Sprite

an animation manipulation language
Final Report

Team Leader
Dave Smith

Team Members
Dan Benamy
John Morales
Monica Ranadive

Table of Contents

Introduction....5
A. Background & Motivation....5
B. Language Characteristics....5
B1. Simple & Efficient....5
B2. Memory and Compatibility....5
Language Tutorial....7
My First Sprite Program....7
A. Comments and statements....7
B. Printing in Sprite....7
C. Display an Image....7
D. Variables....7
E. Operations....8
F. Arrays....8
G. Loops and Conditional Statements....8
Loops....8
Conditional Statements....8
H. Functions....9
I. Objects....10
Compile and Run a Sprite Program....10
Example: Complete Program....11
Language Reference Manual....12
Introduction....12
Lexical Conventions....12
A. Tokens....12
B. Line Breaks & White Space....12
C. Comments....12
D. Identifiers....12
Feels Like the First Time....13
Namespaces and Bindings....13
Example 1....13
Example 2....13
D1. Primitive Data Types....14
Numbers....14
String Literals....14
D2. Objects....14
D3. Arrays....14
E. Keywords....14
F. Separators....15
G. Type-casting....15
H. Garbage Collection....15
Expressions....15
A. Primary Expressions....15
B. Arithmetic Expressions....16
B1. Unary Expressions....16
B2. Multiplicative Expressions....16
B3. Additive Expressions....16
B4. Logical Expressions....16
C. Assignment....17
Primitives are assigned by value. Variables as assigned by reference....17
Example 1....17
Example 2....17
C1. Left-Value Expressions....17
C2. Right-Value Expressions....18
Sprite Programs....18
A. Programs....18

B. Statements....19
B1. Functions....19
B2. Conditional Statements....19
B3. Add Statements....19
B4. Count Statements....20
B5. Iteration Statements....20
B6. Function Statements....20
B7. Object Statements....20
B8. Print Statements....21
B9. Sprite Statements....21
B10. Return Statements....21
Scoping Conventions....21
A. Scopes....21
B. Global vs. Local Scoping....22
Grammar....22
Project Plan....25
Overview....25
Team Member Responsibilities....25
Programming Style Guide....25
A. Antlr Programming Style....25
B. Java Programming Style....25
Project Environment....26
A. Operating Systems....26
B. ANTLR....26
C. Subversion – Version Control....26
D. Java....26
Project Log....26
Architectural Design....28
Testing....29
Testing Overview....29
Regression Testing Suite....29
Portion of Testing Text File....29
Advanced Testing Examples....30
Lessons We Have Learned....31
Appendix A – Testing Suite....32
SpriteTest.java – Testing Framework....32
Comments and Whitespace....36
Variables with Primitive Data Types....37
Arithmetic Operations....40
For Loops....41
Conditional Statements....43
Arrays....46
Objects....48
Functions....51
Scoping....59
Appendix B – Example Sprite Programs....65
BallAndNinjaGame....65
Sprite File....65
HTML Output File....66
Appendix C – Source Code....70
sprite.g....72
CommandLineSpriteCompiler.java....75
SpriteCompiler.java....77
ArrayAccess.java....78
ASTBuildingException.java....79
BinaryOperator.java....80

Block.java....81
Boilerplate.java....82
Comparison.java....83
ElseIf.java....84
Equals.java....85
ForLoop.java....86
Func.java....87
FunctionCall.java....90
GroupedExpression.java....92
HeterogeneousTree.java....93
Identifier.java....94
If.java....97
MemberHead.java....98
New.java....99
Node.java....100
NodeStringNode.java....103
Not.java....104
Num.java....105
Obj.java....106
OrAnd.java....107
Return.java....108
Scope.java....109
StatementList.java....111
Str.java....112
TargetLanguage.java....113
InvalidArgumentCountException.java....114
InvalidTypesException.java....115
LocalException.java....116
RedefinedException.java....117
ScopeException.java....118
SSAErrorMessages.java....119
SSAException.java....120
Type.java....121
UndefinedFuncException.java....122
UndefinedObjectException.java....123
UndefinedVariableException.java....124
Asserter.java....125
ASTViewer.java....126
File.java....128
HomogeneousTreePrinter.java....129
Numbers.java....130
BadJSExecutionException.java....131

Introduction

Sprite is a simple, fun language for displaying and moving 2-d images, or Sprites. You can use Sprite to program simulations or even video games. We wrote a compiler that translates Sprite programs into cross browser compatible JavaScript. A person does not need to have the Sprite compiler in order to use the compiled Sprite file since the file is in HTML. Accordingly, you can easily put your Sprite programs on your website, and your users can run them just by visiting a URL.

A. Background & Motivation

We built Sprite to solve several important problems.

1. Video games are very complicated to program right now. Difficult APIs, huge libraries, and complicated development environment installations provide a large barrier to entry for developers. On the other hand, Sprite simply requires a compiler and a browser.
2. Javascript is error prone and cross browser incompatible by nature. Any form of user interaction tends to be particularly inscrutable. Sprite abstracts away these problems, providing fewer bugs and cross browser compatibility.
3. The web simplifies the deployment process by removing the requirement for client installation. Sprite brings that advantage to 2-d simulations and video games.

The motivation behind Sprite was to take one popular aspect of JavaScript, sprite manipulation, and make it both powerful and easier to program. The general idea is to make sprite manipulation accessible to a programmer of any level as well as non-programmers. Computer graphics programs typically consume numerous lines of code and require unnecessarily idiosyncratic programming from the developer in order to create a finished product. Our language offers a streamlined and more efficient way of displaying and manipulating sprites without programming in JavaScript or Action-Script.

B. Language Characteristics

B1. Simple & Efficient

A Sprite program can be written, compiled, and the end result viewed within minutes. The language was designed to be used by both experienced programmers as well as individuals who are brand new to programming. The language is not intimidating like other languages that provide specifications for animations. The following program is all that is needed to display a JPG image in an HTML window at the coordinates (50, 50):

```
sprite("Ball.jpg", 1, 50, 50)
```

Our compiler takes this bit of code, adds the necessary HTML and JavaScript tags, and converts the Sprite code to legal JavaScript code.

B2. Memory and Compatibility

A Sprite program is compiled to JavaScript so memory management and garbage collection are handled by the browser the user has. Memory management and garbage collection may depend on the user's browser settings.

Sprite is a portable language. The Sprite compiler also handles browser compatibility issues with JavaScript. All Sprite programs can be compiled and run on any system since the source language compiles to JavaScript. As long as the user has a browser to open a .html file, it is Sprite-compatible.

Language Tutorial

My First Sprite Program

Here is a basic Sprite program that we will use to get started:

```
<< a simple Sprite program
print("Hello world!")
```

A. Comments and statements

The first line of the above program is a comment. Comments can be placed in a Sprite program using <<, and anything from the << to the end of the line of code is ignored by the compiler. Everything else in Sprite is a statement and will compile to JavaScript.

B. Printing in Sprite

The built-in `print()` function will print the parameters within a browser window. In the above case it is used to output the literal string "Hello world!".

C. Display an Image

In order to print a sprite within a browser window, you would use the sprite() function.

sprite(imageName, imageIDnum, topCoordinate, leftCoordinate)

This line will display the imageName with the top left corner of the image positioned at the coordinate (topCoordinate, leftCoordinate).

The imageIDnum will define whether a new image is created or an old one updated.

D. Variables

The most basic building blocks of a sprite program are variables. Variables can equal a number, boolean, array, or string. For example:

```
foo = 9
```

sets the variable foo to 9. Since Sprite is dynamically typed, this same variable can be assigned a different value later on in the program:

```
foo = "bar"
```

Variable names can only contain letters, numbers, and underscores. The name must begin with a letter or underscore.

E. Operations

Sprite uses the follow arithmetic and logical operators:

```
2 + 4                                   << addition of 2 and 4
6 - 2                                   << subtraction of 2 from 6
8 / 2                                   << division of 2 into 8
2 * 5                                   << multiplication of 2 and 5
5 mod 2                                 << 5 modular 2
a || b                                  << boolean value of a or b
a && b                                  << boolean value of a and b
!a                                      << boolean value of not a
a > b                                   << boolean of is a greater than b
a < b                                   << boolean of is a less than b
a >= b                                  << boolean of is a greater than or
                                        << equal to b
a <= b                                  << boolean of is a less than or equal
                                        << to b
a == b                                  << boolean of is a equal to b
a != b                                  << boolean of is a not equal to b
```

F. Arrays

Arrays can be created by the following statement:

```
foo = new array
```

The array is empty when it is created. It will expand to the needed size as elements are added.

```
add(foo, 5)
add(foo, "bar")
```

You may add elements of different types to the same array. These elements can then be accessed using the following statement:

```
a = foo[0]                              << a is now the integer value 5
b = foo[1]                              << b is the literal string "bar"
```

If you want to know the size of an array you may call the count() function.

```
array_size = count(foo)                 << in this case, it will return 2
```

G. Loops and Conditional Statements

Loops

For loops are allowed in Sprite. To iterate through a loop n times (from 0 to n), you could program the following loop:

```
for(i = 0; i <= n; i = i + 1) {
    << body of for loop....
}
```

Conditional Statements

Sprite allows if/else statements as well as if/elseif.../else statements. The final else is optional. Here are some examples of implementing a conditional statement.

```
<< if statement
if(a - b == 0){
    << body of if statement
}

<< if/else statement
if(a < b) {
    << body of if statement
}
else {
    << body of else statement
}

<< if/elseif statement
if(a < b) {
    << body of if statement
} elseif(a == b) {
    << body of elseif statement
}

<< if/elseif/else statement
if(a == b){
    << what do in the if statement
} elseif (a > b) {
    << what to do in the elseif statement
} else {
    << what to do in the else statement
}
```

H. Functions

If there is a section of code you want to reuse several different times throughout your program, you could create a function to perform the task(s). A function may take in zero or more parameters. These parameters are only valid within the function and cannot be accessed from outside the function's scope. Function arguments are treated as assignments. Primitive are passed by copy and objects created with `new` are passed by reference.

```
func decrementOne(bar){
    bar = bar - 1
    ret bar
}
foo = 5
foo = decrementOne(foo)
```

In this example the function will take in a parameter and decrease that parameter by one. In order for this function to make changes outside of its scope, we needed to add a return statement. Return statements are only allowed within a function. It is possible to have multiple return statements:

```
func compare(a,b){
    if(a == b) {
        ret true
    } else {
        ret false
    }
}
```

This function acts like the == operator.

You may also add a callevery*(arg)* attribute that will continuously run the function every *arg* milliseconds.

```
func decrease(bar) callevery(10) {
    bar = bar - 1
    print(bar)
}
```

This function will run an infinite loop of decreasing the parameter *bar* by 1 at each iteration and printing it on the screen. The callevery() attribute is typically called to continually update the location of a graphic on the screen (aids in the animation of sprites). Our team was also able to integrate user input into the programs in the form of on keyboard events.

```
x = 100
func printx() onkeyboard(1) {
    print(x)
    x = x - 1
}
```

Once this program is running, each time the user presses the "1" key, the variable x will be printed and then decreased by one. The `onkeyboard` attribute only accepts single keyboard strokes in 0...9 or a...z character ranges.

I. Objects

Objects allow you to create a new data structure that can contain attributes. Objects may contain variables that are equal to any data type or even another object. The variables assigned to an object may be accessed using *objectName.attribute.* If you wanted to create a new object that would describe a person you could do the following:

```
obj person {
    firstName = "Jane"
    lastName = "Doe"
    age = 23
    birthday = "February 16th"
    id = 0
}
jane = new person
func marriage(wife, husband){
    wife.lastName = husband.lastName
}
john = new person
john.firstName = "John"
john.lastName = "Smith"
marriage(jane, john)
```

This program creates a new person named "Jane Doe" and modifies her last name using the function marriage().

Compile and Run a Sprite Program

You can write Sprite programs using any text editor and save in the format *fileName.sprite*. Once you have our compiler installed on your system;

```
java CommandLineSpriteCompiler fileName.sprite
```

This command will compile and output the HTML file to the same directory as the file.

Example: Complete Program

The sample of code describes a single sprite, ball, bouncing around the screen of a browser window.

```
obj Ball
{
     left = 50
     top = 50
     downDirection = 3
     rightDirection = 3
     id = 0
}

newBall = new Ball
newBall.id = 1

func MoveBall() callevery(10)
{
     newBall.top = newBall.top + newBall.downDirection
     newBall.left = newBall.left + newBall.rightDirection

     if (newBall.top <= 0 || newBall.top >= 400) {
          newBall.downDirection = - 1 * newBall.downDirection
     }

     if (newBall.left <= 0 || newBall.left >= 700) {
          newBall.rightDirection = - 1 * newBall.rightDirection
     }

     sprite("Ball.jpg", newBall.id, newBall.top, newBall.left)
}
```

Language Reference Manual

Introduction

This language reference manual outlines the Sprite programming language. In order to describe the language, regular expression notation has been used. The following is the notation that will be used within the regular expressions:

( ) – used for grouping both terminals and nonterminals

'a' – terminal

a? – the symbol 'a' is optional

a* – the symbol 'a' may occur, zero, one, or more times

a+ – the symbol 'a' may occur, one or more times

a | b – a choice between the symbols 'a' and 'b'

The syntax notation used in this manual is formatted as follows: syntactic categories are indicated by *italic* type, and literal words and characters are in `typewriter` style.

Lexical Conventions

A program consists of one Sprite specification stores in a file. Programs are to be written using the ASCII character set with characters from 0 to 127.

A. Tokens

The language contains five classes of tokens: identifiers, keywords, numbers, string literals, and operators. Spaces and tabs are classified as “white space” and are ignored by the lexer. Comments are only recognized for the purpose of ignoring them.

B. Line Breaks & White Space

A line break denotes the end of a statement. Allowed line break are '\n', '\r', or '\r\n' to allow native files from different operating systems to be compiled. One or more line breaks are allowed between lines of code. Lines of code may not wrap. Comments and most formatting, which includes spaces and horizontal tabs are ignored. Sprite occasionally requires whitespace to separate adjacent tokens that would otherwise be a single token. For example, identifiers and keywords must have separating whitespace. The newline character at the end of a file is optional.

C. Comments

Single line comments start with `<<` and end at the end of line ('\n', '\r' or '\r\n'). There are no special characters to denote multiple line comments.

D. Identifiers

An identifier is a sequence of letters or digits, the first of which much be a letter. Letters are

considered any letter of the alphabet (upper and lower case) as well as '_'. Upper and lower case letters are different. Two identifiers are the same if they have the same ASCII character for every letter and digit.

identifier : *letter (letter | digit)**

Variables or objects are created the first time an identifier is assigned to an expression (rvalue) and their type is determined automatically. Variables are not declared.

Feels Like the First Time

The first time a variable is referenced can be one of the following:

- Assignment
- Assignment within the initialization stage of a for loop
- An argument of a function

```
<< count, a global variable, is created with an assignment
count = 5

<< x is created as the argument of a function
<< foo is a local variable of myFunction
func myFunction(x){
    local foo = 10
    print(x + foo)
}

<< i is created within the initialization of a loop
for(i = 0; i < count; i = i+1){
    myFunction(i)
}
```

Namespaces and Bindings

Sprite has only one namespace. Once a name is assigned to a variable, function or object, that name cannot be used to create one of the other two types. In addition, local variables may not have the same name as a function or object. Names are statically bound in our language.

Example 1

Using the same name for a variable and a function or object is not allowed. In the case below the function foo is created in the parent scope of the variable foo. Even though the variable foo is only valid within the function bar(), it is still not allowed to share the same name of the function foo().

```
func foo() {
}
func bar() {
    local foo = 1
}
```

Example 2

The following case is not allowed because the variable foo does not have the “local” prefix so it a global variable and the function foo with the same name is in the same scope.

```
func foo() {
}
func bar() {
    foo = 1
}
```

D1. Primitive Data Types

The primitive data types for identifiers are strings, numbers, and boolean values (true/false). Sprite does not require the type of an identifier when it is declared. In Sprite, types are unknown at compile time. Additionally, the type of variables may be changed as the program runs. There is type-casting for Strings in certain situations, see §B9.

Numbers

Numbers are a sequence of digits representing an integer. The number -0 is allowed in sprite, it will simply compile to 0. Sprite does not support decimal values. Numbers must be in the range -2147483648 to 2147483647 inclusive.

number : *digit+*

String Literals

A string literal, or string constant, is a sequence of characters surrounded by double quotes, like " ... ". String literals can be concatenated using the addition operator. String literals may not contain double quotes.

D2. Objects

Objects may contain other objects and primitive data types. Objects specify defaults for all primitive data types. Objects cannot contain functions.

```
obj foo
{
    bar = 1
    rab = true
    arb = "Hello world"
}
```

D3. Arrays

Arrays are created with the `new array` keywords. The `add(`*array_identifier*`,` *identifier*`)` function may append any primitive or object to an array. The `count()` function returns the number of elements in an array. Access the nth element of an array using *array_identifier*[*n – 1*]. Arrays manage their own memory, and allocate space as more elements are added.

Array out of bounds errors will return a special value "undefined". If this is used in a multiplication or division, the result will be NaN (not a number).

E. Keywords

The following identifiers are reserved for use as keywords, all keywords use only lowercase letters.

```
add             false           onkeyboard
array           for             print
callevery       func            sprite
count           local           ret
if              mod             true
else            new
elseif          obj
```

F. Separators

The following characters are separators:

```
{ } , . [ ]
```

NOTE: In order to keep the examples short, curly brackets may appear on the same line in the grammar. This will not compile correctly, please read below.

The left curly bracket { may only appear at the end of a line or on its own line. The right curly bracket } can usually appear on its own line or at the end of a line. However if it precedes an `elseif` or an `else`, it must appear on the same line as the following keyword.

G. Type-casting

Addition of a String with any primitive non-String implicitly type casts the non-String to a String. The `print(`*expression*`)` function implicitly type casts any non-String argument as a String.

H. Garbage Collection

Objects created on the heap (with new) will be garbage collected at some point after the last reference to them is destroyed. Primitives (variables assigned a value not created with new) are places on the stack and disappear when the function they're in returns. If they are in the global scope, they stay in existence for the length of the program.

Compiling to JavaScript has its positives and negatives. Fortunately, anyone with a browser can run a compiled Sprite program. Unfortunately, not all browser settings are the same. Garbage collection and memory allocation are going to depend somewhat on the browser's implementation of JavaScript.

Expressions

The precedence of expression operators is identical to the order in which they are documented below. Within each subsection, the operators have the same precedence. Each subsection is left-associative unless otherwise specified. The order of evaluation of expressions is left to right.

