BEATER2D: MOTOR GRÁFICO BIDIMENSIONAL UTILIZANDO LAS TECNOLOGÍAS PROVISTAS POR HTML5

WILLIAM FERNANDO GARCIA MUÑOZ
CRISTIAN MAURICIO PORRAS DUARTE

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2011

BEATER2D: MOTOR GRÁFICO BIDIMENSIONAL UTILIZANDO LAS TECNOLOGÍAS PROVISTAS POR HTML5

WILLIAM FERNANDO GARCIA MUÑOZ
(Cód. 2050254)
CRISTIAN MAURICIO PORRAS DUARTE
(Cód. 2050189)

Proyecto de grado presentado para optar al título de
Ingeniero de Sistemas

Director de Proyecto:
Ing. SERGIO HENRY RICO RANGEL

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2011

A los incomprendidos, artífices anónimos de sueños utópicos.

Cristian

A los soñadores e irreverentes creadores de ilusiones.

William

Agradecimientos

Este trabajo no se habría materializado sin el apoyo incondicional de Angely, Elivia y Pablo; gracias a ellos siempre he contado con el ambiente perfecto para el nacimiento y materialización de mis ideas.

Deseo agradecer, además, al ingeniero Sergio Rico por creer en nosotros, ser un amigo y apoyarnos; y a la Comunidad Universitaria de Software Libre, hogar de los compañeros con quienes más he crecido como ingeniero y compartido conocimiento.

Cristian

Al culminar esta etapa de mi vida y encontrarme con la satisfacción de deber cumplido, brindo mis agradecimientos más calurosos y sinceros a mis padres Honorio e Isolina, ya que en su amor e incondicional apoyo, encontré el aliciente para continuar cada día con mi carrera.

A la Comunidad Universitaria de Software Libre, cuna de grandes amigos e inmejorables ideas.

Finalmente un reconocimiento especial al Ingeniero Sergio Henry Rico Rangel por su apoyo en el desarrollo de este proyecto

William

CONTENIDO

pág

LISTA DE FIGURAS

pág

LISTA DE TABLAS

pág

LISTA DE ANEXOS

pág

GLOSARIO

ADOBE FLASH: anteriormente conocido como *Macromedia Flash* es una plataforma multimedia usada para agregar animaciones, videos y distintas capas de interactividad a páginas Web. *Flash* es frecuentemente usado para la presentación de publicidad y videojuegos. Se ha posicionado como una herramienta para el desarrollo de Aplicaciones de Internet Enriquecidas. *Flash* manipula gráficos vectoriales y tipo *raster* para proveer animaciones en texto, dibujos e imágenes. Flash contiene un lenguaje orientado a objetos llamado *ActionScript*.

AJAX: Asynchronous JavaScript and XML, es un término que describe una de las técnicas de desarrollo web usada para implementar RIAs. Plantea el uso de un conjunto de tecnologías existentes juntas, incluyendo HTML o XHTML, CSS, JavaScript, DOM, XML, XSLT, y el objeto XMLHttpRequest; para desarrollar páginas web que mantienen comunicación asíncrona entre el navegador y el servidor. Esto permite que se realicen cambios sobre las páginas web dinámicamente sin necesidad de recargarlas.

API: Interfaz de Programación de Aplicaciones o *API* (del inglés *Application Programming Interface*) es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

CSS: las Hojas de Estilo en Cascada (Cascade Style Sheet), son un lenguaje usado para definir el estilo de presentación y formato de las etiquetas que

conforman un documento HTML o XHTML. Permiten diferenciar la capa que representan la estructura de una página web de la que define su presentación.

DOM: *Document Object Model* (traducido al español Modelo en Objetos para la representación de Documentos) es esencialmente una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos *HTML* y *XML*, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos.

FRAMEWORK: un *Framework* de Software es una abstracción en la cual código común que provee funcionalidades genéricas puede ser selectivamente sobre-escrito o especializado por el código de un desarrollador para proveer funcionalidades especificas. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

JAVAFX: es una plataforma basada en Java para la creación de Aplicaciones de Internet Enriquecidas que puede correr en una amplia variedad de dispositivos conectados a la red. Posibilita la construcción de aplicaciones para navegadores de escritorio y de dispositivos móviles. Para desarrollar aplicaciones en *JavaFX* se utiliza un lenguaje llamado *JavaFX Script*; código en Java puede ser integrado dentro de los programas en *JavaFX*.

MICROSOFT SILVERLIGHT: es un *framework* que provee funcionalidades similares a las de Adobe Flash para aplicaciones Web, integra multimedia,

gráficos, animaciones e interactividad. En las aplicaciones de Silverlight las interfaces de usuario son declaradas en *Extensible Application Markup Language* (*XAML*) y programadas usando un subconjunto del *framework .NET*.

PLUG-IN: un plug-in o complemento es un conjunto de componentes de software diseñado para agregar determinada funcionalidad a una aplicación de mayor tamaño. Los plug-ins son construidos siguiendo las directrices concebidas por los desarrolladores de la pieza mayor de software y son usados para que terceros puedan añadir nuevas funcionalidades al programa original.

RIA: una Aplicación de Internet Enriquecida es una página web que trata de emular las principales características de las aplicaciones de escritorio tradicionales, brindando una mejor usabilidad.

SOFTWARE LIBRE: el termino software libre denomina a los programas desarrollados con una filosofía que enfatiza en los derechos y libertad de los usuarios de un paquete de software. Para que una aplicación sea considerada como software debe estar bajo una licencia que garantice su uso, copia, distribución, y estudio y modificación de su código fuente libremente.

SPRITE: en el ámbito de los gráficos por computador un sprite es una imagen o animación en dos dimensiones que es integrada en una escena mayor. En los videojuegos el uso de sprites facilita la creación de personajes animados que pueden desplazarse por toda la pantalla sin la necesidad de actualizar todo el mapa de bits al cual pertenecen.

W3C: el *World Wide Web Consortium* o *W3C* es la principal organización internacional de estandarización para la *World Wide Web*. El objetivo del *W3C* es guiar la Web hacia su máximo potencial a través del desarrollo de protocolos y pautas que aseguren el crecimiento futuro de la Web. Fue fundado en el *Laboratorio para las Ciencias de la Computación* en el *Instituto de Tecnologia de Massachusetts* (*MIT*) con el apoyo de la *Comisión Europea* y de la *Agencia de Investigación de Proyectos Avanzados de Defensa* (*DARPA*).

WHATWG: el *Web Hypertext Application Technology Working Group* es una comunidad de personas interesadas en la evolución de la Web. Se centra primariamente en el desarrollo de *HTML* y de *APIs* necesarias para el desarrollo de aplicaciones Web. El *WHATWG* fue fundado por individuos relacionados con *Apple*, la *Fundación Mozilla* y *Opera Software* en 2004, luego de una sesión de trabajo del *W3C*. Su principal trabajo es el desarrollo de *HTML5*.

XML: de las siglas en inglés *eXtensible Markup Language* (lenguaje de marcas extensible), es un conjunto de reglas para codificar documentos de una forma entendible por maquinas. Sus especificaciones fueron definidas por el W3C. Permite definir la gramática de otros lenguajes específicos. Entre los lenguajes que usan XML para su definición se encuentra XHTML.

RESUMEN

TÍTULO: BEATER2D, MOTOR GRÁFICO BIDIMENSIONAL UTILIZANDO LAS TECNOLOGÍAS PROVISTAS POR HTML5* ***

AUTORES: GARCIA MUÑOZ, William Fernando**
PORRAS DUARTE, Cristian Mauricio**

PALABRAS CLAVE: Motor Gráfico, Motor Físico, Aplicaciones de Internet Enriquecidas, HTML5, Elemento Canvas.

DESCRIPCIÓN: El rápido crecimiento del acceso a la Internet por parte de usuarios finales ha evidenciado la necesidad de aplicaciones web variadas y complejas. Para suplir esta carencia se han desarrollado diversas tecnologías que facilitan el desarrollo de aplicaciones enriquecidas para la web. Las tecnologías que se han masificado tienen la falencia de no cumplir estándares y ser de naturaleza cerrada.

Las organizaciones reguladores de Internet han iniciado el desarrollo de la próxima versión del lenguaje HTML, llamada HTML5, mediante la cual se estandarizan y añaden métodos, etiquetas, campos e interfaces de aplicación diseñados para el desarrollo de Aplicaciones de Internet Enriquecidas. Destaca el elemento Canvas, el cual permite la generación de gráficos en forma dinámica por medio de programación y scripting dentro de una página web.

En este documento se presenta y discute el análisis, diseño y desarrollo de los módulos y funciones fundamentales para la construcción de un prototipo de motor gráfico bi-dimensional usando HTML5 y el elemento Canvas, unido al motor de física de cuerpo rígido en dos dimensiones Box2DJS. Los módulos que conforman el prototipo de motor gráfico son: módulo central, módulo de gráficos, módulo de sonido, módulo de eventos y módulo físico. Así mismo se documenta la implementación de dos aplicaciones de demostración construidas sobre el motor gráfico Beater2D.

---

* Trabajo de Grado en la Modalidad de Investigación
** Facultad de Ingenierías Físico-Mecánicas, Ingeniería de Sistemas e Informática
*** Director: Ingeniero Sergio Henry Rico Rangel

ABSTRACT

TITLE: BEATER2D, TWO-DIMENSIONAL GRAPHICS ENGINE USING THE TECHNOLOGIES PROVIDED BY HTML5* ***

AUTHORS: GARCIA MUÑOZ, William Fernando**
PORRAS DUARTE, Cristian Mauricio**

Keywords: Graphics engine, Physics engine, Rich Internet Applications, HTML5, Canvas element.

Description: The fast growth of the Internet access by end users has proved the need for varied and complex web applications. To fill this gap there have been developed various technologies that facilitate the development of rich applications for the Web. The massified technologies have the shortcoming of not meeting standards and to be of a closed nature.

The Internet regulatory organizations have begun the development of the next version of HTML language, called HTML 5, in which are added and standardized new methods, tags, fields and application interfaces designed for the development of Rich Internet Applications. Stands out the Canvas element, which allows the dynamically generation of graphics through the programming and scripting within a web page.

This document presents and discusses the analysis, design and development of the essential modules and fundamental functions implemented in the construction of the prototype of a two-dimensional graphics engine using HTML5 and the Canvas element, connected with the two dimensions rigid body physics engine Box2DJS.The modules that make up the graphics engine prototype are: central module, graphics module, sound module, events module and physical module. This work also document the implementation of two demonstration applications built with the graphics engine prototype Beater2D.

---

* Degree work, research mode
** Faculty of Physics-Mechanics Engineering, Systems Engineering and Informatics School
*** Director: Engineer Sergio Henry Rico Rangel

INTRODUCCIÓN

Desde el año 2004 se encuentra en desarrollo la próxima revisión mayor del lenguaje estándar para la Web, *HTML* (*HyperText Markup Language*), denominada como *HTML5*[1] e inmediato sucesor de *HTML 4.01* y *XHTML 1.1*. Uno de los objetivos de esta nueva revisión es reducir la dependencia de complementos y tecnologías propietarias como *Adobe Flash* o *Microsoft Silverlight* para desarrollar Aplicaciones de Internet Enriquecidas o *Rich Internet Applications*, RIA por sus siglas en ingles.

El *Web Hypertext Application Technology Working Group* (*WHATWG*) comenzó el trabajo de la especificación en Junio de 2004[2]. En Marzo de 2010 la especificación llego al estado de *Borrador de Estándar* para la *WHATWG* y al de *Borrador de Trabajo* para la *W3C* (*World Wide Web Consortium*)[3].

Aunque la implementación de las nuevas especificaciones de *HTML* están en progreso y se espera que la versión final no llegue al estado de *Recomendación Candidata de la W3C* hasta 2012 y de *Recomendación de la W3C* hasta 2022, muchas partes de *HTML5* están siendo terminadas e implementadas en los principales Navegadores Web antes de que la especificación completa obtenga el estado de *Recomendación*. En palabras de la *WHATWG* estas partes estables ya pueden ser implementadas en productos software: "Algunas secciones son

[1] HTML: THE MARKUP LANGUAGE. World Wide Web Consortium. Unofficial Editor's Draft 18 December 2010. http://dev.w3.org/html5/markup

[2] WHAT OPEN MAILING LIST ANNOUNCEMENT. WHAT Working Group. http://lists.whatwg.org/htdig.cgi/whatwg-whatwg.org/2004-June/000005.html

[3] WHEN WILL HTML5 BE FINISHED? WHAT Working Group. http://wiki.whatwg.org/wiki/FAQ#When_will_HTML5_be_finished.3F

relativamente estables y hay implementaciones que ya son bastante cercanas a su fin, y esas características pueden ser utilizadas en la actualidad"[4].

Dentro de los nuevos elementos introducidos por *HTML5* se encuentra *Canvas*, el cual permite la presentación dinámica y la renderización de imágenes en dos dimensiones y mapas de bits. Esto a bajo nivel, dado que es un modelo de procedimientos que actualiza un mapa de bits y no tiene una escena de gráficos propia. *Canvas* fue inicialmente introducido por *Apple* para uso con el motor *Webkit* para su navegador *Safari*, posteriormente fue adoptado por los navegadores que utilizan el motor *Gecko* (ej. *Mozilla Firefox*) y por *Opera*; y estandarizado por el grupo *WHATWG*[5]. Alrededor del verano de 2009, el *API* de texto y la manipulación de píxeles fueron implementadas en algunos navegadores web, lo que ha incrementado de manera considerable el interés por el elemento *Canvas* entre los desarrolladores web, utilizándolo para la presentación de animaciones, webs interactivas y juegos de demostración.

El presente trabajo documenta el proceso llevado a cabo en el desarrollo de un prototipo de motor gráfico en dos dimensiones usando el *API* del Elemento *Canvas* y el lenguaje de programación *JavaScript*, diseñado para facilitar la implementación de aplicaciones web que utilicen animaciones y simulaciones de física de cuerpo rígido en dos dimensiones.

En el primer capítulo se presentan las especificaciones del proyecto, como lo son sus objetivos, generales y específicos, la descripción del problema que se abordó y su alcance. Paso seguido en el Marco Teórico se describen las principales

---

[4] WHEN WILL HTML5 BE FINISHED? WHAT Working Group. http://wiki.whatwg.org/wiki/FAQ#When_will_HTML5_be_finished.3F

[5] THE CANVAS ELEMENT. WHAT Working Group. http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html

novedades de *HTML5*, el uso del Elemento *Canvas* y demás tecnologías usadas en el desarrollo del proyecto.

En los capítulos 3 y 4 se describe la metodología usada en el desarrollo del *Beater2D*, los pasos seguidos al aplicarla y los resultados de seguirla. También en el capítulo 4 se describe la estructura del motor gráfico implementado y las funcionalidades de sus módulos.

El capítulo 5 está dedicado a presentar los resultados de las pruebas de integridad, funcionalidad y rendimiento realizadas sobre la implementación del motor gráfico. El sexto capítulo habla sobre el tipo de aplicaciones que se pueden desarrollar con el prototipo, ilustrando con dos videojuegos de demostración construidos sobre las librerías del motor gráfico por los autores del proyecto.

Para facilitar el primer contacto con el motor por parte de potenciales desarrolladores en el capítulo 7 se presentan el manual de instalación de *Beater2D* y la guía de referencia del *API* del motor gráfico.

Finalmente en el penúltimo y último capítulo se presentan, respectivamente, las conclusiones y recomendaciones extraídas del desarrollo del proyecto.

Adicionalmente para quienes deseen profundizar aun más en el proceso de análisis, diseño y codificación del proyecto, en la sección de anexos se presentan los documentos fruto de estás fases de la metodología.

1. ESPECIFICACIONES DEL PROYECTO

1.1. OBJETIVOS

1.1.1. Objetivo general

Analizar, diseñar y desarrollar un prototipo de motor gráfico bidimensional, integrado al motor físico *Box2DJS*, utilizando las tecnologías provistas por el borrador de la *W3C* para el estándar *HTML5*; orientado al desarrollo de aplicaciones enriquecidas para la Web.

1.1.2. Objetivos específicos

- Diseñar un motor gráfico bidimensional, que permita el desarrollo de videojuegos y aplicaciones Web enriquecidas utilizando *HTML*, *JavaScript* y *CSS*.

- Desarrollar un prototipo de motor gráfico bidimensional, el cual proveerá un *API* con los módulos de: Audio y Sonido, proporcionando la posibilidad de decodificar archivos de audio y sirviendo como interfaz con los dispositivos de salida de sonido; Módulo de Gráficos e Imágenes, que permitirá la representación en pantalla y control de diferentes tipos de imágenes y animaciones; Módulo de Temporización y Eventos, para el control del flujo de eventos; Módulo de Física de Cuerpo Rígido, utilizando el motor físico *Box2JS*.

- Elaborar una aplicación de demostración con las funcionalidades de cada modulo del motor gráfico acompañada de su documentación.

1.2. DESCRIPCIÓN DEL PROBLEMA

El desarrollo de Aplicaciones de Internet Enriquecidas, *RIAs*, tradicionalmente se ha apoyado en el uso de tecnologías privativas no estandarizadas, las cuales han sido ampliamente usadas en la industria como estándares *de facto*[6], pero que debilitan la neutralidad e interoperabilidad de la red.

En el caso de las animaciones, simulaciones y videojuegos la tecnología más usada es *Adobe Flash*[7], seguida de *Java* y *Microsoft SilverLight.* Estos plugins han sido creados para suplir las falencias de la actual implementación de *HTML*, *XHTML* y *CSS*.

Adobe Flash, aún siendo tan comúnmente utilizado, presenta una serie de inconvenientes. El principal de ellos es su naturaleza cerrada, lo cual impide la estandarización y portabilidad a cualquier sistema que acceda a la web, la presencia de *Flash* en la web sólo responde a los intereses de *Adobe* y no a una entidad neutral como por ejemplo la *W3C*. *Flash* también tiene problemas de rendimiento, ya que se no encuentra correctamente implementado en algunas plataformas, como conocidos sistemas operativos de dispositivos portátiles, teniendo un consumo elevado de recursos *hardware*. Del lado del usuario las

[6] RIA WAR IS BREWING. eWeek Magazine, Jim Rapoza. http://etech.eweek.com/content/application_development/ria_war_is_brewing.html

[7] RICH INTERNET APPLICATION MARKET SHARE. Stat Owl. http://www.statowl.com/custom_ria_market_penetration.php

tecnologías propietarias también representan malestares, tales como la necesidad de instalar *plug-ins* o extensiones de terceros que no se integran correctamente con los navegadores o sistemas operativos de los usuarios, lo cual se materializa en problemas de usabilidad. Adicionalmente la presencia de *plug-ins* de terceros representa potenciales riesgos de seguridad.

Figura 1. Porcentajes de soporte de plugins para RIAs en los computadores conectados a la red

Fuente: http://www.riastats.com/

Agregado a los problemas ya expuestos, las tecnologías actualmente utilizadas para desarrollar *RIAs* tienen una pobre integración con los estándares web, por ejemplo, no existe una estrategia clara para posicionar páginas web desarrolladas con estas tecnologías con los distintos buscadores web, es decir, implementar *Search Engine Optimization* o *SEO*. La indexación de sitios que realizan los buscadores se hace en base a especificaciones estandarizadas.

1.3. ALCANCE

Durante el desarrollo del proyecto se diseñará e implementará el prototipo inicial de un motor gráfico bidimensional basado en *HTML5*, desarrollando los módulos

necesarios para la ejecución de animaciones y simulaciones físicas en dos dimensiones. Estos módulos son: el núcleo central del motor gráfico, el cual proveerá un *API* para comunicar los otros módulos del motor gráfico con el navegador Web; el modulo de audio y sonido; el modulo de gráficos e imágenes; el modulo de temporización y eventos; y el modulo que integrará el motor físico *Box2DJS*. Junto al prototipo inicial se implementará una aplicación de demostración para analizar el rendimiento del motor.

El alcance del proyecto no incluye el diseño e implementación de interfaces para manejar interacción de los usuarios con teclado, *mouse* o *joystick*. Tampoco incluye el desarrollo de aplicaciones específicas.

1.4. JUSTIFICACIÓN

En el campo de las ciencias de la computación los cambios en las tecnologías y estándares de la industria suceden rápidamente. En algunas ocasiones este rápido cambio de tecnologías conlleva la creación de estándares *de facto*, basados en tecnologías cerradas, usados por gran parte de la industria. Este caso se dio en el campo de las Aplicaciones de Internet Enriquecidas o *Rich Internet Applications* (*RIAs*), las cuales son aplicaciones web que tienen ciertas características de las aplicaciones de escritorio. Las *RIAs* han sido típicamente construidas sobre *Adobe Flash*, *Java* y *Microsoft Silverlight*[8].

La creación de un motor gráfico basado en *HTML5* orientado al desarrollo de aplicaciones enriquecidas para la web como: animaciones, simulaciones y

[8] RICH INTERNET APPLICATION STATISTICS. Real world stats of RIA plugin deployments. http://www.riastats.com/

videojuegos; facilitará la incursión en estas nuevas tecnologías y servirá de soporte para migración progresiva del uso de tecnologías privativas hacia el estándar provisto por los grupos que regulan la *World Wide Web*.

