Universitat Politècnica de València
Estructura responsable del grado al que opto

# Videojuego de construcción mediante piezas ensamblables en navegador

Trabajo Final de Máster

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital.

Autor: Victor Gascó Ortiz
Director: Roberto Agustín Vivó Hernando

2014/2015

# Resumen

Los gráficos 3D en tiempo real en navegador web antiguamente eran un quebradero de cabeza, productos como Virtools, Unity o Stage3D permitían su ejecución, pero eran soluciones cerradas, APIs mantenidas por las correspondientes empresas, y por lo tanto dependían de éstas para saber su continuidad y evolución. Además, estos productos necesitaban de plugins, siendo en algunos casos foco de problema y para muchas personas, demasiado invasivo.

En 2006, hizo su aparición en los gráficos 3D en navegador web, **WebGL** [1], ésta tecnología nos permite mostrar gráficos acelerados por GPU en nuestro navegador, sin necesidad de instalar plug-in alguno, el único requisito es que la plataforma soporte OpenGL ES 2.0, de hecho muchos consideran que WebGL es una API de OpenGL ES para **JavaScript** [2].

Debido que es accesible por todo el mundo y a que requiere un desarrollo de bajo nivel, por lo que añade flexibilidad al proceso de desarrollo, WebGL es una librería muy popular entre los desarrolladores de gráficos 3D, sin embargo, se necesita mucho tiempo para aprender este lenguaje, por lo que generalmente se utilizan librerías de más alto nivel, como por ejemplo BabylonJS, TDL, O3D y **Three.js** [3], ésta última es la que utilizamos en este proyecto.

Three.js utiliza el elemento canvas de **HTML5** [4] o WebGL, para mostrar gráficos animados por ordenador en 3D, es una librería de alto nivel, lo que facilita su uso y aplicación.

Esto supuso un cambio en los videojuegos de navegador, los cuales se podían ejecutar con independencia de la plataforma, debido a que la gran mayoría necesitaba los ya mencionados plugins, sin mencionar que muchos juegos dependen de diferentes plugins. Por lo tanto esta librería aporta una gran comodidad al usuario, que tan solo ha de ir a la página web y jugar.

Utilizando la librería Three.js y con conocimientos web, HTML5, CSS3 y sobre todo Javascript, en este proyecto se pretende crear un videojuego de construcción en el propio navegador web, mostrando así la potencia de WebGL, llegando a tener un producto final que nos permita jugar la mayoría de ordenadores sin necesidad de instalar ningún plug-in ni elemento.

Para facilitar el proceso de desarrollo hemos utilizado la librería **jQuery** [5], la cual facilita la el trato con HTML. Adicionalmente también se han utilizado multitud de librerías de apoyo en Three.js, tanto para control de la cámara como para control de posibles errores, de estas librerías hablaremos más a fondo en el capítulo 4: Diseño de la Herramienta.

Como resultado de este proyecto se muestra un videojuego como producto final, con funcionalidad comprobada en la mayoría de navegadores, y pudiendo comercializarse sin problemas.

**Palabras clave:** WebGL, Three.js, JavaScript, videojuegos, HTML.

# Abstract

The 3D graphics in real time on web browser were once a headache, products like Virtools, Unity or Stage3D allowed their execution, but they were closed solutions, APIs maintained by the respective companies, and therefore dependent on them to know their continuity and development.

Furthermore, these products needed plugins, being in some cases a focus of problem and, for many people, too invasive.

In 2006, appeared in 3D graphics on web browser, WebGL [1], this technology allows us to use accelerated graphics by GPU in our browser, without installing any plug-in, the only requirement is that the platform supports OpenGL 2.0, in fact many think that WebGL is an API for OpenGL ES API written in JavaScript [2].

Because it is accessible by everyone because and it requires a low-level development, which adds flexibility to the development process, WebGL is a very popular library among developers of 3D graphics, however, it takes a long time to learn this language, so generally higher level libraries are used, such as BabylonJS, TDL, O3D and Three.js [3], the last one is which we are going to use in this project.

Three.js uses the HTML5 canvas element [4] or WebGL to display animated 3D computer graphics, is a high-level library, which facilitates its use and application.

This was a change in the web browser games, which could run regardless of the platform, because the majority were needed the mentioned plugins. Therefore this library provides great convenience to the user, who just have go to the website and start playing.

Using the Three.js library, and with web knowledge, HTML5, CSS3 and Javascript especially, this project aims to create a video game built on the web browser, showing the power of WebGL, getting to have a final product which allow us to play in most computers without installing any plug-in or element.

To make easier the development process, we used jQuery [5] library, which facilitates the HTML management. In addition many support libraries of Three.js were used, for camera control as for error control, we will analyze some of these.

As a result of this project, a video game as a final product is shown, which runs properly in most browsers and can be commercialized without problems.



**Keywords:** WebGL, Three.js, JavaScript, videogames, HTML.

# Índice general

# Índice de figuras

# Capítulo 1. Introducción

## Motivación

En los últimos años, debido a la gran cantidad de contenido de internet y a las nuevas tecnologías, ha cambiado las principales actividades realizadas por los usuarios en Internet, pasando de una mera búsqueda y la navegación por la información almacenada a una interacción directa con otros usuarios. Los usuarios han pasado de ser consumidores de información a ser los productores reales (lo que se conoce como la transición de la Web 1.0 a la Web 2.0).

Debido a este fenómeno cada vez son más los productos que encontramos en la web habiendo pasado de su versión de escritorio a una versión online. Los videojuegos no son una excepción a este fenómeno, desde hace tiempo encontramos videojuegos en navegador web e incluso webs dedicadas exclusivamente a recolectar videojuegos que se ejecutan en navegador. Estos videojuegos no suelen requerir instalación alguna, pero en muchos casos requieren de algún plugin adicional, la gran mayoría eran desarrollados en Adobe Flash [2].

Por ello, mucha gente se siente incómoda utilizando esta tecnología, adicionalmente al hecho de tener que instalar estos plugins, se encuentran motivos por los que puede fallar: por la versión del navegador, la versión de Java instalada, etc. Todo ello finalmente crea un problema de seguridad, y debido a esto los plugins suelen requerir actualizaciones constantes para solventar este tipo de problemas.

Otro dato significativo de la limitación de esta tecnología es el hecho de que los videojuegos más jugados en navegador son los de estrategia en tiempo real, y en muchos casos carentes de gráficos 3D.

Finalmente otro de los problemas que encontramos es la movilidad, puesto que la mayoría de tablets y Smartphones no admiten instalaciones de plugins y complementos.

Por todo esto en cuanto salió la tecnología WebGL, que utilizaba la potencia de la GPU local, tan y como hacen los videojuegos en local, que requieren instalación, su uso se fue extendiendo, hoy día son más los dispositivos compatibles con WebGL que con Flash.

Como principales ventajas de esta tecnología encontramos, que al ser lenguaje interpretado no necesitas compilar tu código antes de renderizarlo, la compatibilidad existente con distintos navegadores y plataformas, la gran compatibilidad que presenta con los elementos HTML y la ya mencionada ventaja de ser gráficos 3D acelerados por hardware (CPU y GPU).

El presente proyecto pretende desarrollar un videojuego interactivo de construcción 3D en navegador web, utilizando la librería Three.js [6], basada en WebGL. Para ello se va a aprovechar la posibilidad que brinda esta librería para dotar al sistema de un control muy intuitivo y sencillo para el usuario, mediante desplazamiento del ratón y teclas de acceso directo. Adicionalmente aprovechar las interactividad que nos brinda el hecho de trabajar en un entorno 3D y dándole un poco de realismo a la escena mediante luces y sombras.

Para desarrollar la herramienta se ha hecho uso de HTML5 y mediante JavaScript y con CSS3, se ha creado un menú que facilita enormemente la interacción del usuario con esta, proporcionándole tanto opciones para interactuar en el propio *canvas,* como mostrándole información de ayuda y de acceso rápido de la herramienta. Básicamente se ha creado un sencillo juego de construcción basado en una variedad ilimitada de cubos (puesto que el usuario puede añadir los suyos propios), pudiendo de esta forma crear todo tipo de figuras de una forma sencilla y rápida.

Estas tecnología son bastante jóvenes y aún podemos sacarles mucho partido, especialmente debido al uso que damos los usuarios de hoy día a internet, del que proviene casi totalidad de nuestro entretenimiento e información cuando estamos en un ordenador.

Con esto, en el presente proyecto se pretende mostrar la potencia de las jóvenes tecnologías de Three.js y WebGL como herramientas 3D y desarrollar un producto final y comercial, es decir, un videojuego en un navegador web, al que siempre podríamos estar añadiendo funcionalidades, tal y como se hacen en la gran mayoría de videojuegos con acceso online, dando pie a futura salida profesional y mejora del propio proyecto.

## Objetivos

El objetivo de este Proyecto de Fin de Máster es el de realizar un videojuego de construcción 3D para navegador web, basado en la tecnología WebGL, su posterior evaluación y validación, comprobando que éste funciona tal y como se desea.

Por lo tanto, la finalidad no es otra que la de cualquier videojuego, es decir, entretener al usuario, pero en este caso la plataforma el navegador web, la consola cualquier ordenador que podamos tener en casa y la tecnología, gráficos 3D en navegador acelerados por hardware, lo cuál es la principal novedad.

Los requerimientos de este proyecto parten de las existentes librerías para WebGL, en nuestro caso las que corresponden a Three.js, importando estas a nuestro proyecto e implantándolas en conjunto para que funcionen correctamente. Posteriormente se ha de programar el funcionamiento de todo el entorno canvas, y acoplarlo a la interfaz HTML, esta es la parte central del proyecto.

El objetivo principal podemos dividirlo en subobjetivos, facilitando el proceso de desarrollo y la organización del proyecto, además de permitir controlar mejor su temporalización. Estos subobjetivos son:

- Establecer los requisitos básicos y funcionalidad necesaria, así como los elementos con los que el usuario interactuará.

- Determinar los diferentes controles que el usuario utilizará en el proyecto y cómo estos funcionarán. Es decir, decidir las opciones básicas que el usuario tendrá y aplicarlas a junto con los controles de la cámara en nuestro canvas.

- Conseguir crear cubos dinámicamente, partiendo de los ya creados y que su posicionamiento en el canvas sea intuitivo y sencillo.

- Añadir opciones básicas para el usuario, edición de cubos creados y adición de cubos personalizados.

- Crear la interfaz de usuario, con opciones adicionales, cambios de textura de los cubos, posicionamiento de la cámara, capturas de pantalla, menú de ayuda, ...

- Diseño de la red social basada en un modelo de confianza entre los agentes, favoreciendo así las recomendaciones recibidas desde amigos más fiables.

- Implementación final de la herramienta con todas las funcionalidades y creación de la página web.

- Evaluación del proyecto con sus deferentes funcionalidades en diferentes navegadores web.

## Estructura de la memoria

La presente memoria se ha dividido varios capítulos, la estructura global de la memoria es la siguiente:

- En el capítulo 1 se hecho una breve introducción de este proyecto y se han comentado los objetivos a tratar, junto con una descripción detallada de los subobjetivos, con el fin de conseguir una versión operativa.

- En el capítulo 2 se tratan los principales problemas que tratamos en este proyecto, en que fase de evolución están ahora las tecnologías utilizadas en este (Three.js, HTML, JavaScript,…), y que mejoras aporta el presente proyecto respecto a otros.

- En el capítulo 3 se realiza el primer acercamiento a nuestro videojuego, público objetivo, diseño inicial y requisitos,…

- En el capítulo 4 se muestra el diseño de la herramienta, se comentan las librerías utilizadas y su funcionamiento, métodos de interacción con el usuario, interfaz, …

- En el capítulo 5 se realizan y comentan, pruebas de ejecución llevadas a cabo para comprobar el correcto funcionamiento del producto.

- Finalmente, en el capítulo 6 se muestran las conclusiones, comentando además la posibilidad de trabajos futuros basándose en este proyecto.

Como recomendaciones para lectores con algo de experiencia en el mundo de gráficos 3D por ordenador, si no han utilizado Three.js se les recomienda saltar directamente a este apartado en el capítulo 3, en caso de que también tengan conocimientos de esta librería, directamente saltar al capítulo 4, en el que se comienza a hablar de la propia herramienta.

# Capítulo 2. Antecedentes

## Descripción del problema

Tal y como se ha dicho anteriormente, en términos sencillos, lo que se va a desarrollar en este proyecto es una videojuego, del estilo de los que jugamos en otro tipo de consolas, es decir, en 3D peo que en este caso se jugará desde nuestro navegador web (Google Chrome, p.e), sin necesidad de realizar ninguna instalación. ¿A qué nos referimos con esto? Si habéis comprado alguna vez un videojuego para el ordenador, sabréis que necesitáis, como mínimo, instalarlo. Sin embargo, el videojuego desarrollado en este proyecto no requiere nada de esto, simplemente abres el navegador, vas a la dirección web del videojuego, y listo.

