**SIT323 Cloud Application Development**

**Trimester 2, 2018**

**Assessment Task 1 – Programming Task 14**

Student Name : Richardo Wijaya

Student ID Number : 216408619

Partner : Connor Vernon

**Contents**

1. Introduction 3
2. Test Scenario 1 3
3. Test Scenario 2 4
4. Test Scenario 3 5
5. Test Scenario 4 7
6. Test Scenario 5 8
7. Test Scenario 6 9
8. Test Scenario 7 11
9. **Introduction**

Unit test result is a result of implementation of unit test design which was created from the old program. Implementation of unit test design uses the new program which has been updated according to the specific tasks which allows the program to be able to read .czl, .cfg, and .seq. By implementing the unit test design to the application, the developer can predict all possible input including false or predicted error to certain methods and prevent the error or out of bound input from happening.

In this crozzle application, 7 methods are to be tested throughout the program which range from checking that the value of the crozzle properties are the same within the crozzle files, analysing that the correct values can be retrieved from new changes and see that the

expected error message is displayed when there is a program breaking change.

1. **Test Scenario**
   1. Justifications
      1. Test case that failed

None of the Justification is failed

* + 1. Test case that was not implemented

There is not any other input beside anything and true and false.

* 1. Learning

What I learned from the test result on the isBoolean function is to transform a particular string to a Boolean value with a TryParse method inside it and if the string is not a Boolean value which are not true and false, the function returns as a false. From all the test case, it can be said that form of string except true and false

* 1. Test Scenario Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Scenario ID** | **Test Description** | | | | |
| 1 | Check that the strings are correctly interpreted as a Boolean value. If the expected and actual return values are true, the method works correctly to identify Boolean values. Expected and actual false returns mean that it can identify non-Boolean. | | | | |
| **Test Method** | **Method Tested** | | | | |
| UnitTest1.TestMethod1() | Boolean Validator.isBoolean(String field, out Boolean aBoolean) | | | | |
| **Test Case ID** | **Parameters** | **Expected Data** | **Actual Data** | **Test Result** | **Test Comments** |
| 1.1 | field = “false” | expectedReturn1 = true | actualReturn1=true | passed | “false” is a proper Boolean variable |
| 1.2 | field =”falsee” | expectedReturn2 = false | ActualReturn2=false | passed | Program returns false due to invalid Boolean value |

1. **Test Scenario 2**
   1. Justifications
      1. Test case that failed

None of the Justification is failed

* + 1. Test case that was not implemented

String character will create an error to the application so it was not implemented.

* 1. Learning

Similar to isBoolean(), this is almost the entirely the same thing but transform string to int. Any other string beside number cannot be transformed.

* 1. Test Scenario Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Scenario ID** | **Test Description** | | | | |
| 2 | Check that the strings are correctly interpreted as integer values. If the expected and actual return values are true, the method works correctly to identify Integer values. Expected and actual false returns mean that it can identify non-Integers. | | | | |
| **Test Method** | **Method Tested** | | | | |
| UnitTest1.TestMethod2() | Boolean Validator.isInt32(String field, out int anInteger) | | | | |
| **Test Case ID** | **Parameters** | **Expected Data** | **Actual Data** | **Test Result** | **Test Comments** |
| 2.1 | field = “25” | expectedReturn1 = true | actualReturn1=true | Passed | True as the string value can be interpreted as integer |
| 2.2 | field = “25.32” | expectedReturn2 = false | actualReturn2=false | Passed | Found that floating point numbers cannot be read |
| 2.3 | field = “-25” | expectedReturn3 = true | actualReturn3=true | Passed | True as the string value can be interpreted as integer |

1. **Test Scenario 3**
   1. Justifications
      1. Test case that failed

None of the Justification is failed

* + 1. Test case that was not implemented

String character will create an error to the application, so it was not implemented.

