FOGL (Figure Oriented Graphics Language)
Julian Rosenblum, Evgenia Nitishinskaya, Richard Zou

0 - Introduction

FOGL is a language for the sole purpose of creating static and interactive graphics. As the name suggests, programs are primarily made up of specialized data structures called figures. Figures are similar to classes except they are more tailored towards graphics.

1 - Figures

A figure is a class-like definition of an object that can be drawn from its components given parameters. Like classes, figures can have multiple instances. Unlike classes, all properties are public and there is no inheritance. Figures can contain any number of any of the following within it:

1.1 - Parameters

```
param rad,
loc = <0,0>,
col = #ff0
```

Parameters are the given arguments that are automatically assigned to properties of the same identifier. If a parameter is given a default value, then it is not required. Otherwise, it is. When an instance of a figure is defined, its parameters are given in the following form:

```
PacMan(rad=4, loc=<6,6>)
```

1.2 - Components

```
comp body = Arc(loc=loc, radius=rad, col=col),
mouth = Arc(loc=loc, radius=rad, start=PI/4, end=7*PI/4,
col=background, fill=true),
eye = Arc(loc=loc+<rad/2,-rad/2>, radius=3, col=#000)
```

Component are properties that are drawn whenever the figure is drawn. A component must be a figure (built-in or user-defined) or a list of figures.

1.3 - Properties

```
var mouthOpenness = 1,
isAlive = true
```

A property is a variable that is defined in a figure whose value is unique for each instance of the figure. Parameters and components are both types of properties, but the term "property" generally refers to a property that is neither of the two. Properties are public.

1.4 - Methods

```
def doSomething(arg1, arg2) {
    ; statements
}
```

Methods are routines that are defined for a figure but manipulate properties for whichever instance called it. Arguments in method and function calls are referenced by numerical order, not by name.

1.5 - Recursive Figures

If a figure is declared as recursive using the `rec` keyword, it can contain components of itself. Instances of recursive figures must be declared with a maximum recursion depth. For example, consider the Sierpinski triangle (assume the figure `Equilateral` has already been defined with loc referring to the top vertex):

```
figure rec Sierpinski {
      param loc = <0,0>, sideLength
      comp triangle = Equilateral(loc=loc, length=sideLength,
fill=false),
      s1 = Sierpinski(loc=loc, sideLength=sideLength/2),
      s2 = Sierpinski(loc=loc+<-sideLength/4, sideLength/4*RT_3>,
sideLength=sideLength/2),
      s3 = Sierpinski(loc=loc+<sideLength/4, sideLength/4*RT_3>,
sideLength=sideLength/2)
}
var depth = 10
draw Sierpinski:depth(sideLength=5)
```

The depth is required and can be any expression that evaluates to a non-negative integer.

2 - Data Types

2.1 - Int

An int is a 32-bit signed integer.

2.2 - Float

A float is a 64-bit signed floating-point number.

2.3 - Boolean

A boolean value defined using the keywords `true` and `false`.

2.4 - String

A string is a Java String.

2.5 - Ordered Pair

An ordered pair is a pair of x and y floats which can be expressed as a literal `<x,y>`. Note: the origin is the top left corner of the canvas and the y-value increases in a downward direction.

2.6 - List

A list is a Linked List which can be expressed as comma-separated items inside brackets. Lists can only contains elements of the same type. A list cannot contain two different types of figures.

2.7 - Color

A color is a hex colorvalue which can be expressed as the literal `#rrggbb`, `#rgb`, `#rrggbb@a`, or `#rgb@a` where `r`, `g`, and `b` are hex digits and `a` is alpha (opacity) on a decimal

scale from 0 to 1. Note: if single digit hex values are used, the digits are duplicated to produce double-digit values. For example, `#f03@.5` evaluates as `#ff0033@.5`.

2.8 - Figure

See Section 1.

3 - Operators

3.1 - Arithmetic Operators

- + (addition, string concatenation)
- - (subtraction, unary negation)
- * (multiplication)
- / (floating-point division, can be used on integers)
- // (floor division)
- ^ (exponentiation)

3.2 - Assignment Operators

- = (assigns RHS to LHS or passes RHS to argument with the identifier LHS)
- An arithmetic operator followed by = performs the operation with LHS and RHS and then assigns the value to LHS.

3.2 - Comparison Operators

- == (equal to)
- != (not equal to)
- > (greater than)
- < (less than)
- >= (greater than or equal to)
- <= (less than or equal to)

3.3 - Logical Operators

- ! (unary, logical not, casts operand to boolean)
- & (logical and, casts operands to boolean)
- | (logical or, casts operands to boolean)
- ? (unary, casts operand to boolean)

3.4 - Misc. Operators

- . (dot operator) - Selects property or method of a figure.
- : (recursion depth operator) - Specifies recursion depth of a recursive figure.
- [n] (list indexing operator) - Returns the nth element of the list (0-indexed)

4 - Statements, Comments, etc.

