

# Testing Framework of **Mars Map Maker**

---

Cormac Conahan, Robert Niggebrugge, Seth Hinson



# Table of Contents

- I. Introduction
- II. Explanation of the Testing Framework
- III. Setup Instructions
- IV. Examples
- V. Fault Injection
- VI. Reflections

## I. Introduction

This report provides information regarding the testing framework that “Best Team” has created for the open-source CIRDLES project “Mars Map Maker.” This report provides a detailed explanation of the testing framework, examples of test cases and their respective requirements and code, precise setup instructions for the framework, an explanation of the fault injection branch, and final reflections on the project and CSCI 362 as a whole.

Initially, a different open-source project, “Apertium,” had been chosen to act as the project being tested but was changed to Mars Map Maker after a mutual agreement was reached by all group members that it would be more beneficial to test a project with which one of the group members, Robert Niggebrugge, has had extensive experience.

The testing framework is non-exhaustive; it is more of a proof of concept for how a complete testing suite would be constructed for Mars Map Maker.

The individual roles of Best Team are as follows: Seth is the team lead, Cormac is the team scribe, and Robert is the main researcher. We all plan on taking part in the planning and coding of the testing platform, but the extra tasks required in a team environment require specialized roles. Seth handles the meetings, making sure deadlines are met and that reports are professional and complete, Cormac handles documenting the development process, everything from meeting notes to upholding the commit documentation standards on the team’s Github and Robert handles constructing creative solutions to arising problems via personal research or inter-team collaboration, all of which will be cited and used accordingly. Robert also serves as the main technical lead as his knowledge of Mars Map Maker means he is best at navigating the code and structure of the project.

## II. Explanation of the Testing Framework

The testing approach uses a combination of bash and javascript to create and run the scripts for testing Mars Map Maker. We also use Javascript as the driver because that is the language it is written in.

### Testing Architecture:

In the team's github, TestAutomation is the main folder where the framework lives. Once opened one will see docs, oracles, project, scripts, reports, temp, testCases, and testCasesExecutables.


**scripts: this directory is where the driver file and main scripts are located that execute the tests**

- runAllTests.sh: this file is the driver file, runs all needed scripts and commands needed to run the tests and opens the final results for viewing after the scripts complete
- runAllTests.js: this file creates creates the testRunner.js file inside of the MarsMapMaker project within its src/tests directory and creates the testOracles.js file
- testRunner.js: this file creates instances of the objects so that we can run the tests, it allows for inputs and outputs to be tested and verified. It also creates the actualResults.json file
- verifyResults.js: this file compares the actualResults.json file to the testOracles.json to make sure the expected outputs and actual outputs are the same for each test, thus a pass or fail

**oracles: this directory is where the oracle file(s) are created and stored during testing**

- testOracles.json: this file is created by the runAllTests.js and contains all of the test cases from the testCases directory

**testCases: this directory contains the json test cases files used during testing, they are structured with all the info needed for each test to be completed such as the module name, function name, input, and expected output. Each test case name is formatted as TestCaseN.json where N is the test case's specific ID number.**



**testCasesExecutables:** this directory contains the modules that are being tested (this is for ease of access for humans, these files are not used by the script and only allow for easy listing of the tested modules)

**reports:** this directory is where the **finalReport.css** is located, this is also where the **actualResults.json** and **finalReport.html** will be created during testing.

- **finalResults.css:** the css file for the final output
- **finalReport.html:** the html file containing the final results of the tests
- **actualResults.json:** this file contains the outputs of the functions we are testing, it will be compared to the **testOracles.json** file to get final results

### III. Test Case Examples

Below is a requirements table for the first 5 test cases. Each of the test cases follow the same format and unit-test similar pieces of code with arithmetic and logic tests. A complete requirements table is available on the team's GitHub.

ID	Component	Requirement	Method	Input	Expected Output
1	DropDown.js	No object found with key of "value" and non empty string pair, return -1.	entWithContent()	[]	-1
2	DropDown.js	No object found with key of "value" and non empty string pair, return -1.	entWithContent()	[[{value: ""}]]	-1
3	DropDown.js	Object found with key of "value" and non empty string pair, return the last index of the occurrence.	entWithContent()	[[{value: ""}], {value: "Foo"}]	1
4	CardList.js	Input Value is an integer, return "number".	typeField()	"8"	"number"
5	CardList.js	Input Value is empty string, return "both".	typeField()	""	"both"

The code format of any given test case is as follows:

"module": "DropDown", // chooses the module from which the tested function is retrieved

"ID": 1, // sets the test case ID

"functionName": "entWithContent", // chooses which function from the module to test

"metaData": "Must take in an array of objects and return the last index that contains non empty strings that are present with the key \"value\" or return -1", // short description of expected inputs and expected outputs

"metaDataShort": "DropDown.entWithContent(input) must return -1", // precise expectations of this test case

"input": [], // input for the function to test functionality described above

"expectedOutput": -1 // expected output of the function with given input

## IV. Setup Instructions

### Hardware Requirements:

- Physical machine with resources to run Linux Distribution or physical machine with resources to run Linux distribution via virtual machine.

