CS 7140 - Advanced Software Engineering

Fall 2020

Project Test Documentation

Team Members:

Lancius (Lance) Matthieu

Abhishek Pandya

Griffin Mosley

1. Introduction

This test document describes that our software is working fine. We have implemented some smoke tests and a stress test which tells about how our software gives proper output for every ASCII art.

2. Referenced Documents

2.1 DITAA

The main bulk of code for this project will be used from DITAA. DITAA allows for this transition between ASCII and rendered bitmap images. DITAA can be found with the link below.
https://github.com/stathissideris/ditaa

2.2 Count Lines of Code (CLOC)

CLOC is the software that the team utilized to measure the number of lines of code in the DITAA project, as a whole and by class. CLOC counts blank lines, comment lines, and physical lines of source code in the Java programming language, as well as others.
http://cloc.sourceforge.net/
https://github.com/AlDanial/cloc

2.3 IntelliJ IDEA Statistics Plugin

Statistics is a plugin for the IntelliJ IDEA integrated development environment. This plugin was utilized as a second tool to measure the number of lines of code in the DITAA project, as a whole and by class. Statistics counts blank lines, comment lines, physical lines of source code, the number and location of each file function, and the number of file types.
https://plugins.jetbrains.com/plugin/12415-statistics

3. Smoke Testing

3.1 Course Program Functionality

To ensure the DITAA program is able to execute at a course level, two separate smoke tests will be run. The main purpose of this test is to verify that the program is able to execute at a course level before proceeding with more stringent tests.

The first test is to verify the DITAA program runs with no input and the appropriate help menu is displayed when run. The command that will be used for this purpose can be seen below.

$ java -jar ditaa.jar

The second test that will be run will be with a simple, verified correct, input ascii art text file (Example 6.txt). This test is to confirm that the DITAA program can run when given a correct input file and the expected output png is rendered. The command that will be used for this purpose can be seen below:

$ java -jar ditaa.jar “Example 6.txt”

3.2 Smoke Test #1

Smoke test #1 is the running of the DITAA program with no file input. The expected result is the output of the help menu to the console.

3.2.1 Smoke Test #1 Input

```
java -jar .\ditaa.jar
```

3.2.2 Smoke Test #1 Output

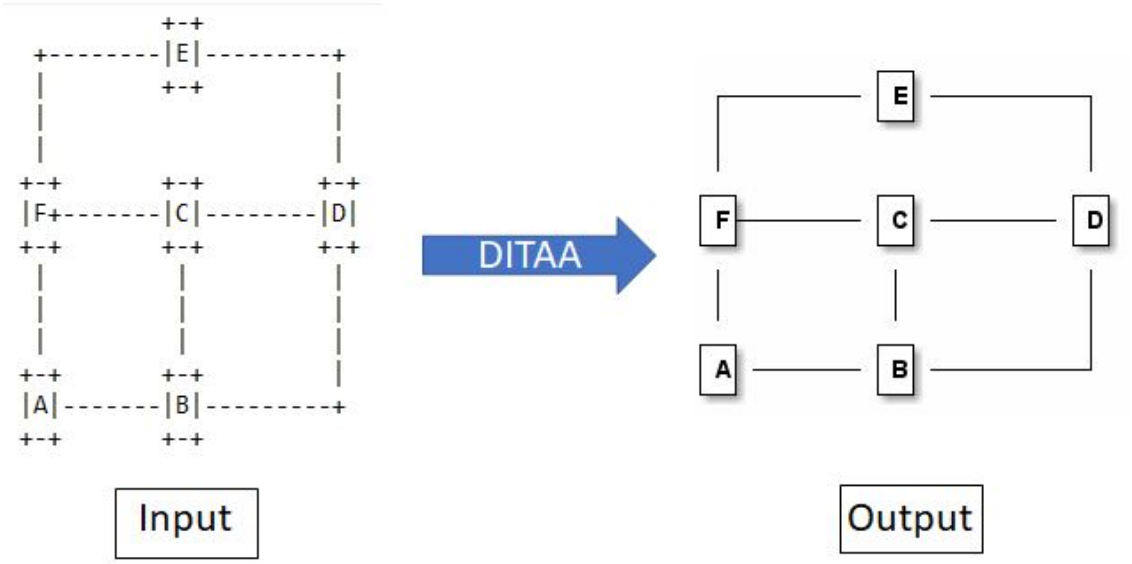
```
usage: java -jar ditaa.jar <INPFILE> [OUTFILE] [-A] [-b <BACKGROUND>] [-d]
       [-E] [-e <ENCODING>] [-h] [--help] [-o] [-r] [-S] [-s <SCALE>]
       [--svg] [--svg-font-url <FONT>] [-T] [-t <TABS>] [-v] [-W]
 -A,--no-antialias               Turns anti-aliasing off.
 -b,--background <BACKGROUND>    The background colour of the image. The
                                 format should be a six-digit hexadecimal
                                 number (as in HTML, FF0000 for red). Pass
                                 an eight-digit hex to define transparency.
                                 This is overridden by --transparent.
 -d,--debug                      Renders the debug grid over the resulting
                                 image.
 -E,--no-separation              Prevents the separation of common edges of
                                 shapes.
 -e,--encoding <ENCODING>        The encoding of the input file.
 -h,--html                       In this case the input is an HTML file.
                                 The contents of the <pre
                                 class="textdiagram"> tags are rendered as
                                 diagrams and saved in the images directory
                                 and a new HTML file is produced with the
                                 appropriate <img> tags.
    --help                       Prints usage help.
 -o,--overwrite                  If the filename of the destination image
                                 already exists, an alternative name is
                                 chosen. If the overwrite option is
                                 selected, the image file is instead
                                 overwriten.
 -r,--round-corners              Causes all corners to be rendered as round
                                 corners.
 -S,--no-shadows                 Turns off the drop-shadow effect.
 -s,--scale <SCALE>              A natural number that determines the size
                                 of the rendered image. The units are
                                 fractions of the default size (2.5 renders
                                 1.5 times bigger than the default).
    --svg                        Write an SVG image as destination file.
    --svg-font-url <FONT>        SVG font URL.
 -T,--transparent                Causes the diagram to be rendered on a
                                 transparent background. Overrides
                                 --background.
 -t,--tabs <TABS>                Tabs are normally interpreted as 8 spaces
                                 but it is possible to change that using
                                 this option. It is not advisable to use
                                 tabs in your diagrams.
 -v,--verbose                    Makes ditaa more verbose.
 -W,--fixed-slope                Makes sides of parallelograms and
                                 trapezoids fixed slope instead of fixed
                                 width.
```

