Task 1 – Unit Testing

Test Scenario ID	**Test Description**				
1	Check attributes of a newly constructed Grid object.				
Test Method	**Method Tested**				
UnitTest1.TestMethod1()	Grid.Grid(String rawGrid)				
Test Case ID	**Parameters**	**Expected Data**	**Actual Data**	**Test Result**	**Test Comments**
1.1	testGrid = "APPLE\r\n" + "N E\r\n" + "TRIAL\r\n" + " F"	expectedRows = 4	grid.Rows = 4	pass	
1.2	testGrid = "APPLE\r\n" + "N E\r\n" + "TRIAL\r\n" + " F"	expectedColumns = 5	grid.Columns = 5	pass	
1.3	testGrid = "APPLE\r\n" + "N E\r\n" + "TRIAL\r\n" + " F"	expectedRow1 = "APPLE"	grid.GetRow(1) = "APPLE"	pass	
1.4	testGrid = "APPLE\r\n" + "N E\r\n" + "TRIAL\r\n" + " F"	expectedRow2 = "N E "	grid.GetRow(2) = "N E "	pass	
1.5	testGrid = "APPLE\r\n" + "N E\r\n" + "TRIAL\r\n" + " F"	expectedRow3 = "TRIAL"	grid.GetRow(3) = "TRIAL"	pass	
1.6	testGrid = "APPLE\r\n" + "N E\r\n" + "TRIAL\r\n" + " F"	expectedRow4 = " F "	grid.GetRow(4) = " F "	pass	

Test Scenario ID	Test Description				
2	Check that the intersections are correct for a 4x5 grid.				
Test Method	**Method Tested**				
UnitTest1.TestMethod2()	String Grid.GetIntersections()				
Test Case ID	**Parameters**	**Expected Data**	**Actual Data**	**Test Result**	**Test Comments**

2.1	new Grid(`"APPLE\r\n" +`
	`"N  E\r\n" +`
	`"TRIAL\r\n" +`
	`"   F")`
		expectedIntersections = "ALTA"	intersections = "ALTA"	pass	

Test Scenario ID	Test Description				
3	Check that the intersections are correct for a 12x8 grid.				
Test Method	**Method Tested**				
UnitTest1.TestMethod3()	String Grid.GetIntersections()				
Test Case ID	**Parameters**	**Expected Data**	**Actual Data**	**Test Result**	**Test Comments**

3.1	new Grid(`"APPLE   PEAR\r\n" +`
	`"N  E    A  A\r\n" +`
	`"TRIAL   N  I\r\n" +`
	`"E  FUN  DAWN\r\n" +`
	`"L   N   A\r\n" +`
	`"O   A\r\n" +`
	`"POLAR\r\n" +`
	`"E")`
		expectedIntersections = "ALPRTALFUDNPR"	intersections = "ALPRTALFUDNPR"	pass	

Task 2 – Unit Test Design

Test Scenario ID	**Test Description**				
1	Check a value against a range, inclusive.				
Test Method	**Method Tested**				
	Boolean Validator.TryRange(int n, int lowerLimit, int upperLimit)				
Test Case ID	**Parameters**	**Expected Data**	**Actual Data**	**Test Result**	**Test Comments**
1.1	n = 123 lowerLimit = 100 upperLimit = 200	expectedReturn = true			
1.2	n = 98 lowerLimit = 100 upperLimit = 200	expectedReturn = false			
1.3	n = 321 lowerLimit = 100 upperLimit = 200	expectedReturn = false			
1.4	n = 99 lowerLimit = 100 upperLimit = 200	expectedReturn = false			
1.5	n = 100 lowerLimit = 100 upperLimit = 200	expectedReturn = true			
1.6	n = 101 lowerLimit = 100 upperLimit = 200	expectedReturn = true			
1.7	n = 199 lowerLimit = 100 upperLimit = 200	expectedReturn = true			
1.8	n = 200 lowerLimit = 100 upperLimit = 200	expectedReturn = true			
1.9	n = 201 lowerLimit = 100 upperLimit = 200	expectedReturn = false			

