My Turtle Game

Virag Varga

Table of Contents

Design

Out of the 3 components of the assignment, I found the Domain Specific Language development and decision making on drawing library the hardest.

Throughout the whole assignment, I continuously improved my DLS to suit the new challenges of each part. The successfully completed deliverables were:

- support for the basic functionality for manipulating and managing the turtle state, such as “forward”, “backward”, “left”, “right”, “penUp”, “penDown”, “setColor” and “home”
- allowing sequencing commands
- utilizing monad transformers

The second part required the most dedication and attention to details. During the development of this section, I faced with the faults of my DLS and it gave me a good chance to improve that as well. The assignment for this phase is able to:

- draw on a canvas step-by-step with visual feedback (all using SVG), in any color (using RGBA)
- load and run for long period of time
- take input from control interface and utilize the information

The L-System and Fractal generation part was simpler to implement, especially with the help of the provided introductory website[1]. The provided L-System function proved to be a challenge as I found its instructions unclear. The assignment is able to:

- implement string rewriting and execute commands according to it
- use State monad for position/heading stack implementation
- use the existing DLS to implement Turtle state management and manipulation
- use user defined parameters

DLS Architecture

At the very beginning of this assignment, I collected the requirements for the DLS and how I could meet these requirements. One component was to use StateT monad transformer for the turtle state handling. For error handling, I considered Either monad, however it introduced unnecessary complexities to the program. I realized that this can be avoided with Maybe monad, while keeping the ability to pass on state update information, even at function failures.

For turtle state data type, I needed to create 3 new data types called *‘Point’* (Point Double Double), *‘Pen’* (Up | Down), *‘Color’* (r :: Int, g :: Int, b :: Int, a :: Float).

[1] http://opensask.ca/Python/Strings/LSystems.html

The data type *'Turtle'* (Turtle {pos :: Point, dir :: Point, pen :: Pen, color :: Color}) is not a monad, but a state of the turtle used in StateT Turtle Maybe. It is updated by functions like 'runTurtle (forward n) turtleState' and returned with a conformation or error message depending on the success of the function.

Functions in Turtle DLS	Description
initTurtle	Creates an initial Turtle with position at (0,0), facing direction to (0,1), pen set down and color set to black
getPosition	Takes a *'Turtle'* type variable and returns its position (Point)
setPosition	Takes a *'Turtle'* type variable, a *'Point'* type variable and returns a new *'Turtle'*
getDirection	Takes a *'Turtle'* type variable and returns its facing direction (Point)
getPen	Takes a *'Turtle'* type variable and returns its pen state (Pen = Up | Down)
isPenUp / isPenDown	Returns StateT Turtle Maybe Bool
isPenUpBool / isPenDownBool	Takes a *'Turtle'* type variable and returns True/False depending on the pen state
createColor	Take in 3 integers (r, g, b) between 0 and 255, a float between 0 and 1, and returns a Color corresponding to the RGBA (red, green, blue, alpha) color code
black/red/green/blue/yellow	Returns *'Color'* type variable corresponding to the name
setColor	Takes a *'Turtle'* type variable, a *'Color'* type variable and returns a new *'Turtle'*
getColor	Takes a *'Turtle'* type variable and returns its position (Point)
str2Color	Takes a String name of a colour and returns the corresponding Color
applyInt	Takes a *'Point'* variable, a function and an Int and applies the Int to the coordinates of the Point according to the function, returning the new Point
applyPoint	Takes two *'Point'* variables, and a function and applies the function to the x coordinates and y coordinates, returning a new Point (function x_1 x_2, function y_1 y_2
unitDir	Takes a *'Point'* variable (direction) and turns it into a unit vector, which is returned as a Point
rotateDir	Takes a *'Turtle'* variable and a Double (degrees), and returns a Turtle with

	changed direction according to the Double
changePos	Takes a *'Turtle'* and an Int, and returns a Turtle that's position is changed by Int amount towards its facing direction
errorProtocol	Takes a String and *'Turtle'* variable and returns aStateT Turtle Maybe ErrorMessage
Plus the required movement functions	

I defined the getter and setter functions, for the attributes of Turtle whenever it was required for the program.

GUI

When I started the 2nd part of the assignment, I discovered an example GitHub repository[2] for Threepenny which showcased Threepenny's Canvas library, similarly to what I was aiming to do. However, I struggled to create and later change the position of a primitive turtle on the canvas. I researched for other libraries that would help to create an image for the turtle, however they were not compatible with my already in-use libraries. After implementing the turtle with the help Threepenny's SVG, I decided to completely convert to SVG library for simplicity and uniform implementation.

