ESCUELA POLITÉCNICA SUPERIOR **UNIVERSIDAD DE CÓRDOBA**

TRABAJO DE FIN DE GRADO

INGENIERÍA INFORMÁTICA

DISEÑO Y DESARROLLO DE MÓDULO DE GAMIFICACIÓN PARA ENTORNOS E-LEARNING BASADO EN LA LIBRERÍA 3D THREEJS

BASICTHREE

MEMORIA

Miguel Morales Carmona

Directores: **Enrique García Salcines**

Agosto, 2016

ESCUELA POLITÉCNICA SUPERIOR **UNIVERSIDAD DE CÓRDOBA**

TRABAJO DE FIN DE GRADO

INGENIERÍA INFORMÁTICA

DISEÑO Y DESARROLLO DE MÓDULO DE GAMIFICACIÓN PARA ENTORNOS E-LEARNING BASADO EN LA LIBRERÍA 3D THREEJS

BASICTHREE

MEMORIA

Miguel Morales Carmona

Directores: **Enrique García Salcines**

Agosto, 2016

INFORME DEL DIRECTOR

Firma del Director del Proyecto:

Firma del alumno y autor del Proyecto:

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1. DEFINICIÓN DEL PROYECTO

1.1. Introducción

Gamificación [1] (*gamification* en el ámbito anglosajón) es el empleo de mecánicas de juego en entornos y aplicaciones no lúdicas con el fin de potenciar la motivación, la concentración, el esfuerzo, la fidelización y otros valores positivos comunes a todos los juegos. Se trata de una nueva y poderosa estrategia para influir y motivar a grupos de personas utilizando la estrategia de aprender jugando.

El crecimiento exponencial del uso de videojuegos en los últimos años ha despertado el interés de expertos en comunicación, psicología, educación, salud, productividad -y casi cualquier área de actividad humana- por descifrar las claves que hacen del videojuego un medio tan eficaz.

Además, se ha experimentado un gran avance en las tecnologías web como el lenguaje HTML5, que incluye componentes nativos como el Canvas o WebGL, que permiten diseñar en 2D y 3D en la propia web, sin necesidad de herramientas muy complejas, caras o *plugins* adicionales cuya compatibilidad dependía en gran medida del navegador.

Sin embargo, ahora es cuando comienzan a vislumbrarse las potentes aplicaciones y juegos 3D que pueden hacerse utilizando estas herramientas. El único problema es que al ser tan recientes estos desarrollos existen muy pocas herramientas o librerías que faciliten el trabajo a alguien no experto en programación, como puede ser el creador o diseñador de un curso de *e-Learning* o un profesor de primaria.

Todo lo anterior, nos ha motivado a desarrollar un módulo de gamificación que pueda utilizarse en cualquier curso *web* de *e-Learning* y que facilite la interacción del usuario a la hora de desarrollar escenarios de aprendizajes basados en juegos.

1.1.1. Ventajas de la gamificación

La gamificación no sólo otorga beneficios en el contexto educativo, como puede ser la escuela, sino incluso en empresas para motivar a sus clientes.

En el contexto educativo o lúdico, las ventajas están presentes en todos sus niveles de educación; desde los niños que cursan infantil, hasta los universitarios. Los motivos son los siguientes:

- Aumenta la motivación del alumno, lo que es desencadenante de muchas otras mejoras o ventajas.
- Mejora el comportamiento en clase, en ocasiones como consecuencia directa de la existencia de motivación. Esto conlleva un mayor trabajo y esfuerzo.
- Comprenden la relevancia de lo que están aprendiendo ya que lo están aplicando en ese mismo momento.
- Los objetivos y la estructura del temario quedan mejor reflejados.
- No sólo pueden ser actividades de adquisición de conocimiento, sino también de refuerzo como actividad complementaria a las clases.

- Aumenta la conexión social y las relaciones humanas en el caso de juegos competitivos.
- Sentimiento palpable de progresión de conocimiento.

En cuanto a las empresas, en relación con sus clientes, tienen las siguientes ventajas al crear sus productos con cierta dosis de gamificación:

- Incentivo para completar tareas que son de interés para la empresa. Como, por ejemplo, completar de rellenar un perfil de usuario o recoger opiniones.
- Atracción de nuevos clientes o usuarios. Es innegable que los sistemas de puntos de algunas empresas está siendo todo un éxito.
- Ayuda a conseguir un *feedback* por parte del usuario. La gamificación hace que aumente la atención del usuario, lo que produce más información para el *feedback*.

1.2. Objetivos

Con el desarrollo de este proyecto se pretende desarrollar una librería que permita el desarrollo de aplicaciones *e-Learning* de forma fácil y al alcance de todos. Los objetivos específicos son los siguientes:

1) Estudio minucioso de la librería ThreeJS, ya que será la librería "padre" de la que queremos extraer y simplificar sus funcionalidades. Para ello:

a. Se deberá obtener dominio de la librería ThreeJS y tener conocimiento de su uso.

b. Captar y distinguir las funcionalidades que nos serán de interés y desechar aquellas que son más complejas e innecesarias para la aplicación que se busca.

2) Encapsulación, modularización y abstracción de las funcionalidades que necesitamos. Es importante mantener la simplicidad.

3) Las funcionalidades que son buscadas son las enumeradas y descritas a continuación:

a. Presentación y preparación de escena: Se tratarían de funciones para controlar el tamaño de la escena, cámara, objetos y su posición, texturas…etc.

b. Módulo de control de movimientos.

c. Módulo detector de colisiones.

d. Módulo avatar: Personificación del usuario.

e. Módulo de configuración *e-Learning*: Situar puntos de acción en el entorno que den lugar a algún tipo de pregunta o interacción (sopa de letras, test, emparejamientos…etc.).

4) Estudiar, comprender y utilizar la lógica de cualquier escena para el desarrollo y la aplicación de estos módulos.

5) Probar y evaluar a abstracción buscada. Realización de aplicación *e-Learning* que ponga a prueba el funcionamiento y refleje, a su vez, simplicidad de código.

2. ANTECEDENTES

A continuación vamos a detallar algunos de los antecedentes más relevantes en cuanto a herramientas y librerías web que permiten el diseño de aplicaciones de gamificación utilizando tecnologías web.

Stage3D [3] es la API que Adobe puso en funcionamiento en el año 2011. Podía usar aceleramiento por GPU, y hacía uso tanto de WebGL como de DirectX. Es una librería de bajo nivel, por lo que era muy tedioso usarla. En cuanto a gamificación, se usaba Adobe Gaming SDK. Su uso disminuyó drásticamente ya que en la versión Flash Player 11.8 se eliminó el aceleramiento por GPU, lo que hizo que las aplicaciones desarrolladas con Stage3D se volviesen muy pesadas.

De igual modo, por parte de Java, surgió Java3D [4] en el año 2004. Utiliza Direct3D u OpenGL. Era más ligero que Stage3D pero la gamificación es más difícil de lograr y es una librería de alto nivel, lo que conlleva una dificultad de aprendizaje menor.

Todas ellas suponen un peso computacional añadido a la aplicación web ya que necesitan de motores gráficos y aplicaciones externas para hacerlas funcionar.

Por otra parte, recientemente han surgido herramientas o librerías más livianas que utilizan los últimos avances de Javacript y HTML5, en concreto sus componentes Canvas y WegGL que son lenguajes enfocados a la renderización de formas 3D y 2D en espacios web. Hay numerosas librerías de este tipo, pero nos centraremos en aquellas que tienen una gran potencial de gamificación y que utilicen tanto WebGL como Canvas.

Blend4Web [5] surgió en 2016, pero ha crecido muy rápidamente por el gran grupo de desarrollo tras él. Es una librería muy potente pero con mucha complejidad. Podemos descargar su editor, que es considerado uno de los más potentes de este tipo de APIs. Con él podemos realizar el proceso de gamificación. Sin embargo, el problema esencial que encontramos con Blend4Web es que tiene una versión del SDK gratuito y otra de pago, donde la versión gratuita es muy limitada.

ThreeJS [6] es una librería de Javascript que se basa en los componentes Canvas y WebGL de HTML5, para simplificar en gran medida la programación en estos entornos. No obstante, es necesario cierto grado de conocimiento de programación y con una curva de aprendizaje bastante plana que indica el grado de dificultad.

BabylonJS [7] es bastante más similar a ThreeJS, en cambio, tiene unas diferencias muy importantes. BabylonJS necesita crear obligatoriamente los elementos Canvas previamente en la plantilla HTML5 para posteriormente manipularlo mediante JavaScript. Además, tiene una serie de limitaciones cuando se crean diversas animaciones y escenas 3D.

En las últimas versiones de ThreeJS, se han incluido numerosas novedades como la capacidad de importar diseños de herramientas de diseño 3D como 3DSMAX y Unity, lo cual la convierte en una

herramienta muy atractiva para su utilización en juegos de aprendizaje en *e-Learning* y es por la que hemos optado finalmente en nuestro proyecto.

La idea por tanto, es desarrollar una librería por encima de ThreeJS en nivel de abstracción con las funcionalidades necesarias para el desarrollo de juegos y aplicaciones de aprendizaje y educativas, buscando cumplir una serie de objetivos que a continuación se detallan en el siguiente apartado.

2.1. Canvas y WebGL

Canvas es un elemento de HTML5 que es capaz de generar gráficos por medio de scripts y animarlos. Mientras el navegador lo soporte, podrá reproducirlos. Fue desarrollado originalmente para Safari.

Solamente es capaz de generar gráficos 2D pero mediante utilidades y librerías como la que se utiliza en el presente proyecto, se pueden recrear espacios de 3 dimensiones con uso de múltiples vistas renderizadas conjuntamente.

En cambio, WebGL fue generado para Mozilla, utilizando OpenGL. Por lo tanto, genera 3 dimensiones sin necesidad de utilidades, tan sólo con aceleración por hardware. Es una ventaja frente a Canvas, pero su codificación es mucho más compleja al trabajar sobre GPU, por lo que es necesario el uso de bibliotecas de Javascript para utilizarlo. Una de las más utilizadas hoy en día, es ThreeJS.

Por lo tanto, ambas son soportadas por ThreeJS, simplificando WebGL que es de bajo nivel y aumentando las posibilidades de Canvas 2D. Las equivalencias en cuanto a funcionalidades vienen recogidas en la siguiente tabla.

Canvas API	WebGL equivalent	Three.js framework equivalent
scale	texImage2D + modify vertices	Camera
rotate	texImage2D + modify vertices	Camera
translate	texImage2D + modify vertices	Camera
transform	texImage2D + modify vertices	Camera
clearRect	clear+viewport, or teximage2d	Textures
fillRect	clear+viewport, or teximage2d	Textures
strokeRect	teximage2d	Textures
path	teximage2d+stencil + vertex paths	Camera + objects
fillText	teximage2d+stencil + vertex paths	Camera + objects + textures
strokeText	teximage2d+stencil + vertex paths	Camera + objects + textures
drawImage	teximage2d	textures
createImageData	fill buffer of teximage2d in CPU application	CPU application code
getImageData	readpixels	readpixels
putImageData	fill buffer of teximage2d in CPU application	fill buffer of teximage2d in CPU application
	+ programmable shaders (vertex, fragment)	
	+ offscreen buffers	
	+ efficient 3D representation of depth	
	+ efficient POINTS rendering	
		+ Convenience methods - Materials (sprite ..)
		+ Convenience methods - Lights
		+ Convenience methods - Scenes

Tabla 1. Comparativa de utilidades de Canvas 2D, WebGL y la librería THREEjs

En cambio, Canvas 2D ha sido más utilizada para crear diseños inteligentes y elegantes de páginas webs ya que, en muchos casos, reduce el peso y el tiempo de carga al ser dinámico y sin tener que descargar del servidor ninguna imagen. En cuanto a gamificación, Canvas 2D es capaz de crear aplicaciones fáciles sin necesidad del asiduo y tradicional Java. Sin embargo, al incluir tridimensionalidad, empieza a suponer una carga mayor de cómputo frente a otras opciones.

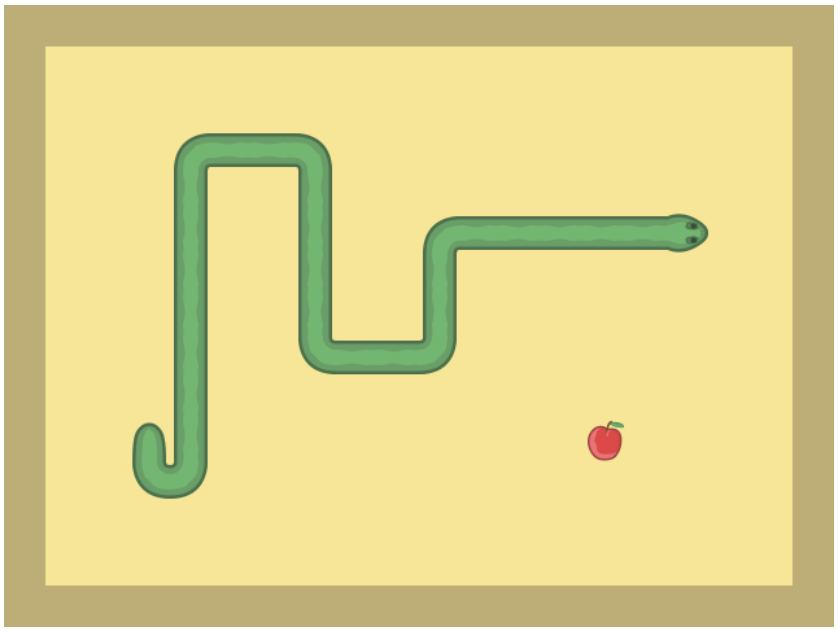

Ilustración 1. Juego cásico de Snake hecho con Canvas 2D.

WebGL puede cargar diseños tridimensionales desde archivos y también crearlos en tiempo real y personalizarlos, tal y como fue preprogramado. Son aplicaciones muy ligeras hasta que se le añade lógica de control densa pero son la mejor opción para 3 dimensiones si se puede disponer de esta tecnología. Tiene una alta compatibilidad con programas de diseño (3DSMax, Unity…etc.) lo que aumenta su abanico de posibilidades de manera importante.

Ilustracion 2. Ejemplo de creación en tiempo real de una caja con WebGL.

2.2. Tecnología SVG

Esta tecnología merece un apartado propio para explicar por qué no se tiene en cuenta en el presente proyecto. SVG son las siguientes *Scalable Vector Graphics.* Al igual que Canvas está pensando para gráficos en 2 dimensiones, aunque también puede simular la tridimensionalidad pero con peores resultados. Sus limitaciones se encuentran, sobretodo, en la deficiencia de rendimiento al añadir muchos elementos porque estos se añaden directamente sobre el documento en formato HTML5.