Expressions are the building blocks of statements and programs in the Sprite Language.

A. Primary Expressions

factor:

| **(** *expression* **)**

| *left-value*
| *number*
| *boolean*
| *string*
| *function-call*

B. Arithmetic Expressions

Arithmetic expressions take the primary expressions as their operands. While the productions of the expressions appear to create right associative rules, when implemented in ANTLR, the language is left-associative.

B1. Unary Expressions

unary:
| **-** *unary*
| **!** *unary*
| *factor*

A unary expression is an operation with at most one operand ('-' or '!'). The prefix notation binary operators '-' and '!' represent negative and not.

B2. Multiplicative Expressions

mult-expression:
| *unary* (* *unary*)*
| *unary* (/ *unary*)*
| *unary* (`mod` *unary*)*

The binary operators '*' , '/', 'mod' represent multiplication, division, and modulus.

B3. Additive Expressions

add-expression:
| *mult-expression* (+ *mult-expression*)*
| *mult-expression* (- *mult-expression*)*

The binary operators '+' and '-' indicate addition and subtraction.

B4. Logical Expressions

comp-expression:
| *add-expression* (> *add-expression*)*
| *add-expression* (< *add-expression*)*
| *add-expression* (>= *add-expression*)*
| *add-expression* (<= *add-expression*)*
equal-expression:
| *comp-expression* (== *comp-expression*)*
| *comp-expression* (!= *comp-expression*)*
join-expression:
| *equal-expression* (&& *equal-expression*)*

expression:
| *join-expression* (|| *join-expression*)*

Logical expressions can intuitively be used to compare primary expressions. Using the Kleene Closure, an expression can evaluate to any combination of comparisons and mathematical expressions or to a simple terminal such as a number or string.

C. Assignment

To assign an argument a value, a declaration must be made as detailed below:

assignment:
| *left-value* = *right-value*

Primitives are assigned by value. Variables as assigned by reference.

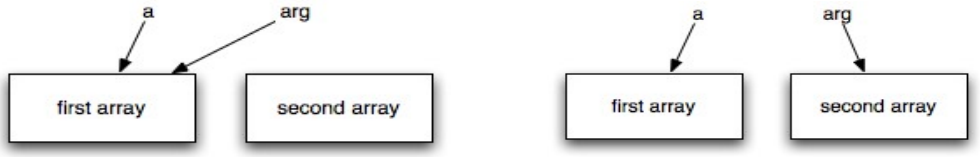
Example 1

```
a = new array
b = a
add(b,5)
```

In this example, at the end both a and b still have the same value. There is only one array, both a and b point to it.

Example 2

```
func foo(arg)
{
    << arg = new array
    add (arg, 2)
}
a = new array
add (a, 5)
foo(a)
```



With the comment left in, the above code will evaluate to the left figure. If the line is uncommented, a new array will actually be created that arg points to, as in the right figure.

C1. Left-Value Expressions

left-value:
| `local` *identifier*
| *identifier (array* | *member)*
array:
| `[` *expression* `]`

member:

| (**.** *identifier*)+

A left-value expression describes all valid syntax that may appear on the left hand side of an assignment. This is used for declaring/accessing identifiers, arrays, and characteristics of identifiers. When the keyword `local` precedes an identifier within a function block, that identifier's scope is limited to that particular function block.

C2. Right-Value Expressions

right-value:

| `new array`
| `new` *identifier*
| *expression*
| *function*

A right-value expression describes all valid syntax that may appear on the right hand size of an assignment. This is used for declaring and accessing arrays, identifies, expressions, and functions.

Sprite Programs

A. Programs

Sprite programs are a sequence of one or more statements.

program:

| *statement* +

Certain types of statements may contain blocks of code made up of more statements which will be formatted as follows:

block:

| **{** *statement* * **}**

B. Statements

Statements are executed in sequence. Statements can fall into several groups.

statement:
| *function-call*
| *conditional-statement*
| *add-statement*
| *count-statement*
| *iteration-statement*
| *func-statement*
| *object-statement*
| *print-statement*
| *sprite-statement*
| *ret-statement*
| *assignment*

B1. Functions

Functions are created using a function-statement, with a name (their identifier) as well as a list of parameters (expressions) which the function will take upon declaration. Functions may accept zero or more parameters, each separated with a comma. The function-call statement refers to the correct grammar for calling a function. Forward declarations are allowed within Sprite. Parameters of a function are evaluated left to right; this is important because it is possible to have side effects in Sprite. A function call is valid as long as the function is defined somewhere in the Sprite file.

function-call:
| *identifier* (*(expression (, expression)*)?*)

B2. Conditional Statements

Conditional statements must be formatted as follows.

conditional-statement:
| `if` (*expression*) *block* (`elseif` (*expression*) *block*)*
(`else` *block*)?

In both forms of the `if` statement, the *expression* is evaluated and if it evaluates to `true` the *block* on the following line is executed. In the second form, the second *block* is executed if the *expression* evaluates to false. There is no else ambiguity because of the rule that { }'s must surround a block.

B3. Add Statements

Add statements allow elements to be added to arrays.

add-statement:
| `add` (*identifier , expression*)

An add statement consists of an identifier representing an array and an expression representing the

element to be added to the array. Arrays are heterogeneous so it does not matter to what type *expression* evaluates.

B4. Count Statements

Count statements will evaluate the number of elements in a given array.

count-statement:
| `count ( identifier )`

The *identifier* represents an array and the statement will evaluate to the number of elements held in the array specified. Since Sprite is dynamically typed, the compiler can't tell if the argument is an array or not. Calling `count()` on a variable that isn't an array will result in a run-time error.

B5. Iteration Statements

Iteration statements specify looping.

iteration-statement:
| `for (` *assignment* `;` *expression* `;` *assignment* `)`
block

Unlike other languages, the arguments within the *for* statement are required for the loop to compile correctly. In the `for` statement, the first assignment is evaluated once, and specifies the initialization for the loop. Curly brackets are required around the body of the loop. The contents of the *block* must start on a new line of code; only comments and the open bracket may go on the same row as a `for` statement.

B6. Function Statements

func-statement:
| `func` *identifier(callEvery | onKeyboard)?*
block
callEvery:
| `callevery (` *number* | *left-value* `)`
onKeyboard:
| `onkeyboard (` *number* | *string* | *left-value* `)`

Functions executes a block of sprite code with optional arguments, passed by reference. A function may include a callevery option. The callevery(*num*) option causes the function to be repeatedly executed every *num* milliseconds. Callevery will only accept nonnegative integers or expressions that evaluate to nonnegative integers. The onkeyboard(*char*) option causes the function to only execute on a keyboard event. The *char* can be the ranges 0...9 or a...z or can be an expression that evaluates to a character in that range. If it is not, the function will simply not be called.

B7. Object Statements

A object statement can be written only one specific way. Object statements are used to declare *objects* in Sprite.

object-statement:
| `obj` *identifier* { *(assignment end)* * }

Object statements may contain a sequence of assignments with newlines. These assignments will become members of the object.

B8. Print Statements

The print statement can be used to display data. It follows this flow control.

print-statement:
| `print (` *expression* `)`

If the statement is passed something other than a string, the expression will be cast to a string.

B9. Sprite Statements

Sprite statements are obviously a particularly important section of the language.

sprite-statement:
| `sprite (` *string*|*member* , *number*|*member* ,
number|*member*, *number*|*member* `)`

A sprite statement accepts a string/member (ex. the name of the .jpg file) as well as three additional parameters that evaluate to numbers. The numbers refer to the coordinates where the image file will be displayed.

B10. Return Statements

A return statement is only used within a function to return a particular right-value.

ret-statement:
| `ret` *rvalue*

Scoping Conventions

A. Scopes

Every variable in a Sprite has a scope: either global or local. This scope determines where throughout the program the variable may be accessed. Since our variables are not declared, just created or modified, our programming language using static scoping since all instances of 'x' refer to x. Functions do have access to global variables, Sprite has open scoping.

B. Global vs. Local Scoping

All variables are global by default. Variables can be made local in two ways. First, arguments of a function are only valid within that particular function. Second, when a variable created within a function is first used the `local` keyword may precede the left-value. This keyword will make the variable local to that particular function.

```
x = 5
func foo(y) {
    << x is a global variable while y is a local variable
    number = x + y
    print("number = " + number + " and y = " + y)

    << newNumber is a local variable as well
    local newNumber = number + y
    print("new number = " + newNumber)
}
```

Grammar

Below is a recapitulation of the grammar that was given in the earlier part of this section. It has the same content, but in a different and condensed order. Once again, symbol is *italics* are non-terminals, while symbols in `typewriter` style are terminals given literally. The undefined terminal symbols are string and digit. A string is a sequence of one or more characters 'a'..'z' – for onkeyboard the only valid string is a single lowercase character. Digits are the numbers 0 .. 9. This grammar can be used to create a Sprite file that will be successfully parsed by the Sprite compiler.

identifier : *letter (letter* | *digit)**

number : *digit*+

left-value:
| `local` *identifier*
| *identifier (array* | *member)*

array:
| `[` *expression* `]`

member:
| *(* `.` *identifier)*+

right-value:
| `new array`
| `new` *identifier*
| *expression*
| *function*

assignment:
| *left-value* = *right-value*

expression:
| *join-expression* (`||` *join-expression*)*

join-expression:
| *equal-expression* (`&&` *equal-expression*)*

equal-expression:
| *comp-expression* (== *comp-expression*)*
| *comp-expression* (!= *comp-expression*)*

comp-expression:
| *add-expression* (> *add-expression*)*
| *add-expression* (< *add-expression*)*
| *add-expression* (>= *add-expression*)*
| *add-expression* (<= *add-expression*)*

add-expression:
| *mult-expression* (+ *mult-expression*)*
| *mult-expression* (– *mult-expression*)*

mult-expression:
| *unary* (* *unary*)*
| *unary* (/ *unary*)*
| *unary* (mod *unary*)*

unary:
| – *unary*
| ! *unary*
| *factor*

factor:
| (*expression*)
| *left-value*
| *number*
| *boolean*
| *string*
| *function-call*

program:
| *statement* +

block:
| { *statement* * }

statement:
| *function-call*
| *conditional-statement*
| *add-statement*
| *count-statement*
| *iteration-statement*
| *func-statement*
| *object-statement*
| *print-statement*
| *sprite-statement*
| *ret-statement*
| *assignment*

function-call:
| *identifier* (*(expression (, expression)*)?*)

conditional-statement:

| `if` *(* *expression* *)* *block* *(* `elseif` `(` *expression* `)` *block* *)** *(* `else` *block* *)?*

add-statement:

| `add` `(` *identifier* , *expression* `)`

count-statement:

| `count` *(* *identifier* *)*

iteration-statement:

| `for` *(* *assignment* `;` *expression* `;` *assignment* *)* *block*

func-statement:

| `func` *identifier(callEvery | onKeyboard)?* *block*

callEvery:

| `callevery` `(` *number* | *left-value* `)`

onKeyboard:

| `onkeyboard` `(` *number* | *string* | *left-value* `)`

object-statement:

| `obj` *identifier* { *(assignment* `\n`*)* * }

print-statement:

| `print` `(` *expression* `)`

sprite-statement:

| `sprite` `(` *string*|*member* , *number*|*member* , *number*|*member*, *number*|*member* `)`

ret-statement:

| `ret` *r-value*

Project Plan

Overview

We successfully divided the project into weekly team and individual goals. This allowed our group to work together on major decisions as a team while individually contributing our own talents to the project throughout the week.

Within the first month we had divided up initial responsibilities that we would be in charge of throughout the project (Dave = Leader & Backend, John = Frontend with Dan helping out with Grammar, Monica = Documentation, Dan = Testing). As new tasks came along, Dave delegated the work to the group evenly and continuously made sure each person had something to do before the next weekly meeting.

Team Member Responsibilities

Our team worked well together, our interests and strengths greatly complimented each other. We met at least once a week over the past four months for an hour (and in some cases longer). Each meeting we worked on making decisions as a group as well as assigning individual goals to achieve for the following meeting.

Dave Smith	Leader, Abstract Syntax Tree, Backend, Testing
Dan Benamy	Testing Framework, Grammar, Testing
John Morales	Static Semantic Analysis, Grammar, Testing
Monica Ranadive	Documentation, Static Semantic Analysis, Testing

Programming Style Guide

The following sections detail the coding conventions that were employed throughout our project.

A. Antlr Programming Style

The colon ':' always starts at the 4th space (one tab), if the rule is short enough everything can be placed on a single line. Each rule/action starts at the 8th space (two tabs). The ';' is placed at the end of the line unless the rule has more than one action. With multiple rules, the semi-colon ';' is placed on its own line indented one tab.

Token names (ex. LPAREN RPAREN) appear in upper case while non-terminals (ex. expression, lvalue) appear in lower case.

B. Java Programming Style

Our team used the standard Java Coding Conventions found on Sun's website. The most important aspect of the code is indentation and spacing. By keeping the formatting consistent, our code is more readable and easier to debug.

- Indentation at each level is an additional 4 spaces (one tab) from the previous level

- The left brace '{' appears at the end of a line of code. The right brace '}' always occupies its own line unless immediately followed by an `else` or `elseif` statement.
- Brackets { } are **always** used to enclose blocks of code.
- Parameters are typed `(param1, param2, param3, ...)`

Project Environment

The majority of the programs are coded in Java (or translated into Java from ANTLR).

A. Operating Systems

Our group members used both Windows and Mac OS to complete this project. Since Eclipse is a cross-platform application, we had very few compatibility problems. Our SVN server was hosted on one of the machines in the CLIC lab.

B. ANTLR

Both the Lexer and Parser for Sprite are written in ANTLR v2. Antlr assists in the construction of a programming language compiler using simple grammatical statements.

C. Subversion – Version Control

In order for 4 people to collaborate on a single project, we needed some form of version control. Dan suggested using Subversion (SVN) and was able to set up the server on a CLIC lab machine. Our team used the SVN plug-in for Eclipse to commit and update our project.

D. Java

Antlr was used just to generate the lexer and parser using our grammar file. The remainder of our project was coded in Java. The abstract syntax tree, static semantic analysis, as well as the testing suite were all implemented using Java.

Project Log

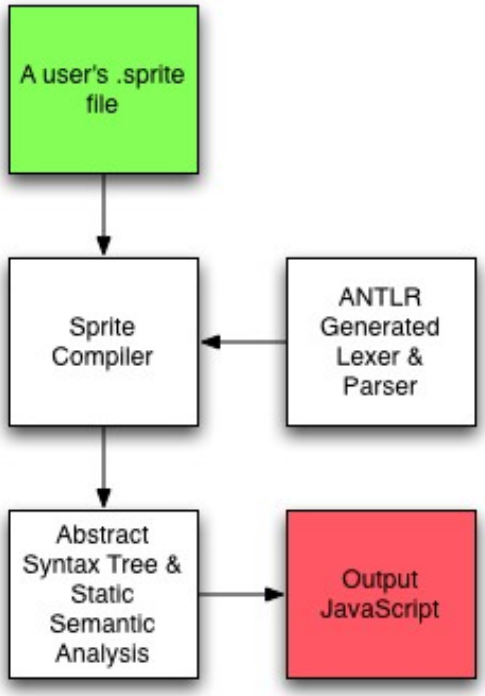
Here are the dates of significant project milestones.

September 12	First team meeting, brainstorm language ideas
September 19	Decided on Sprite language and began the proposal
September 25	Submitted programming language proposal
September 26	SVN Repository Created – all team members set-up for SVN
October 1	Grammar file started, began LRM
October 8	Backend/Treewalker started
October 17	Finished the LRM
October 19	Grammar file completed, Backend/Treewalker completed
November 1	Testing Suite created, testing started, began final report

November 14	Static Semantic Analysis started
December 2	SSA completed, project completely in testing phase
December 18	Completed project, submitted final report

Architectural Design

Our programming language compiles to another programming language.



A program begins as a .sprite file. When compiled it is first run through the lexer and parser that we created using ANTLR. If the program is syntactically correct it will be built into an abstract syntax tree. The tree walker, static semantic analysis, and symbol table are all built into the AST package. In order to check the semantics of the program we take two passes over the generated tree, the first while the tree is being built, and the second after the tree has been completed. Overriding toString() methods we build the respective JavaScript. The output is in the form of a HTML file.

Testing

Testing Overview

The goal of testing is to find faults in the compiler that we might not catch quickly if at all. We tried to minimize the amount of testing needed by testing our project at different stages (correct grammar, correct AST, correct semantics, etc.). While it's impossible to finish with a perfect compiler, we found more bugs by doing incremental testing and greatly improved the overall quality of our project. The biggest form of automated testing we did involved compiling Sprite programs and running them (where applicable) to make sure the output was correct.

Our programming language compiles to JavaScript, so it was difficult to test the output. We used a Java library called httpunit, which is an implementation of the JavaScript language where we could programmatically inspect the results. We created tests that were supposed to fail compiling, compile, or compile and produce a certain output when executed.

Regression Testing Suite

We built a custom automated testing framework to test the Sprite compiler. It's a java program which uses the httpunit library. We had a number of files with small test cases in them which each tested some particular behavior. The test files have a very simple grammar where “\n>>>\n” marks the start of the test, and “\n>>> X\n” marks the end of the test. X can be “OK”, “BAD”, or an actual value to be produced.

The tester ran each snippet through the compiler and checked whether it compiled or not compared to what it was supposed to do. If the test case specified an output the code was expected to produce, it would run the output through httpunit which would execute the javascript. Then it would compare that result against the test case's. Information about the test was printed to the screen. This includes back traces for certain compile errors. The more difficult problems were with incorrect code which compiled correctly.

The httpunit library has an embedded http server and had a web client which downloads web pages and run any JavaScript present. We could then check the state of the DOM in the client and make sure it was correct.

Out test suite does not provide full coverage for all the language's features and interactions. It does, however, cover a great deal and was very useful in preventing regressions.

Portion of Testing Text File

```
>>>
x = new array
add(x, 5)
>>> OK

>>>
x = new array
add(x, .a5)
>>> BAD
```

```
>>>
x = new array
add(x, "foo")
>>> OK

>>>
func zero(x) {
	if (x <= 0) {
		print(0)
	} else {
		zero(x - 1)
	}
}
zero(4)
>>> 0
```

Please see Appendix A for a more detailed list of some of the tests that we ran through our compiler.

Advanced Testing Examples

All of the tests that were run using our regression suite were short programs. We also needed to test more complicated programs that used numerous features of our language at once. These programs we tested on a case by case basis. Not only did these tests reveal some bugs in our compiler, but we were able to see the power of our compiler. Since we were ahead of schedule we were able to implement on-keyboard events so that our animation language could be extended to video games.

