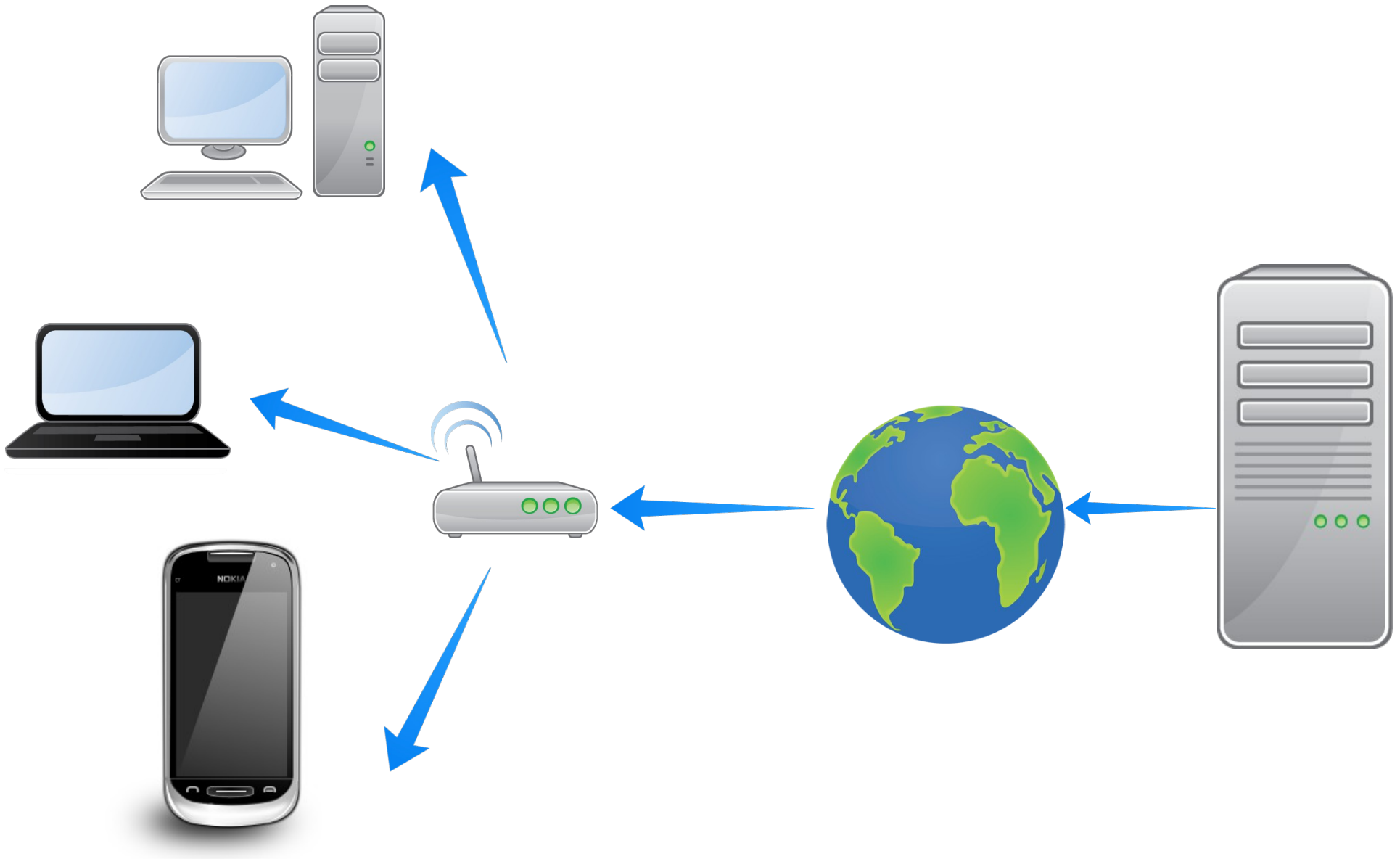
Introduccion a HTML5 para videojuegos

Javier Arevalo

Master de Programacion de videojuegos, UCM 2013

HTML5

- El estandar moderno de HTML
- Concebido para crear paginas web
  - Con contenidos ricos en multimedia
  - Con animaciones y transiciones complejas
  - Con logica compleja
  - Con capacidad de comunicacion con *servers*
  - Solido, eficiente y universalmente soportado
    - *Ahem*


NOKIA

Piezas

- Pagina web
  - HTML
- Hojas de estilo
  - CSS
- Codigo ejecutable
  - JavaScript
- Contenidos
  - PNG, MP3, etc

Servidor local

- Podemos desarrollar nuestro HTML5 en local, sin un servidor en Internet
  - Los navegadores aceptan file://blahblah.html
  - Pero no nos sirve para archivos de sonido, de datos, o imagenes cargadas dinamicamente
- Servidor web local
  - Apache, nginx, httpd, XAMPP/WAMPP
  - php -S, python -m SimpleHttpServer, http-server

HTML

- Archivo de texto con *tags* para marcar *elementos*
  - <html>, <head>, <body>
  - <img>, <audio>, <canvas>, <svg>
  - <script>
  - <h1>, <a>, <p>, ,
- Los tags pueden tener atributos
  - <a **href="http://blahblah.com/yeah.html"**>

HTML

- Uso de tags
  - Conteniendo texto: Texto...
  - Sueltos:
  - Sin cierre: <img src="...">
- Atributos especiales
  - id: blahblah...
  - Class: ...