ThreeJS incluye esta tecnología, sin embargo, no está muy desarrollada debido a sus limitaciones obvias. Es soportada por todos los navegadores actualmente. Aunque, utilizarlo, simplificaría mucho el resultado final ya que no haría falta renderizar iterativamente, pero no merece la pena debido a que las funcionalidades serían muy deficientes.

Los mejores enfoques para SVG son otros muy distintos [8]. Un ejemplo de uso de SVG puede ser guías de recorridos dinámicas en la web. [9]

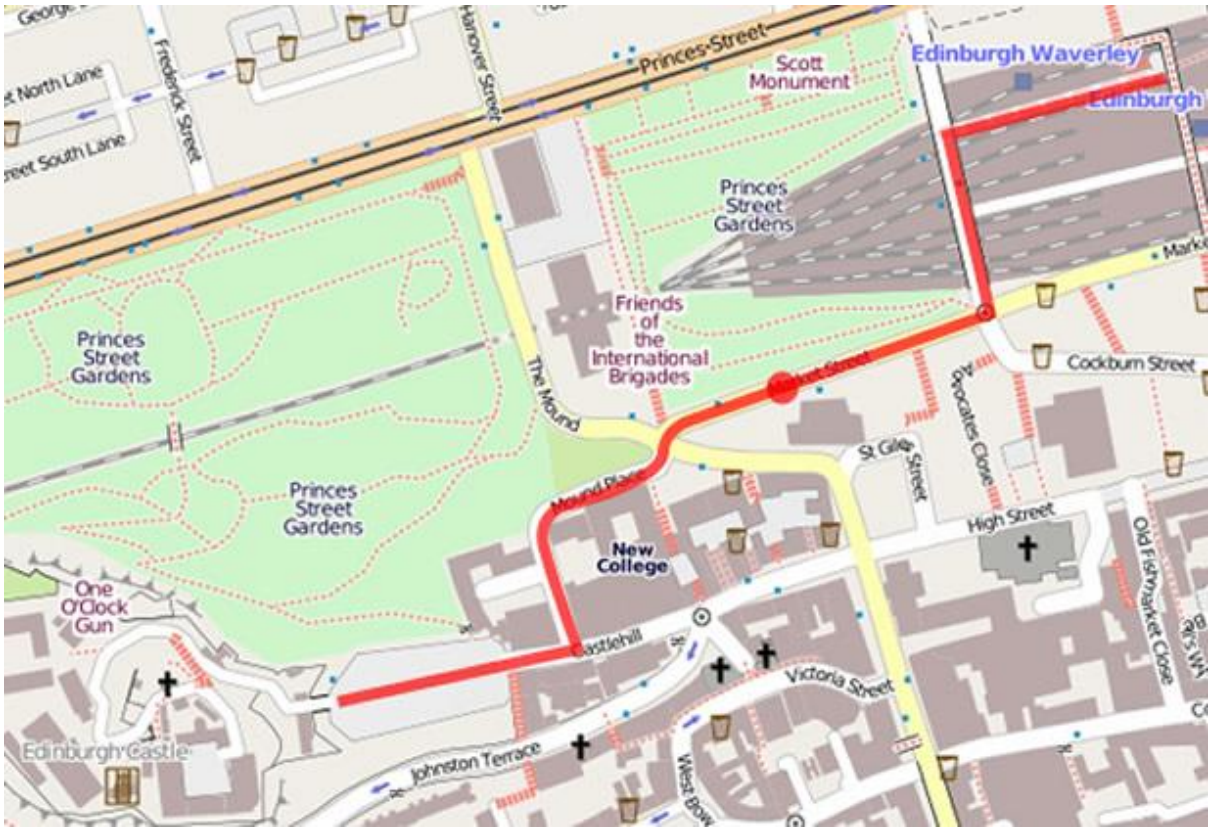

Ilustración 3. Trazado dinámico de recorridos con SVG.

2.3. THREEjs: Desarrollo de la comunidad

La comunidad de ThreeJS, como ya se ha dicho anteriormente, es una de las más grandes e importantes en cuanto a recreación de 3 dimensiones. Es capaz de abstraer la dificultad de usar Canvas para 3D cuando originalmente es para 2D y, también, se encarga de abstraer el lenguaje de bajo nivel que utiliza WebGL como intermediario con la GPU.

Generar una escena sencilla con unos pocos conocimientos de lo que se está creando es muy sencillo, pero es más difícil realizar funcionalidades más complejas debido a los conocimientos que se deben tener de geometría, matemáticas, modelado y, en ocasiones, física.

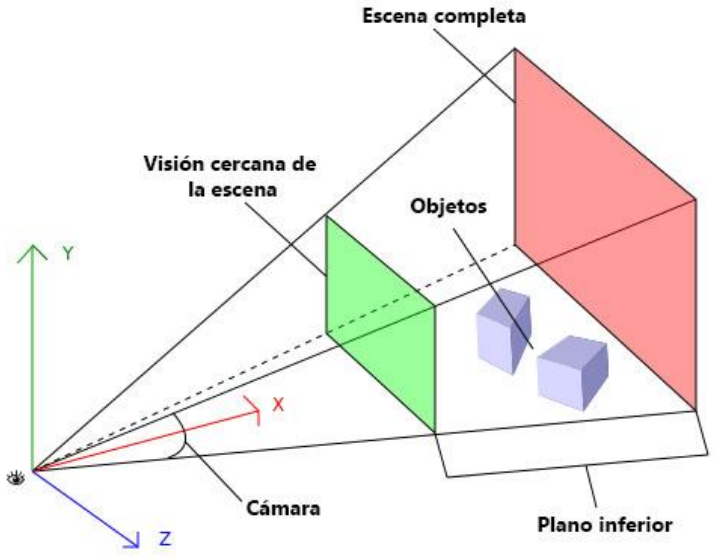

Ilustración 4. Lógica de escena básica.

En aproximadamente dos años de vida, los programadores que utilizan ThreeJS han creado muchas funciones de gran utilidad para el diseño de aplicaciones, utilizando las herramientas que la librería ya les daba, que en gran parte son básicas y primitivas, pero definitivamente son las justas y necesarias para desarrollar cualquier funcionalidad.

Estas funcionalidades creadas por la comunidad, en parte, son complejas y difíciles de usar por programadores inexpertos, ni qué decir tiene por usuarios que no sean programadores. Esto sin tener en cuenta, que hay que enfrentarse a las peculiaridades de cada aplicación. No obstante, la comunidad es una fuente de información potente para crear aplicaciones con controles de movimiento, colisiones, interacción...etc. Todo lo necesario para la gamificación de la que hablábamos en la introducción.

Con esta librería, el límite para alguien que sabe programar es su imaginación y el tiempo que disponga para darle vida.

Ilustración 5. Juego The Aviator [10]

2.4. ¿Cuántos desarrolladores y programadores hay?

Esta pregunta es interesante para dar sentido al proyecto, ya que está pensado para personas que saben programar muy poco o no saben programar. Por lo tanto, se formula la pregunta ¿Es realmente necesario este proyecto?

Tomaré como referencias datos recogidos por el IDC en 2014 a nivel mundial [2]. Recogen información de quiénes saben programar, tanto por hobby como profesionalmente.

		En 2014
Desarrolladores de Software/Programadores	Desarrolladores profesionales	11.005.000 personas
	Desarrolladores por hobby	7.534.500 personas
	Total	18.539.500 personas

Tabla 2. Estadística de programadores en el mundo en 2014.

Con esta información y si, redondeando, hay 7.000.000.000 habitantes en el mundo, podemos concluir que alrededor de un 0.003% de las personas del mundo, sabe programar. De los cuales sólo un 0.001%, proveniente de los desarrolladores profesionales, podrá usar la librería ThreeJS en su totalidad sin problemas.

Por lo tanto, que se encuentre un profesor, de cualquier rama de conocimiento o nivel educativo, que sepa programar para hacer una actividad lúdica y gamificada para sus alumnos es muy poco probable, incluso a nivel mundial. Los cálculos son algo imprecisos, ya que en los habitantes en el mundo hay incluidos bebés, niños…etc., que desvirtúan la estadística, aunque hablando en cifras tan grandes y dispares, no cambiaría la conclusión.

Si tenemos en cuenta tan sólo a la población activa, que son 3.384.000.000 en 2014, entonces los programadores (profesionales o por *hobby*) son 0.005% del total, de los cuales 0.003% son profesionales. Aun así, como se decía anteriormente, es poco probable que se dé la situación ya que hay 5 personas de cada 100.000 que saben programar.

3. RESTRICCIONES

En este capítulo se definirán las restricciones existentes en la fase de diseño y que condicionan la estructura y actitud de las funcionalidades. Las restricciones pueden ser factores de dos tipos:

- **Factores dato**: son aquellos que no pueden ser modificados durante el transcurso del proyecto, como pueden ser el presupuesto económico, la duración, las fechas de entrega…
- **Factores estratégicos**: son variables de la fase de diseño que da varios caminos posibles al ingeniero. Según la elección del mismo, podrá variar el resultado final, por lo que es importante analizar las opciones presentes.

3.1. Factores dato

En el desarrollo de este proyecto se van a tener los siguientes factores dato:

- Usar JavaScript. JavaScript es un lenguaje interpretado y orientado a objetos.
- Usar la librería ThreeJS. ThreeJS es una librería para la creación de aplicaciones 3D en web.
- Implementar las funciones que sean de utilidad para la gamificación y desechar las demás.
- Mantener la simplicidad de uso, aunque para la acción de renderización iterativa se deberá usar el procedimiento habitual de ThreeJS.

3.2. Factores estratégicos

Los factores estratégicos que se tienen en cuenta en este proyecto son los siguientes.

3.2.1. Entorno

Por tratarse de una herramienta destinada a ser usada por personas inexpertas en el ámbito de la programación, se considera oportuno que la librería sea formada por funciones simples y secuenciales de fácil aprendizaje.

La aplicación se desarrollará mediante Sublime Text, compatible tanto con Windows como con GNU/Linux, y usado para gran diversidad de lenguajes de programación. Muy útil en nuestro caso que utilizaremos JavaScript para la herramienta y HTML para ponerla a prueba.

3.2.2. Estructura

La librería deberá tener los cuatro módulos descritos anteriormente, aunque será de vital importancia estudiar el grado de solapamiento idóneo que deben tener algunos casos entre ellos para mantener la personalización y el uso sencillo.

Se harán lo más flexibles posibles para alargar la vida útil del proyecto con posibles futuras versiones y aplicaciones alternativas.

4. ESPECIFICACIÓN DE REQUISITOS

El proyecto que se desea abordar consiste en el diseño y desarrollo de una librería que sirva como herramienta para crear juegos lúdicos de manera sencilla y al alcance de todos.

4.1. Requisitos funcionales

Los requisitos funcionales de la librería describen lo que ésta debe hacer.

Requisito funcional 1. La librería debe contener una serie de funciones que creen una escena inicializada y añadir o crear elementos a antojo. Dichas funciones recibirán información por parte del diseñador de la aplicación para describir cómo quiere que sea su escena, su iluminación, su cámara y los elementos que hay en ella.

Requisito funcional 2. La librería debe contener una serie de funciones que doten a un elemento de movimiento y de una cámara apropiada para su función. A éste lo denominaremos Avatar y será la personificación del usuario final en la aplicación creada con la librería.

Requisito funcional 3. La librería debe contener una serie de funciones que detecten colisiones al existir movimiento y actuar en consecuencia, ya sea parándose o ejecutando una acción. Debe poder usar las acciones predefinidas por el módulo *e-Learning* o por funciones hechas por el creador de la aplicación.

Requisito funcional 4. La librería debe contener una serie de funciones por defecto que puedan ser atractivas de usar para la gamificación como pueden ser preguntas o explicaciones.

4.2. Requisitos de información

Los requisitos de información de la librería describen lo que debe almacenar la misma para su correcto funcionamiento.

Requisito de información 1. Debe almacenar la escena, la cámara y el *render* ya que son el centro neurálgico de la escena 3D.

Requisito de información 2. Se mantendrá constancia de todos los elementos de la escena. ThreeJS internamente ya se encarga de almacenar su posición por lo que al guardar los elementos, bastaría.

Requisito de información 3. La información necesaria sobre el avatar, su movimiento, su posición y cuál elemento le corresponde.

Requisito de información 4. El estado actual de la escena en todo momento para actuar en consecuencia ante cualquier evento. Esto recoge posición de todos los elementos, en especial el Avatar, si entran en escena o salen y también la actualización de la cámara orbital.

Requisito de información 5. Es necesario que se almacene toda la información necesaria para las aplicaciones *e-Learning* bajo un patrón claramente definido.

Requisito de información 5.1. En el caso de las explicaciones se almacenará, por cada explicación; una cabecera, un pie y un cuerpo.

Requisito de información 5.2. En el caso de las preguntas test se almacenará, por cada test y por cada pregunta del test; pregunta, respuesta, explicación (si se requiere) e imagen (si se requiere).

Si se añaden más aplicaciones, la estructura de la información será similar pero con las peculiaridades de cada una.

4.3. Requisitos de interfaz

Los requisitos de interfaz son aquellos que explican cómo debe ser la interacción del usuario con la librería.

Requisito de interfaz 1. Se exportará la librería con el método habitual de inclusión de script de JavaScript.

Requisito de interfaz 2. El usuario podrá acceder a las funciones mediante código JavaScript de manera intuitiva y sencilla, sin necesidad de bucles ni condiciones (necesariamente). A excepción del proceso de renderización iterativa, que es obligatoria para el proceso de actualización.

Requisito de interfaz 3. El modo de utilización de funciones debe ser secuencial, de manera que normalmente, primero se utilicen unas funciones y luego otras. El orden lógico sería; inicializar la escena, añadir elementos, establecer contenido de las aplicaciones, definir avatar. Aunque esto debe ser flexible en medida de lo posible.

5. RECURSOS

6.1. Recursos humanos.

- Autor: Miguel Morales Carmona.
- Director: D. Enrique García Salcines.

6.2. Recursos materiales.

6.2.1. Hardware.

El desarrollo del proyecto se realizará haciendo uso del siguiente hardware:

- Ordenador portátil Lenovo Thinkpad,
    - Procesador Intel Core i3 a 2.40 GHz.
    - Memoria Ram de 4 GB a 1067 MHz, DD3.
    - Disco Duro de 500 GB.