Lessons We Have Learned

Throughout this semester we have learned some lessons:

- Designing your own programming language is really exciting but don't get caught up in thinking about all the features you want to implement – get the basics working and improve on what you have. Fortunately we had extra time at the end of the semester to implement another feature of our language (user input events).
- While we were told this at the beginning of the semester, it can never be stressed enough—start early. Even though our group was typically ahead of schedule and never felt too pressured, it would not have hurt to have a few extra days at each milestone to really make sure our code/reports were solid.
- We were amazed at how many details actually went into each portion of the language design process (formatting conditionals, scoping, where return statements are allowed, etc). It can be a little overwhelming at first but Antlr was a big help. When beginning the design process, Antlr was really helpful with quickly getting the syntax for your language outlined.
- It's nice to have four people in a group to bounce ideas off of and complain to about bugs in your language. Having this many people can also be problematic simply because everyone has a different work ethic – it's important to communicate with your group members. Two types of people who are important to group are one to think about the immediate task at hand (to make sure tasks are completed on time) and a second to think about the long term goal for the project (have the whole picture in mind of what needs to be accomplished). Groups should learn about each other's strengths (and weaknesses) before beginning the project to make sure that everyone can contribute in their own way.
- Our group worked quickly toward getting a working product – but the automated testing helped show us where all our mistakes were. By putting together a testing suite, we were able to quickly isolate bugs and correct them. It's intimidating and difficult to implement all the tests for the language at once. A much better plan is to get the testing framework in place early and then add tests as the project proceeds to test out the different things that come up or bugs that are caught in other ways.

Appendix A – Testing Suite

SpriteTest.java – Testing Framework

```
package test;

import heterogeneousAST.ASTBuildingException;

import java.io.BufferedReader;
import java.io.ByteArrayInputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;

import org.xml.sax.SAXException;

import ssa.SSAException;
import antlr.RecognitionException;
import antlr.TokenStreamException;

import com.meterware.httpunit.HTMLElement;
import com.meterware.httpunit.WebConversation;
import com.meterware.httpunit.WebResponse;
import com.meterware.pseudoserver.PseudoServer;
import compiler.SpriteCompiler;

public class SpriteTest {
	PseudoServer server;
	String url;
	static int realTotalTests;
	static int totalPassedTests;
	private int totalTests; // per instance / file
	private int passedTests; // per instance / file
	static String testFile = null;
	final static String[] testFiles = { "test/files/commentswhite.txt",
			"test/files/primidentifiers.txt", "test/files/operators.txt",
			"test/files/conditionals.txt", "test/files/objects.txt",
			"test/files/arrays.txt", "test/files/loops.txt",
			"test/files/functions.txt", "test/files/scoping.txt" };
	private int lineNumber;
	private boolean justFailed = true;

	public static void main(String[] params) {
		realTotalTests = 0;
		totalPassedTests = 0;
		for (String fileName : testFiles) {
			testFile = fileName;
			try {
				SpriteTest tester = new SpriteTest();
				tester.runAllTests();
				tester.printSummary();
				realTotalTests += tester.totalTests;
				totalPassedTests += tester.passedTests;
			} catch (IOException e1) {
				e1.printStackTrace();
				return;
			}
		}
		System.out.println("Total: " + totalPassedTests + "/" + realTotalTests + " ("
				+ totalPassedTests * 100 / realTotalTests + "%) " + "tests passed.");
	}

	public void printSummary() {
		System.out.println("\n" + passedTests + "/" + totalTests + " ("
				+ passedTests * 100 / totalTests + "%) " + "tests passed.\n");
	}
```

```
private void runAllTests() throws IOException {

        String code = "";
        String correctResult;
        BufferedReader reader = new BufferedReader(new FileReader(new File(
                        testFile)));
        String line;
        lineNumber = 1;
        while ((line = reader.readLine()) != null) {
                if (line.equals(">>>")) {
                        code = "";
                } else if (line.length() > 3 && line.substring(0, 3).equals(">>>")) {
                        correctResult = line.substring(3).trim();
                        runTest(code, correctResult);
                } else {
                        code += line + "\n";
                }
                lineNumber++;
        }
        reader.close();
        server.shutDown();
}

public SpriteTest() throws IOException {
        System.out.println("File: " + testFile);
        server = new PseudoServer();
        totalTests = 0;
        passedTests = 0;
        url = "http://localhost:" + server.getConnectedPort();
        System.out.flush();
}

/**
 * @param correctResult
 *            "OK" to indicate the code must compile, "BAD" to indicate is
 *            should not, or anything else as a string which is the value
 *            that the execution of this code should produce.
 */
private void runTest(String spriteCode, String correctResult) {
        String js = "";
        Boolean compiled = false;
        String errorMessage = "";
        totalTests++;
        try {
                js = compile(spriteCode);
                compiled = true;
        } catch (RecognitionException e) {
                errorMessage = e.getMessage() + "\n" + e.getStackTrace();
        } catch (TokenStreamException e) {
                errorMessage = e.getMessage() + "\n" + e.getStackTrace();
        } catch (ASTBuildingException e) {
                errorMessage = e.getMessage() + "\n" + e.getStackTrace();
        } catch (SSAException e) {
                errorMessage = e.getMessage() + "\n" + e.getStackTrace();
        } catch (Exception e) {
                error();
                printCode(spriteCode);
                printException(e);
                return;
        }
        if (correctResult.equals("OK")) {
                if (compiled) {
                        passedTests++;
                        if (justFailed)
                                System.out.print("PASSED");
                        System.out.print(".");
                        justFailed = false;
                        return;
                } else {
                        failed("Should have compiled but didn't.");
                        printCode(spriteCode);
                        printError(errorMessage);
                        return;
                }
```

```
        } else if (correctResult.equals("BAD")) {
                if (!compiled) {
                        passedTests++;
                        if (justFailed)
                                System.out.print("PASSED");
                        System.out.print(".");
                        justFailed = false;
                        return;
                } else {
                        failed("Incorrect code compiled.");
                        printCode(spriteCode);
                        return;
                }
        } // else we have an expected execution result and need to keep going

        String result = "";
        try {
                result = execute(js);
        } catch (Exception e) {
                error();
                printCode(spriteCode);
                printException(e);
                return;
        }
        if (result.equals(correctResult)) {
                passedTests++;
                if (justFailed)
                        System.out.print("PASSED");
                System.out.print(".");
                justFailed = false;
        } else {
                failed();
                printCode(spriteCode);
                System.out.println("        Execution produced '" + result + "'");
        }
}

private void failed() {
        failed("");
}

private void failed(String message) {
        justFailed = true;
        System.out.println("FAILED. Line " + lineNumber + ". " + message);
}

private void error() {
        error("");
}

private void error(String message) {
        System.out.println("ERROR. Line " + lineNumber + ". " + message);
}

private void printException(Exception e) {
        System.out.println("        " + e);
        for (StackTraceElement line : e.getStackTrace()) {
                System.out.println("        " + line);
        }
}

private void printError(String errorMessage) {
        System.out.println("        Error message:");
        String[] message = errorMessage.split("\n");
        for (String line : message) {
                System.out.println("        " + line);
        }
}

private void printCode(String spriteCode) {
        System.out.println("        Code:");
        String[] code = spriteCode.split("\n");
        for (String line : code) {
                System.out.println("        " + line);
```

```
                }
        }

        private String compile(String code) throws RecognitionException,
                        TokenStreamException, IOException, ASTBuildingException,
                        SSAException {
                SpriteCompiler compiler = new SpriteCompiler();
                InputStream inputStream;
                DataInputStream dataInputStream;

                inputStream = new ByteArrayInputStream(code.getBytes("UTF-8"));
                dataInputStream = new DataInputStream(inputStream);
                compiler.doFrontend(dataInputStream, "1.sprite");
                compiler.doSecondPassAnalysis();
                return compiler.compileToJavascript();
        }

        private String execute(String js) throws MalformedURLException,
                        IOException, SAXException, BadJSExecutionException {
                // create the conversation object which will maintain state for us
                WebConversation wc = new WebConversation();
                WebResponse response;
                HTMLElement element;

                server.setResource("/1", js);
                response = wc.getResponse(url + "/1");
                element = response.getElementWithID("body");
                if (element == null) {
                        throw new BadJSExecutionException(response.getText());
                }
                return element.getText();
        }
}
```

Comments and Whitespace

```
>>>
<< I am a comment.
>>> OK

>>>
x = 5 << Comment
>>> OK

>>>
<< 2 Lines of comment
<< wow
>>> OK

>>>
<< Comment << inside a comment
>>> OK

>>>
<< comment



<< another comment
>>> OK

>>>
                          << some white space and a comment
>>> OK
```

Variables with Primitive Data Types

```
>>>
x = 2
>>> OK

>>>
Foo = 2
>>> OK

>>>
_bar = 2
>>> OK

>>>
_____ = 5
>>> OK

>>>
Dog5 = 2
>>> OK

>>>
a = 5
a = a
>>> OK

>>>
a = a
>>> BAD

>>>
mod = 4
>>> BAD

>>>
2two = 2
>>> BAD

>>>
*pie = 2
>>> BAD

>>>
x == 2
>>> BAD

>>>
~ = 2
>>> BAD

>>>
x = ""
>>> OK

>>>
x = "hello world!"
>>> OK

>>>
x = "x & food -123123123 !@#$%^&*()~_+{}:<>?"
>>> OK

>>>
x = "x aldkfjasldkfj askldfjsad"dsfkjlsadkfjlkadsjfa"
>>> BAD

>>>
x = "dfsdfsdf

sdfsdfsdf"
>>> BAD
```

```
>>>
x = 5
>>> OK

>>>
x = -2
>>> OK

>>>
x = 0
>>> OK

>>>
x = -0
>>> OK

>>>
x = 109372193762139871871982l324
>>> BAD

>>>
x = -2147483647
>>> OK

>>>
x = -2147483648
>>> BAD

>>>
x = 2147483647
>>> OK

>>>
x = 2147483648
>>> BAD

>>>
x = 0.0
>>> BAD

>>>
x = .23432
>>> BAD

>>>
x = 2432.2134
>>> BAD

>>>
x = 1.
>>> BAD

>>>
x = -.1232
>>> BAD

>>>
5 = x
>>> BAD

>>>
x = 1.4.6
>>> BAD

>>>
x = 1.a
>>> BAD

>>>
x = 4.*!
>>> BAD

>>>
```

```
x = -777h
>>> BAD

>>>
x = true
>>> OK

>>>
x = false
>>> OK

**this is ok but we don't want to be able to do this....
>>>
x = 4 + false
>>> OK

>>>
x = (false || true)
>>> OK

>>>
a = beta
print(beta)
>>> BAD

>>>
a = beta + alpha
>>> BAD

>>>
a = beta
>>> BAD

>>>
a = "string"
print(a)
>>> string

>>>
a = true
b = !!a
print(b)
>>> true

>>>
a = false
if(!a){
        print(5)
} else {
        print(1)
}
>>> 5

>>>
print("String",2)
>>> BAD
```

Arithmetic Operations

```
>>>
x = 5
print(x)
x = "hello world"
print(x + 5)
>>> OK

>>>
x = 3 + 5
>>> OK

>>>
a = "hello"
b = "world"
c = a + " " + b
>>> OK

>>>
a = 5+6 = 7-4
>>> BAD

>>>
foo = -3 - -5
print(foo)
>>> 2

>>>
bar = 5 / 2
>>> OK

>>>
f = 5 * 8
>>> OK

>>>
x = 4 - - - 5
print(x)
>>> -1

>>>
x = 4 - --5
print(x)
>>> -1

>>>
x = 4 + .3
>>> BAD

>>>
x = 5 mod 3
>>> OK

>>>
a = 999
b = -13
c = a - b
>>> OK
```

For Loops

```
>>>
for(i = 5; i < 10; i = i + 1) {
        local x = 7
}
>>> BAD

>>>
for(i; i < 10; i = i + 1) {
        print(i)
}
>>> BAD

>>>
for( i = 5; i < 10; i = i + 1) {
        ret 5
}
>>> BAD

>>>
for(i = 0; i < 10; i = i + 1) {
        for(j = 0; j < 10; j = j + 1) {
                print(i + j)
        }
}
>>> OK

>>>
func forloop() {
        a = 100
        for(i = 0; i < 100; i = i + 1) {
                a = a - 1
                local x = a
                print(x)
        }
}
>>> OK

>>>
a = 8
for(a; a < 10; a = a + 1){
}
>>> BAD

>>>
for(a = -1; a > -10; a = a-1){
}
>>> OK

>>>
func myfunc() {
}
func myfunc() {
}
>>> BAD

>>>
myfunc()
func myfunc() {
}
>>> OK

>>>
func myfunc() {
        func myEmbeddedFunc() {
        }
}
>>> BAD

>>>
myfunc(3)
```

```
func myfunc() {
}
>>> BAD

>>>
myfunc()
func myfunc(a, b) {
}
>>> BAD

** a is local to the function because it is an argument
>>>
func myfunc(a, b) {
        print(b)
}
print(a)
>>> BAD

>>>
x = 5
func foo(y) {
        << x is a global variable while y is a local variable
        number = x + y
        print("number = " + number + " and y = " + y)

        << newNumber is a local variable as well
        local newNumber = number + y
        print("new number = " + newNumber)
}
>>> OK

>>>
x = 5
func foo(y) {
        << x is a global variable while y is a local variable
        number = x + y
        print("number = " + number + " and y = " + y)

        << newNumber is a local variable as well
        local newNumber = number + y
        print("new number = " + newNumber)
}
print(x)
print(y)
>>> BAD
```

Conditional Statements

```
>>>
a = 5
b = 2
if(a == b){
} elseif (a > b) {
} else {
}
>>> OK

>>>
a = 5
b = 2
if(a < b) {
} elseif(a == b) {
}
>>> OK

>>>
a = true
b = false
c = true
d = true
if( (a || b) && (d || c) ){
        print("true")
}
>>> OK

>>>
a = true
b = false
c = true
d = true
if( (a || b) && (d || c) ){
        print(true)
}
>>> OK

>>>
a = true
b = false
c = true
d = true
if( (a || b) && (d || c) ){
        ret true
}
>>> BAD

>>>
a = 1
b = 54
if(a < b) {
        << body of if statement
}
else {
        << body of else statement
}
>>> BAD

>>>
a = 8
b = 3
if(a - b == 0){
        << body of if statement
}
>>> OK

>>>
a = false
if(a){
} elseif(b){
```

```
}
>>> BAD

>>>
if(false){
} elseif (true) {
}
>>> OK

>>>
a = 8
b = 3
if((a - b) == 0){
        << body of if statement
}
>>> OK

>>>
a = false
if(a || b) {
        print(a)
}
>>> BAD

>>>
a = true
if(a) {
        print(a)
}
>>> true

>>>
a = -19
b = 100
if(a-b == 0){
} else {
        print(a-b)
}
>>> -119

>>>
a = false
b = true
if(a || b) {
        else ret -1
}
>>> BAD

>>>
a = false
if(a){
        goodbye = 1
}
else {
        hello = 5
}
>>> BAD

>>>
if(a){
        b = 5
}
else{
        b = new array
}
>>> BAD

>>>
x = 5
if (x == 6){
        ret false
}else {
        ret true
}
```

```
>>> BAD

>>>
if(false){
}
else {
}
>>> BAD

>>>
if(false){
}
else {
        ret false
}
>>> BAD

>>>
else {
        x = 1
}
>>> BAD

>>>

else {
        x = 1
}
>>> BAD
```

Arrays

```
>>>
x = new array
>>> OK

>>>
x = new array 4
>>> BAD

>>>
x = new new
>>> BAD

>>>
x = new 5
>>> BAD

>>>
x = array
>>> BAD

>>>
array = 5
>>> BAD

>>>
x = new
>>> BAD

>>>
array x
>>> BAD

>>>
array = new array
>>> BAD

>>>
x = new array
add(x, 5)
>>> OK

>>>
x = new array
add(x, .55)
>>> BAD

>>>
x = new array
add(x, 5.2)
>>> BAD

>>>
x = new array
add(x, "foo")
>>> OK

>>>
bar = 3
x = new array
add(x, bar)
>>> OK

>>>
x = new array
add(x, 5)
count(x)
>>> OK


we realized you can't always check the type of the argument, so
```

```
this case would be correct, but produce a runtime error
>>>
x = 5
count(x)
>>> OK

>>>
add(x, 4)
>>> BAD

>>>
add(x = new array, 4)
>>> BAD

this will compile and run on our compiler---but will produce a runtime error
>>>
a = new array
add(a, 1)
b = a[6]
>>> OK

>>>
add(a)
>>> BAD

>>>
add(a, b, c)
>>> BAD

>>>
a[8] = 5
>>> BAD

>>>
a = new array
add(a, b)
>>> BAD

>>>
a = true
b = false
if(a || b) {
         c = new array
} else {
         c = 5
}
print(count(c))
>>> OK
```

Objects

```
>>>
x = new obj
>>> BAD

>>>
obj foo
{
        bar = 1
        rab = true
        arb = "Hello world"
}
>>> OK

>>>
obj foo
{
        bar = 1
        rab = true
        arb = "Hello world"
}
>>> OK

>>>
obj foo
{
        bar = 1
        rab = true
        arb
}
>>> BAD

>>>
obj foo
{
        bar = 1
        rab == 0
}
>>> BAD

>>>
obj person {
        firstName = "Jane"
        lastName = "Doe"
        age = 23
        birthday = "February 16th"
        id = 0
}
jane = new person
jane.id = 1
func marriage(fiance, newLastName){
        fiance.lastName = newLastName
}
marriage(jane, "Smith")
>>> OK

>>>
obj foo
{
}
>>> OK

>>>
obj foo
{ a = 2
}
>>> BAD

>>>
obj foo {
a = 2 = 6
```

```
}
>>> BAD

>>>
obj foo { a = 2 }
>>> BAD

>>>
obj foo { a = 2
}
>>> BAD

>>>
obj foo {
    a = 2
}
>>> OK

>>>
func foo() {
}
foo = 5
obj foo {
}
>>> BAD

>>>
a = new fooObj
>>> BAD

>>>
obj foo {
        bar = 1
}
obj foo {
        far = 2
}
>>> BAD

>>>
obj foo {
        for(i = 0; i < 10; i = i + 1) {
                print(i)
        }
}
>>> BAD

>>>
obj foo {
        func bar(x){
                print(x)
        }
}
>>> BAD

>>>
func bar(x){
        print(x)
}
obj bar {
        x = 10
        myFunc = bar(x)
}
>>> BAD

>>>
obj foo {
```

```
        a = 5


}
>>> OK

>>>
obj Y {
        z = 0
}
obj X {
        y = new Y
}
obj W {
        x = new X
}
w = new W
w.x.y.z = 3
>>> OK

>>>
person.name = "sarah"
>>> BAD

>>>
person = new people
person.name = "sarah"
>>> BAD

>>>
obj foo{
}
a = new array
a = new foo
>>> OK

>>>
obj foo{
        obj bar {
        }
}
>>> BAD

>>>
obj foo{
        foo = "test"
}
>>> BAD
```

