COMP1100 Assignment 2
Technical Report

Tim James
Lab 14 - Wed (14:00 – 16:00) - Eric Pan
u6947396

1. Introduction

The objective of my Haskell program is to allow for 2D shapes to be constructed through issuing commands which a ‘turtle’ responds to. These shapes are rendered on a plane in a web browser through the implementation of functions from the CodeWorld API, specifically `polyline`, which allows for lines drawn by the turtle to be represented as a `Picture`.

The structure of my program, key response events, and the behaviour a turtle makes to a particular command, have not been altered from the assignment specifications. One notable change, however, is that the function `polygon` has been renamed to `myPolygon` due to a conflict with the CodeWorld function of the same name, and `render` from the `Main.hs` module has been updated the reflect this.

2. Content

2.1 Program Design

When constructing a list of `TurtleCommand`s for the turtle to respond to and build a desired shape, two assumptions were made. First, the turtle would start and end the construction of a shape with the pen up, as to allow movement between drawing shapes. Second, after drawing a complete shape, the turtle should end in the same position and direction facing to where it started.

The `triangle` function was designed to be used within the `sierpinski` function, highlighting the importance of the two assumptions. With the only input being the length of the sides, the number of commands should be static. Thus, we can use the two commands required to move forward by the inputted units, then turn $\frac{2\pi}{3}$ (the external angle of a triangle in radians), three times in a row. We must start by putting the pen down and end by putting the pen up. The final turn ensures the final direction facing is the same as the initial one.

The `myPolygon` function takes a similar approach, except now the number of commands depends on the number of sides inputted. The use of a helper function is employed to track the number of sides still left to construct. Each recursion of the helper function reduces the number of sides left by 1, and adds the commands required to move the turtle forward by the inputted side length, and turn by $\frac{\pi}{180} \times (180 - 180\left(\frac{n-2}{2}\right))$ (the external angle of a polygon in radians, where n is the number of sides).

The `sierpinski` function made use of the `triangle` function for drawing each individual triangle. Fractal levels or layers were constructed one at a time, then combined for the final output. A helper function, `levelBuilder`, was employed to determine the size the base triangle of each level, and then generate the required commands to place the correct number of those triangles in the correct positions, using the `triangleCompounder` helper. `triangleCompounder` allowed for a base list of commands (which at the lowest level is a `triangle`) to be inputted, and to output commands required to execute the base list of commands at every point of a triangle which is not drawn. When used recursively, this gives the repeating triangle pattern we desire for a fractal level.

Finally, the `runTurtle` function responds to a list of `TurtleCommand`s, and builds a `Picture`. This utilizes a helper function which keeps track of the current state of the turtle, stored as a `TurtleState` type which holds data on whether the pen is up or down, the current position and current direction facing. This state is updated depending on each `TurtleCommand` in the list. Whenever the pen was down, and the turtle moved forward, a `polyline` was added to a carry `Picture` input which would be outputted when the end of the list of `TurtleCommand`s had been reached.

2.2 Useability

No additional useability features were added outside of the assignment specifications, aside from guards to ensure negative side lengths or number of sides could not be inputted, and that `myPolygon` could not have less than 3 sides.

2.3 Testing

Due to the limitations of testing the CodeWorld `Picture` type directly, focus was placed on testing certain properties of functions, which was achieved by writing additional functions exclusively used for testing.

For the `triangle`, `myPolygon`, and `sierpinski` functions, several different properties were tested. The total distance drawn from a particular list of `TurtleCommand`s was found with the function `totalDist`, allowing for that property to be tested. We also aimed for these functions to return commands which would leave the turtle in the same position and direction facing as when they initially ran, with the pen up. This was tested with the `finalState` function, which returned what the `TurtleState` would be after a list of `TurtleCommand`s had ran. Finally, I composed my own list of `TurtleCommand`s by drawing on a piece of paper and responding to commands in the same manner a turtle would. For each of the three functions, it was tested that from given inputs, a verified list of `TurtleCommand`s would be outputted.

Note that the functions `totalDist` and `finalState` (and the associated `finalIsPenDown`, `finalPos` and `finalFacing`) were used only for testing. These functions were themselves tested against lists of `TurtleCommand`s verified by calculations done by hand.

With these three different properties being tested, we can be confident that the intended effect is being achieved. The `runTurtle` function, however, could not be tested directly. The `finalState` and `totalDist` functions employ a similar structure to the `runTurtle` function, and I have tested these, however this only gives a small degree of confidence in `runTurtle`. Instead, I ran the program and manipulated the `render` function from the `Main.hs` module, and observed the picture drawn on the screen. For example, the `comp1100` list of `TurtleCommand`s was inputted into `runTurtle`, and the picture drawn on the 2D plane appears the same as that from the assignment specifications.

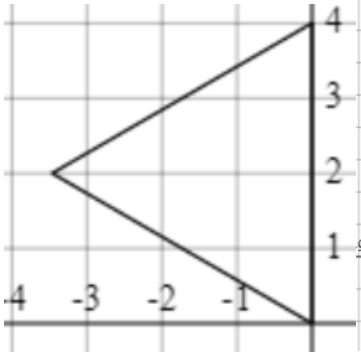

Figure 1: `triangle 4.0`

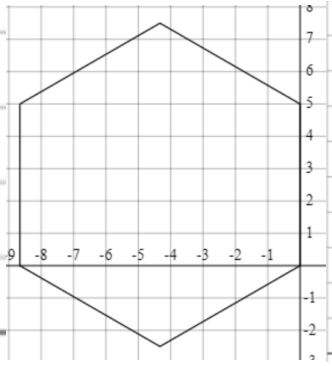

Figure 2: `myPolygon 6 5.0`

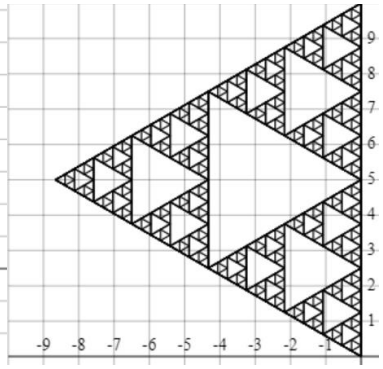

Figure 3: `sierpinski 6 10.0`

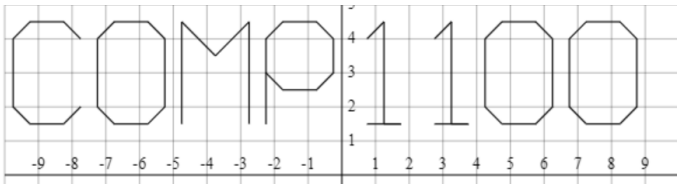

Figure 4: `comp1100`

3. Reflection

3.1 Challenges

Difficulties faced were largely mathematical, such as finding the correct angle to turn when constructing a polygon. The `sierpinski` function proved challenging, but after working through the problem on a piece of paper and observing the pattern, it was not too difficult to translate it into a function given an understanding of recursion.

3.2 Improvements

While my code fulfils the purpose given by the assignment specifications, more properties of functions could have been tested (such as checking `TurtleState` after a given number of steps, not just the final state), especially in regard to `runTurtle`. More key press events could have also been added, such as keys to change the number of fractal levels, which would allow for a more dynamic visualization of the `sierpinski` function.

Word Count: 1090 (within 10% of limit)