6.2.2. Software.

El desarrollo del proyecto se realizará haciendo uso del siguiente software:

- Sistema operativo: Windows 10.
- JavaScript.
- Sublime Text.
- ThreeJS.
- HTML5.
- AutoCAD, 3DSMax o Unity.
- Navegadores Webs: Safari, Mozilla Firefox e Internet Explorer.

6. DISEÑO DEL SISTEMA

En este capítulo se describe cómo ha sido diseñada la librería.

6.1. Introducción

Un requisito fundamental de este proyecto es mantener la simplicidad. ThreeJS tiene sus propias clases/objetos que ya recogen todo lo necesario para que el sistema funcione por lo que añadir nuevas clases sería añadir complejidad. Eso supondría romper el principal requisito del proyecto y añadir dificultad a la curva de aprendizaje de quien use la librería.

Por lo tanto, la librería sólo se conformará de funciones complejas con un alto contenido de lógica de control, que se utilizarán mediante una simple línea o, a lo sumo, dos líneas. Para ello las funciones y métodos se han creado a medida las unas para las otras y que así funcionasen al unísono.

6.2. Diseño de los módulos

El diseño de los módulos que contienen y organizan, de manera lógica, las funciones de esta librería se especifica en este apartado.

6.2.1. Módulo de Inicialización y creación de escena

Este módulo es el núcleo de todos, sin él no podrá funcionar ninguno de los consecuentes. Lo componen una serie de funciones que dan al usuario todo lo necesario para recrear la escena y colocar los elementos a su antojo en ella. Se describen a continuación las funciones que lo conforman:

- Inicializar la escena en el espacio tridimensional y mostrarlo.
- Añadir un elemento a la escena.
- Cargar una figura tridimensional desde un fichero JSON externo y colocarla en escena.
- Seleccionar un elemento de la escena.
- Funciones propias para dar distintos tipos de material a las figuras.
- Iluminar la escena de un determinado color.

6.2.2. Módulo de Control de movimiento

Este módulo se encarga de determinar un Avatar, que será la personificación del jugador, y controlar su movimiento a lo largo de la escena con una cámara apropiada.

- Determinar cuál es el Avatar y activarlo como tal para que sea controlado.
- Crear los controles para la cámara.

6.2.3. Módulo de Detección de colisiones

Este módulo se encarga de detectar colisiones cuando el Avatar se desplace o rote. Es su única función. Este módulo está previsto que se solape con el módulo de *e-Learning* y con el módulo de control, pero tiene una función muy diferenciada y compleja que es:

- Detectar colisiones y actuar en consecuencia según con qué elemento colisione.

6.2.4. Módulo de Aplicaciones e-Learning

Este módulo define una serie de funciones predefinidas que sirven para realizar actividades gamificadas. Sin embargo, el sistema es capaz de aceptar y ejecutar funciones externas de otras APIs u hechas por el programador, por lo que las posibilidades van más allá de este módulo que contiene:

- Lanzador de un Popup con explicaciones.
- Lanzador de un Popup con cuestionario.
- Lanzador de un Popup con preguntas de respuesta corta.
- Lanzador de un Popup para dictados.
- Lanzador de un Popup para copiados.

6.3. Diseño de Procedimientos

En este apartado se detallan los procedimientos que forman parte de la librería desarrollada.

6.3.1. Inicializar la escena

Para inicializar la escena se realizará una serie de configuraciones previas y predefinidas que darán sentido y coexistencia a la aplicación. Aparte de establecer el *renderer* (WebGL o Canvas), el tipo de cámara y la escena, también se crea suelo sobre el que interaccionarán, por defecto, todos los elementos y además volverá responsiva la escena a distinta resoluciones.

El tipo de cámara es de vital importancia saber lo que se quiere, ya que dependiendo de cuál cámara utilicemos hay grandes diferencias, en especial entre la ortográfica y la de perspectiva. En la primera, las cosas se verán justo del tamaño del que son, esté a la distancia a la que esté, por lo que normalmente veremos objetos más pequeños que en la de perspectiva; las experiencias de profundidad de la tridimensionalidad son muy distintas como se puede observar en la siguiente ilustración.

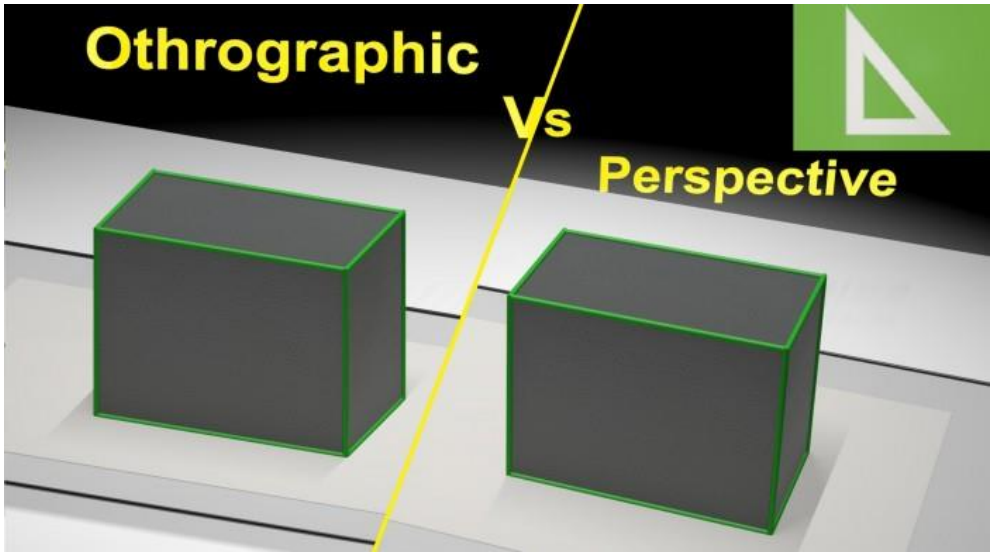

Ilustración 6. Cámara Ortográfica vs Cámara Perspectiva.

El pseudocódigo de este proceso es el siguiente:

Require:
Camera Type. CT
Renderer Type. RT
1: **if** CT is PerspectiveCamera
2: create camera(PerspectiveCamera)
3: **else if** CT is OrthographicCamera
4: create camera(OrthographicCamera)
5: **else if** CT is CubeCamera
6: create camera(PerspectiveCamera)
7: create camera(CubeCamera)
8: **endif**
9: **if** RT is WebGL
10: create renderer(WebGL)
11: **else if** RT is Canvas
12: create renderer(Canvas)
13: renderer.size <- window.width, window.height
14: renderer.color <- White
15: document.add(renderer)
16: window.listener(resize) <- responsive function
17: addElement(/path ground)
18: global variable camera <- camera
19: global variable scene <- scene
20: global variable renderer <- renderer

El diagrama de flujo que describe las acciones de este proceso es el siguiente:

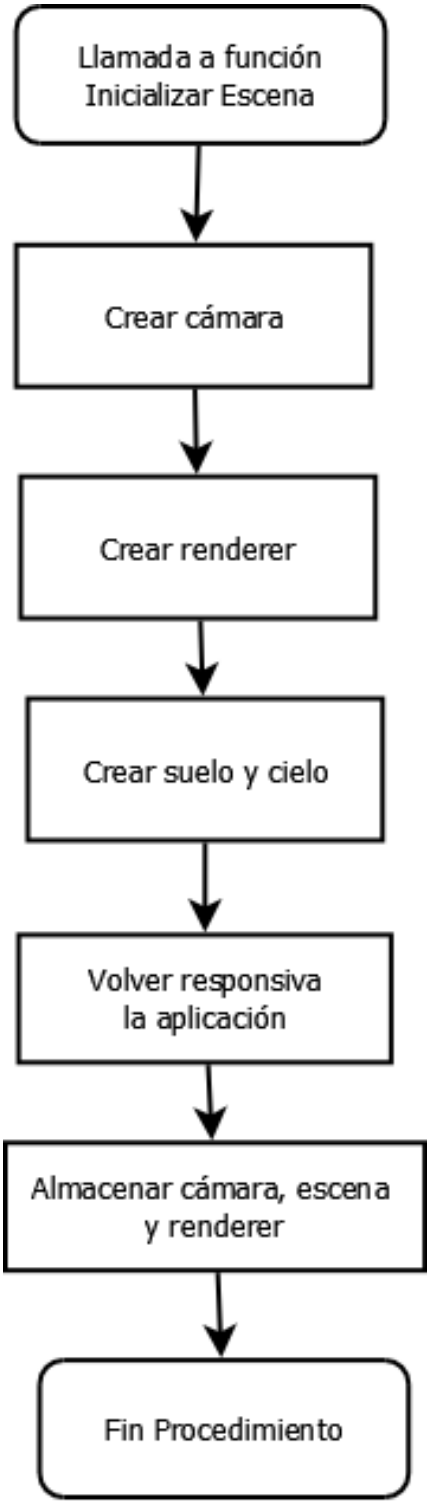

Ilustración 7. Diagrama de flujo del procedimiento Inicializar Escena.

6.3.2. Añadir elemento a la escena

Este procedimiento se encarga de añadir un elemento a la escena de manera automática. Para realizar este proceso es necesario configurar la forma geométrica, el material, la posición sobre el suelo (ya que el eje y es fijo de manera predeterminada) y datos extra para hacer funciones módulos superiores a este, como pueden ser, si es colisionable y cuál es su función en la aplicación.

Con este método y modo de realizarlo, se consigue que en el mismo momento de crear un elemento, ya esté predefinida su función. Esto da secuencialidad al código, lo que añade sencillez. Ya quedarán todos los objetos añadidos tanto al sistema de colisiones como al de aplicaciones, para que se gestione en cuanto se empiece a poner en funcionamiento mediante el sistema Avatar.

El pseudocódigo de este proceso es el siguiente:

```
Require:
        Geometry Form. GF
        Material. M
        X Position. XPoint
        Z Position. ZPoint
        Collidable. C (boolean)
        Function. F.
1: create Mesh(GF, M)
2: set position(XPoint, -215+mesh.height, YPoint)
3: if C is true
4:      Global Variable arrayCollidables <- Mesh
5:      Global Variable arrayFunctions <- F
6: if F is Explication
7:      Initialize Explication
8: else if F is Quiz
9:      Initialize Quiz
10: else if F is Question
11:     Initialize Question
12: return Mesh
```

El diagrama de flujo que describe las acciones de este proceso es el siguiente:

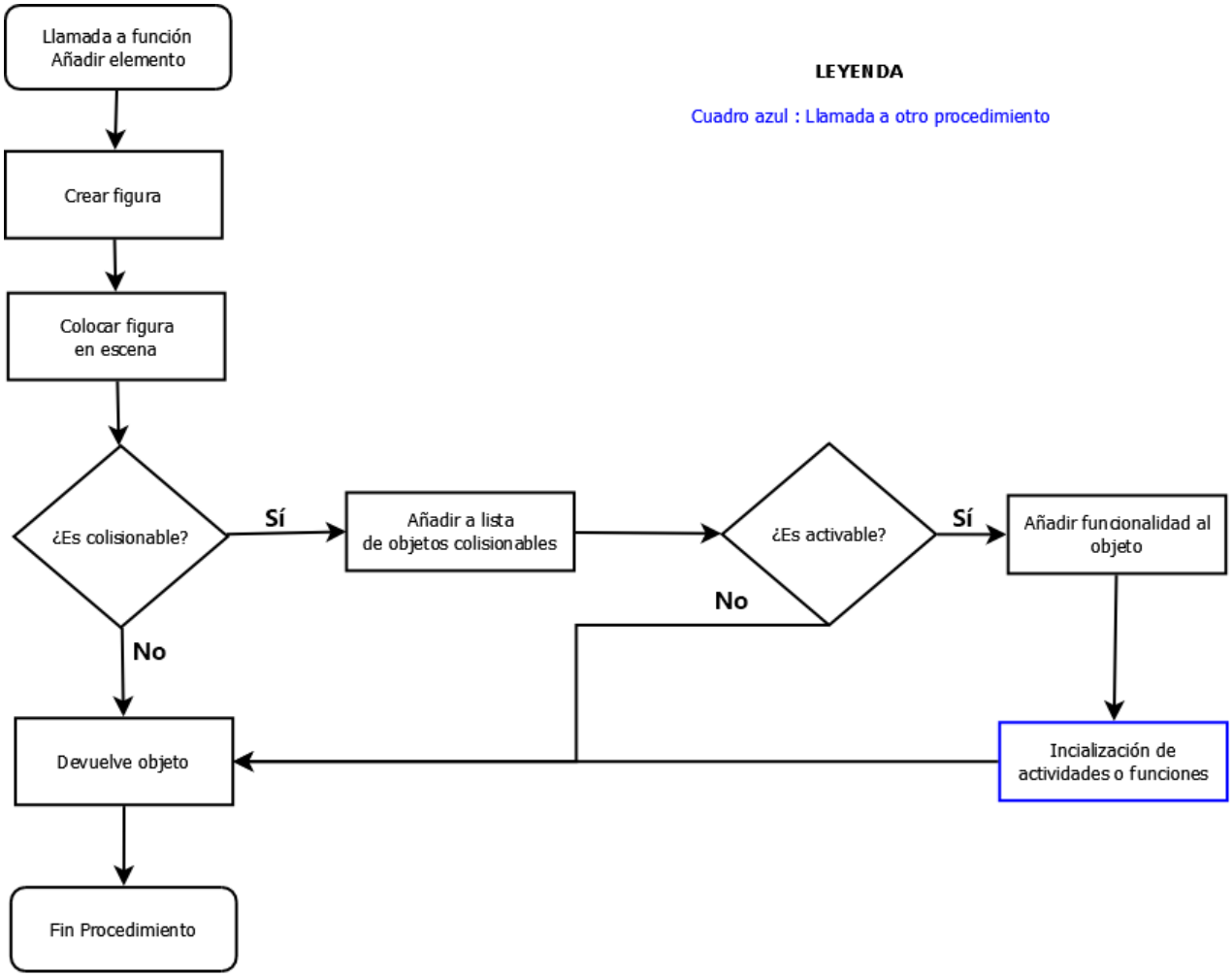

Ilustración 8. Diagrama de flujo del procedimiento Añadir Elemento.

6.3.3. Cargar figura desde archivo JSON

Este método se encarga de añadir un elemento a la escena, de igual manera que añadir elemento a escena, pero con sus peculiaridades. Se utiliza un *loader* ya existente en ThreeJS y que funciona muy eficientemente. Tal es la eficiencia de este método de la librería, que si el modelo es pequeño, es más rápido computacionalmente, cargarlo desde archivo que crearlo.