Functions

```
>>>
func count(){
        ret 5
}
>>> BAD

>>>
func funcname() {
        ret x
}
>>> BAD

** doesn't see that the parent-parent node of b is a return node
>>>
a = 5
func funcname() {
        ret a + b
}
>>> BAD

>>>
b = 5
func funcname() {
        ret a + b
}
>>> BAD

>>>
b = 5
func funcname() {
        c = a + b
        ret c
}
>>> BAD

>>>
b = a
func returnb() {
        ret b
}
>>> BAD

>>>
a = 6
b = 5
func one() {
        ret a
}
func two() {
        ret b
}
print(one())
>>> 6

>>>
func funcname()
{
}
>>> OK

>>>
func funcname(arg) {
}
>>> OK

>>>
func funcname(arg1, arg2, arg3) {
}
>>> OK
```

```
>>>
func funcname(arg1[2]) {
}
>>> BAD

>>>
func funcname(id.field) {
}
>>> BAD

>>>
func funcname(345) {
}
>>> BAD

>>>
func compare(a,b){
        if(a == b) {
                ret true
        } else {
                ret false
        }
}
>>> OK

>>>
func decrementOne(bar){
        bar = bar - 1
        ret bar
}
>>> OK

>>>
func decrease(bar) callevery(10) {
        bar = bar - 1
        print(bar)
}
>>> OK

>>>
func funcname() callevery(10.444){
        ret 4
}
>>> BAD

>>>
func funcname(arg1, arg2, arg3) callevery(4) {
}
>>> OK

>>>
func a() onkeyboard("a") {
}
>>> OK

>>>
func a() onkeyboard(a) {
}
>>> OK

>>>
func a() onkeyboard(b) {
}
>>> BAD

>>>
obj a {
}
func b () onkeyboard(a) {
}
>>> BAD

>>>
foo = "a"
```

```
func funcname() onkeyboard(foo) {
}
>>> OK

>>>
foo = 4
func funcname() onkeyboard(foo){
}
>>> OK

>>>
func funcname() onkeyboard(%){
}
>>> BAD

>>>
func funcname() onkeyboard(4) {
}
>>> OK

>>>
func funcname(arg1) callevery(-10) {
}
>>> BAD

>>>
id = 10000
func funcname(arg1, arg2, arg3) callevery(id) {
}
>>> OK

>>>
obj newObj{
         field = 0
}
myobj = new newObj
myobj.field = 4
func funcname(arg1, arg2, arg3) callevery(myobj.field) {
}
>>> OK

>>>
obj newObj{
         field = 0
}
myobj = new nonExistentObject
myobj.field = 4
func funcname(arg1, arg2, arg3) callevery(myobj.field) {
}
>>> BAD


>>>
b = 1
a = new array
add(a, 5)
add(a, 10000)
func funcname(arg1, arg2, arg3) callevery(a[b]) {
}
>>> OK

>>>
a = new array
add(a, 1)
add(a, 2)
add(a, 3)
add(a, 4)
func funcname(arg1, arg2, arg3) callevery(a[3]) {
}
>>> OK

>>>
func callevery(10) {
```

```
}
>>> BAD

>>>
func onkeyboard(b){
}
>>> BAD

>>>
func true() {
}
>>> BAD

>>>
func funcname() {
        x = 5
}
>>> OK

>>>
func funcname() {
        local x = 5
}
>>> OK

>>>
func funcname(arg1) callevery() {
        ret 5
}
>>> BAD

>>>
func funcname() onkeyboard() {
        print(5)
}
>>> BAD

>>>
func funcname() {
        local x
}
>>> BAD

** decide whether or not this is allowed
>>>
func foo(){
        local foo = 5
        ret foo
}
>>> OK

>>>
func funcname() {
        local x = 6
        ret x
}
>>> OK

>>>
func func1(){
        local x = 6
}
func func2(){
        ret x
}
>>> BAD

>>>
func funcname() {
        x = 5
        ret x
}
>>> OK
```

```
>>>
func funcname() {
        ret 6
}
>>> OK

>>>
b = 1
a = new array
add(a, 1)
add(a, 2)
func funcname() {
        ret a[b]
}
>>> OK

>>>
a = new array
add(a, 3)
func funcname() {
        ret a[0]
}
>>> OK

>>>
obj blah{
        field = 5
}
myobj = new blah
func funcname() {
        ret myobj.field
}
>>> OK

# can't return stuff anywhere
>>>
ret x
>>> BAD

>>>
a = 1
b = 2
c = 3
d = 4
sprite(a, b, c, d)
>>> OK

>>>
sprite(1, 2, 3, 4, "too many arguments")
>>> BAD

>>>
sprite("ball.png", object.field, myarray[i], myfunc())
>>> BAD

>>>
b()
func b() {
}
>>> OK

>>>
a(1 + 2)
func a(b) {
}
>>> OK

>>>
func a(b) {
        a(b)
}
>>> OK

>>>
```

```
func a(b) {
        if (b > 0) {
        }
}
>>> OK

>>>
func a(b) {
        local x
}
>>> BAD

>>>
func a(b) {
        local x = 5
}
print(x)
>>> BAD

>>>
func a(b) {
        local x = 5
        print(x)
}
>>> OK

>>>
func a(b) {
        local x = 5
}
>>> OK

>>>
func a(b) {
        local x = 5
        b = x
        print(b)
}
>>> OK

>>>
func a(b) {
        local x = 5
        c = b + x
}
>>> OK

>>>
func a(b) {
        local x = 5
        print(b + x)
}
>>> OK

>>>
func a(b) {
        local x = 5
        c = b + x
        print(c)
}
>>> OK

>>>
func a(b) {
        local x = 5
}
print(x)
>>> BAD

>>>
func a(b) {
        x = 5
        print(x)
}
```

```
>>> OK

>>>
func a(b) {
        x = 5
}
print(x)
>>> OK

>>>
a = isEven(23)
print(a)
func isEven(num){
        if(num mod 2 == 0) {
                ret true
        } else {
                ret false
        }
}
>>> OK

>>>
x = 0
func one(){
        x = x + 1
}
func two(){
        x = 2
}
two()
one()
print(x)
>>> 3

>>>
x = 5
func myFunc(x) {
        ret x + 5
}
print(myFunc(x))
>>> 10

>>>
func myFunc(x) {
        ret x + 5
}
print(myFunc(x))
>>> BAD

>>>
x = 2
func foo(){
        x = x * 2
}
func bar(){
        x = x + 2
}
func call(a, b){
        print(x)
}
call(foo(),bar())
>>> 6

>>>
func foo() {
        obj ball{
                num = 5
        }
}
>>> BAD

>>>
func foo() {
        func bar() {
```

```
        }
}
>>> BAD

>>>
func zero(x) {
        if (x <= 0) {
                print(0)
        } else {
                zero(x - 1)
        }
}
zero(4)
>>> 0
```

Scoping

```
>>>
func f() {
        local x = 1
}
>>> OK

>>>
func f(x) {
        local x = 1
}
>>> BAD

>>>
func f(x) {
        x = 1
}
>>> OK

>>>
func f(x) {
        x = 1
}
print(x)
>>> BAD

>>>
func f() {
        local x = 1
        local x = 2
}
>>> BAD

>>>
func f() {
        local x = 1
        x = 2
}
>>> OK

>>>
func f() {
        local x = 1
}
print(x)
>>> BAD

>>>
for (i = 0; i < 3; i = i + 1) {
        local x = 1
}
>>> BAD

>>>
obj foo {
        local bar = 3
        x = 5
}
>>> BAD

>>>
func f() {
        local x = 5
        ret x
}
func g() {
        local x = 1
        ret x
}
f()
print(g())
```

```
>>> 1

>>>
x = 5
func foo(y) {
        << x is a global variable while y is a local variable
        number = x + y
        print("number = " + number + " and y = " + y)

        << newNumber is a local variable as well
        local newNumber = number + y
        print("new number = " + newNumber)
}
>>> OK

>>>
x = 0
func f() {
        local x = 1
        print(x)
}
f()
>>> 1

>>>
x = 0
func f() {
        local x = 1
}
print(x)
>>> 0

>>>
x = 0
func f() {
        local x = 1
        x = x + 1
}
f()
print(x)
>>> 0

>>>
func f(x) {
        x = 1
}
a = 5
f(a)
print(a)
>>> 5

>>>
func f(a) {
        a = 1
}
a = 5
f(a)
print(a)
>>> 5

>>>
func f() {
        a = 1
}
a = 5
f()
print(a)
>>> 1

>>>
func f(x) {
        ret (x + 1)
}
a = 5
```

```
a = f(a)
print(a)
>>> 6

>>>
x = 2
for (i = 0; i < 3; i = i + 1) {
        local x = 1
        print(x)
}
>>> BAD

>>>
x = 2
for (i = 0; i < 3; i = i + 1) {
        local x = 1
}
print(x)
>>> BAD

>>>
x = 20
func f() {
        local x = 1
}
func g() {
        x = 1
}
print(x)
>>> 20

>>>
x = 20
func f() {
        local x = 1
}
func g() {
        x = 1
}
g()
print(x)
>>> 1

>>>
x = 5
func foo(y) {
        number = x + y
        print(number)
}
foo(2)
>>> 7

>>>
x = 5
func foo(y) {
        number = x + y
}
foo(2)
print(number)
>>> 7

>>>
x = 5
func foo(y) {
        number = x + y
        ret number
}
print(foo(2))
>>> 7

>>>
x = 1
func f() {
        local x = 2
```

```
}
>>> OK

>>>
x = 1
func f() {
        local x = 2
        print(x)
}
f()
>>> 2

>>>
x = 1

func two(){
        x = 2
}
func three() {
        x = 3
}
two()
three()
print(x)
>>> 3

>>>
x = 1
func f() {
        local x = 2
}
f()
print(x)
>>> 1

>>>
a = 1
b = 2
func one() {
        local b = a + 5
        ret b
}
print(one())
>>> 6

>>>
local x = 3
>>> BAD

>>>
x = 0
func one(){
        local x = x + 1
}
func two(){
        x = 2
}
two()
one()
print(x)
>>> 2

>>>
obj foo {
}
func foo {
}
>>> BAD

>>>
func myFunc(){
        local foo = 4
}
obj foo {
```

```
}
>>> OK

>>>
func foo() {
}
func f(){
        local foo = 5
}
>>> OK

>>>
obj foo {
}
foo = new foo
>>> BAD

>>>
obj bar{
        foo = 5
}
func foo(){
        bar = 1
}
>>> BAD

>>>
func foo() {
}
foo = 5
>>> BAD

>>>
func foo() {
        foo = 5
}
>>> BAD

>>>
func foo() {
        local foo = 5
}
>>> OK

** not in the same scope
>>>
func foo(foo) {
        ret foo
}
>>> OK

>>>
obj a {
}
func b() {
        c = new a
}
>>> OK

>>>
obj foo{
}
a = new foo
func bar() {
        a = 5
}
bar()
print(a)
>>> 5

>>>
a = "hello"
func bar() {
        local a = 5
```

```
}
bar()
print(a)
>>> hello

>>>
func b() {
        c = new a
}
obj a {
}
>>> OK

>>>
func foo(){
        local a = new array
        add(a,5)
}
foo()
print(a[0])
>>> BAD

>>>
obj a {
}
func a() {
}
>>> BAD
```

Appendix B – Example Sprite Programs

BallAndNinjaGame

Sprite File

```
<< In this program, you control a ninja as he runs around the screen. A ball also
<< bounces around. If the ball hits the ninja, the ninja stays squished for a little
<< while before he gets up and starts running again.

obj Ball {
        left = 0
        top = 0
        downDirection = 6
        rightDirection = 6
        id = 0
        sprite = "Ball.png"
}

obj Ninja {
        left = 0
        top = 100
        downDirection = 0
        rightDirection = 3
        formerDown = 0
        formerRight = 0
        id = 0
        sprite = "NinjaRight1.png"
        imageIndex = 1
        squashedMoments = 0
}

ball = new Ball
ball.id = 1
ninja = new Ninja
ninja.id = 2

stuff = new Array
add(stuff, ninja)
add(stuff, ball)

score = 0

func Left() onkeyboard("a") {
        UpdateNinjaDirection(-6, 0)
}

func Up() onkeyboard("w") {
        UpdateNinjaDirection(0, -6)
}

func Right() onkeyboard("d") {
        UpdateNinjaDirection(6, 0)
}

func Down() onkeyboard("s") {
        UpdateNinjaDirection(0, 6)
}

func UpdateNinjaDirection(right, down) {
        if (ninja.squashedMoments <= 0) {
                ninja.downDirection = down
                ninja.rightDirection = right
        }
}

print("Left - a, Up - w, Right - d, Down - s. Don't get squished!")
stillPlaying = 1
```

```
func MoveStuff() callevery(10) {
	if (1 == stillPlaying) {
		for (local i = 0; i < count(stuff); i = i + 1) {
			local nextThing = stuff[i]
			nextThing.top = nextThing.top + nextThing.downDirection
			nextThing.left = nextThing.left + nextThing.rightDirection
			if (nextThing.top <= 0 || nextThing.top >= 400) {
				nextThing.downDirection = - 1 * nextThing.downDirection
			}
			if (nextThing.left <= 0 || nextThing.left >= 700) {
				nextThing.rightDirection = - 1 * nextThing.rightDirection
			}
			sprite(nextThing.sprite, nextThing.id, nextThing.top, nextThing.left)
		}
	} else {
		sprite("YouLose.jpg", 3, 20, 20)
	}
}

func UpdateNinja() callevery(100) {
	if (Distance(ninja.left, ninja.top, ball.left, ball.top) < 100) {
		ninja.squashedMoments = 10
		ninja.sprite = "SquishedNinja.png"
		if (ninja.rightDirection != 0 || ninja.downDirection != 0) {
			score = score + 1
			print("Score: Ball - " + score + ", Ninja - 0")
			if (score >= 5) {
				stillPlaying = 0
			}
			ninja.formerRight = ninja.rightDirection
			ninja.rightDirection = 0
			ninja.formerDown = ninja.downDirection
			ninja.downDirection = 0
		}
	} elseif (ninja.squashedMoments > 0) {
		ninja.squashedMoments = ninja.squashedMoments - 1
		if (ninja.squashedMoments == 0) {
			ninja.rightDirection = ninja.formerRight
			ninja.downDirection = ninja.formerDown
		}
	} else {
		nextSprite = "Ninja"
		if (ninja.rightDirection > 0) {
			nextSprite = nextSprite + "Right"
		} else {
			nextSprite = nextSprite + "Left"
		}
		ninja.imageIndex = ninja.imageIndex mod 3 + 1
		nextSprite = nextSprite + ninja.imageIndex + ".png"
		ninja.sprite = nextSprite
	}
}

func Distance(left1, top1, left2, top2) {
	local totalDistance = 0
	if (left1 > left2) {
		totalDistance = totalDistance + left1 - left2
	} else {
		totalDistance = totalDistance + left2 - left1
	}
	if (top1 > top2) {
		totalDistance = totalDistance + top1 - top2
	} else {
		totalDistance = totalDistance + top2 - top1
	}
	ret totalDistance
}
```

