***Group 4 Final Document***

***Abstract***

***Introduction to the document***

***Description of the company***

Group 4 solutions is a new company made up of four individuals from different backgrounds to provide turnkey solutions to your software needs. We have around ten years coding experience within Java in academia and industry, working on numerous different types of projects. We have a strong background in delivering solutions on time and within budget by using strong software engineering techniques. As a company we believe that to give the client what they need we must first understand the requirements and we take a large amount of time making sure that we know what the requirements are. Using our experience in industry we are able to talk to business’s about their company and help them to accurately describe what they want. Once we know what is required we are able to use our industry and academic experience to find elegant and sophisticated solutions which we are able to deliver on time.

The company is owned and operated by four equal shareholders all with differing experiences which combine to make Group 4 solutions a high value proposition for business’s looking at their software needs. Wang is the technical director and has many years’ experience working on technical projects in Java in academia and industry. He drives the company forward looking at cutting edge techniques to ensure that we deliver the best and most suitable products. Donal has been in academia for many years working on complex problems in physics and maths. He has strong problem solving skills and is our testing director, he is responsible for making sure that all of products have no bugs and fulfil the requirements. Donovan is an all-round technical guy, he is responsible for working on all technical aspects of the project. He has a huge role in the design of the project by working on the diagrams which ensure that we will not go over time and over budget. Richard has many years’ industrial experience working for the end user, he is skilled in project management and organisation/administration. His main tasks involve liaising with all the parties and ensuring the brief is correct and keeping lines of communication open all the way through the project cycle.

***Quality section on tools, schedule***

Design for the project will be done in Visual Paradigm, a UML-based software that is widely used and has multiple versions depending on the budget of the user and the intended use to which it will be put, including a free version, which will be used for this project. Visual Paradigm supports multiple types of diagrams and is more than sufficient for the purposes of this project, since it will be unlikely that the program produced will be exceptionally complex.

The language that will be used to program this project is Java, which is one of the most widely used programming languages in the world, being free, extremely well documented and the subject of one of the modules studied by the team in the previous semester, meaning that all team members should share at least a basic level of familiarity with it. Java is also capable of producing programs that will run on multiple different operating systems, which will increase the utility of the final product.

The programming of the project will be completed in Eclipse, a Java-based IDE that is very widely used and with which one member of the team is very familiar, which should prevent time being lost to becoming familiar with the tool. Eclipse is free and well-documented, and has both Mac and Windows versions, which will allow all members of the team to work on the programming using their own computers.

Version control and source code management for the project will be managed via GitHub, a Git repository hosting service that is very widely used, free, and with which a member of the team is very familiar, once again reducing time lost to learning how to use a new tool. GitHub also provides a desktop client for both Windows and Mac which allows synchronization with the remote repository and uploading of local changes, which should enable all members of the team to easily contribute to the code and follow changes made by others.

For GUI design and implementation within the project, the built-in Java AWT library will be used. Being a part of the standard Java API for providing GUIs, AWT is very well-supported and requires no additional installation, which will streamline the process for creating a GUI.

***User Requirements***

* A 2D CGS model of a shape or set of shapes that is stored internally as a tree.
* The tree shall consist of nodes which shall be either shapes, transforms or mixes. It should be possible to create a new tree, to modify an existing tree by adding or removing nodes within it and to delete an existing tree.
* It should be possible to create new nodes and to delete existing nodes.
* Shapes shall be either circles, squares or triangles defined by their radius, rectangles defined by their horizontal and vertical radii or closed curves constructed from connected vertices and/or Bezier curve segments. All shapes shall have their default position at the origin of the 2D coordinate system in which they exist. A shape node shall always be a leaf node of the tree.
* Transforms shall be either rotation by some angle in radians, scaling by some real number factor in x and y or translation in x and y by some real number amount. It should also be possible to apply a particular transform to a node multiple times with a single instruction. A transform shall be applied to one existing node.
* Mixes shall be either the + (Union), - (Difference) or & (Intersection) of two or more existing nodes.
* There shall be some user interface that allows interacting with a text representation of the tree and also shows a graphical representation of the tree as a 2D image.
* It should be possible to save the current state as a file and to load it back anytime. Each time a saved file is loaded, it should be possible to resume editing the drawing.

