Lab 10 - Squares And More (SAM)

Gerardo Morales, Will Confoy

0.1 Introduction

Art is hard. How hard varies from person to person but for those who are not artistically inclined, it can be really difficult to get over the initial hurdle of making what's in your head appear on canvas. Squares And More (SAM) helps to remove this barrier, by providing a rigorous language for creating regular polygons of nearly any configuration. Additionally, SAM supports a syntax that is easy to understand for anyone fluent in English, further increasing its usefulness.

SAM's usage especially lessens the burden on anyone who is disabled or has difficulty with fine motor skills. So long as you can put words on a screen, you can use SAM to produce the exact image you want, so long as that image is made up of regular polygons. This coupled with the fairly natural language approach of SAM means that SAM works well with the many diverse methods by which people have found to interface with computers.

0.2 Design Principles

While designing SAM, we kept in mind two main principles: ease of use and abstraction. We wanted to ensure that SAM was easily understandable to English speakers so that it would be easy for people who hadn't seen it before to hop in and begin writing programs, and being able to put the image in their brain on canvas. Of course, there are some specifics that require user input, as it is a rigorous language. However, we did our best to abstract as much of the definitions away from the sight of the user, using the minimum definitions of shapes possible to deterministically define shapes. For example, you really only need a point, a distance, a number of sides, and an orientation to fully define any regular n-gon.

0.3 Examples

Here are a few sample programs:

```
[<canvas(600,600)>]


4 4-gon with radius 120 and center(300,390), step[85,85,85],
rotation[45.0], color[#b4bcb1,#f7d85b,#fdd30b,#f0c55f],
centerDelta [(0,-30),(0,-30),(0,-30)], normal(0.0,0.0) and granularity 5
```

This has been extended to multiple lines to fit on the page, but in actuality the way it would normally look is it would have the canvas part on one line, then a newline and then the ngon line on one line. You could, however, put it all on one line. The output is below.

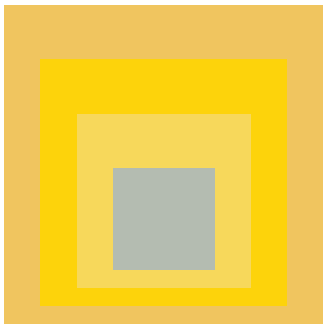

Figure 1: Homage to the Square

And another program:

```
[<canvas(400,400)>]

4 4-gon with radius 40 and center (200,200), step[5,4,4],
rotation[0.0,45.0], color[#f9c406,#FF0000,#00FF00,#0000FF],
centerDelta [(75,-50),(5,-5)], normal (0.0,0.0) and granularity 5
```

As before, this has been changed to fit on the page. The output is below:

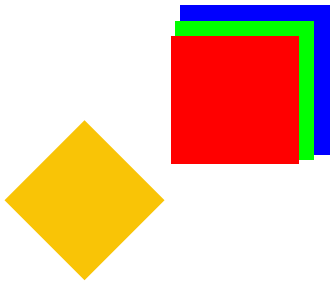

Figure 2: Rotation List Test Figure

and finally, a more complex program:

```
[<canvas (700,700)>]


8 60-gon with radius 60 and center (350,350), step [10], rotation [0.0],
color [#f9c406,#FF0000,#995668,#f9c406,#f9c406,#FF0000,#995668,#f9c406],
centerDelta[(-25,-25)], normal (0.0,10.0) and granularity 50
8 60-gon with radius 60 and center (350,350), step [10], rotation [0.0],
color [#f9c406,#FF0000,#995668,#f9c406,#f9c406,#FF0000,#995668,#f9c406],
centerDelta[(25,25)], normal (0.0,10.0) and granularity 50
8 60-gon with radius 60 and center (350,350), step [10], rotation [0.0],
color [#f9c406,#FF0000,#995668,#f9c406,#f9c406,#FF0000,#995668,#f9c406],
centerDelta[(25,-25)], normal (0.0,10.0) and granularity 50
8 60-gon with radius 60 and center (350,350), step [10], rotation [0.0],
color [#f9c406,#FF0000,#995668,#f9c406,#f9c406,#FF0000,#995668,#f9c406],
centerDelta[(-25,25)], normal (0.0,10.0) and granularity 50
8 60-gon with radius 60 and center (350,350), step [10], rotation [0.0],
color [#f9c406,#FF0000,#995668,#f9c406,#f9c406,#FF0000,#995668,#f9c406],
centerDelta[(0,25)], normal (0.0,10.0) and granularity 50
8 60-gon with radius 60 and center (350,350), step [10], rotation [0.0],
color [#f9c406,#FF0000,#995668,#f9c406,#f9c406,#FF0000,#995668,#f9c406],
centerDelta[(0,-25)], normal (0.0,10.0) and granularity 50
8 60-gon with radius 60 and center (350,350), step [10], rotation [0.0],
color [#f9c406,#FF0000,#995668,#f9c406,#f9c406,#FF0000,#995668,#f9c406],
centerDelta[(25,0)], normal (0.0,10.0) and granularity 50
8 60-gon with radius 60 and center (350,350), step [10], rotation [0.0],
color [#f9c406,#FF0000,#995668,#f9c406,#f9c406,#FF0000,#995668,#f9c406],
centerDelta[(-25,0)], normal (0.0,10.0) and granularity 50
1 60-gon with radius 60 and center (350,350), step [], rotation [0.0],
color [#0055BB], centerDelta [(0,0)], normal (0.0,0.0) and granularity 50
```