HTML Output File

```
<html>
```

```
<head>
	<title>test/BallAndNinjaGame.sprite</title>
</head>
<body id="body" onload="MoveStuffTiming();UpdateNinjaTiming();">
	<div id="DivSprites" />
	<div id="DivPrints" />
	<script>
		function Ball() {
			this.left = 0;
			this.top = 0;
			this.downDirection = 6;
			this.rightDirection = 6;
			this.id = 0;
			this.sprite = "Ball.png";
		}

		function Ninja() {
			this.left = 0;
			this.top = 100;
			this.downDirection = 0;
			this.rightDirection = 3;
			this.formerDown = 0;
			this.formerRight = 0;
			this.id = 0;
			this.sprite = "NinjaRight1.png";
			this.imageIndex = 1;
			this.squashedMoments = 0;
		}

		ball = new Ball();
		ball.id = 1;
		ninja = new Ninja();
		ninja.id = 2;
		stuff = new Array();
		add(stuff, ninja);
		add(stuff, ball);
		score = 0;
		function Left () {
			UpdateNinjaDirection(-6, 0);
		}
		function Up () {
			UpdateNinjaDirection(0, -6);
		}
		function Right () {
			UpdateNinjaDirection(6, 0);
		}
		function Down () {
			UpdateNinjaDirection(0, 6);
		}
		function UpdateNinjaDirection (right, down) {
			if (ninja.squashedMoments <= 0) {
				ninja.downDirection = down;
				ninja.rightDirection = right;
			}

		}
		print("Left - a, Up - w, Right - d, Down - s. Don't get squished!");
		stillPlaying = 1;
		function MoveStuff () {
			if (1 == stillPlaying) {
				for (i = 0; i < count(stuff); i = i + 1) {
					nextThing = stuff[i];
					nextThing.top = nextThing.top +
					nextThing.downDirection;
					nextThing.left = nextThing.left +
					nextThing.rightDirection;
					if (nextThing.top <= 0 || nextThing.top >= 400) {
						nextThing.downDirection = -1 *
						nextThing.downDirection;
					}

					if (nextThing.left <= 0 || nextThing.left >= 700) {
						nextThing.rightDirection = -1 *
						nextThing.rightDirection;
```

```
                        }

                        sprite(nextThing.sprite, nextThing.id,
                        nextThing.top, nextThing.left);
                }
        }
        else {
                sprite("YouLose.jpg", 3, 20, 20);
        }

}
function MoveStuffTiming() {
        MoveStuff();
        window.setTimeout("MoveStuffTiming()", 10);
}
function UpdateNinja () {
        if (Distance(ninja.left, ninja.top, ball.left, ball.top) < 100) {
                ninja.squashedMoments = 10;
                ninja.sprite = "SquishedNinja.png";
                if (ninja.rightDirection != 0 ||
                        ninja.downDirection != 0) {
                        score = score + 1;
                        print("Score: Ball - " + score + ", Ninja - 0");
                        if (score >= 5) {
                                stillPlaying = 0;
                        }

                        ninja.formerRight = ninja.rightDirection;
                        ninja.rightDirection = 0;
                        ninja.formerDown = ninja.downDirection;
                        ninja.downDirection = 0;
                }

        }
        else if (ninja.squashedMoments > 0) {
                ninja.squashedMoments = ninja.squashedMoments - 1;
                if (ninja.squashedMoments == 0) {
                        ninja.rightDirection = ninja.formerRight;
                        ninja.downDirection = ninja.formerDown;
                }

        }
        else {
                nextSprite = "Ninja";
                if (ninja.rightDirection > 0) {
                        nextSprite = nextSprite + "Right";
                }
                else {
                        nextSprite = nextSprite + "Left";
                }

                ninja.imageIndex = ninja.imageIndex % 3 + 1;
                nextSprite = nextSprite + ninja.imageIndex + ".png";
                ninja.sprite = nextSprite;
        }

}
function UpdateNinjaTiming() {
        UpdateNinja();
        window.setTimeout("UpdateNinjaTiming()", 100);
}
function Distance (left1, top1, left2, top2) {
        totalDistance = 0;
        if (left1 > left2) {
                totalDistance = totalDistance + left1 - left2;
        }
        else {
                totalDistance = totalDistance + left2 - left1;
        }

        if (top1 > top2) {
                totalDistance = totalDistance + top1 - top2;
        }
        else {
```

```
                                totalDistance = totalDistance + top2 - top1;
                        }

                        return totalDistance;
                }

                document.onkeypress = KeyWasPressed;
                function KeyWasPressed(keyStroke) {
                        if(window.event) { // IE
                                keynum = window.event.keyCode;
                        } else if (keyStroke.which) { // Netscape/Firefox/Opera
                                keynum = keyStroke.which;
                        }
                        keychar = String.fromCharCode(keynum);
                        if (keychar == "a") { Left(); }
                        else if (keychar == "w") { Up(); }
                        else if (keychar == "d") { Right(); }
                        else if (keychar == "s") { Down(); }
                }

                sprites = new Array();
                function sprite(image, id, top, left) {
                        foundSprite = false;
                        for (var i = 0; i < sprites.length; i++) {
                                if (sprites[i] == id) {
                                        foundSprite = true;
                                }
                        }
                        if (! foundSprite) {
                                var style = "position: absolute; top:" + top
                                                + ";left:" + left;
                                if (document.createTextNode) {
                                        var div = document.getElementById("DivSprites");
                                        img = document.createElement("img");
                                        img.setAttribute("id", "Sprite" + id);
                                        img.setAttribute("src", image);
                                        img.setAttribute("style", style);
                                        div.appendChild(img);
                                } else {
                                        document.write("<img id=\"Sprite" + id + "\" src=\""
                                + image + "\" style=\"" + style + "\"/>");
                                }
                                var currentCount = sprites.length;
                                sprites[currentCount] = id;
                        }
                        var spriteImage = document.getElementById("Sprite" + id);
                        spriteImage.src = image;
                        spriteImage.style.top = top;
                        spriteImage.style.left = left;
                }

                function print(message) {
                        if (document.createTextNode) {
                                newMessage = document.createTextNode(message);
                                brk = document.createElement("br");
                                document.getElementById("DivPrints").appendChild(newMessage);
                                document.getElementById("DivPrints").appendChild(brk);
                        } else {
                                document.write(message + "<br/>");
                        }
                }

                function add(arr, item) {
                        var length = arr.length;
                        arr[length] = item;
                }

                function count(arr) {
                        return arr.length;
                }
        </script>

    </body>
</html>
```

Appendix C – Source Code

Our team worked really well together at assigning tasks to different individuals. One person was the primary (they essentially create the "rough draft") and then the contributers helped to refine that draft to the finished product.

	Monica	Dan	John	Dave
Documentation	Primary	Contributer	Contributer	Contributer
Backend				Primary
SSA	Contributer		Primary	Contributer
Testing	Contributer	Primary	Contributer	
Grammar	Contributer	Primary	Contributer	Contributer
Environment Setup		Primary		
Example Programs				Primary

sprite.g

Compiler

1. CommandLineSpriteCompiler.java
2. SpriteCompiler.java

HeterogeneousAST

1. ASTBuildingException.java
2. ArrayAccess.java
3. BinaryOperator.java
4. Block.java
5. Boilerplate.java
6. BoilerplateHtml.txt
7. Comparison.java
8. ElseIf.java
9. Equals.java
10. ForLoop.java
11. Func.java
12. FunctionCall.java
13. GroupedExpression.java
14. HeterogeneousTree.java

15. Identifier.java
16. If.java
17. MemberHead.java
18. New.java
19. Node.java
20. NodeStringNode.java
21. Not.java
22. Num.java
23. Obj.java
24. OrAnd.java
25. Return.java
26. Scope.java
27. StatementList.java
28. Str.java
29. TargetLanguage.java

Static Semantic Analysis

1. InvalidArgumentCountException.java
2. InvalidTypesException.java
3. LocalException.java
4. RedefinedException.java
5. SSAErrorMessages.java
6. SSAException.java
7. ScopeException.java
8. Type.java
9. UndefinedFuncException.java
10. UndefinedObjectException.java
11. UndefinedVariableException.java

BadJSExecutionException.java

sprite.g

```
header {
        package frontend;
}

class SpriteParser extends Parser;

options {
        k = 2;
    buildAST = true;
    defaultErrorHandler = false;
}

program
        :       (stmt)+ EOF!;

block
        :       (end)? LBRACE^ end (stmt)* RBRACE! ;

funcblock
        :       (end)? LBRACE^ end (stmt)* RBRACE! ;

objblock
        :       (end)? LBRACE^ end ((assignment)? end)* RBRACE! ;

stmt:   assignment end
        |       "if"^ LPAREN! expression RPAREN! block (elseifdef)* (elsedef)? end
        |       "for"^ LPAREN! assignment SEMI! expression SEMI! assignment RPAREN! block end
        |       "func"^ functiondef ( ("callevery" LPAREN! (NUM | lvalue) RPAREN!) |
                                                        ("onkeyboard" LPAREN! (NUM | string | lvalue) RPAREN!) )?
                                                        funcblock end
        |       "obj"^ ID objblock end
        |       functioncall end
        |       retSt
        |       NEWLINE!
        ;

elseifdef
        :       (end)? "elseif"! LPAREN! expression RPAREN! block ;

elsedef
        :       "else"! block ;

assignment
        :       lvalue ASSIGN^ rvalue ;

rvalue
        :       newobj
        |       argument
        ;

retSt
        :       "ret"^ rvalue end ;

newobj
        :       "new"^ ("array" | ID) ;

argument
        :       expression ;

functiondef
        :       ID^ functiondefend ;
functiondefend
        :       LPAREN! ( ID (COMMA! ID)*)? RPAREN! ;

functioncall
        :       ID^ functioncallend ;
functioncallend
        :       LPAREN^ ( argument (COMMA! argument)*)? RPAREN!;

expression
```

```
        :       join (OR^ join)* ;
join
        :       equality (AND^ equality)* ;
equality
        :       relationship ((EQ^ | NE^) relationship)* ;
relationship
        :       addition ((LT^ | LE^ | GT^ | GE^) addition)* ;
addition
        :       multiplication ((PLUS^ | MINUS^) multiplication)* ;
multiplication
        :       unary ((MUL^ | DIV^ | "mod"^) unary)* ;

unary
        : MINUS^ unary
        | NOT^ unary
        | factor
        ;

factor
        : LPAREN^ expression RPAREN!
        | lvalue
        | NUM
        | "true"
        | "false"
        | STRING
        | functioncall
        ;

lvalue
        : "local"^ ID
        | ID^ ( array | member )?
        ;

array
        : (LBRACK^ expression RBRACK)+ ;
member
        : (DOT^ ID)+ ;

end : (NEWLINE!)
        ;

class SpriteLexer extends Lexer;

options {
        k = 2;
    defaultErrorHandler = false;
}

WHITESPACE
        :       (' ' | '\t')+ { $setType(Token.SKIP); } ;
COMMENT
        :       "<<" (~('\n'|'\r'))* { $setType(Token.SKIP); } ;

protected DIGITS
        :       (DIGIT)+ ;

protected DIGIT
        :       '0'..'9' ;

protected LETTER
        :       'a'..'z' ;

NUM     :       DIGITS ;

AND     :       "&&" ;
LE      :       "<=" ;
SEMI    :       ';' ;
OR      :       "||" ;
GT      :       '>' ;
LPAREN
        :       '(' ;
ASSIGN
        :       '=' ;
GE      :       ">=" ;
```

```
RPAREN
        :       ')' ;
EQ      :       "==" ;
LBRACE
        :       '{' ;
PLUS
        :       '+' ;
NOT     :       '!' ;
RBRACE
        :       '}' ;
MINUS
        :       '-';
NE      :       "!=" ;
LBRACK
        :       '[' ;
MUL     :       '*' ;
LT      :       '<' ;
RBRACK  :       ']' ;
DIV     :       '/' ;

COMMA   :       ',' ;
DOT     :       '.' ;

ID      :       ('_' | 'a'..'z' | 'A'..'Z') ('_' | 'a'..'z' | 'A'..'Z' | '0'..'9')* ;

STRING
        :       '"' (~('"' | '\r' | '\n'))* '"';
NEWLINE
        :       ( "\r\n" | '\r' | '\n' ) { newline(); } ;
```

CommandLineSpriteCompiler.java

```
package compiler;

import java.io.*;
import ssa.SSAException;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import jargs.gnu.CmdLineParser;

class CommandLineSpriteCompiler {
    public static void main(String[] args) {
        // Parse command line args
        CmdLineParser cmdLineParser = new CmdLineParser();
        CmdLineParser.Option printAstOption = cmdLineParser.addBooleanOption('a', "print-ast");
        CmdLineParser.Option graphAstOption = cmdLineParser.addStringOption('g', "graph-ast");
        CmdLineParser.Option ssa = cmdLineParser.addBooleanOption('s', "perform-ssa");
        CmdLineParser.Option help = cmdLineParser.addBooleanOption('h', "help");
        try {
        cmdLineParser.parse(args);
        } catch (CmdLineParser.OptionException error) {
        System.err.println(error.getMessage());
        printUsage();
        System.exit(1);
        }
        if ((Boolean) cmdLineParser.getOptionValue(help, Boolean.FALSE)) {
        printUsage();
        System.exit(0);
        }
        Boolean printAst = (Boolean) cmdLineParser.getOptionValue(printAstOption, Boolean.FALSE);
        Boolean doSSA = (Boolean) cmdLineParser.getOptionValue(ssa, Boolean.TRUE);
        String graphAst = (String) cmdLineParser.getOptionValue(graphAstOption, "");
        String[] otherArgs = cmdLineParser.getRemainingArgs();

        try {
                String fileName;
                DataInputStream dataInputStream;

                // Set up where we're getting the source code from
        if (otherArgs.length >= 1) {
                fileName = otherArgs[0];
                utility.File reader = new utility.File();
                String fileContents = reader.FileContents(fileName) + "\n";
                dataInputStream = new DataInputStream(new ByteArrayInputStream(fileContents.getBytes("UTF-8")));
        } else {
                dataInputStream = new DataInputStream(System.in);
                fileName = "Input Stream.sprite";
        }

        // Compile!
                SpriteCompiler compiler = new SpriteCompiler();
                compiler.doFrontend(dataInputStream, fileName);

                if (printAst) {
                        compiler.printAst();
                }
                if (graphAst != "") {
                        compiler.showHomogeneousAST(graphAst);
                }
                if (doSSA) {
                        compiler.doSecondPassAnalysis();
                }

                // Write out target code
        utility.File writer = new utility.File();
        String root = writer.StripExtension(fileName);
        writer.SetFileContents(root + " Pretty.sprite", compiler.compileToSprite());
                writer.SetFileContents(root + ".html", compiler.compileToJavascript());
        } catch(FileNotFoundException error) {
                System.out.println("Source file " + otherArgs[0] + " not found.");
                return;
       } catch (RecognitionException error) {
```

```
                        System.out.print("Parse error, ");
                        String file;
                        if (error.getFilename() == null) {
                                file = "line ";
                        } else {
                                file = error.getFilename() + ":";
                        }
            String message = file + error.getLine() + ", " + error.getMessage();
            System.out.println(message);
           } catch (TokenStreamException error) {
            String message = error.getMessage();
            System.out.println(message);
                    } catch (SSAException error) {
                            System.err.println("SSA: " + error.getMessage());
                    } catch (Exception error) {
                            System.out.println(error.getMessage());
                            error.printStackTrace();
                    }
            }

        static void printUsage() {
            System.out.println("Usage: CommandLineSpriteCompiler [options] [source file]\n");
            System.out.println("If the source is ommitted, reads from stdin.\n");
            System.out.println("Options:");
            System.out.println("   -a   --print-ast     Print out the ast while compiling.");
            System.out.println("   -g   --graph-ast=<path to dot>    Displays usage.");
            System.out.println("   -h   --help    Displays usage.");
        }
}
```

SpriteCompiler.java

```
package compiler;

import frontend.SpriteLexer;
import frontend.SpriteParser;
import heterogeneousAST.ASTBuildingException;
import heterogeneousAST.HeterogeneousTree;
import heterogeneousAST.Scope;
import heterogeneousAST.TargetLanguage;

import java.io.DataInputStream;
import java.io.IOException;

import ssa.SSAException;
import utility.ASTViewer;
import utility.HomogeneousTreePrinter;
import antlr.CommonAST;
import antlr.RecognitionException;
import antlr.TokenStreamException;

public class SpriteCompiler {
        protected CommonAST homogeneousAst = null; // antlr's ast
        protected HeterogeneousTree heterogeneousAst = null; // our heterogeneous
        // ast
        protected Scope topScope = null;

        public void doFrontend(DataInputStream dataInputStream, String inputFileName)
                        throws RecognitionException, TokenStreamException, IOException,
                        ASTBuildingException, SSAException {
                // Scan and parse source code
                SpriteLexer lexer = new SpriteLexer(dataInputStream);
                SpriteParser parser = new SpriteParser(lexer);
                parser.program();

                // Build our new AST
                utility.File writer = new utility.File();
                String title = writer.FileName(inputFileName);
                homogeneousAst = (antlr.CommonAST) parser.getAST();
                topScope = new Scope();
                heterogeneousAst = new HeterogeneousTree(title, homogeneousAst,
                                topScope);
        }

        /**
         * @param dotProgram
         *            The full path to the dot program that comes with graphviz.
         */
        public void showHomogeneousAST(String dotProgram) {
                if (homogeneousAst == null) {
                        return;
                }
                new ASTViewer().show(homogeneousAst, dotProgram);
        }

        public void printAst() {
                HomogeneousTreePrinter treePrinter = new HomogeneousTreePrinter(
                                homogeneousAst);
                treePrinter.PrintTree();
        }

        public String compileToJavascript() {
                return heterogeneousAst.toString(TargetLanguage.Javascript);
        }

        public String compileToSprite() {
                return heterogeneousAst.toString(TargetLanguage.Sprite);
        }

        public void doSecondPassAnalysis() throws SSAException {
                topScope.SecondPassAnalysis();
```

```
        }
}
```

ArrayAccess.java

```
package heterogeneousAST;

import ssa.InvalidTypesException;
import ssa.SSAException;
import ssa.Type;
import antlr.BaseAST;

public class ArrayAccess extends Node {
        private Identifier thisArrayName;
        private Node thisAccessStatement;

        public ArrayAccess(BaseAST homogeneousNode, Scope scope)
                        throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope);
        }

        public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
                        SSAException {
                thisArrayName = new Identifier(homogeneousNode, getCurrentScope(), this);
                thisAccessStatement = NodeForHomogeneous((BaseAST) homogeneousNode
                                .getFirstChild().getFirstChild());
        }

        public void SecondPassAnalysis() throws SSAException {
                String funcString = thisArrayName.toString();
                if (getCurrentScope().containsName(funcString)) {
                        if (getCurrentScope().getType(funcString).getType() != Type.ARRAY) {
                                throw new InvalidTypesException("variable " + funcString
                                                + " is not an array");
                        }
                }
        }

        public String toString(TargetLanguage language) {
                StringBuilder builder = new StringBuilder();
                builder.append(thisArrayName.toString(language));
                builder.append("[");
                builder.append(thisAccessStatement.toString(language));
                builder.append("]");
                return builder.toString();
        }
}
```

ASTBuildingException.java

```
package heterogeneousAST;

public class ASTBuildingException extends Exception {

	public ASTBuildingException(String message) {
		super(message);
	}

	private static final long serialVersionUID = -2550453469350246881L;

}
```

BinaryOperator.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

/* Binary operators are things like +, -, /, *, etc...
 * Expressions are combinations of binary operators and atoms.
 */
public class BinaryOperator extends NodeStringNode {
        public BinaryOperator(BaseAST homogeneousNode, Scope scope)
                        throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope);
        }
}
```

Block.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

public class Block extends StatementList {
	public Block(BaseAST homogeneousNode, Scope scope)
			throws ASTBuildingException, SSAException {
		super(homogeneousNode, scope);
	}

	public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
			SSAException {
		LoadChildren((BaseAST) homogeneousNode.getFirstChild());
	}

	public String toString(TargetLanguage language) {
		StringBuilder builder = new StringBuilder();
		if (language == TargetLanguage.Sprite) {
			builder.append("\n");
			builder.append(TabsString());
		} else {
			builder.append(" ");
		}
		builder.append("{\n");
		builder.append(ChildString(language, tabs + 1));
		builder.append(TabsString());
		builder.append("}\n");
		return builder.toString();
	}

	public String LineEnding(TargetLanguage language) {
		return "";
	}
}
```

Boilerplate.java

```
package heterogeneousAST;

import java.io.FileNotFoundException;
import java.io.IOException;

/* This class is for loading the boilerplate JavaScript and HTML */
public class Boilerplate {
	public String BoilerplateHtml() {
		utility.File boilerplateFile = new utility.File();
		String returnValue = "";
		try {
			returnValue = boilerplateFile
				.FileContents("bin/heterogeneousAST/BoilerplateHtml.txt");
		} catch (FileNotFoundException e) {
			e.printStackTrace();
			returnValue = e.toString();
		} catch (IOException e) {
			e.printStackTrace();
			returnValue = e.toString();
		}
		return returnValue;
	}

	public String CompleteSpriteFile(String title, String spriteCode,
			String keyboardOptions, String onLoad) {
		return String.format(BoilerplateHtml(), title, onLoad, spriteCode,
				keyboardOptions);
	}
}
```