Rápido, sencillo y fácil de utilizar, en grandes rasgos este es el problema que solucionamos al utilizar este tipo de tecnología, mostramos un producto, que no necesita de tantos preparativos como anteriormente se venía haciendo, evitamos descargas e instalaciones innecesarias y, en algunos casos, demasiado engorrosas.

## Estado del arte

En el contexto de este proyecto vamos a comenzar hablando de los gráficos 3D en navegador web, como se ha dicho anteriormente, antiguamente los gráficos no estaban integrados directamente en nuestros navegadores web y había varias tecnologías, entre las que destacaba Adobe Flash [2], que dependían de software externo, lo que en muchos casos provocaba que el usuario abandonase la página.

Esta tecnología estaba un poco estancada en cuanto a confianza por parte del usuario se refiere, además de resultar bastante molesta, sin embargo con la aparición de WebGL, esta tendencia cambió, hasta el punto de que, tras el lanzamiento de Windows 8.1 con Internet Explorer 11, todos los navegadores pudieran hacer uso de esta tecnología sin ninguna instalación adicional.

WebGL muestra gráficos 3D de alta calidad y trabaja de tal forma que se puede interactuar con ellos, todo ello se muestra dentro del elemento canvas de HTML5, por lo tanto, vamos a comentar por encima la tecnología HTML5 y, por supuesto CSS3, ya que ambos suelen ir de la mano.

## *HTML5 y CSS3*

HTML5 es la quinta revisión de HTML, publicada en octubre de 2014 y que debido a la cantidad de características y etiquetas, no es reconocido por antiguos navegadores.

Esta versión de HTML contiene un amplio conjunto de tecnologías que permite a los sitios Web y a las aplicaciones ser más diversas y de gran alcance. Este conjunto de tecnologías popularmente se cono como HTML5 y *amigos*.

HTML5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos, añadiendo significado semántico a algunas etiquetas, como por ejemplo <footer>.

Los diferentes y variados recursos de HTML5 se pueden clasificar en varios grupos según su función:

- Semántica, la cual nos permite describir con mayor precisión cuál es el contenido de dicho contenedor.
- Conectividad, nos permite comunicarnos con el servidor de varias formas innovadoras.
- Fuera de línea y almacenamiento, gracias a lo que podemos almacenar datos, localmente, en el lado del cliente y operar fuera de línea de manera eficiente.
- Multimedia, nos otorga un excelente soporte para utilizar contenido multimedia, como son audio y video, nativamente.
- Gráficos y efectos 2D/3D, el apartado más novedoso que nos proporciona una amplia gama de nuevas características que se ocupan de los gráficos en la web como lo son el lienzo 2D, **WebGL**, SVG, etc.
- Rendimiento e Integración, optimización de la velocidad y uso del hardware.
- Acceso al dispositivo, nos proporciona APIs para el uso de varios componentes internos de entrada y salida de nuestro dispositivo.
- CSS, nos ofrece una nueva gran variedad de opciones para la sofisticación del diseño.

Respecto a CSS3, es el recurso utilizado de HTML5, gracias al cual se ha dado vistosidad a la interfaz del proyecto.

En primer lugar vamos a comentar de forma resumida que es CSS, puesto que la gran mayoría de personas lo conoce bastante a fondo. CSS es un lenguaje usado para definir y crear la presentación de un documento estructurado escrito en HTML o XML, básicamente la idea tras el desarrollo de CSS es separar la estructura de un documento de su presentación.

CSS tiene una sintaxis gracias a la que, utilizando palabras clave en inglés cambiamos los atributos de varias propiedades del estilo, en definitiva es una hoja de estilo que se compone de una lista de reglas las cuales consiste en uno o varios selectores con los estilos a aplicar para cada de los elementos que cumplan con el selector. Esta hoja de estilo se le de arriba abajo y se sobrescriben los estilos conflictivos, de todo esto proviene el nombre de CSS (siglas *en inglés de cascading style sheets*).

Finalmente en cuanto a CSS3, la última versión de este lenguaje, la primera diferencia que encontramos respecto a las versiones anteriores es que está dividida en módulos separados, y en cada uno de estos módulos se han añadido las nuevas funcionalidades respecto a las definidas en la versión anterior (CSS2), de tal forma que se mantiene la compatibilidad entre ambas.

Pese a que se comenzó a trabajar en CSS3 nada más publicar la recomendación oficial de CSS2 y los primeros borradores de CSS3 fueron liberados en 1999 [7]**,** en el año 2011 había alrededor de 50 módulos publicados, pero algunos otros pese a ser razonablemente estables siguen en fase de candidatos y sus implementaciones en los diferentes navegadores son señaladas con los prefijos del motor del mismo [8]. Sin ir más lejos, en este proyecto se ha utiliza la propiedad de CSS3 *transition*, que hemos de definir varias veces, cada una según el navegador en el que se lanzará el proyecto.

```
-o-transition: left 1s;
-ms-transition: left 1s;
-webkit-transition: left 1s;
transition: left 1s;
```

A continuación vamos a comentar más a fondo la tecnología más importante utilizada en este proyecto, como ya se ha mencionado anteriormente se trata de WebGL, veremos un poco de historia sobre esta tecnología y algunos ejemplos de los que se puede llegar a conseguir con ella.

### *WebGL*

En 2006, hizo su aparición en los gráficos 3D en navegador web, WebGl, aunque no fue hasta 2009 que se formó el WebGL Working Group, formado por Mozilla y Khronos [1], con el fin de ir añadiendo esta tecnología a los navegadores actuales.

WebGL nos permite mostrar gráficos acelerados por GPU en nuestro navegador, sin necesidad de instalar plug-in alguno, el único requisito es que la plataforma soporte OpenGL ES 2.0, de hecho muchos consideran que WebGL es una API de OpenGL ES para JavaScript.

Actualmente está soportado de forma total por Google Chrome, Internet Explorer 11 o superior y Mozilla Firefox, sin embargo para los navegadores Opera y Safari, funciona pero con ciertas limitaciones.

En cuanto al apartado de modelado hay que destacar que WebGL es un poco tedioso de programar, por lo que se suele hacer uso de librerías de más alto nivel, la más conocida es Three.js de la que hablaremos más adelantes, puesto que es la librería utilizada en este proyecto. Sin embargo, el contenido mostrado por WebGL puede ser creado por herramientas de desarrollo de contenidos 3D, como bien puede ser Blender o Autodesk Maya. Adicionalmente también podemos encontrar servicios que nos permiten publicar contenido en línea 3D interactivo utilizando WebGL como p3d.in y Sketchfab.

Otra ventaja de WebGL es que si creas unas escena 3D y el navegador desde la que se lanza no soporta WebGL, por cualquier problema o por alguna tecnología que le impide al navegador utilizar la tecnología OpenGL, pero sigue pudiendo mostrar gráficos 3D, podrá mostrarte la visualización pero sin la aceleración hardware que WebGL proporciona. El problema es que con otras librerías, a pesar de basarse en WebGL eso no sucede, y en esos casos se nos muestra una imagen de la escena.

Algunos ejemplos con esta tecnología los podemos encontrar fácilmente en internet, de hecho no es raro encontrar demos de la propia compañía Google, como por ejemplo https://www.chromeexperiments.com/globe [9]. Este demo se trata de un mapa mundial 3D desarrollado con WebGL que cualquiera de nosotros puede descargar en el repositorio GitHub, introducirle datos y finalmente mostrar un gráfico sobre el planeta mostrando cualquier tipo de información representada en este.

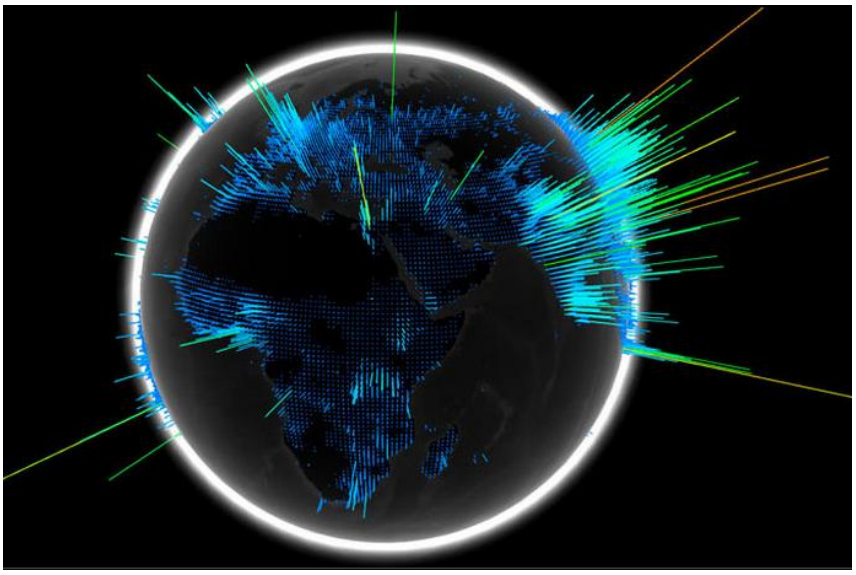


**Figura 1: Gráfica sobre planeta Tierra con WebGL**

En este otro ejemplo, que comenzó con algo para probar la tecnologñia y terminó siendo un producto con beneficios y muy completo, se puede ver claramente la utilidad y potencia de WebGL para aprender y enseñar. El proyecto se llama ZygoteBODY [10] y lo que ofrecen es una visión del cuerpo humano e incluso muestran embarazos y la evolución de estos mostrando la anatomía humana en 3D, de forma muy interactiva.


**Figura 2: Captura de ZygoteBODY.**

Vistos estos ejemplos y la gran cantidad que podemos encontrar y probar, todo esto sin instalaciones adicionales, queda patente la potencia de WebGL y su continuidad, ya que para muchísima gente es la mejor opción para mostrar gráficos 3D en tiempo real y de alta calidad en la Web.

Finalmente concluir con el gran pero que tiene WebGL, y no es otro que la ya comentada dificultad de programación, solo hemos de echar un ojo al código de un *shader* y pensar que eso solo es una mínima parte del trabajo a realizar. Por este motivo es por el que la gran mayoría de gente opta por utilizar librerías que ayudan a trabajar con WebGL, y de entre todas ellas la más conocida es Three.js, que ya hemos mencionado, y es con la que se va a desarrollar el proyecto por lo que a continuación vamos a introducirla y a comentar algunas características de esta.

### *Three.js*

Es la librería más popular para la creación de las animaciones en WebGL, está escrita en JavaScript y se utiliza para mostrar gráficos animados 3D en navegador Web. Al estar escrita en JavaScript, un lenguaje de alto nivel, es muy fácil de aprender e intuitiva y el navegador como entorno de desarrollo prácticamente no requiere configuración.

Three.js fue creada por el español Ricardo Cabello en el año 2010, en sus inicios fue escrita en ActionScript y posteriormente traducida a JavaScript, principalmente por la importancia de no tener que compilar el código antes de cada carga, y al ser javaScript un lenguaje de *scripting* no es requerido esto.

Posteriormente otras personas se unieron al proyecto, Branislav Ulicny, Joshua Koo, Paul brunt, etc. Actualmente contribuyen más de 90 programadores, debido al potencial que muestran tanto WebGL como Three.js cada son más las personas que se unen al crecimiento de estas tecnologías.

Al igual que sucede con WebGL, en Three.js podemos importar modelos creados con software de edición 3D, por lo que gracias a un plugin podemos exportar modelos desde otros programas a formato JSON para leerlo fácilmente con Three.js. Adicionalmente podemos crear modelos tabíen con otros lenguajes de programación y convertirlos al formato JSON mediante scripts que podemos encontrar en el repositorio en GitHub de Three.js, https://github.com/mrdoob/three.js/.

Como principales características para crear entornos y modelos 3D, encontramos desde efectos en tiempo de ejecución que se añaden en la escena, como por ejemplo niebla, hasta las mencionadas funcionalidades de importación y exportación de modelos, pasando por los elementos básicos de cámaras, luces, materiales, geometría, etc. Uno de los elementos que tiene más importancia cobra es le “Emisión de rayos” (*Raycasting* en inglés), principalmente porque sirve para seleccionar objetos de la escena y en WebGL no hay nada similar, pero sobre todo por lo que supone para el usuario ser capaz de interactuar con la escena de forma tan dinámica.

Esta funcionalidad se basa en crear una línea virtual entre dos puntos del espacio 3D, de tal forma que devuelve los objetos con que intercepta en un array y de forma ordenada, es decir, si lanzamos esta opción creando esta línea y haciéndola pasar por la cámara y el ratón, gracias a esto podemos conseguir que el usuario interactúe de forma muy dinámica con la escena.