***Preliminary design***

Given the relatively specific nature of the user requirements in terms of internal data structure, design of the program focused mainly on the mechanisms of class interaction rather than overall structure. The primary consideration during the design phase was what sort of information would need to be passed between classes in order to facilitate drawing. It was decided that each node would contain references to its children (if any) and information relating to itself, rather than explicitly recording all information needed to draw the image represented by the node. It was decided that every node would have a draw() method and a print() method that would recursively traverse the tree in postfix order to allow drawing an image to the screen and printing a text representation of the tree respectively.

In addition to the classes that would comprise the image tree, it was decided that some utility classes were also needed. These consisted of LoadFile and SaveFile, which are responsible for loading and saving trees, Parser, which is responsible for parsing user text input and generating a tree from it, as well as checking that input for syntactic correctness, and CubicBezier, which contains methods needed for handling cubic Bezier curves in the ClosedCurve class. These classes provide various functionality to the program, but are not directly involved in the internal representation of the tree or its drawing.

The final major consideration in designing the program was some form of GUI. It was decided that a main window containing a drawing canvas and a text box for user input would be used, along with Save, Load, Draw, Clear and Help buttons. The Help button would produce a secondary window containing explanations of the various possible instructions and the correct syntax to use them. A mock-up of the proposed layout of the GUI is shown below.

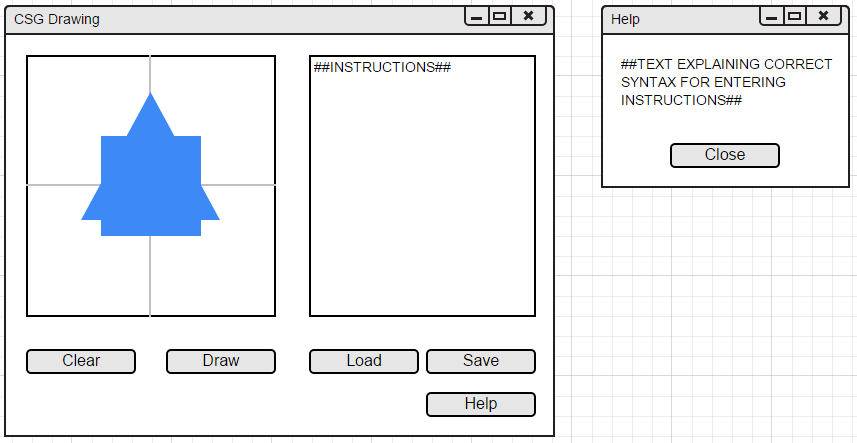
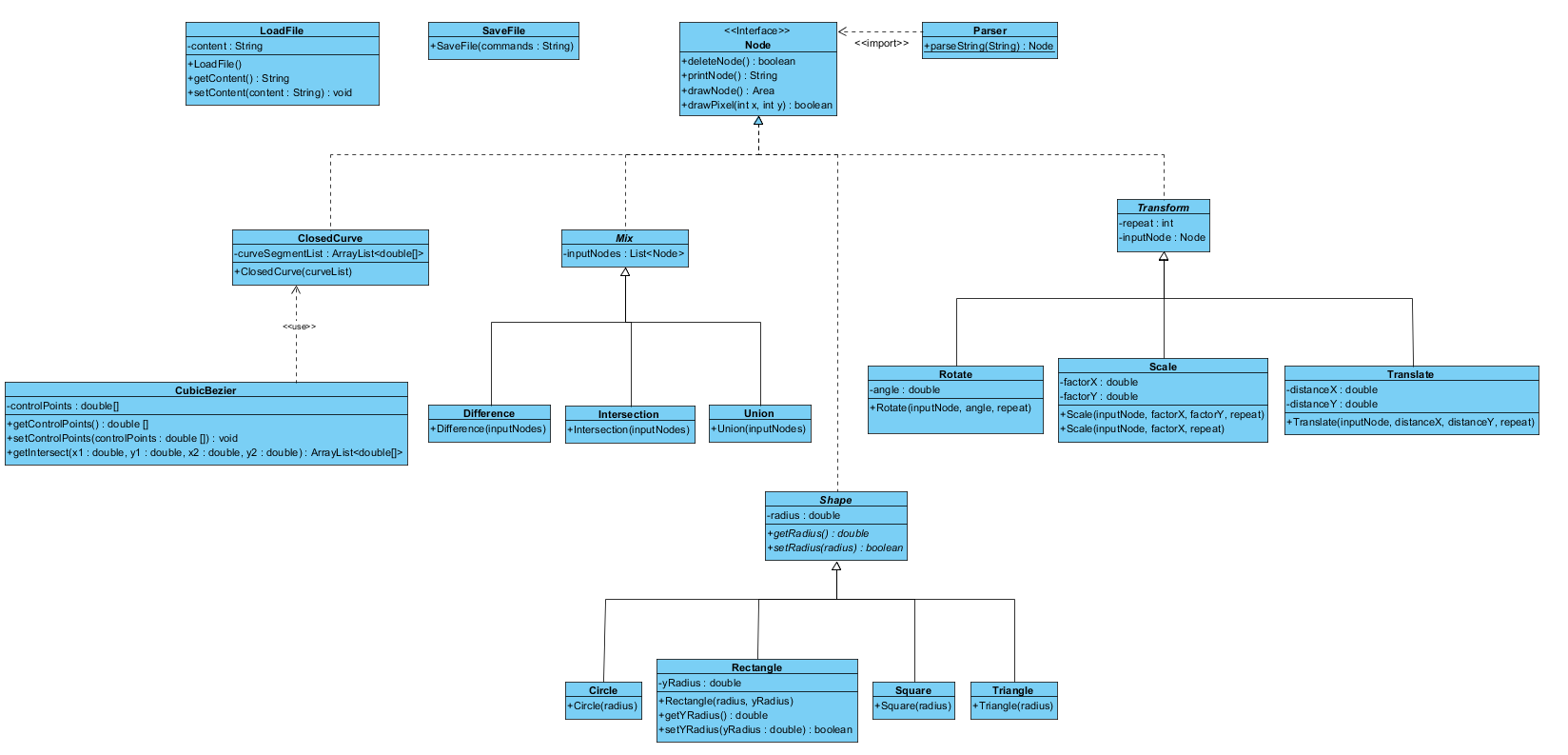