3.2.3 Result - SUCCESS

The output of the DITAA program was what was expected and thus smoke test #1 is considered successful.

3.3 Smoke Test #2

Smoke test #2 is the running of the DITAA program with a simple, verified correct, ascii art text file. The input file contents (Example 6.txt) and the expected output can be seen below.

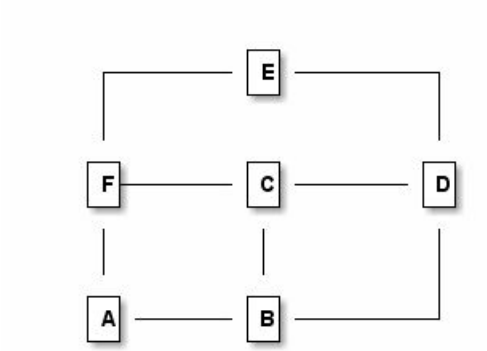


3.3.1 Smoke Test #1 Input

```
java -jar .\ditaa.jar 'F:\Users\lance\Downloads\Example 6.txt'
```

3.3.2 Smoke Test #1 Output

```
ditaa version 0.11, Copyright (C) 2004--2017  Efstathios (Stathis) Sideris

Running with options:
Reading file: F:\Users\lance\Downloads\Example 6.txt
Rendering to file: F:\Users\lance\Downloads\Example 6.png
Done in 0sec
```



3.3.3 Result - SUCCESS

The output of the DITAA program was what was expected and thus smoke test #2 is considered successful.

4. White Box Testing

JUnit testing will be used to white box test 2 critical classes of the DITAA program, TextGrid and CellSet.

4.1 TextGrid Class

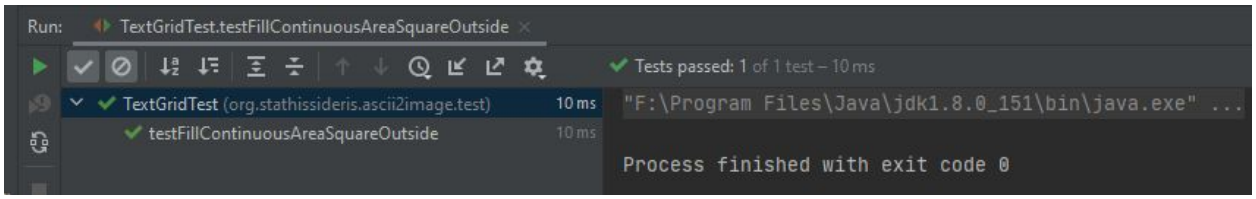
4.1.1 testFillContinuousAreaSquareOutside()