* 1. Learning

Similar to isBoolean() and isInt32(), this function transform string parameter to a hex color in which the string hex colour must exist. Any other string beside existing hex cannot be transformed.

* 1. Test Scenario Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Scenario ID** | **Test Description** | | | | |
| 3 | Check that the strings are correctly interpreted as colour hex values. If the expected and actual return values are true, the method works correctly to Hex values. Expected and actual false returns mean that it can identify illegal characters | | | | |
| **Test Method** | **Method Tested** | | | | |
| UnitTest1.TestMethod3() | Boolean Validator.isHexColourCode(String hexColour) | | | | |
| **Test Case ID** | **Parameters** | **Expected Data** | **Actual Data** | **Test Result** | **Test Comments** |
| 3.1 | hexColour = “#2ecc71” | expectedReturn1 = true | actualReturn1 = true | Passed | The colour Green. Passed as it met the requirements of REGEX |
| 3.2 | hexColour = “##ecc71” | expectedReturn2 = false | actualReturn2 = false | Passed | False due to illegal symbol after the initial hash tag |
| 3.3 | hexColour = “#2ecc7g” | expectedReturn3 = false | actualReturn3 = false | Passed | False due to a character being out of range |

1. **Test Scenario 4**
   1. Justifications
      1. Test case that failed

None of the Justification is failed

* + 1. Test case that was not implemented

All the form of input has been attempted anything beside that will create an error.

* 1. Learning

This case is used to take result after an equal sign and whatever it is as long as it is not a null or empty, it can be parsed out. This helped me learn how to find the word with certain pattern and move what is inside the equal sign and put it in one variable.

* 1. Test Scenario Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Scenario ID** | **Test Description** | | | | |
| 4 | Checking that the string value passed matches a key-value pair and returns true if the string meets the conditions of the REGEX key pattern | | | | |
| **Test Method** | **Method Tested** | | | | |
| UnitTest1.TestMethod4() | Boolean KeyValue.TryParse(String originalKeyValueData, String keyPattern, out KeyValue aKeyValue) | | | | |
| **Test Case ID** | **Parameters** | **Expected Data** | **Actual Data** | **Test Result** | **Test Comments** |
| 4.1 | originalKeyValueData = "MAXIMUM = 100"; KeyPattern = @"[a-zA-Z0-9]"; | expectedReturn1 = true | actualReturn1 = true | Passed | True because String passed has a equals sign |
| 4.2 | originalKeyValueData = "MAXIMUM=";  KeyPattern = @"[a-zA-Z0-9]"; | expectedReturn2 = false | actualReturn2 = false | Passed | False due to no value given for one of the pairs |
| 4.3 | originalKeyValueData = "MAXIMUM= ";  KeyPattern = @"[a-zA-Z0-9]"; | expectedReturn3 = true | actualReturn3 = true | Passed | Whitespace added after “=”. |

1. **Test Scenario 5**
   1. Justifications
      1. Test case that failed

None of the Justification is failed

* + 1. Test case that was not implemented

There is not any other input beside the same one, lower > upper and lower < upper.

* 1. Learning

This function check for how many duplicated word can be in this crozzle. This kind of validation taught me to be more precise and certain error notification must be mentioned to support this.

* 1. Test Scenario Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Scenario ID** | **Test Description** | | | | |
| 5 | Checking that the Crozzle application can successfully find duplicates within the Crozzle. If errors are detected, the application method can successfully log errors if the duplicate word count exceeds the max limit specified in the config files. | | | | |
| **Test Method** | **Method Tested** | | | | |
| UnitTest1.TestMethod6() | void Crozzle.CheckDuplicateWords(int lowerLimit, int upperLimit) | | | | |
| **Test Case ID** | **Parameters** | **Expected Data** | **Actual Data** | **Test Result** | **Test Comments** |
| 6.1 | lowerLimit = 1  upperLimit = 1 | expectedFalseErrors = false | crozzleSequences.ErrorsDetected = false | Passed | No errors can be detected in the original files |
| 6.2 | lowerLimit = 1  upperLimit = 2 | expectedFalseErrorsNew = false | crozzleSequences2.ErrorsDetected = false | Passed | No errors as two duplicates allowed in new file |
| 6.3 | lowerLimit = 2  upperLimit = 1 | expectedTrueErrorsNew = true | crozzleSequences3.ErrorsDetected = true | Passed | Errors returned as duplicates exceed max duplicates |