Once again, this has been edited to fit on the page. The output is below:

Figure 3: Many 60-gons

0.4 Language Concepts

The core ideas can essentially be boiled down into what a regular n-gon is, and different attributes that they may have, such as orientation, size, and color. For some of the specific syntax, it might be useful to understand what an array is, as we use them to specify values for different polygons. Since we are only exposing creation of 1 type of shape, users don't need to worry about many different primitives or combined forms. Underneath the hood, there will be plenty of points, lines, and whatnot, but that's all abstracted away. For example, a polygon is a composed of a point, a radius, a color, and a rotation (in degrees). We ask the user for all of the data, but it's in English and we handle putting it all together.

0.5 Syntax

$$
\begin{aligned}
< expr > &:= < newline >^* \ [< canvas(< n >, < n >) >] < newline^* >< ngon >^+ \\
< ngon > &:= < n > \sqcup < n > -\text{gon\_with\_radius } < n > \text{ and\_center}(< n >, < n >), \\
&\quad \text{step } < step >, \text{ rotation } < rotation >, \text{ color } < color >, \text{ centerDelta } < centerDelta >, \\
&\quad < dist > \text{ and\_granularity } < n >< newline >^+ \\
< n > &:= < digit >^+ \\
< negn > &:= - < n > \\
< int > &:= < n > \mid < negn > \\
< digit > &:= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \\
< f > &:= < int > . < n > \\
< step > &:= [< steps >^*] \\
< steps > &:= < n >, < n > \\
&\quad \mid < n > \\
< rotation > &:= [< rotations >^*] \\
< rotations > &:= < f >, < f > \\
&\quad \mid < f > \\
< color > &:= [< hex >^+] \\
< hex > &:= \# < hdigit >^+ \\
< hdigit > &:= 0 \mid 1 \mid 2 \mid \ldots \mid F \\
< centerDelta > &:= [< delta >^*] \\
< delta > &:= (< n >, < n >), (< n >, < n >) \\
&\quad \mid (< n >, < n >) \\
< dist > &:= < name > (< f >, < f >) \\
< name > &:= \text{“normal”} \mid \text{“uniform”} \mid \text{“laplace”} \mid \text{“fish”} \\
< newline > &:= \text{‘}\backslash n\text{’}
\end{aligned}
$$

0.6 Semantics

$< n >$	A positive integer ranging from 0 to the greatest integer F# can represent
$< negn >$	A negative integer ranging from the smallest integer F# can represent to 0
$< f >$	A floating point value that can express the same values as a float in F#
$< canvas >$	Canvas is a tuple of positive integers that specifies the dimensions of the SVG canvas
$< n > n-$gon	The second n specifies the number of sides of each n-gon and the first specifies how many to draw
$< radius >$	The size of the radius of the first n-gon drawn
$< center >$	The x and y coordinates of the center point of the first n-gon drawn
$< step >$	The change in the size of the radius of each n-gon
$< roation >$	The amount each n-gon is rotated about its center point in degrees
$< color >$	The hex values corresponding to the color of each n-gon
$< centerDelta >$	A list of int tuples what indicates how much the change the x and y coordinate of the next n-gon

0.7 Remaining Work

There is little remaining that we intend to implement. Really just fixing some bugs and maybe finding ways to make the language relax some of its parsing with regard to newlines or with regard to ints and floats. For example, for rotation, if you want to rotate 45 degrees, that needs to be 45.0 degrees currently. It would be nice to just be able to write 45 and have the language understand that. Beyond that, there are several things that we could do, and we're just not sure if we want to tackle them for our final version or maybe leave them as future work beyond the class. Among

them are implementing variables and functions, as well as messing around with optional fields with default values so that there's less typing for each set of ngons. We've already done some of the major things we wanted, like extending the lists so that you don't need to type out 60 things into a list for 60 ngons. We just take the last element and make it the rest of the list.