The goal of this test is to verify that a connected square that is read and parsed from a file is the same as when created programmatically. A text file containing a simple closed square (simple_square01.txt) is used to initially create a filled area. This simple square is filled and tested to be accurate before moving on to programmatically producing the same expected square and testing that they are equal..Below are the contents of simple_square01.txt and the JUnit code to test this case respectively.

```
+-----+
|     |
|     |
|     |
+-----+
```

```java
@Test public void testFillContinuousAreaSquareOutside() throws FileNotFoundException, IOException {
    TextGrid squareGrid;
    squareGrid = new TextGrid();
    //squareGrid.loadFrom("..\\..\\..\\..\\..\\test-resources\\text\\simple_square01.txt");
    squareGrid.loadFrom( filename: "F:\\Users\\lance\\Documents\\WSU\\CS 7140 - Advanced Software Engineering\\" +
            "ditaa\\test-resources\\text\\simple_square01.txt");

    CellSet filledArea = squareGrid.fillContinuousArea( x: 0,  y: 0,  c: '*');
    int size = filledArea.size();
    assertEquals( expected: 64, size);

    CellSet expectedFilledArea = new CellSet();
    addSquareToCellSet(squareGrid, expectedFilledArea,  x: 0, y: 0,  width: 11, height: 2);
    addSquareToCellSet(squareGrid, expectedFilledArea,  x: 0, y: 7,  width: 11, height: 2);
    addSquareToCellSet(squareGrid, expectedFilledArea,  x: 0, y: 2,  width: 2, height: 5);
    addSquareToCellSet(squareGrid, expectedFilledArea,  x: 9, y: 2,  width: 2, height: 5);
    assertEquals(expectedFilledArea, filledArea);
}
```

The output of running this test case through the IntelliJ IDEA IDE can be found below.



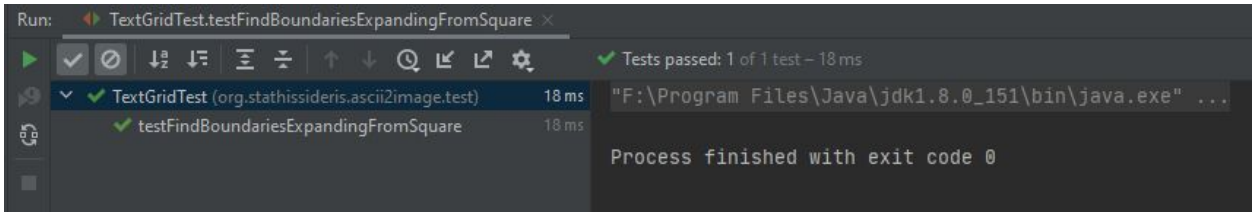
4.1.2 testFindBoundariesExpandingFromSquare()

The goal of this test is to verify that the boundaries of a connected square parsed from a file is the same as a connected square created programmatically. A text file containing a simple closed square (simple_square01.txt) is used to initially create and determine the boundary size. Below are the contents of simple_square01.txt and the JUnit code to test this case respectively.

```
+-----+
|     |
|     |
|     |
+-----+
```

```java
@Test public void testFindBoundariesExpandingFromSquare() throws FileNotFoundException, IOException {
    TextGrid grid;
    grid = new TextGrid();
    //grid.loadFrom("tests/text/simple_square01.txt");
    grid.loadFrom( filename: "F:\\Users\\lance\\Documents\\WSU\\CS 7140 - Advanced Software Engineering\\" +
            "ditaa\\test-resources\\text\\simple_square01.txt");

    CellSet wholeGridSet = new CellSet();
    addSquareToCellSet(grid, wholeGridSet,  x: 0, y: 0, grid.getWidth(),grid.getHeight());

    TextGrid copyGrid = new AbstractionGrid(grid, wholeGridSet).getCopyOfInternalBuffer();
    CellSet boundaries = copyGrid.findBoundariesExpandingFrom(copyGrid.new Cell( x: 8,  y: 8));
    int size = boundaries.size();

    boundaries.printAsGrid();
    assertEquals( expected: 56, size);

    CellSet expectedBoundaries = new CellSet();

    addSquareToCellSet(copyGrid, expectedBoundaries,  x: 8,  y: 7,  width: 17, height: 1);
    addSquareToCellSet(copyGrid, expectedBoundaries,  x: 8, y: 19,  width: 17, height: 1);

    addSquareToCellSet(copyGrid, expectedBoundaries,  x: 7,  y: 8,  width: 1, height: 11);
    addSquareToCellSet(copyGrid, expectedBoundaries, x: 25,  y: 8,  width: 1, height: 11);

    expectedBoundaries.printAsGrid();
    assertEquals(expectedBoundaries, boundaries);

}
```

The output of running this test case through the IntelliJ IDEA IDE can be found below.