Para cargarlo tan sólo es necesario el URL o *Path* donde se encuentre, y un nombre identificativo para poder acceder posteriormente a él con su método correspondiente. Es útil y necesario el nombre pues el método tiene un sistema *callback* que no permitirá añadir funciones o identificarlos como colisionables directamente, por lo tanto, será necesario una serie de pasos extra para llevarlo a cabo con uso de este parámetro.

El pseudocódigo de este proceso es el siguiente:

Require:
URL JSON. UJ.
Name ID. ID.
X Position. XPoint
Z Position. ZPoint
Collidable. C (boolean)
Function. F.
1: Create loaderJSON.
2: geometry, materials <- loader(UJ)
3: create Mesh(geometry, materials)
4: **if** C is true
5: Global Variable arrayCollidables <- Mesh
6: Global Variable arrayFunctions <- F
7: **if** F is Explication
8: Initialize Explication
9: **else if** F is Quiz
10: Initialize Quiz
11: **else if** F is Question
12: Initialize Question
13: **return** Mesh

El diagrama de flujo que describe las acciones de este proceso es el siguiente:

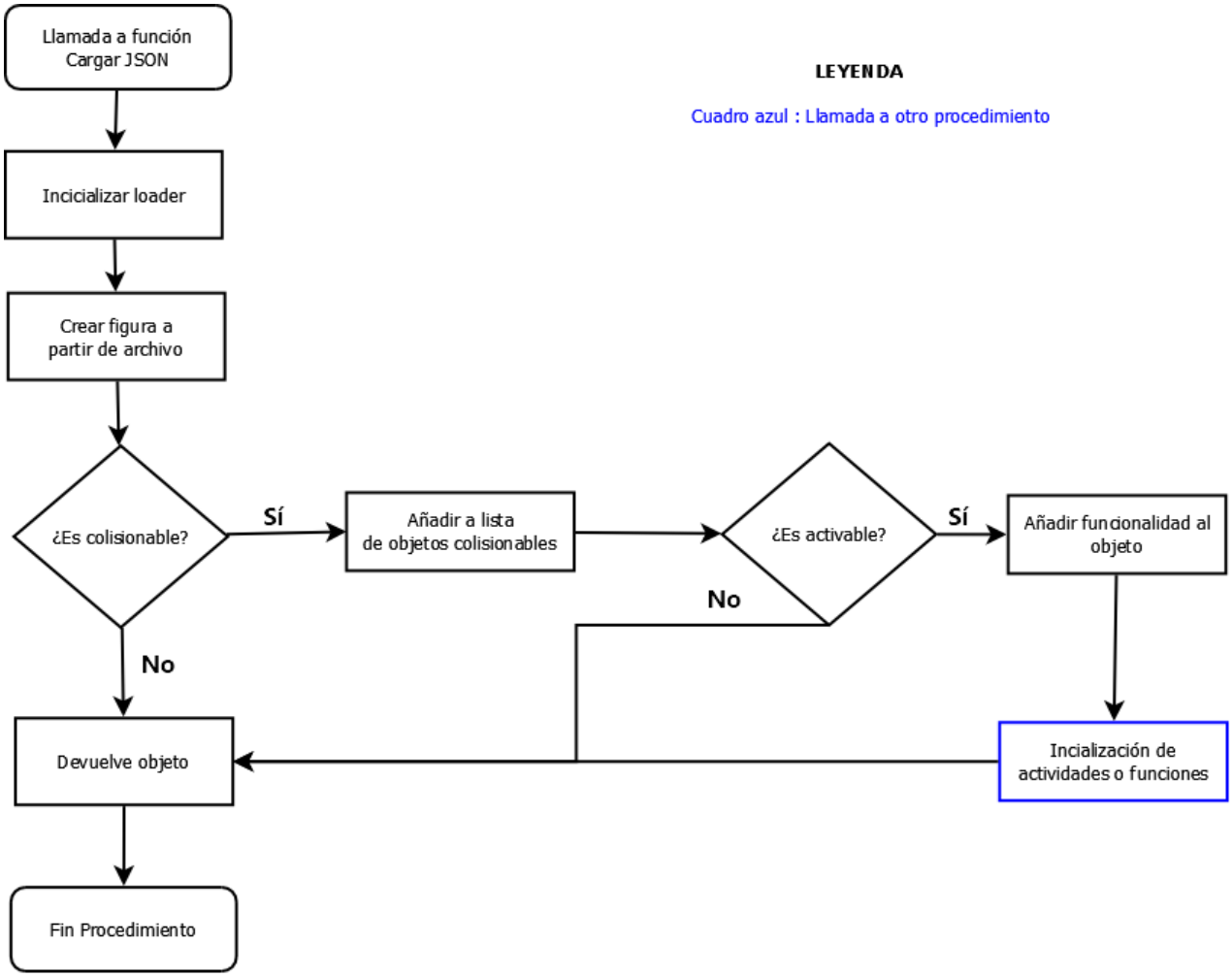

Ilustración 9. Diagrama de flujo del procedimiento Cargar desde archivo JSON.

6.3.4. Selección de objeto en escena

Este método no es esencial, pero si quizás útil para desarrollar aplicaciones propias, por lo que es de interés este *get* que devuelve un elemento identificado por un nombre previamente nombrado como en la carga de JSON aunque sólo lo detectará una vez renderizada la escena.

El pseudocódigo es el siguiente:

Require:
Name ID. ID
Get Object By Name. GOBN {ID}
1: **return** GOBN(ID)

El diagrama de flujo que describe las acciones de este proceso es el siguiente:

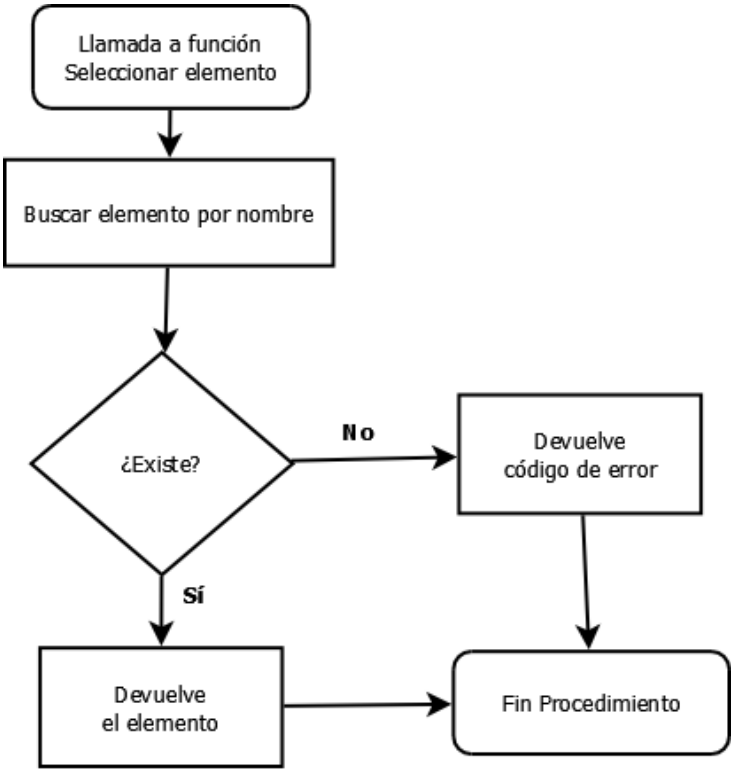

Ilustración 10. Diagrama de flujo del procedimiento Selección de elemento.

6.3.5. Materiales

Este método, al igual que los dos siguientes, es puramente estético, pero de vital importancia para captar la atención del usuario final. En caso de que sea sombreado o brillante, recibe como argumento un color creado predeterminadamente en la librería para no necesitar un conocimiento de hexadecimal o si el usuario inexperto que utilice la librería no sabe dónde buscar dicho código. No obstante, los usuarios experimentados podrán seguir usando estos códigos hexadecimales para una mayor personalización. En caso de que el material sea colorido, no necesita parámetros ni argumentos ya que el color es aleatorio

Si el material es brillante, interactúa con la luz mediante los brillos. Si es sombreado, mediante las sombras. En cambio, si es colorido, no necesita de la iluminación.

Este método llama al método de la iluminación en caso de que aún no exista. Esto es debido a que sin iluminación no se ve el material brillante o sombreado.

En la siguiente imagen podemos ver ejemplos de los tres tipos de materiales básicos; colorido, sombreado y brillante respectivamente.

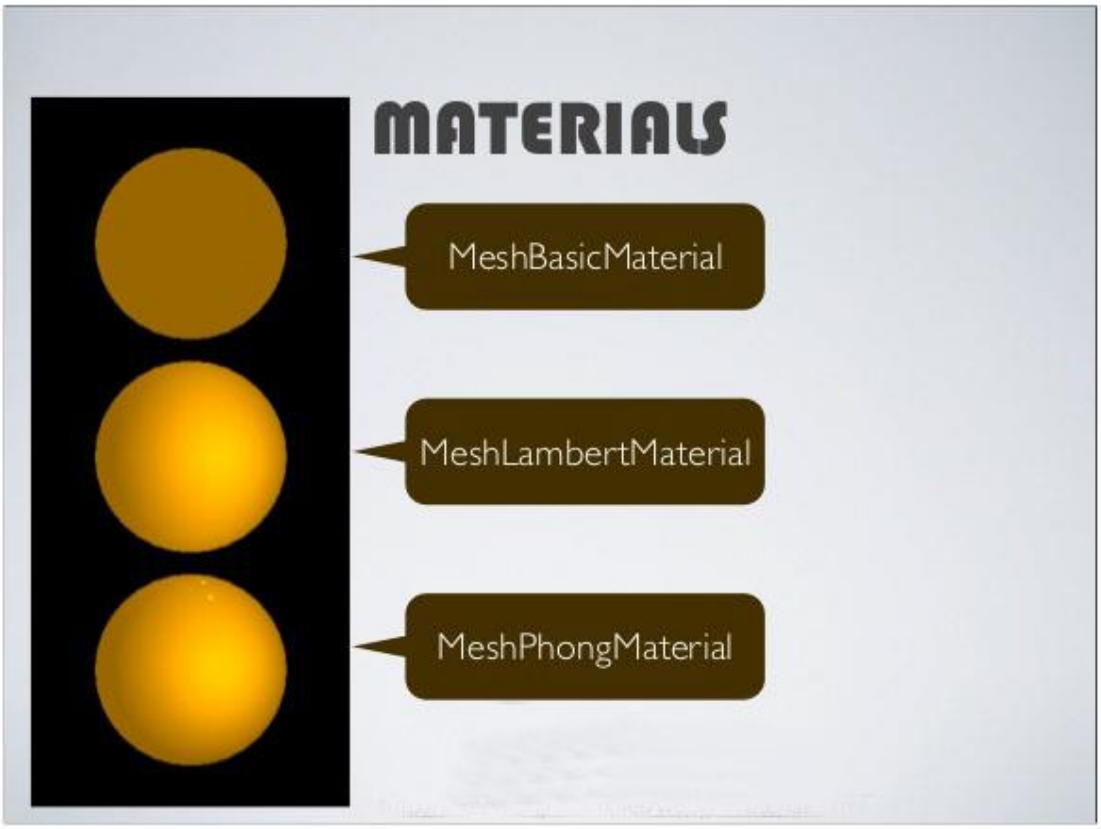

Ilustración 11. Tipos de materiales.

El pseudocódigo es el siguiente para cualquiera de los tipos:

Require:
    Color Material. CM
1: **if** CM is null **and** type is not colorful
2:    CM <- Default Color
3: **if** Variable Global iluminación is null **and** type is not colorful
4:    Método iluminación()
5: material <- create material(CM, shiny/shaded/colorful)
6: **return** material

El diagrama de flujo que describe las acciones de este proceso es el siguiente:

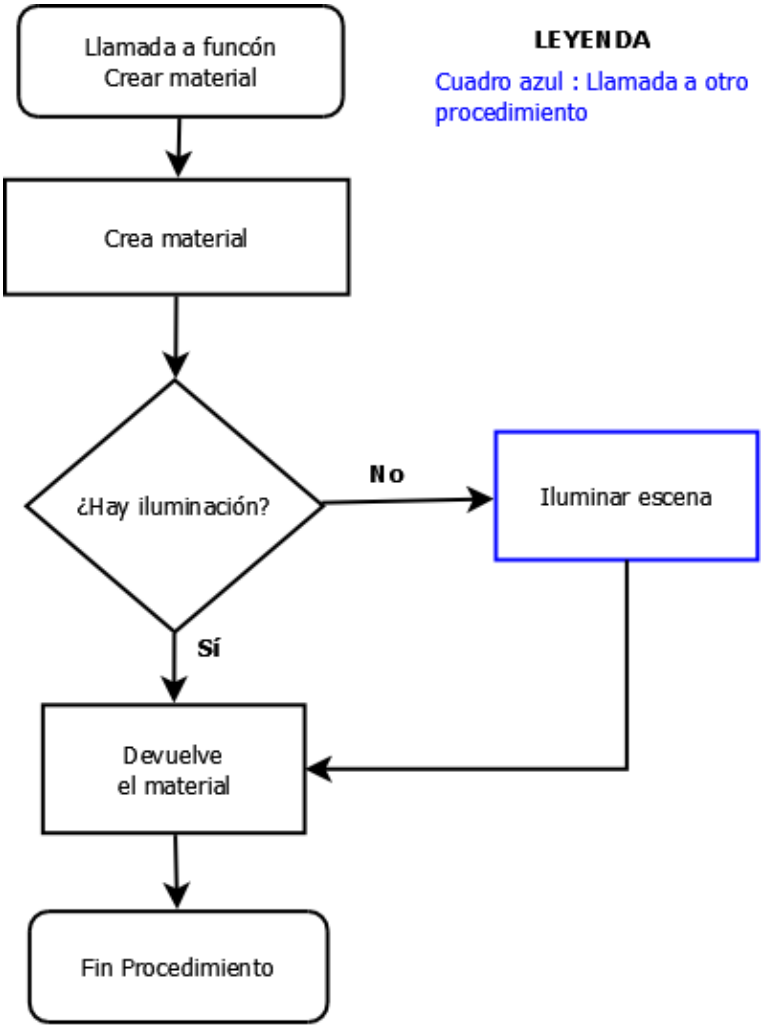

Ilustración 12. Diagrama de flujo del procedimiento Crear material.

6.3.6. Iluminación

La funcionalidad de la iluminación, como ya se ha nombrado con anterioridad, es una función básica ya que en la mayoría de los casos, dota de visión a la escena. Puede determinarse el color de la iluminación y el sistema es de interés que guarde dicha iluminación por si quiera manipularla en sus propias aplicaciones.

