Sprite

Project Proposal

Dave Smith, Dan Benamy, John Morales, Monica Ranadive

Table of Contents

Introduction

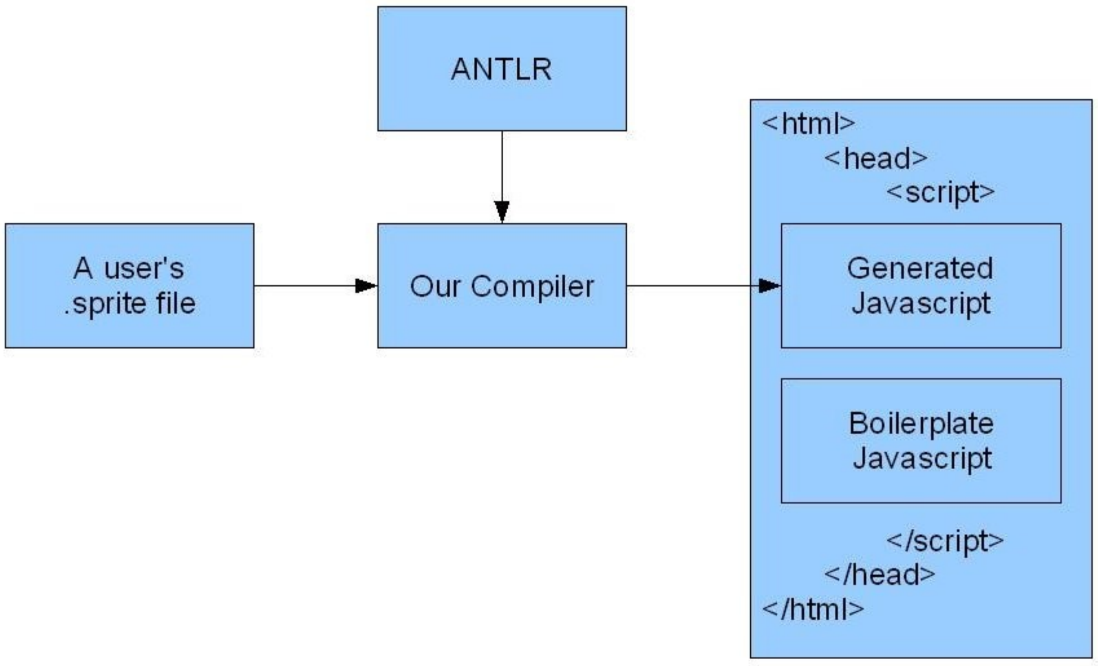
Sprite is a language designed for the purpose of rendering two dimensional images and animations within a web browser. In the computer graphics field, these images and animations are called sprites (hence our language's creative name). Typically, computer graphics programs consume numerous lines of code 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.

Motivation

The most basic graphics can be incredibly intimidating to program from a beginner's point of view. We don't want potential programmers to be dissuaded because the code looks too complicated. Sprite is a minimal, and more intuitive graphics programming language. The driving force behind Sprite is to make sprites reachable for users who may not have a degree in computer science yet are interested in graphics. A Sprite program can be written, compiled, and the end result viewed within minutes.

Structure

Our project will compile users' Sprite files to cross-browser compatible HTML files containing the necessary Javascript to display the Sprites. Our compiler is going to generate both HTML and JavaScript from the user's Sprite file. This will not only save the user from having to learn HTML in addition to JavaScript but also allow them to concentrate on the most important part – the actual coding needed to manipulate the sprites.



Why Javascript?

Flash MX is a popular way to create/view graphics and play animated video games. The reason that we chose Javascript instead of Flash is because this will allow all browsers and operating systems to run a compiled Sprite program. Unlike Flash files, our compiled programs won't need an additional plug-in to run – just a browser window! In addition, you would be able to double-click a .htm file on your hard drive and see what the sprites do while not too many systems will allow flash files to be displayed in this manner.

Types

Primitive Data Types

- Integer
- String
- Boolean
  True or False. There is no Null boolean value.

Objects

Objects may contain other objects and primitive data types. Objects specify defaults for all primitive data types. Objects will not contain constructors or subroutines.

```
Obj Foo </
    bar = 1
    rab = True
    arb = "Hello World!"
/>
```

Arrays

Two sprites are better than one, and ten are much better than two – so our language will be able to accommodate multiple sprites with a built in array object. Any primitive or object may be added to an array. We created an Add function for the Array object to make adding objects to an array more intuitive. Arrays are still indexed using [integer]. Arrays manage their own memory and grow as needed. The built in function **Count** returns the number of items in an array. **Count** only accepts Arrays.

```
testing = New Array
Add(testing, 1)
Add(testing, 2)
For (i = 0; i < Count(testing); i ++) </
    Print(testing[i])
/>
```

Case Sensitivity

Similar to JavaScript, Sprite is case sensitive. This consistency not only gives the compiler an easier job, it makes the program more understandable to the user(s). All identifiers must be spelled with the same capitalization to be properly recognized.

