SEE++
Language Reference Manual

By Adar Tulloch, Vishrut Tiwari, Winston Zhang, and Jack LaVelle

Introduction:

SEE++ is a language capable of making grayscale digital art. Starting with pixels as the base building block users will be able to build lines, shapes, 2D custom graphics, and more. They can combine with other user made models and build on previous projects. This project is inspired by the programming language Processing and we will follow the object oriented programming paradigm mixed with the simplified Java like syntax.

1. Lexical conventions

Tokens: the types of See++ tokens are described below: keywords, comments, and identifiers.

Keywords: the following are reserved keywords

1. int
2. char
3. float
4. bool
5. if
6. else
7. else if
8. for
9. main

Comments: all comments are introduced by /* and terminated by */.

Identifiers: Identifiers are a case-sensitive sequence of letters, numbers, and the underscore symbol. The first character of an identifier must be a letter or underscore. Characters must be members of the Unicode character set.

2. Data Types:

See++ supports fundamental types such as Pixel, Integer, Strings, and Booleans:

Boolean (declared and called `boolean`) singular bit of data used to assign true or false value

Integers (`int`) 32 bit and signed data

Strings (`String`) sequence of letters stored in the heap

Pixels (`Pixel(int)`) 8 bits of data to represent grayscale

Canvas(`Canvas(int,int)`) made up of two integers of 8 bits to initialize screen size

Operators:

Operations	Storage
!	Logical NOT operator
* /	Multiplication and division
+ -	Addition and subtraction
> < >= <=	Greater than, less than, greater than or equal to, less than or equal to
== !=	Equality, inequality
&& ||	Logical AND, logical OR
=	Assignment operation

Note: the above operations are listed in precedence with the operators at the top having the most precedence.

3. Functionality and Syntax

Function calls: See++ follows Java-like function calls:

```
type name(args){
...
}
```

The `type` here is the return value and the `name` is the function name. In SEE++ the first function sets up the canvas and is called set up and does not return anything back:

```
void setup(args){
canvas(500,500)
}
```

We also have another function called `draw()` which is the main function that continually runs code contained within its braces.

```
void draw(){
...
  }
```

List of built-in functions:

Functions	Description
`vertex(x,y, pixel)`	Places a point on the canvas where `x` and `y` are the coordinates and pixel carries the grayscale value
`fill(g)`	Set the grayscale color values of a canvas which have to be an integer between 0-255

`line(x1,y1, x2,y2)`	Draws a line with two coordinates
`triangle(x1,y1, x2,y2, x3,y3)`	Draws a triangle with float as input for position and three coordinates
`rect(a,b,c,d)`	Draws a rectangle with float as input for position `(a,b)`, one coordinate, width `(c)`, and height `(d)`
`quad(x1,y1, x2,y2, x3,y3,x4,y4)`	Draws a quadrilateral with float as input and four coordinates
`ellipse(a,b,c,d)`	Draws a ellipse with float as input for position `(a,b)`, one coordinate, width `(c)`, and height `(d)`
`background(g)`	Set canvas pixels to a color between 0-255

Variable declarations: similar to java ex: `int a = 5;`

Scope: variables can be declared globally and will be known to the draw and setup methods. However variables declared inside of the draw and setup methods otherwise known as lexically scoped will not be known outside that scope and to other methods.

4. Statements

Statements are sequences of code that are executed in order unless otherwise specified.

Blocks: statements may be grouped in blocks, delineated by braces `{}`.

Conditionals: conditional statements follow similarly from Java and include **if, else,** and **else if**. Conditional statements must be followed by a block of code that is executed when the condition is met.

```
if (exp1) { expr2 }
if (expr1) {
        expr2
} else if (expr2) {
        expr3
} else {
        expr4
}
```

To resolve “dangling-else” problem, See++ matches **else** with the closest elseless **if.**

For Loop: for loops are a form of iteration and syntax follows similarly from Java. A for loop contains initialization, a condition, and an updating clause, and must be followed by a code block.

```
for(int i=0; i<10; i++) {
/* code goes here */
}
```

While: while loops are another form of iteration similar to for loops. While loops contain a condition and a block of code to be executed while the condition is still met. The following is equivalent to the for loop example above.

```
        int i = 0;
        while(i < 10) {
                /* code goes here */
                i = i + 1;
}
```

Return Statement: the return statement returns a particular value from a function and exits the function. All non-void functions must return a value that matches the function signature. Void functions may use the simple return keyword to indicate the termination of a function.

Example:

```
void draw(){
int i = 0;
return  i;
  }
```

References

C Reference Manual

CompArt Final Report

PIXL LRM

Photon LRM