UNIVERSIDAD DE BELGRANO

Las tesinas de Belgrano

Facultad de Ingeniería y Tecnología Informática
Carrera Licenciatura en Sistemas de Información

PhoneGap y HTML5: la solución para codificar aplicaciones multiplataforma

N° 810 Gonzalo Becker León

Tutor: Ing. Sergio Omar Aguilera
Directora Carrera: Lic. Paula Angeleri

Departamento de Investigaciones
Fecha defensa de tesina: 18 de mayo de 2015

Universidad de Belgrano
Zabala 1837 (C1426DQ6)
Ciudad Autónoma de Buenos Aires - Argentina
Tel.: 011-4788-5400 int. 2533
e-mail: invest@ub.edu.ar
url: http://www.ub.edu.ar/investigaciones

ABSTRACT

La gran parte de la programación web está integrada por elementos de HTML (lenguaje de marcado para la elaboración de páginas web), JavaScript (es un lenguaje de programación interpretado, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas) y CSS (lenguaje usado para definir y crear la presentación de un documento estructurado), creando sitios dinámicos, agradables a la vista y adaptables a diferentes tipos de dispositivos. Y esto, sumado al framework Phonegap (framework para el desarrollo de aplicaciones móviles que permite desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas), brinda posibilidades muy convenientes para programadores.

Para el caso de estudio de esta tesina (desarrollar una aplicación móvil multiplataforma y multiresolución, utilizando lenguajes de programación web, y sensible de ser ejecutado en diferentes plataformas de Sistemas Operativos Mobile y Hardware de Celulares, además cualquier navegador, sea móvil o de escritorio, sin importar la resolución de la pantalla), se decidió limitar el testeo a navegadores móviles y de escritorio, y las plataformas móviles Android y iOS, las más utilizadas por los usuarios.

A lo largo de este trabajo, se desarrollara toda la información para comprender el desarrollo de un juego móvil isométrico y multiresolución, sin aplicativos ni extensiones extras, y como portabilizarlo a las plataformas en las que se desea ejecutar el juego.

Agradecimientos

Agradezco profundamente a todos los profesores de la carrera que me motivaron e impulsaron a interesarme en sus materias, al personal administrativo por su ayuda y predisposición, a Sergio Aguilera por su asistencia y motivación constante durante el desarrollo de esta tesina, a mis compañeros de curso por la buena onda para crear un clima agradable para aprender y a mi familia por apoyarme durante todo el proceso.

TABLA DE CONTENIDOS

TABLA DE ILUSTRACIONES

INTRODUCCIÓN

PLANTEAMIENTO E INTRODUCCIÓN DEL PROBLEMA

Hace un tiempo, cuando alguien decía navegar por internet, chequear la casilla de correo electrónico o leer un diario online, la imagen que a todos evocaba era la de estar sentado en una silla frente a un monitor, con teclado y mouse, e ingresar al navegador o programa instalado para realizar estas actividades.

La realidad actual es que este modelo fue relegado parcialmente, debido a la aparición de un nuevo paradigma: la navegación móvil.

La introducción de internet móvil celular y tecnologías de conexión inalámbrica (como WiFi, por ejemplo) ha propiciado la aparición y popularidad de los smartphones. Aunque la idea de un dispositivo que combinaba telefonía con computación fue conceptualizada hace más de cuarenta años, no fue hasta el comienzo del siglo 21 que comenzaron a aparecer dispositivos que transformaban al celular en una unidad de trabajo y ocio integral portátil, evolucionando hasta poseer prestaciones similares a la de una computadora de escritorio.

Cuando se plantea la pregunta de dónde reside el futuro de la programación, o incluso de la informática como un conjunto, es bastante improbable que alguien discuta que ese futuro es la ***programación móvil***. Esto se debe a que esta premisa no se basa en una suposición; nos encontramos actualmente en el desarrollo de la misma, es una realidad, solo basta con algunos ejemplos para convencerse de actividades que podemos realizar en nuestros dispositivos móviles:

- Posibilidad de ver películas y series online
- Lectura de libros, revistas y diarios directamente en el navegador
- Servicio de almacenamiento de archivos en Internet
- Servicios de ***hosting*** de videos e imágenes en servidores dedicados
- ***Streaming*** de música
- Realización de pagos y operaciones bancarias seguras
- Conexión integral con redes sociales

Estos son meros ejemplos del porqué del gran boom de dispositivos móviles, tanto en ámbitos empresariales como para uso meramente personal. El gran atractivo es la posibilidad de disponer de todos estos servicios y muchos más, pudiendo elegir el proveedor de los mismos, todo desde la palma de la mano, desde cualquier lugar con acceso a una conexión de internet.

Ante un sector tan prolífero como el móvil, numerosas empresas se decidieron a proveer su propio producto, ya sean dispositivos o sistemas operativos móviles. Esto produce que el mercado, a pesar de tener empresas dominantes del sector, cuente con numerosas opciones diferentes para el usuario final.

Pero una heterogeneidad de productos resulta en una heterogeneidad de plataformas, y por ende, diferente programación para las aplicaciones de terceros. Esto presupone un problema para los desarrolladores aplicaciones, ya que si quieren que su desarrollo llegue a la mayoría de los usuarios de aplicaciones móviles, deben presentarla en la mayor cantidad de plataformas posibles. Pero esto significaría una necesidad de programar en tantos lenguajes como plataformas se pretende soportar.

IDEA DIRECTRIZ DE LA TESINA

La tentativa solución al problema del desarrollo en múltiples plataformas vendría de la mano de una nueva plataforma de desarrollo, llamada *PhoneGap* (http://phonegap.com/).

PhoneGap es un framework para el desarrollo de aplicaciones móviles que permite a los programadores desarrollar aplicaciones para dispositivos móviles utilizando herramientas genéricas tales como JavaScript, HTML5 y CSS3.

Esto significa que programando con estas herramientas (teniendo en cuenta ciertas reglas de diseño) se puede crear un código web funcional, que puede ser ***portable*** a virtualmente todos los sistemas operativos móviles actuales, creando un archivo instalable en cada plataforma, lo que significa que cumple con los estándares de la gran mayoría de ***marketplaces*** de aplicaciones móviles.

Estas capacidades de la herramienta presuponen grandes beneficios a la hora de programar, diseñar, exportar, mantener y dar soporte a la aplicación, ya que solo se trabaja con un mismo código, reflejando un cambio en todas las plataformas simultáneamente.

Y una de las principales ventajas del producto es que no es necesario desarrollar las aplicaciones en un framework determinado. Se puede utilizar cualquier ambiente de desarrollo de aplicaciones web, y con el código final se lo procesa con Phonegap para producir instalables para las plataformas que se quieran soportar.

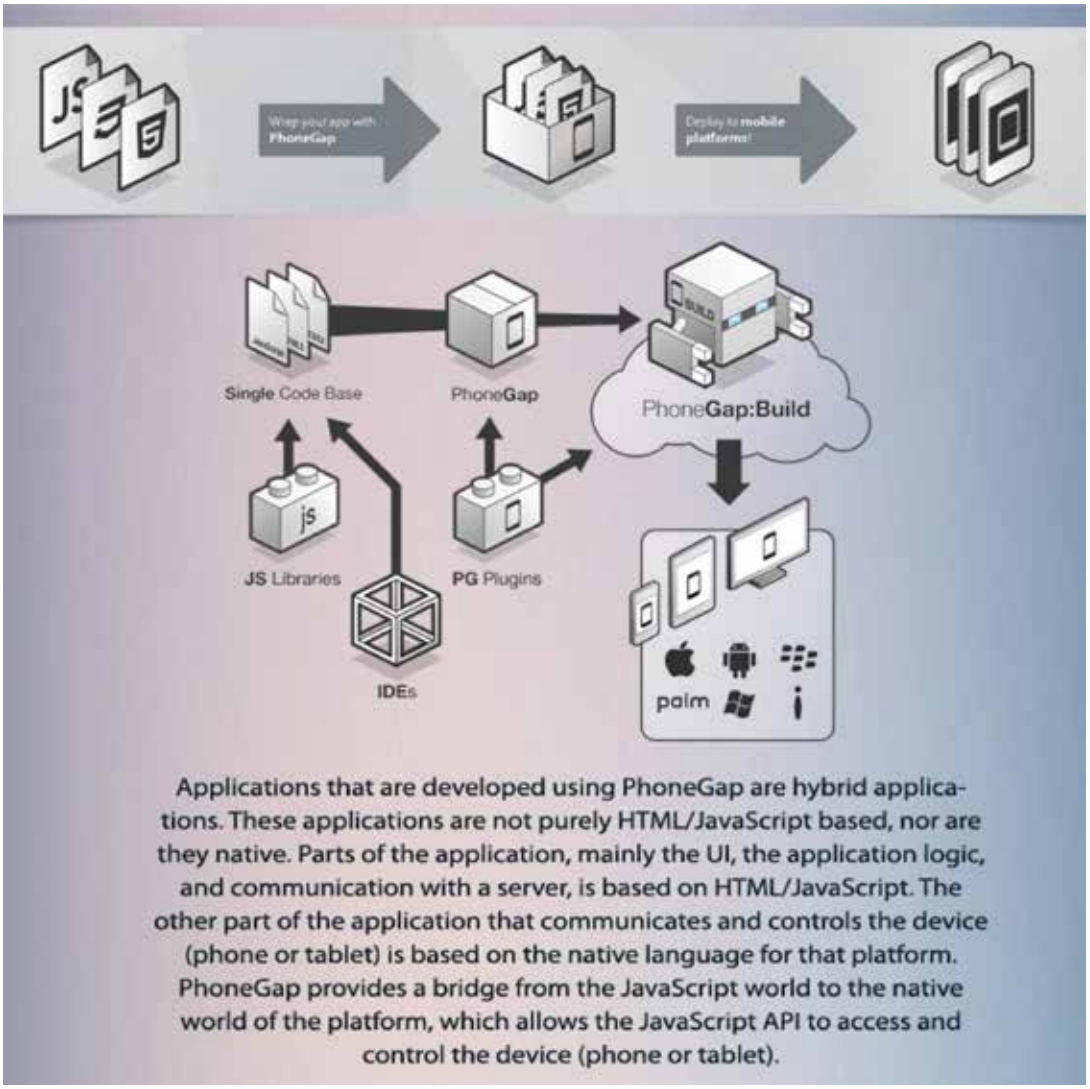


ILUSTRACIÓN 1 - ILUSTRACIÓN SOBRE FUNCIONAMIENTO DE PHONEGAP (QUANTUMCLOUD, 2013)

HIPÓTESIS DE TRABAJO

Esta tesina tiene como propósito de estudio la creación de un juego multiplataforma, aprovechando la versatilidad de la programación de aplicaciones web y móviles utilizando HTML5, CSS3 y JavaScript como herramientas de programación. Esta investigación que se presenta se acoge con la finalidad de establecer un marco teórico-práctico completo e íntegro de los elementos que se utilizarán, analizando:

- HTML5
  - Elementos básicos
  - Diferencias con versiones anteriores
  - Semántica
  - ***APIs***
  - Integración con otras plataformas

- CSS3
  - Sintaxis
  - Notaciones y utilización
  - Soporte de otras plataformas

- JavaScript
  - Propiedades
  - Funcionalidad
  - Dinámica
  - Sintaxis
  - Utilización

- Aprovechamiento de los componentes de los celulares
  - Conexión
  - ***Geolocalización***
  - Navegador del dispositivo

Asimismo, se realizará un análisis comparativo con la programación nativa de aplicaciones móviles en algunas de las plataformas más populares del género.

El objetivo integrador final de este trabajo es lograr la realización de un código único íntegramente programado con HTML5, CSS3 y JavaScript. A partir de este código, utilizando PhoneGap, se desarrollarían instalables de aplicaciones para Android e iOS, totalmente funcionales, aprovechando nativamente de las prestaciones del dispositivo. Además, este código podrá ser utilizado para un sitio web con diseño adaptable a las diferentes resoluciones de los dispositivos.

Entonces, la hipótesis a probar sería la posibilidad de desarrollar una aplicación móvil multiplataforma y multiresolución, utilizando lenguajes de programación web globales, junto con herramientas normalizadas y convencionales; tal que permita que un mismo código pueda ejecutarse en diferentes plataformas de Sistemas Operativos Mobile y Hardware de Celulares, además de ser sensibles de ser ejecutados desde cualquier navegador, sea móvil o de escritorio, sin importar la resolución de la pantalla.

OBJETIVO GENERAL Y ESPECÍFICOS

El objetivo de este proyecto consiste en la creación de un juego web ***isométrico*** (representando un escenario de 2 dimensiones como 3, visualizando el mapa en forma romboidal) de exploración de calabozos, utilizando manejo de iluminación, estructuración de mapas, con terrenos, paredes y elementos de decoración mediante ***sprites***, y con controles físicos y mediante pantalla táctil. Este template de juego web luego será utilizado para la realización de aplicaciones móviles para las plataformas Android y iOS, además de ser un juego web adaptable a cualquier resolución de pantalla.

METODOLOGÍA

Para esta tesina, se aplicará una metodología deductiva de integración de aplicaciones en el desarrollo de software en plataformas móviles y web.

JUSTIFICACIÓN DEL TRABAJO

Como se planteó anteriormente, en el sector de dispositivos móviles, las principales empresas se decidieron a lanzar una plataforma propietaria para sus dispositivos. En consecuencia, esto causa que haya numerosas opciones diferentes para el usuario final.

Pero el hecho de que se cuente con numerosos productos en el mercado, conlleva que se presenten diferentes plataformas, y por ende, diferente programación para las aplicaciones de terceros. Este no es un escenario ideal para desarrolladores de aplicaciones móviles, que si pretenden que su producto llegue a la mayoría de los usuarios de aplicaciones móviles, deberán programarla para la mayor cantidad de plataformas posibles.

En un escenario básico, un desarrollador móvil con experiencia en el rubro, tarda un promedio de 18 semanas en crear una primera versión de una aplicación simple pero completamente funcional. Dentro de este período, se comprenden la creación y configuración de almacenamiento, lógica del servidor, integración de datos, sincronización e interfaz de usuario. En el caso de tener que interactuar con otro ***framework*** o tener que interactuar con información bancaria, el tiempo de programación y trámites administrativos aumenta. (Kinvey, 2013)

En el supuesto caso que el desarrollador de este escenario planteado quiera que su aplicación esté disponible para Android, iOS y Windows Phone (3 de los ***sistemas operativos móviles*** más populares), el tiempo de desarrollo se multiplicaría por 3. Pero también hay que tener en cuenta que estas aplicaciones requieren un mantenimiento constante, y en un mercado tan competitivo como el móvil, debe estar a la vanguardia de la tecnología. Todos estos factores producen que el costo de soporte, mantenimiento y ***gestión de versiones*** se triplique. Y todo esto sin contar lo que presupondría poner en funcionamiento una versión web de la aplicación.

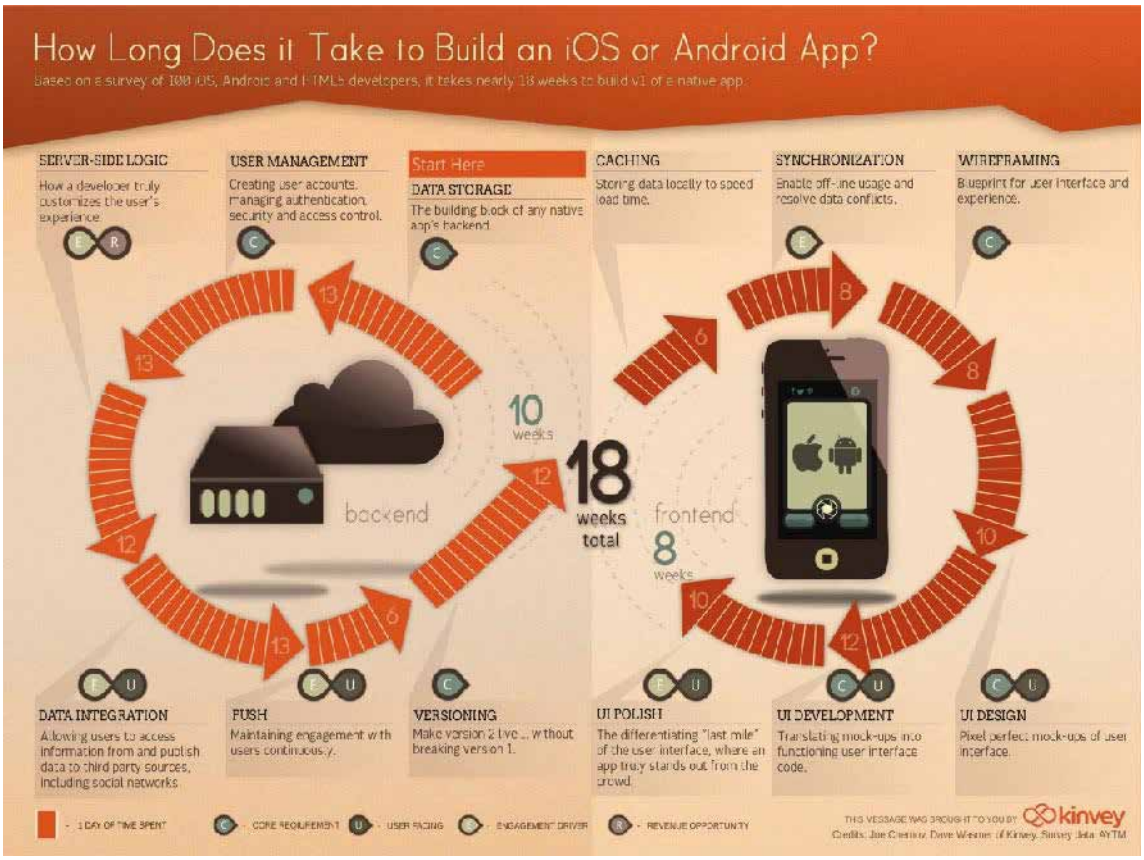


ILUSTRACIÓN 2 - INFOGRAFÍA DE DESARROLLO WEB (KINVEY, 2013)

DELIMITACIONES O ALCANCES DE LA TESINA

El proyecto consistirá en el análisis de la plataforma de desarrollo PhoneGap, en conjunto con los conceptos y programación de HTML5, CSS3 y JavaScript, redactando una introducción al tema, para dar paso a un estudio más profundo sobre las características específicas de la plataforma, y su interacción con los elementos de programación precedentemente mencionados. Con esta información se establecerán las bases para el posterior desarrollo de un juego multiplataforma desarrollado enteramente en HTML5, CSS3 y JavaScript, además de una versión web ***multiresolución*** de la misma.

MARCO TEÓRICO

El estudio teórico objetivo comprende variadas tecnologías web complementarias cuyo objetivo es lograr un producto funcional, eficiente y atractivo. Pero un fácil entendimiento, se las diseccionará individualmente en primera instancia, para luego analizar su ensamblaje e interacción entre sí.

HTML5

¿QUÉ ES HTML5?

HTML5 es un lenguaje ***markup*** (de hecho, las siglas de HTML significan Hyper Text Markup Language) usado para estructurar y presentar el contenido para la web. Es uno de los aspectos fundamentales para el funcionamiento de los sitios, pero no es el primero. Es de hecho la quinta revisión del estándar que fue creado en 1990. A fines del año pasado, la W3C (http://www.w3.org/) la recomendó para transformarse en el estándar a ser usado en el desarrollo de proyectos venideros. Con HTML5, tenemos otras posibilidades para explotar usando menos recursos.

Este se trata de un sistema para formatear el ***layout*** de nuestras páginas, así como hacer algunos ajustes a su aspecto. Con HTML5, los navegadores como Firefox, Chrome, Explorer, Safari, entre otros pueden saber cómo mostrar una determinada página web, saber dónde están los elementos, dónde poner las imágenes, dónde ubicar el texto. En este sentido, el HTML5 no se diferencia demasiado de su predecesor. La diferencia principal, sin embargo, es el nivel de sofisticación del código que se puede construir usando HTML5. (Pavan, 2013)

UN POCO DE HISTORIA

HTML se remonta a mucho tiempo atrás. Fue publicado por primera vez como un proyecto de Internet en 1993. Los años 90 vieron una enorme cantidad de actividad en torno a HTML, con la versión 2.0, las versiones 3.2 y 4.0 todas en el mismo año, y finalmente, en 1999, la versión 4.01. En el curso de su desarrollo, el World Wide Web Consortium (W3C) asumió el control de la especificación.

Sin embargo, después de la rápida entrega de estas cuatro versiones, HTML fue ampliamente considerado como un callejón sin salida, desplazando el enfoque de los estándares web a XML y XHTML, quedando HTML en un segundo plano. A pesar de esto, la mayoría del contenido en la web continuó siendo HTML. Para habilitar nuevas aplicaciones web y hacer frente a las deficiencias de HTML, se necesitaban nuevas características y especificaciones.

Queriendo llevar la plataforma web a un nuevo nivel, se creó el Web Hypertext Application Working Group (http://wiki.whatwg.org/wiki/FAQ) en 2004. Esta agrupación fue la responsable de la especificación HTML5. También comenzó a trabajar en nuevas características orientadas específicamente a paliar deficiencias en aplicaciones web. Fue durante esta época que el término ***Web 2.0*** fue acuñado. Y así fue como sitios web estáticos dieron paso a sitios más dinámicos y sociales.

El W3C se involucró con HTML de nuevo en 2006, publicando el primer borrador de trabajo de HTML5 en 2008. En contraposición, el grupo de trabajo de XHTML 2 se detuvo en 2009. Debido a que HTML5

resuelve problemas muy prácticos, los fabricantes de navegadores están implementando sus nuevas características, a pesar de que la especificación no ha sido completamente cerrada. La experimentación de los navegadores retroalimenta y mejora la especificación. HTML5 está evolucionando rápidamente para hacer frente a las mejoras reales y prácticas a la plataforma web. (Peter Lubbers, 2011)

COMPONENTES EXCLUSIVOS HTML5

CANVAS

HTML5 define el elemento <canvas> como “un lienzo de mapa de bits dependiente de la resolución, que se puede utilizar para los gráficos de transformación, gráficos de juegos u otras imágenes visuales ***on-the-fly***.”. El canvas es un rectángulo en la página donde se puede usar JavaScript para dibujar todo lo que quieras. HTML5 define también un conjunto de funciones (“la API de canvas”) para dibujar formas, definir de caminos, crear gradientes, y aplicar transformaciones.

Si un navegador es compatible con la API de canvas, el ***objeto DOM*** que se crea para representar una serie de elementos <*canvas*> tendrán un método *getContext ()*. Si un navegador no soporta la API de canvas, el objeto DOM que crea para un <*canvas*> sólo tendrá el conjunto de elementos de propiedades comunes, pero no específico de canvas.

```
<!DOCTYPE HTML>
<html>
  <head>
    <style>
      body {
        margin: 0px;
        padding: 0px;
      }
    </style>
  </head>
  <body>
    <canvas id="myCanvas" width="578" height="200"></canvas>
    <script>
      var canvas = document.getElementById('myCanvas');
      var context = canvas.getContext('2d');

      // Begin custom shape
      context.beginPath();
      context.moveTo(170, 80);
      context.bezierCurveTo(130, 100, 130, 150, 230, 150);
      context.bezierCurveTo(250, 180, 320, 180, 340, 150);
      context.bezierCurveTo(420, 150, 420, 120, 390, 100);
      context.bezierCurveTo(430, 40, 370, 30, 340, 50);
      context.bezierCurveTo(320, 5, 250, 20, 250, 50);
      context.bezierCurveTo(200, 5, 150, 20, 170, 80);

      // complete custom shape
      context.closePath();
      context.lineWidth = 5;
      context.strokeStyle = 'blue';
      context.stroke();
    </script>
  </body>
</html>
```

ILUSTRACIÓN 3 – EJEMPLO DE UN GRÁFICO DIBUJADO EN CANVAS (DESARROLLO DE AUTOR)

CANVAS TEXT

Incluso si un navegador es compatible con la API de canvas, podría no soportar el texto de la API de canvas text. La API de canvas creció con el tiempo, y las funciones de texto se añadieron al final del desarrollo.