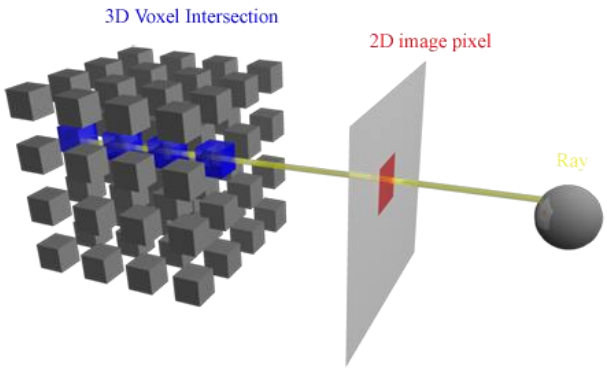



**Figura 3: Funcionamient raycasting en Three.js**

Para más información sobre esta librería, en la página http://threejs.org/ podemos encontrar la API de Three.js y muchos ejemplos para aprender y sobre cómo se utilizan las librerías y funciones. Inclusive tenemos una guía de inicio rápido y numerosos ejemplos para mostrar la potencia de la librería.

Como ejemplo destacable podemos ver un modelo 3D del coche de Batman para el videojuego The Arkam Knight [11], en el que podemos cambiar la vista con el ratón e ir viendo las diferentes funciones del coche con gran detalle y con una forma de mostrarlo muy futurística.

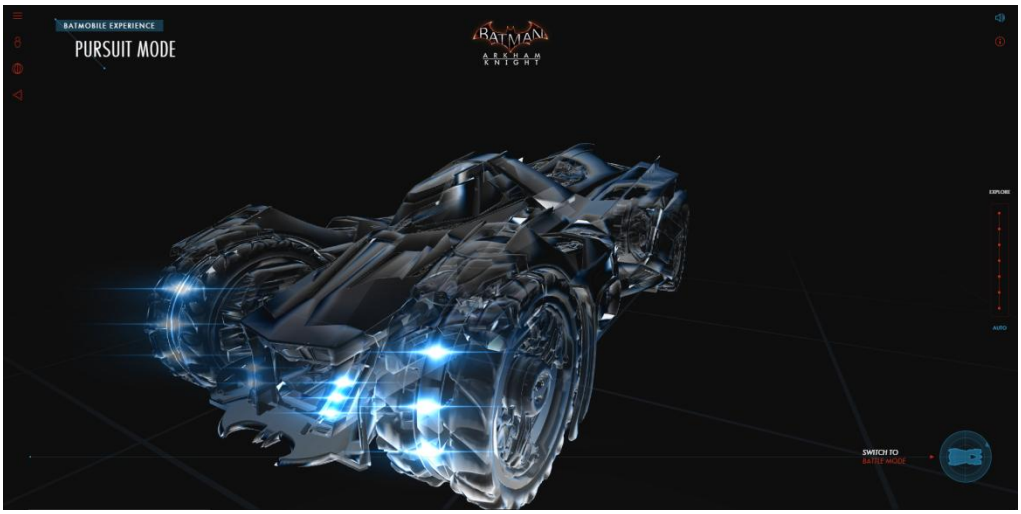



**Figura 4: Captura del batmovil en Three.js.**

Para solventar esto, la arquitectura realiza una serie de pasos, cada uno de ellos con una duración limitada y por tanto, con posibilidad de reacción ante el entorno.

## Conclusiones

Como hemos visto las nuevas tecnologías para mostrar 3D en navegador son extremadamente potentes en comparación a sus predecesoras, lo que nos permite crear libremente páginas web con contenido 3D sobre cualquier tema y con una velocidad de renderizado nada despreciable.

Aunque generalmente los ejemplos que encontramos en la web es software que muestra la información en forma de gráficos 3D, en este proyecto vamos a mostrar que estas tecnologías (WebGL, Three.js,…) también tienen potencial en el desarrollo de videojuegos.

A continuación vamos a pasar a describir el diseño de la herramienta y la funcionalidad que se desea obtener.

# Capítulo 3. Diseño de la solución

## Análisis

En este proyecto se va a desarrollar un producto comercial en navegador web, cuya principal funcionalidad es el poder jugar a un juego de construcción a varios niveles, según el usuario, ya que podemos utilizar la simple funcionalidad de construir modelos simples para que incluso los niños se puedan entretener o llegar a construir y guardar nuestros propios modelos 3D con piezas personalizadas y guardarlos para poder crear grandes modelos a largo plazo.

Debido a la gran variedad de público objetivo, entre los cuales se encuentran los niños, se ha programar un sistema que simule, en la medida de lo posible, la física de la realidad, con el objetivo de mejorar la capacidad espacial y lógica del usuario. Para conseguir esta funcionalidad y cierto grado de realismo, se utiliza en gran medida la función de *raycasting* de Three.js, tanto para posicionar las piezas en la escena a la hora de construir, como para calcular las interacciones entre ellas y el propio entorno.

Gracias a las posibilidades que brinda Three.js se ha dotado a las piezas de un aspecto similar a los que utiliza LEGO http://www.lego.com/es-es , tal y como se puede observar en la figura 5, con ello se pretende mejorar la percepción espacial, puesto que las piezas solo se pueden acoplar mediante los anclajes superiores de las piezas.

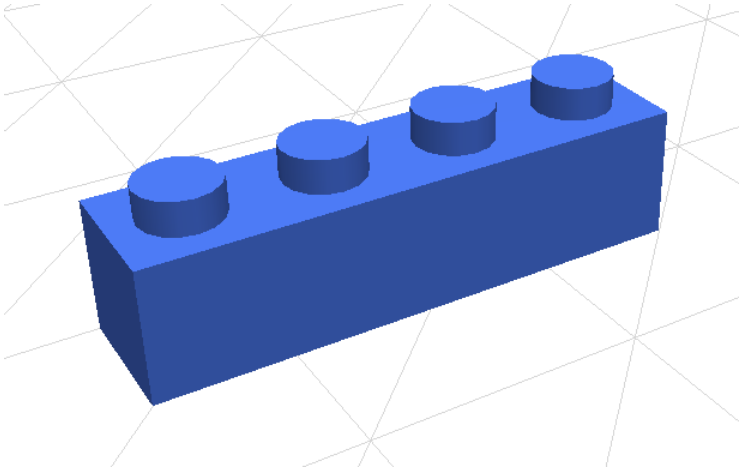


**Figura 5: Ejemplo de pieza de Lego.**

La idea general es simular el comportamiento que tienen estas piezas, es decir, que se tenga que anclar unas con otras para poder sujetarse. Por poner un ejemplo, a una pieza que este en alto no se le podría acoplar otra en el lateral, pero si hacerlo arriba o abajo, según los huecos libres.

Añadiendo a lo dicho anteriormente, como requisito indispensable del proyecto, es su funcionamiento en la mayoría de navegadores web, entre los que se han de encontrar a los más utilizados, como son: Google Chrome, Mozilla Firefox e Internet Explorer.

El funcionamiento del juego ha de ser simple, la idea principal es que con el ratón y sin muchas dificultades cualquier persona sea capaz de utilizarlo, adicionalmente se han añadido controles, tanto con acceso directo como iconos en la barra de acción, de tal forma que sean visibles. Ni que decir tiene que el juego ha de tener una buena tasa de refresco, es decir, ha de ser ligero y evitar tirones.

Una vez comentados los requisitos básicos del proyecto, a continuación vamos a explicar el diseño de este.

## Diseño

El desarrollo del diseño partió del elemento más básico y al mismo tiempo fundamental en el videojuego, el bloqué. El funcionamiento esencial debe de ser seleccionar una pieza, de entre las disponibles, y veamos como la estamos colocando en la escena, antes de "soltarla" definitivamente. Por lo tanto partimos de dos elementos fundamentales, la plataforma sobre la que construimos y las piezas que utilizamos.

Si nos fijamos en la Figura 5, las piezas con las que construimos se basan en un cubo, es decir, una figura geométrica con medidas de altura, anchura y profundidad, a la que le añadimos unos anclajes, jugando con esas medidas podemos calcular la posición para piezas adyacentes, tanto para colocarlas junto a ellas como para evitar que se solapen, como sucede en la siguiente figura.

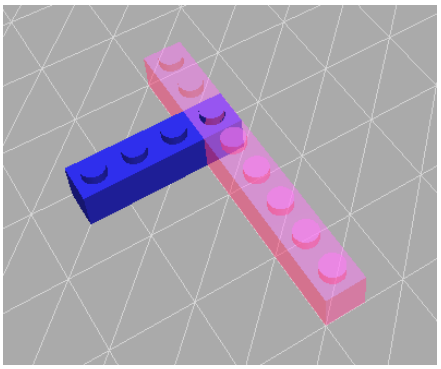


**Figura 6: Solapamiento de piezas.**

Como podemos observar, en la Figura 6 se produce un solapamiento, ambas piezas son deberían de ser del mismo color, y la pieza que estamos intentando colocar actualmente es la que es tiene cierta transparencia. Al producirse solapamiento, el juego no nos dejaría colocar la pieza, y nos lo muestra dotando a esta de un tono rojizo, esto sucede siempre que estamos intentando colocar una pieza en una posición errónea.

Para poder detectar estos posibles errores de posicionamiento cuando vamos desplazando el ratón por la escena arrastrando una pieza para colocarla, lo que hacemos es generar, partiendo de ella, los ya mencionados *raycasting* de Three.js partiendo siempre desde el centro de la pieza, para que la pieza pueda "ver" a su alrededor y actuar en consecuencia.

Tenemos dos posibles problemas que hemos de solucionar utilizando este método, el primero es detectar si estamos solapando piezas, y el segundo es ir mirando la parte superior e inferior de la pieza, en los lugares en los que se localizan los anclajes, para ver si hay alguna otra pieza con la que acoplarse. Una vez ambos test dan un resultado correcto, se permite al usuario colocar la pieza.

Todo lo mencionado anteriormente sucede cada vez que movemos la pieza, pero sin afectar al rendimiento gracias a la aceleración por hardware, de hecho la pieza sigue moviéndose con fluidez y en ningún momento hay ninguna interrupción.

Vamos a comentar los cálculos a seguir para detectar ambos casos, y como se realizan los test:

- En el primer caso, el solapamiento de piezas, el más trivial. Partiendo del centro creamos un *raycasting* hasta cada uno de los vértices de la pieza, 8 en total, si en cualquiera de ellos detecta otra pieza, la función devuelve como verdadero el resultado sobre el test de pieza bloqueada, por lo que nos impide colocar la pieza, y al mismo tiempo detiene el siguiente cálculo, puesto que la pieza ya está bloqueada y de esta forma evitamos más coste computacional.

- En el segundo caso, comprobar si la pieza está anclada, requiere de muchos más *raycasting* para "ver" si alguno de los anclajes está en contacto, y por lo tanto enganchado, a otra pieza. Para ello se van creando raycasting hacia arriba y hacia abajo en la posición de cada uno de los anclajes (obviamente, estas posiciones dependen de cada pieza), el tamaño del *raycasting* en este caso es de la mitad de la altura del cubo más una constante para detectar solo si hay una pieza colindante y no más lejana, en caso de encontrar una pieza acoplada se permite al usuario colocar la pieza. A continuación en las Figuras 7 y 8 se muestra el funcionamiento de este cálculo.

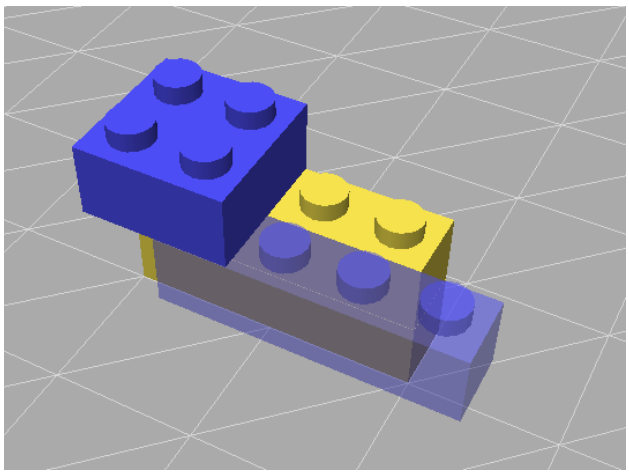


**Figura 7: Posición correcta con anclaje superior.**

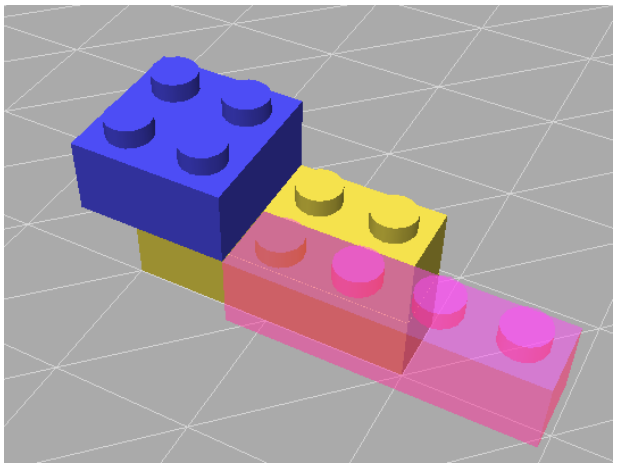


Figura 8: Posición incorrecta sin anclaje.