Esfuerzos están siendo orientados hacia la pronta migración a *HTML5* por parte gigantes de Internet, quienes ven en el uso del futuro estándar el siguiente paso para brindarles una mejor experiencia a sus usuarios, ya que *HTML5* se integrará nativamente con sus navegadores y no requerirá el uso de *plug-ins* de terceros.

Dado este panorama, el presente proyecto apunta hacia donde están siendo enfocados los esfuerzos globales para la rápida adopción de *HTML5*. Su impacto podrá ser global y se dará en la medida que nuevos proyectos adopten el motor gráfico para desarrollar aplicaciones web y/o contribuyan ampliando las funcionalidades del mismo. La adopción del motor gráfico a desarrollar podrá significar una disminución de costos en licencias de software y tiempos de desarrollo en las organizaciones.

El uso de herramientas libres tales como *IDE's*, motores físicos y *frameworks*, entre otros, representan un ahorro sustancial en el pago de licencias de uso y brindan distintas facilidades técnicas, convirtiéndose en un factor importante en la viabilidad del presente proyecto. Del éxito en el desarrollo de este proyecto podrán desprenderse otra serie de proyectos, como la implementación de nuevas funcionalidades o su implantación en diversos proyectos que se llevan a cabo en la UIS, por ejemplo el Portal del Profesor, el cual puede valerse de las posibilidades del motor para el diseño de distintos tipos de materiales pedagógicos.

2. MARCO TEÓRICO

2.1. HTML5

HTML5 (*HyperText Markup Language*, versión 5) está actualmente desarrollándose como la próxima revisión del estándar.

HTML es un estándar para la estructuración y presentación de contenidos en la *World Wide Web*, *HTML5* especifica dos variantes para la sintaxis de *HTML*: un «clásico» *HTML* (*text/html*), también conocida como sintaxis *HTML5* y una variante *XHTML* conocida como sintaxis *XHTML5* que deberá ser servida al usuario como *XML* (*XHTML*) (*application/xhtml+xml*)[9]. Esta vez *HTML* y *XHTML* se han desarrollado en forma paralela. Uno de los objetivos de esta nueva revisión es reducir la necesidad de usar complementos y tecnologías propietarias como *Adobe Flash* o *Microsoft Silverlight* para desarrollar Aplicaciones de Internet Enriquecidas, *RIAs*. El desarrollo de este estándar es regulado por el Consorcio *W3C*.

Las principales características del borrador *HTML5* son:

- Define un solo lenguaje llamado *HTML5* el cual puede ser escrito en sintaxis *HTML* y en sintaxis *XML*.
- Define los modelos detallados de procesamiento para fomentar las implementaciones interoperables.

---

[9] HTML5 OVERVIEW. World Wide Web Consortium. http://dev.w3.org/html5/spec/Overview.html#html-vs-xhtml

- Mejora el etiquetado en los documentos.
- Introduce etiquetas y *APIs* para paradigmas emergentes, tales como el de las Aplicaciones Enriquecidas para la web.

2.1.1. Novedades en etiquetas *HTML5* introduce nuevos elementos y atributos que son utilizados en el desarrollo de sitios Web modernos. Algunos de ellos son reemplazos semánticos para bloques genéricos comunes como (<*div*>) y (<*span*>) elementos, por ejemplo <*nav*> (bloque de navegación web) y <*footer*> (bloque de la parte inferior de la página web o para las últimas líneas antes de cerrar la etiqueta *HTML*). Otros elementos proporcionan una funcionalidad nueva a través de una interfaz estandarizada, como los elementos <*audio*> y <*video*>[10].

Tabla 1. Principales novedades en las etiquetas de HTML5

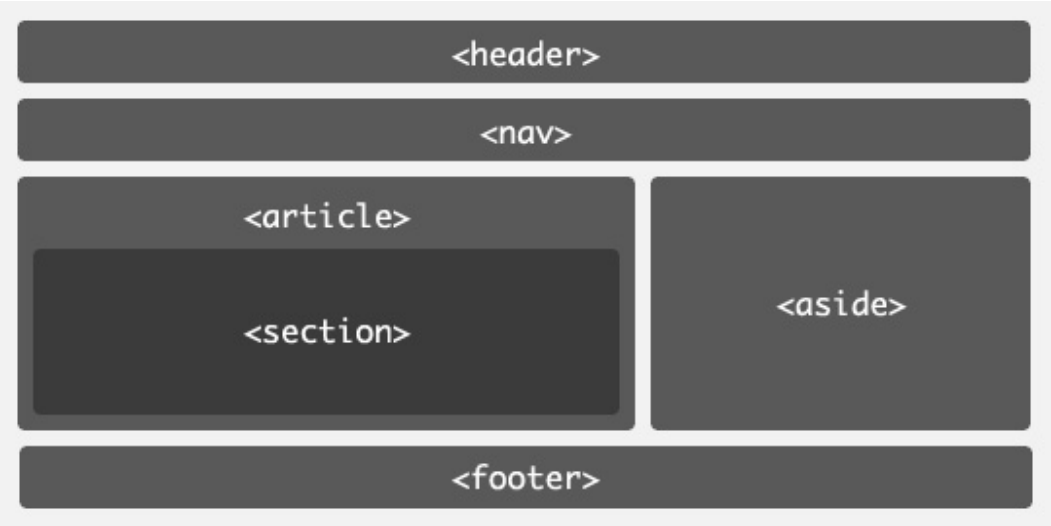
Etiqueta	Descripción
`article`	Permite declarar un trozo del contenido como artículo
`aside`	Representa un trozo de contenido que se relaciona muy levemente con el resto del contenido
`dialog`	Permite representar conversaciones
`figure`	Relaciona contenido incrustado con un texto descriptivo
`footer`	Define una sección de la página para contener información de pie de página
`header`	Representa a la sección de cabecera
`nav`	Representa la sección de la página orientada a la navegación
`section`	Indica en inicio una sección genérica
`audio` y `video`	Permiten añadir contenido multimedia
`embed`	Es un elemento dedicado para contenido de plugins
`m`	Representa el texto marcado
`meter`	Sirve para representar medidas, por ejemplo el tamaño del disco usado

[10] NEW ELEMENTS IN HTML 5. IBM Developer Works. http://www.ibm.com/developerworks/library/x-html5/?ca=dgr-lnxw01NewHTML

`time`	Usado para mostrar fechas y/o tiempo
`canvas`	Permite renderizar gráficos en tiempo real
`commnad`	Relaciona comandos que el usuario puede invocar
`datagrid`	Representa un árbol de datos o una tabla tabulada

Fuente: Autor(es)

Figura 2. Representación semántica de una página web utilizando HTML5



Fuente: http://www.trazos-web.com/2010/02/01/html5-que-es-y-como-usarlo/

También se han definido nuevos tipos de *input* como por ejemplo: *search*, *url*, *email*, *date*, entre otros. La idea de los nuevos tipos de entradas es que los distintos navegadores puedan proveer una interfaz de usuario como calendario para elegir fechas y enviar la información al servidor. Esto brindara una mejor experiencia a los usuarios, ya que sus entradas podrán ser verificadas antes de enviarlas al servidor, lo que para el usuario significa esperar menos tiempo para obtener la retro-alimentación.

Tabla 2. Nuevos campos de entrada para formularios planteados en HTML5

Etiqueta	Uso
`search`	Para cajas de búsqueda
`number`	Para sumar o restar números mediante botones
`range`	Para seleccionar un valor entre dos valores predeterminados

color	Para poder seleccionar un color
tel	Campo para números de teléfonos
url	Campo para direcciones web
email	Direcciones de email
date	Calendario para seleccionar un día
month	Campo para meses
week	Para semanas
time	Para fechas
datetime	Para indicara una fecha exacta

Fuente: Autor(es)

Algunos elementos de HTML 4.01 han sido eliminados, como las etiquetas de presentación <*font*> y <*center*>, cuyos efectos se consiguen mediante el uso de las Hojas de Estilo en Cascada (*CSS*).

También hay un renovado énfasis en la importancia de *scripting DOM* (por ejemplo *JavaScript*) en el comportamiento de la web[11].

La sintaxis de *HTML5* ya no se basa en *SGML* a pesar de la similitud de sus marcas. Aun así *HTML5* ha sido diseñado para ser compatible con versiones anteriores de *HTML*.

2.1.2. Novedades del API *HTML5* especifica interfaces de programación de aplicaciones (*API*) para ayudar a la creación de aplicaciones web. Estas *APIs* pueden ser usadas junto a los nuevos tipos de etiquetas y atributos en las aplicaciones[12].

[11] HTML5 DIFFERENCES FROM HTML4. World Wide Web Consortium. http://dev.w3.org/html5/html4-differences

[12] HTML5 DIFFERENCES FROM HTML4: APIS. World Wide Web Consortium. http://dev.w3.org/html5/html4-differences/#apis

Entre las nuevas *APIs* implementadas en *HTML5* se encuentran:

- El elemento *canvas* que permite renderizar gráficas en *2D*.
- *API* para la reproducción de audio y vídeo.
- Almacenamiento en base de datos *Offline* (Aplicaciones web *Offline*).
- Edición de documentos.
- *API* para arrastrar y soltar elementos.
- Mensajería por Internet.
- Administración del historial de navegación.
- Microdatos.

2.1.3. El elemento Canvas Este elemento hace parte de *HTML5*, permite la presentación dinámica y la renderización de imágenes y mapas de bits en dos dimensiones. Esta representación se realiza a bajo nivel, dado que es un modelo de procedimientos que actualiza un mapa de bits y no tiene una escena de gráficos propia.

Canvas se compone de una región dibujable definida en el código *HTML* con los atributos de altura y anchura. Mediante código *JavaScript* se accede a la zona a través de un conjunto completo de funciones de dibujo similar al de otras *API 2D*, permitiendo así generar gráficos dinámicamente. Algunos de los usos previstos de

canvas incluyen la creación de gráficos, animaciones, juegos, y la composición de imágenes[13].

Figura 3. Ejemplo del uso del elemento Canvas

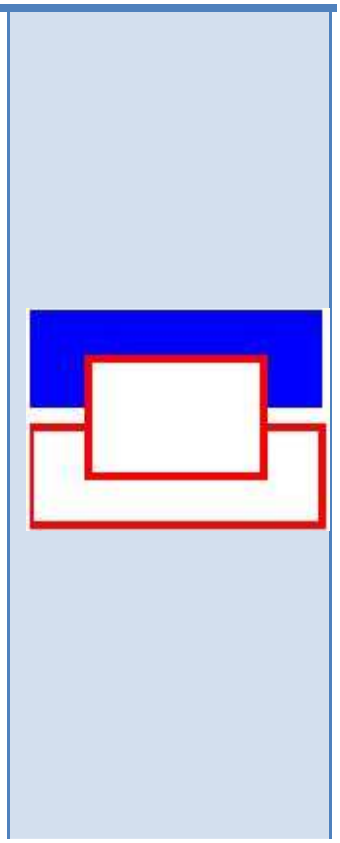
Código

```
<html>
 <head>
  <script type="application/javascript">
    function draw() {
      var canvas =
document.getElementById("canvas");
      if (canvas.getContext) {
        var ctx = canvas.getContext("2d");

        ctx.fillStyle   = '#00f'; // azul
        ctx.strokeStyle = '#f00'; // rojo
        ctx.lineWidth   = 4;

        // Dibuja algunos rectángulos.
        ctx.fillRect  (0,   0, 150, 50);
        ctx.strokeRect(0,  60, 150, 50);
        ctx.clearRect (30, 25,  90, 60);
        ctx.strokeRect(30, 25,  90, 60);
      }
    }
  </script>
 </head>
 <body onload="draw();">
   <canvas id="canvas" width="150" height="150">
   </canvas>
 </body>
</html>
```

Resultado



Fuente: Autor(es)

Los desarrolladores no deben usar el elemento *canvas* en un documento cuando existe algún elemento más adecuado. Por ejemplo, es inapropiado usar *canvas* para renderizar el encabezado de una página: la representación del encabezado debe ser enmarcado usando los elementos apropiados (generalmente *<h1>*) y estilizado usando *CSS*.

[13] CANVAS 2D API SPECIFICATION 1.0. World Wide Web Consortium. http://dev.w3.org/html5/canvas-api/canvas-2d-api.html

2.2. JAVASCRIPT

JavaScript es un lenguaje interpretado utilizado para acceder a objetos en aplicaciones. Se utiliza integrado al navegador web permitiendo el desarrollo de interfaces de usuario y páginas web dinámicas. *JavaScript* es un dialecto de *ECMAScript* y se caracteriza por ser un lenguaje basado en prototipos, con entrada dinámica y con funciones de primera clase[14]. *JavaScript* ha tenido influencia de múltiples lenguajes y se diseñó con una sintaxis similar al lenguaje de programación *Java*.

Todos los navegadores modernos interpretan el código *JavaScript* integrado dentro de las páginas web en HTML. JavaScript puede transformar dinámicamente el contenido de una página web debido a su compatibilidad con el *Document Object Model* o *DOM* (modelo de objetos del documento), lo cual le sirve como interfaz para acceder y modificar el contenido, estructura y presentación de un documento *HTML*.

JavaScript fue inicialmente implementado por el empleado de *Netscape Communications* Brendan Eich bajo el nombre de *Mocha*, para posteriormente ser renombrado como *LiveScript*, y finalmente en 1995 en un acuerdo entre *Sun Microsystems* y *Netscape* fue presentado al público como *JavaScript*[15].

En Noviembre de 1996 *Netscape* anunció el envío de *JavaScript* a *ECMA International* (*European Computer Manufacturers Association International*) para su consideración como estándar de la industria. Como trabajo consecuente resulto la versión estandarizada de *JavaScript*, llamada *ECMAScript*[14].

---

[14] ECMASCRIPT LANGUAGE SPECIFICATION. ECMA International. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf

[15] THE A-Z OF PROGRAMMING LANGUAGES: JAVASCRIPT. Naomi Hamilton, http://www.computerworld.com.au/article/255293/a-z_programming_languages_javascript/

Desde su estandarización e implementación en los navegadores web se ha utilizado en documentos *HTML* para realizar operaciones en la aplicación usada por el cliente para ver documentos escritos en *HTML*. *JavaScript* ha facilitado el desarrollo de los conceptos que han marcado el rumbo de la Web 2.0 como *Ajax* y *RIAs*.

2.3. COMPUTACIÓN GRÁFICA

La computación gráfica se centra en los gráficos creados y manipulados por ordenadores, los avances de esta rama de las ciencias de la computación han facilitado la interacción entre el ordenador y el humano, y ha mejorado la compresión e interpretación de muchos tipos de datos. Los avances en los gráficos por computador han impactado diversos tipos de medios de comunicación y ha revolucionado la animación, las películas y en base a ella se creó la industria de los videojuegos.

Se han creado diversas herramientas que permiten visualizar datos, lo que permite mejorar el análisis de los mismos. Los gráficos generados por ordenador se pueden clasificar en varios tipos como *2D*, *3D*, isométricos y gráficos animados. Gracias al avance de la tecnología, los gráficos en 3 dimensiones por computadora se han vuelto más comunes, aunque los gráficos en *2D* son utilizados ampliamente por su sencillez. Durante las última década, se han desarrollados campos especializados en torno a la computación gráfica, como la visualización de la información y visualización científica más centrados en “la visualización de fenómenos en tres dimensiones (arquitectónicos, meteorológicos,

médicos, biológicos, etc), en las cuales se enfatiza en las representaciones realistas de volúmenes, superficies, fuente de iluminación entre otras"[16].

Actualmente, los gráficos generados por ordenador son utilizados en diversas aplicaciones, como paseos virtuales, interfaces gráficas de usuario, diseño arquitectónico, simulaciones meteorológicas, aplicaciones médicas y juegos de vídeo, entre otras.

2.3.1. Gráficos 2D por computador El término **'**gráficos *2D* por computador', abarca las técnicas para la generación de imágenes digitales en dos dimensiones. Se utilizan principalmente en aplicaciones que fueron desarrolladas originalmente para la impresión y tecnologías tradicionales de dibujo, como la cartografía, dibujo técnico, publicidad, etcétera[17].

Los gráficos en dos dimensiones se utilizan principalmente en aplicaciones que fueron desarrolladas originalmente para la impresión y técnicas tradicionales de dibujo, como la tipografía, la cartografía, dibujo técnico, publicidad, etc. En estas aplicaciones, la imagen de dos dimensiones no es sólo una representación de un objeto del mundo real, sino un artefacto independiente con valor añadido semántica, modelos de dos dimensiones, por lo tanto preferible, porque dan un control más directo de la imagen que los gráficos *3D* (cuyo enfoque es más parecido al de la fotografía que al de la tipografía).

---

[16] MILESTONES IN THE HISTORY OF THEMATIC CARTOGRAPHY, STATISTICAL GRAPHICS, AND DATA VISUALIZATION. Michael Friendly. 2008

[17] WHAT ARE COMPUTER GRAPHICS? University of Leeds ISS. http://iss.leeds.ac.uk/info/306/graphics/215/overview_of_computer_graphics/2

El concepto de los gráficos por computador está íntimamente ligado al término Motor Gráfico. Un motor gráfico es una colección de librerías y métodos diseñado para facilitar la creación de animaciones y video juegos. Las funcionalidades básicas de un motor gráfico incluyen una interfaz de renderizado, un motor físico o de detección de colisiones, gestión de sonido, scripting, animación, inteligencia artificial y todo lo necesario para agilizar el desarrollo de un juego en base a este.

2.4. MOTOR FÍSICO

Motor Físico es un término que hace referencia a las rutinas de un programa que se encargan de simular las interacciones físicas entre elementos de una animación, dando a este mayor realismo. La Animación física por computadora es un tipo de programación, en donde supone la introducción de las leyes de Física en un simulador, particularmente en los gráficos *3D* por computadora, aunque también existe en gráficos *2D*. El propósito es hacer que los efectos físicos de los objetos creados o modelados tengan las mismas características o aproximadas a las que posee en la vida real, teniendo en cuenta, por ejemplo, gravedad, masa, fricción, restitución, etc[18].

Un motor de físico es un programa de computador que proporciona una simulación aproximada de ciertos sistemas físicos simples, tales como dinámicas de cuerpos rígidos (incluyendo detección de colisiones), la dinámica de cuerpos blandos, y la dinámica de fluidos. Su uso se da en los ámbitos de los gráficos por ordenador, los

[18] BOX2D: FEATURES. Box2D Physics Engine. http://box2d.org/features.html

videojuegos y el cine[19]. Su más notorio usos es en los juegos de vídeo (como *middleware*), en cuyo caso la simulación es en tiempo real. El término se utiliza a veces de manera más general para describir cualquier sistema de software para la simulación de fenómenos físicos, como el alto rendimiento de simulación científica.

En general hay dos clases de motores físicos, en tiempo real y de alta precisión. Motores de alta precisión física requieren más poder de procesamiento para el cálculo de la física muy precisa y suelen ser utilizados por los científicos y para la creación de películas animadas. En los juegos de vídeo, u otras formas de computación interactiva, el motor de física simplifica sus cálculos y reduce su precisión para que puedan realizarse en tiempo real, para que el juego tenga la capacidad de responder adecuadamente sin ralentizar el juego. Esto se conoce como la física en tiempo real.

Los motores físicos suelen tener dos componentes básicos, uno de detección de colisiones ó sistema de respuesta a las colisiones[20], y el componente de simulación dinámica[21], encargado de resolver las fuerzas que afectan a los objetos simulados. Los motores modernos también pueden contener simulaciones de fluidos, sistemas de animación y herramientas de control activo de integración.

---

[19] PROJECTS USING THE BULLET ENGINE. Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Bullet_%28software%29#Projects_using_the_engine

[20] BOX2D MANUAL: COLLISION MODULE. Box2D Physics Engine. http://box2d.org/manual.html#_Toc258082970

[21] BOX2D MANUAL: DYNAMICS MODULE. Box2D Physics Engine. http://box2d.org/manual.html#_Toc258082971

3. METODOLOGÍA

En el capítulo presente se describe detalladamente la metodología empleada en el desarrollo del prototipo de motor gráfico *Beater2D*. De su seguimiento dependió el éxito del desarrollo del mismo.

3.1. METODOLOGÍA DE DESARROLLO

La mayoría de las metodologías para el desarrollo de software contemplan las siguientes fases: análisis, diseño, desarrollo e implementación del sistema, y liberación. La metodología que se siguió en el proyecto se diseñó teniendo en cuenta las cuatro fases mencionadas y para cada fase se diseñó una estructura pensada para ajustarse a la naturaleza del proyecto.

El motor gráfico fue concebido para ser desarrollado modularmente, cada uno de los módulos funcionales es independiente, brindando así la posibilidad de planificar el desarrollo en paralelo de cada uno de ellos, para así al finalizar el desarrollo de todos los módulos estos se pudieran integrar como el motor gráfico terminado.

La metodología planteada está fundamentada en el enfoque de desarrollo que brinda la programación extrema (con sus valores de simplicidad, comunicación, retroalimentación)[22].