Si un navegador es compatible con la API de canvas, el objeto DOM crea para representar un <*canvas*> tendrá el método *getContext()*. Si un navegador no soporta la API de canvas, el objeto DOM que crea para un <*canvas*> sólo tendrá el conjunto de elementos de propiedades comunes, pero no específico de canvas.

VIDEO

HTML5 define un nuevo elemento <*video*> para la incorporación de vídeo en páginas web o desarrollos. ***Embeber*** videos solía ser imposible sin plugins de terceros, tales como *QuickTime* o *Adobe Flash*.

El elemento <*video*> está diseñado para poder utilizarse sin ningún script de detección. Se puede especificar varios archivos de vídeo y navegadores que soportan video HTML5 elegirá uno basado en lo que los formatos de vídeo que soportan.

Los navegadores que no soportan HTML5 video ignorarán los elementos <*video*> por completo, pero se puede se puede aprovechar esto e indicarle que reproduzca el vídeo a través de un plugin de terceros en su lugar.

Los formatos soportados son:

- Ogg Theora
- H.264 video
- WebM

ILUSTRACIÓN 4 – VIDEO CON BOTONES HECHOS EN HTML, SIN REPRODUCTOR EXTERNO

LOCAL STORAGE

Almacenamiento HTML5 proporciona una forma para que los sitios web almacenen información en las computadoras cliente y recuperarla más tarde. El concepto es similar a las ***cookies***, pero está diseñado para grandes cantidades de información. Las cookies son limitadas en tamaño, y su navegador las envía de vuelta al servidor web cada vez que se solicita una nueva página (que tarda más tiempo y utiliza valioso ancho de banda). Almacenamiento HTML5 se queda en el equipo, y los sitios web pueden acceder a él con JavaScript después de cargar la página.

Dentro del navegador, cualquier sitio web puede leer y modificar sus valores, pero los sitios no pueden acceder a los valores almacenados por otros sitios. Esto se conoce como restricción del mismo origen.

WEB WORKERS

Los web workers proporcionan una forma estándar en los navegadores web para ejecutar JavaScript en el ***background***. Con los web workers, se puede generar múltiples “hilos” que corren prácticamente al mismo tiempo. Estos hilos en backgrond pueden hacer cálculos matemáticos complejos, realizar solicitudes de red, o acceder al almacenamiento local, mientras que la página web principal responde al desplazamiento del usuario, clics, o escritura del usuario.

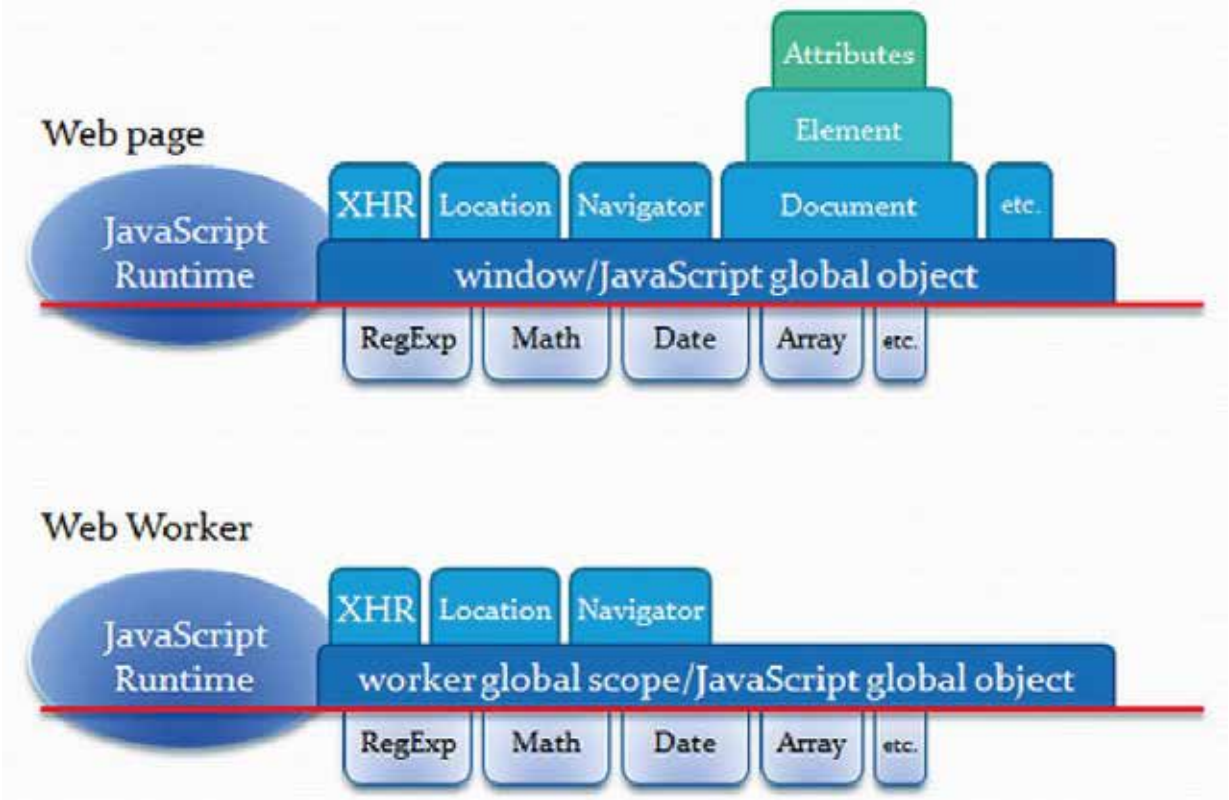


ILUSTRACIÓN 5 – ALCANCE DEL WEBWORKER, COMPARADO CON EL HILO JAVASCRIPT DE LA VENTANA

APLICACIONES WEB FUERA DE LINEA

Las aplicaciones web fuera de línea comienzan como “en línea”. La primera vez que se visita un sitio web con modo offline habilitado, el servidor web le indica al navegador qué archivos que necesita para trabajar sin conexión. Estos archivos pueden ser cualquier cosa - HTML, JavaScript, imágenes, e incluso vídeos. Una vez que su navegador descarga todos los archivos necesarios, puede visitar ese sitio web, incluso si no se está conectado a Internet. El navegador se dará cuenta de que se está fuera de línea y utilizará los archivos que ya ha descargado. Al volver al modo online de nuevo, cualquier cambio realizado se puede subir al servidor web.

GEOLOCALIZACIÓN

Geolocalización es la capacidad de averiguar dónde se encuentra en el mundo el usuario y (opcionalmente) compartir esa información con personas de su confianza. Hay más de una manera de averiguar dónde se encuentra - su dirección IP, la conexión inalámbrica de la red, desde qué antena de telefonía móvil el teléfono se está comunicando, o mediante hardware GPS dedicado que calcula latitud y la longitud de la información enviada por satélites especializados.

NUEVOS TIPOS DE INPUT

Un formulario web dentro de una página web permite al usuario introducir datos los cuales son enviados a un servidor para ser procesados. En los formularios web los usuarios los llenan usando casillas de selección, botones de opción, o campos de texto.

HTML5 define 13 nuevos tipos de inputs para ser utilizados en los forms. Estos son:

- *<input type="search">* para búsquedas
- *<input type="number">* para selección de números
- *<input type="range">* para sliders
- *<input type="color">* para selector de colores
- *<input type="tel">* para números telefónicos
- *<input type="url">* para sitios web
- *<input type="email">* para direcciones de e-mail
- *<input type="date">* para selectores de fecha
- *<input type="month">* para selector de meses
- *<input type="week">* para selector de semanas
- *<input type="time">* para selector de horarios
- *<input type="datetime">* para selector de hora y fecha específicas
- *<input type="datetime-local">* para fechas y horarios en hora local

ILUSTRACIÓN 6 – DIFERENCIA ENTRE UN INPUT HTML4 Y EL INPUT DE FECHAS HTML5 (KOCAK, 2013)

PLACEHOLDER

Un placeholder es un texto que se muestra en el interior del campo de entrada, siempre y cuando el campo está vacío y no se centra. En cuanto hace clic en el campo de entrada, el texto del placeholder desaparece.

ILUSTRACIÓN 7 - EJEMPLO DE PLACEHOLDER

AUTOFOCO DE FORMS

HTML5 introduce un atributo autofocus en todos los controles de formulario web. El atributo mueve el foco a un campo de entrada particular. Pero debido a que es sólo markup en lugar de un script, el comportamiento será consistente a través de todos los sitios web. Además, los proveedores de navegadores (o autores de extensión) pueden ofrecer a los usuarios una manera de desactivar el comportamiento de auto-enfoque.

MICRODATA

Microdata es una forma estándar para proporcionar semántica adicional en las páginas web. Por ejemplo, puede utilizar las microdata para declarar que una fotografía está disponible bajo una licencia específica. Los navegadores, extensiones del navegador, y los motores de búsqueda pueden convertir las microdata HTML5 en una vCard, un formato estándar para el intercambio de información de contacto. También se puede definir vocabularios propios de microdata.

El estándar de microdata HTML5 incluye tanto el formato HTML (sobre todo para los motores de búsqueda) y un conjunto de funciones DOM (sobre todo para los navegadores).

HISTORY API

History API HTML5 es una manera estandarizada de manipular el historial del navegador a través de scripts. Parte de este API - navegar el historial- ha estado disponible en las versiones anteriores de HTML. La parte nueva de HTML5 es poder agregar entradas al historial del navegador y responder quitando esas entradas de la pila cuando el usuario pulsa el botón atrás del navegador. Esto significa que la URL puede seguir haciendo su trabajo como identificador único del recurso actual, incluso en aplicaciones con scripts pesados que nunca realizan una actualización de la página completa. (Pilgrim, 2010)

CINCO PUNTOS A TENER EN CUENTA

MÚLTIPLES MÓDULOS

HTML5 no es una gran unidad, sino un conjunto de características individuales. Así que no se puede hablar de “soporte HTML5,” porque eso no tiene ningún sentido. Pero se puede detectar el soporte de características individuales, como canvas, vídeo o geolocalización.

Los tags son una parte importante de HTML, pero no lo es todo. La especificación HTML5 define también cómo esos tags interactúan con JavaScript a través del Document Object Model (DOM). HTML5 no se limita a definir un tag <*video*>, sino que también brinda un correspondiente API DOM para los objetos de vídeo en el DOM. Se puede utilizar esta API para detectar soporte para diferentes formatos de vídeo, reproducir un vídeo, pausar, silenciar el audio, hacer un seguimiento de la cantidad de vídeo que se ha descargado, y todo lo que se necesita para crear una experiencia de usuario rica en torno al propio tag <*video*> .

RETROCOMPATIBILIDAD

HTML5 es compatible con todos los controles de formularios de HTML 4, e incluyendo nuevos controles de entrada, como reglas deslizantes y selectores de fecha, entre otros. Por ejemplo, el tipo de entrada de correo electrónico se parece a un cuadro de texto, pero los navegadores móviles personalizan el teclado en pantalla para que sea más fácil escribir direcciones de correo electrónico. Los navegadores más antiguos que no soportan el tipo de entrada de correo electrónico se tratan como un campo de texto regular, y el form sigue funcionando sin cambios externos.

FACILIDAD DE APRENDIZAJE

"Actualizar" a HTML5 puede ser tan simple como cambiar el ***doctype***. El doctype ya debería estar en la primera línea de cada página HTML. Las versiones anteriores de HTML definen numerosos doctypes, y elegir el más adecuado puede ser difícil. En HTML5, sólo hay un doctype: <*! DOCTYPE html*>

La actualización a la doctype HTML5 no romperá su markup existente, ya que los elementos obsoletos definidos previamente en HTML 4 todavía funcionan en HTML5. Pero va a permitir el uso - y validación- de nuevos elementos semánticos como <*article*>, <*section*>, <*header*> y <*footer*>.

FUNCIONALIDAD

Si se desea dibujar en un canvas, reproducir vídeo, diseñar mejores formas, o construir aplicaciones web que funcionan offline, es evidente que HTML5 ya tiene un amplio soporte. Firefox, Safari, Chrome, Opera y los navegadores móviles ya son compatibles con canvas, video, geolocalización, el almacenamiento local, y más. Google además ya soporta anotaciones de microdata. Incluso Microsoft es compatible con la mayoría de las características de HTML5 en Internet Explorer 9.

ACTUALIDAD

El W3C comentó sobre el futuro de los estándares web, en julio de 2009:

"Hoy, el Director anunció que cuando la carta XHTML Grupo de Trabajo 2 expira en la fecha prevista a finales de 2009, la carta no será renovada. De este modo, y mediante el aumento de los recursos en el Grupo de Trabajo de HTML, el W3C espera acelerar el progreso de HTML5 y aclarar la posición del W3C sobre el futuro de HTML." (Pilgrim, 2010)

JAVASCRIPT

¿QUÉ ES JAVASCRIPT?

JavaScript es un lenguaje de programación ***interpretado***, definido como ***orientado a objetos***, basado en ***prototipos***, ***imperativo***, débilmente ***tipado*** y dinámico.

Se utiliza principalmente en su forma del lado cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor. Su uso en aplicaciones externas a la web, por ejemplo en documentos PDF, aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

A pesar que Javascript es usado para múltiples propósitos, siempre fue considerado como un complemento hasta que se realizaron desarrollos de nuevos motores de interpretación, creados para acelerar el procesamiento de código. La clave de los motores más exitosos fue transformar el código Javascript en código máquina para lograr velocidades de ejecución similares a aquellas encontradas en aplicaciones de escritorio. Esta mejorada capacidad permitió superar viejas limitaciones de rendimiento y confirmar el lenguaje Javascript como la mejor opción para la web.

Para aprovechar esta prometedora plataforma de trabajo ofrecida por los nuevos navegadores, Javascript fue expandido en relación con portabilidad e integración. A la vez, interfaces de programación de aplicaciones (APIs) fueron incorporadas por defecto en cada navegador para asistir al lenguaje en funciones elementales. La idea es hacer disponible poderosas funciones a través de técnicas de programación sencillas y estándares, expandiendo el alcance del lenguaje y facilitando la creación de programas útiles para la web. (Gauchat, 2012)

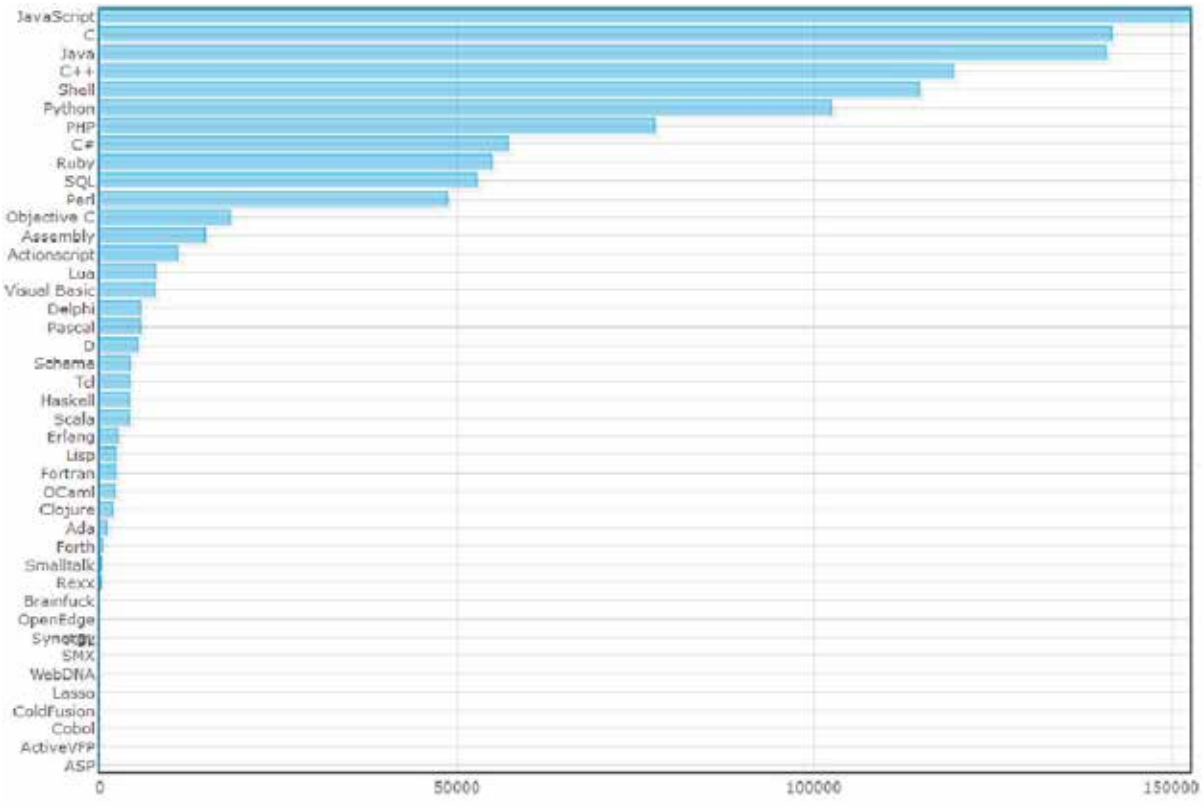


ILUSTRACIÓN 8 – UTILIZACIÓN MUNDIAL DE JAVASCRIPT EN COMPARACIÓN CON OTROS LENGUAJES (OHLOH SEARCH RESULTS, 2013)

UN POCO DE HISTORIA

A principios de los años 90, la mayoría de usuarios que se conectaban a Internet lo hacían con módems a una velocidad máxima de 28.8 kbps. En esa época, empezaban a desarrollarse las primeras aplicaciones web y por tanto, las páginas web comenzaban a incluir formularios complejos.

Con unas aplicaciones web cada vez más complejas y una velocidad de navegación tan lenta, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario. De esta forma, si el usuario no rellenaba correctamente un formulario, no se le hacía esperar mucho tiempo hasta que el servidor volviera a mostrar el formulario indicando los errores existentes.

Brendan Eich, un programador que trabajaba en Netscape, pensó que podría solucionar este problema adaptando otras tecnologías existentes (como ScriptEase) al navegador Netscape Navigator 2.0, que iba a lanzarse en 1995. Inicialmente, Eich denominó a su lenguaje LiveScript.

Posteriormente, Netscape firmó una alianza con Sun Microsystems para el desarrollo del nuevo lenguaje de programación. Además, justo antes del lanzamiento Netscape decidió cambiar el nombre por el de JavaScript. La razón del cambio de nombre fue exclusivamente por marketing, ya que Java era la palabra de moda en el mundo informático y de Internet de la época.

La primera versión de JavaScript fue un completo éxito y Netscape Navigator 3.0 ya incorporaba la siguiente versión del lenguaje, la versión 1.1. Al mismo tiempo, Microsoft lanzó JScript con su navega-

dor Internet Explorer 3. JScript era una copia de JavaScript al que le cambiaron el nombre para evitar problemas legales.

Para evitar una guerra de tecnologías, Netscape decidió que lo mejor sería estandarizar el lenguaje JavaScript. De esta forma, en 1997 se envió la especificación JavaScript 1.1 al organismo ECMA European Computer Manufacturers Association).

ECMA creó el comité TC39 con el objetivo de "estandarizar de un lenguaje de script multiplataforma e independiente de cualquier empresa". El primer estándar que creó el comité TC39 se denominó ECMA-262, en el que se definió por primera vez el lenguaje ECMAScript.

Por este motivo, algunos programadores prefieren la denominación ECMAScript para referirse al lenguaje JavaScript. De hecho, JavaScript no es más que la implementación que realizó la empresa Netscape del estándar ECMAScript.

La organización internacional para la estandarización (ISO) adoptó el estándar ECMA-262 a través de su comisión IEC, dando lugar al estándar ISO/IEC-16262.

CONCEPTOS BÁSICOS

SINTAXIS

DECLARACIONES VS EXPRESIONES

JavaScript tiene dos grandes categorías sintácticas: declaraciones y expresiones.

- Declaraciones "hacen cosas". Un programa es una secuencia de sentencias. Un ejemplo de una declaración sería una que crea una variable foo:

```
foo var;
```

- Expresiones producen valores. Son el lado derecho de una asignación, argumentos de funciones, etc. Un ejemplo de una expresión sería:

```
3 * 7
```

La distinción entre las declaraciones y expresiones se ilustra mejor por el hecho de que JavaScript (como Java) tiene dos maneras diferentes de hacer if-then-else. Ya sea como una declaración:

```
var x;

if (y > = 0) {

    x = y;

} Else {

      X = - Y;
}
```

O como una expresión:

```
      var x = y > = 0 ? Y: - Y;
```

Se puede utilizar el segundo como un argumento de función (pero no el primero):

```
      myFunction (y> = 0 y: ? -y)
```

Por último, mientras JavaScript espera una declaración, también se puede utilizar una expresión. Por ejemplo:

```
foo ( bar ( 7 , 1 ));
```

foo (...) es una sentencia (una “declaración expresión”) , bar (7 , 1) es una expresión. Ambas son llamadas a funciones.

CONTROL DE FLUJO

Para las declaraciones de flujo de control, el cuerpo puede ser una sola sentencia. Dos ejemplos:

```
if (obj == null!) obj.foo ();
```

```
while (x> 0) x -;
```

Sin embargo, cualquier declaración siempre puede ser reemplazada por un bloque, cuyas llaves pueden contener cero o más estados. De este modo, también se puede escribir:

```
if (obj! == null) {

    obj.foo ();
}
```

```
while (x> 0) {

    x -;

}
```

COMENTARIOS

JavaScript tiene dos tipos de comentarios: comentarios de una sola línea y comentarios multilínea. Comentarios de una sola línea comienzan con / / y se terminan al final de la línea:

```
x + +; / /-line solo comentario
```

Comentarios de varias líneas se delimitan con / * y * /

```
/ * Esto es
  una multilínea
  de comentario.
 * /
```

VARIABLES Y ASIGNACIONES

Se puede declarar una variable y asignarle un valor a la vez:

```
var foo = 6;
```

ASIGNACIONES

También se puede asignar un valor a una variable existente:

```
foo = 4 / / cambiar la variable foo
```

OPERADORES DE ASIGNACIÓN COMPUESTA

Hay operadores de asignación compuestos como +=. Las dos asignaciones siguientes son equivalentes:

```
x + = 1;

x = x + 1;
```

IDENTIFICADORES Y NOMBRE DE VARIABLES

Los identificadores son los nombres de las cosas, que desempeñan diversas funciones sintácticas de JavaScript. Por ejemplo, el nombre de una variable es un identificador.

A grandes rasgos, el primer carácter de un identificador puede ser cualquier letra Unicode, un signo de dólar ($) o un guion bajo (_). Caracteres posteriores pueden ser, además, cualquier dígito Unicode. Por lo tanto, los siguientes son los identificadores legales:

```
arg0

_tmp

$ elem

π
```

Varios identificadores son “palabras reservadas”. Son parte de la sintaxis y no se pueden utilizar como nombres de variables:

arguments break case catch class const continue debugger default delete do else enum eval export extends false finally for function if implements import in instanceof interface let new null package private protected public return static super switch this throw true try typeof var void while with yield.