Comparison.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

public class Comparison extends NodeStringNode {
        public Comparison(BaseAST homogeneousNode, Scope scope)
                        throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope);
        }
}
```

ElseIf.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

public class ElseIf extends Node {
	private Node thisCondition;
	private Block thisBlock;

	public ElseIf(BaseAST condition, BaseAST block, Scope scope)
			throws ASTBuildingException, SSAException {
		super(condition, scope);
		thisCondition = NodeForHomogeneous(condition);
		thisBlock = new Block(block, scope);
	}

	public void Load(BaseAST homogeneousNode) throws ASTBuildingException { }

	public String toString(TargetLanguage language) {
		StringBuilder builder = new StringBuilder();
		builder.append(TabsString());
		if (TargetLanguage.Sprite == language) {
			builder.append("elseif");
		} else {
			builder.append("else if");
		}
		builder.append(" (");
		builder.append(thisCondition.toString(language));
		builder.append(")");
		thisBlock.tabs = tabs;
		builder.append(thisBlock.toString(language));
		return builder.toString();
	}
}
```

Equals.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

public class Equals extends NodeStringNode {
        public Equals(BaseAST homogeneousNode, Scope scope)
                        throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope);
        }
}
```

ForLoop.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

public class ForLoop extends Node {
        private Node initialize, condition, modify, block;

        public ForLoop(BaseAST homogeneousNode, Scope scope)
                        throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope);
        }

        public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
                        SSAException {
                BaseAST child = (BaseAST) homogeneousNode.getFirstChild();
                initialize = NodeForHomogeneous(child);
                child = (BaseAST) child.getNextSibling();
                condition = NodeForHomogeneous(child);
                child = (BaseAST) child.getNextSibling();
                modify = NodeForHomogeneous(child);
                child = (BaseAST) child.getNextSibling();
                block = NodeForHomogeneous(child);
        }

        public String LineEnding(TargetLanguage language) {
                return "\n";
        }

        public String toString(TargetLanguage language) {
                StringBuilder builder = new StringBuilder();
                builder.append("for (");
                builder.append(initialize.toString(language));
                builder.append("; ");
                builder.append(condition.toString(language));
                builder.append("; ");
                builder.append(modify.toString(language));
                builder.append(")");
                block.tabs = tabs;
                builder.append(block.toString(language));
                return builder.toString();
        }
}
```

Func.java

```
package heterogeneousAST;

import java.util.ArrayList;

import ssa.RedefinedException;
import ssa.SSAErrorMessages;
import ssa.SSAException;
import ssa.ScopeException;
import ssa.Type;
import antlr.BaseAST;

public class Func extends Node {
        private Identifier thisFunctionName;
        private ArrayList<Identifier> thisArguments;
        private Block thisBlock;

        public Func(BaseAST homogeneousNode, Scope scope)
                        throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope);
        }

        public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
                        SSAException {
                thisArguments = new ArrayList<Identifier>();
                BaseAST firstChild = (BaseAST) homogeneousNode.getFirstChild();
                thisFunctionName = new Identifier(firstChild, getCurrentScope(), this);
                BaseAST argument = (BaseAST) firstChild.getFirstChild();
                while (null != argument) {
                        Identifier argumentId = new Identifier(argument, getCurrentScope(),
                                        this);
                        thisArguments.add(argumentId);
                        getCurrentScope().addSymbol(argumentId, argumentId.toString(),
                                        new Type(Type.ID));
                        argument = (BaseAST) argument.getNextSibling();
                }
                BaseAST nextChild = (BaseAST) firstChild.getNextSibling();
                if (!nextChild.getText().equals("{")) {
                        String modifier = nextChild.getText();
                        nextChild = (BaseAST) nextChild.getNextSibling();
                        if (modifier.equals("callevery")) {
                                callEvery = NodeForHomogeneous(nextChild);
                        } else {
                                onKeyboard = NodeForHomogeneous(nextChild);
                        }
                        nextChild = (BaseAST) nextChild.getNextSibling();
                }
                thisBlock = new Block(nextChild, getCurrentScope());
        }

        public void FirstPassAnalysis() throws SSAException {
                if (getCurrentScope().getParentScope().getParentScope() != null) {
                        throw new ScopeException(SSAErrorMessages
                                        .getFunctionScopeError(thisFunctionName.toString()));
                } else if (getCurrentScope().getParentScope().containsSymbol(
                                thisFunctionName.toString(), Type.FUNC)) {
                        throw new RedefinedException(SSAErrorMessages
                                        .getRedefinedFuncError(thisFunctionName.toString()));
                } else if (getCurrentScope().getParentScope().containsSymbol(
                                thisFunctionName.toString(), Type.ID)) {
                        throw new RedefinedException("already used variable '"
                                        + thisFunctionName.toString() + "' as a function!");
                } else if (getCurrentScope().getParentScope() != null) {
                        for (Identifier nextNode : thisArguments) {
                                if (!getCurrentScope().getParentScope().containsName(
                                                nextNode.toString())) {
                                        getCurrentScope().addSymbol(null, nextNode.toString(),
                                                        new Type(Type.ID));
                                }
                        }
                }
```

```
        getCurrentScope().getParentScope().addSymbol(this,
                        thisFunctionName.toString(),
                        new Type(Type.FUNC, thisArguments.size()));
}

public String toString(TargetLanguage language) {
        thisBlock.tabs = tabs;
        StringBuilder builder = new StringBuilder();
        if (language == TargetLanguage.Javascript) {
                builder.append("function ");
                builder.append(ArgumentsString(language));
        } else if (language == TargetLanguage.Sprite) {
                builder.append("func ");
                builder.append(ArgumentsString(language));
                if (callEvery != null) {
                        builder.append(" callevery(");
                        builder.append(callEvery.toString(language));
                        builder.append(")");
                } else if (onKeyboard != null) {
                        builder.append(" onkeyboard(");
                        builder.append(onKeyboard.toString(language));
                        builder.append(")");
                }
        }
        builder.append(thisBlock.toString(language));
        if (language == TargetLanguage.Javascript && callEvery != null) {
                builder.append(TimingFunction(language));
        }
        return builder.toString();
}

public String LineEnding(TargetLanguage language) {
        return "";
}

private String TimingFunction(TargetLanguage language) {
        StringBuilder builder = new StringBuilder();
        String func = TimingFunctionName(language);
        builder.append(TabsString());
        builder.append("function ");
        builder.append(func);
        builder.append("() {\n\t");
        builder.append(TabsString());
        builder.append(thisFunctionName.toString(language));
        builder.append("();\n\t");
        builder.append(TabsString());
        builder.append("window.setTimeout(\"");
        builder.append(func);
        builder.append("()\", ");
        builder.append(callEvery.toString(language));
        builder.append(");\n");
        builder.append(TabsString());
        builder.append("}\n");
        return builder.toString();
}

public String TimingFunctionName(TargetLanguage language) {
        return thisFunctionName.toString(language) + "Timing";
}

public String FunctionName(TargetLanguage language) {
        return thisFunctionName.toString(language);
}

private String ArgumentsString(TargetLanguage language) {
        StringBuilder builder = new StringBuilder();
        builder.append(thisFunctionName.toString(language));
        builder.append(" (");
        boolean first = true;
        for (Identifier nextArgument : thisArguments) {
                if (first) {
                        first = false;
                } else {
                        builder.append(", ");
```

```
                        }
                        builder.append(nextArgument.toString(language));
                }
                builder.append(")");
                return builder.toString();
        }
}
```

FunctionCall.java

```
package heterogeneousAST;

import frontend.SpriteParserTokenTypes;

import java.util.ArrayList;

import ssa.InvalidArgumentCountException;
import ssa.SSAErrorMessages;
import ssa.SSAException;
import ssa.Type;
import ssa.UndefinedFuncException;
import ssa.UndefinedVariableException;
import antlr.BaseAST;

/* Function calls have an identifier for the function name and a list of
 * arguments. Arguments are expressions.
 */
public class FunctionCall extends Node {
        private ArrayList<Node> thisArguments;
        private Identifier thisFunctionName;

        public FunctionCall(BaseAST homogeneousNode, Scope scope)
                        throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope);
        }

        public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
        SSAException {
                thisArguments = new ArrayList<Node>();
                thisFunctionName = new Identifier(homogeneousNode, getCurrentScope(),
                                this);
                BaseAST child = (BaseAST) homogeneousNode.getFirstChild()
                .getFirstChild();
                while (null != child) {
                        thisArguments.add(NodeForHomogeneous(child));
                        child = (BaseAST) child.getNextSibling();
                }
        }

        public void FirstPassAnalysis() throws SSAException {
                if (getCurrentScope().containsName(thisFunctionName.toString())) {
                        SecondPassAnalysis();
                }
                if (!getCurrentScope().containsName(thisFunctionName.toString())) {
                        getCurrentScope().addSymbol((Node) this,
                                        thisFunctionName.toString(),
                                        new Type(Type.FUNC_CALL, thisArguments.size()));
                }
        }

        public void SecondPassAnalysis() throws SSAException {
                if (getCurrentScope().containsSymbol(thisFunctionName.toString(),
                                Type.FUNC)) {
                        Type type = getCurrentScope().getType(thisFunctionName.toString());
                        if (thisArguments.size() != type.getNumArguments()) {
                                throw new InvalidArgumentCountException(SSAErrorMessages
                                                .getInvalidArgumentCountError(thisFunctionName
                                                                .toString(), thisArguments.size(), type
                                                                .getNumArguments()));
                        }
                } else if (getCurrentScope().getParentScope() != null
                                && getCurrentScope().getParentScope().containsSymbol(
                                                thisFunctionName.toString(), Type.FUNC)) {
                        Type type = getCurrentScope().getType(thisFunctionName.toString());
                        if (thisArguments.size() != type.getNumArguments()) {
                                throw new InvalidArgumentCountException(SSAErrorMessages
                                                .getInvalidArgumentCountError(thisFunctionName
                                                                .toString(), thisArguments.size(), type
                                                                .getNumArguments()));
                        }
```

```
                } else {
                        throw new UndefinedFuncException(SSAErrorMessages
                                        .getUndefinedFuncError(thisFunctionName.toString()));
                }

                for (Node nextNode : thisArguments) {
                        int homogeneous_type = nextNode.getHomogeneousNodeType();
                        if (homogeneous_type == SpriteParserTokenTypes.ID) {
                                if (getCurrentScope().getParentScope() == null
                                                && !getCurrentScope().containsName(
                                                                nextNode.getHomogeneousNode().getText())) {
                                        throw new UndefinedVariableException(SSAErrorMessages
                                                        .getUndefinedVarError(nextNode.getHomogeneousNode()
                                                                        .getText()));
                                }
                                if (getCurrentScope().getParentScope() != null
                                                && !getCurrentScope().containsName(
                                                                nextNode.getHomogeneousNode().getText())
                                                && !getCurrentScope().getParentScope().containsName(
                                                                nextNode.getHomogeneousNode().getText())) {
                                        throw new UndefinedVariableException(SSAErrorMessages
                                                        .getUndefinedVarError(nextNode.getHomogeneousNode()
                                                                        .getText()));
                                }
                        }
                }
        }

        public String toString(TargetLanguage language) {
                StringBuilder builder = new StringBuilder();
                builder.append(thisFunctionName.toString(language));
                builder.append("(");
                int i = 0;
                for (Node nextNode : thisArguments) {
                        if (i > 0) {
                                builder.append(", ");
                        }
                        builder.append(nextNode.toString(language));
                        i++;
                }
                builder.append(")");
                return builder.toString();
        }
}
```

GroupedExpression.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

public class GroupedExpression extends Node {
        private Node innerExpression;

        public GroupedExpression(BaseAST condition, Scope scope)
                        throws ASTBuildingException, SSAException {
                super(condition, scope);
        }

        public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
                        SSAException {
                BaseAST child = (BaseAST) homogeneousNode.getFirstChild();
                innerExpression = NodeForHomogeneous(child);
                utility.Asserter.Assert(null == child.getNextSibling(),
                                "Inner expressions should only have 1 child node");
        }

        public String toString(TargetLanguage language) {
                StringBuilder builder = new StringBuilder();
                builder.append("(");
                builder.append(innerExpression.toString(language));
                builder.append(")");
                return builder.toString();
        }
}
```

HeterogeneousTree.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

/*
 * This is the highest level node of the heterogeneous tree.
 */
public class HeterogeneousTree extends StatementList {
	private String thisTitle;

	public HeterogeneousTree(String title, BaseAST homogeneousNode, Scope scope)
			throws ASTBuildingException, SSAException {
		super(homogeneousNode, scope);
		thisTitle = title;
	}

	public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
			SSAException {
		LoadChildren((BaseAST) homogeneousNode);
	}

	public String toString(TargetLanguage language) {
		StringBuilder timing = new StringBuilder();
		StringBuilder keyboard = new StringBuilder();
		for (Node nextNode : thisNodes) {
			if (nextNode.callEvery != null) {
				timing.append(((Func) nextNode).TimingFunctionName(language));
				timing.append("();");
			}
			if (nextNode.onKeyboard != null) {
				if (keyboard.length() > 0) {
					keyboard.append("\n\t\t\t\telse ");
				}
				keyboard.append("if (keychar == ");
				keyboard.append(nextNode.onKeyboard.toString(language));
				keyboard.append(") { ");
				keyboard.append(((Func) nextNode).FunctionName(language));
				keyboard.append("(); }");
			}
		}
		String returnValue = "";
		int depth = 0;
		if (language == TargetLanguage.Javascript) {
			depth = 3;
		}
		String childString = ChildString(language, depth);
		if (language == TargetLanguage.Javascript) {
			Boilerplate boiler = new Boilerplate();
			returnValue = boiler.CompleteSpriteFile(thisTitle, childString,
					keyboard.toString(), timing.toString());
		} else if (language == TargetLanguage.Sprite) {
			returnValue = childString;
		}
		return returnValue;
	}
}
```

Identifier.java

```
package heterogeneousAST;

import ssa.LocalException;
import ssa.RedefinedException;
import ssa.SSAErrorMessages;
import ssa.SSAException;
import ssa.Type;
import ssa.UndefinedVariableException;
import antlr.BaseAST;

/* Identifiers are variables, function names, and class names. */
public class Identifier extends Node {
        private String thisName;

        public Identifier(BaseAST homogeneousNode, Scope scope, Node parent)
                        throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope, parent);
        }

        public Identifier(BaseAST homogeneousNode, Scope scope, Node parent,
                        boolean local) throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope, parent, local);
        }

        public Identifier(BaseAST homogeneousNode, Scope scope, Node parent,
                        boolean local, boolean isRightValue) throws ASTBuildingException,
                        SSAException {
                super(homogeneousNode, scope, parent, local, isRightValue);
        }

        public void Load(BaseAST homogeneousNode) {
                thisName = homogeneousNode.getText();
        }

        public void FirstPassAnalysis() throws SSAException {
                Object parentClass;
                if (null == parent) {
                        parentClass = Object.class;
                } else {
                        parentClass = parent.getClass();
                }

                if (Return.class.equals(parentClass)
                                || BinaryOperator.class.equals(parentClass)
                                || If.class.equals(parentClass)
                                || OrAnd.class.equals(parentClass)
                                || ArrayAccess.class.equals(parentClass)
                                || ElseIf.class.equals(parentClass)) {
                        checkVar();
                } else if (Equals.class.equals(parentClass)) {
                        checkEquals();
                } else if (Obj.class.equals(parentClass)) {
                        checkObject();
                } else if (Object.class.equals(parentClass)) {
                        checkVar();
                }
        }

        private void checkVar() throws UndefinedVariableException {
                if (!getCurrentScope().containsName(thisName)) {
                        if (getCurrentScope().getParentScope() == null
                                        || !getCurrentScope().getParentScope().containsName(
                                                        thisName)) {
                                throw new UndefinedVariableException(SSAErrorMessages
                                                .getUndefinedVarError(thisName));
                        }
                }
        }

        private void checkObject() throws SSAException {
```

```
        if (getCurrentScope().getParentScope() != null) {
                throw new SSAException(
                                "You are not allowed to define an object within a function.");
        } else if (getCurrentScope().containsSymbol(thisName, Type.UNDEFINED)) {
                getCurrentScope().updateType(thisName, new Type(Type.OBJ));
        } else if (!getCurrentScope().containsName(thisName)) {
                getCurrentScope().addSymbol((Node) this, thisName,
                                new Type(Type.OBJ));
        } else if (getCurrentScope().containsName(thisName)) {
                throw new RedefinedException(SSAErrorMessages
                                .getRedefinedObject(thisName));
        }
}

private void checkEquals() throws SSAException {
        if (thisLocal) {
                if (getCurrentScope().containsName(thisName)) {
                        throw new SSAException(SSAErrorMessages
                                        .getRedefinedLocalError(thisName));
                }
        }
        if ((getCurrentScope().getParentScope() != null && getCurrentScope()
                        .getParentScope().containsName(thisName)))
                return;

        if ((getCurrentScope().getParentScope() != null && getCurrentScope()
                        .getParentScope().containsSymbol(thisName, Type.FUNC))
                        || getCurrentScope().containsSymbol(thisName, Type.FUNC)) {
                throw new SSAException("is this an func or ID?");
        }
        if ((getCurrentScope().getParentScope() != null && getCurrentScope()
                        .getParentScope().containsSymbol(thisName, Type.OBJ))
                        || getCurrentScope().containsSymbol(thisName, Type.OBJ)) {
                throw new SSAException("is this an object or ID?");
        }
        if (!getCurrentScope().containsName(thisName)) {
                if (isRightValue) {
                        throw new UndefinedVariableException(SSAErrorMessages
                                        .getUndefinedVarError(thisName));
                } else if (thisLocal || getCurrentScope().getParentScope() == null) {
                        getCurrentScope().addSymbol((Node) this, thisName,
                                        new Type(Type.ID));
                } else {
                        getCurrentScope().getParentScope().addSymbol((Node) this,
                                        thisName, new Type(Type.ID));
                }
        }
}

public void SecondPassAnalysis() throws SSAException {
        if (thisLocal && getCurrentScope().getParentScope() == null) {
                throw new LocalException(SSAErrorMessages.getLocalAtGlobalError());
        }
}

/*
 * To reproduce Sprite's scoping rules in JavaScript, we need to put var in
 * front of each variable the first time it appears.
 */
public String toString(TargetLanguage language) {
        if (TargetLanguage.Javascript == language && thisName.equals("array")) {
                return "Array";
        } else {
                String returnValue = "";
                if (thisLocal) {
                        if (TargetLanguage.Javascript == language) {
                                returnValue = "var ";
                        } else {
                                returnValue = "local ";
                        }
                }
                return returnValue + thisName;
        }
}
```

```
        public String toString() {
                return thisName;
        }
}
```

