Sphere Spidermonkey Optimizations

Where to optimize? Readability versus Speed

Readability is very important. Always write clean code. Always comment your code. Later on, when you forgot your code, its easier to re-understand it. Except... where you can score more FPS (Frames per second), for example in big loops, but especially in the:

SetUpdateScript()
SetRenderScript()
SetLayerScript()

And remember: FPS is everything! (From a coder point of view)

Golden rules

Just some things I learned by trial and error.

- Use leading spaces (indentation). There is no speed penalty for using them.
- Don't leave trailing spaces. They serve no purpose and make your program slower.
- Inside code, use spaces only if needed. (more on that later)
- Don't leave empty lines. If you indented correctly, the code should still be readable.
- Don't comment inside your routine. Just 1 comment can eat up 1 fps!
- Always use ; to end a statement. Tell JS the statement has ended, don't let it guess.
- Refactor. Less code works faster. (Well... most of the time. Less operations is always better)
- Where possible use do{}while() instead of for(;;){} (more on that later)
- If you must use for(), use only constants in your evaluations.
- Don't use with(); instead, define a temporal local variable.
- Use precalculation as much as possible.
- Never divide. if you want to divide by 5, then multiply by 0.2
- If you need to divide by 2 and keep the result an integer, think of >>1 (big numbers only)
- Use a low resolution for your game.
- If possible, use Math.floor() instead of slower Math.round()
- Experiment yourself. Sometimes its not obvious why its faster.
- If you have a gradient window style, then you could blit it ones, and capture the image and use that image blit constantly (include the text!)
- Use local variables. They are looked-up first by the interpreter.

How to test the speed of a routine.

<talk here about big for loop>

For and do

```
for(var i=0;i<50;i++)
{}
```

Will execute something 50 times. I will go from 0 to 49. (i=0..49) If you don't use the value of i, then you can rewrite it as:

```
var i=50;
do
{}
while(--i);
```

This loop also will execute 50 times and is much faster than the for loop. The value of i will go from 50 down to 1 (i=50..1). Now, if you DO want to use the i, in the for loop it goes i=0..49. You can rewrite it like this:

```
var i=50-1;
do
{}
while(i--);
```

Which will make i go like i=49..0 and is still faster than the for loop. And slightly less readable.

Short references

When you use a certain Math function more than once, even if its not looped, its faster to assign the function to a temporal variable like this.

```
var Sin= Math.sin;
```

Now, instead of writing

```
var z=Math.sin(x)+Math.sin(y);
```

you write

```
var Sin= Math.sin;
var z=Sin(x)+Sin(y);
```

This is especially true for very long variables.

Local Variables are much faster

If you use GetInputPerson() and similar functions a lot, define a local variable for it, and use that variable:

var GIP=GetInputPerson();
var GIP_x= GetPersonX(GIP);
var GIP_y= GetPersonY(GIP);

note: Don't be shy when declaring local variables. If you use a slow method like GetPersonX() more than once, its faster to cache it in a local variable. Local variables are looked up first by the interpreter, so they are faster.

SLOWER:

```
var Random=Math,random;
var Floor=Math.floor;
if(tfi.h>100){tfi.x+=Floor(Random()*3)+this.wind;++tfi.y;}
```

FASTER (don't ask why, try without the {}):

```
var Random=Math,random;
var Floor=Math.floor;
if(tfi.h>100){
          tfi.x+=Floor(Random()*3)+this.wind;++tfi.y;
}
```

SLOWER:

```
 if (typeof this.offsetX == "undefined")
          var Random=Math,random;
```

FASTER (note the spaces, thats right, just the spaces):

```
 if(typeof this.offsetX=="undefined")
```

If NewOffsetX doesn't change, then still the second expression is faster:

```
tdi.x+=this.wind;
if (NewOffsetX) tdi.x = (tdi.x+NewOffsetX) % this.GSW;
```

```
tdi.x = (tdi.x+this.wind+NewOffsetX) % this.GSW;
```

References

http://www.developer-x.com/tutorials/oopjs/
http://www.crockford.com/javascript/private.html