Técnicamente, los siguientes tres identificadores no son palabras reservadas, pero no se deben utilizar como nombres de variables, ya sea:

```
Infinity NaN undefined
```

VALORES

JavaScript tiene todos los valores que podemos esperar de los lenguajes de programación: booleanos, números, cadenas, matrices, etc. Todos los valores en JavaScript tienen propiedades. Cada propiedad tiene un nombre y un valor. Se utiliza el punto operario leer una propiedad (.):

```
value.propKey
```

Un ejemplo: La cadena ‘abc’ tiene la característica longitud.

```
> Var str = ‘abc’;

> str.length

3
```

Lo anterior también se puede escribir como:

```
> ‘Abc’. Longitud
3
```

El operador de punto también se utiliza para asignar un valor a una propiedad:

```
> Var obj = {}; / / objeto vacío

> Obj.foo = 123; / / crear foo propiedad ``, la pusieron a 123

123

> obj.foo

123
```

Se puede llamar a los métodos a través de:

```
> 'Hola'. ToUpperCase ()

'HOLA'
```

VALORES PRIMITIVOS

Los valores primitivos (abreviado: primitivos) son:

- Booleanos: true, false
- Números: 1.736, 1.351
- Strings: 'abc', "abc"
- Dos "no-valores": indefinido, nulos

Características de primitivos:

- En comparación con el valor: el "contenido" se compara.

```
> 3 === 3

true

> 'Abc' === 'abc'

true
```

- Siempre inmutable: los valores de las propiedades no se pueden cambiar, no hay propiedades que puedan ser añadidas o eliminadas.

```
> Var str = 'abc';

> Str.foo = 3 / / tratar de crear propiedad foo ``⇨ ningún efecto

> Str.foo / / propiedad desconocida

undefined
```

(Lectura de una propiedad desconocida siempre devuelve undefined.)

- Un conjunto fijo de valores: no se puede definir sus propias primitivas.

OBJETOS

Todos los valores que no son primitivos son objetos. Los tipos más comunes de los objetos son:

- Objetos lisos (tipo *Objeto*) se pueden crear objetos literales:

```
{
   firstName : ‘Jane’ ,
   lastName : ‘Pérez’
}
```

El objeto anterior tiene dos propiedades: el valor de la propiedad firstName es ‘Jane’, el valor de la propiedad lastName es ‘Pérez’.

- Arrays (tipo *Array*) pueden ser creadas por literales de array:

```
[‘Manzana’, ‘banana’, ‘cherry’]
```

La matriz anterior tiene tres elementos que se pueden acceder a través de los índices numéricos. Por ejemplo, el índice de ‘ manzana ‘ es 0.

- Las expresiones regulares (tipo *RegExp*) pueden ser creados por los literales de expresiones regulares:

```
/ ^ a + b + $ /
```

Características de los objetos:

- En comparación por referencia: identidades se comparan, cada valor tiene su propia identidad.

```
> {} === {}  / / Dos objetos vacíos diferentes
false
> Var obj1 = {};
> Var obj2 = obj1;
> Obj1 === obj2
true
```

- Mutable por defecto.

```
> Var obj = {};
> Obj.foo = 123;
> obj.foo
123
```

- Extensible por el usuario: se puede definir nuevos tipos de objetos, a través de constructores.

Todas las estructuras de datos (tales como las matrices) son objetos, pero no todos los objetos son estructuras de datos. Por ejemplo: Las expresiones regulares son objetos, pero no es una estructura de datos.

VALORES PRIMITIVOS VS OBJETOS

JavaScript hace una distinción un tanto arbitraria entre los valores:

- Los valores primitivos son: booleanos, números, cadenas, nula, no definidos.
- Todos los demás valores son objetos. Eso es en realidad cómo se definen los objetos - todos los valores que no son primitivos.

La principal diferencia entre los dos es la forma en que se comparan: cada objeto tiene una identidad única y sólo es igual a sí mismo:

```
> Obj1 var = {}; / / un objeto vacío

> Var obj2 = {}; / / otro objeto vacío

> Obj1 === obj2

false

> Obj1 === obj1

true
```

En contraste, todos los valores primitivos que codifican el mismo valor se consideran la misma:

```
> Var PRIM1 = 123;

> Var prim2 = 123;

> PRIM1 === prim2

true
```

UNDEFINED Y NULL

Innecesariamente, JavaScript tiene dos “no-valores”: no *undefined* y *null*.

- Undefined significa “ningún valor”. Las variables sin inicializar son *undefined*:

```
> var foo;

> foo

undefined
```

Si se lee una propiedad inexistente, usted también consigue *undefined*:

```
> Var obj = {}; / / objeto vacío

> obj.foo

undefined
```

Parámetros que faltan también son *undefined*:

```
> function f (x) {return x}

> f ()

undefined
```

- *null* significa “no objeto”. Se utiliza como un no-valor cuando se espera que un objeto (parámetros, último elemento en un array de objetos, etc.)

Normalmente, se debe tratar *undefined* y *null* de manera equivalente. Un modo de comprobarlos es a través de una comparación explícita:

```
if (x === undefined | | x === null) {

    ...

}
```

Otra forma de hacerlo es aprovechar el hecho de que tanto *undefined* como *null* se consideran falsos:

```
if (! x) {

    ...

}
```

Advertencia: false, 0, NaN’’ y también se consideran falsa.

PROTOTIPOS

Las instancias de tipos de objetos Foo (incluidos los tipos incorporados como Array y cualquier tipo personalizados) obtienen sus métodos del objeto *Foo.prototype*. Se puede verificar esto mediante la lectura de un método sin invocarlo:

```
> [ ] . push === Array.prototype.push

verdadero
```

Por el contrario, los primitivos no tienen tipos que puede acceder en el lenguaje, por lo que cada tipo primitivo tiene un tipo asociado, el llamado tipo *wrapper*:

El tipo *wrapper* de booleanos es boolean. Booleanos obtienen sus métodos de *Boolean.prototype* :

```
> True.toString === Boolean.prototype.toString

verdadero
```

Se debe tener en cuenta que el nombre del tipo wrapper se inicia con una mayúscula. Si el tipo de booleanos sería accesible en JavaScript, es probable que se llame boolean.

- El tipo de contenedor de números es el Number.
- El tipo de contenedor de cadenas es String.

Tipos Wrapper también tienen instancias (sus instancias son objetos), pero prácticamente no se utili-

zan. En cambio, los tipos *wrapper* sirven a otro propósito: Si se los llama como funciones, convierten los valores de tipos primitivos.

```
> Number (‘123’)

123

> String (true)

‘true’
```

MANEJO DE EXCEPCIONES

La forma más común de tratamiento de excepciones es:

```
function lanzarExcepcion () {
    throw new Error (‘Problema’);
}
```

```
try {
    lanzarExcepcion ();
} Catch (e) {
    console.log (e) / / Error: Problema!
    console.log (e.stack) / / no estándar, pero a menudo soportado
}
```

La cláusula try rodea código crítico, la cláusula catch se ejecuta si se produce una excepción dentro del bloque try.

OBJETOS Y HERENCIA

Al igual que todos los valores, los objetos tienen propiedades. Se podría considerar que un objeto es un conjunto de propiedades, donde cada propiedad es un par (clave, valor). La clave es una cadena, el valor es cualquier valor de JavaScript. Existe una forma adicional de acceder a las propiedades que pueden manejar cadenas arbitrarias como claves.

OBJETOS INDIVIDUALES

En JavaScript, se pueden crear directamente objetos, a través de *object literals*:

```
var jane = {
    name: “Jane” ,

    describe : function ( ) {
```

```
        'use strict';

        return 'Person named'+ this.name ;

    }

} ;
```

El objeto anterior tiene las propiedades *name* y *describe*. Se puede leer ("get") y escribir ("set") propiedades:

```
> Jane.name / / get

'Jane'

> Jane.name = 'John '; / / conjunto

> Jane.newProperty = ' abc'; / / crea automáticamente
```

Las propiedades de función de valor tales como *describe* se denominan métodos. Estos usan *this* para hacer referencia al objeto que se utiliza para llamarlos.

```
> Jane.describe () / / método de llamada

'Person named John'

> Jane.name = ' Jane ';

> Jane.describe ( )

'Person named Jane'
```

El operador *in* controla si una propiedad existe:

```
> 'NewProperty' in jane

true
> 'Foo' en jane

false
```

Si se lee una propiedad que no existe, se obtiene el valor *undefined*. Por lo tanto, los dos controles anteriores también podrían ser realizados de esta manera:

```
> Jane.newProperty !== undefined

true
> Jane.foo !== undefined

false
```

El operador *delete* elimina una propiedad

```
> delete jane.newProperty

true
```

```
> ‘ newProperty ‘ en jane

false
```

CLAVES DE PROPIEDAD ARBITRARIA

Una clave de propiedad puede ser cualquier cadena. Pero sólo se pueden usar de esa manera si son identificadores. Si desea utilizar otras cadenas como claves, hay que citar en un literal de objeto y utilizar corchetes para obtener y establecer la propiedad:

```
> Var obj = {‘no es un identificador’ : 123 };

> Obj [‘no es un identificador’]

123

> Obj [‘no es un identificador’ ] = 456;
```

Los corchetes también permiten calcular la clave de una propiedad:

```
> Var x = “nombre”;

> Jane [x]

‘Jane’

> Jane [‘ na ‘ + ‘yo’]

‘Jane’
```

MÉTODOS DE EXTRACCIÓN

Si se extrae un método, se pierde la conexión con el objeto. En sí mismo, una función no es más un método y *this* tiene el valor undefined (en modo estricto).

```
> var func = jane.describe ;

> func ( )

TypeError: No se puede leer la propiedad ‘nombre’ de undefined
```

La solución es utilizar el método *bind()* que todas las funciones tienen. Este crea una nueva función cuyo this siempre tiene el valor dado.

```
> Var func2 = jane.describe.bind (jane);

> func2 ()

‘Person named Jane’
```

FUNCIONES DENTRO DE UN MÉTODO

Cada función tiene la variable especial this. Esto es un inconveniente si se anida una función dentro de un método, porque no se puede acceder al método *this* desde la función. El siguiente es un ejemplo en el que llamamos forEach con una función para iterar sobre una matriz:

```
var jane = {
    name: “Jane” ,
    friends : [‘Tarzán’, ‘Cheeta’] ,
    logHiToFriends : function ( ) {
        ‘use strict’;
        this.friends.forEach (function (friend) {
            / / ` this ‘ no está definido aquí
            console.log (this.name + ‘ says hi to ‘ + friend);
        } ) ;
    }
}
```

Llamar LogHiToFriends produce un error:

```
> Jane.logHiToFriends ( )
TypeError: No se puede leer la propiedad “name” de undefined
```

Hay dos maneras de arreglar esto.

- Guardar *this* en una variable diferente.

```
logHiToFriends : function ( ) {
    ‘use strict’;
    var that = this;
    this.friends.forEach (function (friend) {
        console.log (that.name + ‘ says hi to ‘ + friend) ;
    } ) ;
}
```

- forEach tiene un segundo parámetro que le permite proporcionar un valor para *this*.

```
logHiToFriends : function ( ) {
    ‘use strict’;
    this.friends.forEach (function (friend) {
        console.log (this.name + ‘ says hi to ‘ + friend) ;
    } , this ) ;
}
```

Las expresiones de función se utilizan a menudo como argumentos en llamadas a funciones en JavaScript. Siempre se debe tener cuidado cuando se hace referencia a *this* desde una de las expresiones de función.

CONSTRUCTORES: FÁBRICAS DE OBJETOS

Los objetos de JavaScript también soportan una función orientada a objetos verdaderamente: la herencia.

Además de ser funciones "reales" y los métodos, las funciones desempeñan una tercera función en JavaScript: Se convierten en constructores, fábricas de objetos, si se invoca a través del operador *new*. Los constructores son análogos a las clases en otros idiomas. Por convención, los nombres de los constructores comienzan con letras mayúsculas. Por ejemplo:

```
/ / Establecer los datos de instancia

function Point (x , y) {

    this.x = x;

    this.y = y;

}

/ / Métodos

Point.prototype.dist = function ( ) {

    return Math.sqrt ( this.x * this.x + this.y * this.y );

} ;
```

Podemos ver que un constructor tiene dos partes: En primer lugar, la función *Point* configura los datos de la instancia. En segundo lugar, la propiedad Point.prototype contiene un objeto con los métodos. Los parámetros anteriores se especifican para cada instancia, este último compartiendo datos entre todas las instancias.

Para utilizar *Point*, invocamos a ella a través del nuevo operador:

```
> Var p = new Point (3,5);

> p.x

3

> P.dist ( )

5.830951894845301
```

p es un ejemplo de *Point*:

```
> P instanceof Point

true

> Typeof p

'object'
```

(Rauschmayer, 2013)

CSS

¿QUÉ ES CSS?

CSS es un lenguaje de hojas de estilos creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados "documentos semánticos"). Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes.

Al crear una página web, se utiliza en primer lugar el lenguaje HTML/XHTML para marcar los contenidos, es decir, para designar la función de cada elemento dentro de la página: párrafo, titular, texto destacado, tabla, lista de elementos, etc.

Una vez creados los contenidos, se utiliza el lenguaje CSS para definir el aspecto de cada elemento: color, tamaño y tipo de letra del texto, separación horizontal y vertical entre elementos, posición de cada elemento dentro de la página, etc.

UN POCO DE HISTORIA

Las hojas de estilos aparecieron poco después que el lenguaje de etiquetas SGML, alrededor del año 1970. Desde la creación de SGML, se observó la necesidad de definir un mecanismo que permitiera aplicar de forma consistente diferentes estilos a los documentos electrónicos.

El gran impulso de los lenguajes de hojas de estilos se produjo con el boom de Internet y el crecimiento exponencial del lenguaje HTML para la creación de documentos electrónicos. La guerra de navegadores y la falta de un estándar para la definición de los estilos dificultaban la creación de documentos con la misma apariencia en diferentes navegadores.

El organismo W3C (World Wide Web Consortium), encargado de crear todos los estándares relacionados con la web, propuso la creación de un lenguaje de hojas de estilos específico para el lenguaje HTML y se presentaron nueve propuestas. Las dos propuestas que se tuvieron en cuenta fueron la CHSS (Cascading HTML Style Sheets) y la SSP (Stream-based Style Sheet Proposal).

La propuesta CHSS fue realizada por Håkon Wium Lie y SSP fue propuesto por Bert Bos. Entre finales de 1994 y 1995 Lie y Bos se unieron para definir un nuevo lenguaje que tomaba lo mejor de cada propuesta y lo llamaron CSS (Cascading Style Sheets).

En 1995, el W3C decidió apostar por el desarrollo y estandarización de CSS y lo añadió a su grupo de trabajo de HTML. A finales de 1996, el W3C publicó la primera recomendación oficial, conocida como "CSS nivel 1".

A principios de 1997, el W3C decide separar los trabajos del grupo de HTML en tres secciones: el grupo de trabajo de HTML, el grupo de trabajo de DOM y el grupo de trabajo de CSS.

El 12 de Mayo de 1998, el grupo de trabajo de CSS publica su segunda recomendación oficial, conocida como “CSS nivel 2”. La versión de CSS que utilizan todos los navegadores de hoy en día es CSS 2.1. Al mismo tiempo, la siguiente recomendación de CSS, conocida como “CSS nivel 3”, continúa en desarrollo desde 1998 y hasta el momento sólo se han publicado borradores.

La adopción de CSS por parte de los navegadores ha requerido un largo período de tiempo. El mismo año que se publicó CSS 1, Microsoft lanzaba su navegador Internet Explorer 3.0, que disponía de un soporte bastante reducido de CSS. El primer navegador con soporte completo de CSS 1 fue la versión para Mac de Internet Explorer 5, que se publicó en el año 2000. (Eguiluz, 2011)

CONCEPTOS BÁSICOS

FUNCIONAMIENTO BÁSICO

Antes de que se generalizara el uso de CSS, los diseñadores de páginas web utilizaban etiquetas HTML especiales para modificar el aspecto de los elementos de la página. El siguiente ejemplo muestra una página HTML con estilos definidos sin utilizar CSS:

```
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN”
  “http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>
<html xmlns=”http://www.w3.org/1999/xhtml”>
<head>
  <meta http-equiv=”Content-Type” content=”text/html; charset=iso-8859-1” />
  <title>Ejemplo de estilos sin CSS</title>
</head>
<body>
  <h1><font color=”red” face=”Arial” size=”5”>Titular de la página</font></h1>
  <p><font color=”gray” face=”Verdana” size=”2”>Un párrafo de texto no muy largo.</font></p>
</body>
</html>
```

El ejemplo anterior utiliza la etiqueta <*font*> con sus atributos *color*, *face* y *size* para definir el color, el tipo y el tamaño de letra de cada elemento de la página.

El problema de utilizar este método para definir el aspecto de los elementos se puede ver claramente con el siguiente ejemplo: si la página tuviera 50 elementos diferentes, habría que insertar 50 etiquetas <*font*>. Si el sitio web entero se compone de 10.000 páginas diferentes, habría que definir 500.000 etiquetas <*font*>. Como cada etiqueta <font> tiene tres atributos, habría que definir 1.5 millones de atributos.

Como el diseño de los sitios web está en constante evolución, es habitual modificar cada cierto tiempo el aspecto de las páginas del sitio. Siguiendo con el ejemplo anterior, cambiar el aspecto del sitio requeriría modificar 500.000 etiquetas y 1.5 millones de atributos.

La solución que propone CSS es mucho mejor, como se puede ver en el siguiente ejemplo:

```
<!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.0 Transitional//EN”
  “http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd”>
<html xmlns=”http://www.w3.org/1999/xhtml”>
<head>
<meta http-equiv=”Content-Type” content=”text/html; charset=iso-8859-1” />
<title>Ejemplo de estilos con CSS</title>
<style type=”text/css”>
  h1 { color: red;  font-family: Arial;   font-size: large;  }
  p  { color: gray; font-family: Verdana; font-size: medium; }
</style>
</head>
<body>
  <h1>Titular de la página</h1>
  <p>Un párrafo de texto no muy largo.</p>
</body>
</html>
```

CSS permite separar los contenidos de la página y la información sobre su aspecto. En el ejemplo anterior, dentro de la propia página HTML se crea una zona especial en la que se incluye toda la información relacionada con los estilos de la página.

Utilizando CSS, se pueden establecer los mismos estilos con menos esfuerzo y sin ensuciar el código HTML de los contenidos con etiquetas <*font*>. Como se verá más adelante, la etiqueta <*style*> crea una zona especial donde se incluyen todas las reglas CSS que se aplican en la página.

En el ejemplo anterior, dentro de la zona de CSS se indica que todas las etiquetas <*h1*> de la página se deben ver de color rojo, con un tipo de letra Arial y con un tamaño de letra grande. Además, las etiquetas <*p*> de la página se deben ver de color gris, con un tipo de letra Verdana y con un tamaño de letra medio.

MEDIOS CSS

Una de las características más importantes de las hojas de estilos CSS es que permiten definir diferentes estilos para diferentes medios o dispositivos: pantallas, impresoras, móviles, proyectores, etc.

Además, CSS define algunas propiedades específicamente para determinados medios, como por ejemplo la paginación y los saltos de página para los medios impresos o el volumen y tipo de voz para los medios de audio. La siguiente tabla muestra el nombre que CSS utiliza para identificar cada medio y su descripción:

Medio	Descripción
all	Todos los medios definidos
braille	Dispositivos táctiles que emplean el sistema braille
embosed	Impresoras braille
handheld	Dispositivos de mano: móviles, PDA, etc.
print	Impresoras y navegadores en el modo “Vista Previa para Imprimir”
projection	Proyectores y dispositivos para presentaciones
screen	Pantallas de ordenador
speech	Sintetizadores para navegadores de voz utilizados por personas discapacitadas
tty	Dispositivos textuales limitados como teletipos y terminales de texto
tv	televisores y dispositivos con resolución baja

SINTAXIS

Cada regla CSS describe cómo se debe representar determinado elemento en el navegador. Toda regla tiene un *selector*, ese selector es el que define a qué elemento en particular se aplica el diseño. Luego se escribe la *declaración* entre llaves que es el grupo de características que va a tener ese elemento. Entre las llaves hay una o más *propiedades* con uno o más *valores*. Cada propiedad se separa con punto y coma.

```
h1 { color: red;  font-family: Arial;   font-size: large;  }
```

ALGUNOS EJEMPLOS DE SELECTORES

Selector	Descripción
*	Selector universal, son todos los elementos del CSS
E	E representa cualquier elemento del tipo E (span, p, …)
E F	Todos los elementos F que sean descendentes de E
E > F	Todos los elementos F que sean hijos de E
E:first-child	De esta forma podemos seleccionar el primer elemento de tipo E
E:link , E:visited	Selecciona los elementos E que sean un enlaces y no hayan sido visitados (:link) y los si visitados (:visited)
E:active , E:hover , E:focus	Selecciona los elementos de tipo E , en sus correspondientes acciones.
E:lang(c)	Cogemos los elementos del tipo E que estén en el idioma (humano) especificado en (c).
E + F	Se trata de cualquier elemento F inmediatamente después del elemento del tipo E

E[foo]	Elementos del tipo E con el atributo foo
E[foo="ejemplo"]	Elementos del tipo E con el atributo foo igual a "ejemplo"
E[foo~="ejemplo"]	Elementos del tipo E con el atributo foo contenga "ejemplo". Se pueden añadir varias palabras separadas por espacios. (~ =ALT + 0126)
E[lang|="es"]	Similar al anterior, pero se referirá a todos los elemento E tal que su atributo lang comience por "es". Por ejemplo: "es_ES", "es_CA",…
E[foo$="ejemplo"]	Elementos del tipo E en el que el atributo foo termine con "ejemplo".
DIV.ejemplo	Todos los elementos DIV que sean de la clase ejemplo
E#miID	El elemento E en el que su ID sea igual miID

ESTILOS DE FUENTES

La propiedad font-family debe contener varios nombres de fuente como un sistema de "red de seguridad", para asegurar la máxima compatibilidad entre navegadores / sistemas operativos. Si el navegador no es compatible con la primera fuente, que trata la siguiente fuente.

Se comienza con la fuente que se desea, y se termina con una familia genérica, para permitir que el navegador elija una fuente similar en la familia genérica, si no hay otras fuentes están disponibles. Por ejemplo:

p{font-family:"Times New Roman", Times, serif}

Entre las familias más comunes, se encuentran:

- Serif Fonts
  - Georgia, serif
  - "Palatino Linotype", "Book Antiqua", Palatino, serif
  - "Times New Roman", Times, serif
- Sans-Serif Fonts
  - Arial, Helvetica, sans-serif
  - "Arial Black", Gadget, sans-serif
  - "Comic Sans MS", cursive, sans-serif
  - Impact, Charcoal, sans-serif
  - "Lucida Sans Unicode", "Lucida Grande", sans-serif
  - Tahoma, Geneva, sans-serif
  - "Trebuchet MS", Helvetica, sans-serif
  - Verdana, Geneva, sans-serif
- Monospace Fonts
  - "Courier New", Courier, monospace
  - "Lucida Console", Monaco, monospace

ILUSTRACIÓN 9 – EJEMPLOS DE FUENTES ACEPTADAS (SCHMITZ, 2012)

UNIDADES DE MEDIDA

Unit	Description
%	percentage
in	inch
cm	centimeter
mm	millimeter
em	1em is equal to the current font size. 2em means 2 times the size of the current font. E.g., if an element is displayed with a font of 12 pt, then '2em' is 24 pt. The 'em' is a very useful unit in CSS, since it can adapt automatically to the font that the reader uses
ex	one ex is the x-height of a font (x-height is usually about half the font-size)
pt	point (1 pt is the same as 1/72 inch)
pc	pica (1 pc is the same as 12 points)
px	pixels (a dot on the computer screen)

COLORES

Los colores en CSS se pueden indicar de cinco formas diferentes:

- Palabras clave
  - aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, orange, purple, red, silver, teal, white, yellow...

- Colores del sistema
  - ActiveBorder, ActiveCaption, AppWorkspace, Background, ButtonFace, ButtonHighlight...

- RGB hexadecimal
  - #4762B0, #000000, #FF0000, #00FF00, #0000FF, #FFFF00, #00FFFF, #FF00FF, #C0C0C0, #FFFFFF...

- RGB numérico
  - rgb(0,0,0), rgb(255,0,0), rgb(0,255,0), rgb(0,0,255), rgb(255,255,0), rgb(0,255,255), rgb(255,0,255), rgb(192,192,192), rgb(255,255,255)...

- RGB porcentual.
  - hsl(120,100%,50%), hsl(120,100%,75%), hsl(120,100%,25%), hsl(120,60%,70%), hsl(290,100%,50%), hsl(290,60%,70%)...