If.java

```
package heterogeneousAST;

import java.util.ArrayList;

import ssa.SSAException;
import antlr.BaseAST;

public class If extends Node {
	private Node topBoolean;
	private Block ifBlock;
	private ArrayList<ElseIf> elseIfBlocks;
	private Block elseBlock;

	public If(BaseAST homogeneousNode, Scope scope)
			throws ASTBuildingException, SSAException {
		super(homogeneousNode, scope);
	}

	public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
			SSAException {
		elseIfBlocks = new ArrayList<ElseIf>();
		BaseAST child = (BaseAST) homogeneousNode.getFirstChild();
		topBoolean = NodeForHomogeneous(child);
		child = (BaseAST) child.getNextSibling();
		ifBlock = new Block(child, getCurrentScope());
		child = (BaseAST) child.getNextSibling();
		while (null != child) {
			if (child.getText().equals("{")) {
				elseBlock = new Block(child, getCurrentScope());
			} else {
				elseIfBlocks.add(new ElseIf(child, (BaseAST) child
						.getNextSibling(), getCurrentScope()));
				child = (BaseAST) child.getNextSibling();
			}
			child = (BaseAST) child.getNextSibling();
		}
	}

	public String LineEnding(TargetLanguage language) {
		return "\n";
	}

	public String toString(TargetLanguage language) {
		StringBuilder builder = new StringBuilder();
		builder.append("if (");
		builder.append(topBoolean.toString(language));
		builder.append(")");
		ifBlock.tabs = tabs;
		builder.append(ifBlock.toString(language));
		for (ElseIf nextElseIf : elseIfBlocks) {
			nextElseIf.tabs = tabs;
			builder.append(nextElseIf.toString(language));
		}
		if (null != elseBlock) {
			elseBlock.tabs = tabs;
			builder.append(TabsString());
			builder.append("else");
			builder.append(elseBlock.toString(language));
		}
		return builder.toString();
	}
}
```

MemberHead.java

```
package heterogeneousAST;

import java.util.ArrayList;

import ssa.SSAException;
import antlr.BaseAST;

public class MemberHead extends Node {
	private Node head;
	private ArrayList<Identifier> members;

	public MemberHead(BaseAST homogeneousNode, Scope scope)
			throws ASTBuildingException, SSAException {
		super(homogeneousNode, scope);
	}

	public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
			SSAException {
		head = new Identifier(homogeneousNode, getCurrentScope(), null);
		members = new ArrayList<Identifier>();
		BaseAST child1 = (BaseAST) homogeneousNode.getFirstChild();
		BaseAST child2 = null;
		while (child1 != null) {
			child2 = (BaseAST) child1.getFirstChild();
			Identifier nextTail = new Identifier(child2, getCurrentScope(),
					this);
			members.add(nextTail);
			child1 = (BaseAST) child2.getFirstChild();
		}
	}

	public String toString(TargetLanguage language) {
		StringBuilder builder = new StringBuilder();
		builder.append(head.toString(language));
		for (Identifier nextChild : members) {
			builder.append(".");
			builder.append(nextChild.toString(language));
		}
		return builder.toString();
	}
}
```

New.java

```
package heterogeneousAST;

import ssa.SSAErrorMessages;
import ssa.SSAException;
import ssa.Type;
import ssa.UndefinedObjectException;
import antlr.BaseAST;

/* New is the node for one type of rvalue, it will be the new keyword followed by an object name */
public class New extends Node {
	protected Identifier objectName;

	public New(BaseAST homogeneousNode, Scope scope)
			throws ASTBuildingException, SSAException {
		super(homogeneousNode, scope);
	}

	public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
			SSAException {
		objectName = new Identifier((BaseAST) homogeneousNode.getFirstChild(),
				getCurrentScope(), this);
	}

	public void FirstPassAnalysis() throws SSAException {
		if (getCurrentScope().getParentScope() == null
				&& !getCurrentScope().containsName(objectName.toString())) {
			getCurrentScope().addSymbol((Node) this, objectName.toString(),
					new Type(Type.UNDEFINED));
		} else if (getCurrentScope().getParentScope() != null
				&& !getCurrentScope().getParentScope().containsName(
						objectName.toString())) {
			getCurrentScope().getParentScope().addSymbol((Node) this,
					objectName.toString(), new Type(Type.UNDEFINED));
		}
	}

	public void SecondPassAnalysis() throws SSAException {
		if (getCurrentScope().containsSymbol(objectName.toString(),
				Type.UNDEFINED)) {
			throw new UndefinedObjectException(SSAErrorMessages
					.getUndefinedObjectError(objectName.toString()));
		}
	}

	public String toString(TargetLanguage language) {
		StringBuilder builder = new StringBuilder();
		builder.append("new ");
		builder.append(objectName.toString(language));
		if (TargetLanguage.Javascript == language) {
			builder.append("()");
		}
		return builder.toString();
	}
}
```

Node.java

```
package heterogeneousAST;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

import ssa.SSAException;
import utility.Asserter;
import antlr.BaseAST;

/* Node is the base class for anything in the structured abstract syntax tree. */
public abstract class Node {
        /* How many tabs to display before this Node */
        public int tabs = 0;

        public String TabsString() {
                StringBuilder builder = new StringBuilder();
                for (int tab = 0; tab < tabs; tab++) {
                        builder.append("\t");
                }
                return builder.toString();
        }

        /*
         * The top level node (which is a StructuredAST object) and functions both
         * contain lists of nodes. If those nodes are expressions, then each
         * expression should get its own line. StructuredAST and Func must both tell
         * their child nodes that they are on their own lines if they are
         * expressions.
         */
        public boolean onOwnLine = false;

        /*
         * Every node must support a toString function based on a specific language.
         * I put the job of rendering each node right in the node because I didn't
         * want to have another huge lists of classes like AtomRenderer,
         * BinaryOperatorRenderer, etc... I thought it would just be easier to put
         * rendering information right in the structuredAST. I also decided to use
         * just one toString function instead of something like toPrettyString and
         * toJavascriptString. This is because most of the time, we actually render
         * the exact same way, so we can often ignore the language parameter.
         */
        public abstract String toString(heterogeneousAST.TargetLanguage language);

        /*
         * You can specify that a function must be called every so many
         * milliseconds. To make it so I don't have to check what kind of nodes I'm
         * looking at, I'll just check what the callevery value is. If it's > 0 I'll
         * wire up callevery code. This means that programmers should be smart
         * enough to never set the call every value of a for loop or anything stupid
         * like that.
         */
        public Node callEvery;
        public Node onKeyboard;

        /*
         * Every node must know how to load itself from a homogeneous node
         */
        public abstract void Load(BaseAST homogeneousNode)
                        throws ASTBuildingException, SSAException;

        public void FirstPassAnalysis() throws SSAException { };
        public void SecondPassAnalysis() throws SSAException { };

        private Scope thisScope;
        private BaseAST homogeneousNode;
        public boolean thisLocal = false;
        public boolean isRightValue = false;

        protected Node parent;
```

```
public Scope getCurrentScope() {
        return thisScope;
}

public int getHomogeneousNodeType() {
        return homogeneousNode.getType();
}

public BaseAST getHomogeneousNode() {
        return homogeneousNode;
}

public String getLineAndColumn() {
        return homogeneousNode.getLine() + ":" + homogeneousNode.getColumn();
}

public Node(BaseAST homogeneousNode, Scope scope)
                throws ASTBuildingException, SSAException {
        this(homogeneousNode, scope, null, false);
}

public Node(BaseAST homogeneousNode, Scope scope, Node parent)
                throws ASTBuildingException, SSAException {
        this(homogeneousNode, scope, parent, false);
}

public Node(BaseAST homogeneousNode, Scope scope, Node parent,
                boolean thisLocal) throws ASTBuildingException, SSAException {
        this(homogeneousNode, scope, parent, thisLocal, false);
}

public Node(BaseAST homogeneousNode, Scope scope, Node parent,
                boolean thisLocal, boolean isRightValue)
                throws ASTBuildingException, SSAException {
        this.homogeneousNode = homogeneousNode;
        this.thisScope = scope;
        this.parent = parent;
        this.thisLocal = thisLocal;
        this.isRightValue = isRightValue;
        Load(homogeneousNode);
        FirstPassAnalysis();
}

/*
 * This is the default line ending for any node that is on its own line.
 * Objects and Functions will, for example, override this functionality.
 */
public String LineEnding(TargetLanguage language) {
        if (TargetLanguage.Javascript == language) {
                return ";\n";
        }
        return "\n";
}

public Node NodeForHomogeneous(BaseAST homogeneousNode, boolean isAssignment)
                throws ASTBuildingException, SSAException {
        isRightValue = isAssignment;
        Asserter.Assert(null != homogeneousNode, "homogeneousNode null");
        Node returnValue;
        String text = homogeneousNode.getText();
        Matcher isIdentifier = Pattern.compile("^[\\w_][\\w\\d_]*$").matcher(
                        text);
        Matcher isComparison = Pattern.compile("^(==|!=|<=|>=|>|<)$").matcher(
                        text);
        utility.Numbers numbers = new utility.Numbers();
        if ("obj".equals(text)) {
                returnValue = new Obj(homogeneousNode, thisScope);
        } else if ("func".equals(text)) {
                returnValue = new Func(homogeneousNode, new Scope(thisScope));
        } else if ("!".equals(text)) {
                returnValue = new Not(homogeneousNode, new Scope(thisScope));
        } else if ("for".equals(text)) {
                returnValue = new ForLoop(homogeneousNode, thisScope);
        } else if ("if".equals(text)) {
```

```
                        returnValue = new If(homogeneousNode, thisScope);
                } else if ("new".equals(text)) {
                        returnValue = new New(homogeneousNode, thisScope);
                } else if ("ret".equals(text)) {
                        returnValue = new Return(homogeneousNode, thisScope);
                } else if ("local".equals(text)) {
                        returnValue = new Identifier((BaseAST) homogeneousNode
                                        .getFirstChild(), thisScope, this, true);
                } else if (text.indexOf("\"") >= 0) {
                        returnValue = new Str(homogeneousNode, thisScope);
                } else if ("=".equals(text)) {
                        returnValue = new Equals(homogeneousNode, thisScope);
                } else if (isComparison.matches()) {
                        returnValue = new Comparison(homogeneousNode, thisScope);
                } else if ("&&".equals(text) || "||".equals(text)) {
                        returnValue = new OrAnd(homogeneousNode, thisScope);
                } else if ("mod".equals(text)
                                || (text.length() == 1 && "+-/*".indexOf(text) >= 0)) {
                        returnValue = new BinaryOperator(homogeneousNode, thisScope);
                } else if (text.equals("{")) {
                        returnValue = new Block(homogeneousNode, thisScope);
                } else if (text.equals("(")) {
                        returnValue = new GroupedExpression(homogeneousNode, thisScope);
                } else if (numbers.isNumeric(text)) {
                        returnValue = new Num(homogeneousNode, thisScope);
                } else if (isIdentifier.matches()) {
                        BaseAST firstChild = (BaseAST) homogeneousNode.getFirstChild();
                        if (null == firstChild) {
                                if (isRightValue) {
                                        returnValue = new Identifier(homogeneousNode, thisScope,
                                                        this, false, true);
                                } else {
                                        returnValue = new Identifier(homogeneousNode, thisScope,
                                                        this);
                                }
                        } else if (".".equals(firstChild.getText())
                                        && null == firstChild.getNextSibling()) {
                                returnValue = new MemberHead(homogeneousNode, thisScope);
                        } else if ("[".equals(firstChild.getText())
                                        && null == firstChild.getNextSibling()) {
                                returnValue = new ArrayAccess(homogeneousNode, thisScope);
                        } else if ("(".equals(firstChild.getText())
                                        && null == firstChild.getNextSibling()) {
                                returnValue = new FunctionCall(homogeneousNode, thisScope);
                        } else {
                                throw new ASTBuildingException("Unsure what to do with id "
                                                + text + " with first child " + firstChild.getText());
                        }
                } else if ("".equals(text)) {
                        throw new ASTBuildingException("Unable to handle empty nodes");
                } else {
                        throw new ASTBuildingException(String
                                        .format("Unable to handle this kind of node: "
                                                        + text.toString()));
                }
                return returnValue;
        }

        public Node NodeForHomogeneous(BaseAST homogeneousNode)
                        throws ASTBuildingException, SSAException {
                return NodeForHomogeneous(homogeneousNode, false);
        }
}
```

NodeStringNode.java

```
package heterogeneousAST;

import ssa.SSAException;
import utility.Asserter;
import antlr.BaseAST;

public abstract class NodeStringNode extends Node {
	protected Node thisLeftValue;
	protected String thisString;
	protected Node thisRightValue;

	public NodeStringNode(BaseAST homogeneousNode, Scope scope)
			throws ASTBuildingException, SSAException {
		super(homogeneousNode, scope);
	}

	public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
			SSAException {
		thisString = homogeneousNode.getText();
		BaseAST child1 = (BaseAST) homogeneousNode.getFirstChild();
		if (child1.getNextSibling() != null) {
			thisRightValue = NodeForHomogeneous((BaseAST) child1
					.getNextSibling(), true);
		} else {
			Asserter.Assert(homogeneousNode.getText().equals("-"),
					"Only the - binary operator is allowed to have one child. "
							+ homogeneousNode.getText() + " is not allowed.");
		}
		thisLeftValue = NodeForHomogeneous(child1);
	}

	public String toString(TargetLanguage language) {
		if (thisString.equals("mod") && TargetLanguage.Javascript == language) {
			thisString = "%";
		}
		StringBuilder builder = new StringBuilder();
		if (thisRightValue == null) {
			builder.append(" ");
			builder.append(thisString);
		}
		builder.append(thisLeftValue.toString(language));
		if (thisRightValue != null) {
			builder.append(" ");
			builder.append(thisString);
			builder.append(" ");
			builder.append(thisRightValue.toString(language));
		}
		return builder.toString();
	}
}
```

Not.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

public class Not extends Node {
        protected Node notThis;
        public Not(BaseAST homogeneousNode, Scope scope)
                        throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope);
        }

        public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
                SSAException {
                notThis = NodeForHomogeneous((BaseAST) homogeneousNode.getFirstChild());
        }

        public String toString(TargetLanguage language) {
                return "! " + notThis.toString(language);
        }
}
```

Num.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

public class Num extends Node {
	private int thisValueI;

	public Num(BaseAST homogeneousNode, Scope scope)
			throws ASTBuildingException, SSAException {
		super(homogeneousNode, scope);
	}

	public void Load(BaseAST homogeneousNode) throws ASTBuildingException {
		thisValueI = 0;
		try {
			thisValueI = Integer.parseInt(homogeneousNode.getText());
		} catch (NumberFormatException e) {
			throw new ASTBuildingException("overflow: ("
					+ homogeneousNode.getText() + ") numbers must be > "
					+ Integer.MIN_VALUE + " and <= " + Integer.MAX_VALUE);
		}
	}

	public void FirstPassAnalysis() throws SSAException {
		if (Math.abs(thisValueI) > Integer.MAX_VALUE) {
			throw new SSAException("Overflow: " + thisValueI);
		}
	}

	public String toString(TargetLanguage language) {
		return Integer.toString(thisValueI);
	}
}
```

Obj.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

public class Obj extends Node {
	private Identifier name;
	private Block block;

	public Obj(BaseAST homogeneousNode, Scope scope)
			throws ASTBuildingException, SSAException {
		super(homogeneousNode, scope);
	}

	public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
			SSAException {
		BaseAST firstChild = (BaseAST) homogeneousNode.getFirstChild();
		name = new Identifier(firstChild, getCurrentScope(), this);
		block = new Block((BaseAST) firstChild.getNextSibling(),
				getCurrentScope());
	}

	public String toString(TargetLanguage language) {
		StringBuilder builder = new StringBuilder();
		if (language == TargetLanguage.Javascript) {
			builder.append("function ");
		} else if (language == TargetLanguage.Sprite) {
			builder.append("obj ");
		}
		builder.append(name.toString(language));
		if (language == TargetLanguage.Javascript) {
			builder.append("()");
		}
		block.tabs = tabs;
		block.isObjectBlock = true;
		builder.append(block.toString(language));
		return builder.toString();
	}

	public String LineEnding(TargetLanguage language) {
		return "\n";
	}
}
```

OrAnd.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

public class OrAnd extends NodeStringNode {
        public OrAnd(BaseAST homogeneousNode, Scope scope)
                        throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope);
        }
}
```

Return.java

```
package heterogeneousAST;

import ssa.SSAErrorMessages;
import ssa.SSAException;
import antlr.BaseAST;

public class Return extends Node {
	private Node returnValue;

	public Return(BaseAST condition, Scope scope) throws ASTBuildingException,
			SSAException {
		super(condition, scope);
	}

	public void Load(BaseAST homogeneousNode) throws ASTBuildingException,
			SSAException {
		returnValue = NodeForHomogeneous((BaseAST) homogeneousNode
				.getFirstChild());
	}

	public void FirstPassAnalysis() throws SSAException {
		if (getCurrentScope().getParentScope() == null) {
			throw new SSAException(SSAErrorMessages
					.getInvalidReturnError(returnValue.toString()));
		}
	}

	public String toString(TargetLanguage language) {
		StringBuilder builder = new StringBuilder();
		if (TargetLanguage.Sprite == language) {
			builder.append("ret ");
		} else {
			builder.append("return ");
		}
		builder.append(returnValue.toString(language));
		return builder.toString();
	}
}
```

