Final Project Proposal

Introduction

Production of computer graphics is a creative process; programs like Photoshop, Gimp, and, yes, even Microsoft Paint, allow the user's creative energy to run free. When it comes to a more algorithmic, geometric approach, however, humans cannot compete with computers. Programming languages like Logo serve this purpose and can be an educational way to learn geometry and programming for newbies. However, its educational bent, while valuable, provides an interface that is not optimal, due to its lack of indentation. Our language emphasizes syntactic and semantic transparency and readability in order to create a clean, consistent, and powerful interface for the production of vector graphics.

Design Principles

No meaning or functionality is hidden in class hierarchy. While emulating natural languages (as Logo does), clarity of logical flow takes the forefront. Whitespace acts as a way of delineating code blocks and parameters. This aids in readability and removes ambiguity of scope.

Examples

```
ahead 50        // moves the turtle ahead by 50 steps
clockwise 45    // rotates the turtle 45 degrees clockwise
cw 45           // also rotates the turtle 45 degrees clockwise

// Function called square that creates a square with sides of
// size length.
func square length {
    loop 4 {
        ahead length
        cw 90
    }
}

square 45       // calls function square
```

```
// Function called eqtri that creates an equilateral triangle
// with sides of size length.
func eqtri length {
    cw 30
    loop 3 {
        ahead length
        cw 120
    }
    ccw 30
}

// A recursive function called spiral that creates a shape that
// spirals outward with initial length initial and a final length
// final, rotating clockwise by angle degrees.
func spiral initial final angle {
    if initial > final) exit // base case
    ahead(initial)
    clockwise(angle)
    spiral(initial + 2, final, angle)
}
```

Language Concepts

The programmer must understand the concept of the turtle (the item which is moved around) and the pen (the item that draws the line. The basic commands all follow from these two ideas. If the programmer understands how to build modularly, they will succeed. Primitives of the language also aid in compositionality; these include the basic commands and the numbers given as arguments to these commands.

Syntax

Basic commands:

```
COMMAND          SHORT FORM    DESCRIPTION
ahead x          a x         // moves turtle ahead by x steps
behind x         b x         // moves turtle back by x steps
clockwise x      cw x        // rotates turtle x degrees clockwise
counterwise x    ccw x       // rotates the turtle x degrees
                             // counterclockwise
home                         // move turtle to center screen
eraser                       // change from the pen to the eraser
pen                          // change from eraser to pen
```

```
lift                        // allows one to move turtle without
                            // draw/erasing
press                       // allows one to resume draw/erasing
clear                       // clear the screen, return turtle
                            // home
pencolor x       pc x       // sets pen color
screencolor x    sc x       // sets screen color
```

Control flow:

```
if condition {              // Executes statement based on the
      statement             // verity of condition
}
if condition statement      // one-line if statement

if condition1 {             // Executes statement1 based on the
      statement1            // verity of condition1, else executes
} else if condition2 {      // statement2 based on condition2,
      statement2            // continuing until final else is
} else if ...               // reached (if it exists)
      ...
} else {
      default_statement
}

loop x {                    // executes statement1,2,3,...,k
                            // x times
      statement1
      statement2
      statement3
            .
            .
            .
      statementk

}
```

Function definition:

```
func name param1 param2 ... paramk { // code block definition
      statement1
      statement2
```

```
        statement3
                .
                .
                .
        statementk
}
```

Semantics

Primitives

Numbers, strings, and booleans comprise of the primitives of the language. Strings allow for the naming of new functions, while numbers are passed in as arguments to functions. In addition, vector lines form the basis for each command that is carried out by the virtual pen and turtle; from this we understand lines to be a primitive type. The window space that the lines are drawn in (the canvas) may also be understood as a primitive type. Booleans are used in control structures to provide logical flow.