Se optó por un proceso de desarrollo de software ágil ya que se deseaba tener especial énfasis en la adaptabilidad del motor gráfico conforme avanzaba su desarrollo. Aun así gran parte de los requisitos se definieron al inicio del proyecto teniendo en cuenta la flexibilidad para adaptarse a los pequeños cambios que pudieran surgir en desarrollo.

La metodología usada está direccionada por el Documento de Especificación de Requisitos y cuenta con un proceso iterativo e incremental.

Los Casos de Uso son una técnica que sirve para obtener los requisitos del sistema en términos de las funcionalidades que el usuario espera que estén implementadas en el software. Se define un Caso de Uso como un fragmento de funcionalidad del sistema[23].

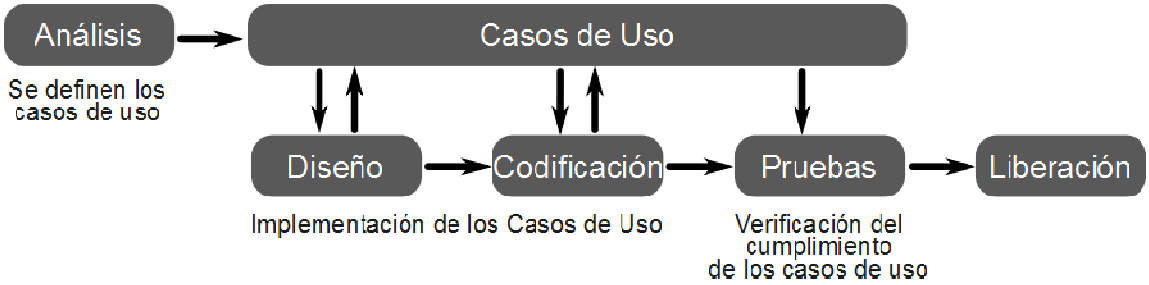
En el caso del presente proyecto los Casos de Uso no sólo fueron utilizados como una herramienta para especificar los requisitos del sistema. También se usaron extensamente como guía en las fases de diseño, implementación y pruebas, pero teniendo la particularidad que en cada fase estos Casos de Uso podían sufrir pequeños cambios dada la retroalimentación que se planteó en la metodología utilizada.

Los Casos de Uso se establecen como un elemento integrador y una guía de trabajo que evoluciona junto al sistema, como se muestra en la figura 4.

[22] UNA EXPLICACIÓN DE LA PROGRAMACIÓN EXTREMA: ACEPTAR EL CAMBIO. Kent Beck . Addison-Wesley Iberoamericana Espanya, S.A. 2002

[23] GETTING FROM USE CASES TO CODE, PART 1: USE-CASE ANALYSIS. Gary Evans. http://www.ibm.com/developerworks/rational/library/5383.html

Figura 4. Los Casos de Uso integran el trabajo y sufren de retroalimentación en las fases de Diseño y Codificación



Fuente: Autor(es)

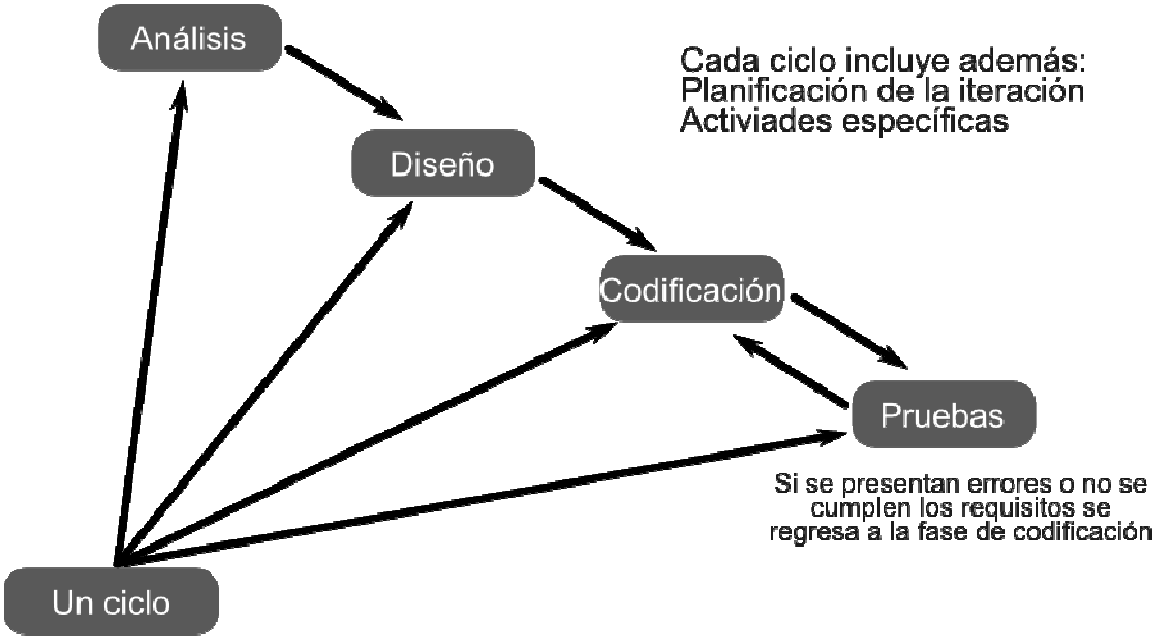
Dado que el motor gráfico se planeó como un sistema modular el trabajo sobre este adoptó un proceso iterativo e incremental, con esta estrategia el desarrollo del motor se pudo dividir en partes más pequeñas o mini-proyectos, que en este caso se refieren a cada uno de los módulos. Esto permitió subdividir los casos de uso en los necesarios por cada módulo, teniendo así un mayor control sobre sus cambios gracias a la división de sus funcionalidades.

Cada mini-proyecto se puede ver como un ciclo en el cual se produce un módulo del motor gráfico. Un ciclo se puede representar por medio de una cascada de acciones como se muestra en la Figura 5.

Adicionalmente la metodología del proyecto adoptó técnicas típicas de la programación extrema como la programación en parejas, que recomienda que el desarrollo se realice por dos personas a la vez. La programación en parejas ayuda a brindar una mayor calidad al código[24] y fue utilizada en partes críticas del sistema como el módulo central. Se formalizó también el proceso de corrección de errores antes de añadir nuevas funcionalidades a los módulos y la refactorización del código cuando fue necesaria.

[24] LA PROGRAMACIÓN EXTREMA EN LA PRÁCTICA. James Newkirk, Robert C. Martin. Addison-Wesley Iberoamericana Espanya, S.A. 2002

Figura 5. Ciclos de cada mini-proyecto o módulo del sistema



Fuente: Autor(es)

3.2. FASES DE LA METODOLOGÍA DE DESARROLLO

La metodología utilizada se puede descomponer en 4 fases, que constituyen el ciclo de vida de desarrollo del prototipo, estas fases son Análisis, Diseño, Codificación y Liberación. Cada una de estas fases se divide en iteraciones, cuyo número es variable según cada fase.

Cada fase tiene un tiempo de inicio y final bien definidos, así como las metas que se tienen que alcanzar en ellas, las cuales se tienen que alcanzar plenamente antes de poder dar por finalizada una fase y proseguir en el desarrollo del prototipo.

En el siguiente capítulo se profundiza sobre los planteamientos y resultados de las distintas fases de la metodología de desarrollo.

4. DESARROLLO DEL PROYECTO

4.1. FASE DE ANÁLISIS

El objetivo de la fase inicial fue recoger los requisitos del sistema y generar con estos las especificaciones que describen qué implementar y cómo se debe componer el motor gráfico.

El análisis se caracterizó por conservar una visión sobre el sistema que se preocupaba por especificar qué hace, de modo que en esta fase toda la atención se centro en los requisitos funcionales de *Beater2D*.

En esta primera etapa del proyecto se analizaron los requisitos básicos y necesidades a suplir del prototipo de motor gráfico. Para hacer esto se estudió el estado del borrador del futuro estándar *HTML5* y su implementación en distintos navegadores web. Este estudio se centró en el *API* de *Canvas*, la nueva interfaz de *HTML* para representar imágenes en pantalla a través de scripts, específicamente de *JavaScript*[25]. Del análisis inicial de la implementación de *HTML5* se obtuvo un cuadro comparativo del estado de esta en varios navegadores modernos, esta información puede consultarse en el Anexo B del presente texto.

Posterior al contacto inicial con *HTML5* se definió el alcance del proyecto (Ver capítulo 1.3) y se realizó un esbozo de los módulos que compondrían el motor gráfico. Sobre esta base se definieron los Casos de Uso. Estos Casos de Uso

[25] THE CANVAS ELEMENT. World Wide Web Consortium. http://www.w3.org/TR/html5/the-canvas-element.html#the-canvas-element

enmarcan las funcionalidades que ofrece el prototipo de motor gráfico y sus capacidades de interacción externa, es decir el conjunto de funciones y clases con las cuales pueden contar los desarrolladores que decidan usar *Beater2D* como librería para sus proyectos. En el anexo C (Documento de Especificación de Requisitos de Software) se encuentran los Cuadros de Casos de Uso, clarificando los requerimientos sobre los cuales se desarrolló el sistema.

Una vez definidos los procedimientos que se debían implementar en el motor gráfico se inició el estudio de una serie de tecnologías que potencialmente serían utilizadas en el desarrollo del sistema para facilitar dicho proceso. Principalmente se debatió sobre qué *framework* de *JavaScript* se construiría el motor, llegando a la conclusión de que el más idóneo para el proyecto era *jQuery*[26], dado su estructura modular que permite ampliar las funcionalidades del prototipo de motor gráfico con los módulos compatibles con *jQuery*. Esta subfase también produjo un cuadro comparativo entre los *frameworks* de *JavaScript* que puede ser consultado en el Anexo A.

También en la fase de análisis se agrupó, ordenó y analizó el código fuente y la documentación del motor de física *Box2DJS* y se estudiaron las diversas maneras en las que se podría acoplar este desarrollo del motor gráfico.

4.2. FASE DE DISEÑO

En la fase de diseño se llevó a cabo una profundización en la información recopilada en la fase de análisis, teniendo en cuenta los requisitos no funcionales

[26] JQUERY: THE WRITE LESS, DO MORE, JAVASCRIPT LIBRARY. http://jquery.com/

del sistema, y se planteó la manera más idónea para que el sistema cumpliera con estos requisitos.

El objetivo principal de la segunda fase fue definir la arquitectura sobre la cual se construyó el sistema. Se identificaron claramente las funcionalidades de cada módulo y se definieron los métodos que componen cada uno de estos.

En la fase de diseño se llevó a cabo el proceso de modelado del software, teniendo en cuenta el Documento de Especificación de Requisitos del sistema, describiendo todos los aspectos que el motor gráfico debe cumplir. Una vez finalizada esta etapa se tuvo plasmada una imagen completa de lo que hace el prototipo de motor gráfico en todos sus dominios.

Los criterios que rigieron el diseño de *Beater2D* fueron los siguientes:

- El diseño tiene que describir una estructura jerárquica, haciendo uso inteligente de las relaciones de control entre los módulos del programa.
- La arquitectura debe ser modular y contar con una separación lógica del código en clases diseñadas para cumplir funciones dadas.
- Los módulos funcionales deben estar diseñados para poder ser usados independientemente.

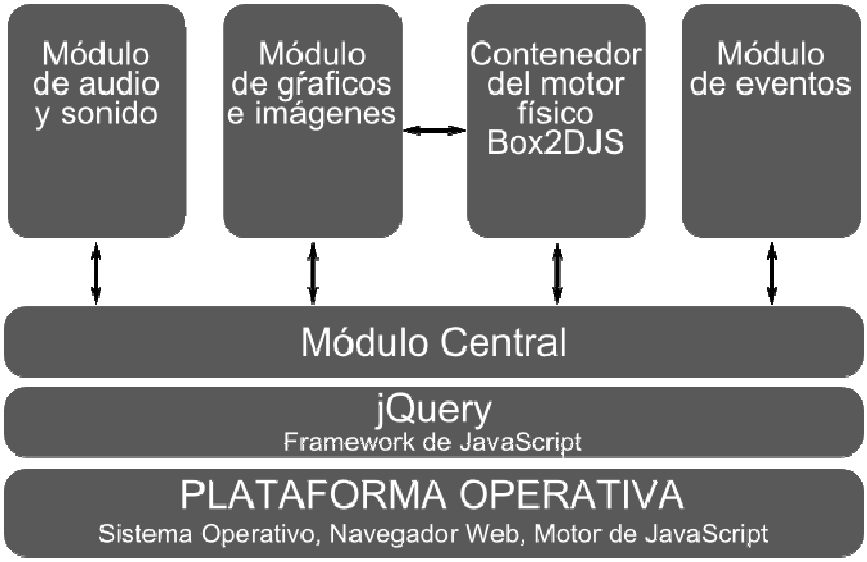
En esta fase del proyecto se diseñó el modelo funcional del motor gráfico, se definieron las funciones de cada módulo y las relaciones entre ellos. Se establecieron las normas de sintaxis y funcionalidades que brinda del motor, las cuales se resumen en las siguientes dos secciones del presente capítulo.

Conjunto al modelo funcional de *Beater2D* se diseñó la metodología para realizar las pruebas del motor gráfico, como lo son las pruebas de unidad para cada uno

de sus módulos y las pruebas sobre el sistema terminado: de rendimiento, verificación y validación del funcionamiento del sistema.

4.2.1. Arquitectura del motor gráfico El motor gráfico está construido pensando en una plataforma operativa, como se muestra en la figura 6, conformada por el navegador web, sus motores de renderizado y *JavaScript*, y la librería *jQuery*. Sobre dicha plataforma operativa se encuentran interactuando las dos capas del motor gráfico, la del módulo central y la de los módulos operativos.

Figura 6. Arquitectura de Beater2D



Fuente: Autor(es)

4.2.2. Descripción de los módulos de Beater2D El prototipo de motor gráfico bidimensional desarrollado se encuentra dividido en módulos, dado que dicha estructura facilita su entendimiento, uso, mantenimiento, escalabilidad, pruebas, documentación y desarrollo.

Esta sección pretende dar una visión global de cada uno de los módulos de *Beater2D* y sus funcionalidades, de la siguiente manera:

Descripción: brinda una descripción general del módulo en cuestión.

Funciones asociados: enuncia las funciones que ofrece cada módulo y cómo las puede usar el desarrollador en sus propias aplicaciones.

4.2.2.1. Módulo Central

Descripción: El Módulo Central es el encargado de gestionar el flujo de un programa desarrollado con el motor gráfico *Beater2D*. Este también es el pilar del sistema, ya que sobre su prototípado (la programación en *JavaScript* está basada en prototipos y no en clases) se construyen los otros módulos y se enlazan los unos a los otros. El módulo central es el primero que se debe instanciar en el desarrollo de una aplicación basada en *Beater2D*.

Funciones asociadas: La principal función del Módulo Central es el manejo del *loop* o bucle principal de un programa desarrollado con el motor gráfico. Es decir que a partir de las órdenes que da este módulo se realiza la actualización de los diferentes elementos que forman parte de un programa basado en la librería.

El mencionado *loop* o bucle principal se rige en base al temporizador que también gestiona el actual módulo y está ligado directamente al control de los Cuadros Por Segundo o *FPS* (*Frames Per Second*, según sus siglas en inglés).

También en este módulo se maneja el sistema de renderización y representación gráfica del motor. Se utiliza el método de doble buffer, esta técnica separa la capa de trabajo de la de representación, lo cual permite tener una salida gráfica más fluida y su impacto en el rendimiento es casi nulo.

El Módulo Central igualmente gestiona el flujo del programa, por medio de este es posible iniciar, pausar, reanudar y detener el programa. Dentro de estas cabe resaltar la gestión de la carga y descarga de los diferentes recursos como imágenes o sonidos necesarios por un programa.

4.2.2.2. Módulo de Audio y Sonido

Descripción: El Módulo de Audio y Sonido es el encargado de crear una interfaz con los dispositivos de audio de un equipo de cómputo para la reproducción de música y sonidos. Dicha interfaz se compone de distintos canales de audio, lo que permite la reproducción de varios sonidos en un mismo instante de tiempo. Para cada canal de audio se encuentran disponibles las mismas funciones y métodos.

Funciones asociadas: Inicialmente es responsable de cargar los sonidos a utilizar con el motor y tiene soporte para sonidos codificados con el códec *Ogg Vorbis* y secuencias *MIDI* (Interfaz Digital de Instrumentos Musicales).

Al cargar un archivo de sonido se crea un canal u objeto de audio, el cual puede reproducirse, pausarse, reanudarse y detenerse. Adicionalmente a estas órdenes básicas cada canal de audio tiene distintos parámetros como lo son su velocidad de reproducción, volumen, posición actual y veces de repetición.

4.2.2.3. Módulo de Gráficos e Imágenes

Descripción: Este módulo ayuda a dibujar nuevas imágenes en pantalla cuando están basadas en las primitivas que provee el *API* del elemento *Canvas* o de cargar las imágenes desde una *URL* en el caso de que sean provistas externamente. Las imágenes importadas pueden ser de tipo *PNG* o *JPEG*.

Sobre las imágenes definidas por este módulo puede realizarse diversas acciones de movimiento, escala, sobre-posición de una imagen sobre otra y varios tipos de animaciones.

Funciones asociadas: El Módulo de Gráficos e Imágenes cuenta con una superclase sobre la cual se definen parámetros y acciones comunes para los diferentes tipos de imágenes. Esta superclase brinda la posibilidad de cargar o definir una imagen y asignarle una serie de atributos para que rijan su comportamiento al añadirla al mundo gráfico. Dentro de las propiedades comunes a todos los tipos de imágenes se encuentran: posición, tamaño, ángulo, escala y profundidad, así mismo como los métodos para obtener y modificar estas propiedades.

Beater2D tiene una capa de abstracción sobre los métodos nativos del *API* de *Canvas* para la representación de algunas figuras básicas o primitivas. Dicha capa de abstracción hereda de la superclase del Módulo de Gráficos e Imágenes obteniendo así sus características básicas y agregando el facilidades al manejo y representación de cuadriláteros, círculos y arcos, polígonos y curvas de Bézier. Adicional a los métodos básicos de la superclase para estos tipos de imágenes es posible definir su estilo, color de línea y de relleno.

Los gráficos externos con los que trabaja el módulo gráfico pueden ser de dos tipos: imágenes estáticas y *sprites*.

Para el manejo de imágenes estáticas se añade a la superclase del modulo la capacidad de cargar y gestionar su representación, la cual puede ser en una región o regiones que pueden cambiar con el tiempo.

Los *sprites* (conjuntos de mapas de bits que representan los diferentes cuadros de una animación) son manejados como una sola imagen que contiene todos los cuadros que puede necesitar la animación de un objeto. Sobre estos es posible definir el número de cuadros que conforman la animación, la velocidad con la que estos cambian y su clase cuenta con diferentes métodos para obtener su estado y modificarlo.

4.2.2.4. Módulo de Eventos

Descripción: El Módulo de Eventos es el encargado de manejar la interacción de un usuario con una instancia del motor gráfico, como por ejemplo el accionar de una tecla, y crear una pila para el tratamiento ordenado de dichas acciones.

A su vez, provee una interfaz para mapear el teclado y gestionar las acciones del *mouse*.

Funciones asociadas: Se divide en tres submódulos, el primero para el manejo del teclado, el segundo para recibir las acciones del *mouse* y el último para la pila de acciones.

El submódulo del teclado se encarga de vigilar las acciones que realiza un usuario con el teclado y, si tienen asignadas acciones en el programa desarrollado con *Beater2D*, estas pasan a la pila de eventos. Las posibles acciones que se pueden llevar a cabo con el teclado y gestiona el motor gráfico son: presionar una tecla, soltar la tecla y dejar presionada una tecla por un tiempo determinado.

La detección de acciones del *mouse* es más compleja y su submódulo permite obtener la posición en píxeles del cursor del mouse sobre una instancia del motor, y responder por acciones como: clic, clic sostenido, arrastre y liberación del mouse.

Una vez detectada una acción del usuario esta pasa a la pila de eventos, la cual es gestionada en el módulo central según las reglas que el desarrollador de una aplicación basada en el motor *Beater2D* decida.

4.2.2.5. Contenedor del Motor Físico Box2DSJ

Descripción: Para el desarrollo de aplicaciones y juegos más realistas *Beater2D* está integrado al motor físico *Box2DJS*, el cual permite realizar simulaciones de física de cuerpo rígido en dos dimensiones, pudiendo así reproducir cuerpos compuestos por polígonos y círculos afectados por fuerzas como gravedad, fricción y restitución.

La capa de abstracción sobre *Box2DJS* permite combinar el manejo del mundo físico con el mundo gráfico de *Beater2D* y así el resultado de las simulaciones físicas son representadas mediante el Módulo de Gráficos e Imágenes.

Funciones Asociadas: Para que un objeto del mundo gráfico sea manejado por el contenedor del motor físico tiene que ser declarado como parte del mundo físico, del cual por defecto ningún objeto hace parte. Una vez definidas sus propiedades físicas el objeto comienza a ser tomado en cuenta para la simulación.