El pseudocódigo es el siguiente para cualquiera de los tipos:

Require:
Color luminación. CI
1: **if** CI is null
2: CI <- Default Color
3: Variable Global PointLight <- create Iluminación(CI)

El diagrama de flujo que describe las acciones de este proceso es el siguiente:

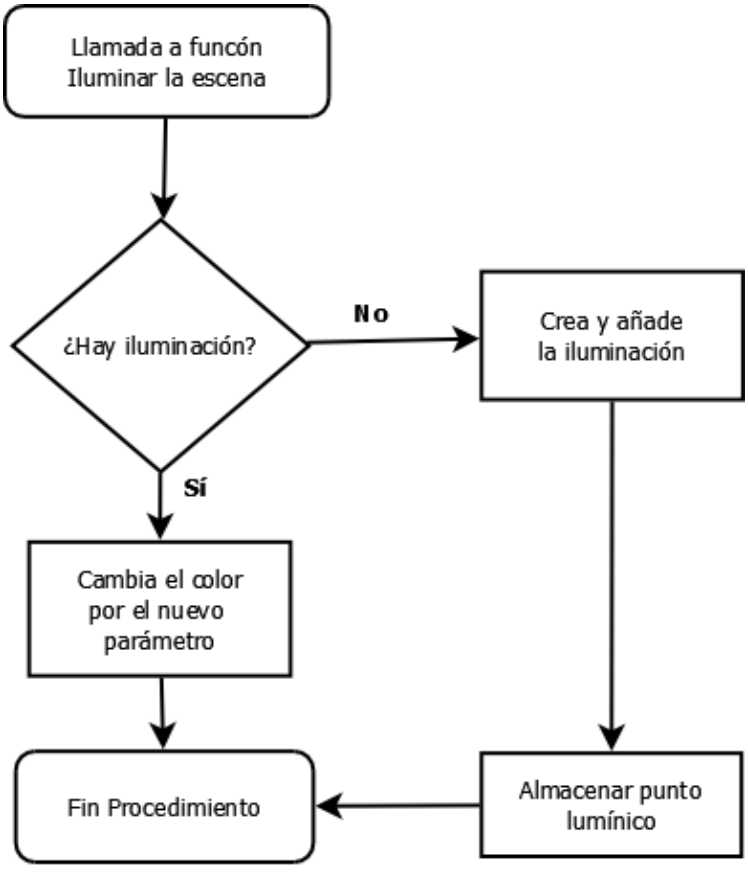

Ilustración 13. Diagrama de flujo del procedimiento Iluminación.

6.3.7. Sistema Avatar

Crear el Avatar y dotarlo de vida basta con una sola función o método que lo único que necesita saber es qué objeto será el que personificará al usuario. Esto se enlazará con funcionalidades posteriores como la detección de colisiones.

El Sistema Avatar se encargará de controlar los movimientos del mismo, comprobando continuamente su situación. Es una funcionalidad que debe lanzarse de manera iterativa junto al renderizado de la aplicación para que así se refresque con frecuencia (tiempo real).

Por defecto está configurado para funcionar mediante dos desplazamientos distintos, rotación y también un reseteo de posición. Todo mediante el teclado.

- Los dos tipos de desplazamiento son global y condicionado.
  - Global se mueve según los ejes x, y, z. Se realizará con las teclas AWSD.
  - Condicionado se mueve según la orientación del Avatar, así se ve afectado por los giros. Se realiza con las flechas del teclado ←↑→↓.
- La rotación se realiza mediante las teclas QEFZ. Es un giro condicionado, no existe el giro global, ya que lo que se rota es el eje de abscisas del Avatar.
- El reseteo volverá a la posición de inicio del Avatar. Este valor de posición es una variable global **no constante**, lo que implica que se puede manipular por el programador en aplicaciones externas. Puede ser útil para los famosos *check points* de los juegos. Se realiza mediante la tecla X.

Todo el sistema de Avatar está controlado por *flags*, ya que permitirá que todos estos movimientos se activen o desactiven a la hora de comenzar, estando activados movimiento condicionado y reseteo, por defecto. En adhesión a estos *flags*, también existe un *flag* que controla que no se pueda continuar en una dirección en la que existe una colisión.

La cámara continuamente seguirá al Avatar, gracias a la cámara de control que creará la funcionalidad que se expondrá su diseño a continuación.

Este es el pseudocódigo del Sistema Avatar:

```
Require:
        Object Avatar. OA.
        Conditioned Movement. CM.
        Global Movement. GM.
        Rotate Flag. RotF.
        Reset Flag. ResF.
1: if first execution
2:      Global Variable resetPosition <- Position(OA)
3:      CenterCamera(OA)
4:      Global Variable Avatar <- OA
5: create keyboard listener:
6:      testCollision()
7:      if key is key arrow and no collision and CM is true
8:              move condionated(distance)
9:      if key is awsd and no collision and GM is true
10:             move global(distance)
11:     if key is {QEFZ} and RotF is true
```

```
12:                 rotate( grade )
13:         if key is X and Res F is true
14:                 Avatar.position <- Global Variable resetPosition
```

El diagrama de flujo que describe las acciones de este proceso es el siguiente:

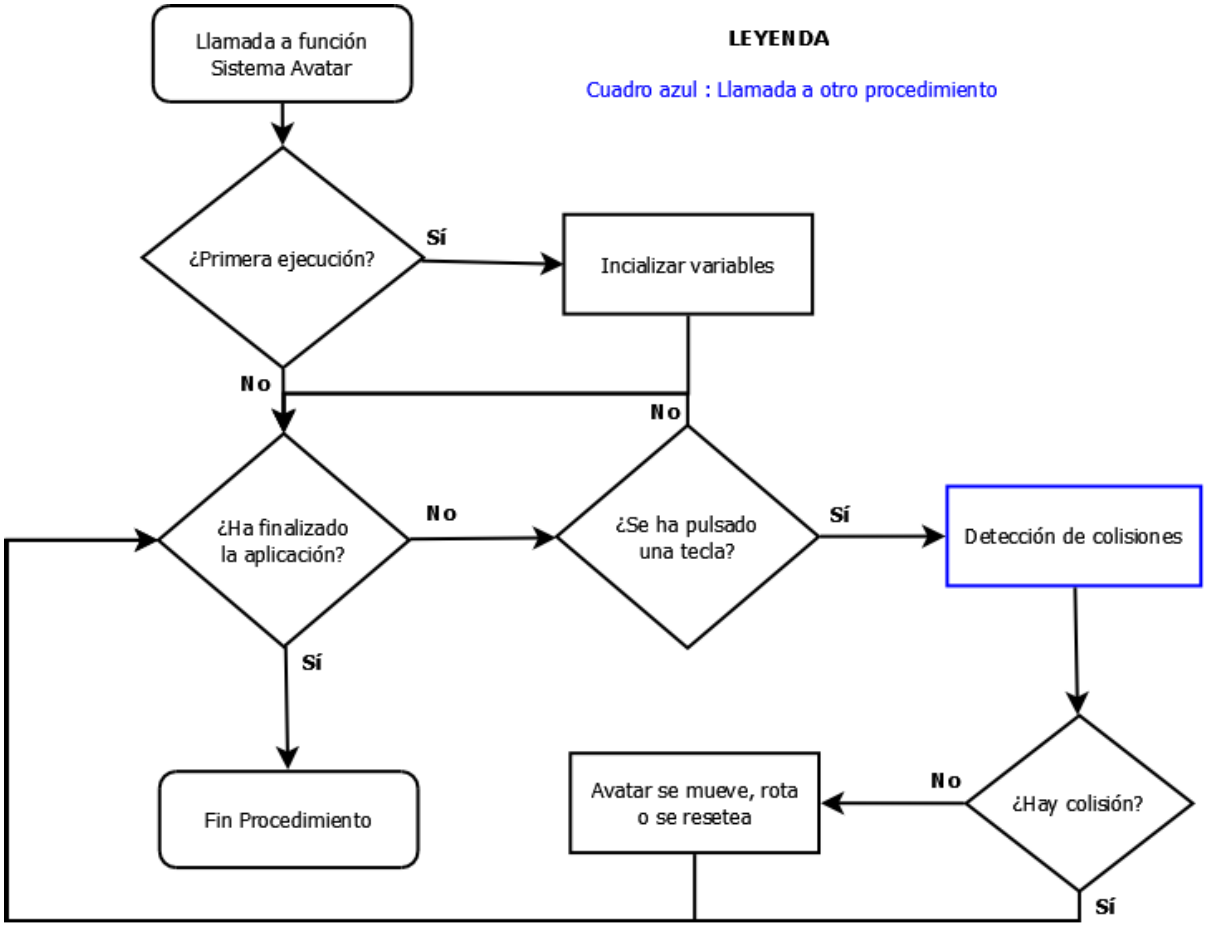

Ilustración 14. Diagrama de flujo del procedimiento Sistema Avatar.

6.3.8. Controles de cámara

Esta funcionalidad es básica en la gamificación para poder ver el entorno que te rodea y decidir qué hacer. No necesita parámetros ni ningún tipo de configuración extra, tan sólo crearla y controlarla a placer. Aunque si posteriormente se desea configurar o personalizar, se podrá mediante la clase de ThreeJS que ya está plenamente funcional desde las últimas revisiones de la librería.

El pseudocódigo es el siguiente:

```
Require:
        Nothing.
1: controls <- create orbitControls
2. return controls
```

El diagrama de flujo que describe las acciones de este proceso es el siguiente:

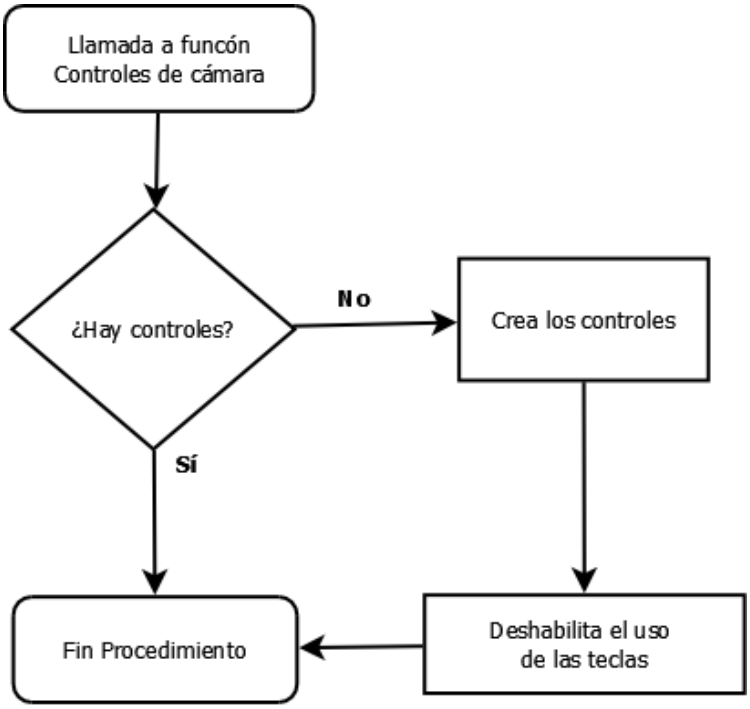

Ilustración 15. Diagrama de flujo del procedimiento Controles de cámara.

6.3.9. Detección de colisiones

Esta es la función más compleja de la librería. No sólo se encarga de detectar si existe colisión con los objetos colisionables sino que si estos tienen una función, la activan.

La detección de colisión sigue una técnica de *Raycasting*. El *raycasting* mide la distancia del objeto actual a todos los demás. Al ser cuerpos con volumen, y en ocasiones irregulares, hay que realizar el *raycasting* desde todos sus vértices. Si hay muchos vértices, la necesidad de cómputo gráfico es mucho mayor.

Con uso de la velocidad de desplazamiento, es capaz de preveer una colisión con este *raycasting*. Es decir, si corta un rayo, hay colisión. La ilustración demuestra gráficamente el funcionamiento de esta técnica.

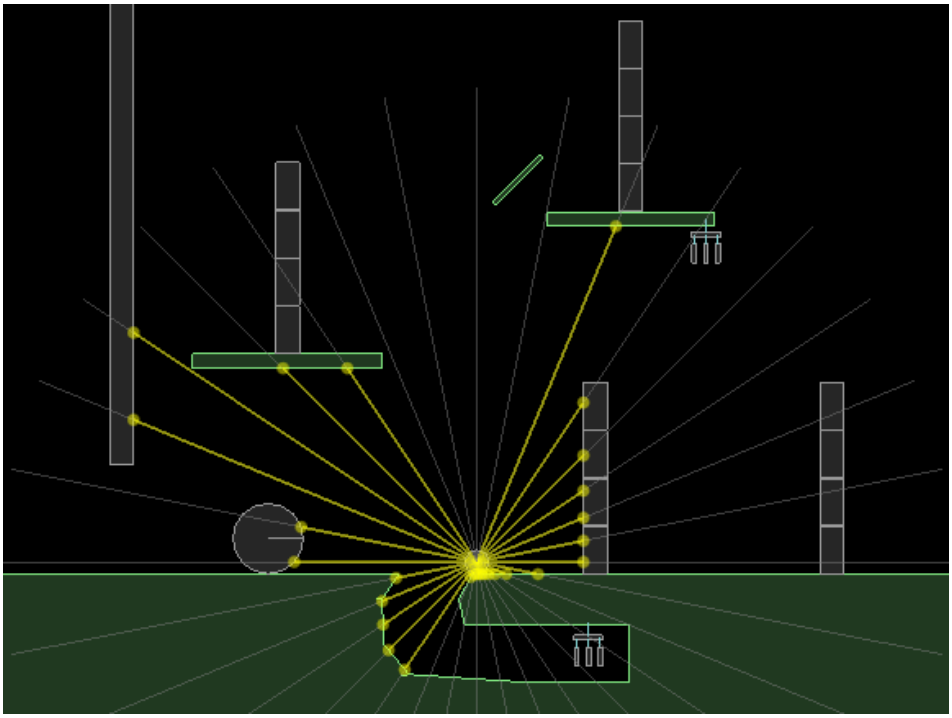

Ilustración 16. Raycasting.