En las dos figuras superiores se puede comprobar como funciona el test de anclaje, en ambos casos se ha superado correctamente el test de solapamiento, sin embargo en la Figura 7 vemos como al estar ancladas las dos piezas azules nos permite dejarla en esa posición y por el contrario en la Figura 8, al estar la pieza flotando no nos permite su colocación.

Estos test se realizan para dotar al juego de mayor realismo, como se ha comentado anteriormente, se pretende mejorar la capacidad espacial y esto no sería posible si pudiéramos atravesar cubos y dejar otros flotando.

Siguiendo con la colocación de las piezas, mientras tenemos una seleccionada y estamos pensando colocarla, podemos hacer ciertos cambios en esta, entre ellos podemos rotarla 90% en el eje Y, lo que nos deja disponibles las 2 posiciones básicas de cualquier pieza lego.

Finalmente, en cuanto añadimos una pieza, se nos queda el ratón libre para poder editar la escena, básicamente podemos seleccionar las piezas y recolocarlas, siempre siguiendo los mismos principios que al soltarlas, pero con una nueva restricción de no poder mover una pieza que es el soporte de otra.

Adicionalmente tenemos la opción de colorear las piezas, tanto antes de colocarlas como posteriormente, de esto hablaremos a continuación.

Una vez establecido los requerimientos de la funcionalidad básica para el videojuego, vamos a pasar a mostrar la interfaz y describir sus componentes y funcionalidades.

### *Interfaz*

Una vez se ha conseguido la base del videojuego hemos de dejar claro la interfaz y las funcionalidades que queremos conseguir con ella. Debido a que se trata de un juego de construcción era necesaria una interfaz del estilo de modeladores 3D, se ha escogido algo similar a la interfaz Blender https://www.blender.org/ (Figura 9), que se utiliza para la iluminación, renderizado, animación y, por supuesto, modelado de gráficos 3D.

Vamos a comentar a muy grandes rasgos que podemos encontrar en la interfaz de Blender. A la derecha tenemos distintas formas de editar la escena, grupos, propiedades, etc. A la izquierda, dependiendo de la selección actual, nos muestra o bien información sobre la selección, o por otro lado un menú para añadir figuras por defecto. En la parte superior encontramos la barra de opciones, con tales como preferencias, guardar, tipos de visualización… En la parte interior está la interfaz de animación, es similar a una barra de reproducción.

Esta es la interfaz por defecto de Blender, per buen programa de modelado y edición, esta puede ser editada.

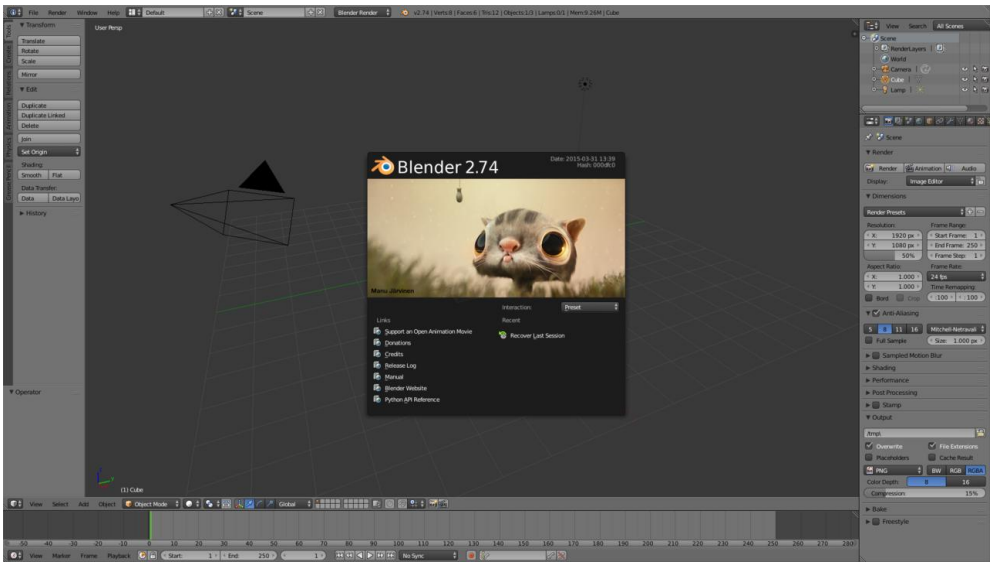



**Figura 9: Interfaz Blender**

Una vez vista la interfaz de un software potente con un equipo detrás y años de desarrollo, en la que nos hemos basado estructuralmente para crear nuestra interfaz, vamos a pasar a introducir la nuestra.

La idea es mostrar en todo momento, en todo momento información sobre lo que está sucediendo, tanto a la hora de elegir dónde colocar la pieza seleccionada, como mientras se edita la escena, especialmente sobre la geometría de las piezas.

Por otro lado, también es fundamental la correlación entre menús, cerrar ciertos avisos al mostrar otros, ocultar menús contextuales cuando hemos abiertos otras opciones, etc.

La interfaz final es la que se puede observar en la imagen inferior (Figura 10).

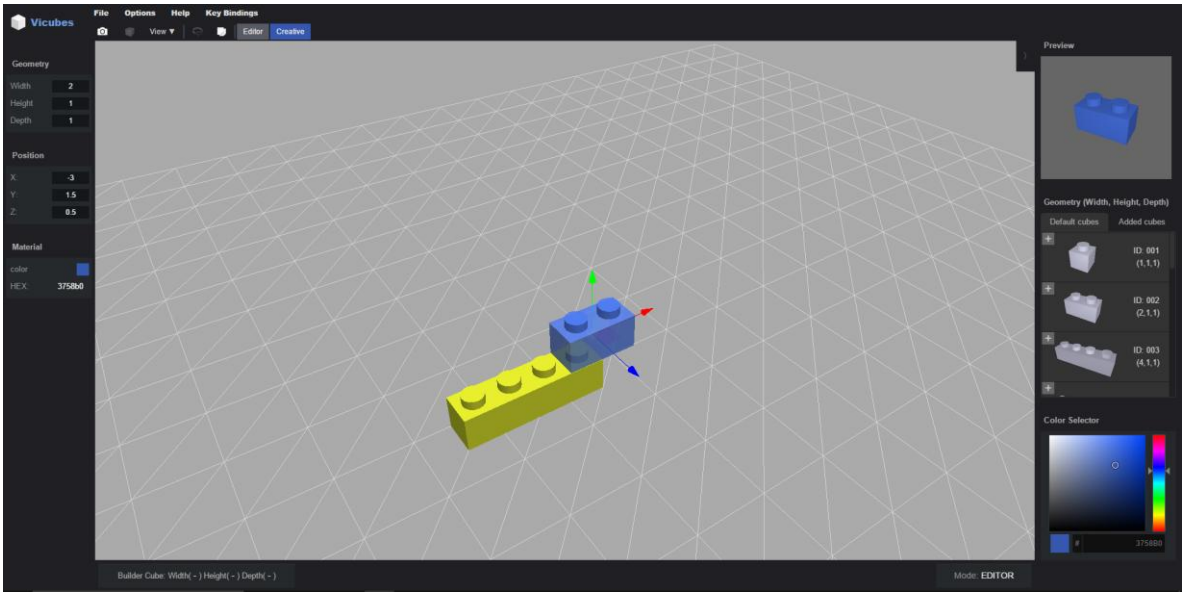



**Figura 10: Interfaz final del videojuego.**

Como podemos observar en la imagen superior, la interfaz es un marco que rodeo toda la escena, con varias opciones, listas de figuras e incluso un pre-visualizador.

Antes de proceder con las funcionalidades de la interfaz, vamos a describir brevemente la interfaz y sus componentes.

En primer lugar nos centramos en la parte más obvia de la interfaz, la parte central, en la que mostramos la escena en tiempo real, y en la que podemos mover la cámara con total libertad, en el ejemplo tenemos seleccionada la pieza azul para edición.

En segundo lugar, vamos a hablar de la barra superior, en esta encontramos a la izquierda del todo el logo de la herramienta e inmediatamente a su derecha distinguimos entre dos filas, la fila superior contiene las opciones para guardar/cargar ficheros de construcciones realizadas, cambiar la configuración, menú de ayuda e información. En cuanto a la barra inferior a esta, tenemos algunas funcionalidades de edición y de visualización, que detallaremos en profundidad más adelante.

En cuanto a la columna de la izquierda, encontramos información sobre la pieza seleccionada actualmente, datos sobre su geometría, posición en la escena y color del material de la pieza, esta columna es retráctil, por lo que cuando no tenemos ninguna pieza seleccionada, se esconde.

Al lado contrario, en la columna de parte derecha, se encuentran tres zonas claramente diferenciadas, la primera de ellas es la pre-visualización de las piezas, sirve tanto a la hora de intentar colocar una pieza, como cuando la estamos editando. Justo debajo tenemos una de las partes más importantes de nuestra interfaz, el menú con las piezas que podemos construir por defecto, así como una pestaña que nos muestra otro menú con piezas de tamaños personalizados y la posibilidad de añadir más.

Posteriormente, vemos una paleta de colores cuya funcionalidad es, por supuesto, cambiar el color de la pieza con la que estemos interactuando actualmente, cuando seleccionamos una geometría para construir se cambia al color de la ultima elegida y a partir de ahí podemos editar, por el contrario, si seleccionamos una ya colocada, el color cambiará al de la selección actual.

Esta columna también es retráctil, pero en este caso se oculta a placer del usuario, esta funcionalidad se ha añadido sobre todo, teniendo en cuenta los monitores pequeños, por si en ciertos momentos el usuario quiere ocultar esta barra para tener una mejor visualización de la escena.

Por último tenemos la barra inferior, que tiene un marco a cada lado, en el de la izquierda se muestra la geometría de la pieza a colocar, si estamos intentando colocar una, llamada 'Builder cube', y justo al lado contrario, en el marco de la derecha, tenemos el modo de juego actual, tenemos 2 de ellos, 'Editor' y 'Creative', simplemente identifican con más claridad si estamos colocando una pieza (Creative) o editando la escena y sus elementos (Editor).

### *Funcionalidades de la Interfaz*

En este apartado se va a comentar las funcionalidades que nos proporciona la interfaz, así como los posibles menús de aviso que nos puede aparecer en la aplicación.

En primer lugar, vamos a hablar del menú de cabecera, la barra de la parte superior, en la que encontramos las opciones de: 'File', 'Options', 'Help' y 'Key Bindings'.

1. *File (Archivo)*: Menú principal para trabajar con los proyectos, al hacer click se nos despliega el siguiente menú (Figura 11).

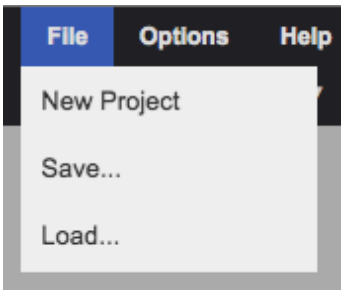



**Figura 11: Menú 'File'.**

1.1. *New Project (Nuevo Proyecto)*: Nos muestra una ventana de alerta para verificar la opción seleccionada (Figura 12), y en caso de aceptar se resetearía el marco de dibujo y se restablecerían las opciones por defecto.


Figura 12: Aviso 'New Project'.

1.2. *Save… (Guardar…):* Sirve para crear un fichero *json* en local con todo lo que se ha construido actualmente y la disposición de todos los elementos, así como los datos del menú opciones, por si alguno ha sido modificado.

La imagen que se muestra para guardar el fichero es la siguiente.


Figura 13: Menú 'Save'.

En este menú introducimos el nombre del fichero y automáticamente en el directorio que nuestro navegador tenga asignado se creara el fichero *json* con los datos de nuestra construcción.

1.3. Load… (Cargar…): En este caso lo que hacemos es seleccionar un fichero *json* de nuestro ordenador y lo cargamos en el editor. Inicialmente nos muestra el siguiente menú.


Figura 14: Menú 'Load'.

Una vez hacemos click en 'seleccionar archivo' ( está en un idioma diferente porque utiliza el lenguaje del navegador y no el de la aplicación ), se nos muestra la típica ventana de exploración de Windows, donde localizamos nuestro fichero json y le damos a abrir. A continuación se nos cargara, en caso de ser un fichero correcto, la escena que estaba guardada.