Cada objeto del mundo físico puede ser afectado por colisiones, cuyo resultado dependerá de la forma del objeto y sus diferentes propiedades. Sobre el submódulo de colisiones se encuentra construido el de dinámica que evalúa cómo

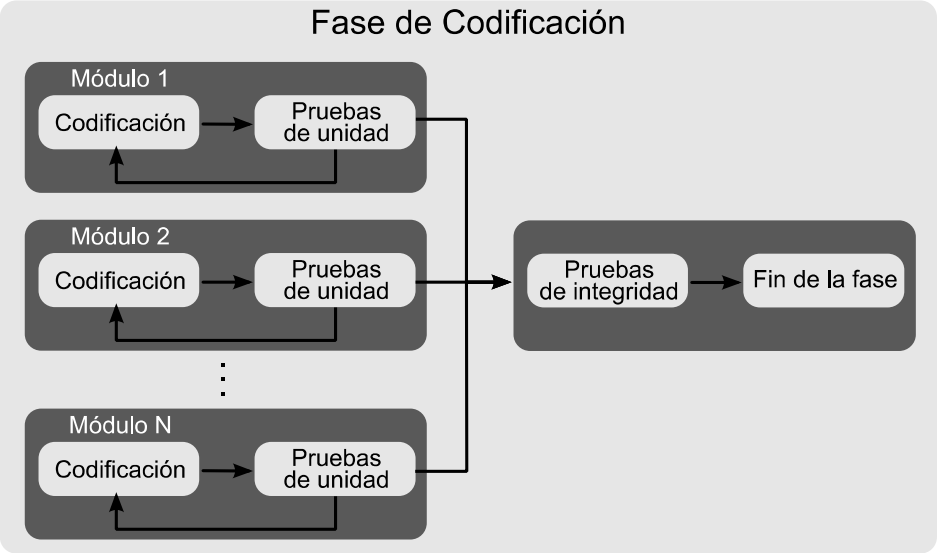
reaccionan los objetos según su forma y maneja el contacto, unión y choque entre ellos.

4.3. FASE DE CODIFICACIÓN

En esta fase se implementaron las funciones y métodos definidos anteriormente en las labores de diseño y se empaquetaron de manera usable. Adicionalmente junto al desarrollo de cada clase se realizaron las pruebas de unidad en la que cada desarrollador era responsable de probar las unidades de código que iba implementando. El resultado final de esta fase, luego de integrar los diferentes módulos con sus respectivas pruebas, es el prototipo de motor gráfico correctamente empaquetado y listo para ser usado.

El desarrollo se llevó a cabo en ciclos paralelos, cada módulo compone un ciclo de desarrollo y pruebas. El desarrollo de cada módulo se realizó siguiendo un proceso iterativo en el que la codificación estaba ligada a una serie de pruebas unitarias, como una forma de probar el correcto funcionamiento de cada modulo del motor gráfico. Cada módulo fue desarrollado por separado, tendiendo ciclos paralelos distribuidos entre el grupo de trabajo, exceptuando el módulo central que fue desarrollado conjuntamente por el equipo de trabajo. Cada módulo contó con una integración incremental, cada clase que se implementaba se añadía en momentos separados, esto facilitó la localización de errores.

Figura 7. Esquema de la fase de codificación



Fuente: Autor(es)

Beater2D es un producto independiente y auto-contenido, es decir, no hace parte de ningún sistema mayor ni necesita invocar funcionalidades de sistemas terceros. Se ha utilizado la librería *jQuery*, complementaria en la codificación ya que contribuye a un desarrollo más rápido del sistema.

En la fase de codificación se realizaron seis ciclos cada uno con un flujo de trabajo similar y aplicando la misma metodología y subfases pero orientadas a distintas partes funcionales del sistema. Para cada ciclo se asignó un módulo de la manera siguiente:

- Ciclo 1: Módulo de Gráficos e Imágenes.
- Ciclo 2: Módulo de Audio y Sonido.
- Ciclo 3: Módulo de Eventos.
- Ciclo 4: Módulo contenedor de las librerías del motor físico *Box2DJS*.

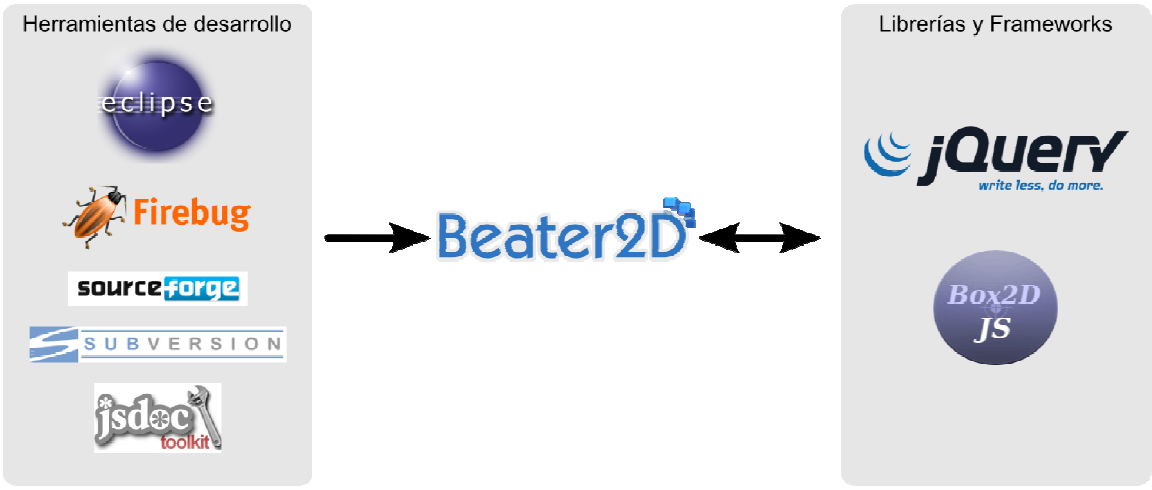
- Ciclo 5: Módulo Central.
- Ciclo 6: Integración del motor gráfico como tal.

Debido a que el desarrollo de *Beater2D* se realizó conjuntamente entre los autores del proyecto, en ciertas fases el desarrollo se distribuía entre los dos autores, dividiendo la elaboración de tareas específicas. Esto hizo necesario el uso de una herramienta que solucionara el problema de la centralización de las versiones del código, ya que era de importancia para el avance el proyecto que cualquiera de los desarrolladores tuviera siempre los últimos avances referentes a la programación, se ha utilizado un repositorio de acceso común donde se lleva el control de las versiones del código. La herramienta utilizada fue *Subversion*, en la sección correspondiente a las tecnologías utilizadas como apoyo al desarrollo del proyecto se profundiza sobre de ella (Ver numeral 4.3.1.6).

4.3.1. Recursos tecnológicos utilizados para la elaboración de Beater2D

Como es habitual en los proyectos de desarrollo de software, en la construcción de *Beater2D* se han utilizado una serie de tecnologías disponibles en el mercado, tecnologías que son usadas no sólo para el funcionamiento del software como tal, sino también las diferentes herramientas de desarrollo como *IDEs*, sistemas manejadores de versiones, gestores de documentación, etc.

Figura 8. Tecnologías y herramientas usadas en la construcción de Beater2D



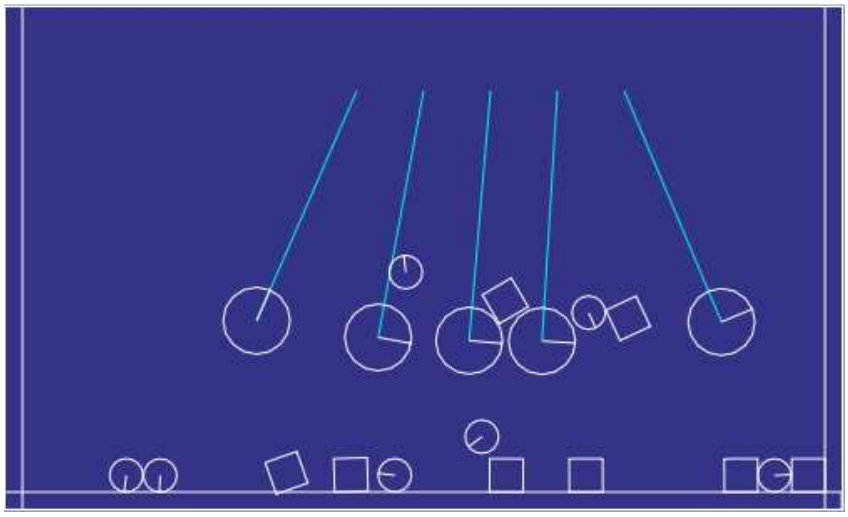
Fuente: Autor(es)

En esta sección se explica de forma breve las razones que llevaron a la selección de cada una de estas tecnologías y su licencia de uso.

4.3.1.1. Box2DJS *Box2DJS* es una adaptación en *JavaScript* del motor físico *Box2D* desarrollado originalmente *C++*[27]. El motor *Box2D* fue diseñado como una colección de funciones para la simulación *2D* de cuerpos rígidos. Los programadores pueden utilizar esta librería para hacer interacciones físicas de sus objetos de manera acertada y eficiente.

[27] BOX2D PHYSICS ENGINE. http://www.box2d.org/

Figura 9. Captura de una simulación física realizada con Box2DJS



Fuente: http://box2d-js.sourceforge.net/index2.html

Trabaja con números de punto flotante, por lo que algunos puntos de tolerancia deben ser usados para que *Box2DJS* tenga un buen desempeño. Esta tolerancia se ha diseñado para trabajar perfectamente con el sistema de unidades MKS (metros-kilogramo-segundo). En particular, *Box2DJS* ha sido adaptado para trabajar bien con objetos móviles de entre 0,1 y 10 metros. Los objetos estáticos pueden ser de hasta 50 metros sin que esto represente problemas para la simulación.

Utiliza un algoritmo de cómputo llamado integrador. Los integradores simulan las ecuaciones físicas en puntos discretos de tiempo. Esto va junto con el ciclo de animación tradicional, donde se tienen esencialmente un *flipbook* animado en la pantalla. Siendo así, es necesario elegir un intervalo de tiempo para *Box2DJS*. Generalmente los motores de física definen como intervalo de tiempo por lo menos de 60 Hz o 1/60 segundo.

Box2DJS está bajo una licencia *Zlib/Libpng*, la cual es una licencia de software libre que permite su utilización como librería sin la necesidad de que los paquetes

de software que utilicen software bajo esta licencia tengan que liberarse bajo la misma.

Dada la necesidad de una librería que permitiera realizar simulaciones de física de cuerpo rígido en dos dimensiones de una manera eficiente *Box2DJS* fue seleccionado por la naturaleza de su licencia y porque es una ramificación del motor *Box2D*, escrito originalmente en *C++*, como ya se dijo, el cual se ha consolidado como la librería física más utilizada en proyectos de software libre.

4.3.1.2. jQuery *jQuery* es una librería de *JavaScript* liviana e ínter-operable entre navegadores web diseñada para simplificar el *scripting* de *HTML* del lado del cliente[28]. Su lanzamiento fue en Enero de 2006. “Es usada por cerca del 27% de los 10.000 sitios web más visitados y es la librería de *JavaScript* más popular de las usadas hoy en día” [29].

Su sintaxis está diseñada para que sea fácil navegar por un documento, seleccionar elementos *DOM*, crear animaciones, manejar eventos y desarrollar aplicaciones en *Ajax*.

Provee a los desarrolladores la capacidad de crear *plug-ins* o extensiones sobre la librería, lo cual permite basarse el *API* de su núcleo para definir a partir de esta las clases que sean necesarias en cualquier proyecto de software que utilice *JavaScript*, ahorrando a los desarrolladores la necesidad de definir la estructura de prototípado a utilizar.

---

[28] JQUERY 1.3 AND THE JQUERY FOUNDATION. The jQuery Project. http://blog.jquery.com/2009/01/14/jquery-13-and-the-jquery-foundation/

[29] JQUERY USAGE STATISTICS. Builtwith. http://trends.builtwith.com/javascript/JQuery

jQuery es software libre y se distribuye bajo dos licencias: *MIT* y *GNU GPLv2*, gracias al doble licenciamiento puede usarse en proyectos libres como privativos, ya que la licencia *MIT* no obliga a que los trabajos que utilicen *jQuery* sean publicados como software libre, contrario al caso de la licencia *GLPv2*.

Dado que *jQuery* provee una base solida, multi-navegador y ampliamente probada; fue seleccionada como la librería que provee la estructura básica sobre la cual se construyo el motor gráfico. Esta decisión también fue tomada debido a la posibilidad de usar e integrar al motor gráfico cualquiera de los *plug-ins* disponibles para *jQuery*.

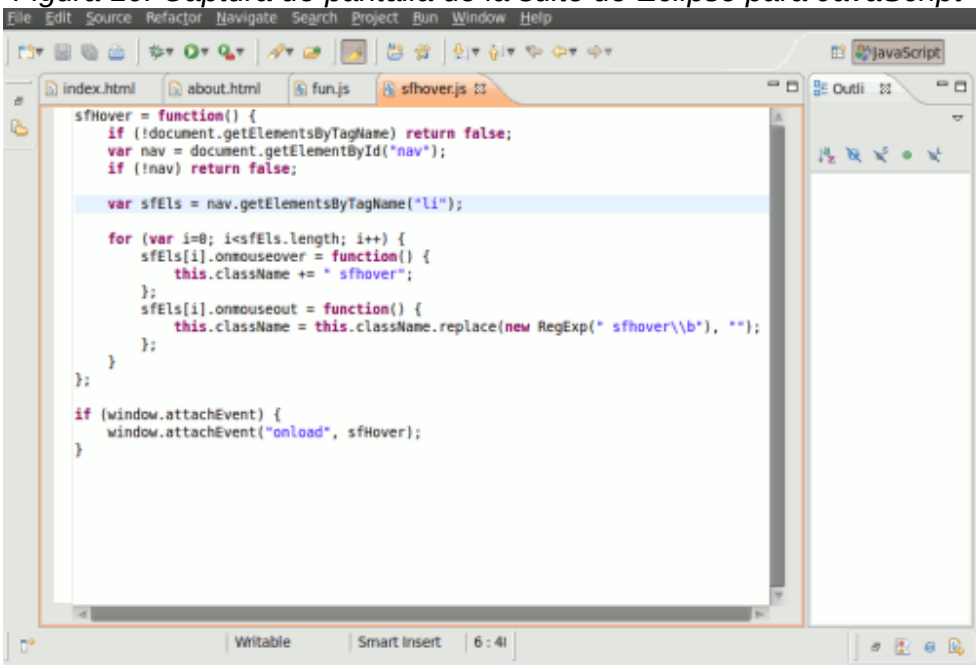
4.3.1.3. Eclipse *Eclipse* es un ambiente de desarrollo de software multi-lenguaje que comprende un Entorno de Desarrollo Integrado (IDE por sus siglas en ingles) y un sistema de *plug-ins* para ampliar sus funcionalidades.

Eclipse fue diseñado principalmente para el desarrollo de aplicaciones en *Java* (también está desarrollado en este lenguaje y es usado para su propio desarrollo), y por medio del sistema de *plug-ins* puede extender su soporte para desarrollar aplicaciones en todo tipo de lenguajes, como por ejemplo *C++*, *PHP* y *JavaScript*.

Dentro de los diferentes paquetes de *Eclipse* oficiales se encuentra la IDE "*Eclipse* para desarrolladores web con *JavaScript*"[30], el cual incluye por defecto herramientas para desarrolladores de dicho lenguaje y también soporte para *HTML*, *CSS* y *XML*. Así mismo mediante el sistema de *plug-ins* es posible obtener soporte para distintas librerías de *JavaScript* (entre ellas *jQuery*).

[30] ECLIPSE IDE FOR JAVASCRIPT WEB DEVELOPERS. The Eclipse Foundation. http://www.eclipse.org/downloads/packages/eclipse-ide-javascript-web-developers/heliossr1

Figura 10. Captura de pantalla de la suite de Eclipse para JavaScript



Fuente: Autor(es)

Eclipse es un programa multi-plataforma, se encuentra disponible para *Windows*, *Mac OSX* y distintos S.O. tipo *Unix* como *GNU/Linux* o *FreeBSD*. Se puede obtener bajo la *Eclipse* Pkublic *License*, la cual es una licencia de software libre avalada por la *Free Software Foundation*.

Se opto por el uso de esta herramienta por la mejora en la productividad que representa para los desarrolladores al simplificar el ciclo de vida de desarrollo debido a su integración con las diferentes tecnologías necesarias para el desarrollo del proyecto.

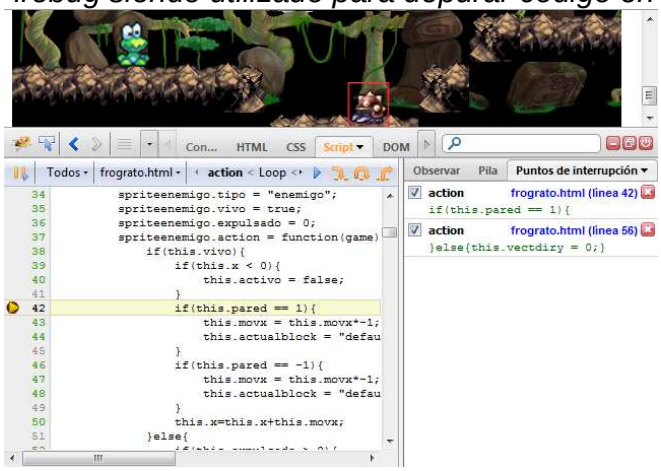
4.3.1.4. Firebug *Firebug* es una extensión del navegador web *Mozilla Firefox* que permite la edición, monitorización y búsqueda de errores en el *CSS*, *HTML*, *DOM* y *JavaScript* de cualquier página web[31].

Su depurador de *JavaScript* permite pausar la ejecución de un programa en cualquier momento e inspeccionar el estado de todas las variables del entorno. Los puntos de rotura pueden ser definidos en una línea cualquiera, activarse sólo si ciertas condiciones se cumplen o activarse automáticamente al presentarse un error en el código.

También permite mejorar el rendimiento del código por medio de su perfilador que permite separar las secciones de código rápidas de las lentas y encontrar cuellos de botella rápidamente.

Firebug, como todas las herramientas utilizadas para el desarrollo del proyecto, es software libre, está licenciado bajo una licencia *BSD* y extensible por medio de módulos, cuenta con uno llamado *FireQuery* para una mayor integración con *jQuery*.

Figura 11. Firebug siendo utilizado para depurar código en JavaScript



Fuente: Autor(es)

[31] FIREBUG, WEB DEVELOPMENT EVOLVED. The Firebug Team. http://getfirebug.com/

4.3.1.5. JSDoc *JSDoc* es una sintaxis utilizada para agregar documentación *en línea* a código fuente escrito en *JavaScript*[32]. Su sintaxis es similar a la de *Javadoc*, usada para documentar código escrito en *Java*, pero está diseñada especialmente para la forma de trabajo más dinámica de *JavaScript*, por lo tanto no es totalmente compatible con *Javadoc*.

El *IDE Eclipse* provee una extensión que entiende la sintaxis de *JSDoc* y la presenta adecuadamente en el entorno de programación.

Uno de las facilidades que trae el uso de *JSDoc* es la facilidad que esta brinda para generar dicha documentación en *HTML*. Para dicha labor en este proyecto se utilizó un *script* escrito en *perl* para exportar toda la documentación del *API* del motor gráfico.

4.3.1.6. Apache Subversion *Subversion* es un sistema de control de versiones originalmente diseñado para reemplazar a *CVS* y ser un mejor sistema concurrente de versiones[33]. Un sistema de control de versiones es un aplicaciones que lleva el registro de todo el trabajo y los cambios en el código fuente que forman parte de un proyecto de software programa y permite que distintos desarrolladores colaboren en un repositorio centralizado.

Utiliza una arquitectura de cliente servidor, el servidor guarda un historial de versiones y la versión actual de un programa. Los clientes se conectan al servidor para obtener la última versión y poder trabajar sobre esta y si el cliente cuenta con los privilegios necesarios puede enviar los cambios al servidor para que el resto de

[32]JSDOC - JAVASCRIPT DOCUMENTATION TOOL. The JSDoc Team. http://jsdoc.sourceforge.net/

[33] APACHE SUBVERSION: "ENTERPRISE-CLASS CENTRALIZED VERSION CONTROL FOR THE MASSES". Apache Foundation. http://subversion.apache.org/

los desarrolladores tengan acceso a sus cambios en el código fuente.

Subversion se encuentra disponible al público bajo la licencia *Apache*, la cual permite su libre acceso y distribución. Es usado por una gran cantidad de proyectos de software libre como por ejemplo la *Apache Software Foundation*, *FreeBSD*, *GCC*, *Django*, *Ruby*, *Mono* y *PHP*.

Dada la naturaleza del proyecto fue necesario el uso de un sistema recurrente de versiones flexible, útil tanto para proyectos grandes como también aquellos con equipos de trabajo pequeños, para así poder distribuir efectivamente el desarrollo de *Beater2D*.