Ejemplo HTML

```
<!DOCTYPE html>
<html>
<head>
 <title>Title</title>
 <style>
    body { background: black; color: white; }
 </style>
</head>
<body>
<h1>Hey this is my page</h1>
<script>
window.onload = function() {
    // Your code goes here...
}
</script>
</body>
</html>
```

Doctype

```
<!DOCTYPE html>
<html>
<head>
 <title>Title</title>
 <style>
    body { background: black; color: white; }
 </style>
</head>
<body>
<h1>Hey this is my page</h1>
<script>
window.onload = function() {
    // Your code goes here...
}
</script>
</body>
</html>
```

Head

```
<!DOCTYPE html>
<html>
<head>
 <title>Title</title>
 <style>
    body { background: black; color: white; }
 </style>
</head>
<body>
<h1>Hey this is my page</h1>
<script>
window.onload = function() {
    // Your code goes here...
}
</script>
</body>
</html>
```

Body

```
<!DOCTYPE html>
<html>
<head>
 <title>Title</title>
 <style>
    body { background: black; color: white; }
 </style>
</head>
<body>
<h1>Hey this is my page</h1>
<script>
window.onload = function() {
    // Your code goes here...
}
</script>
</body>
</html>
```

Estilos CSS

- Formato:
  - selector { estilo1: valor; estilo2: valor; ... }
  - Selector
    - Define a que elementos de la pagina se les va a aplicar ese estilo. Combinables y agrupables.
      - tag { ... }
      - #id { ... }
      - .clase { ... }
  - Estilos y valores
    - Muchos: background, color, font, border, ...

Estilos CSS

- Siempre dentro de <head>!
- Se pueden meter dentro de tags <style>

```
<style>
    body { background: black; color: white; }
</style>
```

- O como archivos separados

```
<link rel="stylesheet" type="text/css" href="styles.css">
```

Estilos CSS

- No quiero engañar a nadie
  - CSS es una locura
  - Docenas de estilos muy heterogeneos
  - Se acumulan y sobreescriben unos a otros sobre cada tag del documento
  - Control de posicionamiento complejisimo y nada intuitivo
    - Centrar verticalmente es toda una odisea
  - Extensiones y particularidades de cada navegador

JavaScript

- Todo un lenguaje de programacion
  - Desarrollado deprisa y corriendo (2 semanas?)
  - Inspirado en Java y C
  - Evoluciona a trompicones a partir de Ecmascript
- Apenas tiene burocracia
- Tipos dinamicos para las variables
  - Transformaciones automaticas a lo loco
- Todo son objetos ...incluso las funciones!

Programando JavaScript

- JavaScript en el navegador
  - Archivo ".js" cargado con <script src="file.js" />
  - Contenido en tag <script> ... </script>
  - En la consola!
    - Ctrl+Shift+J en Chrome
- Como lenguaje de script en programas nativos
  - Node, Riak, Cocos2d-x, etc usan V8 o SpiderMonkey

Variables y valores

numero = 3; (tambien hexadecimales: 0x23af)

booleano = true;

cadena = "hola" + " mundo";

array = [1, 2, 3, 4, 78, -1000];

objeto = { campo: "valor" };

- Se accede a ellos como **objeto.campo**

null, undefined, NaN

- Los arrays y objetos pueden contener a su vez valores de cualquier tipo, recursivamente

Expresiones

```
a = 3;
a += 4;
b = a + 7;
c = "El numero " + b;
```

- Operadores:
  - +, -, *, /, %, |, &, ^, <<, >>, ++, --
  - ||, &&, !, ==, ===, !=, !==, <, <=, >, >=
  - typeof, in,

Control

- Bloque de sentencias

```
{
    a = 3;
    b = 4;
}
```

  - Se comporta como una unica sentencia
  - No confundir con objeto:

    { a: 3 } vs { a = 3; }
  - Los ';' son opcionales en muchos casos

Control

```
if (a == 3) {
  b = "el numero es tres!";
} else {
  b = "el numero NO es tres!";
}
```

```
a = 0;
while (a != 3) {
  a++;
}
```

```
b = 0;
for (a = 0; a != 3; a++) {
  b = b + a;
  if (b > 5)
    break;
}
```