Statements are terminated by the end of a line. Blocks are placed between braces.

4.1 - Comments

Comments are declared using a semicolon and terminate at the end of a line.

4.2 - If-elseif-else Statements

```
if x < 3 {
      ; statements
}
elseif x > 7 {
      ; statements
}
else {
      ; statements
}
```

4.3 - While Statements

```
while i < 5 {
      ; statements
      i += 1
}
```

4.4 - Do-while Statements

```
do {
      ; statements
      i += 1
} while i < 5
```

4.5 - Global Variables

```
var foo = 5, bar = “hello”
```

Global variables can be accessed within any scope.

4.6 - Global Functions

```
def doSomething(arg1, arg2) {
      ; statements
}
```

Arguments in method and function calls are referenced by numerical order, not by name.

5 - Standard Library

5.1 - Built-in figures

- `figure Rect { param loc=<0,0>, width, height, col=#000, fill=true, lineWidth=1 }`
- `figure Arc { param loc=<0,0>, radius, start=0, end=2*PI, col=#000, fill=true, lineWidth=1 }`
- `figure Poly { param loc=<0,0>, points, col=#000, fill=true, lineWidth=1 }`
- `figure Text { param loc=<0,0>, text, col=#000, size=12, font=“Times New Roman”, align=“left” }`

Note: lineWidth only applies if fill=false

5.2 - Built-in statements

- `draw` - Draws a figure to the screen and pushes it to the internal figure list.
- `update` - Updates the figure on the internal figure lists and redraws all figures.
  If `update` is used without an argument inside a method, the instance of the figure that has called the method is updated.
- `erase` - Removes the figure from the internal figure lists and redraws all figures.

5.3 - Built-in functions

- `wait(ms)` - Delay for `ms` milliseconds.
- `toDeg(theta)` - Converts radians to degrees.
- `toRad(theta)` - Converts degrees to radians.
- `rgb(r, g, b)` - Creates a color literal given rgba values.
- `rgba(r, g, b, a)` - Creates a color literal given rgba values.
- `m_sin(x)`, `m_cos(x)`, `m_tan(x)`
- `m_asin(x)`, `m_acos(x)`, `m_atan(x)`, `m_atan2(x, y)`
- `m_log(b, e)`, `m_ln(x)`
- `m_ceil(x)`, `m_floor(x)`
- `m_sqrt(x)`
- `m_round(x)`
- `m_min(x, y)`, `m_max(x, y)`
- `m_abs(x)`
- `m_random()`

5.4 - Built-in constants

- `PI`
- `E`
- `RT_2` (Square root of 2)
- `RT_3` (Square root of 3)

Appendix A - Sample Code

```
color background = #fff ; global variable

figure PacMan { ; define a new figure
      ; arguments which are automatically stored under the same name
      param rad, ; arguments without default values are required
      loc = <0,0>, ; arguments with default values are optional
      col = #ff0

      ; components to be drawn
      comp body = Arc(loc=loc, radius=rad, col=col),
      mouth = Arc(loc=loc, radius=rad, start=PI/4, end=7*PI/4,
col=background, fill=true),
      eye = Arc(loc=loc+<rad/2,-rad/2>, radius=3, col=#000)

      ; misc. properties and methods
```

```
      var mouthOpenness = 1,
      isAlive = true
      def closeOpen() {
            while mouthOpenness > 0 {
                  wait(5)
                  mouthOpenness -= 0.1
                  mouth.start = mouthOpenness * PI / 4
                  mouth.end = mouthOpenness * 7 * PI / 4
                  update
            }
            while mouthOpenness < 1 {
                  wait(5)
                  mouthOpenness += 0.1
                  mouth.start = mouthOpenness * PI / 4
                  mouth.end = mouthOpenness * 7 * PI / 4
                  update
            }
      }
}

figure rec Sierpinski {
      param loc = <0,0>, sideLength
      comp triangle = Equilateral(loc=loc, length=sideLength,
fill=false),
      s1 = Sierpinski(loc=loc, sideLength=sideLength/2),
      s2 = Sierpinski(loc=loc+<-sideLength/4, sideLength/4*RT_3>,
sideLength=sideLength/2),
      s3 = Sierpinski(loc=loc+<sideLength/4, sideLength/4*RT_3>,
sideLength=sideLength/2)
}

draw PacMan(rad=10)
var p2 = PacMan(rad=4, loc=<6,6>)
draw p2
wait(100)
p2.loc += <4,-2>
update p2
p2.closeOpen()
erase p2
var depth = 10
draw Sierpinski:depth(loc=<100, 100>, sideLength=5)
```