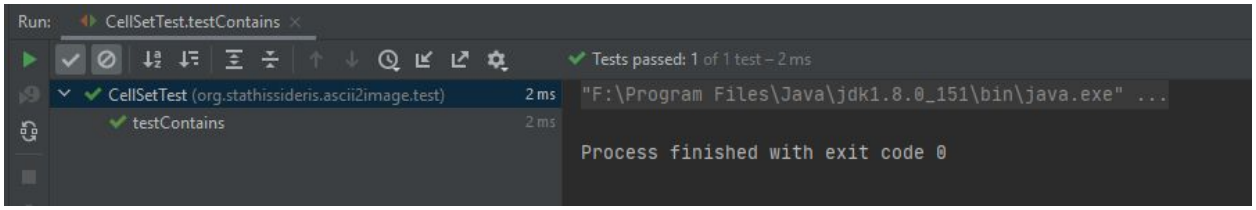
4.2 CellSet Class

4.2.1 testContains()

The goal of this test is to verify that cells are correctly constructed and stored by the TextGrid class, to include duplicate cells. All cells are created programmatically. Below is the JUnit code to test this case. Note that the entire class is shown to include the necessary setup method for testing.

```java
public class CellSetTest {

    TextGrid g = new TextGrid();
    CellSet set = new CellSet();

    @Before public void setUp() {
        set.add(g.new Cell( x: 10,  y: 20));
        set.add(g.new Cell( x: 10,  y: 60));
        set.add(g.new Cell( x: 10,  y: 30));
        set.add(g.new Cell( x: 60,  y: 20));
    }

    @Test public void testContains() {
        TextGrid.Cell cell1 = g.new Cell( x: 10,  y: 20);
        TextGrid.Cell cell2 = g.new Cell( x: 10,  y: 20);

        assertTrue(cell1.equals(cell2));
        assertTrue(set.contains(cell1));
    }
}
```

The output of running this test case through the IntelliJ IDEA IDE can be found below.



5. Black Box Testing

A number of black box tests will be performed by providing input files to the DITAA program and comparing the rendered output png, to the expected output. A number of tests are detailed below.

5.1 Simple Shapes

5.1.1 simple_square01.txt

Input:

```
+------+
|      |
|      |
|      |
+------+
```

Execution:

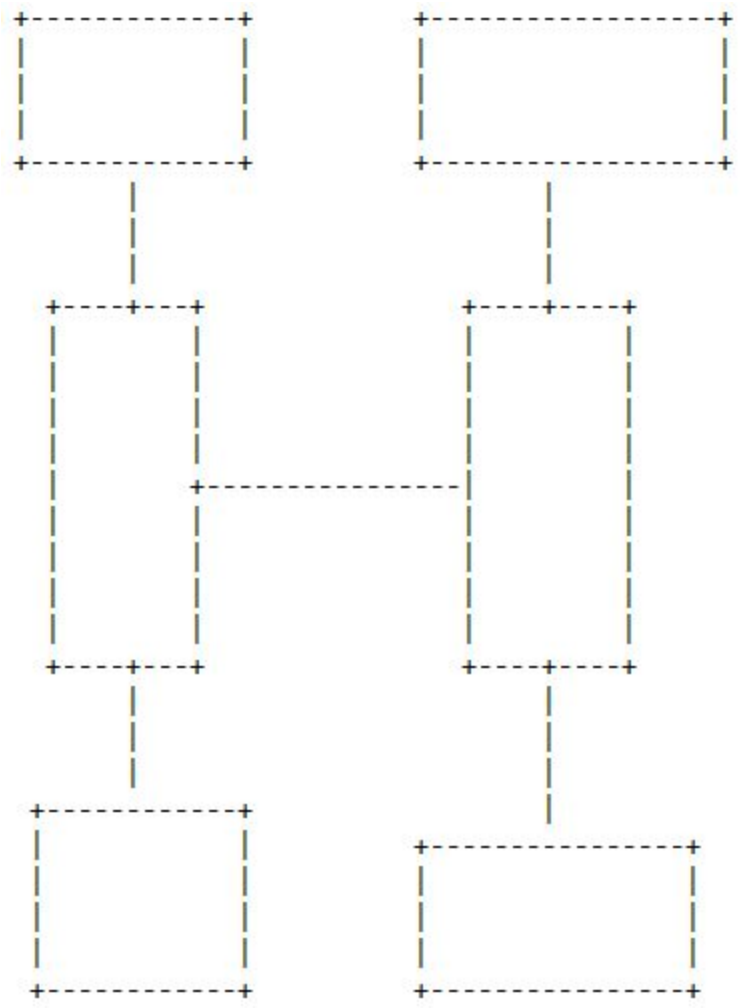
```
java -jar .\ditaa.jar '..\..\test-resources\text\simple_square01.txt' ..\..\..\test_doc\results\simple_square01.png
ditaa version 0.11, Copyright (C) 2004--2017  Efstathios (Stathis) Sideris

Running with options:
Reading file: ..\..\test-resources\text\simple_square01.txt
Rendering to file: ..\..\..\test_doc\results\simple_square01.png
Done in 0sec
```

Output:

Result: **SUCCESS**

5.1.2 Example_7.txt

Input:

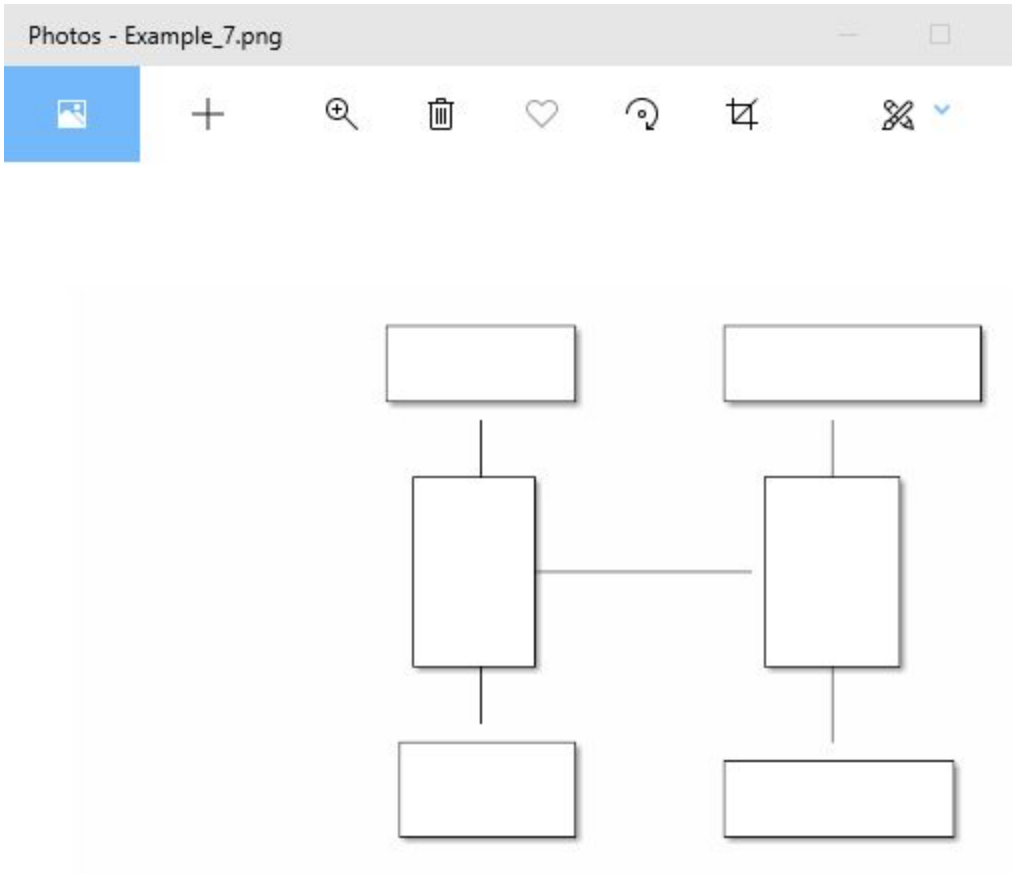


Execution:

```
java -jar .\ditaa.jar '..\..\test-resources\text\Example 7.txt' ..\..\..\test_doc\results\Example_7.png
```

```
ditaa version 0.11, Copyright (C) 2004--2017  Efstathios (Stathis) Sideris

Running with options:
Reading file: ..\..\test-resources\text\Example 7.txt
Rendering to file: ..\..\..\test_doc\results\Example_7.png
Done in 1sec
```

Output:

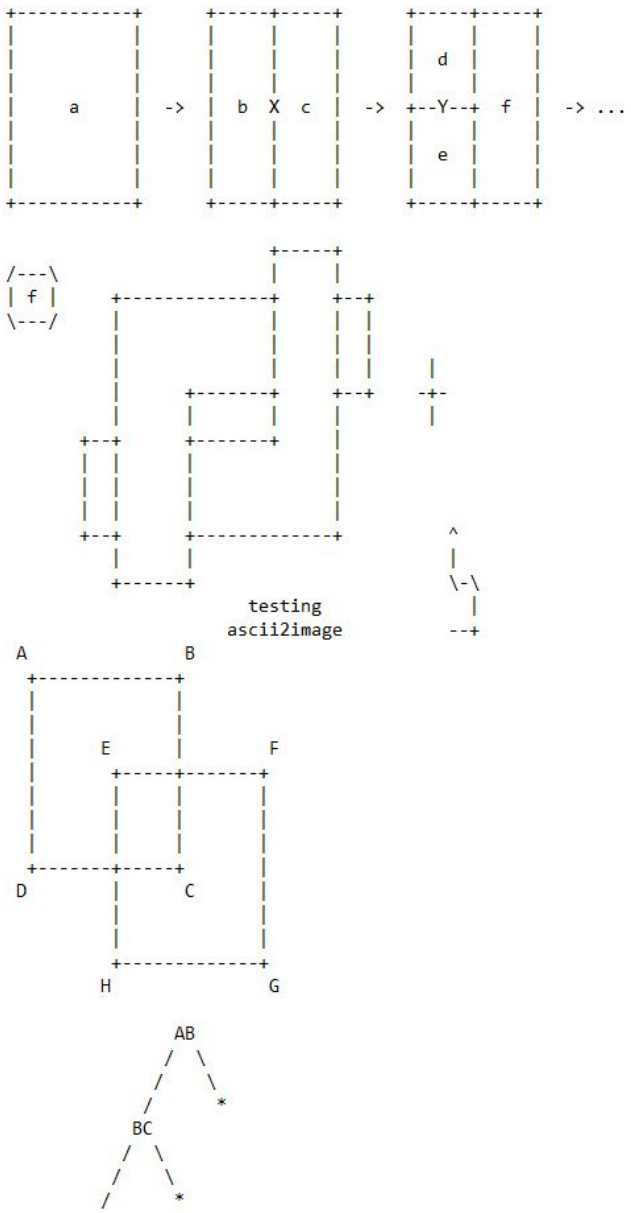


Result: SUCCESS

5.2 Advanced Shapes

5.2.1 art1.txt

Input:

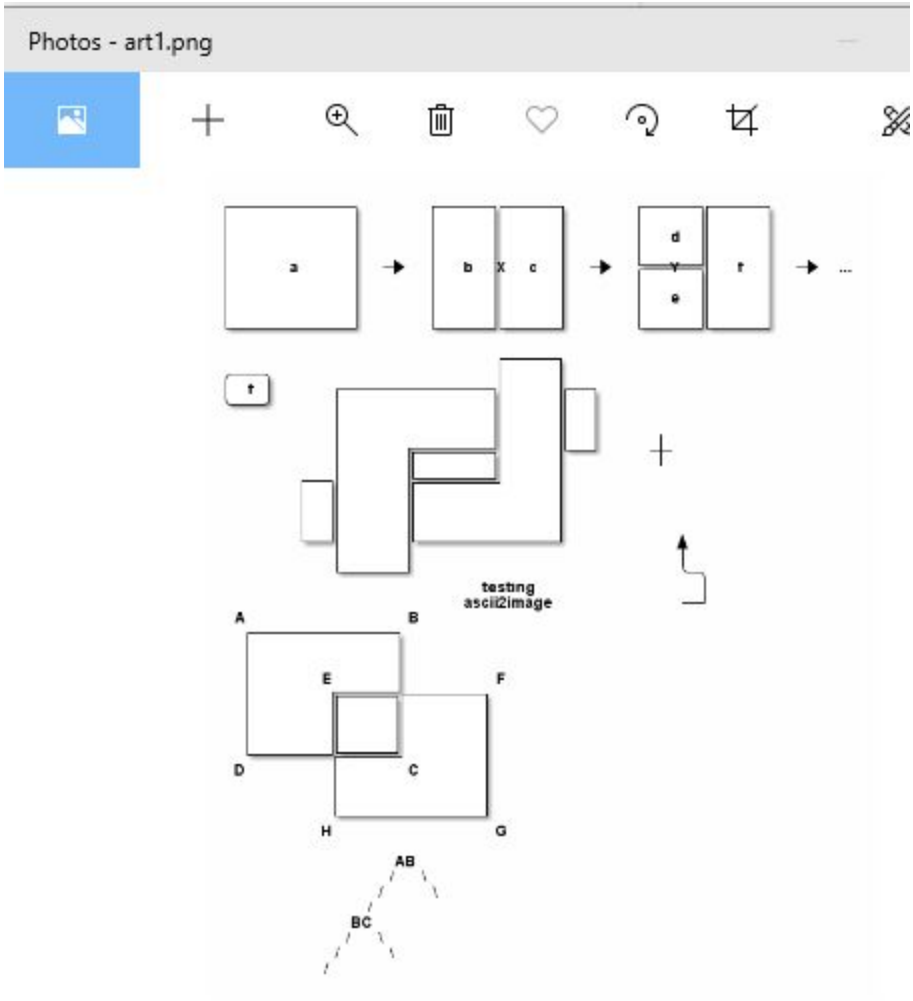


Execution:

```
java -jar .\ditaa.jar '..\..\test-resources\text\art1.txt' ..\..\..\test_doc\results\art1.png
```

```
ditaa version 0.11, Copyright (C) 2004--2017  Efstathios (Stathis) Sideris

Running with options:
Reading file: ..\..\test-resources\text\art1.txt
Rendering to file: ..\..\..\test_doc\results\art1.png
Done in 1sec
```

Output:



Result: **SUCCESS**

5.2.2 bug9_5.txt

Input:

