions are executed.
   * **Error Handling:**
     + *Description:* Verify that the interpreter gracefully handles syntax errors and unknown commands.
     + *Input:* "drve();\nEND"
     + *Expected Output:* "Error: Unknown token 'drve();' at line 1."
   * **Arithmetic and Comparison Operations:**
     + *Description:* Test arithmetic operations and variable assignments within the script.
     + *Input:* "x = 5 + 3;\ny = x \* 2;\nEND"
     + *Expected Output:* Proper execution without console errors and correct assignment if checked.

**Scripts to Implement Test Cases**

You can implement these tests using JavaScript testing frameworks like Jest. Here's an example of how you might write a test for the **drive** function:

const { drive } = require('./car\_interpreter'); // Path might need adjustment based on actual file structure

describe('Car Control Functions', () => {

test('drive function outputs correct message', () => {

console.log = jest.fn(); // Mocking console.log

drive();

expect(console.log).toHaveBeenCalledWith('Car is driving forward.');

});

});

To run this test suite effectively, integrate it into a Node.js project with the necessary Jest configurations. Each function and scenario described in the test cases above should be similarly structured within your testing framework.