INTEGRANDO HTML5, CSS3 Y JAVASCRIPT

Todas las tecnologías analizadas previamente son sumamente útiles en sí mismas, pero mucho más cuando estas se integran para lograr un sitio o aplicación web eficiente, visualmente atractiva, practica y organizada.

Esto tal vez es más comprensible con un ejemplo práctico. Pongamos de ejemplo el siguiente código:

```
<!DOCTYPE html>

  <body>

    <header>

      <h1>Titulo Principal</h1>

    </header>

    <nav>

      <ul>

        <li>Opción 1</li>

        <li>Opción 2</li>

        <li>Opción 3</li>

      </ul>

    </nav>

    <section>

      <h2>Titulo Articulo</h2>

        <h3><strong>Sub-Titulo:</strong> Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa.</h3>

      <article>
```

```
<p><strong>Parrafo:</strong> Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum. Nam quam nunc, blandit vel, luctus pulvinar, hendrerit id, lorem. Maecenas nec odio et ante tincidunt tempus. Donec vitae sapien ut libero venenatis faucibus. Nullam quis ante. Etiam sit amet orci eget eros faucibus tincidunt. Duis leo. Sed fringilla mauris sit amet nibh. Donec sodales sagittis magna. Sed consequat, leo eget bibendum sodales, augue velit cursus nunc.</p>

        </article>

    </section>

</body>
```

Este ejemplo está compuesto solo por elementos HTML. El resultado final sería el siguiente:

ILUSTRACIÓN 10 – EJEMPLO DE PÁGINA WEB HTML PURA (DESARROLLO DE AUTOR)

A pesar que cumple la finalidad que se busca, carece de atractivo y estilo personal, además de no dirigir la atención del usuario a donde se pretende.

Si a este ejemplo se le agrega una plantilla CSS de estilos, se paliaría estas deficiencias. El código con la plantilla css quedaría de la siguiente manera:

```
<!DOCTYPE html>

    <head>

    <style>

    body {

    font-family: arial, helvetica, sans-serif;

    font-size: 14px;
```

```
color: black;
background-color: #ffc;
margin: 20px;
padding: 0;
}
p {
line-height: 21px;
}

h1 {
color: #ffc;
background-color: #900;
font-size: 2em;
margin: 0;
margin-bottom: 7px;
padding: 4px;
font-style: italic;
text-align: center;
letter-spacing: 0.5em;
border-bottom-style: solid;
border-bottom-width: 0.5em;
border-bottom-color: #c00;
}

h2 {
color: white;
background-color: #090;
font-size: 1.5em;
margin: 0;
padding: 2px;
```

```
padding-left: 14px;
}

h3 {

color: #999;

font-size: 1.25em;

}

section {

border-style: groove;

border-width: 2px;

border-color: #ccc;

padding: 5px;

background-color: #fff;

}

a {

text-decoration: none;

}

strong {

font-style: italic;

text-transform: uppercase;

}

li {

color: #900;

font-style: italic;

}

</style>

</head>

<body>
```

```
    <header>
        <h1>Titulo Principal</h1>
      </header>
      <nav>
        <ul>
          <li>Opción 1</li>
          <li>Opción 2</li>
          <li>Opción 3</li>
        </ul>
      </nav>
      <section>
        <h2>Titulo Articulo</h2>
         <h3><strong>Sub-Titulo:</strong> Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa.</h3>
        <article>
            <p><strong>Parrafo:</strong> Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum. Nam quam nunc, blandit vel, luctus pulvinar, hendrerit id, lorem. Maecenas nec odio et ante tincidunt tempus. Donec vitae sapien ut libero venenatis faucibus. Nullam quis ante. Etiam sit amet orci eget eros faucibus tincidunt. Duis leo. Sed fringilla mauris sit amet nibh. Donec sodales sagittis magna. Sed consequat, leo eget bibendum sodales, augue velit cursus nunc.</p>
        </article>
      </section>
    </body>
```

Y el resultado final:

ILUSTRACIÓN 11 – EJEMPLO DE PÁGINA WEB + CSS (DESARROLLO DE AUTOR)

En este ejemplo, la página que se muestra es mucho más agradable a la vista y con sus secciones claramente delimitadas. Pero en su estado actual, no es más que una fachada, no posee ningún tipo de interactividad. Ahí es donde entra en acción JavaScript.

Agregando el script, le podemos dar funcionalidad a las opciones. El código final sería:

```
<!DOCTYPE html>
    <head>
    <style>
    body {
    font-family: arial, helvetica, sans-serif;
    font-size: 14px;
    color: black;
    background-color: #ffc;
    margin: 20px;
    padding: 0;
    }
    p {
    line-height: 21px;
    }

    h1 {
    color: #ffc;
    background-color: #900;
```

```
font-size: 2em;
margin: 0;
margin-bottom: 7px;
padding: 4px;
font-style: italic;
text-align: center;
letter-spacing: 0.5em;
border-bottom-style: solid;
border-bottom-width: 0.5em;
border-bottom-color: #c00;
}

h2 {
color: white;
background-color: #090;
font-size: 1.5em;
margin: 0;
padding: 2px;
padding-left: 14px;
}

h3 {
color: #999;
font-size: 1.25em;
}

section {
border-style: groove;
border-width: 2px;
border-color: #ccc;
padding: 5px;
```

```
background-color: #fff;
}

a {
text-decoration: none;
}

strong {
font-style: italic;
text-transform: uppercase;
}

li {
color: #900;
font-style: italic;
}
</style>
 <script>
 function funcion(opcion)
 {
 var option = document.getElementById(opcion).innerHTML
 alert("Esta es información relevante a la "+option);
 }
 </script>
</head>
<body>
    <header>
        <h1>Titulo Principal</h1>
    </header>
    <nav>
        <ul>
```

```
            <li id="Opcion1" onclick="funcion('Opcion1')">Opción 1</li>

            <li id="Opcion2" onclick="funcion('Opcion2')">Opción 2</li>

            <li id="Opcion3" onclick="funcion('Opcion3')">Opción 3</li>

          </ul>

        </nav>

        <section>

          <h2>Titulo Articulo</h2>

            <h3><strong>Sub-Titulo:</strong> Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa.</h3>

          <article>

           <p><strong>Parrafo:</strong> Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aenean commodo ligula eget dolor. Aenean massa. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Donec quam felis, ultricies nec, pellentesque eu, pretium quis, sem. Nulla consequat massa quis enim. Donec pede justo, fringilla vel, aliquet nec, vulputate eget, arcu. In enim justo, rhoncus ut, imperdiet a, venenatis vitae, justo. Nullam dictum felis eu pede mollis pretium. Integer tincidunt. Cras dapibus. Vivamus elementum semper nisi. Aenean vulputate eleifend tellus. Aenean leo ligula, porttitor eu, consequat vitae, eleifend ac, enim. Aliquam lorem ante, dapibus in, viverra quis, feugiat a, tellus. Phasellus viverra nulla ut metus varius laoreet. Quisque rutrum. Aenean imperdiet. Etiam ultricies nisi vel augue. Curabitur ullamcorper ultricies nisi. Nam eget dui. Etiam rhoncus. Maecenas tempus, tellus eget condimentum rhoncus, sem quam semper libero, sit amet adipiscing sem neque sed ipsum. Nam quam nunc, blandit vel, luctus pulvinar, hendrerit id, lorem. Maecenas nec odio et ante tincidunt tempus. Donec vitae sapien ut libero venenatis faucibus. Nullam quis ante. Etiam sit amet orci eget eros faucibus tincidunt. Duis leo. Sed fringilla mauris sit amet nibh. Donec sodales sagittis magna. Sed consequat, leo eget bibendum sodales, augue velit cursus nunc.</p>

          </article>

        </section>

    </body>
```

Con estos agregados, al hacer clic en alguna de las opciones, se presentará una ventana de alerta con información pertinente, que sería presentada al usuario de la siguiente manera:

ILUSTRACIÓN 12 – EJEMPLO DE PÁGINA WEB HTML + CSS + JAVASCRIPT (DESARROLLO DE AUTOR)

De esta manera, integrando HTML, JavaScript y CSS, podemos lograr un sitio web organizado, atractivo y completamente funcional, con codificación estándar, escalable y reutilizable, en su gran mayoría.

PHONEGAP

¿QUÉ ES PHONEGAP?

PhoneGap es un framework para el desarrollo de aplicaciones móviles producido por la empresa Nitobi, y comprado posteriormente por Adobe Systems. Principalmente, PhoneGap permite a los programadores desarrollar aplicaciones para dispositivos móviles utilizando herramientas tales como JavaScript, HTML5 y CSS3. Las aplicaciones resultantes son híbridas, es decir que no son realmente aplicaciones nativas al dispositivo (ya que el renderizado es realizado mediante vistas web y no con interfaces gráficas específicas a cada sistema), pero no se trata tampoco de aplicaciones web puras (teniendo en cuenta que son aplicaciones que son empaquetadas para poder ser desplegadas en el dispositivo incluso trabajando con el API del sistema nativo).

PhoneGap maneja APIs que permiten tener acceso a elementos como el acelerómetro, cámara, compás, contactos en el dispositivo, red, almacenamiento, notificaciones, geolocalización, etc. PhoneGap además nos permite el desarrollo ya sea ejecutando las aplicaciones en nuestro navegador web, sin tener que utilizar un simulador dedicado a esta tarea, además, nos da la posibilidad de soportar funciones sobre frameworks como Sencha Touch o JQuery Mobile.

PhoneGap puede ser considerado como una distribución de Apache Cordova. La aplicación fue primero llamada solamente “PhoneGap”, y luego “Apache Callback”. Apache Cordova es un software de código abierto.

Este framework permite a los desarrolladores web enfocarse en el desarrollo para los teléfonos inteligentes teniendo como base un código genérico con herramientas tales como JavaScript, HTML, CSS, y creando una interfaz de funciones foráneas para embeber una vista Web en el dispositivo móvil. Por lo tanto la gran ventaja de usar PhoneGap es que el código base de la aplicación será válido para iPhone, Android, BlackBerry OS, WebOS, Windows Phone, Symbian y Bada. De esta forma un mismo código servirá para todas las plataformas de desarrollo de aplicaciones para smartphones y tablets, sólo habrá crear el proyecto en cada IDE de desarrollo y añadir las librerías PhoneGap (Visual Studio .Net para Windows Phone, iOS Dev para iPhone, Eclipse y Android Studio para Android, NetBeans para Simbian, etc.).

Puesto que con PhoneGap se genera una aplicación propia del dispositivo, podrá publicarse en el correspondiente centro de compra y descargas (Google Play, App Store de iTunes, Tienda Windows App, etc.).

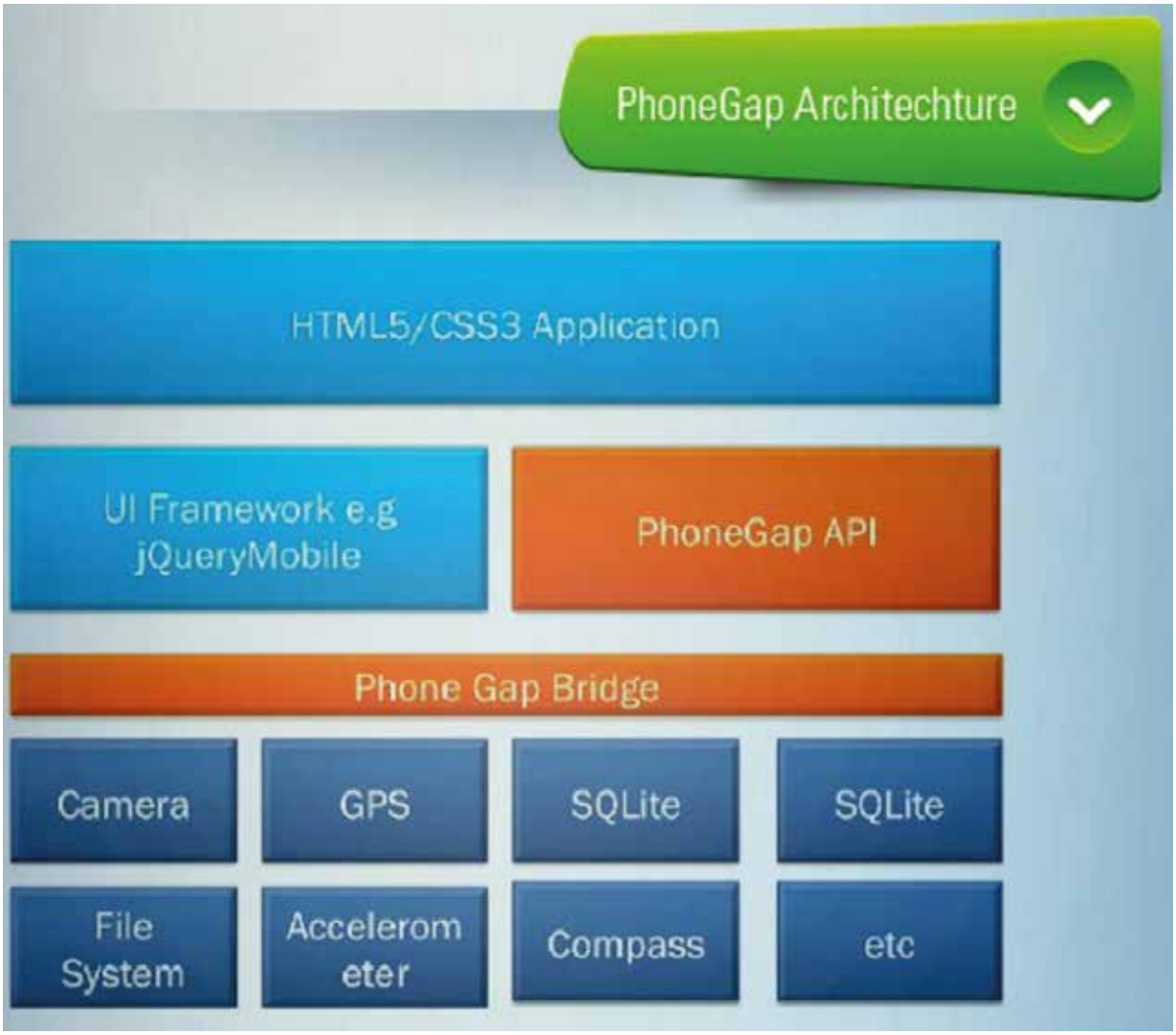


ILUSTRACIÓN 13 – ARQUITECTURA DE PHONEGAP (QUANTUMCLOUD, 2013)

UN POCO DE HISTORIA

Desarrollado por primera vez en un evento iPhoneDevCamp en San Francisco, PhoneGap pasó a ganar el People's Choice Award en la O'Reilly Media's 2009 Web 2.0 Conference y el framework se ha utilizado para desarrollar muchas aplicaciones. Apple Inc. ha confirmado que el marco tiene su aprobación, incluso con los nuevos cambios acuerdo de licencia para desarrolladores 4.0. El framework PhoneGap es utilizado por varias plataformas de aplicaciones móviles tales como ViziApps, Worklight, Convertigo, y appMobi como la columna vertebral del motor de desarrollo de clientes móviles. Adobe anunció oficialmente la adquisición de Nitobi Software (el desarrollador original) el 4 de octubre de 2011. Coincidiendo con esto, el código PhoneGap contribuido a la Fundación de Software Apache para iniciar un nuevo proyecto llamado Apache Córdoba. El nombre original del proyecto, Apache Callback, fue visto como demasiado genérico. Y también aparece en Adobe Systems como Adobe PhoneGap y también como Adobe Phonegap Build, ambos como plataformas gratis y de código libre, bajo la licencia Apache 2.0.

Las primeras versiones de PhoneGap requerían que el desarrollador que hace aplicaciones de iOS tuviese un ordenador de Apple, y un desarrollador que hace Windows Mobile Apps para tener un equipo que ejecuta Windows. Después de septiembre de 2012, el servicio PhoneGap Build de Adobe permite a los programadores cargar código HTML, CSS y JavaScript de código fuente a un "compilador nube " que genera aplicaciones para cada plataforma soportada. (Finley, 2012)

CONOCIMIENTOS BÁSICOS

Aplicaciones Cordova trabajan a partir de un archivo *config.xml* común que proporciona información sobre la aplicación y especifica los parámetros que afectan a la forma en que funciona, como si responde a los cambios de orientación. Este archivo ses consistente con la especificación Packaged Web App del W3C, o especificación ***widget.***

La aplicación en sí se implementa como una página web, llamada index.html por defecto, que referencia a todo archivo CSS, JavaScript, imágenes, archivos multimedia u otros recursos necesarios para que se ejecute. La aplicación se ejecuta como un WebView dentro de la envoltura de aplicación nativa, el cual se distribuye a las tiendas de aplicaciones. Para que la aplicación web pueda interactuar con diferentes dispositivos, debe hacer referencia a un archivo *cordova.js*, que proporciona enlaces de API.

El WebView puede proporcionar a la aplicación la totalidad de su interfaz de usuario. También puede ser un componente dentro de una aplicación más grande e híbrida que mezcla la WebView con los componentes de aplicaciones nativas. Cordova ofrece una interfaz de complementos para que estos componentes se comuniquen entre sí.

PATHS DE DESARROLLO

Desde la versión 3.0, se puede utilizar dos flujos de trabajo básicos para crear una aplicación móvil. Mientras que se puede conseguir lo mismo con los dos flujos de trabajo, algunas tareas son más adecuadas para el uso de un flujo de trabajo por sobre el otro. Por esta razón, se debe comprender los flujos de trabajo para utilizar la mejor herramienta acorde a la situación.

Los dos flujos de trabajo principales que se admiten son el flujo de trabajo de Project Web Dev y el flujo de trabajo nativo Plataforma Dev. .

Se puede pensar en el primer flujo de trabajo como el flujo de trabajo de Project Web Dev. Se debe utilizar este flujo de trabajo cuando se quiere crear una aplicación Cordova que se ejecuta en tantos sistemas operativos móvil como sea posible con el menor trabajo de desarrollo específico de la plataforma como sea posible. Este flujo de trabajo entró en existencia con Córdova 3.0 y la creación de la Cordova Command-line Interface (CLI). La CLI abstrae mucho de la funcionalidad del shell scripts de nivel inferior que se ocupa de los detalles relacionados con la construcción de la aplicación, tales como la copia de sus activos web en las carpetas correctas para cada plataforma móvil, cambios de configuración específicos de la plataforma, o correr scripts de construcción específicos para generar los binarios de aplicación.

El segundo flujo de trabajo puede ser pensado como un flujo de trabajo nativo Plataforma Dev. Se debe utilizar cuando se quiere centrarse en la creación de una aplicación para una sola plataforma y está interesado en cambiar los detalles de la plataforma de nivel inferior. Se puede utilizar este flujo de trabajo para crear aplicaciones de plataforma cruzada, pero la falta de herramientas para abstraer los distintos pasos de generación se hace más difícil. Por ejemplo, se deberá usar Plugman instalar el mismo plugin de una vez para cada plataforma que desea apoyar. La ventaja de utilizar este flujo de trabajo nativo Plataforma Dev. es que le da acceso a los scripts de shell de nivel inferior para construir y probar la aplicación, por lo que si usted está trabajando en la parte nativa, este flujo de trabajo es la forma más eficiente para probar los cambios. Este flujo de trabajo también es apropiado si se desea utilizar el CordovaWebView como una pequeña parte de una aplicación nativa más grande.

Cuando se empieza, podría ser más fácil de utilizar el flujo de trabajo de Project Web Dev. para crear una aplicación. En función del conjunto de plataformas que desea orientar su proyecto, se puede confiar en la CLI para mayores cuotas del ciclo de desarrollo:

- En el escenario más básico, puede utilizar la línea de comandos, simplemente para crear un nuevo proyecto que se rellena con la configuración predeterminada para modificarla.

- Para muchas plataformas móviles, también puede utilizar la CLI para configurar los archivos de proyecto adicionales necesarios para compilar dentro de cada SDK. Para que esto funcione, debe instalar el SDK de cada plataforma de destino.

- Para el apoyo a las plataformas, la CLI puede compilar aplicaciones y ejecutarlas en un emulador de dispositivo basado en el SDK. Para la prueba completa, también puede generar archivos de aplicaciones e instalarlas directamente en un dispositivo.

En cualquier punto en el ciclo de desarrollo, puede pasar a utilizar más del flujo de trabajo nativo Plataforma Dev. Las herramientas del SDK de plataforma específica provistos pueden proporcionar un conjunto más rico de opciones.

Un entorno SDK es más adecuado si desea implementar una aplicación híbrida que mezcla componentes de la aplicación y nativas basadas en la Web. Se puede utilizar la utilidad de línea de comandos para generar inicialmente la aplicación, o de forma iterativa a partir de entonces para alimentar código actualizado a las herramientas del SDK. También puede crear el archivo de configuración de la aplicación a sí mismo.

VENTAJAS DE SU UTILIZACIÓN

- Produce aplicaciones multiplataforma, permitiendo que el código fuente se pueda ejecutar en diferentes plataformas.

- Las aplicaciones creadas son capaces de acceder a todos los recursos del móvil, como pueden ser la cámara o el sistema GPS.

- A diferencia de las aplicaciones web, no es necesaria una conexión a internet para ejecutar la aplicación a excepción de partes concretas de la aplicación que requieran dicha conexión.

- No es necesario utilizar un lenguaje específico para crear la aplicación íntegra, solo es necesario para la plataforma contenedora. En su mayor parte utiliza HTML5, un lenguaje sencillo y por lo tanto, el desarrollo de estas aplicaciones presupone un menor tiempo de desarrollo.

- Es posible subir estas aplicaciones a los marketplace de las diferentes plataformas móviles, lo cual permite al usuario encontrarlas de una forma más rápida y de esta manera, facilitar la distribución de las mismas.

ANALISIS DEL PROBLEMA

SITUACIÓN ACTUAL

Con los datos presentados en el marco teórico, se puede establecer que es posible realizar una aplicación para dispositivos móviles a partir de tecnologías web, con la ayuda del framework Phonegap. Se pueden realizar aplicaciones para diarios online, blogs de diversos temas, sitios de compras, buscadores web, sitios web empresariales o gubernamentales, clientes de email, sitios de resultados deportivos, aplicaciones para eventos y numerosas posibilidades más.

Pero la finalidad de esta tesina no es realizar una aplicación basada en un sitio web clásico, sino desarrollar un juego completamente en HTML5, JavaScript y CSS3, con resolución adaptable a los diferentes dispositivos y un rendimiento similar en las diferentes plataformas.

Hay registros de juegos realizados con estas características, utilizando Phonegap, pero estos son juegos bastante básicos en su funcionamiento, con animaciones y transiciones simples y no muy demandantes (la gran mayoría, utilizando pocos o ningún componentes de HTML5).

Es por eso, que la idea general de esta tesina es expandir ese horizonte en cuanto a los juegos con Phonegap, y se buscará desarrollar un juego de exploración de laberintos en mapas isométrico, con texturas basadas en sprites. Se intentara también utilizar manipulación de luces, ya sea a nivel global como de fuentes específicas, cada una con un color e intensidad variable, y su interacción con el resto de los elementos en el mapa.

A todas estas características, se les suma la adaptabilidad a las diferentes resoluciones y orientaciones de diferentes dispositivos, además de la posibilidad de poder jugar ya sea con el teclado, el mouse o la pantalla táctil.

REQUERIMIENTOS

- Establecer el marco principal de canvas de la aplicación.
- Delimitar unidades de espacio y la grilla principal del juego.
- Diagramar y customizar sprites del mapa, objetos y del personaje principal.
- Establecer mecánicas de movimiento del personaje principal e interacción con el mapa.
- Programar el movimiento del mapa a medida que el personaje principal avanza.
- Crear un mecanismo de luz global y fuentes de luz, y su interacción con el mapa.
- Crear un módulo de adaptabilidad a las diferentes resoluciones de diferentes dispositivos.
- Compilar un instalable para Android, utilizando Phonegap.
- Compilar un instalable para iOS, utilizando Phonegap.