1. **Test Scenario 6**
   1. Justifications
      1. Test case that failed

None of the Justification is failed

* + 1. Test case that was not implemented

These all the input we can think of

* 1. Learning

The return result of ToStringHTML was incredibly long so it was difficult to compare the expected value and the actual value during the debug. It helped me learn that unit tests can be conducted by just changing the position of one of the methods in the unit tests as for this test unit, the output of the HTML string relied on the validation method.

* 1. Test Scenario Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Scenario ID** | **Test Description** | | | | |
| 6 | Confirming that the method returns the correct string value when the validate function has been initiated or not. The string value is dependent on different settings in each file. If the expected value matches the actual value, the application returns the correct results based off changes such as different scores. | | | | |
| **Test Method** | **Method Tested** | | | | |
| UnitTest1.TestMethod8() | String Crozzle.ToStringHTML() | | | | |
| **Test Case ID** | **Parameters** | **Expected Data** | **Actual Data** | **Test Result** | **Test Comments** |
| 8.1 | n/a | expectedAfterValidate = [1] | crozzle.ToStringHTML() = [1] | Passed | Crozzle file is valid when Validate method called before saving and receiving HTML |
| 8.2 | n/a | expectedBeforeValidate = [2] | crozzle.ToStringHTML() = [2] | Passed | Crozzle file invalid when Validate method called before saving and receiving HTML |
| 8.3 | n/a | expectedNewHTML = [3] | crozzle.ToStringHTML() = [3] | Passed | Correct output from new files which has a different score of 336 |

1. **Test Scenario 7**
   1. Justifications
      1. Test case that failed

None of the Justification is failed

* + 1. Test case that was not implemented

Multiple range of group count has been attempted. Minimum wasn’t attempted because it doesn’t really affect with the group count

* 1. Learning

Designing these unit tests got me in the habit to ensure that the GroupCount function can always get the correct number of groups specified in the crozzle files. All the tests cases were very similar in that it was getting the total amount of groups but the repetition was important in case there was error when the crozzle hit a certain amount of total groups. I also found out findings that the score got lower as the more groups there were in the crozzle. Test case 1.2 had two groups with a score of 274 and test case 1.3 had four groups with a score of 247. This was due to less intersecting points as most of the points came from them. For future unit testing, I could have conducted tests to see if the correct error messages could be returned when the group count was out of range from the values specified.

* 1. Test Scenario Form

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Scenario ID** | **Test Description** | | | | |
| 7 | Checking that the crozzle application can successfully count the number of groups of connected words in the application. The test is conducted on different changes to the crozzle files. If the expected matches the actual value, the application can successfully identify the number of groups of connected words. | | | | |
| **Test Method** | **Method Tested** | | | | |
| UnitTest1.TestMethod9() | int CrozzleMap.GroupCount() | | | | |
| **Test Case ID** | **Parameters** | **Expected Data** | **Actual Data** | **Test Result** | **Test Comments** |
| 9.1 | n/a | expectedGroupCount = 1 | crozzleMap.GroupCount() = 1 | Passed | All words are connected on crozzle hence 1 group |
| 9.2 | n/a | expectedNewGroupCount1 = 2 | crozzleMap2.GroupCount() = 2 | Passed | Crozzle has two separate groups hence group count 2 |
| 9.3 | n/a | expectedNewGroupCount2 = 4 | crozzleMap3.GroupCount() = 4 | Passed | Crozzle split into 4 different groups hence group count 4 |

Appendix -