2. *Options* (Opciones): En este menú, encontramos dos accesos para cambiar el entorno.

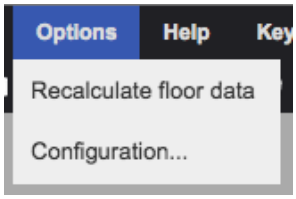



Figura 15: Menú 'Options'.

2.1. *Recalculate floor data* (volver a calcular los datos del suelo): La utilidad de está opción viene a la hora de construir cerca del borde en la plataforma, puesto que es posible construir fuera de esta, cuando queremos ampliarla le damos a esta opción y se añade más plataforma según la posición de las piezas construidas. Cuando estamos desplazando una pieza por el suelo, automáticamente también se va recalculando esto para aumenta el tamaño de este en caso de que sea necesario.

2.2. *Configuration…* (Configuración…): El menú de opciones por excelencia (Figura 16) de nuestra aplicación, aquí podemos editar distintos elementos del entorno y opciones varias, como por ejemplo , el fondo trasparente a la hora de hacer capturas de pantalla.

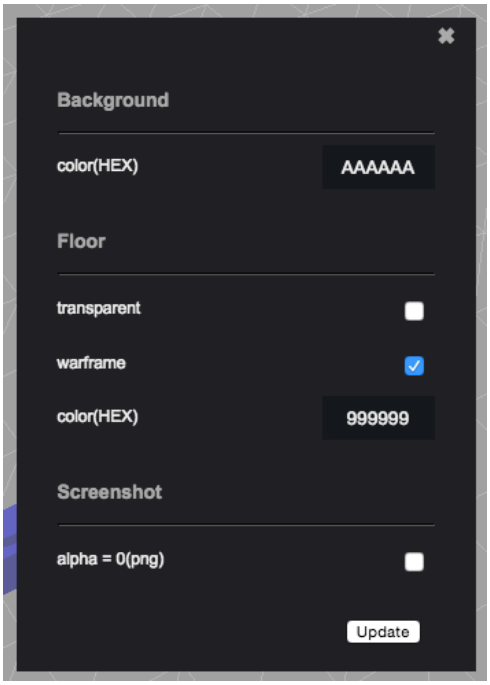



Figura 16: Opciones de configuración.

Vamos a ir comentando las diferentes opciones de configuración de arriba abajo, en primer lugar encontramos el color de fondo de la escena en hexadecimal, por defecto es un gris claro ( 0xAAAAAA).
En segundo lugar tenemos las 3 opciones correspondientes al suelo que por defecto está mostrando solo las uniones entre vértices (*warframe*), con color gris claro (0x999999) y visible (*trasparent*). De tal forma que podemos editar su color y la visualización, tanto el verlo como una superficie plana como hacerlo trasparente, opción que tiene una gran sinergia con la siguiente.
Finalmente tenemos la posibilidad de marcar la opción de *alpha = 0*, para que a la hora de hacer capturas de pantalla, el fonde del *canvas,* sea totalmente trasparente para solo capturar las piezas de la escena en la imagen.

3. En penúltimo lugar tenemos la opción de '*Help'* (Ayuda), que al hacer clic en ella nos muestra un menú de ayuda rápida sobre los controles básicos de la herramienta.

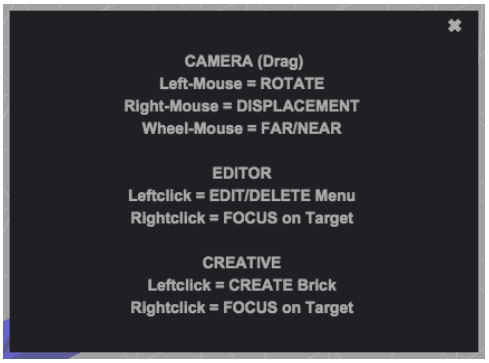
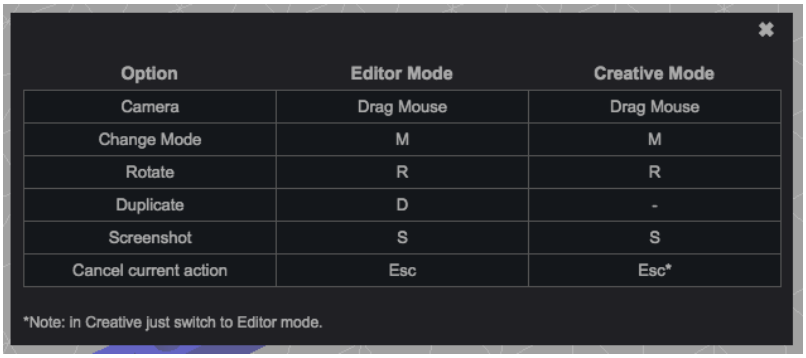



Figura 17: Menú de ayuda.

4. Finalmente tenemos la opción de 'Key Bindings' (Teclas clave), con una lista de los botones de acción rápida, todos ellos también disponibles desde opciones de la interfaz, pero de esta manera es más rápido trabajar con la herramienta.



| Option | Editor Mode | Creative Mode |
|---|---|---|
| Camera | Drag Mouse | Drag Mouse |
| Change Mode | M | M |
| Rotate | R | R |
| Duplicate | D | - |
| Screenshot | S | S |
| Cancel current action | Esc | Esc* |

*Note: in Creative just switch to Editor mode.

Figura 18: Menú de acceso rápido.

A continuación vamos a hablar de las opciones que encontramos en la segunda barra de la parte superior de la interfaz, en ella se encuentran la mayoría de opciones que nos permiten interactuar con la escena de manera directa, la gran mayoría de ella también son accesibles median uso de acceso rápido.

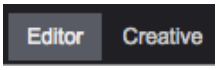
1. Captura de pantalla  (Acceso rápido 'S' ): Este botón nos permite hacer una captura en formato de imagen *png*, nos abre una ventana nueva con la captura de la escena al completo, si en *Configuración* hemos marcado la opción de *alpha = 0,* la imagen será con fondo trasparente.

2. Modelos realistas  : En este caso no tenemos acceso rápido para está opción, puesto que no es de uso común. Al activarla se vuelve la iluminación de los vértices de los piezas, en algunos casos es de utilidad para dotar de realismo a la escena, pero generalmente para construir es mejor diferenciar claramente unos vértices de otros.

3. 'View' (Vista): Es un menú desplegable con las diferentes posiciones de la cámara para orientarla de forma perfecta desde cada uno de los laterales, la parte superior, frontal y trasera.

4. Rotar la selección actual  (Acceso rápido 'R'): Su funcionamiento es sencillo, cuando estamos interactuando con alguna pieza del entorno, esta opción se activa para poder darle y rotar, de ser posible, la pieza actual.

5. Duplicar  (Acceso rápido 'D'): Esta opción solo se encuentra activa cuando estamos editando alguna pieza de la escena, es decir cuando la tenemos seleccionada, al utilizarla pasamos automáticamente al modo de creación para colocar una nueva pieza exactamente igual a la que estábamos editando.

6. Modos de juego 

(Acceso rápido 'M'): Desde este mini-menú podemos alternar entre los dos modos de juego, editor y creativo, como ya se ha dicho anteriormente, el primero es para poder retocar los elementos de la escena y el segundo es el que utilizamos para colocar piezas en esta. El que tiene el fondo en grisáceo es el modo actual, y tanto utilizando la tecla 'M' como dándole a estas opciones podemos ir alternando de modo de juego.

En cuanto a la parte derecha de la interfaz, una de las partes más importantes, ya hemos hablado anteriormente de las tres zonas claramente diferenciadas en ella, por lo que ahora vamos a profundizar para ver como interactuamos con estas.

- En primer lugar encontramos el apartado de '*Preview*' (Previsualización), como su propio nombre indica, vemos una imagen directa de la pieza con la que estamos trabajando, y como quedaría una vez la coloquemos o dejemos de editarla.


**Figura 19: Marco de previsualización.**

Tal y como sucede en la escena principal, esta podemos rotarla y hacer zoom para ver en detalle la pieza en cuestión, pero en este caso, la cámara está permanentemente asociada a la pieza.

- En segundo lugar tenemos el menú de piezas, dentro de este podemos ver en la parte superior dos pestañas: '*Default cubes*' (Cubos por defecto) y '*Added cubes*' (Cubos añadidos).

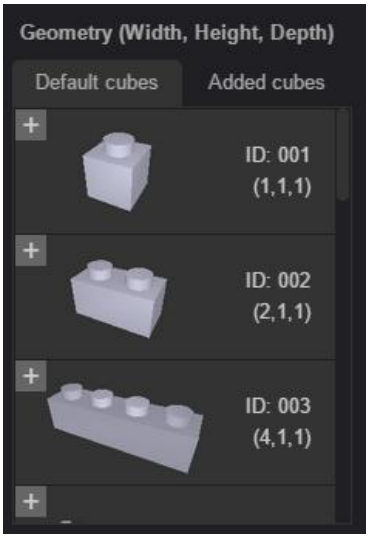



**Figura 20: Captura del menú de piezas.**

En la primera pestaña, activa por defecto, se nos muestran las piezas que el juego trae de serie, en cuanto a la segunda, tenemos una opción para añadir cubos con geometría personalizada.

- ‘*Default cubes*’ (Cubos por defecto): En esta pestaña vemos las imágenes de los cubos guardados por defecto en el juego, en cada contenedor vemos a la izquierda la imagen del cubo en cuestión y a la derecha, justo debajo de su identificación, su medida según la que indica el título ‘*Geometry(Width, Height, Depth)*’, es decir, Anchura, Altura y Profundidad.

- ‘*Added cubes*’ (Cubos añadidos): Esta pestaña, oculta por defecto, nos muestra las piezas añadidos a nuestro proyecto actual, al crear un proyecto la tenemos vacía y tan solo tendremos la opción de añadir nuevo cubo (Figura 21).

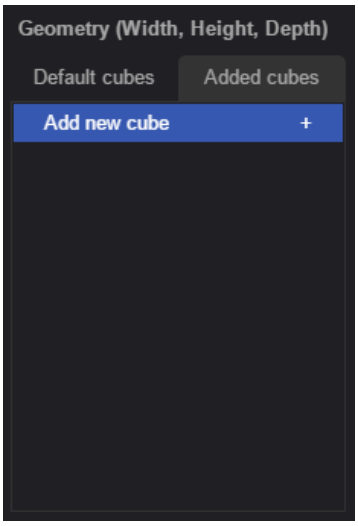



**Figura 21: Pestaña de cubos añadidos.**

Tras hacer clic en ‘Add new cube’ nos muestra una ventana (Figura 22) en la que nos pregunta sobre el tamaño de la nueva pieza, limitándola a valores de 1 o superiores pero solo pares, debido a una limitación en el movimiento al colocar las piezas.


**Figura 22: Menú de creación de pieza.**

❖ Finalmente, en último lugar, tenemos la paleta de colores, bastante sencilla e intuitiva, está dividida en tres partes, a la derecha tenemos los colores disponibles, una vez desplazamos al color elegido, en la parte de la izquierda podemos cambiar tonalidad y niveles de mezcla, por último, en la parte inferior, tenemos un cuadrado con la muestra de color elegido y su valor en hexadecimal, este valor también se puede cambiar por cualquier otro, lo que haría que se ajustase todo el selector de color para mostrar la información de forma correcta.


Figura 23: Paleta de selección de color.

Como última información sobre la columna izquierda de la pantalla, es la utilidad de poder ocultarla, pulsando la flecha  , que se encuentra en la parte superior izquierda de esta, concretamente al lado del nombre 'Preview'. Para volver a mostrar esta parte de la interfaz le damos a esta misma flecha que no llega a ocultarse y volveremos a mostrar toda la columna.

Al lado contrario de la interfaz, en la parte izquierda, tenemos la columna con información directa sobre las piezas seleccionadas para editar, sin embargo, pese a ser muy útil no hay interacción posible con está interfaz, por lo que su utilidad se queda simplemente en mostrar datos, como se ha dicho anteriormente, solo se muestra cuando tenemos alguna información que mostrar, y no de manera permanente.

Antes de continuar con la interfaz, hemos de centrarnos ahora en la parte central de la pantalla, donde construimos la escena. Para interactuar con una pieza, tal y como se muestra en el menú de ayuda, haciendo clic izquierdo con el ratón en una pieza, aparece el siguiente menú.

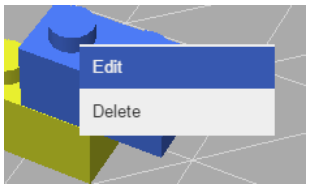



Figura 24: Menú clic del ratón.

En este menú se muestran dos opciones sencilla, '*Edit*' (Editar) que nos sirve para centrarnos en la pieza y así poder cambiar su posición y/o el color, y '*Delete*' (Borrar) para eliminar, de ser posible, la pieza en cuestión.