The drawing interface is entirely written in Threepenny's SVG. The canvas element contains every line as a child along with a primitive turtle, in the form of a triangle. To clear the canvas, the list of children is overwritten by a new instance of the primitive turtle. The lines are drawn according to turtle position by a function, added to canvas and the primitive turtle is updated. If the position or the drawing colour of the turtle is changed, the primitive turtle on the canvas is updated. This is done by deleting the primitive turtle and appending a new one at the right position/with the right colour as a child to the canvas [See colour change of the turtle from black (left) to red (right)].

The state of the turtle is followed by a "global" IORef variable that is initiated and accessed in the function "setup", that initializes and runs the widow of my web application. That "global" state is used as an initial state to run the state modifying functions of Turtle on.

One important problem that arose during the development of the drawing interface, was the flipped coordinate system. During the development of the Turtle DLS and its functions, I interpreted the coordinates as x-axis pointing from left to right and y-axis pointing from bottom to the top. However, Threepenny's coordinate system has a flipped y-axis, which meant that the up direction for me, was down for Threepenny.

[2] https://github.com/HeinrichApfelmus/threepenny-gui/tree/master/lib/threepenny-gui/samples

L-System Implementation

The L-System and fractal implementation had some difficulties of its own as well. The two main problems were

1) to define “lSystem” function, in a way that aligns with my DLS
2) to configure the stack system

“lSystem” uses a separate function called “applyRules”, which does not count the iterations, it only applies the rules to the current “String of instructions” using a function called “applyRules2Char”. The reason for the 2 similar functions is that the first function always needed to have access to all the rules without any recursive action through that given list. The last version of the instructions is received by using “safeLast $ take (n+1) $ iterate …”, then interpreted into functions that is applied to the initially given turtle.

For the function “lSystem”, I altered the return value into a list of turtle states, this way the implementation into a new module is easier. In my canvas drawing module, the list of turtle states was run through its drawing function entry by entry, drawing the different motives. I also changed the “Rules” data type into “Fractal” (Fractal {rules :: Rules, degrees :: Double, step :: Int}), which would also contain the default degrees for turning and default amount of movement on top of having the list of rules.

The setup of a stack was slightly harder, as I had some difficulty visualizing how it would work. I ended up utilizing the State monad for managing the Stack. Every entry of stack, if initialized, is a point value pair for capturing a given position and facing direction (vector). The entries of the stack can be manipulated by a “pop” function, to return the top entry, and a “push” function to store an entry at the top of the stack.

Self-Assessment and Reflection

This assignment certainly had some surprises for me. I expected haskell to be more difficult to write programs in. I also didn’t expect to enjoy writing the assignment and facing its challenge.

Part 1 – Turtle.hs, Draw_1.hs

On this part, most of the work was understanding haskell and the underlying functionalities of monads and monad transformers. I found the topic challenging, however, I’m confident at my implementation of StateT monad transformer. I feel proud of my Turtle module, as it works according to the requirements, it is neatly written. I can see some room for improvement, such as clarifying the places to use Position (Point) and Direction (Point), or expanding the amount of setter and getter functions.

Part 2 – Draw_2.hs, Turtle.hs

Studying had to be done in this part as well, as I have not worked with Threepenny much before. Luckily the sample codes of the Threepenny GUI GitHub repo[3] was proved to be very useful and whatever wasn't introduced there I could easily learn it from the documentation[4] and get familiar with using the library. I did have some difficulties with positioning elements on the webpage, which I still think could be improved within some paragraphs of texts. I also had a lot of functionality to write on my Draw_2 module which made it slightly complicated and hard to follow. I am proud of the order of functions, elements and other components within the setup function, because, while it's a tiny piece, it has an incredibly delicate balance. I believe the functionality was implemented in a very scalable manner and with attention to easy expandability.

Part 3 – Fractals.hs, Draw_2.hs

Writing the Turtle module gave me a good base of knowledge for module writing and prepared me for creating the Fractals module. I find the functions in this module easily readable and logical. I am proud with how it turned out, even if the function "lSystem" could be improved to match the required format more.

Haskell Experience

I understand haskell more now, than when I started. I see how it pattern matches every line of a function. I also understand the types of embedded DLS and how to create both shallow and deep. I enjoyed the different approach and way of thinking haskell required, like a math problem that I have the answer to, I just need to understand how to get to it. I could certainly create other similar projects, however throughout the this project I could not shake the mindset of object-oriented programming. For that reason, I would say haskell is not build for projects, where global variables or changing values are the focus.

[3] https://github.com/HeinrichApfelmus/threepenny-gui/tree/master/lib/threepenny-gui/samples
[4] https://hackage.haskell.org/package/threepenny-gui