El pseudocódigo es el siguiente:

```
Require:
        Array Collidables. AC
        Avatar.
1: Temporizador //Evitar múltiples colisiones
2: originPoint <- Avatar.position
3: for all vértices in Avatar do:
4:      create ray(originPoint, Avatar.directionVector)
5:      collisions <- rayCasting( AC )
6:      if collisions is not null
7:              stop flag <-1
8:              FunctionELearning(collisioned)
9:      endif
10: endfor
```

El diagrama de flujo que describe las acciones de este proceso es el siguiente:

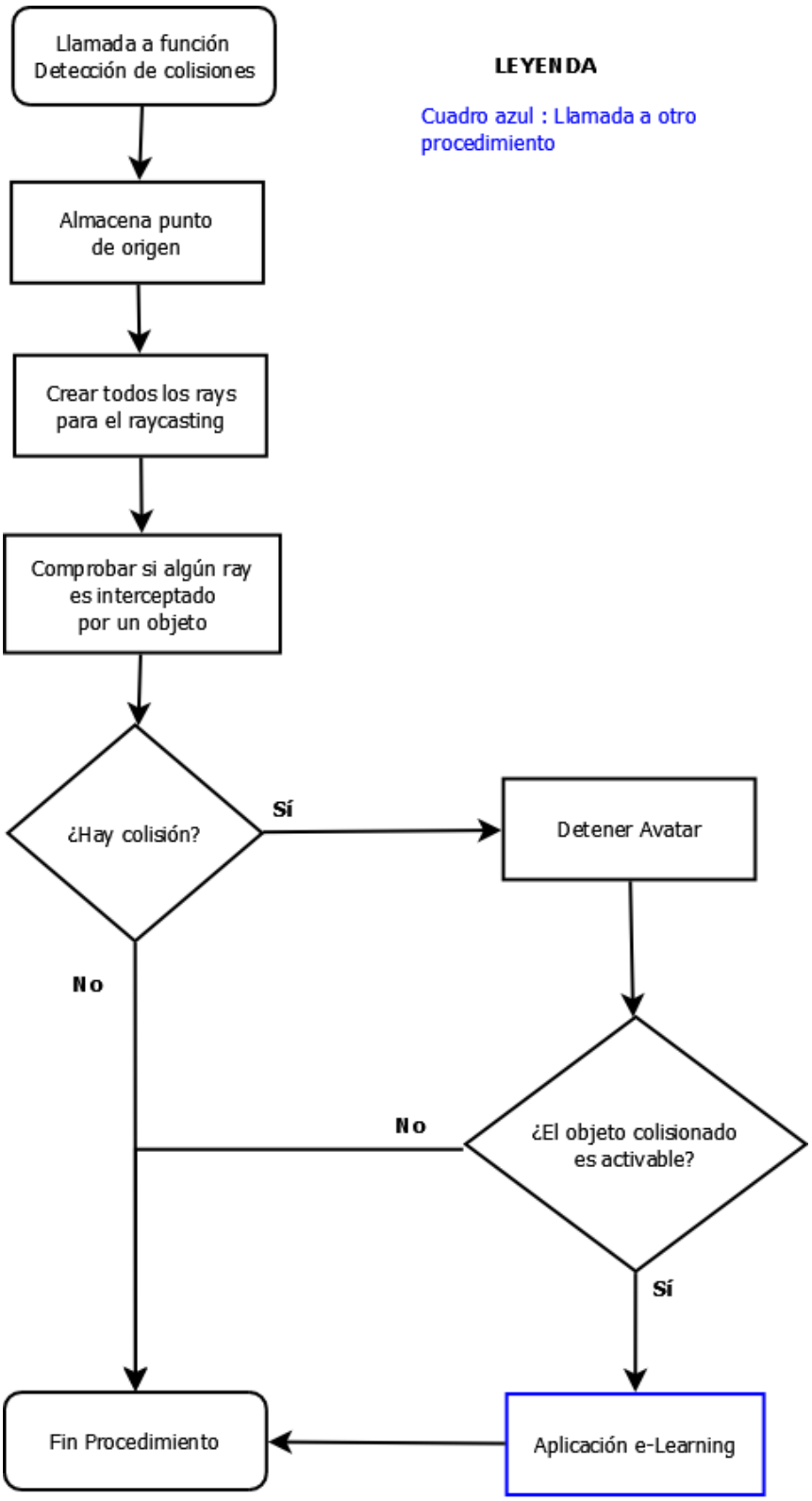

Ilustración 17. Diagrama de flujo del procedimiento Detección de colisiones.

6.3.10. Popups

Antes de entrar en materia con cada *Popup* en particular, hay una serie de procedimientos comunes a todos ellos.

Uno de ellos es el personalizador de imágenes para las notas finales de cada prueba. Recibirá cuatro direcciones de imágenes que identificarán el suspendo, el aprobado, el notable y el perfecto o sobresaliente.

El pseudocódigo es el siguiente:

Require:
    Fail Image -> FI
    Pass Image -> PI
    Very Good Image -> GI
    Top Image -> TI
1: Global Variable Fail <- FI
2: Global Variable Pass <- PI
3: Global Variable Well <- GI
4: Global Variable Perfect <- TI

Por otro lado, aunque con peculiaridades, el diseño de las funciones inicializadoras es la misma en todos los tipos. Guardará la información que compone ese *popup* en un *array* que podrá ser accesible mediante el identificador que lo represente, así al colisionar, se podrá acceder a él fácilmente. Tras esto, se inicializará el contenedor HTML en la web pero manteniéndolo oculto.

Require:
    Content Popup -> CP
    Global Variable ArrayContents -> AC
1: Añadir CP a AC
2: Añadir Contenedor HTML a la web

6.3.11. Popup de Explicación

Se trata de una serie de procedimientos que van desde la inicialización del *Popup* hasta la finalización de la actividad. Sin embargo, ya hemos visto que el diseño del procedimiento de inicialización es el mismo para todas las actividades.

El diseño y pseudocódigo del funcionamiento del *Popup* es el siguiente:

Require:
    Evento colisión
    Contenedor HTML. CHTML
    Global Variable ArrayExplications. AE
    ID Colisionado. IDC
1: **if** CHTML is not exist
2:     Añadir CHTML a la web
3: **endif**
4: Añadir AE[IDC] a CHTML
5: event close is true <- CHTML.visible <- false
6: CHTML.visible <- true

El diagrama de flujo que describe las acciones de este proceso es el siguiente:

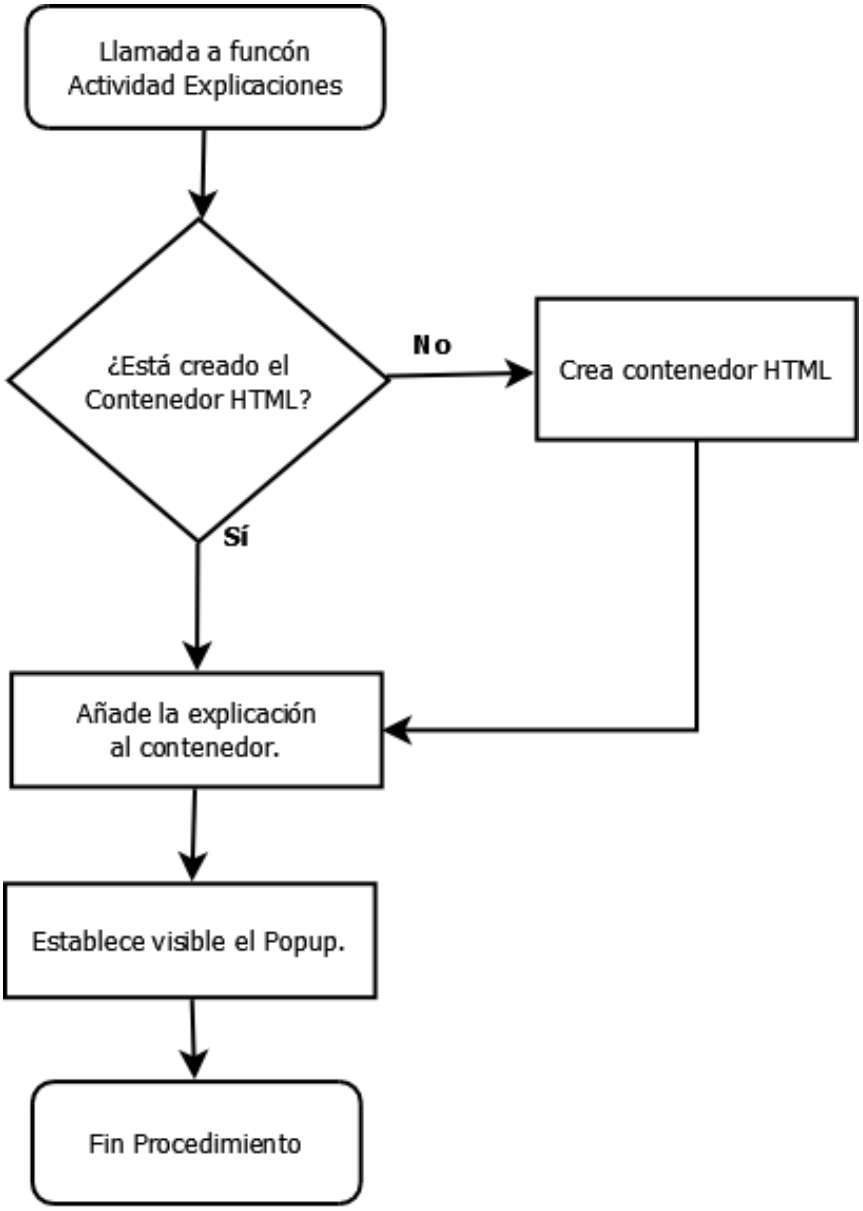

Ilustración 18. Diagrama de flujo del procedimiento Actividad Explicación.

6.3.12. Popups de Quiz y Preguntas

En el diseño de estos dos *popups* también se han tenido en cuenta inicialización, finalización y lanzamiento. Sin embargo, tienen unas peculiaridades que los vuelven más complejos; la evaluación de pregunta y la evaluación final.

El diseño y pseudocódigo son los mismos, pero al implementar tiene unas peculiaridades técnicas a tener en cuenta, tal como pueden ser el formato del HTML o la fuente de la pregunta, ya que en el caso de que la pregunta sea un dictado, la fuente será un audio.

No obstante, todas funcionan igual, existe un botón que servirá como evento para evaluar la pregunta y presentar la siguiente, hasta que finalice la actividad y muestre un resultado.

El diseño y pseudocódigo del funcionamiento del *Popup* es el siguiente:

```
Require:
        Evento colisión
        Contenedor HTML. CHTML
        Global Variable ArrayQuiz. AQ
        ID Colisionado. IDC
1: if CHTML is not exist
2:      Añadir CHTML a la web
3: endif
4: Añadir AE[IDC].primeraPregunta a CHTML
5: event close is true <- CHTML.visible <- false
6: CHTML.visible <- true
7: if user presiona botón enviar
8:      if pregunta correcta
9:              marcador <- marcador + 1
10:     endif
11:     if Hay más preguntas en AE[IDC]
12:             Añadir siguiente pregunta de AE[IDC] a CHTML
13:     elseif
14:             Mostrar nota final
15:             FIN
15:     endif
16: endif
```

El diagrama de flujo que describe las acciones de este proceso es el siguiente:

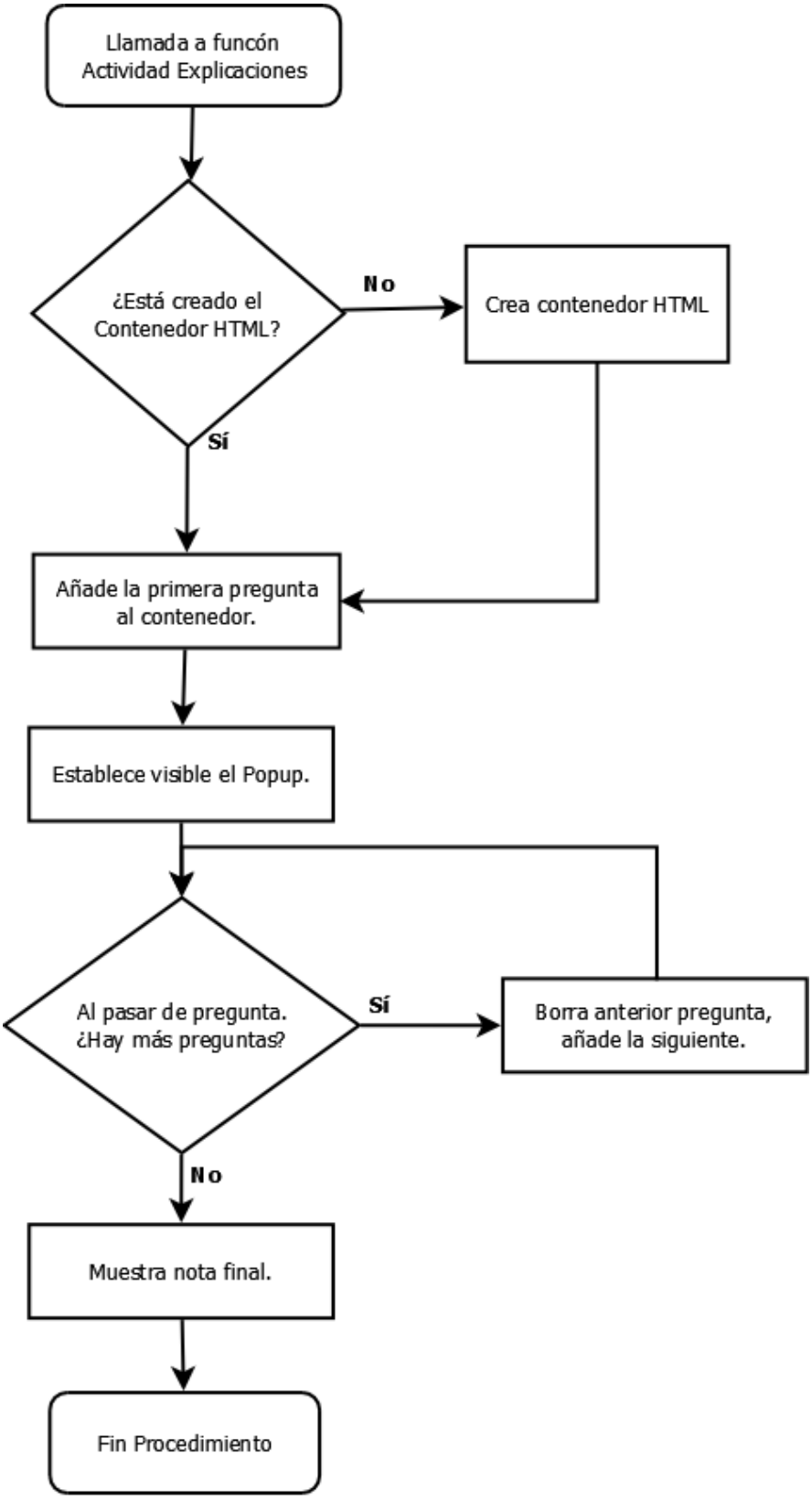

Ilustración 19. Diagrama de flujo del procedimiento Actividades Quiz y Preguntas.

