Mission 7: Diagnostics and Dragonize

Start date: 07 September 2017
Due: 13 September 2017, 23:59

Readings:

- Textbook Sections 1.3.3

This mission has **two** tasks.

Task 1:

One of your fellow Cadets, Pixel, isn't entirely happy with the style of several of the definitions found in this training. In particular, she feels the program goes overboard in inventing names for values that are used infrequently, and this lengthens the program and burdens someone reading the program with remembering the invented names. For example, she thinks the definition

```
function rotate_around_origin(theta) {
    var cth = math_cos(theta);
    var sth = math_sin(theta);
    return function (curve) {
               return function(t) {
                          // Pixel eliminates the declaration of ct
                          var ct = curve(t);
                          var x = x_of(ct);
                          var y = y_of(ct);
                          return make_point((cth * x) - (sth * y),
                                            (sth * x) + (cth * y));
               };
           };
}
```

would be a bit more readable if the name `ct` for the value of `curve(t)` was dropped. She proposes instead:

```
function pixel_rotate(theta) { // rotates around origin, but less efficiently
    var cth = math_cos(theta);
    var sth = math_sin(theta);
    return function (curve) {
               return function(t) {
                          var x = x_of(curve(t)); // Pixel writes curve(t)
                          var y = y_of(curve(t)); // twice
                          return make_point((cth * x) - (sth * y),
                                            (sth * x) + (cth * y));
```

```
                   };
           };
}
```

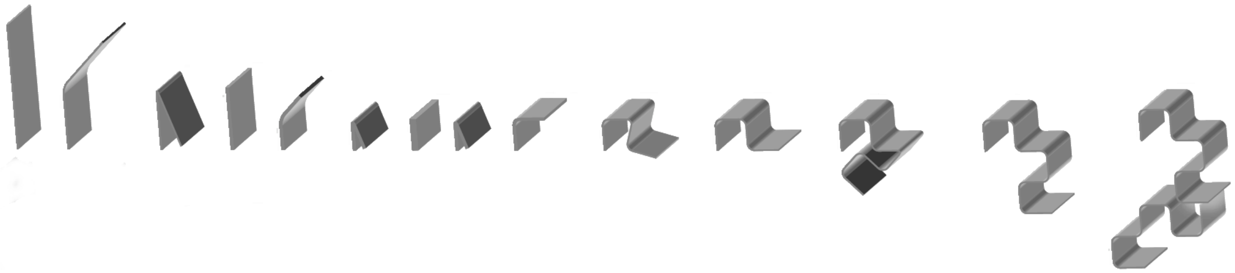
Grandmaster Martin warns Pixel that the var declarations she uses in `pixel_rotate` are actually computationally expensive compared to just using the abbreviation `ct`.

1. Does `pixel_rotate` work and achieve the same purpose as `rotate_around_origin`?
2. Briefly explain why using `pixel_rotate` as a subfunction in place of the original `rotate_around_origin` in the definition of `gosper_curve` will turn a process whose time is linear in the level into one which is exponential in the level.

Task 2:

We can now invent other functions like the Gosper process, and use them to generate fractal curves.

A model of a Heighway dragon curve can be made by repeatedly folding a strip of paper always to the same side, and then unfolding it so that all angles are at 90 degrees, as shown in the figure below.



The above figure presents a way to obtain a dragon curve of order 4. A curve of order of magnitude 200 is presented in Figure 2.

Pixel is trying to write a function that would replicate this process in order to draw the dragon curve. She says “nothing easier, all I need to do to produce a curve of order `n` is connecting the curve of order `n-1` and its rotation by 90 degrees at their ends, and then put the resulting curve in standard position.” So, Pixel comes up with the following function:

```
function pixel_dragonize(n, curve) {
    if(n === 0) {
        return curve;
    } else {
        var c = pixel_dragonize(n - 1, curve);
        return put_in_standard_position(connect_ends
                   ((rotate_around_origin(-math_PI / 2))(c), c));
    }
}
```

However, the function doesn't quite work. Grandmaster Martin suggests that the following function should also be employed in the program:

```
function invert(curve) {
    return function(t) {
               return curve(1 - t);
           };
}
```

This function "inverts" the curve, in the sense that it traverses the curve in the reverse direction (the same curve would appear on the screen). To understand the difference, draw in separate windows the following two curves.

```
connect_ends(unit_line, unit_circle);
```

and

```
connect_ends(invert(unit_line), unit_circle);
```

(Note: use `draw_connected_full_view` to make the curves fit in the windows. It is used in the same way as `draw_connected_squeezed_to_window`. Look at the top part of the figure to see the difference.)

Help Pixel solve the problem.

Write a function `dragonize` that produces Heighway's Dragon Curve. The function should be similar to Pixel's, and it should also make use of the `invert` function. Test your program and make sure you can reproduce the drawing in Figure 2 with:

```
(draw_connected_squeezed_to_window(1000))(dragonize(200, unit_line));
```

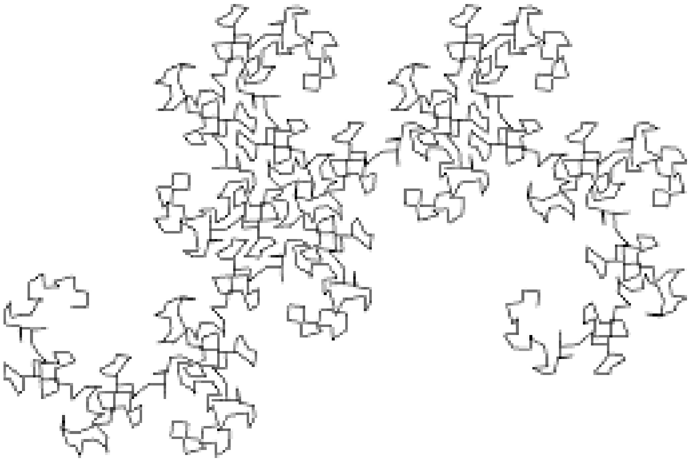


Figure 2: Sample Drawing for Task 2.

Submission

To submit your work for this mission, copy the url on your browser and email it to your respective Avengers. Strictly follow to the deadlines set at the start of this file.

IMPORTANT: Make sure that everything for your programs to work is on the left hand side and **not** in the interpreter pane on the right! This is because only that program is preserved in the url you have emailed to us.