### Software Requirements:

- Ubuntu 20.04.1 or greater or similar Linux distribution
- Node Package Manager
- Git

**Step 1:** Clone the repository "git clone <https://github.com/csci-362-01-2020/Best-Team>"

**Step 2:** Move into the TestAutomation/MarsMapMaker/ and run the command "git submodule update --init"

**Step 3:** In the same directory, run command "git pull <https://github.com/hafey1/MarsMapMaker>"

**Step 4:** Install npm "npm install"

**Step 5:** Once install is complete, change to the scripts directory and run the quickSetUp.sh script (bash quickSetUp.sh)

**Step 6:** Once the quick setup is complete, run the runAllTests.sh script (bash runAllTests.sh)

## V. Fault Injections

In order to properly determine whether test cases were properly passing test cases and not just letting everything pass, it is important to safely and procedurally inject faults into the code being tested such that it creates an error for the test suite to catch. This is exactly what is happening in the second branch of this testing framework, "mutationBranch."

While in the Best-Team folder type the command: "git checkout mutationBranch". This will move one into the branch where the faults have been injected so that one can run the framework on the newly faulted code. From there, move to the TestAutomation directory and run 'bash runAllTests.sh' to see that several of the test cases now fail.

**The following is what was changed in mutationBranch:**

### **Fault 1: Dropdown entWithContent**

In this method, the "!=" in the second if statement was changed to a "===", thus completely flipping what the method did. This change means that the method no longer returns the last index of a string in an array containing a specified value, but returns the last index of a string which doesn't contain that value (this also means the method only returns -1 on an empty array/non-array input.) The affected test cases are: 1, 2, 3, 7, 8, 9, and 10. The results change (failed) for cases 2, 3, and 7. Case 2 failed because it expected a -1 (because the only string in the array did not contain the value input), but instead returned 0 since the string at 0 did not contain the value. Case 3 failed because the test expected an output of 1 because the string at index 1 contained the value, but the code output a 0 since the string at index 0 was the last to not contain the value. Case 7 failed because the test expected a -1, since both strings in the array did not contain the value but the code output a 1 due to the final string (index 1) not having the value. Cases 1, 8, 9, and 10 all continued to pass because they failed the type check before the injected fault is reached, thus are unchanged.

### **Fault 2: CardList typeField**

In this method, "else if (numbers.test(f) === true)" was replaced with "else if (numbers.test(f) !== true)". This means that if an input is not "" or an added\_card (the two if statements before it) and isn't a number (most likely text), then the code will output that it is "numbers". If the input is made of numbers, then it will fail this conditional and return "text" falsely. Effected test cases: 4, 5, 23, 24. Test cases 4 and 23 failed while 5 and 24 did not change and still passed. Test case 4 failed because the input was a number and as stated above, the code output "text" rather than the expected "numbers". Similarly test case 23 failed because the input was a string of text, thus the



code output that it was "numbers" rather than "text". Test cases 5 and 24 passed because their types (empty string and added\_card respectively) were checked and returned before reaching the faulted if conditional.

### **Fault 3: FieldCard lengthCheckedValue**

In this method, "else if (value.length > 25)" was changed to "else if (value.length < 25)". This fault means that strings over length 25 will not be changed while strings under length 25 will be concatenated with "...". For example the code is meant to output "small string" when "small string" is input, but now outputs "small string..." and with strings larger than 25 characters, it just returns the same string and doesn't truncate them to 20 characters with a "..." at the end like it is meant to. Effected test cases: 11 and 13. Both test cases failed, 11 failed because a string smaller than 25 chars returned with a "..." added to it rather than just returning the input string and case 13 failed as a long string returned with no changes, where it was meant to truncate to only 20 chars with "..." at the end.

### **Fault 4: App findDuplicates**

In this method, if "(names[i] === names[j])" was changed to if "(names[i] !== names[j])". This means that instead of the method outputting all the duplicates in an array, it outputs all the unique values in that array (if the repeated values are not the only strings in the array). For example the code is meant to output [1,2,3] if the input is [1,2,3,1,2,3,4] but instead the fault makes it output [1,2,3,4]. Effected test cases: 6, 12, 14, 19, 20, 21, 22. Test cases 6, 14, and 20 failed with the fault. 6 failed because it had two strings that were duplicates in an array, since they were the only strings, the code output an empty array rather than outputting an array containing the value of the duplicated string. 14 failed because the array had 3 duplicates, but the code returned all 6 of the unique values contained in the array. 20 failed because the array contained no duplicates and the code output all of the unique strings rather than an empty array. 12, 19, 21, and 22 all passed because they were either inputting an empty array (case 12) or were incorrect input types (not an array, return an empty array).

### **Fault 5: FieldCard isMetaDataAddCard**

In this method, "let totalAddedCards = 4" was changed to "let totalAddedCards = 2". This change means that if the input card ID is greater than or equal to 2 the code will return false; 2 and 3 return false rather than return true. Effected test case: 17. Test case 17 did not fail as the tested input was not either of the two inputs that would most recently return false, thus the output of true was expected.

## VI. Reflections

Overall, this project has been greatly beneficial to each team member of Best Team. We've learned a great deal about teamwork and communication as well as modern standards of testing software. The work we've completed is definitely something each team member can be proud of. Thanks to a great deal of hard work from Robert, the driver script functions flawlessly for theoretically any number of test cases and the whole team put in a good deal of effort for finding methods which met the criteria for testing and creating their respective test cases. The team met often and did their best to communicate issues and solve problems together as well as documented every step of the way.

As for the course itself in terms of the project assignments, we all agree that the pacing was optimal and the requirements for each assignment were vital for a strong testing framework. However, there were definitely times when we should have sought more guidance and perhaps needed more guidance to begin with. This being said, that doesn't change the fact that this course has been an enjoyable challenge, and the Best Team would like to thank Dr. Bowring for making it so.