Figure 1: Proposed GUI design

After the completion of the basic class structure design, a more detailed pass was made and the various methods and member variables that would be required were designed. Two draw() methods were designed for nodes; one employing a “is the given pixel inside the shape?” approach and one using built-in classes in Java that greatly simplified the required mathematics and coding. The latter approach was intended to be attempted first in order to create a working prototype and confirm the viability of the overall design, with the former approach being added later if time allowed. The pixel-by-pixel drawing method would employ a ray casting algorithm and winding number algorithm to determine if a given pixel lies within a complex shape such as a closed curve, and use simple geometry to determine if a given pixel lies within regular polygons. The transforms would be handled using matrix multiplication and the mixes using simple Boolean operators (OR for Union, AND for Intersection, AND NOT for Difference) on the argument nodes.

A class diagram detailing the structure of the program and the methods and member variables for each class is shown below.



***Test Sections***

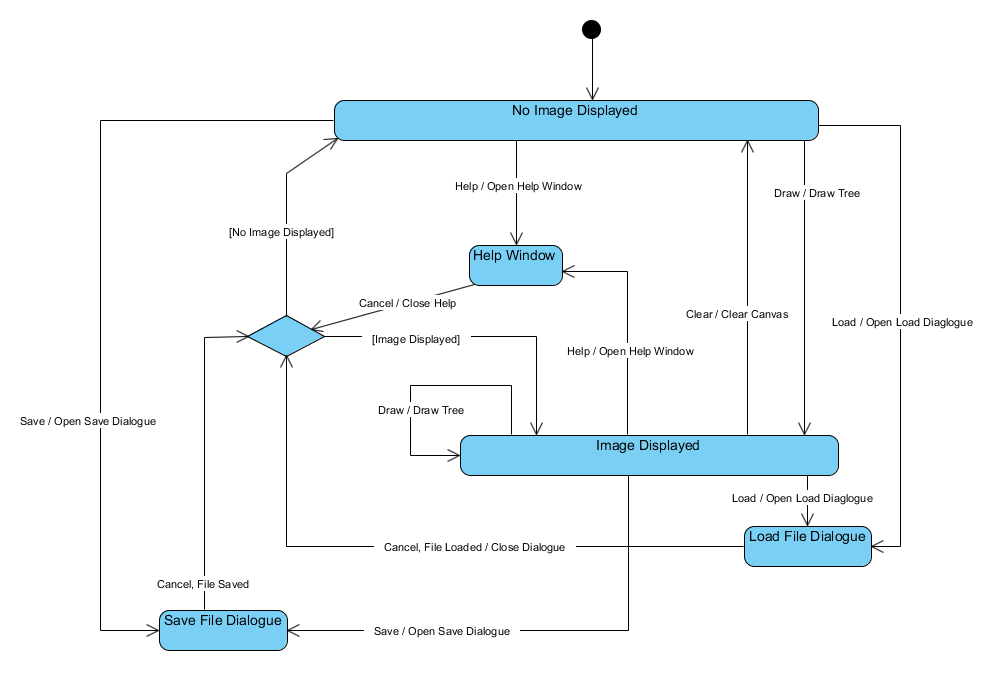
A state diagram representing the GUI, shown below, was created to enable testing of the expected behaviour of the application.