Task 3 – Unit Test Implementation

```
[TestMethod]
public void TestMethod1()
{
   // Arrange.
   int lower = 100;
   int upper = 200;
   int withinRange = 123;
   int belowRange = 98;
   int aboveRange = 321;
   int belowLowerLimit = 99;
   int equalLowerLimit = 100;
   int aboveLowerLimit = 101;
   int belowUpperLimit = 199;
   int equalUpperLimit = 200;
   int aboveUpperLimit = 201;
   Boolean expectedReturnWithin = true;
   Boolean expectedReturnBelow = false;
   Boolean expectedReturnAbove = false;
   Boolean expectedReturnBelowLower = false;
   Boolean expectedReturnEqualLower = true;
   Boolean expectedReturnAboveLower = true;
   Boolean expectedReturnBelowUpper = true;
   Boolean expectedReturnEqualUpper = true;
   Boolean expectedReturnAboveUpper = false;

   // Act.
   Boolean actualReturnWithin = Validator.TryRange(withinRange, lower, upper);
   Boolean actualReturnBelow = Validator.TryRange(belowRange, lower, upper);
   Boolean actualReturnAbove = Validator.TryRange(aboveRange, lower, upper);
```

```
    Boolean actualReturnBelowLower = Validator.TryRange(belowLowerLimit, lower, upper);
    Boolean actualReturnEqualLower = Validator.TryRange(equalLowerLimit, lower, upper);
    Boolean actualReturnAboveLower = Validator.TryRange(aboveLowerLimit, lower, upper);
    Boolean actualReturnBelowUpper = Validator.TryRange(belowUpperLimit, lower, upper);
    Boolean actualReturnEqualUpper = Validator.TryRange(equalUpperLimit, lower, upper);
    Boolean actualReturnAboveUpper = Validator.TryRange(aboveUpperLimit, lower, upper);

    // Assert.
    Assert.AreEqual(expectedReturnWithin, actualReturnWithin, "failed ...");
    Assert.AreEqual(expectedReturnBelow, actualReturnBelow, "failed ...");
    Assert.AreEqual(expectedReturnAbove, actualReturnAbove, "failed ...");
    Assert.AreEqual(expectedReturnBelowLower, actualReturnBelowLower, "failed ...");
    Assert.AreEqual(expectedReturnEqualLower, actualReturnEqualLower, "failed ...");
    Assert.AreEqual(expectedReturnAboveLower, actualReturnAboveLower, "failed ...");
    Assert.AreEqual(expectedReturnBelowUpper, actualReturnBelowUpper, "failed ...");
    Assert.AreEqual(expectedReturnEqualUpper, actualReturnEqualUpper, "failed ...");
    Assert.AreEqual(expectedReturnAboveUpper, actualReturnAboveUpper, "failed ...");
}
```

Test Scenario ID	Test Description				
1	Check a value against a range, inclusive.				
Test Method	**Method Tested**				
UnitTest1.TestMethod1()	Boolean Validator.TryRange(int n, int lowerLimit, int upperLimit)				
Test Case ID	**Parameters**	**Expected Data**	**Actual Data**	**Test Result**	**Test Comments**
1.1	n = 123 lowerLimit = 100 upperLimit = 200	expectedReturn = true	actualReturnWithin = true	pass	
1.2	n = 98 lowerLimit = 100 upperLimit = 200	expectedReturn = false	actualReturnBelow = false	pass	
1.3	n = 321 lowerLimit = 100 upperLimit = 200	expectedReturn = false	actualReturnAbove = false	pass	
1.4	n = 99 lowerLimit = 100 upperLimit = 200	expectedReturn = false	actualReturnBelowLower = false	pass	
1.5	n = 100 lowerLimit = 100 upperLimit = 200	expectedReturn = true	actualReturnEqualLower = true	pass	
1.6	n = 101 lowerLimit = 100 upperLimit = 200	expectedReturn = true	actualReturnAboveLower = true	pass	
1.7	n = 199 lowerLimit = 100 upperLimit = 200	expectedReturn = true	actualReturnBelowUpper = true	pass	
1.8	n = 200 lowerLimit = 100 upperLimit = 200	expectedReturn = true	actualReturnEqualUpper = true	pass	
1.9	n = 201 lowerLimit = 100 upperLimit = 200	expectedReturn = false	actualReturnAboveUpper = false	pass	