DISEÑO DEL SISTEMA

FORMULACIÓN DEL MODELO

En primera instancia, poniendo un poco de contexto al desarrollo, se ha decidido realizar un prototipo de juego de exploración utilizando un mapa isométrico, es decir, con una vista visual del terreno tridimensional en dos dimensiones, en la que los tres ejes principales, al proyectarse, forman ángulos de 120º. La elección de esta premisa sigue varias razones:

- Llenar la pantalla con los elementos necesarios requiere bastantes más cálculos matemáticos que otro tipo de representación 2D (vista de tablero, juego de plataforma en 2D, etc.). Se debe calcular que baldosas mostrar en la pantalla, como presentarlas, como se mueve el personaje, la reacción del ambiente ante un clic del mouse o toque de la pantalla, etc.

- También se requiere un refresco y calculo continuo de los elementos presentes actualmente en pantalla. El hecho que todo el canvas este compuesto con varias capas superpuestas que interactúan entre si provee de un relieve mucho más profundo en el diseño.

- Todas las razones anteriores suponen que el mapa en si debe ser más grande que lo que se muestra en pantalla, para evitar errores visuales y de jugabilidad. Esto presupone la necesidad de más memoria y más ciclos de procesamiento para barajar todas estas actividades.

Estas razones de elegir un paradigma más complejo en lo referido a los juegos móviles es para demostrar que si se logra que este desarrollo funcione correctamente en cualquier dispositivo, cualquier juego 2D de menor complejidad debería presentar menores problemas de realizar.

El objetivo del juego sería la exploración de laberintos con fuentes de iluminación limitada, y ambientación de suspenso. Para esta tesina, se utilizará un mapa simple pero variado, como prueba de concepto.

ILUSTRACIÓN 14 – EJEMPLO DE JUEGO CON VISTA DE TABLERO (THE HEIST 2) Y JUEGO DE PLATAFORMAS EN 2D (AWESOMENAUTS) (WIKIPEDIA)

Con las decisiones básicas tomadas, se procede a darle forma a la idea y comenzar a afrontar los problemas que se pueden presentar.

ALMACENAMIENTO DEL MAPA

En primera instancia, se debe analizar como guardar el mapa. Este es uno de los elementos más importantes que se debe optimizar, ya que este va a ser leído y escrito numerosas veces en milésimas de segundo. Esto obliga a que el acceso al mismo sea realmente rápido.

Javascript por default no tiene arrays-multidimensionales, que sería ideal para esta situación. Las alternativas que existen son los arrays contenidos en otros arrays y tablas hash entre otros.

Además, cada baldosa que conforma el mapa debería llevar cierta información sobre las capas que interactúan en ella. Cada baldosa debería tener un indicador de tipo de baldosa, que tipo de elementos se encuentran en la misma, si interactúa con algún tipo de pared y como interactúa con elementos ambientales, como el personaje, luz, etc. También debería tener la opción de dejar vacías las opciones que no son necesarias, lo que sería un problema al tener que recorrerlas todas de manera secuencial.

Utilizando el Test manager de Chrome, se procede a realizar una serie de pruebas:

- Creando un arrays de arrays de dimensiones 1500x1500x10 (esta última dimensión correspondería a las propiedades de la baldosa), dio un resultado de memoria de 130.5Mbs y una velocidad de respuesta de 0.830ms. Aunque es considerable la memoria utilizada, es justa y este fue un ejemplo exagerado de arrays.

- Utilizando un array simple de 3000 celdas con todos los datos concatenados (es decir ancho, alto, capas, posiciones) dio un espacio en memoria de 102Mbs y un tiempo de lectura de 0.584ms. Este es un mejor tiempo de lectura, pero resulta ineficiente a la hora de determinar todas las capas, ya que en realidad son variables y no establecidas.

- Utilizando un array de todas las celdas, con un array interno para las capas resulto en un espacio de 105.6Mbs y una velocidad de respuesta de 0.934ms, lo que es mayor a los ejemplos utilizados.

- Utilizar un array de arrays con una pequeña clase determinada con capas y propiedades devolvió un tamaño en memoria de 162mb y una velocidad de respuesta de 0.890ms. A pesar que esta respuesta es un poco mayor a algunos de los ejemplos anteriores, este modelo es ideal a para modelizar cómodamente el entorno.

Luego del análisis, se decantó por un array general, donde cada elemento contiene las baldosas en cada una de las “filas” (o sea, todas las baldosas consecutivas en una línea) y donde cada baldosa es una pequeña clase con las propiedades correspondientes.

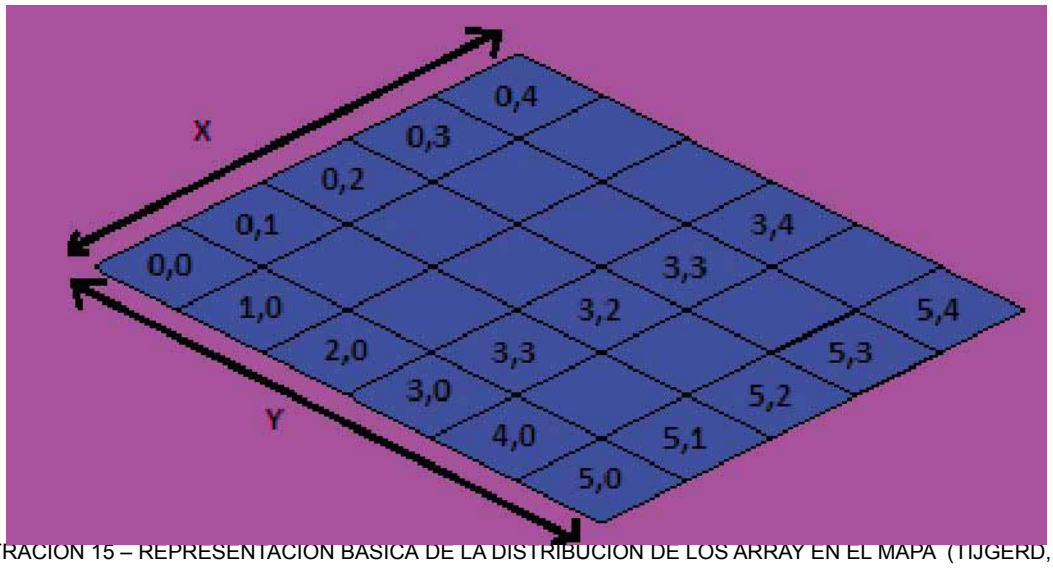


ILUSTRACIÓN 15 – REPRESENTACIÓN BÁSICA DE LA DISTRIBUCIÓN DE LOS ARRAY EN EL MAPA (TIJGERD, 2012)

DIBUJO DEL MAPA

El siguiente paso es el dibujo efectivo del mapa y sus elementos. La metodología elegida para este caso se conoce como **blitting**. Blitting es básicamente un método de representación de gráficos 2D donde se copia una imagen completa desde una parte de la memoria gráfica de la computadora a otra, para ser mostrada, procesada o lo que se desee.

El uso clásico para blitting es renderizar sprites con transparencias sobre un fondo. Por razones de velocidad es normalmente inviable moverse a través de cada pixel y encontrar si esto necesita ser visualizado. Esto trabaja perfectamente con mapas de bit para los sprites y las correspondientes máscaras. Los colores tienen mucha importancia ya que el fondo del sprite y el primer plano de la máscara son negros, almacenados como ceros binarios. Los fondos de las máscaras son blancos, almacenados como unos binarios.

Para esta situación, es ideal este escenario, ya que utilizando esta técnica, con unos sprites elegidos para representar los elementos del juego, se logra un buen desempeño a la hora de la representación y actualización del mapa.

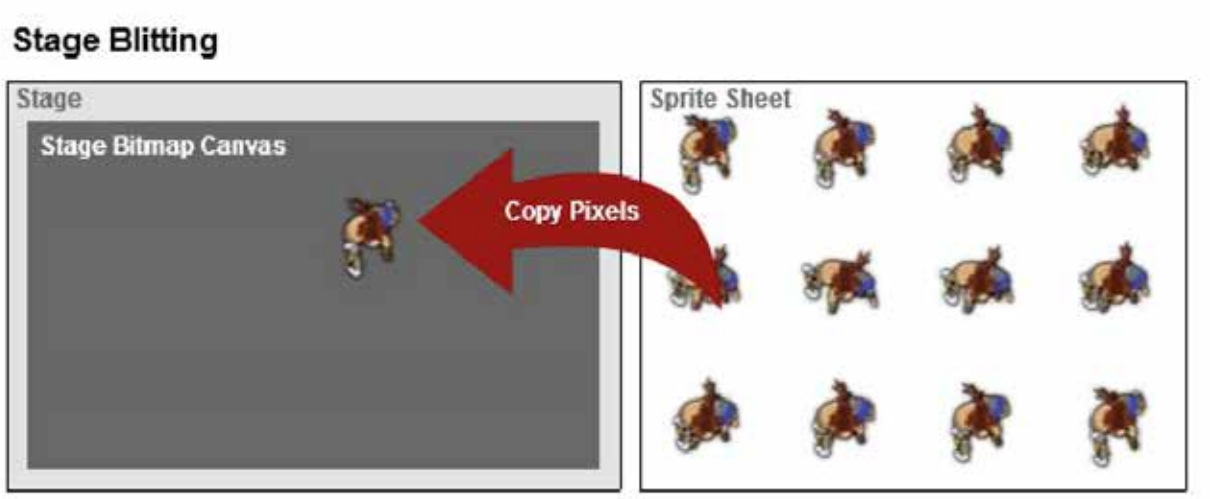
Se procede a hacer pruebas de los enfoques posibles sobre el blitting:

- Dibujar 1000 baldosas en un canvas de 1200x1080 (1080p) toma 4.08ms. Pasado a Fotogramas por segundo, daría un total de 245 FPS (este cálculo resulta de hacer 1/el tiempo que tomo).

- Para una mejor presentación y evitar errores visuales, se limpia el canvas entre los fotogramas. En caso de no hacerlo, se mejora el rendimiento, pero cuando hay muchos cambios, algunos elementos no se presentan correctamente.

- Existe la posibilidad de separar cada baldosa en su propio canvas, procesarlo y luego pegarlo en el canvas principal. Debería ser conveniente a la hora de procesar, pero por experiencia, tarda más que el llamado al sprite directamente.

Esto resulta en que para la optimización del dibujo de cada elemento en pantalla, se utilizará un método conocido como **tiling**, que consiste en la utilización de una sección de una hoja de sprites para cada uno de los componentes del juego, que se recortan fuera de pantalla y se agregan a medida que son necesarios. Esta técnica permite una fácil superposición de elementos.



En un estudio más exhaustivo, hay muchísimas variables que participan en el rendimiento del dibujo del canvas. Dentro de esta, se encuentra el tamaño total del canvas, la cantidad de baldosas a dibujar, la variedad de canvas a dibujar, cuan transparentes u opacas, el método de borrado y refresco del canvas y la invocación.

La siguiente tabla refleja los rendimientos de diferentes dispositivos y navegadores, aplicando los posibles métodos de dibujo en canvas. Para estas pruebas, se utilizaron los navegadores Chrome 39 (1440x900), Firefox 34 (1440x900) y los dispositivos Nexos 4 (1280x768) y Ipad Air (2048×1536). En pos de la simplicidad de los resultados, se obviaron las variables de opacidad y diferencia de sprites, ya que no arrojaron resultados significativos entre sí:

100 Baldosas

fillRect

	Chrome 39	Firefox 34	Nexus 4	Ipad Air
Desde Imagen	317 ms (315 FPS)	394 ms (253 FPS)	903 ms (110 FPS)	620 ms (161 FPS)
	142600 blits (449 per ms, 1426 per frame)	140300 blits (356 per ms, 1403 per frame)	27000 blits (29 per ms, 270 per frame)	78200 blits (126 per ms. 782 per frame)
Canvas Individuales	1184 ms (84 FPS)	585 ms (170 FPS)	1507 ms (66 FPS)	702 ms (142 FPS)
	142600 blits (120 per ms, 1426 per frame)	140300 blits (239 per ms, 1403 per frame)	27000 blits (17 per ms, 270 per frame)	78200 blits (111 per ms, 782 per frame)
Desde URI	302 ms (331 FPS)	417 ms (239 FPS)	1170 ms (85 FPS)	804 ms (124 FPS)
	142600 blits (472 per ms, 1426 per frame)	140300 blits (336 per ms, 1403 per frame)	27000 blits (23 per ms, 270 per frame)	78200 blits (97 per ms, 782 per frame)

clearRect

	Chrome 39	Firefox 34	Nexus 4	Ipad Air
Desde Imagen	321 ms (311 FPS)	417 ms (239 FPS)	867 ms (115 FPS)	621 ms (161 FPS)
	142600 blits (444 per ms, 1426 per frame)	140300 blits (336 per ms, 1403 per frame)	27000 blits (31 per ms, 270 per frame)	78200 blits (125 per ms. 782 per frame)
Canvas Individuales	1197 ms (83 FPS)	578 ms (173 FPS)	1637 ms (61 FPS)	699 ms (143 FPS)
	142600 blits (119 per ms, 1426 per frame)	140300 blits (242 per ms, 1403 per frame)	27000 blits (16 per ms, 270 per frame)	78200 blits (111 per ms, 782 per frame)
Desde URI	302 ms (331 FPS)	414 ms (241 FPS)	1211 ms (82 FPS)	797 ms (125 FPS)
	142600 blits (472 per ms, 1426 per frame)	140300 blits (338 per ms, 1403 per frame)	27000 blits (22 per ms, 270 per frame)	78200 blits (98 per ms, 782 per frame)

Sin Borrado

	Chrome 39	Firefox 34	Nexus 4	Ipad Air
Desde Imagen	342 ms (292 FPS) 142600 blits (416 per ms, 1426 per frame)	403 ms (248 FPS) 140300 blits (348 per ms, 1403 per frame)	784 ms (127 FPS) 27000 blits (34 per ms, 270 per frame)	612 ms (163 FPS) 78200 blits (127 per ms. 782 per frame)
Canvas Individuales	1283 ms (77 FPS) 142600 blits (111 per ms, 1426 per frame)	592 ms (168 FPS) 140300 blits (236 per ms, 1403 per frame)	1508 ms (66 FPS) 27000 blits (17 per ms, 270 per frame)	696 ms (143 FPS) 78200 blits (112 per ms, 782 per frame)
Desde URI	308 ms (324 FPS) 142600 blits (462 per ms, 1426 per frame)	421 ms (237 FPS) 140300 blits (333 per ms, 1403 per frame)	1060 ms (94 FPS) 27000 blits (25 per ms, 270 per frame)	792 ms (126 FPS) 78200 blits (98 per ms, 782 per trame)

1000 Baldosas

fillRect

	Chrome 39	Firefox 34	Nexus 4	Ipad Air
Desde Imagen	3104 ms (322 FPS) 1426000 blits (459 per ms, 1426 per frame)	3920 ms (255 FPS) 1403000 blits (357 per ms, 1403 per frame)	7796 ms (128 FPS) 270000 blits (34 per ms, 270 per frame)	6049 ms (165 FPS) 782000 blits (120 per ms, 782 per frame)
Canvas Individuales	11752 ms (85 FPS) 1426000 blits (121 per ms, 1426 per frame)	6001 ms (166 FPS) 1403000 blits (233 per ms, 1403 per frame)	15106 ms (66 FPS) 270000 blits (17 per ms, 270 per frame)	6783 ms (147 FPS 782000 blits (115 por ms, 782 per frame)
Desde URI	2920 ms (342 FPS) 1426000 blits (488 per ms, 1426 per frame)	4229 ms (236 FPS) 1403000 blits (331 per ms, 1403 per frame)	10140 ms (98 FPS) 270000 blits (26 per ms, 270 per frame)	7745 ms (129 FPS) 782000 blits (100 per ms, 782 per frame)

clearRect

	Chrome 39	Firefox 34	Nexus 4	Ipad Air
Desde Imagen	2726 ms (366 FPS) 1265000 blits (464 per ms, 1265 per frame)	4098 ms (244 FPS) 1403000 blits (342 per ms, 1403 per frame)	8090 ms (123 FPS) 270000 blits (33 per ms, 270 per frame)	6168 ms (162 FPS) 782000 blits (126 per ms, 782 per frame)
Canvas Individuales	10451 ms (95 FPS) 1265000 blits (121 per ms, 1265 per frame)	6104 ms (163 FPS) 1403000 blits (229 per ms, 1403 per frame)	15516 ms (64 FPS) 270000 blits (17 per ms, 270 per frame)	6746 ms (148 FPS) 782000 blits (115 per ms, 782 per frame)
Desde URI	2621 ms (381 FPS) 1265000 blits (482 per ms, 1265 per frame)	4341 ms (230 FPS) 1403000 blits (323 per ms, 1403 per frame)	10086 ms (99 FPS) 270000 blits (26 per ms, 270 per frame)	7812 ms (128 FPS) 782000 blits (100 per ms, 782 per frame)

Sin Borrado

	Chrome 39	Firefox 34	Nexus 4	Ipad Air
Desde Imagen	2731 ms (366 FPS) 1265000 blits (463 per ms, 1265 per frame)	3820 ms (261 FPS) 1403000 blits (367 per ms, 1403 per frame)	8112 ms (123 FPS) 270000 blits (33 per ms, 270 per frame)	6148 ms (162 FPS) 782000 blits (127 per ms. 782 per frame)
Canvas Individuales	10524 ms (95 FPS) 1265000 blits (120 per ms, 1265 per frame)	5855 ms (170 FPS) 1403000 blits (239 per ms, 1403 per frame)	15687 ms (63 FPS) 270000 blits (17 per ms, 270 per frame)	6800 ms (147 FPS) 782000 bits (115 por ms 782 per trame)
Desde URI	2576 ms (388 FPS) 1265000 blits (491 per ms, 1265 per frame)	3907 ms (255 FPS) 1403000 blits (359 per ms, 1403 per frame)	10460 ms (95 FPS) 270000 blits (25 per ms, 270 per frame)	7891 ms (126 FPS) 782000 blits (99 per ms, 782 per frame)

Estos resultados pueden ser visualmente mejor representados por los siguientes graficos

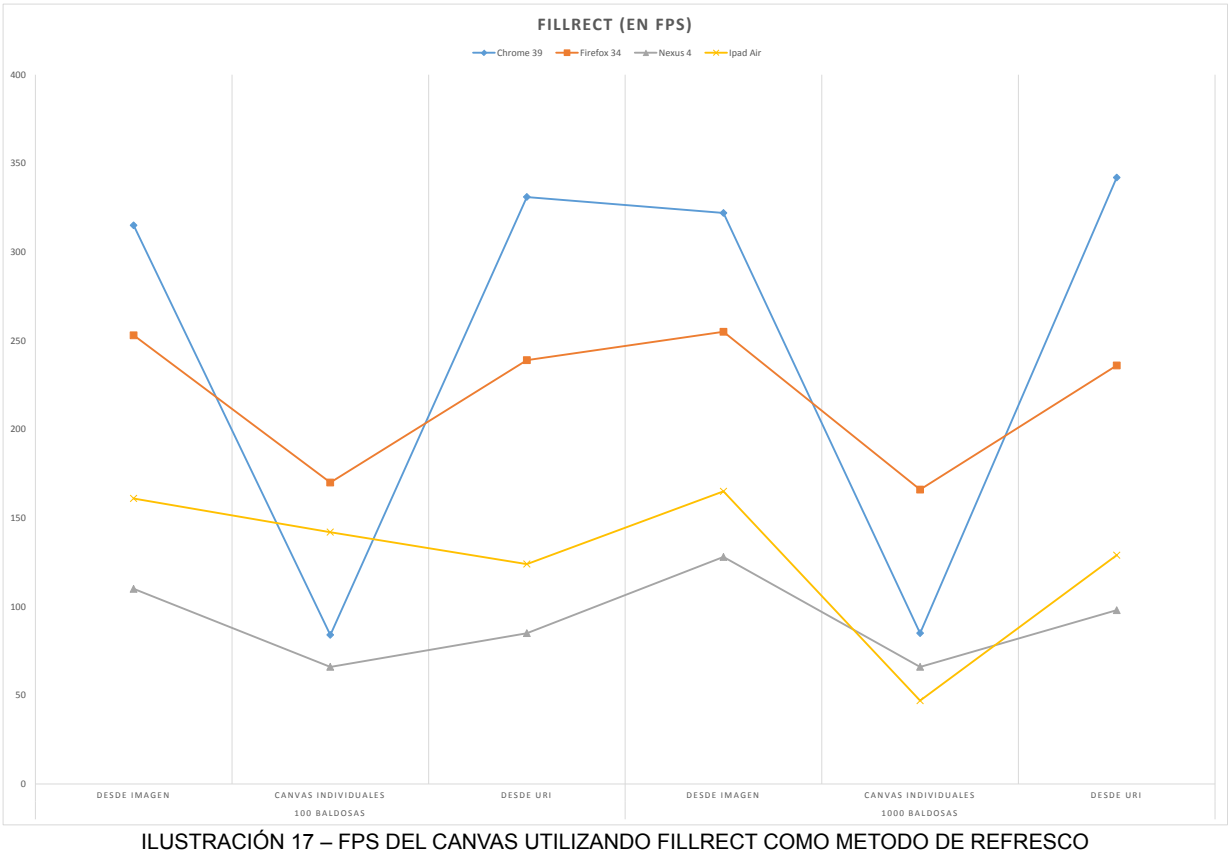


ILUSTRACIÓN 17 – FPS DEL CANVAS UTILIZANDO FILLRECT COMO METODO DE REFRESCO (DESARROLLO DE AUTOR)