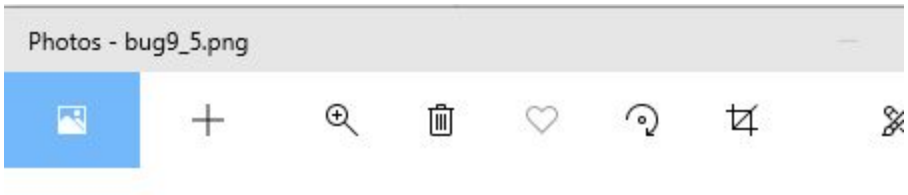
```
   Cell
  +-----+
  |  A  |           |     B
C *--*--+-----------+-----*
  |     |           |
  +-----+
```

Execution:

```
java -jar .\ditaa.jar '..\..\test-resources\text\bug9_5.txt' ..\..\..\test_doc\results\bug9_5.png
```

```
ditaa version 0.11, Copyright (C) 2004--2017  Efstathios (Stathis) Sideris

Running with options:
Reading file: ..\..\test-resources\text\bug9_5.txt
Rendering to file: ..\..\..\test_doc\results\bug9_5.png
Done in 0sec
```

Output:



Result: FAIL

Discussion:

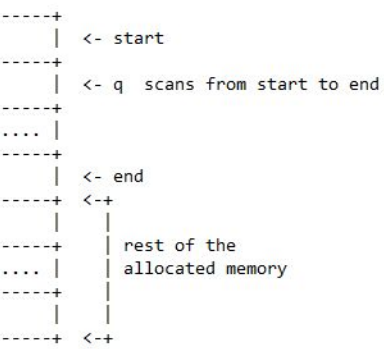
Expected output is that a single connected box would be created, not two. Initial analysis suggests that problem is occurring within Diagram::seperateCommonEdges().

6. Stress Test

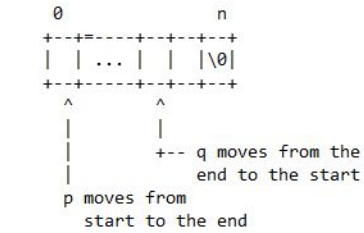
A stress test was performed on the DITAA program in order to see how it would handle a large file size. An ascii art text file was generated that had repeated content (see below) to produce a single large file size compared to the previous test. Previous tests performed would run in the single digit seconds, whereas the stress took 40 seconds on the same compute hardware.

Input (copied multiple times in a single file):

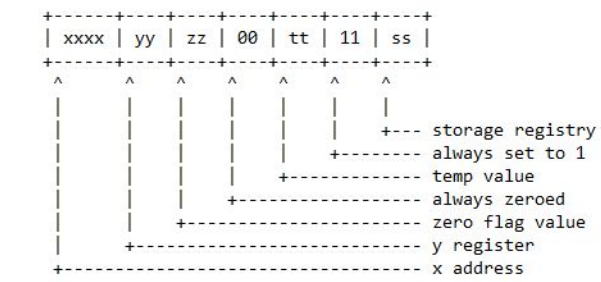
Memory:



Strings:



Sectioned:

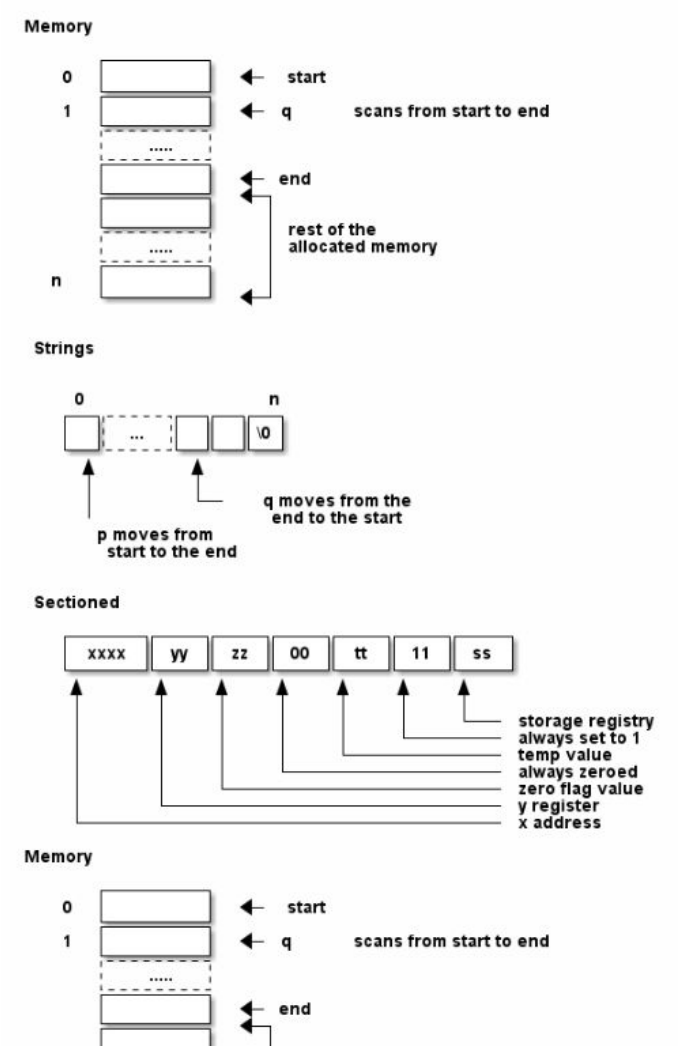


Execution:

```
java -jar .\ditaa.jar '..\..\test-resources\text\Stress Test.txt' ..\..\..\test_doc\results\Stress_Test.png
```

```
ditaa version 0.11, Copyright (C) 2004--2017  Efstathios (Stathis) Sideris

Running with options:
Reading file: ..\..\test-resources\text\Stress Test.txt
Rendering to file: ..\..\..\test_doc\results\Stress_Test.png
Done in 40sec
```

Output:



Result: **SUCCESS**

7. Acceptance Test

The following specifications were utilized for acceptance testing. The following is a summarization of the specification tested, which file it was tested by, and the following result. All tests must be successful for the program to be accepted for release.

Specification	File Tested By	Result
Horizontal Line {‘-’, ‘=’}	art2.txt, art10.txt	**Success**
Vertical Line {‘|’, ‘:’}	art2.txt, art10.txt	**Success**
Corner Characters {‘+’, ‘/’, ‘\’}	art2.txt, art10.txt	**Success**

Closed Shape	art2.txt, art10.txt	**Success**
Dashed Lines {':', '='}	art10.txt	**Success**
Closed Shape + Input Text	art2.txt, art10.txt	**Success**
Closed Shape + Color (Hex)	color_codes_hex.txt	**Success**
Closed Shape + Color (Predefined)	art2.txt, art10.txt	**Success**
Closed Shape + "{d}"	art10.txt	**Success**
Closed Shape + "{s}"	art10.txt	**Success**
Closed Shape + "{io}"	art10.txt	**Success**
Closed Shape + Bullet "o" + Text	art2.txt	**Success**
Arrowhead - Down	art2.txt	**Success**
Arrowhead - Right	art2.txt, art10.txt	**Success**
Arrowhead - Left	art2.txt	**Success**
Arrowhead - Right	art2.txt	**Success**

8. Additional Requirements

8.1 Code Reduction

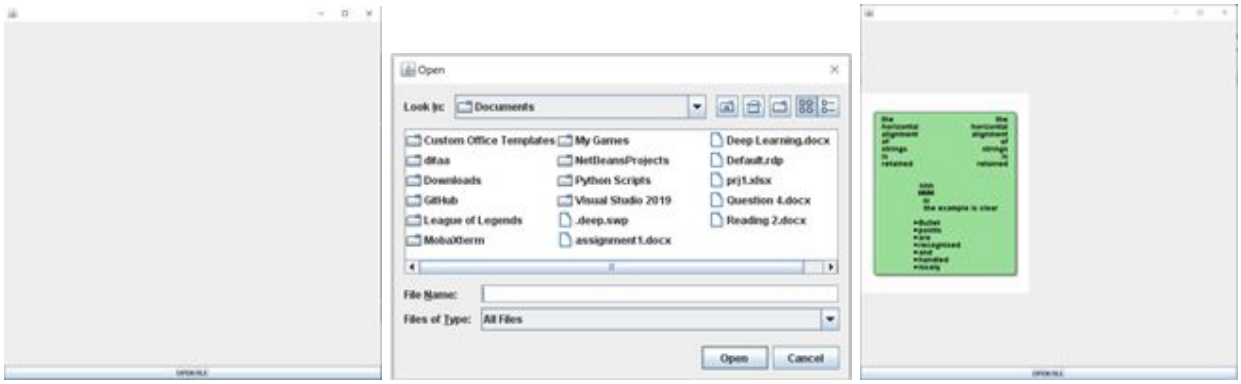
Refer to the implementation section for more information on the reduction of code size.

8.2 Maintainability

Refer to the implementation section for more information on code reduction and its impact on the overall maintainability of the code.

8.3 Graphical User Interface

A GUI was developed as a more accessible front-end for the DITAA program. Below is a screenshot of the developed GUI.


Open
Look In:
Documents
Custom Office Templates
dlfaa
Downloads
GitHub
League of Legends
MobaXterm
My Games
NetBeansProjects
Python Scripts
Visual Studio 2019
.deep.swp
assignment1.docx
Deep Learning.docx
Default.rdp
prj1.xlsx
Question 4.docx
Reading 2.docx
File Name:
Files of Type:
All Files
Open
Cancel