Task 4 – Unit Test Design

Test Scenario ID	**Test Description**				
1	Check that strings are correctly interpreted as Hex colour codes.				
Test Method	**Method Tested**				
	Boolean Validator.IsHexColourCode(String hexColour)				
Test Case ID	**Parameters**	**Expected Data**	**Actual Data**	**Test Result**	**Test Comments**
1.1	hexColour = “#000000”	true			
1.2	hexColour = “#AA0000”	true			
1.3	hexColour = “#FFFFFF”	true			
1.4	hexColour = “000000”	false			
1.5	hexColour = “#00000X”	false			
1.6	hexColour = “#!@#$%^”	false			

Test Scenario ID	**Test Description**				
2	Check that strings are correctly interpreted as a filename.				
Test Method	**Method Tested**				
	Boolean Validator.IsFilename(String name)				
Test Case ID	**Parameters**	**Expected Data**	**Actual Data**	**Test Result**	**Test Comments**
2.1	name = “Test1.czl”	true			
2.2	name = “SIT323\Test1.czl”	true			
2.3	name = “2018\SIT323\Test1.czl”	true			
2.4	name = “Test???.czl”	false			
2.5	name = “<Test1.czl>”	false			
2.6	name = “SIT323|Test1.czl”	false			

Task 5 – Unit Test Implementation

```
[TestMethod]
public void TestMethod4()
{
   // Arrange.
   String hexBlack = "#000000";
   String hexRedish = "#AA0000";
   String hexWhite = "#FFFFFF";
   String invalid1 = "000000";
   String invalid2 = "#00000X";
   String invalid3 = "#!@#$%^";

   Boolean expectedReturnBlack = true;
   Boolean expectedReturnRedish = true;
   Boolean expectedReturnWhite = true;
   Boolean expectedReturnInvalid1 = false;
   Boolean expectedReturnInvalid2 = false;
   Boolean expectedReturnInvalid3 = false;

   // Act.
   Boolean actualReturnBlack = Validator.IsHexColourCode(hexBlack);
   Boolean actualReturnRedish = Validator.IsHexColourCode(hexRedish);
   Boolean actualReturnWhite = Validator.IsHexColourCode(hexWhite);
   Boolean actualReturnInvalid1 = Validator.IsHexColourCode(invalid1);
   Boolean actualReturnInvalid2 = Validator.IsHexColourCode(invalid2);
   Boolean actualReturnInvalid3 = Validator.IsHexColourCode(invalid3);
```

```
        // Assert.
        Assert.AreEqual(expectedReturnBlack, actualReturnBlack, "failed ...");
        Assert.AreEqual(expectedReturnRedish, actualReturnRedish, "failed ...");
        Assert.AreEqual(expectedReturnWhite, actualReturnWhite, "failed ...");
        Assert.AreEqual(expectedReturnInvalid1, actualReturnInvalid1, "failed ...");
        Assert.AreEqual(expectedReturnInvalid2, actualReturnInvalid2, "failed ...");
        Assert.AreEqual(expectedReturnInvalid3, actualReturnInvalid3, "failed ...");
    }
```