Funciones

```
function mifuncion() {
  console.log("Me han llamado... de todo!");
}
```

```
function sumar(a, b) {
  var suma = 0;
  while (a <= b) {
    suma += a;
    a++;
  }
  return suma;
}

console.log(sumar(1, 3));
console.log(sumar(10, 100));
```

Variables

- Las variables no se declaran, se crean en el momento en que se les asigna un valor

```
console.log("La variable a vale " + a); // error
a = 3;
console.log("La variable a vale " + a);
```

- Asi creadas, son variables globales a TODA la pagina web
  - Es muy facil cometer errores o pisarse unas partes del codigo a otras

Variables locales

- Dentro de una funcion puede haber variables locales a esa funcion
  - Solo existen dentro de la funcion
  - Los parametros automaticamente son variables locales

```
function sumar(a, b) {
  var suma = 0;
  while (a <= b) {
    suma += a;
    a++;
  }
  return suma;
}
```

Variables locales

- Se pueden poner en cualquier parte
  - Automaticamente se crean al principio de la funcion
    - NO del bloque anidado

```
function mifuncion(a) {
    console.log("vale " + valor);
    if (a == 3) {
        var valor = 5;
    }
}
→ vale undefined
```

```
function mifuncion(a) {
    var valor;
    console.log("vale " + valor);
    if (a == 3) {
        valor = 5;
    }
}
```

Variables locales

- Si usamos 'var' fuera de una funcion, estamos creando una variable global
  - En realidad, la estamos metiendo dentro del 'objeto global'
    - En los navegadores, 'window'

```
console.log(valor);
→ ReferenceError: valor is not defined
window.valor = 3;
console.log(valor);
→ 3
var valor2 = 4;
console.log(window.valor2);
→ 4
```

Funciones anonimas

- Una funcion es en realidad un valor como cualquier otro
  - Lo podemos asignar a una variable

```
sumar = function(a, b) {
  var suma = 0;
  while (a <= b) {
    suma += a;
    a++;
  }
  return suma;
}
sumar(3,4);
MiFuncionSumar = sumar;
MiFuncionSumar(1, 10);
```

Objetos como namespaces

- Podemos crear objetos con funciones dentro
  - Agrupadas en 'modulos'. P.ej. Math

```
MisFunciones = {
  sumar : function(a, b) {
    var suma = 0;
    while (a <= b) {
      suma += a;
      a++;
    }
    return suma;
  },
  pintarSuma : function(a, b) {
    console.log(MisFunciones.sumar(a, b));
}
MisFunciones.pintarSuma(3,4);
```

Objetos como clases

- Hay muchas formas, esta es muy sencilla:

```
Jugador = function(nombre, nivel) {
  this.nombre = nombre;
  this.nivel = nivel;
  this.maxVida = 200 + nivel*100;
  this.vida = this.maxVida;
}

Jugador.prototype.SubirNivel = function() {
  this.nivel++;
  this.maxVida = this.maxVida + 100;
  this.vida = maxVida;
}

var ThePlayer = new Jugador("Conan", 1);
// ...
ThePlayer.SubirNivel();
```

Polimorfismo

```
Perro = function(nombre) {
  this.nombre = nombre;
}
Perro.prototype.HacerRuido = function() {
  Console.log(this.nombre + " hace Guau!");
}

Gato = function(nombre) {
  this.nombre = nombre;
}
Gato.prototype.HacerRuido = function() {
  Console.log(this.nombre + " hace Miau!");
}

var fido = new Perro("Fido");
var fifi = new Gato("Fifi");
fido.HacerRuido();
fifi.HacerRuido();
```

Callbacks

- Pasar funciones como parametro a otras funciones
  - La funcion pasada puede ser anonima

```
sumar = function(a, b) { return a + b; }
restar = function(a, b) { return a - b; }

function HacerOperacion(a, b, f) {
  console.log(f(a, b));
}

HacerOperacion(3, 4, sumar);
HacerOperacion(6, 2, restar);
HacerOperacion(2, 3, function(a,b) { return a*b; });
```

Callbacks (2)

- La funcion se podria ejecutar en un futuro
  - Ya tenemos un mecanismo para animar cosas!

```
avisar = function() {
  console.log("Por fin aparezco!");
}

setTimeout(avisar, 5000);

funcionPlasta = function() {
  console.log("De mi no te libras!");
}

setInterval(funcionPlasta, 1000);
```

Closures

- Funcion local a otra funcion
  - Puede acceder a las variables de la funcion padre

```
function DarLaPlasta() {
  var count = 0;
  var funcionPlasta = function() {
    count++;
    console.log("He aparecido " + count + " veces!");
    if (count > 5)
      clearInterval(intervalId);
  }
  var intervalId = setInterval(funcionPlasta, 1000);
}
```

Estas variables locales siguen 'vivas' mientras la closure las necesite