Para terminar con la interfaz tenemos la parte inferior, a la que le sucede exactamente lo mismo que la columna de la izquierda, y es que carece de funcionalidad y su única utilidad es mostrar información, tanto sobre la geometría de la pieza que estamos construyendo (parte inferior izquierda) como sobre el modo de juego actual (parte inferior derecha). Por si ha surgido la duda, comentar que la información sobre la geometría en es parte de la interfaz y en la columna de la izquierda no es la misma, en la primera mostramos la geometría de la pieza que estamos arrastrando para colocar y, en la columna de la izquierda, la información mostrada corresponde a  la pieza seleccionada para edición.

Para comprobar el correcto funcionamiento y aplicación del sistema, las pruebas para realizar son sencillas.

En primer lugar, cada vez que se ha implementado alguna opción nueva, se ha hecho pruebas en distintos navegadores y comprobado su correcto funcionamiento. Así mismo, a la hora de aplicar opciones como las del menú de configuración, que trabajan junto con otras partes de la interfaz, se han ido probando una a una de nuevo.

En segundo lugar, para evitar posibles errores de los algoritmos y probar la funcionalidad completa, una vez se carga el juego, se crea alguna escena y se guarda, con una configuración diferente a la original, posteriormente se crea un nuevo proyecto y se repite la acción de crear y guardar, para finalmente cargar el primer fichero y comprobar que todo funciona correctamente. Cualquier prueba es poca en este tipo de sistemas que dan bastante libertad de acción al usuario, es necesario que este sienta que el producto es robusto.

Por último, en el capítulo 5, hablaremos de forma más concienzuda de las diferentes pruebas de robustez realizadas en la herramienta.

## Conclusión

Por lo tanto, basándonos en el tipo de software que queremos hacer, un videojuego, se ha diseñado una interfaz bastante intuitiva y lo más importante que soporte a un sistema robusto.

En cuanto a la interacción entre las piezas de construcción, la máxima prioridad es que se comporten como sucede en la realidad, solo se pueden acoplar mediante los pivotes que sobresalen y no se pueden queda flotando en el aire por arte de magia. Para poder jugar con mayor libertad, hay una breve lista de piezas por defecto con las que ya se pueden hacer grandes construcciones, pero en caso de que el usuario quiera añadir las suyas propias puede hacerlo.

En definitiva, tenemos un juego de construcción con controles rápidos e intuitivos para usuarios que simplemente quieran construir y con otras opciones algo más complejas para usuarios con algo más de conocimiento.

# Capítulo 4. Implementación

## Tecnologías y conceptos

Según los requisitos del propio proyecto, para la creación de los modelos 3D en navegador web se ha utilizado la tecnología WebGL, más concretamente la librería Three.js de alto nivel para programar con mayor comodidad, ambas comentadas en el capítulo anterior.

Entre los conceptos interesantes de Three.js que vamos a utilizar al desarrollar el proyecto, destacan dos:

En primer lugar hemos de saber que conforma de forma genérica cada uno de los elementos de la escena, estos están formados de una malla (*Mesh*), que al crearla requiere de una geometría y un material.

Por lo que respecta al material, todos los elementos utilizan un material sin brillos (*Lambert*), al que posteriormente se le puede cambiar el color sin problemas. Por otro lado, la geometría, depende de cada uno de los elementos, pero la parte importante de las piezas, el cubo, se ha creado utilizando la geometría *BoxGeometry.*

Three.js tiene disponibles varias librerías internas que nos facilitan el uso de la cámara, comprobación de errores y transformación de los elementos de la escena. Como ejemplo de estas librerías encontramos *OrbitAndPanControls.js* [12] y *TransformControls.js* [13].

Adicionalmente se han incluido librerías de JavaScript para la creación y carga de ficheros desde local a nuestra aplicación web, la librería en cuestión se llama FileSave [14]. Nos permite cargar ficheros en tiempo real desde local, y nosotros con JavaScript controlamos si estos son correctos y cargamos su contenido.

En cuanto a la plataforma de desarrollo, el navegador de trabajo ha sido Google Chrome https://www.google.es/chrome/browser/desktop/index.html , en principio tanto Mozilla Firefox como Chrome tienen la misma capacidad para soportar esta tecnología pero debido a lo familiarizado que estoy con Chrome me he decantado por este último.

## Estructura del software

Vamos a comenzar este apartado comentando lo obvio, la aplicación es una videojuego de navegador, por lo que básicamente es una página web escrita en HTML5 y con hoja de estilo CSS3. Posteriormente utilizamos el elemento *canvas* de HTML5 y ahí creamos nuestra escena utilizando Three.js y librerías adicionales. Por lo tanto vamos a hacer un breve resumen de los ficheros que podemos encontrar en cada una de las carpetas y como está organizado el proyecto.

- Proyecto
  - workspace.html
  - /css
    - general.css
    - colpick.css
    - jquery.jscrollpane.css
    - nanoscroller.css
  - /images
    - bricks
  - /js
    - /game
      - cubes_geometry.js
      - functionalities.js
      - KeyAndMouse.js
      - save_load_files.js
      - ui.js
      - init.js
    - /webgl
      - colpick,js
    - jquery-2.1.1.min.js
  - /lib
    - Three.min.js
    - FileSaver.min.js
    - jquery.jscrollpane.min.js
    - jquery.mousewheel.js
    - jquery.nanoscroller.min.js
    - OrbitAndPanControls.js, TransformControls.js
    - Utils.js, Coordinates.js, dat.gui.min.js

Vamos a comentar por encima que tenemos en cada carpeta:

En primer lugar, la carpeta '*css*', tenemos todos los ficheros de estilo necesarios, el fichero general.css es el único que ha sido necesario crear, ya que el resto eran ficheros que otras librerías utilizaban y venían con estas.

A continuación encontramos la carpeta '*images*', en esta se encuentran todas las imágenes utilizadas en el proyecto, y dentro de esta, la carpeta '*bricks*' contiene las capturas de las piezas que se utilizan en el menú de selección de piezas.

Las dos carpetas que tenemos a continuación son las que tienen toda la implementación y funcionalidad de la aplicación.

En la carpeta '*js*', tenemos guardado el fichero para poder utilizar *jquery* en nuestra aplicación y adicionalmente encontramos dos nuevas carpetas, '*game*' y '*webgl*'. En la primera tenemos todos los ficheros JavaScript creados para la funcionalidad de la aplicación, y en la segunda encontramos un fichero llamado 'colpick.js' que es desde donde se crea el selector de color.

Finalmente tenemos la carpeta '*lib*' en la que se encuentra todos los ficheros JavaScript descargados de internet, para conseguir la funcionalidad de carga de archivos y almacenamiento en local, barras de desplazamiento personalizadas, control de la cámara en el canvas, y ficheros extra para el correcto funcionamiento de Three.js.

Vamos a pasar a explicar los elementos fundamentales del juego y que estructura tienen.

Como se ha dicho anteriormente casi toda la funcionalidad del sistema gira en torno a las piezas de construcción, por lo que en primer lugar vamos a profundizar en su estructura y la información a la que podemos acceder de esas.

```
var material = new THREE.MeshLambertMaterial( {color: 0xAA2222} );
var geometry = new new THREE.BoxGeometry( 2, 1, 1 );
var brick = new THREE.Mesh( geometry, material );
```

En la variable 'brick' ya tendríamos creado el cubo con todas las características para poder construir, pero le faltaría realismo al no tener aún los anclajes, el cubo quedaría de la siguiente forma.

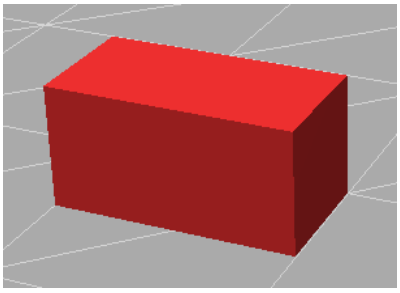


Figura 25: Pieza de construcción sin anclajes.

Para añadir los anclajes a cada uno de los cubos, se ha creado una función que pueda ser utilizada por cualquier objeto malla de Three.js, en JavaScript a este tipo de funciones se les llama prototipo (Prototype), a esta función se le ha llamado '*addStuds()*', y su declaración es la siguiente:

La geometría que se puede ver dentro de la función es la que corresponde a un solo acople de la pieza, posteriormente se consultan los tamaños (anchura, altura y profundidad) de la pieza que ha realizado la llamada y se procede a colocar los correspondientes anclajes, cuyo resultado se puede observar en la siguiente figura.

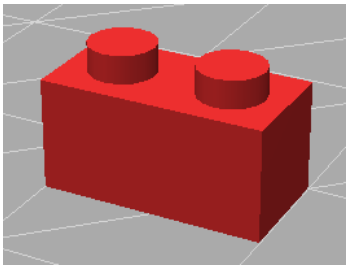


Figura 26: Pieza de construcción con anclajes.

¿En que varía cada pieza a la hora de colocar los anclajes? Es muy sencillo, cuando añades una malla a otra en Three.js, en este caso un anclaje a un cubo, la posición por defecto es el punto (0, 0) de la malla, y esta al crearse, si por ejemplo tiene 8 de anchura, se crearía desde +4 en el eje X hasta -4 en el mismo, por lo que los anclajes sin desplazarlos ni se verían.

Debido a esto, cuando se crea un anclaje, este está desplazado en el eje Y la mitad de la altura de la pieza, para quedar justo en la superficie.

Por otro lado, hay que notar que una pieza con una medida impar, respecto a otra con medida par, varía en sus puntos de anclaje, es decir, poniendo de ejemplo a la pieza de la figura 26 que tiene 1 de profundidad (eje Z), si los anclajes comienzan en el (0, 0, 0), solo tenemos que ir creando en el eje X dando saltos de uno en uno. Por el contrario, si el valor que estamos viendo es par, en ese eje no podemos ir colocando anclajes partiendo desde la posición 0, como sucede en la pieza superior, en el eje X, tiene 2 de ancho. En este caso se comienza en la posición de X +0.5 y -0.5, a la que posteriormente se le va sumando y/o restando 1 hasta que no salimos de la pieza.

Por poner valores a la explicación, nuestra pieza de la figura 26 mida 2 de ancho, 1 de alto y 1 de profundidad. En primer lugar la altura de todos los anclajes sería de 0.5 (alto/2), en segundo lugar tendríamos que en el eje Z, al ser 1 la profundidad, comenzaríamos en el (0, 0, 0) e iríamos desplazando en 1 el valor de Z hasta salirnos de la pieza, en este caso, solo 1 iteración, en la segunda estaríamos en las posiciones +1 Z y -1 Z, saliéndonos de la pieza.

Finalmente tendríamos la anchura, en el eje X, en este caso el valor es 2, par, por lo que comenzamos en las posiciones +0.5 y -0.5, y vamos desplazándonos 1, por lo que en la segunda iteración ya estaríamos en +1.5 y -1.5, saliéndonos de la pieza.

Otro de los elementos más importantes, el ya mencionado *raycasting* de Three.js, líneas virtuales que parten desde un objeto en cierta dirección y devuelven en un array, de manera ordenada, todos aquellos elementos con los que se ha cruzado.

Este elemento cobra una gran importancia tanto en el caso de detectar colisiones entre cubos, caso que ya se ha comentado en el capítulo 3: Diseño de la solución. Pero especialmente en el caso de interacción con el usuario, un rayo trazado desde la posición del ratón a la imagen 2D que muestra el canvas en el monitor es la mejor solución, para que el usuario pueda hacer clic e interactuar con los elementos de la escena.

El objeto raycasting está declarado en una variable global, ya que gracias a la facilidad que nos otorga javaScript con las variables globales, y llamadas entre ficheros, se ha preferido hacer de esta forma para poder consultar los elementos que consulta el usuario en cualquier momento y función.

La declaración de este objeto se realiza de la siguiente forma:

```
//Detect objects
raycaster = new THREE.Raycaster();
```

En el fichero '*init.js*' se crea el objeto y se almacena en la variable global llamada *raycaster,* teniendo el objeto creado, cada vez que se mueve el ratón por la escena se llama a la siguiente función:

```
function detectIntersetedObjects(mouse) {

    raycaster.setFromCamera( mouse, camera );

    intersects = raycaster.intersectObjects( scene.children );

    if ( intersects.length > 0 && intersects[0].distance < 600) {
        ...
    }
}
```

Como parámetro le entra un array de longitud 2, con los valores X e Y que corresponden a la posición del ratón en el canvas. Utilizando esto y la cámara de la escena se crea el rayo que va desde la posición del ratón hacia la imagen 2D que se nos muestra.

En la variable '*intersects*' tenemos almacenado un array con todos los elementos de la escena que cruza este rayo.

