National University of Singapore
School of Computing
CS1101S: Programming Methodology (JavaScript)
Semester I, 2012/2013

Mission 4
Curve Introduction

Start date: 27 August 2012
Due: 01 September 2012, 23:59

Readings:

- Textbook Sections 1.1.5 to 1.1.8

Coming out on top of the JFDI contest, you have thoroughly impressed and convinced the grandmaster to impart you the ways of JFDI.

He has decided to teach all chosen ones a fundamental skill of the JFDI, using the Force to manipulate curves. Such training would ensure that you can master the Force for other important missions.

The grandmaster waves to the disciples as he ascends the stage. He introduces himself as Grandmaster Martin and begins to give a preparatory speech for the disciples.

"Welcome my disciples. I am impressed by your intelligence as I watched you solve the challenging problems of the rune doors and mosaic. However these are just mere basics of the way of the Force. To be a proper disciple of the JFDI, first you must learn how to master the techniques of the Force. It will not be an easy feat. But I will start from scratch and teach you the basics of the Force before you go on to mastering the technique.

I will introduce to you some mind training. You will begin to think of nothing first. Then slowly, start picturing a curve being drawn from one end to another. You will feel your mind strengthening as it codes out the shape of the curve..."

Drawing Curves

A *curve* is a basic drawing element. Complex pictures can be obtained by drawing multiple curves. A curve is a unary function which takes one real argument within the unit interval $[0, 1]$ and returns a point (a pair of reals). Intuitively, the real argument can be thought of as the time, and the point returned by the curve can be thought of as the position of the pen at the time indicated by the argument. In JavaScript, we let Unit-Interval be the type of JS-Nums between 0 and 1, and we represent curves by functions of JavaScript type Curve, where

$$\text{Curve} : \text{Unit-Interval} \rightarrow \text{Point}$$

and Point is some representation of pairs of JS-Num's. If C is a curve, then the starting point of the curve is always $C(0)$, and the ending point is always $C(1)$.

To work with Point, we need a *constructor*, `make_point`, which constructs Points from JS-Nums, and *selectors*, `x_of` and `y_of`, for getting the x and y coordinates of a Point. We require only that the constructors and selectors obey the rules

$$\begin{aligned} \texttt{x\_of}\,(\texttt{make\_point}\,(n, m)) &= n \\ \texttt{y\_of}\,(\texttt{make\_point}\,(n, m)) &= m \end{aligned}$$

for all JS-Num's m, n. Here is one way to do this:

```
function make_point(x, y){ return pair(x, y); }

function x_of(point){ return head(point); }

function y_of(point){ return tail(point); }
```

$$\begin{aligned} \texttt{make\_point} &: (\text{JS-Num}, \text{JS-Num}) \rightarrow \text{Point}, \\ \texttt{x\_of}, \texttt{y\_of} &: (\text{Point}) \rightarrow \text{JS-Num}. \end{aligned}$$

For example, we can define the Curve `unit_circle` and the Curve `unit_line` (along the x-axis):

```
function unit_circle(t){
    return make_point(Math.sin(2 * Math.PI * t),
                      Math.cos(2 * Math.PI * t));
}

function unit_line_at(y){
    return function(t){
               return make_point(t, y);
           }
}

var unit_line = unit_line_at(0);
```

Drawing Functions

In the previous section we introduced the concept of curve as the basic drawing unit. However, in order to actually draw a curve on the window, you will need a drawing function. It is not required that you understand the implementation of the drawing functions, but you should know how to use them in order to visualize and test your solution.

When you launch any of the HTML files provided for this mission in a browser, they will load **`hi_graph.js`**. An empty viewport should be displayed.