Scope.java

```
package heterogeneousAST;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;
import ssa.SSAException;
import ssa.Type;

public class Scope {
	private ArrayList<Node> nodes;
	private ArrayList<Scope> childScopes;
	private Scope parentScope;

	private Map<String, Type> symbolTable = new HashMap<String, Type>();

	public Scope() {
		nodes = new ArrayList<Node>();
		childScopes = new ArrayList<Scope>();
		parentScope = null;
		addBuiltInFunctions();
		addBuiltInObject();
	}

	public Scope(Scope parentScope) {
		nodes = new ArrayList<Node>();
		childScopes = new ArrayList<Scope>();
		this.parentScope = parentScope;
		this.parentScope.AddScope(this);
		addBuiltInFunctions();
		addBuiltInObject();
	}

	private void addBuiltInFunctions() {
		addSymbol(null, "add", new Type(Type.FUNC, 2));
		addSymbol(null, "print", new Type(Type.FUNC, 1));
		addSymbol(null, "sprite", new Type(Type.FUNC, 4));
		addSymbol(null, "count", new Type(Type.FUNC, 1));
	}

	private void addBuiltInObject() {
		addSymbol(null, "array", new Type(Type.OBJ));
		addSymbol(null, "true", new Type(Type.BOOL));
		addSymbol(null, "false", new Type(Type.BOOL));
	}

	public Scope getParentScope() {
		return parentScope;
	}

	public void addSymbol(Node node, String key, Type type) {
		symbolTable.put(key, type);
		addNode(node);
	}

	public void addNode(Node node) {
		if (node != null) {
			nodes.add(node);
		}
	}

	public void AddScope(Scope childScope) {
		childScopes.add(childScope);
	}

	public void SecondPassAnalysis() throws SSAException {
		for (Node nextNode : nodes) {
			nextNode.SecondPassAnalysis();
		}
		for (Scope nextChildScope : childScopes) {
			nextChildScope.SecondPassAnalysis();
```

```
                }
        }

        public Type getType(String name) {
                Type type;
                if (parentScope != null) {
                        type = (Type) parentScope.symbolTable.get(name);
                } else {
                        type = (Type) symbolTable.get(name);
                }
                return (type == null) ? new Type() : type;
        }

        public boolean containsName(String name) {
                return symbolTable.containsKey(name);
        }

        public boolean containsSymbol(String name, int type) {
                Type ourType = (Type) symbolTable.get(name);
                if (ourType == null) {
                        if (parentScope != null) {
                                ourType = (Type) parentScope.symbolTable.get(name);
                        }
                }
                if (ourType == null) {
                        return false;
                } else {
                        return (ourType.getType() == type);
                }
        }

        public void updateType(String name, Type type) {
                symbolTable.remove(name);
                symbolTable.put(name, type);
        }

        public int getTableSize() {
                return symbolTable.size();
        }
}
```

StatementList.java

```
package heterogeneousAST;

import java.util.ArrayList;

import ssa.SSAException;
import antlr.BaseAST;

public abstract class StatementList extends Node {
	public boolean isObjectBlock;
	protected ArrayList<Node> thisNodes;

	public StatementList(BaseAST homogeneousNode, Scope scope)
			throws ASTBuildingException, SSAException {
		super(homogeneousNode, scope);
		isObjectBlock = false;
	}

	protected void LoadChildren(BaseAST firstChild)
			throws ASTBuildingException, SSAException {
		thisNodes = new ArrayList<Node>();
		while (null != firstChild) {
			Node nextNode = NodeForHomogeneous(firstChild);
			nextNode.onOwnLine = true;
			thisNodes.add(nextNode);
			firstChild = (BaseAST) firstChild.getNextSibling();
		}
	}

	protected String ChildString(TargetLanguage language, int childDepth) {
		StringBuilder builder = new StringBuilder();
		for (Node nextNode : thisNodes) {
			nextNode.tabs = childDepth;
			builder.append(nextNode.TabsString());
			if (isObjectBlock && TargetLanguage.Javascript == language) {
				builder.append("this.");
			}
			builder.append(nextNode.toString(language));
			builder.append(nextNode.LineEnding(language));
		}
		return builder.toString();
	}
}
```

Str.java

```
package heterogeneousAST;

import ssa.SSAException;
import antlr.BaseAST;

public class Str extends Node {
        private String thisValue;

        public Str(BaseAST homogeneousNode, Scope scope)
                        throws ASTBuildingException, SSAException {
                super(homogeneousNode, scope);
        }

        public void Load(BaseAST homogeneousNode) {
                thisValue = homogeneousNode.getText();
        }

        public String toString(TargetLanguage language) {
                return thisValue;
        }
}
```

TargetLanguage.java

```
package heterogeneousAST;

public enum TargetLanguage {
        Sprite, Javascript
}
```

InvalidArgumentCountException.java

```
package ssa;

public class InvalidArgumentCountException extends SSAException {

	private static final long serialVersionUID = -3745137393298029277L;
	String message = null;
	Throwable cause = null;

	public InvalidArgumentCountException() {
	}

	public InvalidArgumentCountException(String message) {
		 super(message);
		 this.message = message;
	}

	public InvalidArgumentCountException(Throwable cause) {
		 super(cause);
		 this.cause.initCause(cause);
	}

	public InvalidArgumentCountException(String message, Throwable cause) {
		 super(message, cause);
		 this.message = message;
		 this.cause.initCause(cause);
	}

}
```

InvalidTypesException.java

```
package ssa;

public class InvalidTypesException extends SSAException {
	private static final long serialVersionUID = 5374862147587420756L;

	String message = null;
	Throwable cause = null;

	public InvalidTypesException() {
	}

	public InvalidTypesException(final String message) {
		 super(message);
		 this.message = message;
	}

	public InvalidTypesException(final Throwable cause) {
		 super(cause);
		 this.cause.initCause(cause);
	}

	public InvalidTypesException(final String message, final Throwable cause) {
		 super(message, cause);
		 this.message = message;
		 this.cause.initCause(cause);
	}

	public String getMessage() {
		 return message;
	}

	public String toString() {
		 return "InvalidTypesException: " + message;
	}

}
```

LocalException.java

```
package ssa;

public class LocalException extends SSAException {

	private static final long serialVersionUID = 6810579448115958719L;

	public LocalException() {
	}

	public LocalException(String message) {
		super(message);
	}

	public LocalException(Throwable cause) {
		super(cause);
	}

	public LocalException(String message, Throwable cause) {
		super(message, cause);
	}

}
```

RedefinedException.java

```
package ssa;

public class RedefinedException extends SSAException {

	private static final long serialVersionUID = 5460901350997695919L;

	public RedefinedException() {
	}

	public RedefinedException(String arg0) {
		super(arg0);
	}

	public RedefinedException(Throwable arg0) {
		super(arg0);
	}

	public RedefinedException(String arg0, Throwable arg1) {
		super(arg0, arg1);
	}

}
```

ScopeException.java

```
package ssa;

public class ScopeException extends SSAException {

	private static final long serialVersionUID = -8984267528666829493L;

	public ScopeException() {
	}

	public ScopeException(String arg0) {
		super(arg0);
	}

	public ScopeException(Throwable arg0) {
		super(arg0);
	}

	public ScopeException(String arg0, Throwable arg1) {
		super(arg0, arg1);
	}

}
```

SSAErrorMessages.java

```
package ssa;

public class SSAErrorMessages {
	public static String SSA_UNDEFINED_FUNC_ERROR = "You didn't create a function by the name of ";
	public static String SSA_REDEFINED_FUNC_ERROR = "You already defined a function by the name of ";
	public static String SSA_INVALID_ARG_COUNT_ERROR = "Invalid number of arguments passed to function ";
	public static String SSA_LOCAL_AT_GLOBAL_ERROR = "You cannot define a variable local at the local scope";
	public static String SSA_UNDEFINED_VAR_ERROR =
		"You are you trying to use this variable before assigning it a value: ";
	public static String SSA_UNDEFINED_OBJ_ERROR = "Object does not exist: ";
	public static String SSA_FUNC_SCOPE_ERROR =
		"You cannot defined a function within a function, only at the global scope ";
	public static String SSA_REDEFINED_LOCAL_ERROR =
		"You are trying to make a variable local that has already been used: ";
	public static String SSA_INVALID_RETURN_ERROR = "You are not in a function and are trying to return ";
	public static String SSA_REDEFINED_OBJ_ERROR = "You have already defined an object named ";

	public static String getRedefinedObject(String name) {
		return SSA_REDEFINED_OBJ_ERROR + name;
	}

	public static String getInvalidReturnError(String variable) {
		return SSA_INVALID_RETURN_ERROR + variable;
	}

	public static String getUndefinedFuncError(String function) {
		return SSA_UNDEFINED_FUNC_ERROR + function;
	}

	public static String getRedefinedFuncError(String function) {
		return SSA_REDEFINED_FUNC_ERROR + function;
	}

	public static String getInvalidArgumentCountError(String function,
			int calledWith, int shouldHave) {
		return SSA_INVALID_ARG_COUNT_ERROR + function + ", called with "
				+ calledWith + " arguments when it should have " + shouldHave;
	}

	public static String getLocalAtGlobalError() {
		return SSA_LOCAL_AT_GLOBAL_ERROR;
	}

	public static String getUndefinedVarError(String var) {
		return SSA_UNDEFINED_VAR_ERROR + var;
	}

	public static String getFunctionScopeError(String function) {
		return SSA_FUNC_SCOPE_ERROR + function;
	}

	public static String getUndefinedObjectError(String object) {
		return SSA_UNDEFINED_OBJ_ERROR + object;
	}

	public static String getRedefinedLocalError(String variable) {
		return SSA_REDEFINED_LOCAL_ERROR + variable;
	}
}
```

SSAException.java

```
package ssa;

public class SSAException extends Exception {

	private static final long serialVersionUID = -2278592508835339490L;

	public SSAException() {
		super();
	}

	public SSAException(String message) {
		super(message);
	}

	public SSAException(Throwable throwable) {
		super(throwable);
	}

	public SSAException(String message, Throwable throwable) {
		super(message, throwable);
	}

}
```

Type.java

```
package ssa;

public class Type {

	public static final int INT = 0;
	public static final int REAL = 1;
	public static final int ARRAY = 2;
	public static final int STRING = 3;
	public static final int FUNC = 4;
	public static final int FUNC_CALL = 5;
	public static final int OBJ = 6;
	public static final int ID = 7;
	public static final int BOOL = 8;
	public static final int UNDEFINED = 9;

	private int type;
	private int numArguments;

	public Type() {
		this(UNDEFINED, -1);
	}

	public Type(int type) {
		this(type, -1);
	}

	public Type(int type, int numArguments) {
		this.numArguments = numArguments;
		this.type = type;
	}

	public int getType() {
		return type;
	}

	public int getNumArguments() {
		return numArguments;
	}

	public String toString(int type) {
		switch (type) {
		case 0: return "Integer";
		case 1: return "Real";
		case 2: return "Array";
		case 3: return "String";
		case 4: return "Func";
		case 5: return "Func-call";
		case 6: return "Obj";
		case 7: return "ID";
		case 8: return "Bool";
		default: return "Undefined type";
		}
	}
}
```

UndefinedFuncException.java

```
package ssa;

public class UndefinedFuncException extends SSAException {

	private static final long serialVersionUID = 8371993234777055602L;

	public UndefinedFuncException() {
	}

	public UndefinedFuncException(String message) {
		super(message);
	}

	public UndefinedFuncException(Throwable cause) {
		super(cause);
	}

	public UndefinedFuncException(String message, Throwable cause) {
		super(message, cause);
	}

}
```

UndefinedObjectException.java

```
package ssa;

public class UndefinedObjectException extends SSAException {

	private static final long serialVersionUID = 5831386852769772128L;

	public UndefinedObjectException() {
	}

	public UndefinedObjectException(String message) {
		super(message);
	}

	public UndefinedObjectException(Throwable cause) {
		super(cause);
	}

	public UndefinedObjectException(String message, Throwable cause) {
		super(message, cause);
	}

}
```

UndefinedVariableException.java

```
package ssa;

public class UndefinedVariableException extends SSAException {

	private static final long serialVersionUID = 3137256440726921252L;

	public UndefinedVariableException() {
	}

	public UndefinedVariableException(String arg0) {
		super(arg0);
	}

	public UndefinedVariableException(Throwable arg0) {
		super(arg0);
	}

	public UndefinedVariableException(String arg0, Throwable arg1) {
		super(arg0, arg1);
	}

}
```

Asserter.java

```
package utility;

public class Asserter {
        public static void Assert(boolean condition, String message) {
                if (!condition) {
                        java.lang.AssertionError newerr = new java.lang.AssertionError(
                                        message);
                        throw newerr;
                }
        }
}
```

ASTViewer.java

```
package utility;

import java.awt.BorderLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.image.BufferedImage;
import java.io.BufferedOutputStream;
import java.io.IOException;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;

import antlr.collections.AST;

import com.sun.image.codec.jpeg.JPEGCodec;
import com.sun.image.codec.jpeg.JPEGImageDecoder;

public class ASTViewer {
	protected AST _ast;

	/**
	 * @param dotProgram
	 *            The full path to the dot program that comes with graphviz.
	 */
	public void show(AST ast, String dotProgram) {
		_ast = ast;
		showGraphvizHomogenousAst(dotProgram);
	}

	/**
	 * Returns the graphviz dot format code representing a graph of the ast. It
	 * seems like antlr can create multiple first level nodes, so this creates a
	 * fake root to tie them together.
	 */
	private String dotSourceForHomogenousTree(AST start) {
		String fakeRoot = new String("\"Program (fake root)\"");
		String dotSource = fakeRoot + ";";
		while (start != null) {
			dotSource += "\"" + start.hashCode() + "\";";
			String label = start.toString();
			label = label.replaceAll("\"", "\\\"");
			dotSource += "\"" + start.hashCode() + "\" [label = ";
			dotSource += "\"" + label + "\"];";
			dotSource += fakeRoot + " -- \"" + start.hashCode() + "\";";
			dotSource += dotSourceForHomogenousNode(start);
			start = start.getNextSibling();
		}
		return dotSource;
	}

	/**
	 * Returns the graphviz dot format code representing a graph of the ast
	 * starting from node. It assumes node has already been put in the dot
	 * source.
	 */
	private String dotSourceForHomogenousNode(AST node) {
		AST parent = node;
		String dotSource = "";
		if (parent.getNumberOfChildren() > 0) {
			AST child = parent.getFirstChild();
			while (child != null) {
				dotSource += "\"" + child.hashCode() + "\";";
				String label = child.toString();
				label = label.replaceAll("\"", "\\\"");
				dotSource += "\"" + child.hashCode() + "\" [label = ";
				dotSource += "\"" + label + "\"];";
				dotSource += "\"" + parent.hashCode() + "\" -- ";
```

```
                                dotSource += "\"" + child.hashCode() + "\";";
                                dotSource += dotSourceForHomogenousNode(child);
                                child = child.getNextSibling();
                        }
                }
                return dotSource;
        }

        private void showGraphvizHomogenousAst(String dotProgram) {
                String dotSource = "graph {";
                dotSource += dotSourceForHomogenousTree(_ast);
                dotSource += "}";
                String[] command = { dotProgram, "-Tjpg", "-Kdot" };
                try {
                        Process graphviz = Runtime.getRuntime().exec(command);
                        System.out.println("Ran graphviz with source: " + dotSource);
                        Thread thread = new GraphvizThread(graphviz, dotSource);
                        thread.start();
                        JPEGImageDecoder dec = JPEGCodec.createJPEGDecoder(graphviz
                                        .getInputStream());
                        BufferedImage bim = dec.decodeAsBufferedImage();
                        JFrame window = new JFrame("Homogeneous AST");
                        window.addWindowListener(new WindowAdapter() {
                                public void windowClosing(WindowEvent e) {
                                        System.exit(0);
                                }
                        });
                        JLabel label = new JLabel("", new ImageIcon(bim), JLabel.CENTER);
                        window.getContentPane().add(label, BorderLayout.CENTER);
                        window.pack();
                        window.setVisible(true);
                } catch (IOException e) {
                        System.out
                                        .println("Error creating graph of ast. Do you have graphviz installed?");
                        e.printStackTrace();
                }
        }

}

/**
 * For unknown reasons, something hangs unless the external process is output to
 * from a separate thread. Boo hiss.
 */
class GraphvizThread extends Thread {
        private Process process;
        private String source;

        public GraphvizThread(Process newProcess, String dotSource) {
                process = newProcess;
                source = dotSource;
        }

        public void run() {
                try {
                        BufferedOutputStream bufferout = new BufferedOutputStream(process
                                        .getOutputStream());
                        PrintWriter commandInput = new PrintWriter((new OutputStreamWriter(
                                        bufferout)), true);
                        commandInput.println(source);
                        commandInput.close();
                } catch (Exception e) {
                        System.err.println(e);
                }
        }
}
```

File.java

```
package utility;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class File {
	public String FileContents(String path) throws FileNotFoundException,
			IOException {
		StringBuilder builder = new StringBuilder();
		String fileName = path;
		String line;
		FileReader input = new FileReader(fileName);
		BufferedReader bufRead = new BufferedReader(input);
		line = bufRead.readLine();
		while (line != null) {
			builder.append(line);
			builder.append("\n");
			line = bufRead.readLine();
		}
		bufRead.close();
		return builder.toString();
	}

	public void SetFileContents(String path, String contents)
			throws IOException {
		FileWriter fstream = new FileWriter(path);
		BufferedWriter out = new BufferedWriter(fstream);
		out.write(contents);
		out.close();
	}

	public String FileName(String filePath) {
		return FirstGroup("([^\\/\\\\]+)\\.\\w+$", filePath);
	}

	public String StripExtension(String filePath) {
		return FirstGroup("(.+)\\.\\w+$", filePath);
	}

	private String FirstGroup(String pattern, String matchOn) {
		Matcher mtch = Pattern.compile(pattern).matcher(matchOn);
		if (mtch.matches()) {
			return mtch.group(1);
		}
		return matchOn;
	}
}
```

HomogeneousTreePrinter.java

```
package utility;

import antlr.BaseAST;

public class HomogeneousTreePrinter {
	private BaseAST thisTree;

	public HomogeneousTreePrinter(BaseAST homogeneousNode) {
		thisTree = homogeneousNode;
	}

	public void PrintTree() {
		PrintNode(thisTree, 0);
	}

	private void PrintNode(BaseAST tree, int depth) {
		for (int i = 0; i < depth; i++) {
			System.out.print(" ");
		}
		System.out.println(tree.getText());
		if (tree.getNumberOfChildren() > 0) {
			PrintNode((BaseAST) tree.getFirstChild(), depth + 1);
		}
		if (tree.getNextSibling() != null) {
			PrintNode((BaseAST) tree.getNextSibling(), depth);
		}
	}
}
```

Numbers.java

```
package utility;

import java.util.regex.Matcher;
import java.util.regex.Pattern;

public class Numbers {
	public boolean isNumeric(String input) {
		Pattern isNumericPattern = Pattern.compile("^\\d+(\\.\\d+)?|\\.\\d+$");
		Matcher isNumeric = isNumericPattern.matcher(input);
		return isNumeric.matches();
	}
}
```

BadJSExecutionException.java

```
package test;

public class BadJSExecutionException extends Exception {

	private static final long serialVersionUID = 4022576588385249170L;

	public BadJSExecutionException(String html) {
		super(
				"The result of the java script execution didn't have the required "
					+ "id 'body'. The text the server " + "returned is:\n'"
					+ html + "'");
	}

}
```