Hay tomar nota de algo importante, gracias a introducir como argumento '*scene.children*' en la llamada a la función *'intersectObjects',* los únicos objetos que detectará el rayo serán los hijos directos de la escena, entre los que solo se encuentran las luces, la plataforma y los cubos, remarcar que son solamente los cubos. Los anclajes son hijos de los cubos y están asociados a estos, por lo que no los detecta el rayo, situación que buscamos para que al calcular colisiones y posición respecto a otra pieza solo tengamos en cuenta la geometría de los cubos.

Finalmente si el objeto con el que se intersecta es relevante, se almacena en otra variable global llamada *INTERSECTED*, gracias a la cual podemos ver a tiempo real en cualquier otra función que cubo es con el que estamos interactuando y poder editarlo.

Como último elemento a tener en cuenta tenemos lo que se llama '*BuilderCube'*, se trata simplemente de un cubo declarado con un poco de trasparencia y de geometría variable. Siempre que hablamos de colocar un cubo, en el juego puede parecer que es diferente, pero en realidad siempre es el mismo cubo, la única diferencia es que la geometría que adopta es según el cubo que hayamos seleccionado en el menú, y por supuesto, con el color pasa exactamente lo mismo.

Una vez hacemos clic en una posición, en la que es posible colocar la pieza, lo que el sistema hace es crear una nueva clonando la geometría y materiales que tenía en ese instante el '*BuilderCube'* y apareciendo en la posición que ocupaba.

Este cubo esta declarado también como variable global y por lo tanto es accesible tanto desde los ficheros de funcionalidades como a la hora de realizar acciones con las teclas pulsadas tanto del ratón como del teclado.

Estos 3 elementos, las piezas, el raycasting y el '*BuilderCube*' son los que aportan al juego su utilidad, el resto de funciones y librerías simplemente sirven para aportar mayor número de opciones, pero el juego como tal sin estos dos elementos no sería tan intuitivo para el usuario.

## Interfaz de usuario

Implementación de diálogos, menús, controles, etc. Hojas de construcción, procesos de callback, eventos generados, etc. Mostrar el aspecto más que el modo de uso que irá en un anexo. Resumir la funcionalidad de la interfaz indicando sus características, posibles alternativas, etc., para un lector programador.

La interfaz de usuario ha sido explicada a fondo en el capítulo anterior, sin embargo aquí vamos a comentar las diferentes formas que tenemos de mandar avisos y alertas al usuario.

En primer lugar, cuando estamos colocando una pieza, es decir, estamos trabajando con el '*BuilderCube',* y se encuentra en una posición en la que no la podremos soltar, esta adopta un color rojizo, tal y como se muestra en las Figuras 6 y 8 del capítulo 3. Esto se ha implementado de esta forma, puesto que el ser humano, una de las asociaciones que hace con el color rojo, es la prohibición.

Para no tener que cambiar el color de la pieza en cuestión y mostrar un tono rojizo pero partiendo de su color actual, en el material que utilizamos hay una propiedad llamada '*emissive*', que permite al material emitir una color, más allá de su color base, si el '*BuilderCube'*, está bloqueado y no se debe de poder colocar ahí, el valor de esta propiedad se cambia a un tono rojizo, dando como resultado el que vemos en la escena.

```
builderCube.material.emissive.setHex( 0xff0000 );
```

El parámetro que le pasamos es un color en hexadecimal, con el valor de rojo al máximo y el resto al que tienen por defecto que es 0.

En segundo lugar tenemos avisos que aparecen en la mitad superior del elemento canvas, esto sucede cuando estamos editando de forma incorrecta una pieza. En total hay 3 posibles avisos, 2 errores y una alerta.

Cuando desplazamos una pieza desde una posición válida a otra pero que se quede anclada, es decir, que esta acoplada tanto por arriba como por abajo, al usuario le saldrá el siguiente mensaje (Figura 27), puesto que según las especificaciones solo podemos eliminar una pieza cuando tiene o la parte inferior o la superior libre.


**Figura 27: Mensaje de alerta.**

En segundo lugar tenemos los dos mensajes de error, que se muestran para informar al usuario el motivo porque el que no se ha llevado a cabo la última acción que han realizado, ya que si no llegaría a ser molesto y frustrante para el usuario.

Uno de estos errores viene cuando intentamos borrar una pieza que no puede ser borrada, mostrando el siguiente mensaje:

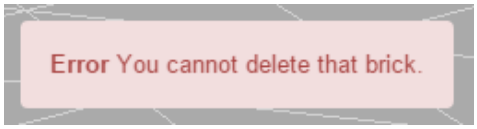



Figura 28: Mensaje de error 1.

El otro error aparece en cuanto desplazando una pieza que ya ha sido colocada en la escena, se intenta mover a una posición en la que no puede estar, y el mensaje es el siguiente:


Figura 29: Mensaje de error 2.

Por lo que respecta a la implementación del resto de la interfaz, como se ha dicho anteriormente es un videojuego en navegador web, por lo tanto la interfaz se basa en el propio HTML5 y CSS3, más allá de esto a la hora de ocultar y mostrar alertas, menús, etc., también se utiliza jquery.

Vamos a poner ejemplos de llamadas a funciones y registro de eventos que se utilizan en el proyecto.

El caso más sencillo es en el propio fichero HTML, dentro de una etiqueta, utilizando el campo *onclick* hacer una llamada a una función escrita en JavaScript.

```
<div class="options-item Game-mode" id="OBJECT" onclick="setGameMode(1)">Editor</div>
```

En el ejemplo superior, tenemos la etiqueta *div*, y vemos en azul el valor *onclick,* que en HTML sirve para lanzar eventos cuando en esta etiqueta se hace clic, por lo tanto cuando en la interfaz hacemos clic en esta casilla, llamamos a la función '*setGameMode*' pasándole como argumento un 1, lo que indica que vamos a establecer el modo de juego a Editor.

Como segunda forma de detectar los clics del usuario en nuestra interfaz, utilizamos los callback de jquery, un ejemplo para esto lo podemos encontrar en la llamada a la función para recalcular el tamaño del suelo.

```
$("#options-update-canvas").click(function (e) {
    $("#menu-options").hide();
    recalculateFloorSize();
});
```

El campo que clicamos es de '*recalculate floor data*', identificado tal y como aparece en la imagen, como '*#options-update-canvas',* y posteriormente registramos este callback y realizamos las funciones que queramos, en este caso ocultamos el menú que había abierto y llamamos a la función '*recalculateFloorSize()*' que hace lo propio.

Finalmente vamos a comentar otro recurso muy utilizado en la interfaz, que es ocultar y mostrar los menús utilizando jquery.

A la hora de ocultar una etiqueta se hace como sucede con el menú del ejemplo anterior, y en cuanto a mostrarlo tampoco tiene mucha diferencia, ya que se utilizan funciones de la librería jquery, a la hora de mostrar un menú sería:

```
$("#menu-options").show();
```

Sin embargo, hay momentos en los que queremos mostrar la etiqueta solamente por una duración, como por ejemplos con los avisos y alertas, esto se hace utilizando la función de JavaScript llamada *SetTimeout()*, y que funciona de la siguiente manera.

```
$("#alert-errorMsg").show();

setTimeout(function(){
    $("#alert-errorMsg").fadeOut(1000);
}, 2000);
```

En primer lugar mostramos la alerta, y con la función de *setTimeout* lo que hacemos es crear un retraso en la ejecución de todo el código que está dentro según el segundo parámetro que le metamos, en este caso es 2000 ms. Esto provoca que a la función para ocultar la alerta se demore 2 segundos, adicionalmente, en este ejemplo se le hace desaparecer a la alerta con un *fadeOut* que provoca que vaya reduciendo su trasparencia hasta ser totalmente trasparente y desaparecer. En caso de no pasar ningún valor a la funcón *fadeOut,* sería instantánea pero el 1000 que le introducimos hace que la animación dure 1 segundo, y se pueda percibir este desvanecimiento.

## Implantación y dependencias

Llegamos a la parte que nos muestra la gran ventaja de esta tecnología, no requiere, en absoluto, ninguna instalación previa. La aplicación está alojada en una web, por lo tanto para lanzarla deberíamos abrir la página web y listo. En ningún es necesario un cambio de librerías, una migración ni linkeo de nada.Esta es la principal ventaja que obtenemos al utilizar WebGL.

En cuanto a las dependencias, deberíamos de tener un hardware que soporte OpenGL 2.0 u OpenGL ES 2.0, es difícil tener algo que no soporte esto, pero en definitiva es un requerimiento y así se ha de mostrar. Dicho esto hay algunos casos en los que directamente nuestra tarjeta gráfica tiene desactivado WebGL pero es posible activarlo, siempre y cuando lo soporte, por supuesto.

Como segunda dependencia tenemos el requerimiento de un navegador que tenga implementado WebGL y lo soporte, las versiones actuales de Google Chrome, Mozilla Firefox, Internet Explorer, Microsoft Edge, Opera y Safari (estos dos últimos en las versiones para Mac) soportan esta tecnología, por lo que si no funciona en ellos, es porque están desactualizados y simplemente habría que hacerlo.

## Conclusión

En este capítulo hemos hablado sobre los detalles de la implementación y que tecnologías hemos utilizado para la interfaz y su implementación. Como datos destacables es la importancia de representar las piezas de construcción, en primera instancia, como cubos.

Adicionalmente, recordar los raycasting, las líneas imaginarias que nos sirve para realizar interacción entre objetos, incluyendo entre el ratón del usuario y las piezas del entorno.

Destacar el uso de HTML5, CSS3 y jquery para realizar las acciones dinámicas de la interfaz, y JavaScript para ciertas funciones que muestrar alertas y avisos con cierto retraso.

Finalmente destacar el dato más importante, que no es otro que el no requerir instalación alguna, ya que al fin y al cabo es una página web y se muestra abre como tal. SI nombrar, el requerimiento de utilizar un navegador actualizado y un hardware compatible con OpenGL 2.0, con no tener un ordenador bastante antiguo, estos requisito los deberíamos solventar sin problema alguno.

# Capítulo 5. Pruebas y Resultados

Debido a que es un proyecto en el que se pretende obtener un producto final y comercial, y no ser un proyecto de investigación, la mejor forma de comprobar el correcto funcionamiento de la herramienta, es probarla.

Lo primero de todo ha sido probar el correcto funcionamiento en las plataformas disponibles, por lo tanto se ha lanzado y realizado pruebas en los siguientes navegadores: Google Chrome, Mozilla Firefox, Internet Explorer, Microsoft Edge y Safari.

En todos los navegadores ha funcionado satisfactoriamente, para probar el funcionamiento se han realizado las mismas pruebas en todos, siguiendo los siguientes pasos:

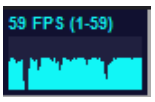
1. Lanzar el videojuego, crear una escena y editarla.
2. Guardar el fichero y crear una nueva escena.
3. Jugar en esta segunda escena.
4. Cargar el fichero guardado anteriormente.
5. Comprobar el correcto funcionamiento del proyecto cargado, así como las funciones de la interfaz, etc.

Adicionalmente, diferentes personas han probado el juego, cada uno en su navegador favorito, y en todos los casos el resultado ha sido satisfactorio. E incluso un niño de 4 años ha jugado, con supervisión, pero sin mayores problemas, consiguiendo crear su propia escena.

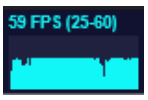
Como segunda prueba se ha creado la misma escena en todos los navegadores nombrados anteriormente y se ha medido el rendimiento de la herramienta en *frames por* segundo. Los resultados en cada navegador han sido los siguientes:

Google Chrome

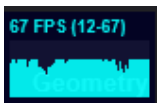

Mozilla Firefox



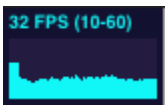

Internet Explorer 11+




Microsoft Edge




Safari (MAC)

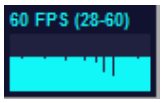



Vemos que el rendimiento es similar para los navegadores, lo que demuestra que es una tecnología que ya está correctamente aplicada en estos, también podemos descartarlos como fuente de errores. Sin embargo al observar los resultados de Safari en Mac, concluimos que el principal factor que puede afectar al rendimiento es solamente el hardware del ordenador.

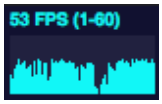
En este segundo test ponemos a prueba con 3 tarjetas gráficas diferentes los resultados obtenidos en el mismo navegador, en este caso Mozilla Firefox.

- nVidia GTX 760


- AMD R9 290




- Intel HD Graphics 3000




Como podemos observar, en este test sí que se pueden observar notorias diferencias, la mejor tarjeta gráfica la AMD R9 290, obtiene de largo los mejores resultados.

Para finalizar, algo destacable de todas las mediciones realizadas son los picos de bajada que encontramos, que se deben en todos los casos a editar piezas grandes, los cálculos para comprobar si estas piezas están bien colocadas y demás comprobaciones son bastante costosos de realizar.