4.3.1.7. SourceForge.net *SourceForge.net* es un repositorio de código fuente basado en la web para la administración del desarrollo de proyectos de software[34]. Es el repositorio de software libre más amplio a nivel mundial, provee servicios gratuitos para el desarrollo colaborativo de software. Según datos de Febrero de 2009 hospeda más de 230,000 proyectos de software[35].

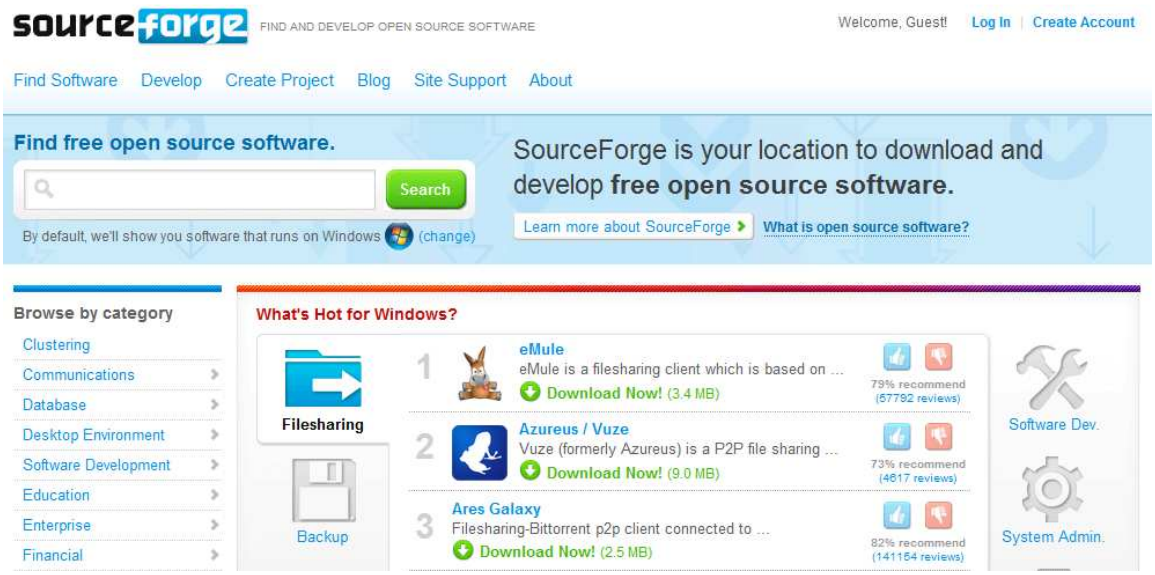
Gracias a *SourceForge.ne*t los desarrolladores tienen acceso centralizado a almacenamiento y herramientas para el manejo de proyectos, aunque es más conocido por proveer sistemas de manejadores de versiones tales como lo son *CVS*, *Subversion*, *Bazaar*, *Git*, *Mercurial*, entre otros.

---

[34] WHAT IS SOURCEFORGE.NET. Geeknet, Inc. http://sourceforge.net/apps/trac/sourceforge/wiki/What%20is%20SourceForge.net

[35] SOURCEFORGE.NET SITE PROFILE. Compete.com. http://siteanalytics.compete.com/sourceforge.net/?metric=uv

Figura 12. Página principal de Sourceforge.net



Fuente: http://www.sourceforge.net

Era patente la necesidad de los servicios que presta un sitio como *SourceFource.net*, y se eligió su uso sobre el de otros sitios similares por ser este el que concentra una cantidad mayor de visitas y usuarios, más de 33 millones de visitantes hasta Agosto de 2009 y 2'000.000 de usuarios registrados hasta 2009, brindando una visibilidad mayor para el proyecto para atraer, en un futuro, nuevos usuarios y desarrolladores de *Beater2D*.

4.4. FASE DE LIBERACIÓN

El objetivo de la fase final del proyecto fue empaquetar el producto desarrollado brindando facilidades para el contacto inicial por parte de los desarrolladores que estén interesados en utilizar el motor gráfico en sus propios proyectos de software.

Se busca asegurar una aceptación y adaptación sin complicaciones del motor gráfico. Enmarcadas en esta fase están las labores de implementación de dos aplicaciones de demostración, que a la vez sirven como guía para el desarrollo de software basado en el motor gráfico. Las aplicaciones de demostración ilustran claramente todas las capacidades del motor. Se profundiza en el desarrollo de estas aplicaciones en el capítulo 6.

Conjunto a las aplicaciones de demostración se elaboraron y publicaron los diferentes documentos de soporte, como la guía de referencia del *API* de *Beater2D* que se puede consultar como el Anexo D del actual documento.

5. PRUEBAS

Conjunto a las pruebas de unidad realizadas durante la fase de codificación en donde se verificaba el correcto funcionamiento de cada unidad de código, asegurando que cada módulo funciona correctamente por separado, se realizaron otra serie de pruebas de integración y rendimiento, cuyos resultados se presentan en este capítulo.

En esta fase del proyecto se estudió exhaustivamente el motor implementado en busca de errores y fallas que pudieran tener un impacto negativo en la integridad, confiabilidad, disponibilidad y rendimiento del motor gráfico desarrollado. Una vez concluida la codificación se han practicado dos tipos de prueba como se enuncian a continuación.

5.1. PRUEBAS DE VERIFICACIÓN Y VALIDACIÓN

Este tipo de pruebas están regidas por la verificación del cumplimiento de las pautas plasmadas en el Documento de Especificación de Requisitos del Sistema y su especificación funcional, definidos respectivamente en las fases de análisis y diseño, que se encuentran documentadas como anexos del presente texto.

Una revisión mesurada del sistema y la comparación de sus funcionalidades finales con las plasmadas en los documentos de especificaciones han comprobado que la implementación de los diferentes módulos del motor gráfico se

ha realizado de manera exitosa y su funcionamiento es el correcto. El proceso de verificación de funcionalidades del prototipo y plan de pruebas se realizó utilizando la lista de chequeo presentada en el Anexo E. La verificación de cada uno de los puntos exigidos por la lista de chequeo se realizó en varias oportunidades hasta que se verifico el cumplimiento de todos ellos en los navegadores web *Mozilla Firef*ox (versión 3.6.12), *Google Chrome* (8.0.552.237) y *Opera* (11.00).

Acompañando a la revisión de los documentos de especificaciones del sistema se ha puesto a prueba su correcto funcionamiento en la fase de liberación, en la cual se desarrollaron dos aplicaciones de demostración tomando como librería base el motor gráfico *Beater2D*. La implementación de las aplicaciones de demostración se realizó exitosamente y sus resultados se presentan en el capítulo 6. Con este desarrollo de prueba y documentación se ha comprobado el correcto funcionamiento del motor gráfico, evaluando todos sus módulos, el correcto procesamiento de órdenes y su eficiencia.

De este modo se concluye que las pruebas de verificación y validación han sido superadas con éxito por el prototipo, cumpliendo así con las funcionalidades previstas.

5.2. PRUEBAS DE RENDIMIENTO

Para evaluar el rendimiento de prototipo desarrollado se han realizado sobre las aplicaciones de demostración una serie de pruebas para verificar la capacidad y rendimiento de un programa construido utilizando el motor gráfico en ejecución.

Los *benchmarks* realizados se basaron en las aplicaciones de demostración desarrolladas con el prototipo *Beater2D*. Estas pruebas se centraron en la medición del uso de la *CPU* cuando las aplicaciones están ejecutándose a determinados Cuadros por segundo o *FPS* (*Frames Per Second*), esto sobre distintos navegadores y sistemas operativos.

Se usaron dos máquinas diferentes para cada una de las pruebas, sus características y los resultados de las pruebas realizadas pueden apreciarse en las siguientes tablas y figuras.

Tabla 3. Características de la máquina de pruebas Número 1

Características	Descripción
Tipo de máquina	Desktop
Procesador	Core 2 Duo
Frecuencia del procesador	2.0 GHz
Memoria Ram	2 GB
Frecuencia de la Ram	667 MHZ
Bus del sistema	800 MHZ
Tarjeta de Gráficos	ATI Radeon HD 4670

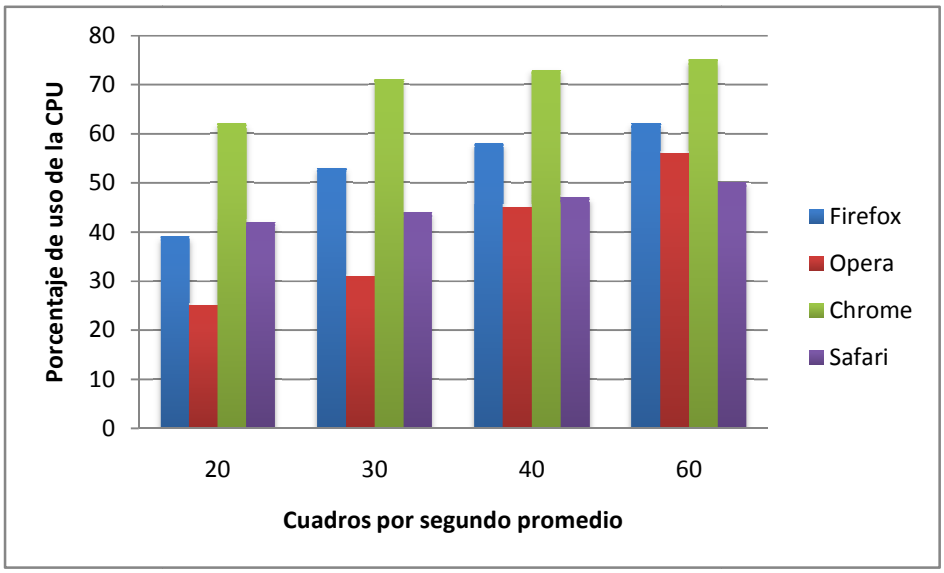
Fuente: Autor(es)

Tabla 4. Características de la máquina de pruebas Número 2

Características	Descripción
Tipo de máquina	Laptop (MacBook)
Procesador	Core 2 Duo
Frecuencia del procesador	2.4 GHz
Memoria Ram	2 GB
Frecuencia de la Ram	1067 MHz
Bus del sistema	1067 MHz
Tarjeta de Gráficos	Nvidia Geforce 320M

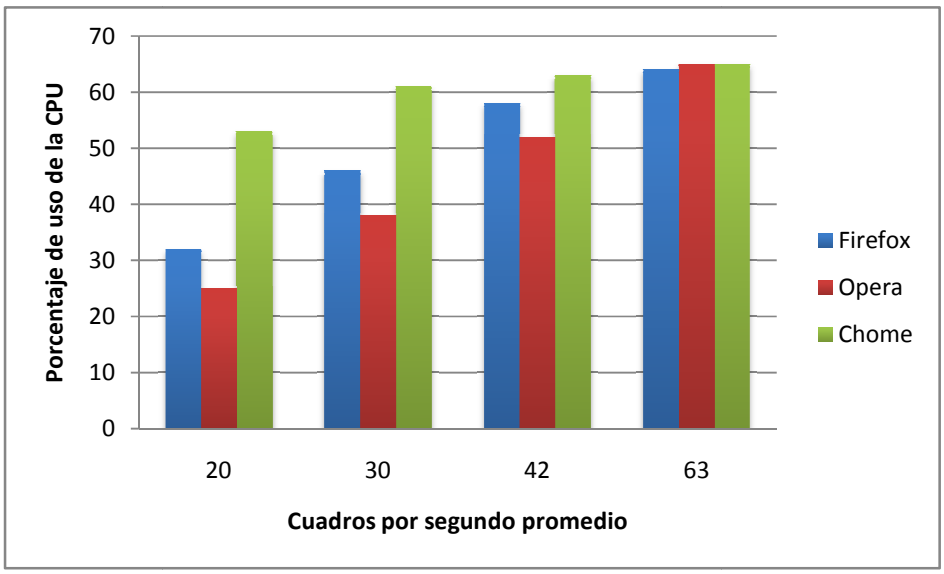
Fuente: Autor(es)

Figura 13. Uso de la CPU por parte de WebFrogatto en el equipo 1 usando el sistema operativo Windows 7



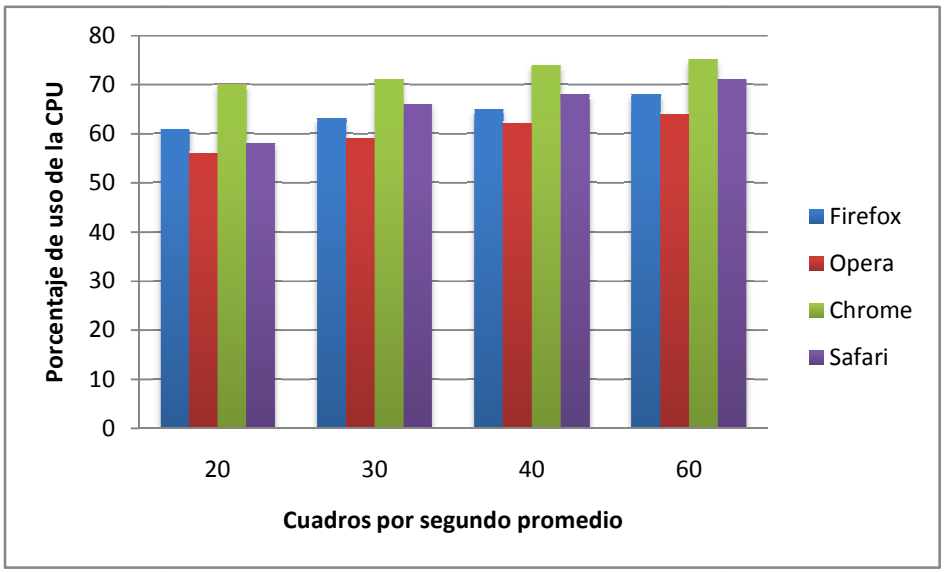
Fuente: Autor(es)

Figura 14. Uso de la CPU por parte de WebFrogatto en el equipo 1 usando el sistema operativo Debian GNU/Linux



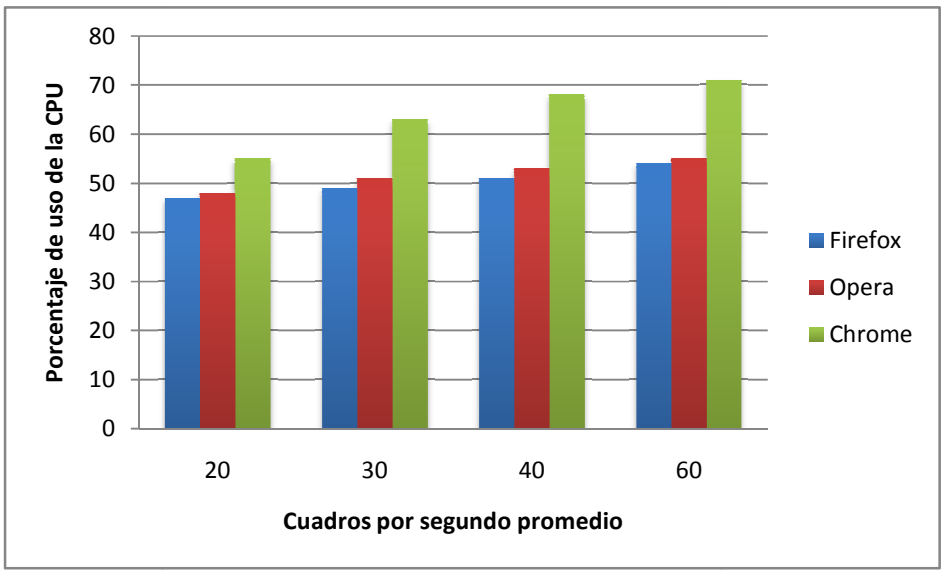
Fuente: Autor(es)

Figura 15. Uso de la CPU por parte de PhysicPaint en el equipo 1 usando el sistema operativo Windows 7



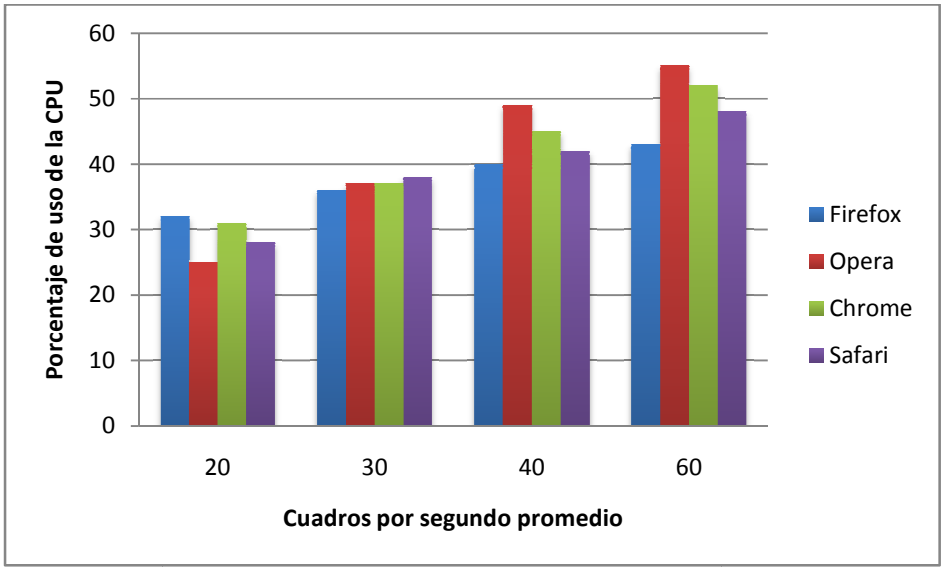
Fuente: Autor(es)

Figura 16. Uso de la CPU por parte de PhysicPaint en el equipo 1 usando el sistema operativo Debian GNU/Linux



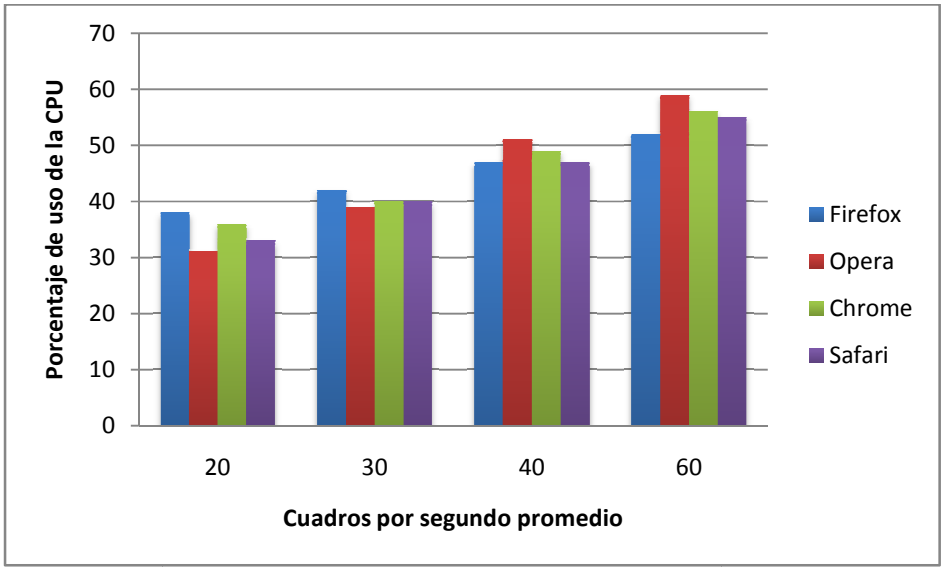
Fuente: Autor(es)

Figura 17. Uso de la CPU por parte de WebFrogatto en el equipo 2 usando el sistema operativo Mac OSX Snow Leopard.



Fuente: Autor(es)

Figura 18. Uso de la CPU por parte de PhysicPaint en el equipo 2 usando el sistema operativo Mac OSX Snow Leopard



Fuente: Autor(es)

Las pruebas revelaron que el motor se comporta de manera consistente, el consumo de recursos del sistema no presenta sorpresas y su demanda se incrementa según lo esperado, al incrementar los cuadros por segundo a los cuales se ejecuta una instancia del motor incremente el uso de la *CPU* por parte de los navegadores sobre los cuales se probaron las aplicaciones de demostración.

También se puede observar que el uso del procesador es mayor en la aplicación que hace uso del motor físico, ya que los cálculos que se tienen que realizar para la simulación de colisiones en dos dimensiones demandan procesamiento intensivo, aumentando el uso del procesador con respecto a la simple representación de gráficos y animaciones pre-determinadas.

En general el consumo de *CPU* obtenido por el motor es aceptable y permite desarrollar aplicaciones complejas, pero cabe anotar que las versiones de los navegadores utilizadas para realizar las pruebas aun no cuentan con aceleración gráfica para representar las instancias del elemento *Canvas*, por lo cual recurren totalmente al procesador principal del computador. Las versiones futuras de todos los navegadores modernos tienen planificado el soporte parar aceleración *2D* y *3D* por medio de tarjetas graficadoras, incrementando así el rendimiento de las instancias del elemento *Canvas* y por lo tanto de *Beater2D*.