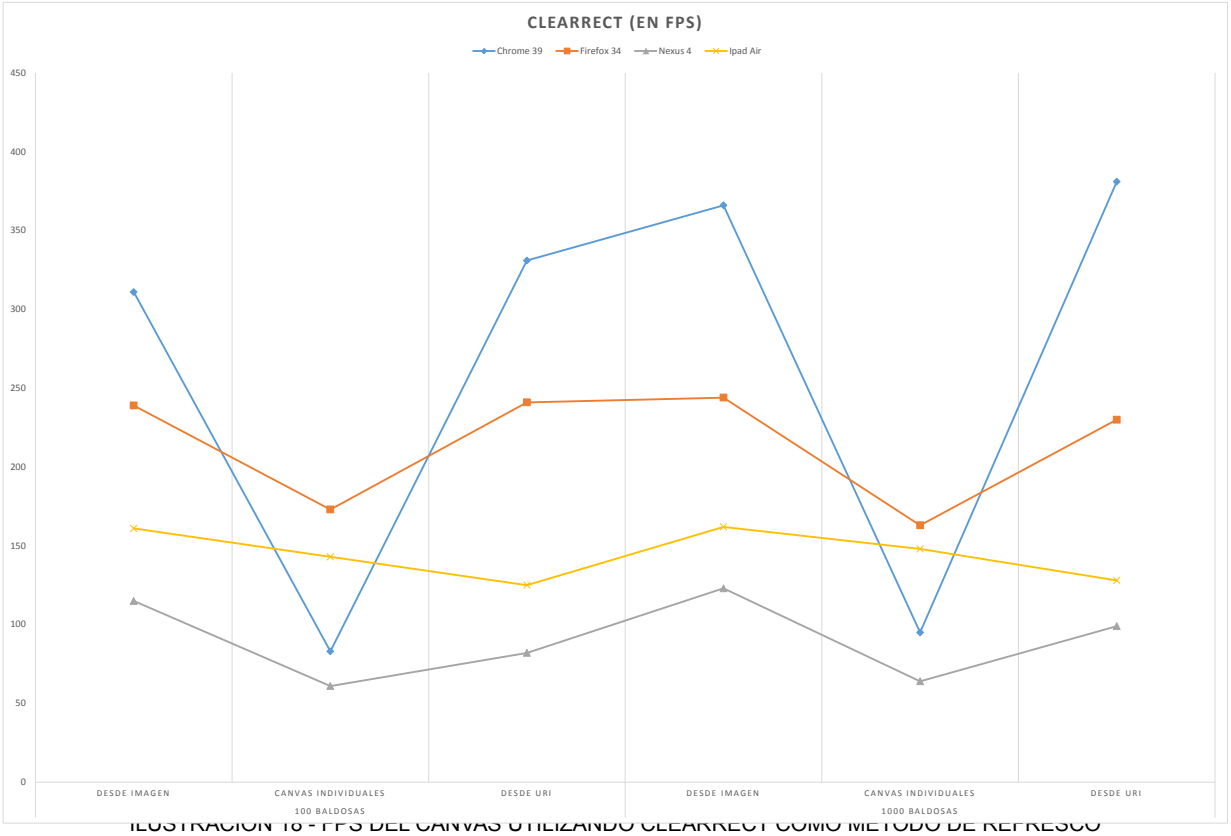


ILUSTRACIÓN 18 - FPS DEL CANVAS UTILIZANDO CLEARRECT COMO METODO DE REFRESCO

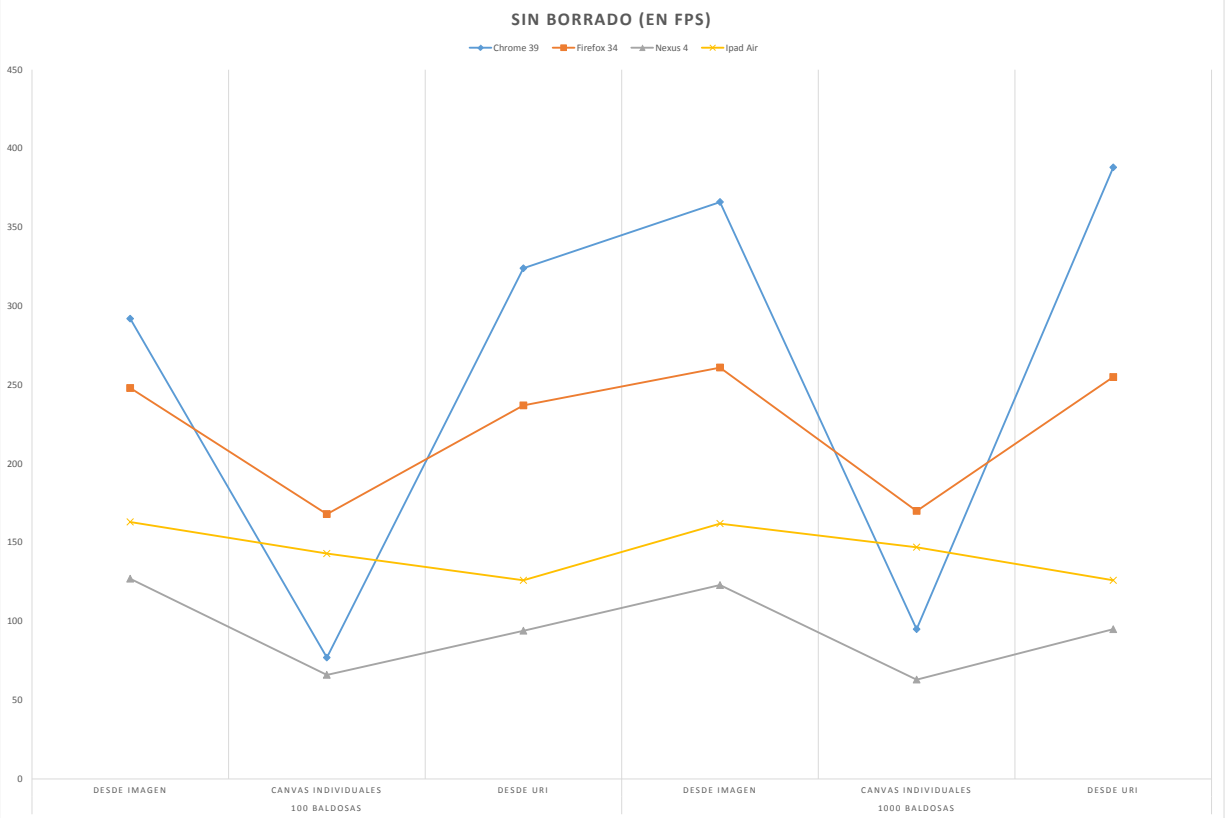


ILUSTRACIÓN 19 - FPS DEL CANVAS SIN BORRADO COMO MÉTODO DE REFRESCO

De estos resultados, se pueden sacar una serie de conclusiones:

- No importa la cantidad de baldosas, o los métodos utilizados, siempre se logra una performance de más de 60 FPS, que es el ideal de rendimiento. Está claro que más adelante, habrá muchas más capas y cálculos en juego, pero es una buena base.

- El método de refresco de las baldosas del canvas es prácticamente irrelevante. Pero a la hora de elegir, fillrect logra un mejor desempeño que clearRect o no borrar directamente, o sea que será el método elegido.

- Cada método se mantiene razonablemente invariable, sin importar la cantidad de baldosas a procesar.

- A pesar que el método de canvas individuales parecía prometedor por su conveniencia, quedó demostrado que es el menos conveniente, ya que presenta un menor desempeño que el resto de métodos de invocación (especialmente en Chrome, exceptuando iOS). Según parece, el llamado desde imagen directamente parecería el más consistente y conveniente de los tres, logrando un buen desempeño a través de las plataformas. Este método es posible de usar tanto en sitios web como aplicaciones, por lo que parece la elección natural.

ELECCIÓN DE LOS SPRITES

La elección o creación de los sprites es una parte visualmente importante para el juego. Deben ser lo suficientemente parecidas entre sí para que ninguna destaque y que todo parezca parte de una misma unidad. En la actualidad existen numerosas herramientas para la creación y edición de sprites, incluso a nivel de pixeles individuales. Y dentro de internet, se encuentran disponibles numerosas plantillas de sprites de numerosos juegos clásicos de diversas plataformas, lo que permite bastante libertad y variadas opciones a la hora de elegir.

Se optó por la utilización de los sprites del juego de diseño de parques de diversiones de 1999, **RollerCoaster Tycoon**, debido a su variedad de terrenos, elementos visuales y diseño de paredes y que ya se encuentra instalado en mi PC hogareña, por lo que es fácil de sacar los sprites individuales con capturas de pantalla y programas de edición de imágenes.

En cuanto al personaje principal, se eligió un sprite de fantasma, utilizado en la versión de iOS del juego de 1994 **Final Fantasy VI**.

ILUSTRACIÓN 20 - EJEMPLO DE CREACIÓN CON ROLLERCOASTER TYCOON Y SPRITE DEL PERSONAJE

Con herramientas de edición de imagen, se retiraron los sprites de las creaciones, que incluyen paredes, arbustos, árboles, caminos, terrenos y elementos varios.

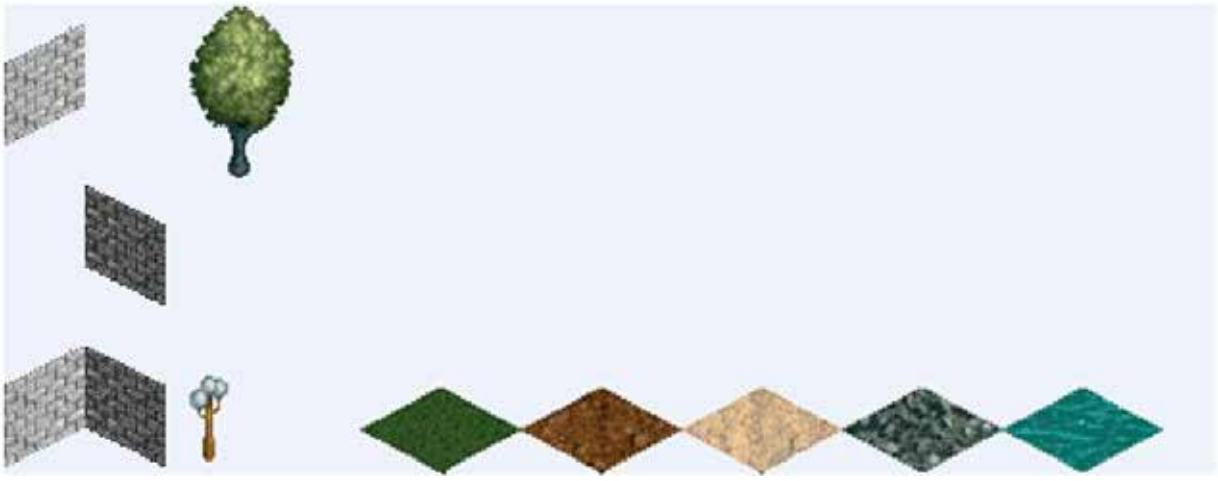


ILUSTRACIÓN 21 - EJEMPLOS DE SPRITES EXTRAÍDAS

MÉCANICA DE MOVIMIENTO DEL PERSONAJE

A la hora de determinar el movimiento del personaje, se debe hacer un cálculo preciso sobre en qué parte de la baldosa se encuentra. Se podría establecer una cuadricula donde el personaje se mueva al centro de cada baldosa y nada más, pero eso le quitaría fluidez al movimiento y haría los controles con teclado bastante aparatoso.

En el caso de una cuadricula de celdas cuadradas, el cálculo sería mucho más simple. Pero al estar trabajando con celdas simétricas, se requiere un poco más de cálculos matemáticos, sobre las diagonales que conforman la celda.

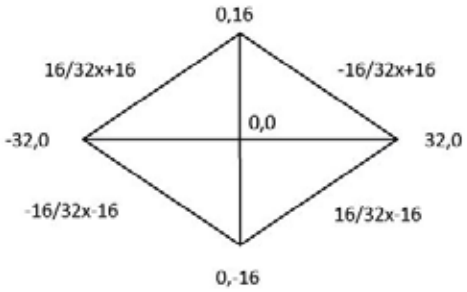


ILUSTRACIÓN 22 – DIAGRAMA DE COORDENADAS DE LAS POSICIONES DE CADA BALDOSA (MAGLIOLA, 2011)

Estos cálculos sirven tanto para el movimiento del personaje como para determinar la ubicación del puntero.

Luego se debe determinar las limitaciones del personaje en cuanto a sus movimientos. Se deben establecer reglas para que se respeten paredes, árboles y demás objetos que no pueden ser atravesados.

Aquí es donde entran en acción los flags o propiedades. Cada baldosa tiene una serie de flags asociados. Por ejemplo, “NonWalkable” significa que esa celda no puede ser transitada y “BlockWalkFrom…”, donde los puntos suspensivos son reemplazados por las 4 direcciones isométricas (NE, NO, SE, SO). Cuando el personaje se mueve, chequeamos el estado de estos 2 flags de las 2 celdas entre las que se mueve para determinar si se debe bloquear el movimiento.

En el caso de que el movimiento se suceda en N, S, E y O, caso que sucede cuando el personaje camina exactamente en el vértice, se hace un doble checkeo de las 2 diagonales. Si alguna de las 2 permite el paso, entonces el personaje puede ingresar en esa celda.

ILUMINACIÓN

Una buena práctica para realizar efectos de iluminación es tomar un canvas perfectamente iluminado e ir oscureciendo toda la plantilla de sprites pixel por pixel, multiplicándolo cada color por un valor menor a 1. Si se guardan estos sprites oscurecidos progresivamente, y guardamos los diferentes “niveles de luz”, después podemos elegir para cada celda cuál de estos niveles dibujar, e insertarlos individualmente, dependiendo de la distancia con la fuente de luz.

El problema de esto es que si para cada celda buscamos cual es la fuente de luz más cercana, esto reduce la velocidad de dibujo considerablemente. La posición de las luces no cambia muy a menudo (siendo la excepción el caso de que el personaje sea una fuente de luz en sí mismo), por lo que podemos pre-calcular un “mapa de luces”, que guarda niveles de luz pre-calculados para cada celda del mapa, y luego, solo referir al mismo cuando se renderiza.

Para calcular el mapa de luces, necesitamos calcular la proyección de luz de todas las fuentes del mapa. Esto se realizaría comenzando desde el centro y procesando las celdas circundantes sucesivamente. Se crea un array con las 8 celdas circundantes, y para cada una, pongo aquellas celdas circundantes que no se repiten.

Pero además, se chequea un flag de bloqueo de luz, que me dice si la luz puede alcanzar la celda desde esa posición. En caso negativo, la celda no se agrega a la cola. De esta manera, si me encuentro en una habitación, puedo bloquear la luz para que no atraviese las paredes, pero sí que pase por puertas y pequeñas aberturas, dando un efecto más realista, especialmente para la luz proveniente del personaje.

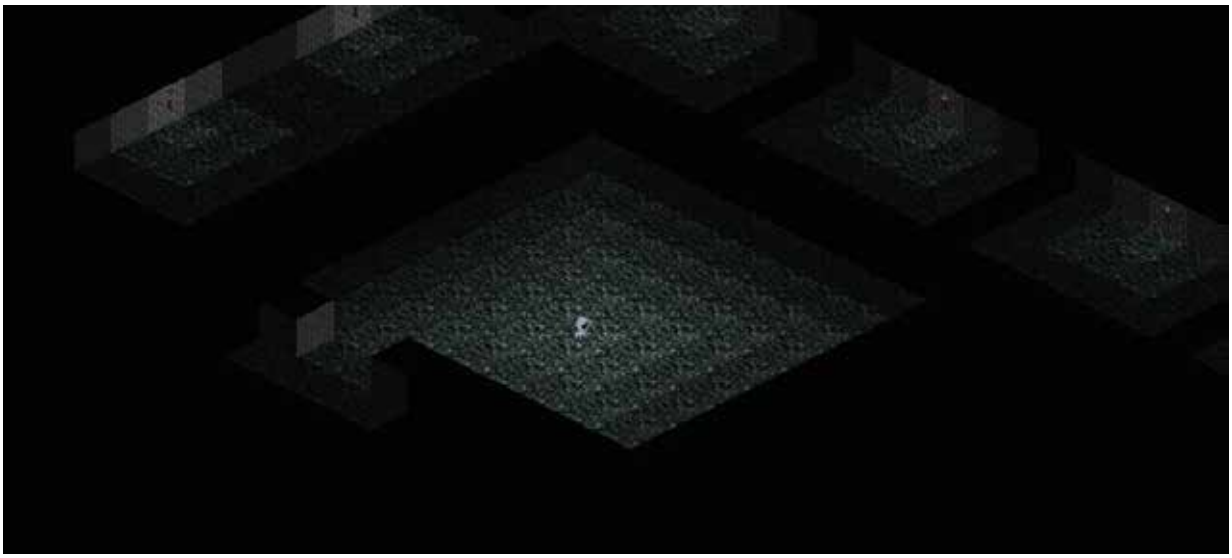


ILUSTRACIÓN 23 – EJEMPLO DE INTENSIDAD Y DISPERSIÓN

MODELO DE DATOS Y PROCESOS

El proyecto consta en una página web html, con un canvas sobre el cual interactúan una serie de archivos JavaScript, donde cada uno cumple una función de movimiento, renderización, o lógica para presentar funcionalmente al juego en cuestión.

Los archivos JavaScript conforman cada uno de los módulos funcionales y son los siguientes:

- **window.js:** Dentro de este módulo se encuentra toda la lógica aplicada a la capacidad de multiresolución de la aplicación. Aquí se manejan todos los eventos de dimensionamiento de la pantalla, adaptable a todo tipo de resolución. También se encarga del redimensionamiento, ya sea porque el usuario cambio manualmente el tamaño del navegador o porque el usuario giro el dispositivo móvil, produciendo un cambio de orientación de vertical a horizontal, o viceversa.

- **character.js:** En este módulo se encuentra toda la modelización del personaje principal, con sus características y flags inherentes. Asimismo, se encuentran contemplados todos los eventos de movimiento, y ubicación del personaje principal en el canvas.

- **drawing.js:** Este es el módulo encargado del dibujo efectivo del canvas, llenándolo de los sprites de celdas, elementos y del personaje en las posiciones correspondientes. También está encargado del redibujado de aquellas celdas que deben ser actualizadas debido que el personaje ha pasado por ellas, para evitar imágenes fantasmas en la renderización. Además, se encarga de todos los cálculos de luminosidad de las celdas, cambios de intensidad y variaciones globales de luz.

- **engine.j**s: Aquí se unifica el resto de los módulos, y orquesta todo el funcionamiento. En primera instancia, carga el mapa seleccionado en memoria, y luego, comienza la renderización efectiva del mismo, comenzando a ejecutarse el resto de los módulos. También contempla todos los datos de desempeño, desde los FPS hasta los blits por segundo.

- **global.js:** Este módulo contiene métodos y modelizaciones diversas, que utilizan el resto de los módulos. Entre ellos, se encuentran los métodos de obtención de un sprite desde una imagen (que luego será un elemento del canvas), oscurecimiento global, copias de elementos, las características de cada punto del canvas y sus métodos, y códigos de direccionamiento.

- **input.js:** En este módulo se manejan los diferentes tipos de interacción del usuario con la aplicación, para mover el personaje. Aquí se contempla el movimiento por las fechas del teclado, movimientos por clic de mouse y movimiento del personaje por toques en la pantalla de un dispositivo móvil.

- **map.js:** Este es el módulo donde se modelizan todos los elementos visuales de la aplicación. Aquí se especifican métodos y propiedades de flags, celdas, capas, luz y diferentes constantes. También se establece el mapa como una unidad y se especifican sus métodos y variables, además de crear los mapas que se utilizaran, con las características y elementos que lo componen.

- **maplogic.js:** Aquí se encuentran algunos métodos diversos generales sobre el comportamiento del mapa, como un efecto ocaso o la ubicación inicial del personaje cuando el mapa es cargado inicialmente.

- **viewport.js:** Aquí se manejan todos los elementos visibles en la pantalla del navegador o dispositivo utilizado. Estipula relaciones entre posiciones del mapa y la vista del navegador, traducción de movimiento del mouse contra posición efectiva en el mapa, y manejo de scroll de pantalla cuando el personaje se dirige hacia afuera de la parte visible del mapa.

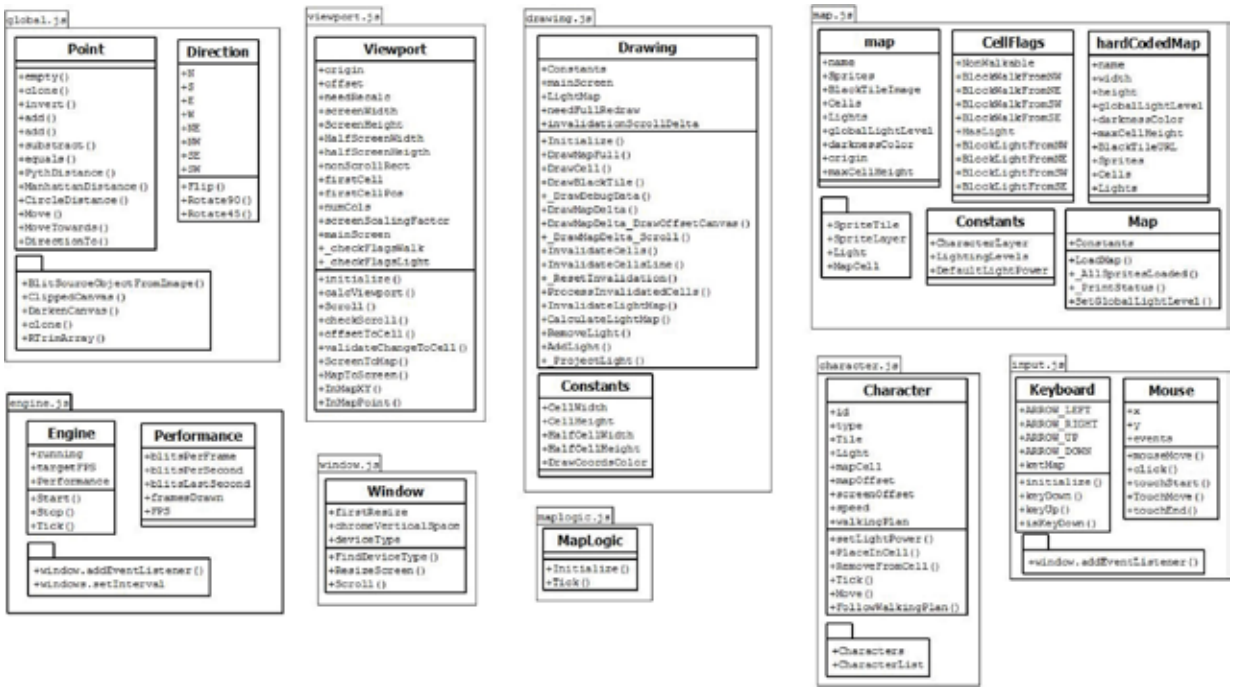


ILUSTRACIÓN 24 – DIAGRAMA DE ARCHIVOS Y CLASES, CON SUS RESPECTIVAS PROPIEDADES Y MÉTODOS (DESARROLLO DE AUTOR)

IMPLEMENTACIÓN DEL PROTOTIPO

Una vez terminada la aplicación y funcionando correctamente, se procede a la serie de pruebas de instalación en las plataformas de destino: **Android** (https://www.android.com/), **iOS** (https://www.apple.com/ios/) y **Windows Phone** (http://www.windowsphone.com/en-us). Hay 3 métodos distintivos que provee Phonegap para la prueba de los proyectos en las plataformas destino, que serán presentados en cada plataforma a modo de prueba, más un test web:

- **Phonegap Plugin:** Phonegap permite compilar proyectos con los entornos de desarrollos propios de cada plataforma, solo necesitando un plugin que se agrega a cada proyecto, para interpretar los archivos html.

- **Phonegap Build:** Servicio online que provee Phonegap, donde solo es necesario enviar el código fuente en un archivo Zip, con cierta estructura y un archivo XML con información específica del compilador. Una vez mandado, el sitio provee descargas de archivos instalables para Android, Windows Phone y iOS.

- **Phonegap Developer App:** Aplicación disponible en los marketplaces de Android, Windows Phone y iOS, que funciona como un encapsulador de un sitio web provisto. Este sistema se basa

en una aplicación servidora donde se encuentra la aplicación web HTML5 y la app en cuestión, donde se le especifica la ip del servidor y renderiza el resultado.

- **Navegador predeterminado:** Se corre la aplicación web desde un servidor local y se comprueba el desempeño de la aplicación web.

ANDROID

PHONEGAP PLUGIN

En el caso de Android, Google provee un entorno de desarrollo llamado **ADT** (Android Development Tools), con todas las herramientas necesarias para desarrollar cualquier aplicación para los sistemas operativos Android en todas sus versiones.

La integración con Phonegap constó de la instalación de un plugin para eclipse llamado **PhonegapForAndroid.** Con este plugin, se agregaba un tipo de proyecto Phonegap dedicado, con todos los elementos básicos y necesarios que integran un instalable utilizando las herramientas de Phonegap. Solo fue necesario una clase java, que direccionara hacia el archivo html principal.