# Capítulo 6. Conclusiones

## Objetivos alcanzados

Una vez visto todo lo que se ha conseguido en los capítulos anteriores podemos concluir que los objetivos descritos en el capítulo 1, han sido conseguidos. La tarea de realizar un videojuego de construcción 3D para navegador web ha sido completada, y comprobado su correcto funcionamiento.

En cuanto a la consecución de los subobjetivos, debido a que eran las etapas que tenían que ir desarrollándose para conseguir desarrollar por completo el software, desde la creación de cubos dinámicamente hasta la implementación de la página web.

## Líneas abiertas

Como posible trabajo futuro no tienen limitaciones establecidas, al igual que con la mayoría de juegos online, es sencillo hacer ampliaciones del contenido, y siempre podríamos añadir nuevas funcionalidades.

Entrando más en materia, la ampliación más directa sería dotar a las piezas de estructura propia, es decir crear una clase pieza que contenga a la que ya tenemos actualmente, y de esta forma en lugar de consultar las medidas de la geometría actual, consultaríamos los valores de altura, anchura y profundidad, extraídos de la caja de inclusión, por lo que sería sencillo implementar piezas de diferentes formas, como por ejemplo, hacer esquinas, y curvas.

Mirando más allá, se podría utilizar la característica de Three.js para fusionar mallas, de tal forma que si todas las piezas de la escena estuvieran como mínimo pegadas a otra, es decir todas conectadas entré si, sería sencillo crear un modelo 3D para posteriormente imprimirlo. E incluso imprimir las piezas por separado para poder construirlo en la realidad.

## Valoración

En mi experiencia ha sido muy gratificante trabajar con este tipo de tecnologías, viendo como poco a poco se iba creando un producto que poder mostrar.

El tiempo empleado ha sido de aproximadamente dos meses, sin incluir el desarrollo de la memoria y el total de líneas de código asciende a las 9000 líneas aproximadamente.

# Bibliografía


1. Khronos Group, “WebGL, OpenGL ES 2.0 for the Web” [02 de Septiembre de 2015]. Disponible en la Web: https://www.khronos.org/webgl/

2. Marc Antonijoan, “3D en la web, ¿qué tecnología usar?” [02 de Septiembre de 2015]. Disponible en la Web: http://blogs.salleurl.edu/dtm-media-technology/3d-en-la-web-%C2%BFque-tecnologia-usar/

3. Juan Andrés Núñez, “Aprendiendo: WebGL, gráficos 3D interactivos para la Web” [02 de Septiembre de 2015]. Disponible en la Web: http://wmedia.es/aprendiendo-webgl-graficos-3d-interactivos-para-la-web/

4. Three.js, “getting started” [24 de Julio de 2015]. Disponible en la Web: http://threejs.org/docs/index.html#Manual/Introduction/Creating_a_scene

5. The jQuery Foundation. “API” [22 de Julio de 2015]. Disponible en la Web: http://api.jquery.com/

6. Three.js “documentation” [24 de Julio de 2015]. Disponible en la Web: http://threejs.org/docs/

7. Bos, Bert “Description of all CSS specifications” [04 de Septiembre de 2015]. World Wide Web Consortium.

8. Etemad, Elika “Cascading Style Sheets (CSS) Snapshot 2010” [04 de Septiembre de 2015]. World Wide Web Consortium.

9. Chrome Experiments comunity “WebGL Experiments” [20 de Julio de 2015]. Disponible en la Web: https://www.chromeexperiments.com/webgl

10. Zygote Media Group “Zygote Body” [20 de Julio de 2015]. Disponible en la Web: https://zygotebody.com/

11. Rocksteady Studios “Batman arkham knight” [21 de Julio de 2014] Disponible en la Web: https://www.batmanarkhamknight.com/en_US/batmobile

12. Three.js “Examples-OrbitControls” [02 de Agosto de 2015]. Disponible en la Web: http://threejs.org/examples/#misc_controls_orbit

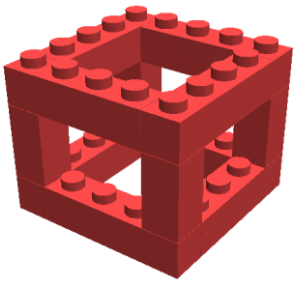
13. Three.js “Examples-TransformControls” [02 de Agosto de 2015]. Disponible en la Web: http://threejs.org/examples/#misc_controls_transform

14. Eli Gray, “FileSaver” [18 de Agosto de 2015]. Disponible en la Web: https://github.com/eligrey/FileSaver.js/

# Apéndice A. Guía del usuario

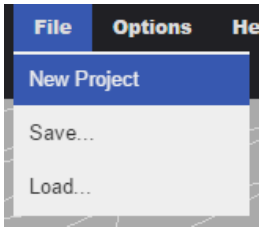
## Ejemplo de uso (tutorial)

Como tutorial de inicio de la aplicación vamos a construir la siguiente figura:



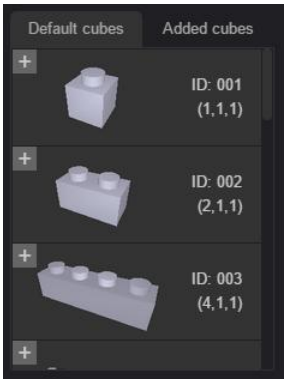
Para ello vamos a ir guiando paso por paso, desde que se carga la página web hasta la propia captura de pantalla de la imagen superior, mostrando la configuración correspondiente, vamos pues a comenzar con el tutorial.

1. En primer lugar, abrimos la página web y nos encontramos con la escena inicial, queremos una escena de 0, por lo que vamos a crear un proyecto nuevo.
2. Para crear un proyecto nuevo hemos de ir a la barra superior y en el menú 'File' darle a 'New Project', tal y como se muestra en la imagen inferior.




Una vez hecho esto, en el menú de confirmación le damos a 'OK'.

3. Ya tenemos la escena en blanco, por lo que ahora vamos a comenzar a construir la figura, vamos a utilizar solamente 2 piezas diferentes, ambas ya vienen por defecto en el juego y por lo tanto no es necesario crearlas, estas piezas són 'ID: 003' e 'ID: 012'. Recordemos que el menú de piezas se muestra como en la imagen inferior, a la parte derecha de la pantalla.



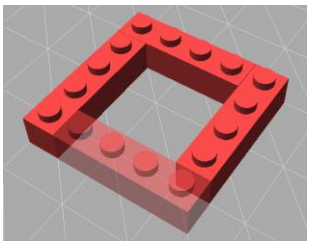
Vamos al menú de piezas para seleccionar la pieza ‘003’, ahora sobre en el ratón se nos muestra una previsualización de esta pieza, de un color azul y un poco trasparente. Antes de colocarla vamos a cambiarle el color.

4. Para cambiar el color, hemos de ir al selector de color, abajo a la derecha y elegirlo, en nuestro caso el color es ‘B53636’, por lo que podemos escribirlo directamente en el campo disponible.


Una vez le hemos cambiado el color a la pieza, vamos a colocarla en una zona más o menos centrada de la plataforma, hacemos clic izquierdo del ratón en una zona que nos guste para colocarla.

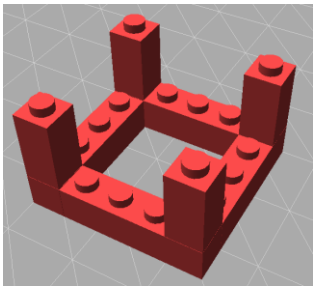
5. Con el fin de crear la vamos hemos de duplicar la misma pieza 4 veces, una forma de hacerlo es ponernos a editar (clic izquierdo sobre ella -> editar) y posteriormente duplicarla, pero dado que es la última que hemos construido, simplemente con volver al modo constructor, se nos volverá a mostrar la pieza que acabamos de construir.

   La disposición de las 4 piezas ha de ser la siguiente:

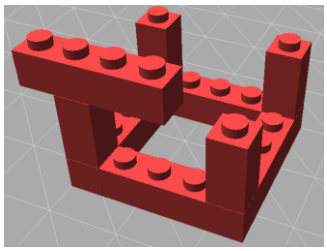


Para rotar las piezas, nos sirve la opción de rotar, en el menú superior, cuyo icono es  , o pulsar la tecla ‘R’ que es el acceso directo.

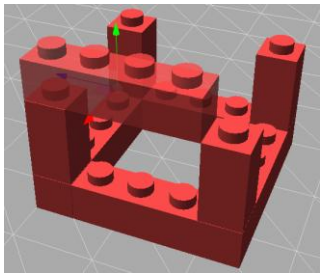
6. A continuación para crear las columnas, hacemos clic en la pieza ‘012’ y la colocamos en cada una de las esquinas de la base.



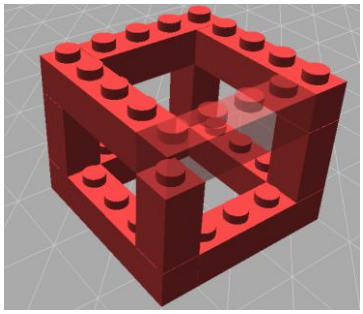
7. Ya solo nos queda añadir las piezas superiores, vamos a ello pues. Volvemos a seleccionar la pieza ‘003’ de la lista, y la colocamos sobre uno de los pilares, veremos que en un principio no queda como queríamos.



Para solucionar esto, hacemos clic izquierdo sobre ella, en el menú que ha aparecido le damos a ‘Edit’ y finalmente desplazamos la pieza, haciendo clic en la flecha del eje en el que la queramos mover y arrastrando.

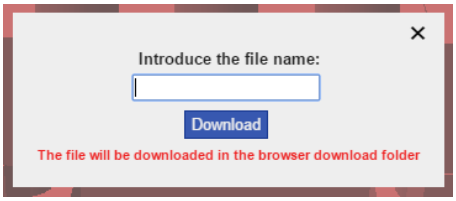


8. Ya que estamos con la pieza seleccionada, en lugar de volver a cambiar de modo, vamos a duplicarla, para ello tenemos la opción en la barra superior  , justo al lado de rotar, o podemos utilizar el acceso directo con la tecla ‘D’.
Para colocar correctamente estas piezas no es necesario editarlas, podemos apoyarnos en la que hemos colocado en el paso anterior, y posicionarlas apoyándonos en ella.



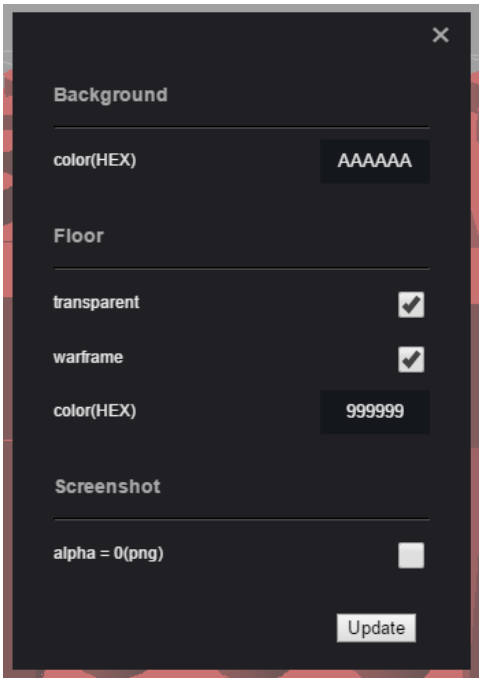
Como podemos observar en la imagen superior la pieza que estamos colocando está bien posicionada, debido a que estamos apuntando con el ratón la pieza de su derecha, y la posición se calcula respecto a esta

9. Una vez hemos terminado la figura, vamos a guardar el proyecto, para ello volvemos al menú ‘File’, mostrado al principio del tutorial, y le damos a ‘Save…’, lo que nos abrirá la siguiente ventana.




En esta debemos escribir, el nombre que queramos darle al proyecto y hacer clic en 'Download' para guardarla en local, la ruta de descarga es la que tenemos puesta en nuestro navegador, y es independiente del videojuego.

10. Tras haber guardado, vamos a hacer una captura de pantalla, tal cual está hecha la del inicio.
    Hemos de ir al menú 'Options' -> 'Configuración…', justo al lado del menú 'File'. Una vez ahí hemos de cambiar las opciones para que al hacer captura de pantalla el fondo sea trasparente, y hacer que el suelo no sea visible. Las opciones quedarán como aparecen en la imagen inferior.




    El resto de opciones las dejamos como estaban por defecto, una vez las hemos cambiado hacemos clic en el botón inferior 'Update' y listo.

11. Finalmente para hacer la captura de pantalla nos vale tanto el icono de la cámara  , como el acceso directo, tecla 'S', se nos abrirá en una ventana la captura realizada, y ya la podemos guardar con el fondo trasparente.