Figure 2: GUI state diagram

From the state diagram it can be seen that the application should start with a blank canvas (no image displayed) and that it should be possible to open the load file dialogue by pressing “load”, the save file dialogue by pressing “save” and the help window by pressing “help” from this state, and to return to it from them. It should also be possible to enter the “image displayed” state by pressing the “Draw” button, providing a valid tree has been entered. Likewise, from the “image displayed” state, it should be possible to open the load file dialogue, the save file dialogue and the help window, and to return to it from them; that is, the displayed image should not be removed when the “load”, “save” or “help” buttons are pressed. It should also be possible to return to the “no image displayed” state by pressing the “clear” button, and to redraw the current image or draw a new image by pressing the “draw” button.

As well as testing the behaviour of the GUI, it is also important to test the operation of the program as a whole by confirming that certain inputs conform to the expected outputs. As there are in theory an infinite number of possible combinations of instructions that can be input by the user of the system, it is not possible to exhaustively test all cases. Furthermore, since the expected output of the system consists of geometric shapes printed to the screen, it is difficult to quantitatively confirm that any given output exactly matches that expected for a given input. However, a qualitative test suite can be created that confirms that the system conforms to the expected behaviour when considering the general nature of the output.

Test1: Square(50) should produce a square at the centre of the drawing area with sides 71 pixels long.

Test2: Circle(50) should produce a circle at the centre of the drawing area with a diameter of 100 pixels.

Test3: Triangle(50) should produce an equilateral triangle at the centre of the drawing area with side length 87 pixels.

Test4: Rotate(Square(50),45) should produce a square with sides 71 pixels long at the centre of the drawing area that has been rotated 45 degrees anti-clockwise.

Test5: Translate(Circle(50),25,25) should produce a circle with a diameter of 100 pixels centred on the point 25 pixels above and 25 pixels to the right of the centre of the drawing area.

Test6: Scale(Triangle(50),2,4) should produce a isosceles triangle with base 173 pixels and height 300 pixels at the centre of the drawing area.

Test7: Union(Triangle(50),Square(30)) should produce an overlaid image of a square with sides 42 pixels long and an equilateral triangle with sides 87 pixels long, both at the centre of the drawing area.

Test8: Intersection(Circle(50),Square(55)) should produce a square with rounded corners, with the diameter of the rounded portion being 100 pixels and the overall height and width of the shape being 85 pixels, positioned at the centre of the drawing area.

Test9: Difference(Circle(50),Square(50)) should produce a circle with diameter 100 pixels with, removed from the centre of it, a square with side length 71 pixels (such that the vertices of the square are on the circumference of the circle), both at the centre of the drawing area.

With this very basic test suite it is possible to confirm that each node type can correctly be displayed, and that every type of shape node can be correctly passed to a transform or mix node as an argument. For invalid inputs (such as syntax errors in instructions or arguments outside the valid input domains), error messages will be displayed to the user. A basic implementation of error reporting will indicate only that an error occurred with the input, but more advanced versions will specify the nature of the error.

***Results section***

***Conclusion***

***Appendix***

*Minutes*

*Code*

*Contribution report*