6. APLICACIONES DE DEMOSTRACIÓN

Como complemento al desarrollo del prototipo de motor gráfico *Beater2D* se planteó el desarrollo de dos aplicaciones de demostración, cuyo fin es servir como material de referencia y documentación sobre el desarrollo de aplicaciones utilizando *Beater2D*.

En el presente capítulo se describen las aplicaciones de demostración implementadas, su uso y las funcionales del motor utilizadas en cada una.

6.1. DEMOSTRACIÓN 1: PHYSICPAINT

6.1.1. Descripción *PhysicPaint* es una aplicación de demostración inspirada en *Crayon Physics Deluxe*[36], un juego de acertijos en el que se hace énfasis en la física de dos dimensiones, sobre todo en los conceptos de gravedad, masa, energía cinética y transferencia de *momentum*.

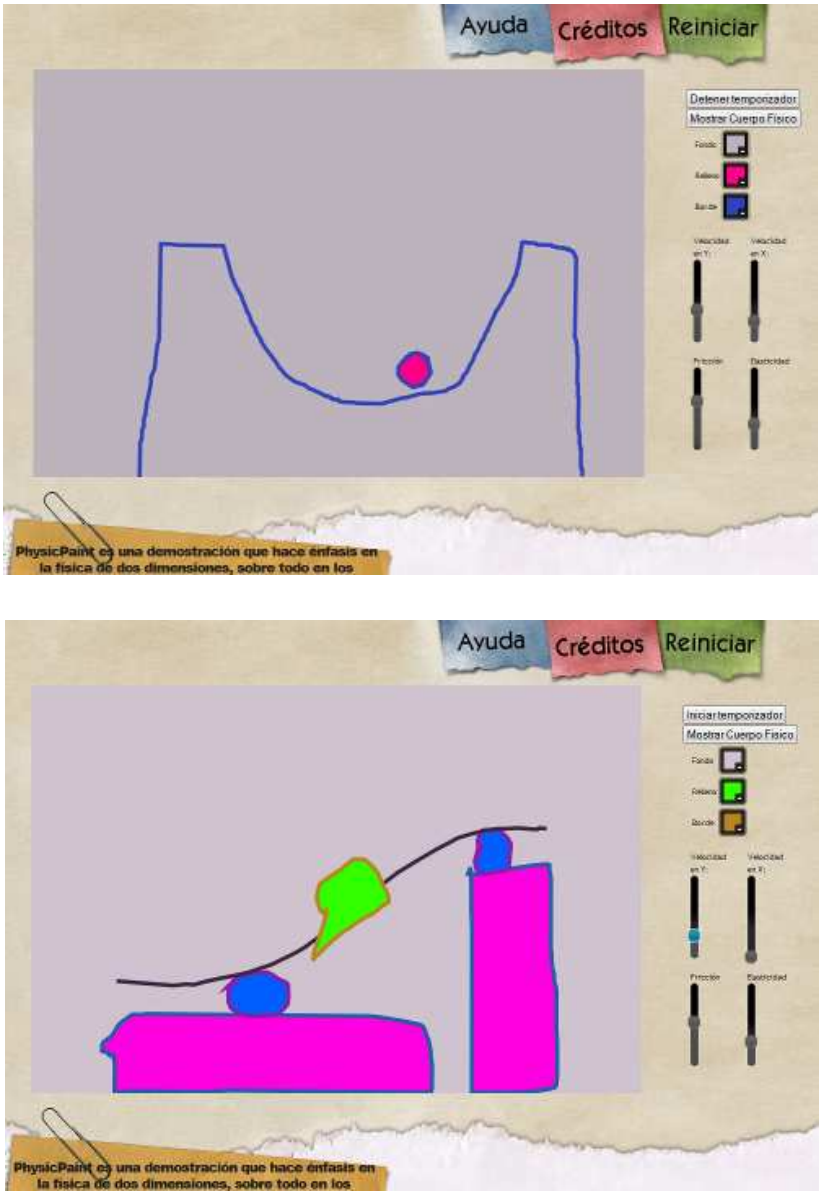
Esta aplicación permite dibujar distintas formas y cuerpos sobre un lienzo que representa el mundo físico del motor gráfico, construido sobre *Box2DJS*. Una vez dibujado y representado gráficamente un objeto, el motor gráfico calcula la forma física que más se aproxima al objeto bosquejado y le asigna unas propiedades básicas para interactuar con el resto de objetos pertenecientes al mundo físico.

[36] CRAYON PHYSICS DELUXE. Petri Purho. http://www.crayonphysics.com/

Así cada objeto que un usuario agregue sobre el lienzo interactuará con los dibujados previamente, obteniendo una simulación física fluida y realista.

La demostración no tiene un objetivo o meta que el usuario deba cumplir más allá de poder interactuar con el mundo físico.

Figura 19. Capturas de pantalla de PhysicPaint en acción



Fuente: Autor(es)

El fin de esta aplicación es ilustrar las capacidades de captura de gestos del mouse, y la representación gráfica y física que *Beater2D* puede realizar de distintos elementos.

6.1.2. Funcionalidades La versión de *PhysicPaint* desarrollada como demostración inicial del prototipo de motor bidimensional *Beater2D* cuenta con las siguientes capacidades:

- Iniciar o detener el temporizador: realizando clic en el botón destinado para esta funcionalidad la simulación puede detenerse y reanudarse, según sea el caso.

- Dibujar una línea: permite representar líneas rectas y curvas sobre el lienzo de la aplicación, el motor físico se aproximará lo máximo posible sin afectar el rendimiento del motor, las líneas dibujadas por defecto cuentan con un color, velocidad en X y Y, elasticidad y coeficiente de fricción. Antes de iniciar el bosquejo de una línea es posible definir las propiedades de una línea, utilizando los campos y botones destinados para esta labor.

- Dibujar cuerpo: si una línea dibujada en el lienzo es cerrada, es decir que su punto inicial y final se sobreponen, la línea se convierte en un cuerpo solido, al cual en el mundo físico se le asigna el sólido que mejor se aproxime a la forma original sin perder rendimiento. En el área de propiedades es posible definir valores de color de línea, colore de relleno, velocidad en X y Y, elasticidad y coeficiente de fricción para los cuerpos que se dibujan en el lienzo.

- Mostrar cuerpo físico: por medio de esta función se puede ver la aproximación usada en el mundo físico de cada objeto dibujado.

6.2. DEMOSTRACIÓN 2: WEBFROGATTO

6.2.1. Descripción *WebFrogatto* es un juego de plataformas en dos dimensiones clásico, inspirado en el juego de código abierto *Frogatto*[37]. Como en muchos juegos de clásicos cuenta con una vista lateral de corte transversal, en la que están representados el personaje principal, una rana verde controlada por el usuario a través del teclado, los enemigos y diversos obstáculos y plataformas.

En esta demostración el personaje controlado por el usuario tiene que avanzar horizontalmente enfrentándose a diversas criaturas y sorteando una serie de obstáculos, evitando ser alcanzo por los ataques enemigos y caer en algún abismo, lo cual representaría el fin del juego.

Figura 20. Captura de pantalla de WebFrogatto en acción

Fuente: Autor(es)

[37] FROGATTO & FRIENDS. The Frogatto Team. http://www.frogatto.com/

6.2.2. Funcionalidades La versión de *WebFrogatto* desarrollada como demostración inicial del prototipo de motor bidimensional *Beater2D* cuenta con las siguientes características:

- Pantalla Inicial: es la presentación inicial del video juego, en ella se presentan los créditos del software y hace posible iniciar un nuevo juego.

- Personaje principal: puede ser manejado por el usuario a través del teclado de su computador, se mueve horizontalmente avanzando hacia la derecha, puede saltar, caminar, correr y lanzar su lengua, con la cual atrapa a sus enemigos y los engulle. Cuando el personaje principal tiene un enemigo en la boca puede escupirlo en cualquier dirección para golpear a otro enemigo, esta es la única manera como los enemigos pueden ser eliminados.

- Enemigos: en el transcurso de un nivel de *WebFrogatto* se presentan en pantalla varios enemigos que pueden detener el avance del personaje principal y dar por terminado el juego. Existen tres tipos de enemigos, cada uno con sus propias características.

- Mundo o nivel: representa las plataformas sobre las cuales puede desplazarse el personaje principal, también se refiere al conjunto de decoraciones con las que cuenta un nivel (imágenes de fondo, árboles, arbustos, etc). Puede definirse por medio de un archivo *XML*, esto hace que realizar nuevos niveles de *WebFrogatto* sea un proceso fácil y rápido.

- Objetos interactivos: puede encontrarse a lo largo de un nivel y reaccionan a alguna acción realizada por el personaje principal, por ejemplo un trampolín que se activa al pasar sobre él o una fuente de agua que recupera los puntos de salud del personaje principal.

7. MANUALES Y GUIAS

A lo largo del presente capítulo se exponen los diferentes tipos de manuales para que tanto los desarrolladores que decidan construir sus paquetes de software sobre el prototipo de motor gráfico *Beater2D* y como los desarrolladores de nuevas funcionalidades del sistema una guía con la que pueden profundizar en el uso y la construcción del sistema.

7.1. MANUAL DE INSTALACIÓN DE BEATER 2D

A continuación se muestran los pasos a seguir para importar el motor gráfico *Beater2D* para ser usado en el desarrollo de aplicaciones web sobre cualquier sistema operativo moderno, válidos desde que se cumplan los requisitos de hardware y software.

Para desarrollar y ejecutar aplicaciones desarrolladas sobre las librerías de *Beater2D* es requisito contar con un dispositivo que tenga la capacidad de correr cualquier navegador web moderno con soporte para *HTML5* y el elemento Canvas. Por ejemplo *Mozilla Firefox* (Pentium 233 MHz (*Recomendado:* Pentium 500MHz o superior), 64 MB RAM (*Recomendado:* 128 MB RAM o más), 52 MB de espacio en disco).

El único requisito de software de *Beater2D* es un navegador web moderno, el prototipo ha sido probado y tiene soporte sobre *Mozilla Firefox* (Versión 3.6.13), *Google Chrome* (8.0.552.224) y *Opera* (11.0), aunque pueden ejecutarse aplicaciones desarrolladas con el motor exitosamente sobre otros navegadores con soporte para *HTLM5* y *Canvas* y también en versiones anteriores de *Firefox*, *Chrome* y *Opera*.

Los pasos a seguir para importar las librerías de *Beater2D* son los siguientes:

- Crear un directorio para el proyecto, por ejemplo “juego”.
- Dentro del directorio del proyecto crear otro donde se alojaran las librerías de *Beater2D*, por ejemplo “juego/beater2d”.
- Copiar los archivos de *Beater2D* dentro del directorio creado en el paso anterior.
- En la carpeta principal del proyecto crear el archivo *HTML* principal del proyecto, por ejemplo “juego/index.html”.
- En la etiqueta *head* en index.html importar los módulos y sub-módulos del prototipo de motor gráfico que se deseen usar.
- Definir una etiqueta <*script type="text/javascript"*> la cual guiará el flujo principal de la aplicación desarrollada.
- En la etiqueta *body* en index.html definir una instancia del elemento *Canvas* con un ID igual al que se use en el script de *JavaScript* que guía la aplicación desarrollada.

A continuación un ejemplo sencillo de uso del prototipo en el cual se anima un *sprite* compuesto por cuatro cuadros:

```
<html>
<head>
    <script src="../beaterjs/libs/jquery-1.4.2.min.js"></script>
    <script src="../beaterjs/graphics/graphics.drawable.js"></script>
    <script src="../beaterjs/graphics/graphics.shape.js"></script>
    <script src="../beaterjs/graphics/graphics.image.js"></script>
    <script src="../beaterjs/graphics/graphics.sprite.js"></script>
    <script type="text/javascript">
        $().ready(function() {

            beater = new $.Beater("canvasgif",200,200);
            beater.lienzo.setBackgroundcolor("222222");
            beater.fps=15;
            spritegif = new $.Sprite(beater.lienzo.context);
            spritegif.addblock("default", "sprite.png", 4)
            spritegif.action = function(){
                this.next();
            }
            beater.addObject(spritegif);
            beater.Start();

        });
    </script>
</head>
<body>
    <canvas id="canvasgif"></canvas>
</body>
</html>
```

Fuente: Autor(es)

Para ejemplos más complejos y material de referencia del uso del resto de las librerías de *Beater2D* se recomienda revisar el código fuente de las aplicaciones de demostración descritas en el capítulo 6.

7.2. GUÍA DE REFERENCIA

Como material de referencia para desarrolladores de aplicaciones como de nuevas características y módulos para *Beater2D* se ha compilado la información relativa a las funciones implementadas en el prototipo inicial de *Beater2D*.

Esta guía de referencia puede consultarse en el Anexo D titulado Manual de referencia del *API* de *Beater2D*.

8. CONCLUSIONES

Los objetivos trazados en el desarrollo del proyecto de grado se han cumplido, se ha implementado un prototipo de motor gráfico en dos dimensiones con las herramientas que ofrece el borrador de *HTML5* y su la actual implementación. Se ha logrado, con el desarrollo de las librerías gráficas básicas facilitar la implementación de aplicaciones web enriquecidas usando *JavaScript* y el elemento *Canvas*. Se prescindió del uso de *CSS* en *Beater2D* ya que no presenta ninguna funcionalidad sobre las instancias del elemento Canvas, esto hace a *CCS* inútil para el prototipo.

Se ha desarrollado un prototipo de motor gráfico robusto, funcional y modular pasando por las diferentes etapas de análisis, diseño e implementación, con base en el documento de Especificación de Requisitos de Software definido por los autores del proyecto, implementando la base en el módulo central y sobre él los módulos de audio, gráficos, eventos, y el contenedor del motor físico *Box2DJS*.

El motor gráfico que se ha implementado es un motor experimental con capacidades básicas cimentadas sobre un modelo robusto, el cual sienta las bases para el desarrollo futuro de un motor gráfico complejo que sirva para cubrir las necesidades de diversos proyectos de software que sigan el paradigma de las aplicaciones web enriquecidas, ya sea para el desarrollo de video juegos, simulaciones físicas en dos dimensiones o simples entornos de usuario gráficos.

El desarrollo del proyecto según la metodología planteada permitió organizar el trabajo de la manera correcta para la naturaleza del proyecto, siguiendo un flujo

que originó un mayor orden y facilitó la producción de la documentación necesaria para publicar el proyecto y organizar trabajos futuros.

Se han logrado implementar aplicaciones de demostración que explotan el potencial de *Beater2D*. Las pruebas realizadas sobre las aplicaciones de demostración presentaron un rendimiento más que aceptable para el estado actual de la implementación del borrador de *HTML5* en los navegadores web en los sistemas operativos usados en las pruebas. Estas demostraciones también cumplen el papel de guía de referencia y muestras del alcance del prototipo de motor gráfico desarrollado.

Cabe destacar que el modelo actual de *HTML5* aun es un borrador del *World Wide Web Consortium* (*W3C*) por lo cual sus características finales aun no están definidas y la implementación de estas por los distintos navegadores web se encuentra en fase de experimentación y desarrollo. Esto significa que trabajar sobre este borrador implica estar a la vanguardia tecnológica en el desarrollo de aplicaciones web y supone un esfuerzo de apropiación tecnológica, lo cual fortalece la imagen institucional de la Universidad Industrial de Santander y la Escuela de Ingeniería de Sistemas e Informática. El estado actual de las aplicaciones en *HTML5* favorece la investigación y publicación de conocimientos en estas nuevas tecnologías, centrarse en el uso y desarrollo de ellas ayudará al proceso de migración de la web a tecnologías abiertas y estándares.

9. RECOMENDACIONES

- Dar continuidad al desarrollo del motor gráfico *Beater2D* mediante una segunda fase, agregando nuevos módulos y funcionalidades como lo son inteligencia artificial para videojuegos, manejo de perspectiva isométrica para gráficos en dos dimensiones, soporte para la etiqueta *video*, transformaciones gráficas con *CSS*, manejo de bases de datos e infraestructura para juegos con soporte para juegos con multi-jugador y masivos por Internet.

- Aun no se recomienda el uso de *Beater2D* para el desarrollo de aplicaciones críticas ya que su estado es el de un prototipo debido al posible cambio en las especificaciones de *HTML5*.

- *Beater2D* sólo cuenta con soporte para formatos de audio libres de patentes para el uso del módulo de sonido, es vital el uso del *codec Ogg Vorbis* para la compresión de los archivos de audio que deseen ser usador en el desarrollo de una aplicación con el prototipo de motor gráfico.

- En los proyectos planeados para darle continuidad al desarrollo de *Beater2D* es necesario la constante actualización sobre el borrador de *HTML5* y su implementación en los diferentes navegadores web, actualizando las funciones desarrolladas hasta ahora para que cumplan los posibles cambios en el borrador. La implementación de nuevos módulos de funcionalidades del motor gráfico debe estar diseñada de tal forma que sea fácil introducir futuros cambios que se deriven de nuevas especificaciones de *HTML5* o cambios en las ya utilizadas.

BIBLIOGRAFÍA

POWERS, Shelley. JavaScript Cookbook. O'Reilly Media, 2010.

LUBBERS, Peter; ALBERS, Brian; SMITH, Ric; SALIM, Frank. Pro HTML5 Programming: Powerful APIs for Richer Internet Application Development. Apress Berkely, 2010.

DAVID, Matthew. HTML5: Designing Rich Internet Applications. Focal Press, 2010.

KLETSCH, Christian; VOLK, Daniel. Towards an AJAX-based game engine. Proceedings of the 2008 Conference on Future Play: Research, Play, Share, New York, USA, 2008.

GOMEZ F., Luis Carlos. Guía para el desarrollo de proyectos de Grado. Departamento de Ingeniería de Sistemas y Computación, UIS, 1993.

WHAT WORKING GROUP. HTML Standard. Disponible en http://www.whatwg.org/specs/web-apps/current-work/multipage/

WORLD WIDE WEB CONSORTIUM. HTML5 Editor's Draft. Disponible en http://dev.w3.org/html5/spec/spec.html

WORLD WIDE WEB CONSORTIUM. HTML5: The Markup Language Reference. Disponible en http://dev.w3.org/html5/markup/

WORLD WIDE WEB CONSORTIUM. HTML 5 Reference, A Web Developer’s Guide to HTML 5. Disponible en http://dev.w3.org/html5/html-author/

WORLD WIDE WEB CONSORTIUM. HTML Design Principles. Disponible en http://dev.w3.org/html5/html-design-principles/

REFERENCIAS

1. HTML: THE MARKUP LANGUAGE. World Wide Web Consortium. Unofficial Editor's Draft 18 December 2010. http://dev.w3.org/html5/markup
2. WHAT OPEN MAILING LIST ANNOUNCEMENT. WHAT Working Group. http://lists.whatwg.org/htdig.cgi/whatwg-whatwg.org/2004-June/000005.html
3. WHEN WILL HTML5 BE FINISHED? WHAT Working Group. http://wiki.whatwg.org/wiki/FAQ#When_will_HTML5_be_finished.3F
4. Ibid.
5. THE CANVAS ELEMENT. WHAT Working Group. http://www.whatwg.org/specs/web-apps/current-work/multipage/the-canvas-element.html
6. RIA WAR IS BREWING. eWeek Magazine, Jim Rapoza. http://etech.eweek.com/content/application_development/ria_war_is_brewing.html
7. RICH INTERNET APPLICATION MARKET SHARE. Stat Owl. http://www.statowl.com/custom_ria_market_penetration.php
8. RICH INTERNET APPLICATION STATISTICS. Real world stats of RIA plugin deployments. http://www.riastats.com/
9. HTML5 OVERVIEW. World Wide Web Consortium. http://dev.w3.org/html5/spec/Overview.html#html-vs-xhtml
10. NEW ELEMENTS IN HTML 5. IBM Developer Works. http://www.ibm.com/developerworks/library/x-html5/?ca=dgr-lnxw01NewHTML
11. HTML5 DIFFERENCES FROM HTML4. World Wide Web Consortium. http://dev.w3.org/html5/html4-differences
12. HTML5 DIFFERENCES FROM HTML4: APIS. World Wide Web Consortium. http://dev.w3.org/html5/html4-differences/#apis
13. CANVAS 2D API SPECIFICATION 1.0. World Wide Web Consortium. http://dev.w3.org/html5/canvas-api/canvas-2d-api.html
14. ECMASCRIPT LANGUAGE SPECIFICATION. ECMA International. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-262.pdf
15. THE A-Z OF PROGRAMMING LANGUAGES: JAVASCRIPT. Naomi Hamilton, http://www.computerworld.com.au/article/255293/a-z_programming_languages_javascript/
16. MILESTONES IN THE HISTORY OF THEMATIC CARTOGRAPHY, STATISTICAL GRAPHICS, AND DATA VISUALIZATION. Michael Friendly. 2008