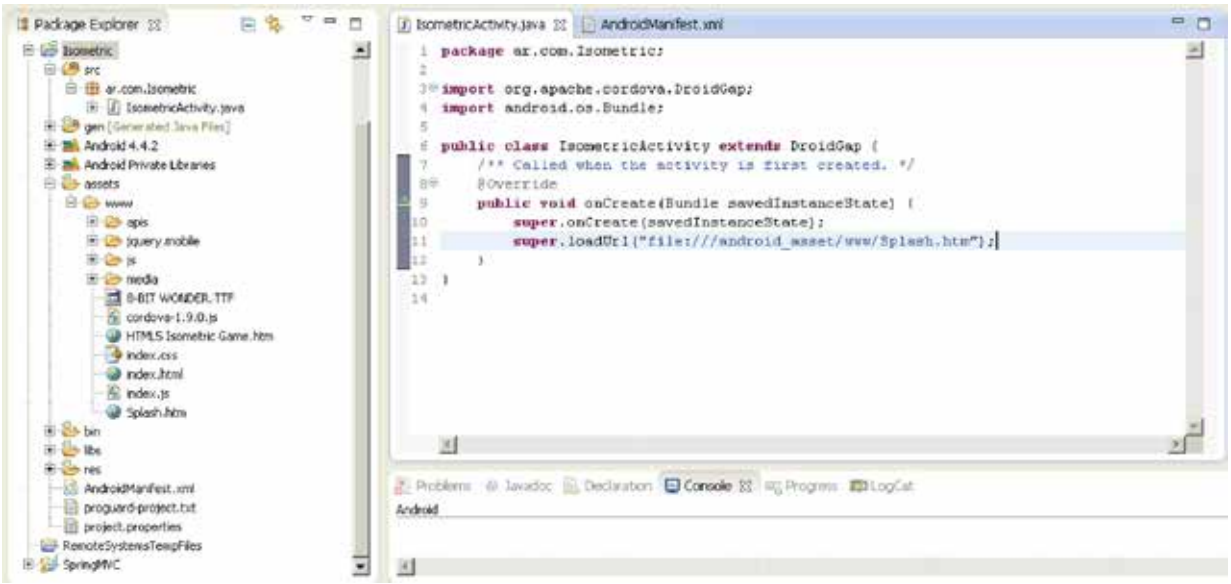


ILUSTRACIÓN 25 – ORGANIZACIÓN DEL PROYECTO, ARBOL DE ARCHIVOS Y PROGRAMACIÓN JAVA

Una vez realizada la programación básica, solo fue necesario revisar el archivo **AndroidManifest.xml**, donde se especifican el nombre y la versión de la aplicación que se está desarrollando, la versión mínima de Android que soporta, para que tipo de pantallas está desarrollado, aquellos permisos que solicita del dispositivo, y todos aquellos cambios que permite en su interfaz (teclado, cambio de pantalla, giro de dispositivo, etc.).

Una vez programado todo, solo fue necesario correr el proyecto, y esto desemboco en un archivo instalable de Android (**apk**). Una vez compilado, se pasó a un dispositivo Android y se testeo la instalación.

ILUSTRACIÓN 26 – PANTALLA DE INSTALACIÓN Y POST-INSTALACIÓN

Una vez instalado, se pudo comprobar que el desempeño era el esperado, con un 30 FPS objetivo, muy buen blitaje, redimensionamiento correcto, mecánicas de juego y efectos de buen desempeño y sin errores visuales. Para datos más precisos, estamos hablando de 30 FPS, 0.5k BPS, una resolución de 384x567, 7 blits por segundo, y un promedio de 17 celdas invalidadas, todo eso con una jugabilidad fluida.

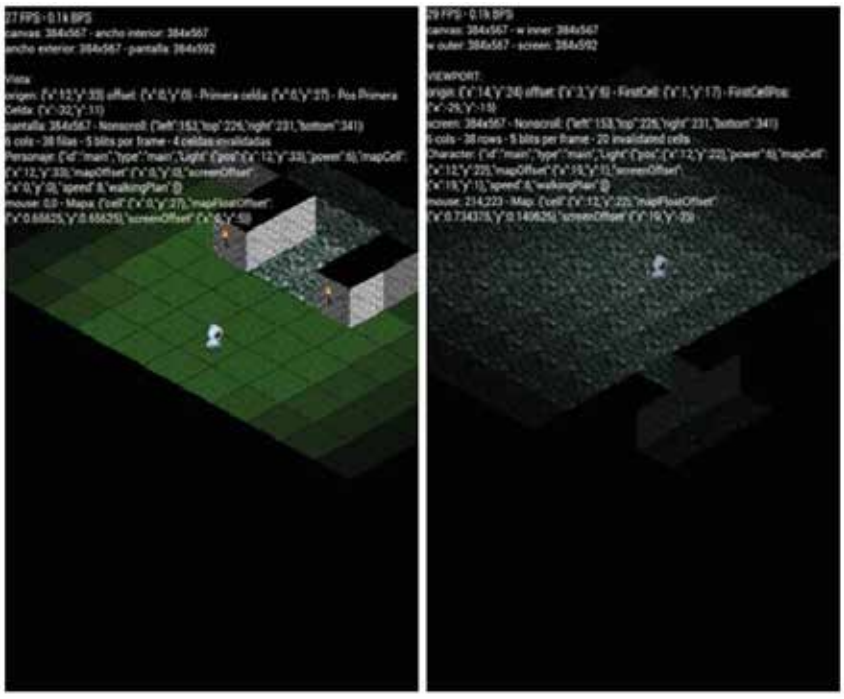


ILUSTRACIÓN 27 – MUESTRAS DE PERFORMANCE Y RENDERIZACIÓN

PHONEGAP BUILD

El instalador se compilo correctamente, el dispositivo lo reconoció y lo instalo sin mayor problema. Pero a la hora de probar la jugabilidad, se nota que a pesar de renderizar y performar de manera adecuada, el feedback táctil y de posición no se manifiesta de manera correcta. Sucede que ante cualquier toque de pantalla, el personaje se mueve hacia arriba, en vez de la dirección deseada.

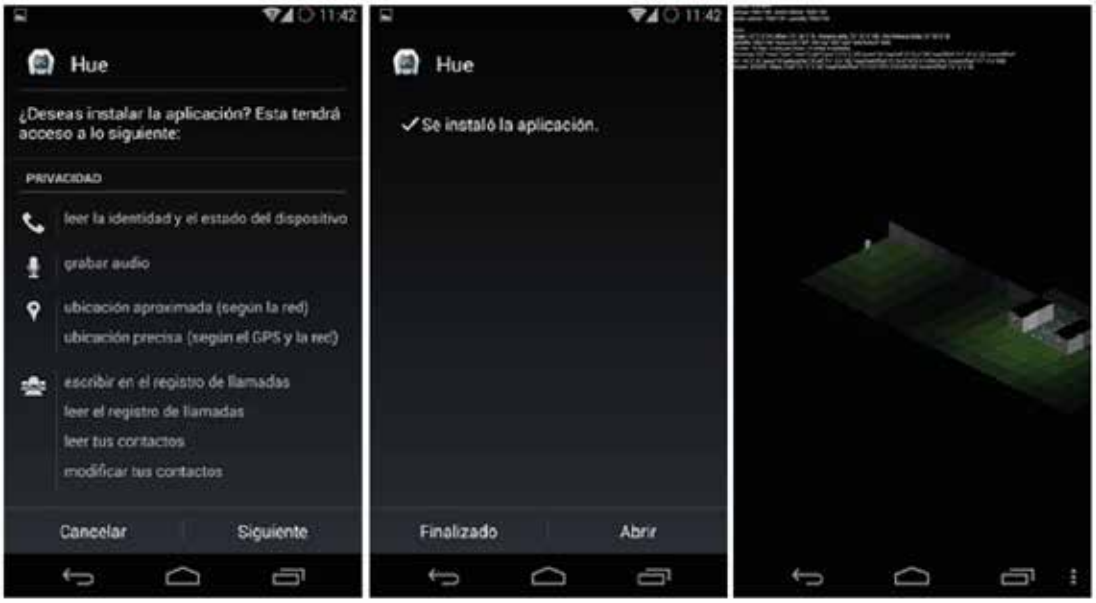


ILUSTRACIÓN 28 – INSTALACIÓN CORRECTA Y DIRECCIÓN A LA QUE TIENDE CUALQUIER TOQUE

PHONEGAP DEVELOPER APP

La aplicación se conectó correctamente al servidor y no presentó al ejecutarse, desarrollándose como era pretendido

ILUSTRACIÓN 29 – CONEXIÓN Y FUNCIONAMIENTO

NAVEGADOR PREDETERMINADO

En el navegador, la aplicación conectaba correctamente, rendía y lucía como era pretendido, al igual que en Phonegap Developer.

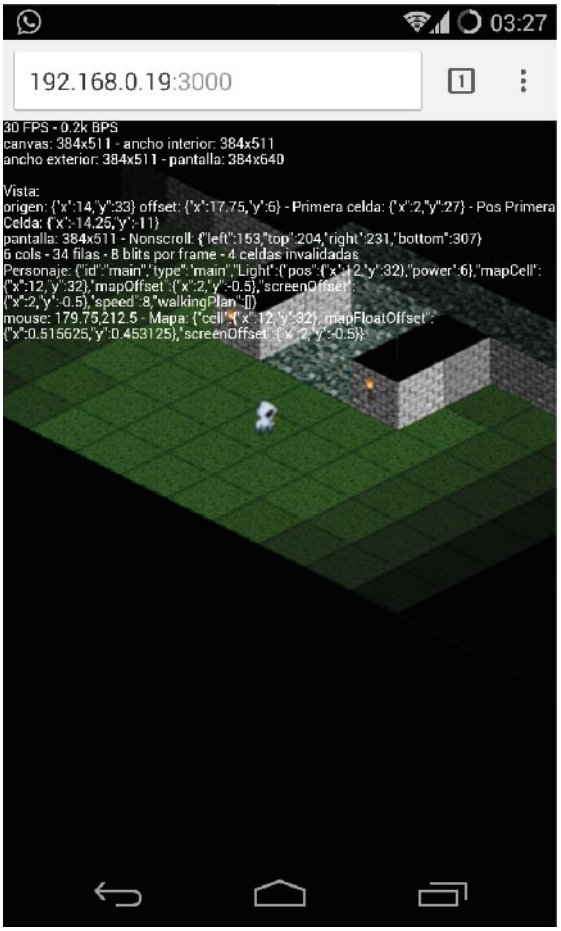


ILUSTRACIÓN 30 – RESULTADO EN EL NAVEGADOR

WINDOWS PHONE

PHONEGAP PLUGIN

Para utilizar el desarrollo para Windows Phone 8, era necesario contar con el sistema operativo de escritorio Windows 8.1 Professional, un procesador con Hardware-Assisted Virtualization, SLAT (Second Level Addres Translation) y DEP (hardware-based Data Execution Prevention).

Ninguna de las maquinas con las que se contaba o se tenía acceso cumplían con todas las características al mismo tiempo, por lo que esta opción era inviable.

PHONEGAP BUILD

En este caso, la descarga del archivo instalable para Windows Phone 8.1 (**xap**) no fue problema, y se descargó satisfactoriamente desde la web.

El problema se presentó a la hora de la instalación, ya que a pesar que el dispositivo interpretaba correctamente el instalable, manifestó la imposibilidad de instalar la aplicación, alegando que había un problema con la aplicación de esta compañía (Phonegap, donde se compiló). Se realizaron cambios en el código, pero todos presentaron el mismo problema.

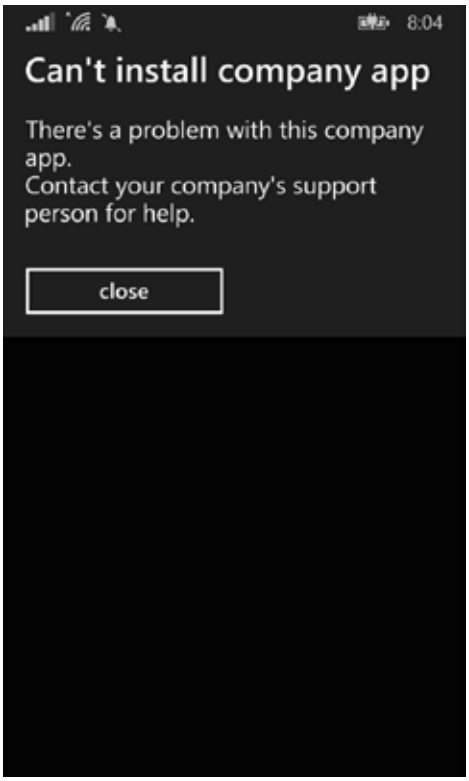


ILUSTRACIÓN 31 – IMAGE DE ERROR AL QUERER INSTALAR LA APLICACIÓN

PHONEGAP DEVELOPER APP

En esta prueba, se ejecutó el servidor de Phonegap. Pero a pesar de conectarse satisfactoriamente, la aplicación se detiene en la carga de los sprites, en diferentes elementos en cada prueba. Finalmente, nunca finalizaba de cargar y la aplicación nunca comenzaba realmente.

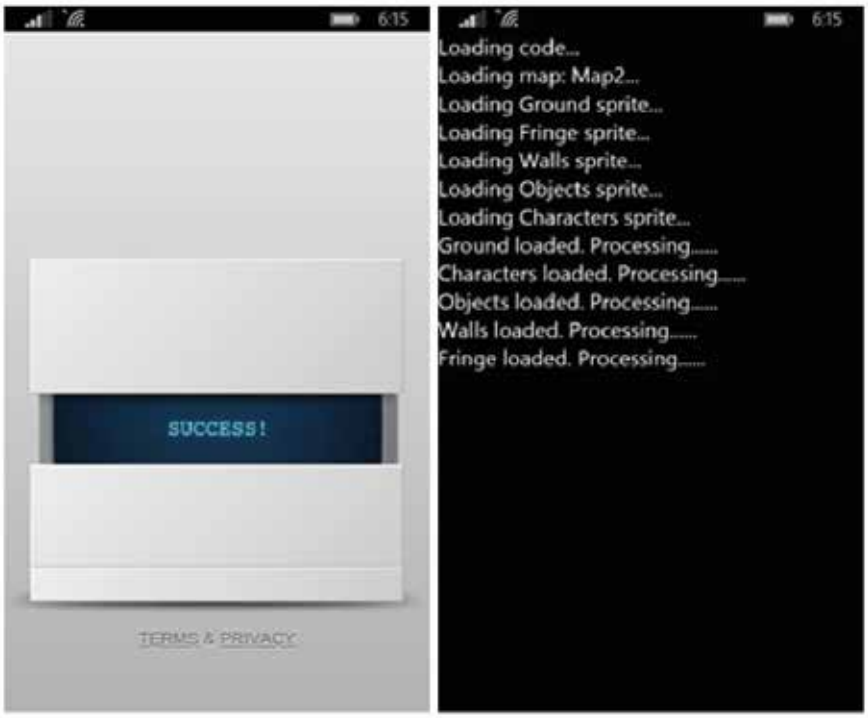


ILUSTRACIÓN 32 – CONEXIÓN Y ERROR AL CARGAR LOS ELEMENTOS DE LA APLICACIÓN

NAVEGADOR PREDETERMINADO

En el navegador predeterminado, se reproducía el mismo problema que con **Phonegap App Developer.**

ILUSTRACIÓN 33 – ERROR AL CARGADO DE ELEMENTOS, QUE NO LLEGABA A COMENZAR

IOS

PHONEGAP PLUGIN

El desarrollo de aplicaciones para iOS 7 hacia arriba requería contar con una computadora de escritorio o laptop Apple, con sistema operativo OS X Lion o más reciente, y una suscripción de desarrollador oficial (100 US$) para poder compilar los instalables. Si se quería probar con algún dispositivo, se debía tener un dispositivo con Jailbreak instalado.

Debido a que no se contaba con un dispositivo que cumpliera con todos los requisitos, ni se podía pagar la suscripción de desarrollador, fue inviable probar este método.

PHONEGAP BUILD

La opción de Phonegap Build también era inviable por el momento, ya que para poder compilar el instalable, era necesaria una clave segura, provista por la cuenta de desarrollador, que no pudo ser adquirida.

ILUSTRACIÓN 34 – ERROR PRESENTADO POR PHONEGAP BUILD

PHONEGAP DEVELOPER APP

En el caso de Phonegap Build, la aplicación conectó correctamente, y performaba correctamente, con un buen nivel de fotogramas por segundos. Pero a pesar de todo esto, había sectores donde faltaban sprites, como por ejemplo en esquinas y elementos, que no tenían razón de estar ausentes.

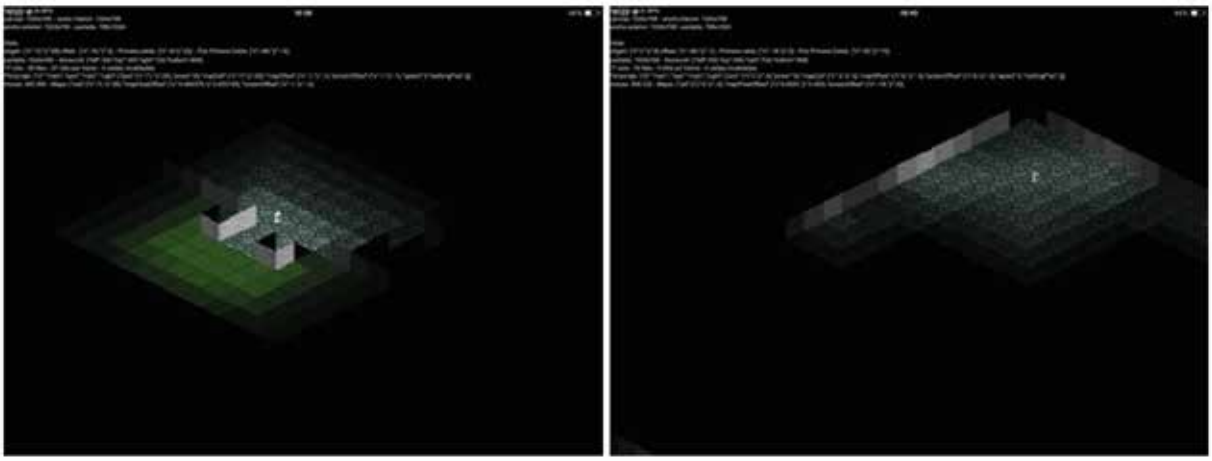


ILUSTRACIÓN 35 – IMÁGENES CON ESQUINAS Y ANTORCHAS FALTANTES

NAVEGADOR PREDETERMINADO

En el navegador, la aplicación tenía el mismo desempeño que con Phonegap Build, pero con los mismos problemas de faltantes de elementos.

ILUSTRACIÓN 36 – IMÁGENES EN EL NAVEGADOR, CON FALTANTES DE IMÁGENES

RESUMEN DE RESULTADOS

Pruebas		Android	iOS	Windows Phone
Phonegap Plugin	Performance:	Correcta (29 FPS)	No testeado por falta de cuenta paga de desarrollador	No testeado por falta de requerimientos de hardware
	Renderización:	Correcta		
Phonegap Build	Performance:	Correcta (28 FPS)	No testeado por falta de cuenta paga de desarrollador	Error al instalar
	Renderización:	Errores de mapeo de toques en pantalla		
Phonegap App Developer	Performance:	Correcta (30 FPS)	Correcta (58 FPS) Errores al presentar ciertos sprites	No termina de cargar los elementos
	Renderización:	Correcta		
Navegador Default	Performance:	Correcta (30 FPS)	Correcta (57 FPS) Errores al presentar ciertos sprites	No termina de cargar los elementos
	Renderización:	Correcta		
Conclusión		**Perfectamente usable**	**Prometedor, requeriría cuenta de desarrollador para pruebas completas**	**Inviable**

CONCLUSIONES

HTML5 ya es a esta altura un término vox-populi, incluso para aquellas personas no versadas en temas informáticos o de desarrollo web. El común denominador sabe que si un sitio web se adapta tanto a su celular como a su navegador de escritorio, es porque esta codificado con estándares HTML5. Si un video puede ser visto en un celular, es gracias a que el reproductor y el video, ambos cumplen con requerimientos HTML5.

Antes de este estándar, los sitios web se veían con el formato que se presentan en los navegadores de una computadora de escritorio. Esta situación no es ideal, ya que debido a la reducida resolución de los nuevos dispositivos móviles, la navegación no era exactamente fluida. La única manera que había de palear estas deficiencias era utilizando Adobe Flash, una extensión multimedia, que permitía hacer sitios adaptables.

Pero esta solución en móviles es poco viable e inconveniente, ya que requería de mucho poder de procesamiento, elemento que en los primeros dispositivos móviles no abundaba.

Era hora de pensar alguna alternativa, y ahí es donde entraba HTML5. La transición era lenta y costosa, ya que Flash se encontraba arraigado en muchos sitios web.

Pero ahí entro en acción uno de los impulsores de HTML5 fue Steve Jobs, que sabía que si un cambio era necesario, tenía que ser HTML5. A él le gustaba pensar en el futuro de la tecnología y ser parte de ella, e impulsando HTML5 sabía que estaría haciendo algo que nadie se atrevería porque sería señal de la necesidad de abandonar plugins como Flash, algo por lo cual recibió muchas críticas negativas, pero en realidad fue una decisión muy importante. Esa decisión hizo que otras empresas tomaran consciencia y empezaran a ver una tecnología que aún estaba como un borrador para empezar a implementarlo. (Gomez Prieto, 2013)

Fue gracias a esta decisión visionaria que ahora todas las grandes empresas emplean estándares HTML5, como por ejemplo Google, que migro todos sus videos y anuncios flash a HTML. Incluso Adobe, propietarios de Flash, se encuentran migrando a la nueva generación móvil.

En cuanto a la hipótesis de la tesina, se puede llegar a la conclusión de que se no se cumple completamente. Actualmente, esta alternativa no se encuentra en un nivel de maduración suficiente como para transformarse en una opción 100% segura, especialmente en juegos, ya que presenta diferencia en desempeño y renderización en las diferentes plataformas enmarcadas en el contexto de la tesina.

A pesar de eso, cabe destacar que los juegos son un caso límite, utilizado en esta tesina para demostrar si rendía hasta casos extremos, con gran cantidad de cálculos matemáticos, renderizaciones, refrescos de pantalla y actualizaciones. En cambio, con aplicaciones de texto (que puede ser un diario online, sitio web empresarial u otro tipo), se podría decir que es posible desarrollar un código íntegramente basado en HTML5, JavaScript y CSS3, de tal manera que sea multiresolución, y con la ayuda del framework Phonegap, sensible de ser portabilizado a cualquiera de las plataformas móviles más populares, todo con un decente nivel de desempeño.

A pesar de que todavía las aplicaciones móviles nativas tienen la ventaja a la hora de mejor integración, esta brecha se va acortando exponencialmente, debido a todos los avances que se producen en el área la programación móvil, sector en franco ascenso y sin intención de detenerse, por lo menos hasta la actualidad.

Pero todo este trabajo de investigación, programación, desarrollo y testeo no hubiese sido posible sin conocimientos y herramientas claves adquiridas en el estudio de la carrera. Materias como Programación, en todas sus expresiones, fueron esenciales para la decisión de la temática y orientación de la tesina, además de proveer todo el conocimiento con el que contaba al comenzar, dándome una gran ventaja en cuanto al pensamiento lógico y estructurado necesario para este tipo de desarrollos. A todo esto se le suman aquellos aportes pertinentes a pautas y buenas prácticas, como la creación de prototipos para clases modelizables, capitalización y jerarquía de clases, etc.

También se debería nombrar a las materias correspondientes a Arquitectura de Software, que proveyeron extensos conocimientos sobre todo lo referente a administración, diagramación, organización, construcción, documentación y mantenimiento de un proyecto. Contar con estas herramientas significo que estas actividades se realicen con gran fluidez.

Tampoco se debe olvidar agregar todo lo aprendido en las materias de Taller de Tesina y sendas Habilitaciones Profesionales, que proporcionaron formación y todo tipo de información referente a las prácticas profesionales y más específicamente, los requerimientos y mejores prácticas correspondiente al desarrollo de una tesina.

A todo esto hay que sumarle los conocimientos que hubo que adquirir sobre las específicas plataformas sobre las que se desarrolló. Hubo que aprender, investigar, leer libros y artículos online sobre sintaxis HTML5, operatoria del canvas y sus elementos, métodos y capacidades de Javascript, organización de plantillas CSS3, todo esto para poder comenzar. Se realizaron investigaciones sobre adaptabilidad, posicionamiento de mouse y toques táctiles, manipulación de sprites en tiempo real y cálculos de posicionamiento y movimiento, para nombrar algunos de los temas.