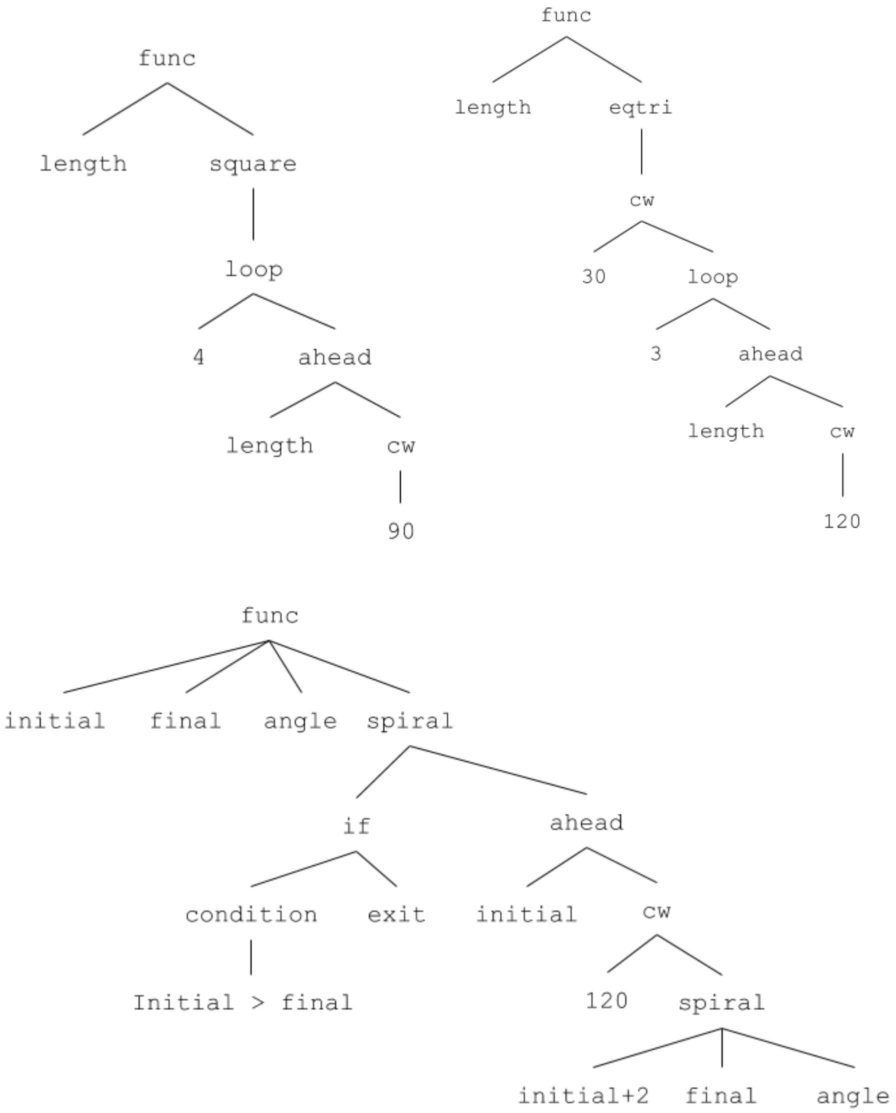
Actions and compositional elements

The compositional elements are the available commands that govern turtle/pen movement (`ahead, behind, clockwise, counterwise, home`) as well as whether to create lines, erase, or lift the pen altogether (`eraser, pen, lift, press`). It also includes the commands which allow a programmer to create and resize the window. These functions are described under “Basic commands” above.

Representation

At a high level, the program is represented with a `State` type, in which the current states of the `Canvas`, `Turtle`, and `Pen` are maintained and updated with every function. `Canvas` contains a list of `Lines` that have been drawn with the `Pen`. Turtle maintains the x and y position of the turtle item, as well as the angle it is pointing towards. `Pen` contains information about the thickness and color of the line, as well as whether or not it is being pressed down on the canvas. Since every function in this language is essentially side-effecting in respect to the `State`, each function should return a new copy of the updated `State`. The output of the program is an SVG file that is opened up in a web browser.

Abstract syntax trees



Evaluation

Programs read in numerical inputs to instruct the actions of the turtle and pen. These step-by-step actions in turn result in vector graphics that are the outputs of evaluation. The effect of evaluating a program is the production of a canvas, represented as a bitmap file. For example, the evaluation of the function square would go as follows:

1. 90 is given as an argument to `cw`
2. `cw` is executed, and length is given as an argument to `ahead`
3. `ahead` is executed, and 4 is given as an argument to `loop`
4. The body of the loop is performed 4 times
5. Finally, `length` is passed as an argument to `square`
6. This is all wrapped in a `func`, a function definition for `square`