6.4. Análisis General de Actividad

En este apartado veremos el comportamiento y la actividad que tienen los distintos procedimientos de la librería. Se estudiará desde el punto de vista de lo que sería una aplicación habitual y genérica.

Las perspectivas a tener en cuenta son:

- La referencia será la aplicación web, pues es dónde se muestra el proceso que se realiza.
- El programador inexperto o experto es tan sólo el desarrollador que hace uso de la librería, pero no es su usuario como tal. Por lo tanto, en el estudio de la actividad o funcionamiento, no tiene relación con el sistema.
- Este diagrama de flujo que se ha utilizado para representar la actividad de la librería, no es fija, ya que dependerá de cómo el programador la utilice. No obstante, es importante el orden de ejecución de los procedimientos.

El diagrama de flujo ha sido dividido en dos ilustraciones consecutivas debido a su gran dimensión. Se muestran a continuación y resumen todo el comportamiento de la librería:

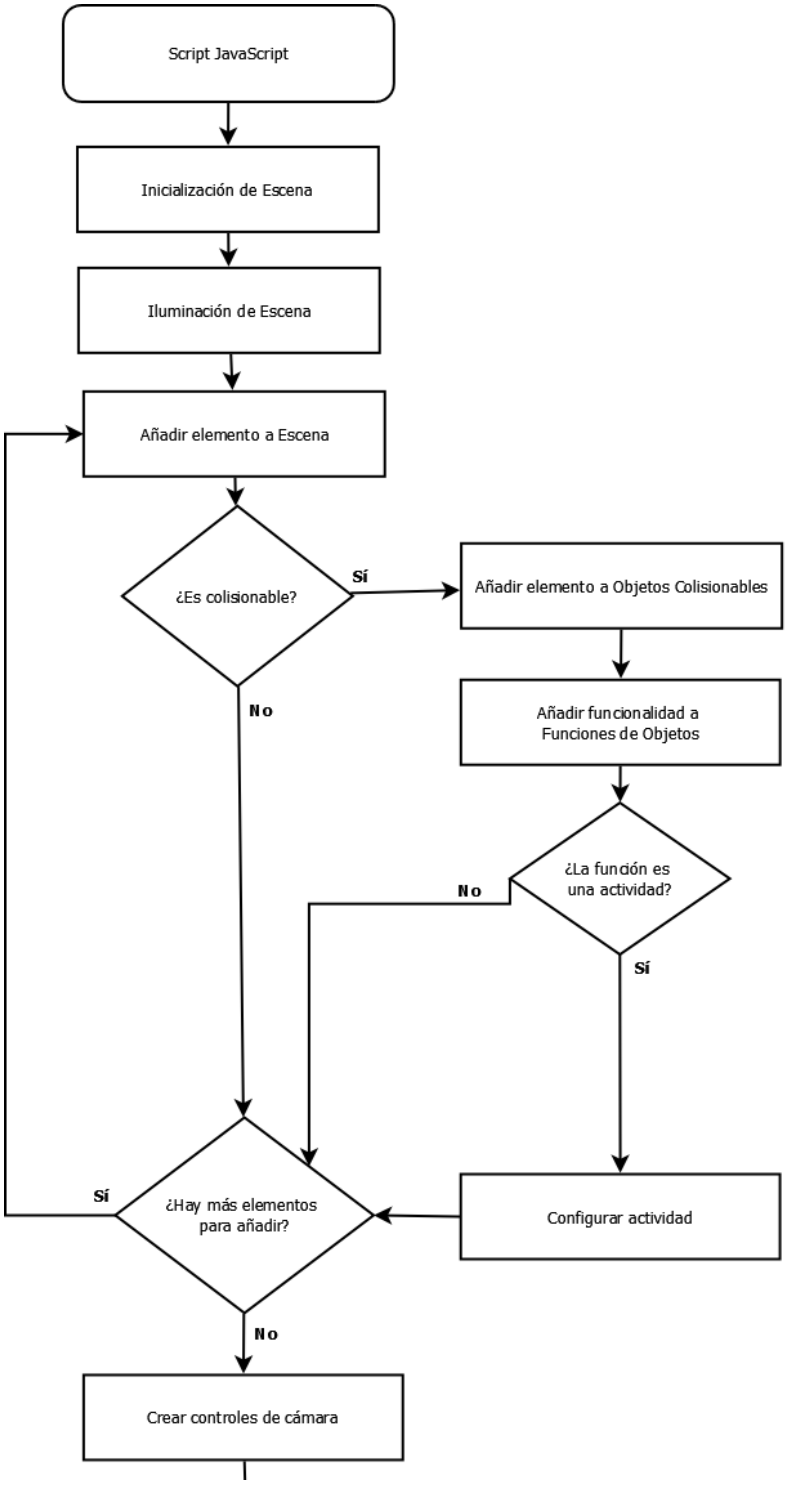

Ilustración 20. Diagrama de flujo general, parte 1.

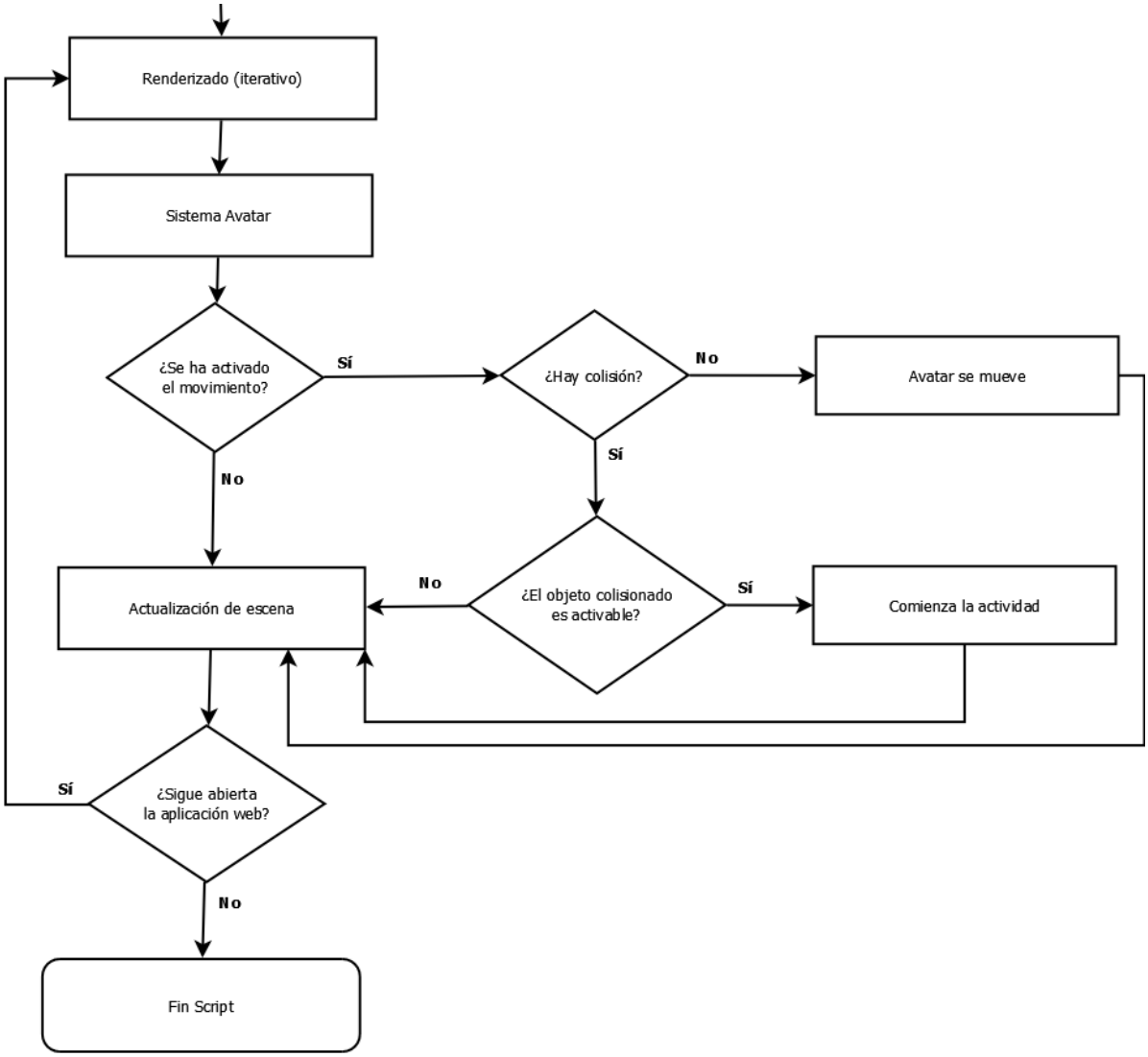

Ilustración 21. Diagrama de flujo general, parte 2.

6.5. Diseño de llamadas a función

Hemos visto el diseño de los procedimientos, pero no cómo se ha diseñado su utilización por parte del programador o usuario que ponga en práctica nuestra librería.

6.5.1. Inicializar escena

La llamada de este procedimiento es de la siguiente manera:

basicTHREE.initiateScene(string cameraType, string rendererType)

Sus argumentos, respectivamente, son los siguientes:

- El tipo de cámara que se quiere utilizar. Puede ser en perspectiva, ortográfica o cúbica. Elegir entre las dos primeras afectará en gran medida a cómo se observa la escena.
- El tipo de *renderer* que se quiere utilizar. Puede ser WebGL o Canvas. SVG fue descartado en su momento por su corta funcionalidad integrada en ThreeJS. Canvas no dispondrá de sombras ni brillos, ni de profundidad, aunque sí cierto efecto tridimensional artificial.

Esta función no devuelve nada, sin embargo, almacena un número importante de variables globales útiles para el funcionamiento adecuado de la aplicación web. Estas variables globales son las siguientes:

- Escena. Útil para añadir nuevos elementos a ella, como por ejemplo objetos o iluminaciones. Se almacena en basicTHREE.scene.
- Cámara de la escena. Útil por si el programador quiere manipularla a su antojo o enfocarla a un objeto en concreto. Se almacena en basicTHREE.camera
- *Renderer*. Necesario para la renderización iterativa de la escena. Se almacena en basicTHREE.render.

Internamente añade suelo y cielo a la escena de manera predeterminada para dar un entorno con sensación de estabilidad para el tipo de aplicación que se busca. Además vuelve responsiva la página web por lo que se adaptará a cualquier resolución.

6.5.2. Añadir elemento a escena

La llamada de este procedimiento es de la siguiente manera:

basicTHREE.addElement(geometry, material, point x, point z, boolean collidable, function function)

Sus argumentos, respectivamente, son los siguientes:

- La geometría es un tipo de objeto de la librería ThreeJS. Recoge la forma del elemento.
- El material es un tipo de objeto de la librería ThreeJS. Recoge el material del que está hecho. La versión actual no soporta texturas.
- La posición respecto el eje x.

- La posición respecto el eje z.
- Booleano que indica si es colisionable o no lo es.
- En caso de ser colisionable, la función o actividad que realizará al colisionar. Será cero o *null* si no realiza nada. No tendrá efecto si el booleano de colisión es falso.

Esta función devuelve el objeto creado, aunque no es necesario que se almacene para nada si no se desea o necesita. Puede ser de utilidad para moverlo posteriormente o para realizar funciones extra aparte de la librería.

Internamente inicializa las aplicaciones, en caso de que sea colisionable y activable. Además, por defecto, el eje y no se puede configurar para que el elemento esté sobre el suelo.

6.5.3. Cargar figura desde JSON

La llamada a esta función es:

basicTHREE.loadMesh(string urlLink,string name, point x, point z, boolean collidable, function funcion)

Sus argumentos, respectivamente, son los siguientes:

- La URL dónde se encuentra el fichero JSON.
- El nombre identificador para poder extraerlo en tiempo de renderización con el procedimiento *get*.
- La posición respecto el eje x.
- La posición respecto el eje z.
- Booleano que indica si es colisionable o no lo es.
- En caso de ser colisionable, la función o actividad que realizará al colisionar. Será cero o *null* si no realiza nada. No tendrá efecto si el booleano de colisión es falso.

Internamente inicializa las aplicaciones, en caso de que sea colisionable y activable. Además, por defecto, el eje y no se puede configurar para que el elemento esté sobre el suelo.

6.5.4. Selección de objeto en escena

La llamada a esta función es:

basicTHREE.getElement(name)

Su único argumento es una cadena y es el identificador o nombre del elemento y devuelve el elemento buscador o -1 si no lo encuentra.

6.5.5. Materiales

Para mantener la simplicidad de uso de la librería, se han creado tres funciones distinguidas para cada tipo de material básico que se quiera usar, aunque el sistema está preparado por si se quiere utilizar cualquier otro material de ThreeJS.

Las funciones son las siguientes:

basicTHREE.shadedMaterial(Color colorA)

basicTHREE.shinyMaterial(Color colorA)

basicTHREE.colorfulMaterial()

El parámetro que reciben *shadedMaterial* y *shinyMaterial* (sombreado y brillante, respectivamente), es el color del material. El color debe incluirse en código hexadecimal o mediante el objeto de ThreeJS Color. Sin embargo, en la librería basicTHREE existen datos constantes que representan a los colores más comunes y etiquetados bajo nombres como *colorWhite* o *colorRed*.

Devuelven el material creado.

6.5.6. Iluminación

Este procedimiento es de mucha utilidad por lo que la simplicidad en ella es máxima como en inicializar la escena. La función que lo llama es la siguiente:

basicTHREE.iluminate(Color colorA)

De igual manera que con el material, el parámetro puede ser un objeto Color de ThreeJS, o un código hexadecimal. Los datos constantes de la librería creada se pueden usar también en esta función para abstraer al usuario de la herramienta de la necesidad de conocer el código hexadecimal de los colores más comunes.

No devuelve nada, la iluminación quedará establecida en escena de manera automática. Aunque almacenará en una variable global el objeto iluminación creado, por si se quisiera manipular externamente. Se almacena en basicTHREE.pointLight.

6.5.7. Sistema Avatar

Este procedimiento es muy importante, y se llama recursivamente en cada renderización de la aplicación web. La llamada al procedimiento es el siguiente:

basicTHREE.avatarLive(MovingMesh, boolean cond, boolean glob, boolean rotate, boolean reset)

Los argumentos que lo componen son los siguientes, respectivamente:

- El Avatar seleccionado.
- Booleano que activará o desactivará el movimiento condicionado.
- Booleano que activará o desactivará el movimiento global.
- Booleano que activará o desactivará la rotación.
- Booleano que activará o desactivará el *reset.*

No devuelve nada, e internamente llama a otro procedimiento que es *testCollidable*, que se explicará más adelante y se encarga de comprobar las colisiones.