Además, hubo que adquirir nociones sobre todo el proceso de Phonegap, como había que programar para lograr una portabilidad total en las diferentes plataformas, y que era necesario para aprovechar todas las capacidades que la plataforma proveía.

FUTURAS INVESTIGACIONES Y DESAROLLOS

Con todas las nociones y aprendizajes adquiridos en este trayecto de desarrollo de tesina, realmente se puede pensar en realizar proyectos en diferentes sectores y hasta de mayor escala, todo con las posibilidades que provee HTML5 como plataforma.

Un sector con rápida adaptación es el de los códigos QR, códigos de barras bidimensionales muy prolíficos, que mediante una matriz de puntos, permite almacenar gran variedad de tipos de información, y que pueden ser interpretados fácilmente por las cámaras de teléfonos celulares. Esta versatilidad y facilidad de uso convierte a los desarrollos con códigos QR en sumamente tentadores para llevar a cabo, ya sea para proyectos personales como proyectos empresariales o de fidelidad/promoción.

Y siguiendo la línea de juegos móviles, conociendo la posibilidad de desarrollar juegos en HTML5 sin mayor problema, los juegos casuales se encuentran en un gran momento, con numerosos teniendo mucho éxito en aquellas personas con largos viajes en transporte público o con un constante contacto con el dispositivo móvil. Y el hecho de que sea relativamente fácil portabilizarlo a las distintas plataformas, provee una gran ventaja para que llegue a la mayor cantidad de personas posibles.

Estos son simplemente algunos ejemplos de las infinitas posibilidades que proveen estas herramientas en el terreno de las aplicaciones móviles, que sirven como ejemplo sobre los rumbos más prometedores de la plataforma móvil, y hacia donde se mueve la industria.

Para continuar profundizando el estudio, también se podría indagar y estudiar a fondo hasta qué punto puede HTML5 aprovechar todas las capacidades de hardware con las que cuentan los dispositivos móviles (notificaciones led, GPS, barómetro, giroscopio, etc.). Y también con cuanta facilidad se podrían estandarizar estas acciones, de manera que se puedan aprovechar a través de las diferentes plataformas móviles.

GLOSARIO

- ***Multiresolución:*** Se refiere a la capacidad de adaptarse a diferentes resoluciones de pantalla (entre celulares, tablets y notebooks, por ejemplo) sin que se produzcan cambios no deseados en la estructura visual del proyecto.

- ***Isométrico:*** Se refiere a una representación visual de un objeto tridimensional en dos dimensiones, en la que los tres ejes ortogonales principales, al proyectarse, forman ángulos de 120º, y las dimensiones paralelas a dichos ejes se miden en una misma escala.

- ***Sprites:*** Se trata de un tipo de mapa de bits dibujados mediante pixeles visibles, muy utilizado en las primeras consolas de video juegos para ahorrar cálculos en el CPU.

- ***Programación móvil:*** Se refiere al desarrollo de aplicaciones para plataformas sobre las que trabajan diversos dispositivos móviles, variando desde reproductores multimedia hasta celulares y tablets.

- ***Framework:*** Es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos de software concretos, que puede servir de base para la organización y desarrollo de software.

- ***Sistemas operativos móviles:*** Es un sistema operativo que controla un dispositivo móvil. caracterizándose por su simpleza y orientación a la conectividad inalámbrica, los formatos multimedia para móviles y las diferentes maneras de introducir información en ellos.

- ***Gestión de versiones:*** Se refiere a la gestión de los diversos cambios que se realizan sobre los elementos de algún producto o una configuración del mismo.

- ***Portar:*** Se define como la acción de desarrollar un software con la capacidad ejecutarse en diferentes plataformas.

- ***Marketplace:*** Se refiere a cualquier tienda en línea donde se puede adquirir software de uno o diversos desarrolladores.

- ***API (o Application Programming Interface):*** Es el conjunto de funciones y procedimientos o métodos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

- ***Geolocalización:*** Se define como la identificación de la locación geográfica real de un objeto, dispositivo o persona.

- ***Markup:*** Es una forma de codificar un documento que, junto con el texto, incorpora etiquetas o marcas que contienen información adicional acerca de la estructura del texto o su presentación.

- ***Layout:*** Se refiere a la distribución de elementos visuales en un espacio.

- ***On-the-fly:*** Se trata de las actividades que se desarrollan u ocurren dinámicamente en tiempo de ejecución, en contraposición de un actividades con un resultado estadísticamente predefinido

- ***Objeto DOM (o Document Object Model):*** Se refiere a un elemento de una interfaz de programación de aplicaciones (API) que proporciona un conjunto estándar de objetos para representar documentos HTML y XML

- ***Dummy:*** Se define como un elemento definido en un sistema pero carente de funcionamiento, usado principalmente en la realización de pruebas de desempeño.

- ***Embeber:*** Significa insertar (incrustar) código de un lenguaje dentro de otro lenguaje

- ***Cookies:*** Es un pequeño fragmento de información enviada por un sitio web y almacenada en el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del usuario.

- ***Background:*** El término se utiliza para nombrar a todos aquellos procesos o rutinas de ejecución que se realizan en segundo plano.

- ***Web 2.0:*** Se refiere a todos aquellos sitios web que facilitan el compartir información, la interoperabilidad, el diseño centrado en el usuario y la colaboración en la Web.

- ***Doctype:*** Es una cadena corta de caracteres de markup que asocia un texto con un tipo de documento específico

- Interpretado: Se refiere a cuando un código fuente es interpretado y ejecutado a medida que se va necesitando, típicamente, instrucción por instrucción. sin guardar dicha traducción.

- ***Orientado a objetos:*** Se trata de un paradigma de programación que usa los objetos en sus interacciones, para diseñar aplicaciones y programas informáticos.

- ***Prototipos:*** Se refiere a objetos no creados mediante la instanciación de clases, sino mediante la clonación de otros objetos o mediante la escritura de código por parte del programador.

- ***Imperativo:*** Se define como un tipo de programación que trabaja en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea

- ***Tipificación:*** Este término define como un lenguaje de programación clasifica los valores y las expresiones en tipos, cómo se pueden manipular estos tipos y cómo interactúan.

- ***Widget***: Se trata de una pequeña aplicación o programa, cuya función es la de dar fácil acceso a funciones frecuentemente usadas y proveer de información visual.

BIBLIOGRAFÍA

Chuburu, L. (18 de Febrero de 2014). *CSS: sintaxis básica y selectores*. Obtenido de Diseño y desarrollo de sitios web de Laura Chuburu: http://www.laurachuburu.com.ar/apuntes/sintaxis-basica-y-selectores-css.html

Eguiluz, J. (2012). Selectores. En J. Eguiluz, *Introducción a CSS* (págs. 40-75). Vitoria-Gasteiz: Creative Commons.

Finley, K. (20 de Septiembre de 2012). *Adobe Launches Hosted PhoneGap Build Service For Creating Cross-Platform Mobile Apps.* Recuperado el Octubre de 2014, de TechCrunch: http://techcrunch.com/2012/09/24/adobe-launches-hosted-phonegap-build-service-for-creating-cross-platform-mobile-apps/

Gauchat, J. D. (2012). Capitulo 4 - Javascript. En J. D. Gauchat, *El gran libro de HTML5, CSS3 y Javascript* (págs. 84-97). Barcelona: Marcombo Editorial.

Gómez Prieto, L. A. (11 de Marzo de 2013). *Steve Jobs vio el futuro y HTML5 estaba en el*. Recuperado el Diciembre de 2014, de HTML5Fácil: http://html5facil.com/informacion/steve-jobs-vio-el-futuro-y-html5-estaba-en-el/

Gustafson, J. (2013). Canvas API. En J. Gustafson, *HTML5 Web Application Development By Example* (págs. 122-154). Sacramento: Packt Publishing.

Haverbeke, M. (2013). En M. Haverbeke, *Eloquent JavaScript* (págs. 12-26). Berlin: Creative Commons.

Kocak, Y. (13 de Junio de 2013). *A Guide to the New HTML5 Form Input Types*. Obtenido de Six Revisions: http://sixrevisions.com/html5/new-html5-form-input-types/

Lubbers, P. (2011). HTML5 History. En P. Lubbers, F. Salim, & B. Albers, *Pro HTML5 Programming* (págs. 40-43). Lake Tahoe: Apress.

MacDonald, M. (2011). Chapter 3 Meaningful Markup . En M. MacDonald, *HTML5: The Missing Manual* (págs. 81-105). Sebastopol: O'Reilly Media.

Magliola, D. (5 de Diciembre de 2011). *Characters! (and moving around)*. Recuperado el Mayo de 2014, de Daniel Magliola's Blog: http://danielmagliola.com/blog/2011/12/characters-and-moving-around/

Ohloh Search Results. (25 de Octubre de 2013). Obtenido de LangPop: http://langpop.com/

Pavan, B. (12 de Marzo de 2014). *Entendiendo HTML5: guía para principiantes*. Obtenido de Bitelia: http://bitelia.com/2013/05/entendiendo-html5-guia-para-principiantes

Pilgrim, M. (2010). Introduction: Five Things You Should Know About HTML5. En M. Pilgrim, *Dive Into HTML5* (págs. 9-11).

Przybylski, R. (s.f.). *Rendering Animated Models*. Recuperado el 16 de 10 de 2014, de Adobe: http://www.adobe.com/devnet/games/articles/rendering-animated-models.html

QuantumCloud. (16 de Julio de 2013). *PhoneGap - the Swiss Army Knife of Mobile Application Development*. Recuperado el Abril de 2014, de Visual.ly: http://visual.ly/phonegap-swiss-army-knife-mobile-application-development

Rauschmayer, A. (2013). Basic JavaScript for the impatient programmer. En A. Rauschmayer, *Basic JavaScript* (págs. 8-48). Munich: Creative Commons.

Schmitz, S. (20 de Julio de 2012). *Fifty new HTML5 compatible fonts now available for your website*. Recuperado el Octubre de 2014, de Cabanova: http://www.cabanova.com/blog/2012/07/fifty-new-html5-compatible-fonts-for-your-website/

Tijgerd. (12 de Diciembre de 2012). *How to transform mouse location in isometric tiling map?* Recuperado el 15 de 10 de 2014, de StackOverflow: http://stackoverflow.com/questions/6915555/how-to-transform-mouse-location-in-isometric-tiling-map

Wikipedia. (s.f.). *Video Game Graphics*. Recuperado el 18 de 10 de 2014, de Wikipedia: http://en.wikipedia.org/wiki/Video_game_graphics

ANEXO

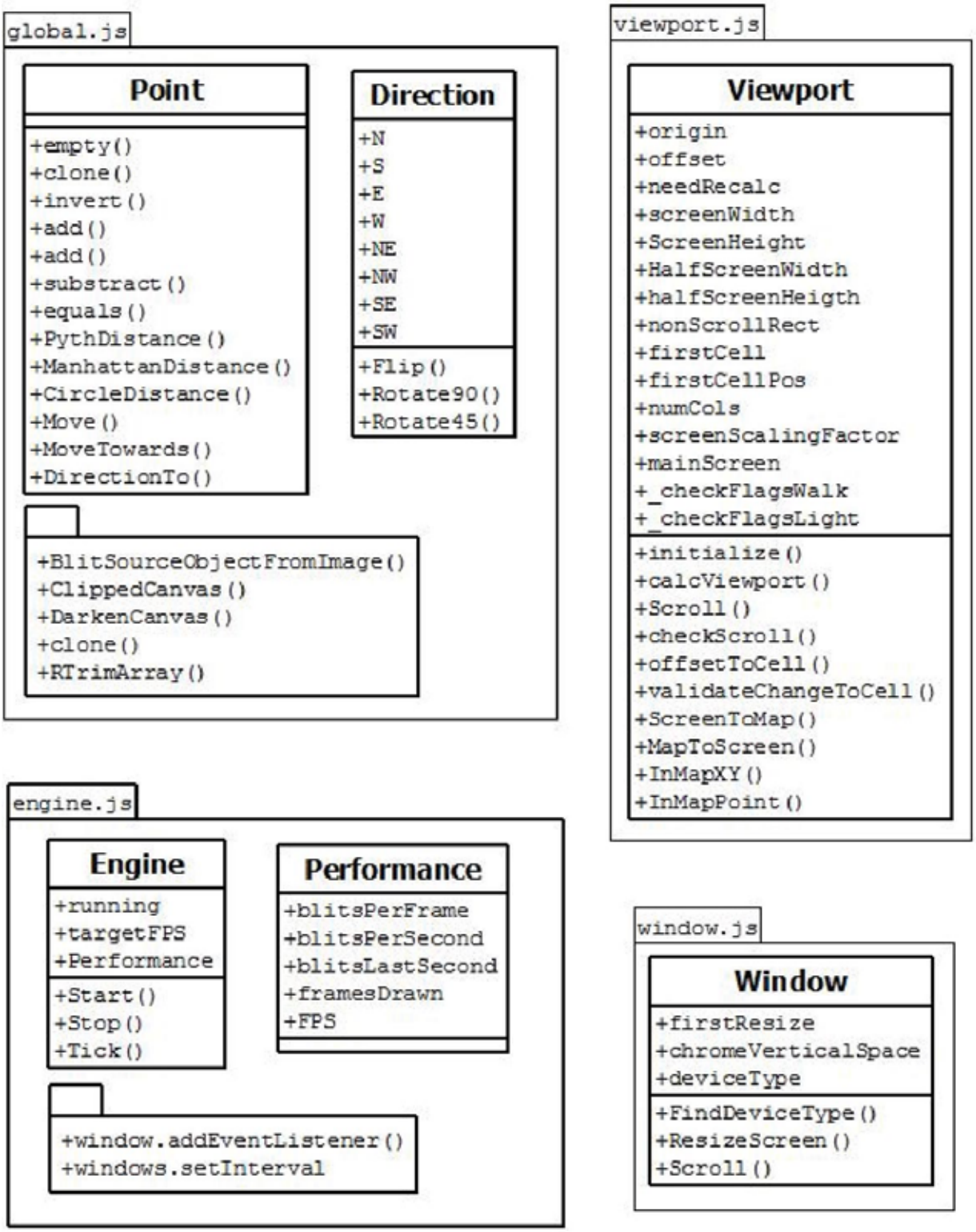


ILUSTRACIÓN 37 - DIAGRAMA DE ARCHIVOS Y CLASES AMPLIADO (PARTE 1)

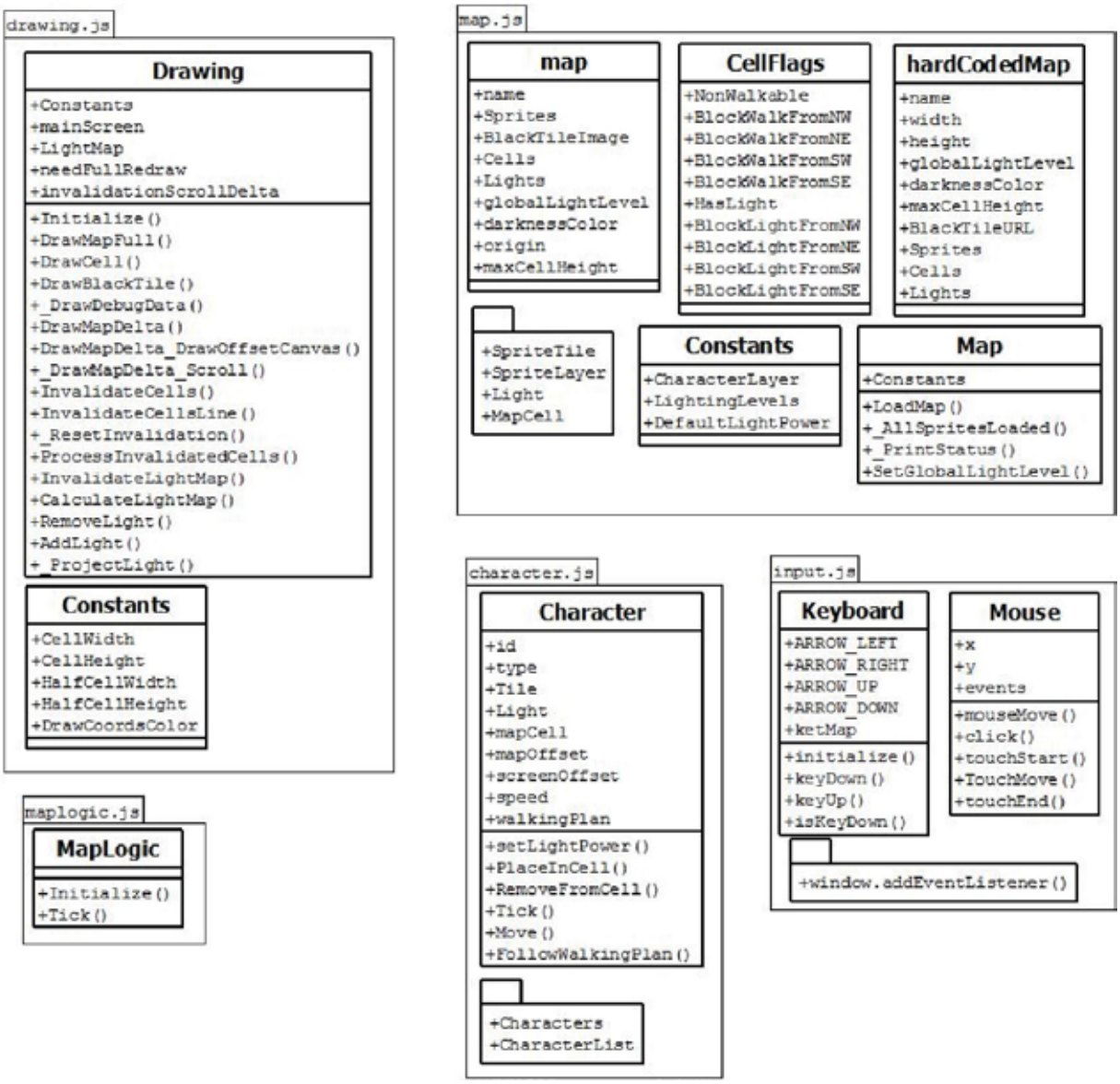


ILUSTRACIÓN 38 - DE ARCHIVOS Y CLASES AMPLIADO (PARTE 2)

ILUSTRACIÓN 39 - EJEMPLO DE ILUMINACIÓN TOTAL CON ÁRBOLES

ILUSTRACIÓN 40 - EJEMPLO CON TEXTURAS DE AGUA Y MUROS BAJOS

ILUSTRACIÓN 41 - EFECTO ANOCHECER, CON FUENTES DE LUZ E INTERACCIÓN CON ELEMENTOS DEL MAPA

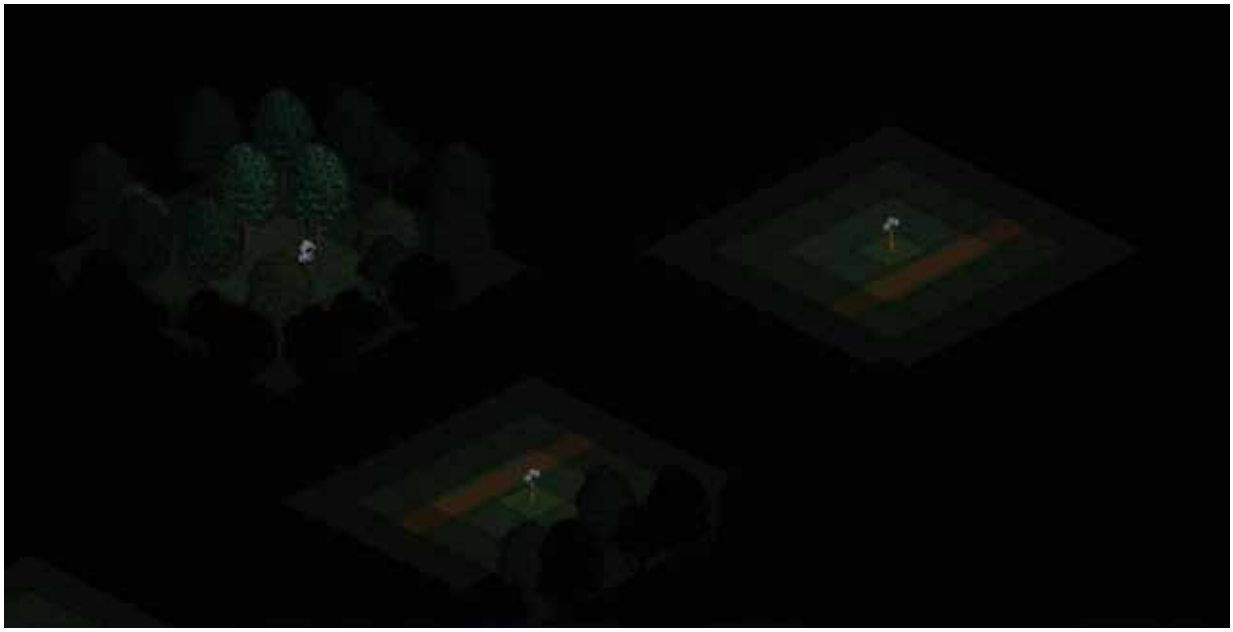


ILUSTRACIÓN 42 - NOCHE CON LUCES ARTIFICIALES Y SU INTERACCIÓN CON LOS ELEMENTOS DEL MAPA

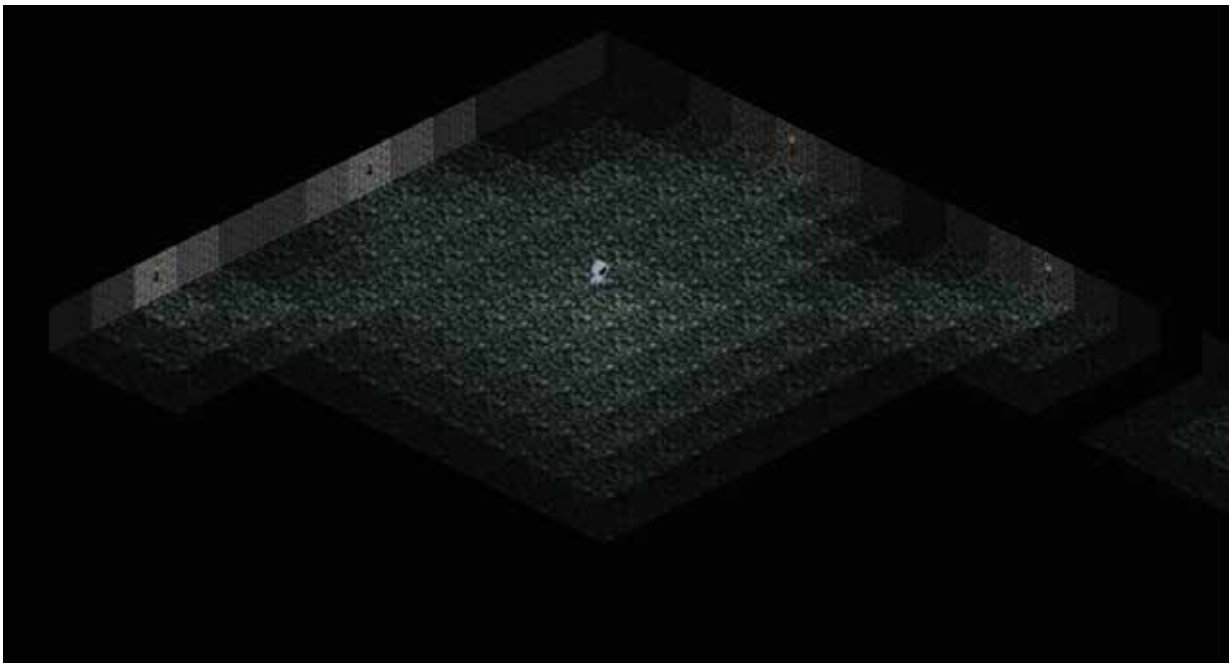


ILUSTRACIÓN 43 - INTERACCIONES DE DIFERENTES FUENTES DE LUZ CERCANAS EN MUROS