17. WHAT ARE COMPUTER GRAPHICS? University of Leeds ISS. http://iss.leeds.ac.uk/info/306/graphics/215/overview_of_computer_graphics/2
18. BOX2D: FEATURES. Box2D Physics Engine. http://box2d.org/features.html
19. PROJECTS USING THE BULLET ENGINE. Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/wiki/Bullet_%28software%29#Projects_using_the_engine
20. BOX2D MANUAL: COLLISION MODULE. Box2D Physics Engine. http://box2d.org/manual.html#_Toc258082970
21. BOX2D MANUAL: DYNAMICS MODULE. Box2D Physics Engine. http://box2d.org/manual.html#_Toc258082971
22. UNA EXPLICACIÓN DE LA PROGRAMACIÓN EXTREMA: ACEPTAR EL CAMBIO. Kent Beck . Addison-Wesley Iberoamericana Espanya, S.A. 2002
23. GETTING FROM USE CASES TO CODE, PART 1: USE-CASE ANALYSIS. Gary Evans. http://www.ibm.com/developerworks/rational/library/5383.html
24. LA PROGRAMACIÓN EXTREMA EN LA PRÁCTICA. James Newkirk, Robert C. Martin. Addison-Wesley Iberoamericana Espanya, S.A. 2002
25. THE CANVAS ELEMENT. World Wide Web Consortium. http://www.w3.org/TR/html5/the-canvas-element.html#the-canvas-element
26. JQUERY: THE WRITE LESS, DO MORE, JAVASCRIPT LIBRARY. http://jquery.com/
27. BOX2D PHYSICS ENGINE. http://www.box2d.org/
28. JQUERY 1.3 AND THE JQUERY FOUNDATION. The jQuery Project. http://blog.jquery.com/2009/01/14/jquery-13-and-the-jquery-foundation/
29. JQUERY USAGE STATISTICS. Builtwith. http://trends.builtwith.com/javascript/JQuery
30. ECLIPSE IDE FOR JAVASCRIPT WEB DEVELOPERS. The Eclipse Foundation. http://www.eclipse.org/downloads/packages/eclipse-ide-javascript-web-developers/heliossr1
31. FIREBUG, WEB DEVELOPMENT EVOLVED. The Firebug Team. http://getfirebug.com/
32. JSDOC - JAVASCRIPT DOCUMENTATION TOOL. The JSDoc Team. http://jsdoc.sourceforge.net/
33. APACHE SUBVERSION: “ENTERPRISE-CLASS CENTRALIZED VERSION CONTROL FOR THE MASSES”. Apache Foundation. http://subversion.apache.org/
34. WHAT IS SOURCEFORGE.NET. Geeknet, Inc. http://sourceforge.net/apps/trac/sourceforge/wiki/What%20is%20SourceForge.net

35. SOURCEFORGE.NET SITE PROFILE. Compete.com. http://siteanalytics.compete.com/sourceforge.net/?metric=uv
36. CRAYON PHYSICS DELUXE. Petri Purho. http://www.crayonphysics.com/
37. FROGATTO & FRIENDS. The Frogatto Team. http://www.frogatto.com/

ANEXOS

Anexo A: Cuadro comparativo de los framework de Javascript

JavaScript se ha convertido en uno de los lenguajes más usados por desarrolladores web. Aunque, inicialmente, muchos programadores profesionales prescindieron del lenguaje por su planteamiento basado en prototipos. El incremento del interés por las Aplicaciones de Internet Enriquecidas o *RIAs* por sus siglas en ingles junto con el advenimiento de *Ajax* ha permitido la proliferación de *frameworks* y librerías, impulsando mejores prácticas de desarrollo en *JavaScript*.

Este aumento da como resultado la existencia de *frameworks* de todo tipo y que suplen diferentes necesidades, por lo tanto no es sencillo hacer una comparación de este tipo.

Dado el carácter del proyecto el tipo de *framework* más adecuado es aquel que cuente con un núcleo pequeño, flexible y rápido. Sobre estos requisitos se estudiaron 4 librerías con capacidades comprobadas y reconocidas, ellas son: *Dojo*, *jQuery*, *Mootools* y *Prototype*.

Las principales características de los *frameworks* nombrados puede encontrase en la siguiente tabla:

Comparativa de frameworks de JavaScript				
Característica	**Dojo**	**jQuery**	**MooTools**	**Prototype**
Información General				
Versión Comparada	1.5.0	1.4.2	1.2.4	1,6,1
Tamaño del paquete base	123 KB	155 KB	101 KB	136 KB
Licencia	BSD y AFL	MIT y GPL	MIT	MIT
Funcionalidades				
	Dojo	**jQuery**	**MooTools**	**Prototype**
Detección de características	No	Sí	No	No
Recuperación de datos con XMLHTTPRequest	Sí	Sí	Sí	Sí
Recuperación de datos con JSON	Sí	Sí	Sí	Sí
Arrastras y soltar	Sí	Sí	Sí	Sí
Efectos visuales simples	Sí	Sí	Sí	Sí
Manejo de eventos	Sí	Sí	Sí	Sí
Manejo del historial	Si	Con complemento	Con Complemento	Sí
Validación de entradas	Sí	Con complemento	Sí	Sí
Grilla	Sí	Con complemento	Con complemento	No
Editor de texto enriquecido	Sí	Con complemento	Sí	No
Herramientas de autocompletado	Sí	Con complemento	Con complemento	Sí
Generación de HTML	Sí	Sí	Sí	Sí
Soporte para eventos táctiles	Sí	Con complemento	Con complemento	No
Acorde a ARIA (Iniciativa para mejor la accesibilidad en las RIAs)	Sí	Sí	No	No
Generación de tablas y cuadros	Sí	Con complemento	No	No
Soporte a navegadores web (versión a partir)				
	Dojo	**jQuery**	**MooTools**	**Prototype**
Internet Explorer	“6+”	“6+”	“6+”	“6+”
Mozilla Firefox	“3+”	“2+”	“1,5+”	“1,5+”
Safari	“4+”	“3+”	“2+”	“2+”
Opera	“10+”	“9+”	“9+”	“9.25+”
Chrome	“3+”	“1+”	“1+”	“1+”

Fuente: Autor(es)

Analizando detenidamente los beneficios y desventajas que trae consigo cada *framework* se optó por el uso de *jQuery*. Esta decisión fue fuertemente influenciada por su expansibilidad por medio de *plug-ins*, los cuales pueden usarse como complemento a las capacidades de *Beater2D*.

Anexo B: Cuadro comparativo de la implementación de HTML5 en diferentes navegadores

La nueva especificación del estándar para la presentación de documentos a través de la web *HTML5* tiene cambios significativos enmarcados en la creación de nuevos tipos de etiquetas para distintos propósitos. Esta especificación aun es un borrador de trabajo, no una recomendación del *World Wide Web Consortium* (*W3C*), por lo cual no es totalmente estable. Aun así los desarrolladores de varios de los motores de renderizado web utilizados por los navegadores más usados han iniciado a implementar varias de estas nuevas etiquetas, haciendo posible la implementación de páginas y aplicaciones web utilizando las nuevas funcionalidades que trae *HTML5*.

A continuación se presenta una tabla comparativa del nivel de implementación de *HTML5* en los motores *Gecko* (usado por *Mozilla Firefox*), *WebKit* (usado por *Safari*, *Google Chrome*, *Konqueror* y otros), *Presto* (*Opera*) y *Trident* (*Internet Explorer*).

Soporte de HTML5 por distintos exploradores				
Característica	Motor de Renderización			
	Gecko (Firefox 3.6.9)	WebKit (Chromium 6.0)	Presto (Opera 10.62)	Trident (Internet Explorer 8)
Reglas de análisis de sintáctico				
<!DOCTYPE html> desencadena modos del estándar	Sí	Sí	Sí	Sí
Muestreo de HTML5	No	No	No	No
Construcción de arboles de HTML5	No	No	No	No
SVG en tipos text/html	No	No	No	No
MathML en tipos text/html	No	No	No	No

Canvas				
Elemento Canvas	Sí	Sí	Sí	No
Contexto 2D	Sí	Sí	Sí	No
Text	Sí	Sí	Sí	No
Audio y Video				
Elemento video	Sí	Sí	Sí	No
Soporte para subtítulos	No	No	No	No
Elemento audio	Sí	Sí	Sí	No
Dispositivos Locales				
Elemento device	No	No	No	No
Nuevos Elementos				
Embeber datos no visibles personalizados	No	No	No	No
Elementos de sección	0/7	7/7	0/7	0/7
Elementos de agrupación de contenido	0/2	0/2	0/2	0/2
Elementos de nivel semántico-texto	0/5	1/5	0/5	0/5
Atributo hidden	No	Sí	No	No
Scroll en vistas	Sí	Sí	Sí	Sí
Atributo contenteditable	No	Sí	Sí	Sí
Formularios				
Tipos de elementos input	“0/13”	“13/13”	“10/13”	“0/13”
Atributos de elementos input	“1/10”	“8/10”	“8/10”	“0/10”
Otra forma de elementos	“0/5”	“3/5”	“2/5”	“0/5”
Validación de formularios	No	Sí	Sí	No
Interacción con el usuario				
Arrastrar y soltar	Sí	Sí	No	No
Historial de deshacer	No	No	No	No
Historial de sesiones	No	Sí	No	No
Selección de texto	Sí	Sí	Sí	No
Microdata				
Microdata	No	No	No	No
Aplicaciones Web				
Cache de aplicaciones	Sí	Sí	Sí	No
Controladores de esquema personalizados	Sí	No	No	No
Controladores de contenido personalizados	Sí	No	No	No

Fuente: Autor(es)

Anexo C: Documento de Especificación de Requisitos de Software

El presente documento recoge la información concerniente a la Especificación de Requisitos de Software (ERS) del motor gráfico en dos dimensiones *Beater2D*, el cual ha sido el producto principal de la fase de Análisis del proyecto.

La Especificación de Requisitos de Software enmarca las funcionalidades que ofrece el sistema, sus requerimientos funcionales y no funcionales. La especificación fue realizada con base en el análisis del borrador del estándar *HTML5*, su implementación en distintos navegadores web y las necesidades a resolver para facilitar el desarrollo de aplicaciones y juegos *2D* apoyados en esta nueva tecnología.

PROPÓSITO

El propósito de realizar el Documento de Especificación de Requerimientos de Software es definir claramente cuáles son los requisitos que debe cumplir un motor gráfico bidimensional como el planteado.

Adicionalmente podrá utilizarse para cuantificar y determinar el avance del desarrollo del proyecto y como documentación de respaldo para posibles desarrollos futuros.

Dada la naturaleza del proyecto el presente documento es redactado por los autores del proyecto y alcanzará su aprobación cuando el director y los autores lleguen a un consenso sobre su contenido.

ALCANCE DEL SISTEMA

El prototipo de motor gráfico bidimensional *Beater2D* se presenta como una librería para facilitar el desarrollo de aplicaciones y juegos para la Web escritos en *JavaScript* que utilicen animaciones en dos dimensiones, simulaciones físicas y sonidos. Además está implementado de forma modular para permitir su futura expansión.

El prototipo está basado en las nuevas tecnologías provistas por *HTML5*, e inicialmente cuenta con los módulos que corresponden a: El núcleo central del motor gráfico, el cual provee un *API* para comunicar los otros módulos del motor gráfico con el navegador Web; el módulo de audio y sonido; el módulo de gráficos e imágenes; el modulo de eventos; y el módulo que integrará el motor físico *Box2DJS*.

El software se ocupa principalmente de manejar la instancia de *Canvas* para la cual ha sido asignado, gestionando así: la frecuencia de Cuadros Por Segundo (*Frames per Second*, *FPS*), el *Doble Buffer* gráfico y la representación de imágenes y animaciones en ellos, los distintos canales de sonido, la asignación de objetos al mundo físico.

Dentro del alcance del proyecto no se incluye:

- El diseño e implementación de interfaces para manejar interacción de los

usuarios con dispositivos de entrada cómo teclados, *mouses* o *joysticks*.

- El desarrollo de aplicaciones especificas.
- La capacidad de renderización de tipografías y fuentes dentro de las instancias del motor gráfico.
- El desarrollo de interfaces gráficas de usuarios (*GUI*) o asistentes generadores de código.

REQUISITOS FUNCIONALES

Casos de uso: Módulo Central

CMC-01: **Asignar instancia de Canvas**

Autor	Cristian Porras
Fecha	24/07/10
Descripción	Permite asignar una instancia del elemento Canvas al motor gráfico
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado la instancia de Canvas
Flujo Normal	1. El actor le indica al motor cual es la instancia en la cual debe trabajar 2. El sistema verifica la existencia y disponibilidad de dicha instancia de canvas 3. El sistema asigna el motor a la instancia provista
Poscondiciones	El motor ha sido correctamente inicializado

CMC-02: **Definir ciclo de FPS**

Autor	Cristian Porras
Fecha	24/07/10
Descripción	Permite definir la frecuencia de cambio de cuadros por segundo para una instancia del motor
Actores	Desarrollador de aplicaciones

Precondiciones	El actor debe haber creado una instancia del motor
Flujo Normal	1. El actor le indica al motor cual es la frecuencia a la cual debe trabajar 2. El sistema verifica que la frecuencia es un valor valido 3. El sistema asigna la frecuencia deseada en el motor
Flujo Alternativo	2. El sistema verifica que la frecuencia es un valor valido, si no es correcto se presenta un mensaje de error
Poscondiciones	El motor ha cambiado de frecuencia de FPS

CMC-03: **Iniciar ciclo de renderizado**

Autor	Cristian Porras
Fecha	24/07/10
Descripción	Permite iniciar el ciclo de renderizado en una instancia del motor
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado una instancia del motor
Flujo Normal	1. El actor le indica al motor que inicie el ciclo 2. El sistema inicia el ciclo de renderizado
Poscondiciones	El motor ha iniciado el ciclo de renderizado

CMC-04: **Pausar ciclo de renderizado**

Autor	Cristian Porras
Fecha	24/07/10
Descripción	Permite pausar el ciclo de renderizado en una instancia del motor
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber iniciado un ciclo de renderización
Flujo Normal	1. El actor le indica al motor que pause el ciclo 2. El sistema pausa el ciclo de renderizado
Poscondiciones	El motor ha pausado el ciclo de renderizado

CMC-05: **Reanudar ciclo de renderizado**

Autor	Cristian Porras
Fecha	24/07/10
Descripción	Permite reanudar el ciclo de renderizado en una instancia del motor
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber pausado un ciclo de renderización
Flujo Normal	1. El actor le indica al motor que reanude el ciclo 2. El sistema reanuda el ciclo de renderizado
Poscondiciones	El motor ha reanudado el ciclo de renderizado

CMC-06: **Añadir elementos a una instancia del motor**

Autor	Cristian Porras
Fecha	24/07/10
Descripción	Permite agregar objetos o elementos a una instancia del motor
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado el elemento y asignado sus propiedades visuales y físicas
Flujo Normal	1. El actor crea el elemento 2. El sistema añade el elemento
Poscondiciones	Se ha añadido un elemento a la instancia del motor

CMC-07: **Detectar Colisiones**

Autor	Cristian Porras
Fecha	24/07/10
Descripción	Permite detectar colisiones entre el entorno del juego y los objetos que interactúan en una instancia del motor
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado una instancia del motor, sin la opción de motor físico
Flujo Normal	1. El motor inicia el renderizado 2. Para cada uno de los frames el sistema verifica las interacciones entre los cuerpos 3. El sistema hace las correcciones necesarias a las posiciones de los elementos
Poscondiciones	El motor ha cambiado las posiciones de los elementos

CMC-08: **Precarga de archivos**

Autor	Cristian Porras
Fecha	24/07/10
Descripción	Permite pre-cargar las imágenes utilizadas por una instancia del motor
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado una instancia del motor
Flujo Normal	1. El actor le indica al motor gráfico que lea el documento XML donde se especifica las imágenes del juego 2. Se obtienen las rutas de las imágenes 3. Se cargan las imágenes
Poscondiciones	Las imágenes disponibles para su uso

CMC-09:	**Administrador de fondos de imágenes**
Autor	Cristian Porras
Fecha	24/07/10
Descripción	Permite manejar fondos de imágenes tipo Wall en los juegos. (Tecnica Parallax)
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe tener una instancia del motor corriendo
Flujo Normal	1. El actor le indica al motor gráfico cuales son las imágenes de fondo con su respectiva tasa de movimiento 2. Cada ciclo de tiempo se mueve el fondo del juego
Poscondiciones	El fondo es animado según la lógica desarrollada por el actor

Casos de Uso: Módulo de gráficos e imágenes

CMG-01:	**Cargar Imagen**
Autor	William Garcia
Fecha	24/07/10
Descripción	Carga una imagen a partir de un archivo
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado una instancia del motor
Flujo Normal	1. El actor le indica al motor la URL donde se encuentra la imagen (puede ser de tipo PNG, JPG o BMP) 2. El actor le indica al motor el estado inicial de la imagen 3. El sistema verifica la existencia de la imagen y los datos del estado inicial 4. El sistema carga la imagen para ser utilizada por el motor 5. El sistema aplica sobre la imagen sus propiedades iníciales
Flujo Alternativo	3. El sistema verifica la existencia de la imagen y los datos del estado inicial, si no son correctos se presenta un mensaje de error
Poscondiciones	La imagen ha sido cargada y se le han aplicado sus propiedades iníciales

CMG-02:	**Cargar Sprite**
Autor	William Garcia
Fecha	24/07/10
Descripción	Carga un sprite a partir de un archivo
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado una instancia del motor
Flujo Normal	1. El actor le indica al motor la URL donde se encuentra el sprite

	(puede ser de tipo PNG, JPG o BMP) 2. El actor le indica al motor el estado inicial del sprite 3. El actor le indica al motor el nombre del sprite 4. El sistema verifica la existencia de la imagen y los datos del estado inicial 5. El sistema carga el sprite para ser utilizado por el motor 6. El sistema aplica sobre el sprite sus propiedades iníciales
Flujo Alternativo	3. El sistema verifica la existencia del sprite y los datos del estado inicial, si no son correctos se presenta un mensaje de error
Poscondiciones	El sprite ha sido cargado y se está animando con sus propiedades iníciales

CMG-03: **Crear Cuadrilátero**

Autor	William Garcia
Fecha	24/07/10
Descripción	Permite crear un cuadrilátero utilizando las primitivas de canvas
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado una instancia del motor
Flujo Normal	1. El actor le indica al motor las propiedades del cuadrilátero a crear 2. El sistema crea el cuadrilátero y lo representa sobre el buffer gráfico
Poscondiciones	El cuadrilátero ha sido representado correctamente por el motor

CMG-04: **Crear Arco**

Autor	William Garcia
Fecha	24/07/10
Descripción	Permite crear un arco utilizando las primitivas de canvas
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado una instancia del motor
Flujo Normal	1. El actor le indica al motor las propiedades del arco a crear 2. El sistema crea el arco y lo representa sobre el buffer gráfico
Poscondiciones	El arco ha sido representado correctamente por el motor

CMG-05: **Crear Línea**

Autor	William Garcia
Fecha	24/07/10
Descripción	Permite crear una línea utilizando las primitivas de canvas
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado una instancia del motor
Flujo Normal	1. El actor le indica al motor las propiedades de la línea a crear 2. El sistema crea la línea y la representa sobre el buffer gráfico

Poscondiciones	La línea ha sido representada correctamente por el motor

CMG-06: **Crear Polígono**

Autor	William Garcia
Fecha	24/07/10
Descripción	Permite crear un polígono utilizando las primitivas de canvas
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado una instancia del motor
Flujo Normal	1. El actor le indica al motor las propiedades del polígono a crear 2. El sistema crea el polígono y lo representa sobre el buffer gráfico
Poscondiciones	El polígono ha sido representada correctamente por el motor

CMG-07: **Obtener Datos Gráfico**

Autor	William Garcia
Fecha	24/07/10
Descripción	Permite obtener las propiedades de algún elemento gráfico (Imagen, Sprite, Cuadrilátero, Arco, Línea o Polígono)
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado al menos una forma gráfica
Flujo Normal	1. El actor le indica al motor de que elemento desea obtener sus propiedades 2. El sistema le indica los datos del elemento al actor
Poscondiciones	El actor conoce los datos del elemento gráfico

CMG-08: **Modificar Datos Gráfico**

Autor	William Garcia
Fecha	24/07/10
Descripción	Permite modificar las propiedades de algún elemento gráfico (Imagen, Sprite, Cuadrilátero, Arco, Línea o Polígono)
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado al menos una forma gráfica
Flujo Normal	1. El actor le indica al motor de que elemento desea modificar sus propiedades 2. El sistema modifica las propiedades del elemento indicado
Poscondiciones	El elemento gráfico ha sido modificado

CMG-09: **Rotar Gráfico**

Autor	William Garcia
Fecha	24/07/10

Descripción	Permite rotar sobre su eje central un elemento gráfico (Imagen y Cuadrilátero)
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado al menos una forma gráfica
Flujo Normal	1. El actor le indica al motor el elemento gráfico que desea rotar, su nuevo ángulo 2. El sistema rota el elemento indicado
Poscondiciones	El elemento gráfico ha sido rotado

CMG-10: **Escalar Imágenes**

Autor	William Garcia
Fecha	24/07/10
Descripción	Permite escalar las Imágenes
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado un elemento de tipo imagen
Flujo Normal	1. El actor le indica al motor la imagen que desea escalar, su nueva escala 2. El sistema escala el elemento indicado
Poscondiciones	El elemento gráfico ha sido escalado