Functions

The functions in the Sprite programming language expect parameters by both value as well as by reference. Primitive data types such as ints and strings are passed by value while more complex parameters such as objects and array are passed to the function by reference. The parameter type does not have to be specified in the function declaration.

If the function must be run on a regular basis (such as displaying a graphic over and over again in different locations on the screen), the keyword **CallEvery** may be used. This keyword functions similarly to a while loop and takes in one parameter, the time in milliseconds for the delay between each successive call. **CallEvery** accepts any statement that evaluates to an Integer.

Functions may return a single value using the Ret keyword.

```
Func DoSomething(parameter1) CallEvery(10) </
    Print("Hello World!")
    Ret 1
/>
```

Built-In Functions

```
<- Prints the string Hello World to the page ->
Print("Hello World!")

<- Shows the given image at the given top and left position. If you call
PrintSprite multiple times with the same id, rather than printing out
several different sprites, the same sprite is simply updated.
Generically, observe PrintSprite(imagePath, id, top, left) ->
PrintSprite("Ball.jpg", 1, 75, 150)

<- Add the number 1 to the array myArray ->
Add(myArray, 1)

<- Count returns the number of items in an array ->
Print(Count(myArray))
```

Control Statements

If, Else If, Else

```
If (True) </
    DoSomething()
/> ElseIf (True) </
    DoSomething()
/> Else </
    DoSomething()
/>
```

For

```
For (i = 0; i < 10; i ++) </
    Print(i)
/>
```

Exceptions

Sprite provides no native exception object or exception handling. We expect the user to check the return codes of calls that can fail.

Operators

All operators are infix operators.

+	Addition of Integers
+	Concatenation of any 2 primitive types if either is not an Integer
-	Subtraction of Integers
*	Multiplication of Integers
/	Division of Integers
%	Modulo of Integers
&&	Logical And of Booleans
||	Logical Or of Booleans

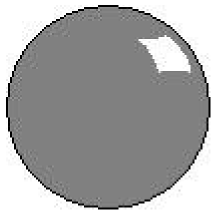
Implicit Type Casting

At this time these are the only places where implicit type casting occurs:

- When the + operator is used and either operand is not an integer, both operands are cast to strings.
- The Print function casts primitive types to strings.

Example Program: The Bouncing Ball

In this program we'll start with a picture of a ball saved in the file Ball.jpg.



We'll define an object to represent the ball, and print out the ball. Finally, every 10 milliseconds we'll update the position of the ball.

```
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
     />
     PrintSprite("Ball.jpg", newBall->id, newBall->top, newBall->left)
/>
```

You can see an example of what the BouncingBall program may compile to at

http://plt.john-morales.com:81.

Formatting

Code Blocks

Code blocks are delimited with </ and />. For example,

```
Func Foo() </
        PrintSprite("Ball.jpg", 1, 20, 20)
/>
```

Line Breaks

A line break denotes the end of a statement. Programming lines may not wrap.

Comments

Comments start at <- and end at the first ->

For example,

```
<-This is a comment->
```

Indentation

While indentation is not required for the program to compile and run, it is recommended for readability.

Memory Management

Users will not have to even think about memory management because Sprite compiles to Javascript. Javascript has built in garbage collection in all browsers, so the user is not required to worry about memory management.

Future Updates for Sprite

In the future we can build user input and other output right into the language. Eventually, you should be able to build whole cross-browser-compatible, sprite based games! Specifically, we can provide built in functions for the following:

- Creating and referencing libraries
- Knowing when users click on sprites
- Knowing where users clicked the page, especially when they click the background
- Capturing user keystrokes
- Playing sounds for the user
- Image Scaling, Rotating, and Flipping
- Image Generating and libraries

Reserved Words

All reserved words are capitalized and use only letters.

Summary

Add

Array

CallEvery

Count

If

Else

ElseIf

False

For

Func

New

Obj

Print

PrintSprite

Ret

True

Conclusion

Sprite is a simple, yet expandable, language for manipulating sprites. We plan to implement basic functionality for our language, and depending on time constraints, we will attempt to add new features to the language.