In the Web Console type

```
draw_connected(200)(unit_circle);
draw_connected_squeezed_to_window(200)(unit_circle);
```

Note that after you have typed each line you will get a response of ”done”. After the first draw statement, you will see a quarter circle while after the second, you will see a full circle. Note that all Curve functions accepted by the draw methods are of the form defined above:

$$\text{Curve} : \text{Unit-Interval} \rightarrow \text{Point}$$

The 200 in `draw_connected` refers to the number of points to draw in this screen. Since 1/200 = 0.005, unit-circle will be called for values of t = 0, 0.005, 0.01, 0.015, 0.02, etc. until t = 1. `draw_connected` will then join two adjacent points returned by `unit_circle` to draw a connected Curve. If you want to draw the points without connecting them, use `draw_points_on`. The following is an example on using `draw_points_on`:

```
draw_points_on(200)(unit_circle);
```

Note that the origin of the drawing window is at the bottom left. Moving right along the drawing window increases the value of the x-axis until the x-coordinate equals 1. Likewise moving up along the window increases the value of the y-axis until the y-coordinate equals 1. This is why `draw_connected` shows only a quadrant of the `unit_circle`; since for some values of t, the x and y-coordinates are outside the range [0, 1]. For example, try:

```
unit_circle(0.5);
```

The y-coordinate of -1.0 is outside the range [0, 1] and hence cannot be displayed. Another function, `draw_connected_squeezed_to_window`, takes care of this by scaling and translating the Curve as required so that all points fall in the range [0, 1] for both axes.

More on Drawing Curves

Now that we have learnt the basics on how to draw the curve on the window, Grandmaster Martin would like us to get even more familiarised with the drawing of curves. Apart from the drawing functions that you have been shown above, there are others you can use to draw curves. The following is a list of the drawing functions available for use:

1. `draw_points_on`,
2. `draw_connected`,
3. `draw_points_squeezed_to_window`,
4. `draw_connected_squeezed_to_window` and
5. `draw_connected_full_view`

The differences between these functions are suggested by their names. The way to call these functions are the same. For example, to draw the pre-defined curve *unit_circle* on the window using 200 points, you may call

    draw_points_on(200)(unit_circle);

In order to draw a connected curve, you may use

```
draw_connected(200)(unit_circle);
```

If you want to make the curve fit in the current window, you may use

```
draw_connected_squeezed_to_window(200)(unit_circle);
```

Note that the more points you use, the more accurately the curve will be drawn.

This mission has **two** tasks.

Task 1:

Recall the definition of `unit_line_at`:

```
function unit_line_at(y){
    return function(t){
               return make_point(t, y);
           }
}
```

(a) What is the type of `unit_line_at`?
Hint: The format for expressing a type and an example is shown below:

$$\begin{array}{rcl} \texttt{<proc>} & : & (\text{<para-type>}[, \ldots]) \rightarrow \text{<output-type>} \\ \texttt{make\_point} & : & (\text{JS-Num}, \text{JS-Num}) \rightarrow \text{Point} \end{array}$$

(b) Define a function `vertical_line` with two arguments, a point and a length, that returns a vertical line of that length beginning at the point. Note that the line should be drawn upwards (i.e., towards the positive-y direction) from the point.

(c) What is the type of `vertical_line`?

(d) Using `draw_connected` and your function `vertical_line` with suitable arguments, draw a vertical line in the window which is centered and has half the length of the sides of the window.

Task Files

- lib/list.js
- lib/misc.js
- lib/graphics.js
- lib/hi_graph.js
- mission_4_1.html
- **mission_4_1.js**

Task 2:

Apply `draw_connected(200)` to `unit_circle`, i.e.

```
draw_connected(200)(unit_circle);
```

Then apply `draw_connected(200)` to `alternative_unit_circle`. Can you see a difference? Now try using `draw_points_on` instead of `draw_connected`.

Also try `draw_points_squeezed_to_window`.

Use appropriate drawing functions to draw unit_circle and alternative_unit_circle. Write down the difference between unit_circle and alternative_unit_circle. You should also point out why this difference exists by examining the code of both unit_circle and alternative_unit_circle in **`hi_graph.js`**.

Task Files

- lib/list.js
- lib/misc.js
- lib/graphics.js
- lib/hi_graph.js
- mission_4_2.html
- **mission_4_2.js**

Submission

To submit your work to the Academy, copy the contents from the template file(s) into the box that says ”Your submission” on the mission page, click ”Save Code”, then click ”Finalize Submission”. Note that submission is final and that any mistakes in submission requires extra effort from a tutor or the lecturer himself to fix.