6.5.8. Controles de cámara

Este procedimiento se ejecuta mediante la siguiente llamada y activará los controles de ratón y teclado para la cámara:

basicTHREE.createCameraControls()

No tiene argumentos y devuelve los controles de la cámara que es tipo Control de ThreeJS.

6.5.9. Detector de colisiones

Este procedimiento es llamado en basicTHREE.avatarLive, por lo que es interno. Se encarga de comprobar si existen colisiones y actuar en consecuencia como hemos visto en el diagrama de flujo.

La llamada es la siguiente:

basicTHREE.testCollidable()

No recibe parámetros, todo lo que usa para el control de colisiones son valores que cambian con cada iteración de renderización. Tampoco devuelve nada, pero utiliza *flag* para indicar la colisión y que *avatarLive* no tenga efecto en el movimiento una vez haya colisión mediante **stop flag**.

Lo he mantenido englobado en los procedimientos de basicTHREE, ya que puede que en algunos casos y algunas creaciones de aplicaciones quieran ser usados por usuarios que utilicen esta librería para crear otras funcionalidades.

6.5.10. Inicialización de actividades

Las actividades han sido englobadas en tres tipos: explicaciones, cuestionarios o *quiz* y preguntas. Estos últimos engloban preguntas, copiados y dictados, ya que su estructura es prácticamente la misma.

Por lo tanto, hay tres inicializaciones distintas que se componen de dos funciones cada una y son:

- Para explicaciones: **popupInit()** y **basicTHREE.setExplication(Array explicación).**
- Para *quiz*: **quizInit()** y **basicTHREE.setQuiz(Array quiz).**
- Para preguntas: **questionInit()** y **basicTHREE.setQuestion(Array preguntas).**

Todos los *setters* reciben un *array* con el contenido de la actividad.

- En el caso de la explicación recibe un *array* que se forma por elementos que contienen cabecera, cuerpo y pie.
- En el caso del *quiz* recibe un *array* con todas las preguntas que son formadas por imágenes (opcional), preguntas, opciones, respuesta y explicación (opcional).
- En el caso de las preguntas recibe un *array* con todas las preguntas que su estructura puede variar según sea un copiado, un dictado o unas simples preguntas.
    - El copiado tendrá textos, imágenes (opcional), y las versiones correctas del copiado.
    - El dictado tendrá audios, imágenes (opcional) y las respuestas correctas.
    - Las preguntas tendrán preguntas, imágenes (opcional) y respuestas correctas.

Algunas de estas funciones no pertenecen al tipo basicTHREE, ya que son procedimientos y funciones internos de cada tipo de actividad.

6.5.11. Ejecución de actividades

Las actividades se ejecutan al colisionar con un objeto colisionable que se asoció a dicha actividad. Las llamadas a las funciones se lanzarán desde el test de colisiones y son:

- **explicationPopup(id)**
    - El parámetro de esta función es el identificador del objeto colisionado. Si hay más de un objeto con explicaciones, este identificador nos permitirá acceder al objeto adecuado.
- **questionPopup(id, type)**
    - El primer parámetro de esta función es el identificador del objeto colisionado. Si hay más de un objeto con preguntas, este identificador nos permitirá acceder al objeto adecuado.
    - El segundo parámetro indicará cuál es el tipo de actividad dentro del mismo tipo pregunta y pueden ser:
        - Tipo 0: Preguntas.
        - Tipo 1: Copiado.
        - Tipo 2: Dictado.

- **quizTest(id)**
  - El parámetro de esta función es el identificador del objeto colisionado. Si hay más de un objeto con *quizes*, este identificador nos permitirá acceder al objeto adecuado.

Estas funciones no pertenecen al tipo basicTHREE, ya que son procedimientos y funciones internos de cada tipo de actividad.

6.6. Validación

La validación de todos los requisitos se realizará sólo sobre los procedimientos. Mientras que, los requisitos de interfaz, se deben cumplir con las llamadas a función.

6.6.1. Matriz Requisitos / Procedimientos

RF/PR	PR.1	PR.2	PR.3	PR.4	PR.5	PR.6	PR.7	PR.8	PR.9	PR.10	PR.11	PR.12
RF.1	X	X	X	X	X	X						
RF.2							X	X				
RF.3									X			
RF.4										X	X	X
RI.1	X											
RI.2	X	X	X	X	X	X		X				
RI.3							X					
RI.4							X	X	X			
RI5											X	X

Tabla 3. Matriz de Validación: Requisitos / Procedimientos

6.6.2. Matriz Requisitos de interfaz / Llamadas a función

RF/PR	LF.1	LF.2	LF.3	LF.4	LF.5	LF.6	LF.7	LF.8	LF.9	LF.10	LF.11
RIF.1											
RIF.2	X	X	X	X	X	X		X	X	X	X
RIF.3	X	X	X				X		X	X	X

Tabla 4. Matriz de Validación: Requisitos de Interfaz / Llamadas a función

El **Requisito de Interfaz 1** se cumple por el simple funcionamiento de Javascript.

El **Requisito de Interfaz 3** se cumple, pero está gráficamente representado en el apartado 6.4. Sólo existe el bucle de renderización, que no se puede prescindir de él. Y en el caso de las condiciones, son todas internas. Es decir, todas las condiciones, en su totalidad, han sido abstraídas de la complejidad del uso de ThreeJS.

7. PRUEBAS

En este capítulo se muestran algunos resultados obtenidos durante las pruebas realizadas a la librería.

7.1. Diseño del experimento

El diseño del experimento recoge un caso posible de lo que un inexperimentado en programación quisiera hacer con la librería que hemos creado a partir de ThreeJS y su comunidad.

Para ello, se han preguntado a personas ajenas a la informática, cómo se imaginan un juego para un alumno de primaria con preguntas, copiados, pruebas de matemáticas...etc. La creación más original que se ha sugerido tiene las siguientes características:

- Una habitación cuadrada y vacía con los siguientes elementos:
  - El Avatar forma de cubo.
  - Una esfera que cuando colisione, te ponga una explicación diciéndote que busque al cubo.
  - Otro cubo que cuando colisione le pregunte cuántos lados tienen los cubos y cuántas aristas.

En primer lugar, hay que saber de qué manera colocar las paredes para que formen una habitación. Esto puede ser una dificultad añadida pero con unos pocos intentos desplazando en el eje x y z es fácil. La rotación de las paredes "laterales" debe ser de 90º por lo que con Pi/2 valdría.

En segundo lugar, esta prueba nos servirá para depurar algún error que hubiese quedado en el proceso de implementación.

En tercer lugar, utilizaremos información como las líneas de código utilizadas frente a las que se tendrían que utilizar si no se usa basicTHREE, así como cuántas estructuras de control se evitan programar de nuevo.

7.2. Resultados

El resultado tras añadir lo restante sin mucha dificultad es el siguiente:

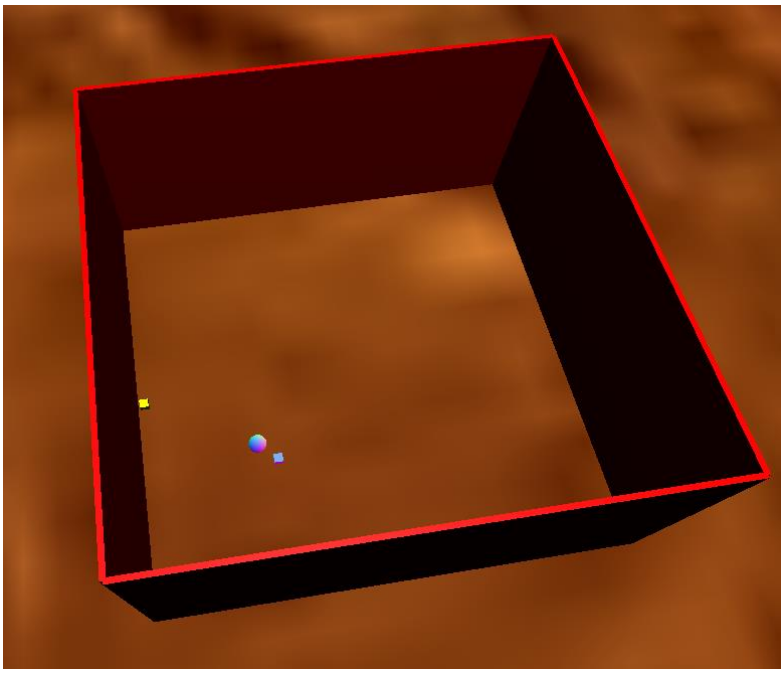

Ilustración 22. Aplicación de prueba.

Son tan sólo unas **50 líneas** de código de Script, sin contar las variables que contienen el texto de la explicación y las preguntas. Recogemos la comparativa de simplicidad con basicTHREE o sin él en la siguiente tabla:

	Sin basicTHREE	**Con basicTHREE**
Líneas JavaScript	960 líneas	50 líneas
Estructuras de control	6 bucles + 4 eventListener + 71 condiciones	1 bucle

Tabla 5. Rendimiento basicTHREE.

7.3. Discusión de los resultados

La prueba ha sido muy satisfactoria, aunque ha habido los siguientes inconvenientes:

- Algunos errores de implementación como contadores mal usados. Han sido solventados gracias a las pruebas.
- Dificultad encontrada a la hora de conocer geométricamente la posición de los objetos para formar entornos deseados.
- Algunas imprecisiones en las colisiones mediante movimiento condicionado.

Líneas y complejidad han sido variables a tener en cuenta para esta prueba. También se ha puesto a prueba la proyección. El concepto de proyección lo asignamos a la capacidad de una persona para crear un entorno y actividad ideados. Una persona ha dado en lenguaje natural una posible aplicación para enseñar a un alumno que está aprendiendo geometría, y se ha creado a partir del mismo.

Esa misma persona, con uso del manual de usuario, podría ser capaz de crear esta misma con algo más de dificultad o más o menos la misma.

Es obvio que, en cuanto a líneas y estructuras de control, ha supuesto una mejora considerable. Además estos números aumentarían exponencialmente conforme la aplicación fuera más compleja.

El ahorro en líneas y complejidad lógica, no sólo refleja simplicidad, sino también un ahorro de tiempo considerable a la hora de desarrollar una aplicación.

8. CONCLUSIONES Y FUTURAS MEJORAS

Una vez concluido el proyecto y todas sus fases de desarrollo, se expondrán las conclusiones a las que se han llegado después de la finalización.

Las conclusiones serán en relación con los objetivos que se plantearon al principio de este propio documento y con los resultados obtenidos en la prueba final y el resultado final de la librería en sí.

8.1. Conclusiones sobre los objetivos

Todos los objetivos planteados en el apartado **1.2** han sido alcanzados y superados.

- Se estudió y aprendió la utilización de ThreeJS en profundidad. Se desecharon las funcionalidades que no eran de interés para mantener la simplicidad del proyecto final.
- Todos los procedimientos están englobados en módulos, y estos procedimientos, si son muy complejos, son divididos en pequeños métodos. Por lo tanto, cumple la modularización y encapsulación.
- Se ha reducido la complejidad y extensión de los códigos de ThreeJS, por lo que se ha logrado el objetivo de abstracción y simplicidad. Uno de los más importantes del presente proyecto.
- Con los módulos considerados y desarrollados se han alcanzado todos los módulos planteados en los objetivos; presentación de escena, movimiento, colisión, avatar y aplicaciones *e-Learning*.
- Se ha establecido un modo secuencial lógico y fácil para el desarrollo de aplicaciones bajo cualquier tipo de escena.
- Finalmente, se ha puesto a prueba la aplicación en la fase de pruebas con una aplicación *e-Learning* ideada por un tercero. Ha sido plenamente funcional y simple, por lo que ha superado dicho objetivo.

8.2. Futuras mejoras

Aunque se han cumplido todos los objetivos, a lo largo de la implementación y de las pruebas, se ha comprobado algunos funcionamientos erráticos o posibles mejoras futuras.

A continuación se van a enumerar un conjunto de mejoras para versiones posteriores en basicTHREE:

- Mejorar el sistema de movimiento y colisión mediante mejores conocimientos de física. Esto hará que el sistema de colisiones nunca falle por movimientos demasiado grandes o una distancia de seguridad de colisión errónea. En la presente versión, estos datos son fijos, pero podrían adaptarse al tamaño del Avatar o los objetos de la escena.

- Idear un sistema para crear entornos sin necesidad de rotar y tantear posiciones de objetos en busca de lo que se quiere. No es una mala idea, pues la idea de diseñar es en base a esto, pero simplificarlo sería un gran avance.
- En relación con lo anterior, sería de interés un editor que permitiese rotar, colocar y otras muchas más funciones, y generar la aplicación o modificar el código ya existente e introducido. Como referencia ya existe el editor [11] de ThreeJS, pero es algo complejo y plagado de demasiadas opciones para un inexperto.
- En cuanto a aplicaciones *e-Learning*, el límite está en la imaginación, por lo que cualquier persona que quiera añadir una funcionalidad, podría hacerlo. Además el sistema ha sido desarrollado para ello.

BIBLIOGRAFÍA

[1] Kapp, K. M. (2012). The gamification of learning and instruction: Game-based methods and strategies for training and education. San Francisco, CA: John Wiley & Sons.

[2] InfoQueue – How Many Software Developers Are Out There? [En Línea] https://www.infoq.com/news/2014/01/IDC-software-developers

[3] Wikipedia – Stage3D [En línea]. https://en.wikipedia.org/wiki/Stage3D

[4] Wikipedia – Java3D [En línea] https://en.wikipedia.org/wiki/Java_3D

[5] Blend4Web [En línea]. https://www.blend4web.com/en/

[6] Documentación de ThreeJS [En línea] http://threejs.org/docs/

[7] BabylonJS. [En línea]. http://www.babylonjs.com/

[8] Creativbloq - Ejemplos de aplicaciones con SVG. [En línea]. http://www.creativebloq.com/design/examples-svg-7112785

[9] Alist Apart - Desarrollo de trazado dinámico de recorridos con SVG. [En línea] http://alistapart.com/article/svg-with-a-little-help-from-raphael

[10] Juego The Aviator, hecho con ThreeJS http://tympanus.net/Tutorials/TheAviator/

[11] Editor ThreeJS [En línea] http://threejs.org/editor/

[12] Documentación de JavaScript. [En línea]. https://developer.mozilla.org/es/docs/Web/JavaScript

[13] **Salas, Lorenzo**. Los Proyectos de Ingeniería.

[14] Jon Duckett. (2015). JavaScript and JQuery: Interactive Front-End Web Development 1st Edition. Idioma Inglés. ISBN: 978-1118531648.