CMG-11: **Trasladar Gráfico**

Autor	William Garcia
Fecha	24/07/10
Descripción	Permite trasladar un elemento gráfico (Imagen, Sprite, Cuadrilátero, Arco, Línea o Polígono)
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado al menos una forma gráfica
Flujo Normal	1. El actor le indica al motor el elemento gráfico que desea trasladar su nueva posición 2. El sistema traslada el elemento indicado
Poscondiciones	El elemento gráfico ha sido trasladado

Casos de Uso: Módulo de audio y sonido

CMA-01: **Cargar Sonido**

Autor	Cristian Porras
Fecha	25/07/10
Descripción	Carga un sonido a partir de un archivo

Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber creado una instancia del motor
Flujo Normal	1. El actor le indica al motor la URL donde se encuentra el sonido (en formato OGG) 2. El actor le indica al motor el estado inicial del sonido 3. El sistema verifica la existencia del archivo 4. El sistema carga el sonido para ser utilizado por el motor
Poscondiciones	El sonido ha sido cargado

CMA-02: **Reproducir Sonido**

Autor	Cristian Porras
Fecha	25/07/10
Descripción	Reproduce un sonido cargado por el motor
Actores	Desarrollador de aplicaciones
Precondiciones	El actor debe haber cargado un sonido al sistema
Flujo Normal	1. El actor le indica al motor el sonido que quiere reproducir 2. El actor le indica al motor el estado inicial del sonido 3. El sistema reproduce el sonido
Poscondiciones	El sonido ha sido reproducido

CMA-03: **Pausar Sonido**

Autor	Cristian Porras
Fecha	25/07/10
Descripción	Pausa la reproducción de un sonido
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe estar reproduciendo algún sonido
Flujo Normal	1. El actor le indica al motor el sonido que se quiere pausar 2. El sistema pausa la reproducción sonido
Poscondiciones	La reproducción del sonido ha sido pausada

CMA-04: **Reanudar Sonido**

Autor	Cristian Porras
Fecha	25/07/10
Descripción	Reanuda la reproducción de un sonido que está pausado
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe tener en estado de pausa algún sonido
Flujo Normal	1. El actor le indica al motor el sonido que quiere resumir 2. El sistema verifica la existencia y estado del sonido 3. El sistema reanuda la reproducción el sonido

Poscondiciones	La reproducción del sonido ha sido reanudada

CMA-05: **Detener Sonido**

Autor	Cristian Porras
Fecha	25/07/10
Descripción	Detiene la reproducción de un sonido
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe tener cargado al menos un sonido
Flujo Normal	1. El actor le indica al motor el sonido que quiere detener 2. El sistema detiene la reproducción del sonido
Poscondiciones	El sonido ha sido detenido

CMA-06: **Cambiar Volumen Sonido**

Autor	Cristian Porras
Fecha	25/07/10
Descripción	Permite cambiar el volumen de reproducción de un sonido
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe tener cargado al menos un sonido
Flujo Normal	1. El actor le indica al motor el sonido al cual le modificara el volumen 2. El actor indica el nuevo volumen de reproducción 3. El sistema verifica la existencia y estado del sonido 4. El sistema cambia el volumen de reproducción del sonido
Poscondiciones	El volumen de reproducción del sonido ha sido modificado

Casos de Uso: Módulo de eventos

CME-01: **Capturar Gesto Teclado**

Autor	William Garcia
Fecha	26/07/10
Descripción	Permite capturar acciones del teclado
Actores	Usuario de aplicaciones desarrolladas con Beater2D
Precondiciones	Se debe tener una instancia del motor corriendo
Flujo Normal	1. El actor presiona alguna tecla en su teclado

	2. El actor mantiene presionada la tecla por algún tiempo 3. El actor libera la tecla presionada 4. El sistema agrega la información de la tecla presionada a un vector de eventos
Poscondiciones	La información de la tecla presionada está en el vector de eventos

CME-02: **Capturar Gesto Mouse**

Autor	William Garcia
Fecha	26/07/10
Descripción	Permite capturar acciones del mouse
Actores	Usuario de aplicaciones desarrolladas con Beater2D
Precondiciones	Se debe tener una instancia del motor corriendo
Flujo Normal	1. El actor presiona algún botón en su mouse 2. El actor mantiene presionado el botón por algún tiempo y arrastra el mouse 3. El actor libera el botón presionado 4. El sistema agrega la información del gesto del mouse a un vector de eventos
Poscondiciones	La información de la tecla presionada está en el vector de eventos

CME-03: **Obtener Evento**

Autor	William Garcia
Fecha	26/07/10
Descripción	Obtiene gestos almacenados en el vector de eventos
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe tener al menos un evento almacenado
Flujo Normal	1. El actor pide al sistema el siguiente evento almacenado en el vector 2. El sistema entrega la información solicitada al actor
Poscondiciones	El actor tiene acceso al siguiente evento realizado por un usuario

Casos de Uso: Módulo físico

CMF-01: **Crear Mundo Físico**

Autor	William Garcia
Fecha	26/07/10
Descripción	Permite crear una instancia del mundo físico de Box2DJS dentro del motor gráfico
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe tener una instancia del motor corriendo
Flujo Normal	1. El actor le indica al motor gráfico que desea crear una instancia del motor físico 2. El sistema crea la instancia del motor físico dentro del motor gráfico
Poscondiciones	La instancia del motor físico debe ha sido creada

CMF-02: **Iniciar Simulación Mundo Físico**

Autor	William Garcia
Fecha	26/07/10
Descripción	Permite iniciar la simulación de las iteraciones de física de cuerpo rígido dentro de Beater2D
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe tener una instancia del motor físico asignada al motor gráfico
Flujo Normal	1. El actor le indica al motor que desea iniciar la simulación física 2. El sistema realiza las simulaciones físicas con los objetos existentes
Poscondiciones	El motor se encuentra realizando las simulaciones físicas

CMF-03: **Parar Simulación Mundo Físico**

Autor	William Garcia
Fecha	26/07/10
Descripción	Permite detener la simulación de las iteraciones de física de cuerpo rígido dentro del motor
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe tener una instancia del motor físico asignada al motor gráfico
Flujo Normal	1. El actor le indica al motor que desea detener la simulación física 2. El sistema detiene las simulaciones físicas
Poscondiciones	El motor ha dejado de realizar las simulaciones físicas

CMF-04:	**Agregar Objeto Mundo Físico**
Autor	William Garcia
Fecha	26/07/10
Descripción	Permite agregar un objeto al mundo físico de Box2DJS
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe tener una instancia del motor físico asignada al motor gráfico
Flujo Normal	1. El actor le indica al sistema las propiedades físicas del nuevo objeto y el objeto gráfico al cual está relacionado 2. El sistema crea el objeto en el mundo físico y lo relaciona con el mundo gráfico
Poscondiciones	Se ha agregado un nuevo objeto al mundo físico para ser simulado

CMF-05:	**Modificar Propiedades Mundo Físico**
Autor	William Garcia
Fecha	26/07/10
Descripción	Permite modificar los parámetros y propiedades de una instancia del motor físico Box2DJS
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe tener una instancia del motor físico creada
Flujo Normal	1. El actor le indica al sistema las nuevas propiedades del mundo físico 2. El sistema modifica los parámetros de simulación del motor físico
Poscondiciones	La instancia del motor físico ha sido modificada

CMF-06:	**Sincronizar mundo Físico con grafico**
Autor	William Garcia
Fecha	26/07/10
Descripción	Permite plasmar las propiedades de los cuerpos en el mundo físico a sus homólogos en el mundo grafico
Actores	Desarrollador de aplicaciones
Precondiciones	Se debe tener una instancia del motor físico creada
Flujo Normal	1. El sistema por cada ciclo realiza la simulación de los cuerpos físicos 2. Envía los datos del mundo físico al grafico
Poscondiciones	La posición de los cuerpos gráficos ha sido actualizada

REQUISITOS NO FUNCIONALES

A continuación se listan todos los requisitos no funcionales que determinan la construcción y el diseño del sistema.

Requisitos de la interfaz de usuario

Debido a que el sistema desarrollado es una librería para desarrollo de aplicaciones, no se debe cumplir algún tipo de requerimiento en cuanto a interfaz de usuario.

Requisitos de la interfaz de hardware

El sistema desarrollado está pensado para ser utilizado en el desarrollo de aplicaciones web, por lo cual no se requiere una capacidad extraordinaria de procesamiento, o hardware externo a la computadora para el uso normal del sistema. Para el adecuado uso del aplicaciones implementadas sobre *Beater2D* sólo se requiere las capacidades mínimas de conexión a Internet banda ancha y que el dispositivo utilizado pueda ejecutar cualquier navegador moderno.

Requisitos de la interfaz de software

El desarrollo de aplicaciones utilizando *Beater2D* no requiere el uso de software adicional, *Beater2D* es auto-contenido e incorpora todas las librerías utilizadas en su construcción.

Para ejecutar aplicaciones desarrolladas sobre *Beater2D* se recomienda el uso de *Mozilla Firefox* o *Google Chrome*, disponibles para la mayoría de sistemas operativos de escritorio; *GNU/Linux*, *Mac OSX*, *Microsoft Windows*, entre otros.

Requisitos de diseño y desarrollo

El prototipo de motor gráfico debe ser construido íntegramente utilizando software libre, las librerías que utilice deben estar bajo licencias aprobadas por la *Free Software Foundation*, para garantizar su libre uso y distribución a futuro.

Se debe construir documentación clara, a todo nivel, del sistema para el desarrollo de nuevas funciones y módulos, facilitar las labores de mantenimiento, y para el desarrollo de aplicaciones basadas en *Beater2D*.

Anexo D: Manual de referencia del API de Beater2D

El presenta manual de referencia tiene como objetivo servir como guía para los desarrolladores de aplicaciones que deseen implementar sus proyectos de software sobre las librerías de *Beater2D*, consta de la definición de las clases, su descripción, parámetros y un ejemplo del uso de cada una.

Class $.Arc

La clase Arc Permite dibujar arcos, estos arcos están definidos por un Angulo inicial y final que determina la abertura del arco, estos ángulos se deben dar en radianes.

Los parámetros al dibujar arcos son los siguientes:

- fillColor: Color de relleno del arco.
- stroke: Booleano que indica si el arco se rellena o solo se dibujan el borde.
- strokeColor: Color del borde del arco.
- lineWidth: Ancho de la línea del borde.
- lineJoin: Forma con la que se unirán las líneas que dibujan el arco.
- miterLimit: Límite de tamaño para la punta de las esquinas del arco.
- Anticlockwise: Determina la orientación del giro del arco horario o antihorario.

Ejemplo:

```
1. lienzo = new $.Beater("lienzo",320,600);
2. Arco = new $.Arc(100, 100, 70, 0, 3.1416, lienzo.buffer.context);
3. Arco.apply();
```

Salida:

Clase definida en: graphics/graphics.shape.js

Class $.Backgroundimage

Es una clase que utilizando la clase Image facilita el manejo de los fondos del juego.

Ejemplo:

```
1. juego = new $.Beater("lienzo",320,600);
2. fondo = new $.Backgroundimage("fondo.jpg",juego.buffer.ctxbuffer, 0, 0, 0,
   600, 320, 0.1);
3. juego.addBackgroundimage(fondo);
4. juego.Start();
```

Salida:

Clase definida en: utils/utils.backgroundimage.js

Class $.Beater

Clase principal que sincroniza el funcionamiento de los módulos del motor grafico convirtiéndolo asi en un motor para juegos en 2D, En esta clase se definen los Frames por segundo con los que se ejecutara el juego, se agregan los objetos que interactuaran en el juego, etc.

Desde esta clase se inicia y pausa el juego. Se analizan las interacciones entre los objetos y el ambiente del juego.

Ejemplo:

1. juego = new $.Beater("lienzo",320,600);
2. fondo = new $.Backgroundimage("fondo.jpg",juego.buffer.ctxbuffer, 0, 0, 0, 600, 320, 0.1);
3. juego.addBackgroundimage(fondo);
4. juego.floor.addimage("default", "frogatto/images/mundo/piso1-1.png");

```
5.  juego.floor.addblock("default", 300 , false );
6.  juego.floor.addblock("default", 280 , false );
7.  juego.floor.addblock("default", 260 , false );
8.  juego.floor.addblock("default", 240 , false );
9.  juego.floor.addblock("default", 220 , false );
10. juego.floor.addblock("default", 200 , false );
11. juego.floor.addblock("default", 180 , false );
12. juego.floor.addblock("default", 200 , false );
13. juego.floor.addblock("default", 220 , false );
14. juego.floor.addblock("default", 240 , false );
15. juego.floor.addblock("default", 260 , false );
16. juego.floor.addblock("default", 280 , false );
17. juego.floor.addblock("default", 300 , false );
18. juego.Start();
```

Salida

Clase definida en: jquery.beater.js

Class $.Bjsound

Clase que permite la reproducir sonidos, los archivos de audio deben ser ogg, mp3 o wav.

Ejemplo:

1. Sonido = new $.Bjsound(source.ogg, 0.5);
2. Sonido.play();

Clase definida en: sound/sound.js

Class $.Circle

Esta clase tiene con función dibujar Circunferencias.

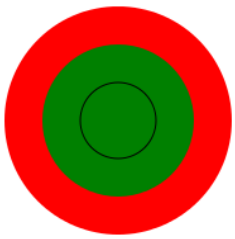
Los parámetros al dibujar círculos son los siguientes:

- fillColor: Color de relleno del círculo.
- stroke: Booleano que indica si el círculo se rellena o de lo contrario solo se traza el borde.
- strokeColor: Color del borde del círculo.
- lineWidth: Ancho de la línea del borde.
- lineJoin: Forma con la que se unirán las líneas que dibujan el circulo.
- miterLimit: Límite de tamaño para la punta de las esquinas del círculo.

Ejemplo:

```
1. beater = new $.Beater("lienzo",320,600);
2. circulo = new $.Circle (130, 130, 90, beater.buffer.context);
3. circulo.custom={fillColor : "red"};
4. circulo.apply();
5. circulo2 = new $.Circle (130, 130, 60, beater.buffer.context);
6. circulo2.custom={fillColor : "green"};
7. circulo2.apply();
8. circulo3 = new $.Circle (130, 130, 30, beater.buffer.context);
9. circulo3.custom={stroke : true};
10. circulo3.apply();
```

Salida



Clase definida en: graphics.shape.js

Class $.Eventos

Clase que permite el manejo de los eventos de teclado Los vectores que obtienen los eventos son desocupados cada ciclo de tiempo.

Ejemplo:

```
1. Eventos = new $.Eventos();
```

Clase definida en: event/event.input.js

Class $.Image

Image es una clase base para importar y exportar imágenes. Los formatos de imágenes soportados son {gif, png, jpeg bmp}.

Los parametros dal dibujar imágenes son los siguientes:

- dh: Altura de la imagen destino que se plasmara en el lienzo.
- dw: Ancho de la imagen destino que se plasmara en el lienzo.
- sh: Altura de la imagen fuente que se captura para ser plasmada en el lienzo.
- sw: Ancho de la imagen fuente que se captura para ser plasmada en el lienzo.
- sx: Posición inicial en X donde se captura la imagen fuente, para plasmarla en el lienzo destino.
- sy: Posición inicial en y donde se captura la imagen fuente, para plasmarla en el lienzo destino.
- resize: Propiedad que permite maximizar o minimizar las imágenes.

Ejemplo:

```
1. beater = new $.Beater("lienzo",320,600);
2. imagen1 = new $.Image("fondo.jpg", beater.buffer.context);
3. imagen1.apply();
4. imagen1.custom = {resize:50}
5. imagen1.y = 180;
```

6. imagen1.apply();
7. imagen1.angle = 3.14;
8. imagen1.x = 180;
9. imagen1.apply();

Salida

Clase definida en: graphics/graphics.image.js

Class $.Line

La clase Line dibuja líneas en el lienzo.

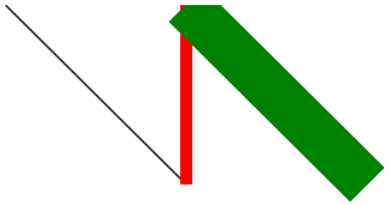
Los parámetros al dibujar líneas son las siguientes:

- lineColor: Color de la línea.
- lineWidth: Ancho de la línea.
- lineJoin: Forma con la que se unirán las líneas dibujadas.
- miterLimit: Límite de tamaño para la punta de interceptación entre lineas.

Ejemplo:

```
1. beater = new $.Beater("lienzo",320,600);
2. Linea = new $.Line (0, 0, 150, 150, beater.buffer.context);
3. Linea.apply();
4. Linea2 = new $.Line (150, 150, 150, 0, beater.buffer.context);
5. Linea2.custom = {lineWidth : 10, lineColor : "red"}
6. Linea2.apply();
7. Linea3 = new $.Line (150, 0 , 300, 150, beater.buffer.context);
8. Linea3.custom = {lineWidth : 40, lineColor : "green"}
9. Linea3.apply();
```

Salida:



Clase definida en: graphics/graphics.shape.js

Class $.Polygon

La clase Polygon dibuja Polígonos en el lienzo solo se necesita especificar las coordenadas de las aristas.

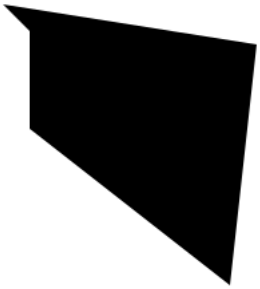
Los parámetros para los polígonos son los siguientes:

1. fillColor: Color de relleno del polígono.
2. stroke: Booleano que indica si el polígono se rellena o de lo contrario dibuja el entorno del polígono
3. strokeColor: Color del borde del polígono.
4. lineWidth: Ancho contorno del borde.
5. lineJoin: Forma con la que se unirán las líneas que dibujan el polígono.
6. miterLimit: Límite de tamaño para la punta de las esquinas del polígono.

Ejemplo:

```
1. beater = new $.Beater("lienzo",320,600);
2. poligono = new $.Polygon(0, 0, [], beater.buffer.context);
3. poligono.addpoint(10, 10);
4. poligono.addpoint(200, 40);
5. poligono.addpoint(180, 220);
6. poligono.addpoint(30, 103);
7. poligono.addpoint(30, 30);
8. poligono.apply();
```

Salida:



Clase definida en: graphics/graphics.shape.js

Class $.Rect

La clase Rect sirve para dibujar rectángulos utilizando las primitivas de Canvas El rectángulo se dibuja comenzando en la posición (x,y) del lienzo y las dimensiones son de (width x heigth).

Las opciones del rectángulo son las siguientes:

1. fillColor: Color de relleno del rectángulo.
2. clear: Booleano que indica si el rectángulo actúa como borrador.
3. stroke: Booleano que indica si el rectángulo se rellena o solo se dibuja el contorno.
4. strokeColor: Color del borde del rectángulo.
5. lineWidth: Ancho de la línea del borde.
6. lineJoin: Forma con la que se unirán las líneas que dibujan el rectángulo.
7. miterLimit: Límite de tamaño para la punta de las esquinas del rectángulo.

Ejemplo:

1. beater = new $.Beater("lienzo",320,600);
2. rectangulo = new $.Rect(30, 30, 160, 160, 0, 1, beater.buffer.context);
3. rectangulo.apply();
4. *rectangulo2 = new $.Rect(60, 60, 100, 100, 0, 1, beater.buffer.context*);
5. rectangulo2.custom ={fillColor : "red"};
6. rectangulo2.apply();

7. rectangulo3 = new $.Rect(80, 80, 60, 60, 0, 1, beater.buffer.context);
8. rectangulo3.custom ={clear : true};
9. rectangulo3.apply();

Salida

Clase definida en: graphics/graphics.shape.js

Anexo E: Lista de chequeo utilizada en la fase de pruebas

Lista de chequeo de funcionalidades de Beater2D				
Responsable:			**Fecha:**	
Característica	**TF**	**FL**	**NF**	**Observaciones**
Asignar instancia de Canvas Definir ciclo de FPS Iniciar ciclo de renderizado Pausar ciclo de renderizado Reanudar ciclo de renderizado Añadir elementos a una instancia del motor Detectar Colisiones Precarga de archivos Administrador de fondos de imágenes Cargar Imagen Cargar Sprite Crear Cuadrilátero Crear Arco Crear Línea Crear Polígono Obtener Datos Gráfico Modificar Datos Gráfico Rotar Gráfico Escalar Imágenes Trasladar Gráfico Cargar Sonido Reproducir Sonido Pausar Sonido Reanudar Sonido Detener Sonido Cambiar Volumen Sonido Capturar Gesto Teclado Capturar Gesto Teclado Obtener Evento Crear Mundo Físico Iniciar Simulación Mundo Físico Parar Simulación Mundo Físico Agregar Objeto Mundo Físico Modificar Propiedades Mundo Físico Sincronizar mundo Físico con grafico				
Totales				Condiciones aceptables: Condiciones no aceptables:

TF: Totalmente funcional **FL:** Funcionalidad limitada **NF:** No funcional