better call **SOL**

SHAPE ORIENTED LANGUAGE

Aditya Naraynamoorthy
an2753

Erik Dyer
ead2174

Gergana Alteva
gla2112

Kunal Baweja
kb2896

September 23, 2017

Introduction

SOL is a simple language that allows programmers to create 2D animations with ease. Programmers will have the ability to define and create objects, known as shapes, and dictate where they appear, and how they move. 2D animations can aid programmers, engineers, and scientists in modelling algorithms, problems, and data. As an object-oriented language, SOL will allow unlimited design opportunities and ease the burden of animation. In addition, SOLs simplicity will save programmers the trouble of learning complicated third-party animation tools, without sacrificing control over behavior of objects. SOL has syntax similar to C++.

Example SOL Programs

SOL will commonly be used to model various types of scientific data, but it can also be applicable in other domains, such as entertainment.

- Environmental engineer modeling groundwater percolation in the refilling of aquifers

- Bored college student making funny meme gifs
- Enhanced data visualization
- The creation of any two dimensional turn-based game

Parts Of The Language

Data Types (Primitives)

1. `int` - Integer
2. `float` - Floating point number
3. `char` - ASCII character
4. `array` - ordered collection of objects

Basic Data Types

1. `point` - A single pixel at a location specified by coordinates in 2-D vector space
2. `curve` - Defined by three points to form Bézier curve

Grouped Types

1. `shape` - Similar to a class in C++; this implements a draw function that specifies how the shapes are statically rendered
2. `motiongroup` -Similar to a class; this type groups shapes together and performs one or more animation commands on all the grouped shapes

Keywords

1. `for` - Allows for quick iteration over an array of values
2. `if` - Allows for code execution if a condition is met
3. `while` - Allows for constant execution of a code block as long as condition is meant
4. `print` - Display in standard output

5. `func` - Declare a new function
6. `construct` - Declare a constructor
7. `main` - Declares mandatory function in which to define sequences of animations

Arithmetic Operators

1. `+` addition
2. `-` subtraction
3. `*` multiplication
4. `/` division
5. `%` modulo

Boolean Operators

1. `&&` AND
2. `||` OR
3. `!` NOT

Comments

- /* This is a comment */

Rendering commands

1. `draw` - Implemented for each shape to specify how the shape is drawn. *Arguments*: None
2. `drawpoint` - Render a single pixel at a location, specified by coordinates. *Arguments*: x (point)
3. `drawcurve` - Render a Bézier curve defined by three different coordinates. *Arguments*: x (point), y (point), z (point)
4. `drawtext` - Print out a string at a particular location. *Arguments*: s (char[]), x (point)

Motion Commands

1. `framerate` - Defined once at the start of the program, to specify the frames rendered per second. *Arguments*: rate (float)
2. `translate` - Move shape from one coordinate to another over a specified time period. *Arguments*: src (point), dest (point), time (float)
3. `rotate` - Rotate a shape around an axis point by a specified number of degrees over a time period. *Arguments*: axis (point), angle (float), time (float)
4. `render` - Describe the set of motions to be rendered. This function can be defined for shapes that need to move, or can be left undefined for non-moving shapes. Within this function, various rotate/translate calls can be made to move and shape. *Arguments*: None
5. `wait` - Pauses animation for a specified amount of time. *Arguments*: time (float)

Sample Interesting Program

The following program displays four triangles in a two dimensional plane. The triangles revolve around a common center point in clockwise direction and then two of the triangles revolve in anti-clockwise direction.

```
/* Make a group of rotating triangles */
framerate(24);

shape Triangle {
        point a;
        point b;
        point c;
        construct triangle(point a_init, point b_init, point c_init)
                a = a_init;
                b = b_init;
                c = c_init;
        }
        func draw() {
                /* Draw lines between the three vertices of the trian
                drawcurve(a, (a + b)/2, b);
```

```
                drawcurve(b, (b + c)/2, c);
                drawcurve(c, (c + a)/2, a);
        }
}
/* Create 4 triangles*/
Triangle triangle1((100, 50), (110, 70), (90, 70));
Triangle triangle2((160, 110), (140, 120), (140, 100));
Triangle triangle3((100, 170), (90, 150), (110, 150));
Triangle triangle4((40, 110), (60, 100), (60, 120));

/* Create a common point about which to rotate them */
point center(100, 110);

motiongroup trianglegroup1 {
        /* Group all 4 triangles together */
        triangle1;
        triangle2;
        triangle3;
        triangle4;
        func render() {
                /* Rotate each triangle about the center point */
                rotate(center, 360, 3);
        }
}

motiongroup trianglegroup2 {
        /* Group two of the triangles together */
        triangle1;
        triangle3;
        func render() {
                /* Rotate each triangle about the center point */
                rotate(center, -90, 1.5);
        }
}

func main() {
        /* Render the triangles statically first */
        triangle1;
```

```
        triangle2;
        triangle3;
        triangle4;
        /* Start the animation sequences */
        trianglegroup1;
        wait(1);
        trianglegroup2;
}
```