Test Scenario ID	**Test Description**				
1	Check that strings are correctly interpreted as Hex colour codes.				
Test Method	**Method Tested**				
UnitTest1.TestMethod4()	Boolean Validator.IsHexColourCode(String hexColour)				
Test Case ID	**Parameters**	**Expected Data**	**Actual Data**	**Test Result**	**Test Comments**
1.1	hexColour = “#000000”	expectedReturnBlack = true	true	pass	
1.2	hexColour = “#AA0000”	expectedReturnRedish = true	true	pass	
1.3	hexColour = “#FFFFFF”	expectedReturnWhite = true	true	pass	
1.4	hexColour = “000000”	actualReturnInvalid1 = false	false	pass	
1.5	hexColour = “#00000X”	actualReturnInvalid2 = false	false	pass	
1.6	hexColour = “#!@#$%^”	actualReturnInvalid3 = false	false	pass	

```
[TestMethod]
public void TestMethod5()
{
   // Arrange.
   String filename = @"Test1.czl";
   String filenameHyphen = @"SIT323-Test1.czl";
   String filenameSpace = @"SIT323 Test1.czl";
   String filenamePipe = @"Test|.czl";
   String filenameAngleBrackets = @"<Test1.czl>";
   String filenameDoubleQuotes = @"SIT323""Test1.czl";

   Boolean expectedReturnFilename = true;
   Boolean expectedReturnFilenameHyphen = true;
   Boolean expectedReturnFilenameSpace = true;
   Boolean expectedReturnFilenamePipe = false;
   Boolean expectedReturnFilenameAngleBrackets = false;
   Boolean expectedReturnFilenameDoubleQuotes = false;

   // Act.
   Boolean actualReturnFilename = Validator.IsFilename(filename);
   Boolean actualReturnFilenameHyphen = Validator.IsFilename(filenameHyphen);
   Boolean actualReturnFilenameSpace = Validator.IsFilename(filenameSpace);
   Boolean actualReturnFilenamePipe = Validator.IsFilename(filenamePipe);
   Boolean actualReturnFilenameAngleBrackets = Validator.IsFilename(filenameAngleBrackets);
   Boolean actualReturnFilenameDoubleQuotes = Validator.IsFilename(filenameDoubleQuotes);

   // Assert.
   Assert.AreEqual(expectedReturnFilename, actualReturnFilename, "failed ...");
   Assert.AreEqual(expectedReturnFilenameHyphen, actualReturnFilenameHyphen, "failed ...");
   Assert.AreEqual(expectedReturnFilenameSpace, actualReturnFilenameSpace, "failed ...");
   Assert.AreEqual(expectedReturnFilenamePipe, actualReturnFilenamePipe, "failed ...");
```

```
    Assert.AreEqual(expectedReturnFilenameAngleBrackets, actualReturnFilenameAngleBrackets, "failed ...");
    Assert.AreEqual(expectedReturnFilenameDoubleQuotes, actualReturnFilenameDoubleQuotes, "failed ...");
}
```

Test Scenario ID	**Test Description**				
2	Check that strings are correctly interpreted as a filename.				
Test Method	**Method Tested**				
UnitTest1.TestMethod5()	Boolean Validator.IsFilename(String name)				
Test Case ID	**Parameters**	**Expected Data**	**Actual Data**	**Test Result**	**Test Comments**
2.1	name = @“Test1.czl”	expectedReturnFilename = true	true	pass	
2.2	name = @“SIT323-Test1.czl”	expectedReturnFilenameHyphen = true	true	pass	
2.3	name = @“SIT323 Test1.czl”	expectedReturnFilenameSpace = true	true	pass	
2.4	name = @“Test|.czl”	expectedReturnFilenamePipe = false	false	pass	
2.5	name = @“<Test1.czl>”	expectedReturnFilenameAngleBrackets = false	false	pass	
2.6	name = @“SIT323””Test1.czl”	expectedReturnFilenameDoubleQuotes = false	false	pass	