Universidad Carlos iii de Madrid

Escuela Politécnica Superior

Grado en Ingeniería de

Sistemas Audiovisuales

trabajo fin de grado

SIMULADOR UNITY PARA ENTORNOS EMPRESARIALES

Autor: Jaime Valle Alonso
Tutora: Mª Carmen Fernandez Panadero

julio de 2013

Trabajo Fin de Grado

SIMULADOR UNITY PARA
ENTORNOS EMPRESARIALES

Autor

Jaime Valle Alonso

Tutor

Mª Carmen Fernández Panadero

La defensa del presente Trabajo Fin de Grado se realizó el día 11 de julio de 2013, siendo calificada por el siguiente tribunal:

Presidente: Luis Enrique García Muñoz

Secretario: Alejando García Lampérez

Vocal: Carlos Rascón Díaz

y habiendo obtenido la siguiente calificación:

Calificación:

Leganés, a 11 de JULIO de 2013

El éxito puede llevarte a lo más alto pero si no tienes con quién compartirlo puedes sentirte el más desdichado.

Agradecimientos

Me gustaría dedicar estas líneas a mi padres, José y Blanca, y a mis hermanos, José Javier y Alberto, pues sin su apoyo nada de esto hubiese sido posible. En especial a mis padres, por soportar la carga que les ha conllevado mi formación y por apoyarme siempre para seguir adelante, enseñándome a afrontar el sacrificio del buen camino.

A mi novia, Estíbaliz, por darme ánimos para finalizar esta travesía. Especialmente por su comprensión y por su paciencia conmigo, estando ahí en todo momento, llenándome de ilusión cada día y haciéndome disfrutar a lo grande cada pequeño momento.

A mis compañeros de clase, por hacerme más amenas y llevaderas las largas jornadas en la Universidad, ya que sin sus buenos ratos todo hubiese sido más difícil.

A todos mis amigos y familiares, por hacerme sentir diferente, único, y lo más importante: querido.

A mi tutora, Mª Carmen Fernández Panadero, por ofrecerme este Trabajo Fin de Grado y mostrarse siempre disponible para proporcionarme ayuda y facilitarme así la consecución de los objetivos.

A mis compañeros de Telefónica I+D, por acogerme, darme la oportunidad de colaborar en proyectos tan enriquecedores y, sobre todo, formarme tanto profesional como personalmente.

A la empresa de simuladores Simumak, por facilitar unos textos que han servido de inspiración para la creación de los diálogos que figuran en la aplicación.

A todo el personal docente que durante estos años han moldeado lo que hoy soy gracias a su esfuerzo y dedicación.

Resumen

En los últimos años, gracias a la evolución y expansión de la informática, los simuladores se están convirtiendo cada vez más una opción complementaria a las pruebas de campo y acercan a los usuarios a situaciones y entornos muy similares a la realidad. Si bien no reúnen todas las virtudes de realizarlas de forma tradicional, son una herramienta de gran utilidad cuya gran ventaja es la de no acometer los riesgos que supondría la inexperiencia de alguien que se enfrenta por primera vez a una situación desconocida hasta el momento.

Generalmente, estos simuladores suelen crearse específicamente para cumplir unos requisitos prefijados y que son acordados con el cliente, pero una vez el software es entregado, el cliente no puede realizar modificaciones sobre éste. En este proyecto se presenta una visión más abierta, donde el cliente puede realizar ciertas modificaciones sobre el simulador después de la fase de desarrollo para adaptarlo conforme sus necesidades vayan variando.

De este modo, dichos simuladores pueden ser modificados tanto en fase de desarrollo por personal técnico, como posteriormente por personal no técnico, haciendo posible que expertos de la materia y del entorno que el simulador recrea puedan adaptar o incluso añadir nuevos casos sin necesidad de depender de un experto en la tecnología en que ha sido desarrollado.

Así, este proyecto pretende realizar una prueba de concepto que demuestre la viabilidad de una arquitectura modular y flexible que ayude tanto a realizar nuevos simuladores rápida y económicamente, como a hacer al cliente partícipe, si así lo desea, de poder adaptar el simulador a diferentes contextos.

Palabras clave

Unity, Sketchup, modelado, modelo, 3D, emprendedor, empresa, aplicación, juego, simulador, renderizado, motor, gráfico, videojuego, software, Windows, Linux, Mac, OS X, FBX, JSON, SCXML, .NET, API, UnityScript, JavaScript, YUIDoc.

Abstract

In recent years, thanks to the evolution and expansion of computer science, simulators are becoming a complementary option to field tests and bring closer to users to situations and environments similar to reality. Although simulators have not all the virtues of testing traditionally they are a useful tool whose great advantage is not to undertake the risks of inexperience of someone who first faced with this situation.

Generally, these simulators are often created specifically to meet certain requirements that are predetermined and agreed with the client, but once the software is delivered, the client can not make modifications to software. This project presents a more open view, where the customer can make some modifications on the simulator after the development phase to adapt it to his needs.

Thus, these simulators can be modified both in development by technical staff and later by non-technical staff, enabling to experts in the field and simulator environment adapt or even add new cases without relying an expert in the technology that has been developed.

Therefore, the main objective of this project is to perform a concept test that demonstrates the viability of a modular and flexible architecture that helps to make new simulators quickly and economically and make to customer free to adapt the simulator to different contexts.

Keywords

Unity, Sketchup, modelling, model, 3D, entrepreneur, company, application, game, simulator, rendering, engine, graphic, videogame, software, Windows, Linux, Mac, OS X, FBX, JSON, SCXML, .NET, API, UnityScript, JavaScript, YUIDoc.

Índice General

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción y Objetivos

1.1. Introducción

Imaginemos que un alumno debe enfrentarse a una situación desconocida y que acarrea cierta peligrosidad, tanto para él como para su entorno. Si el alumno no sabe reaccionar de manera adecuada, las posibilidades de que ocurra una desgracia son elevadas. Reducir esos riesgos son la principal premisa con la que nacieron los simuladores. Gracias a su evolución, hoy en día son usados ampliamente para llevar a cabo tareas muy diferentes, pero pueden enmarcarse dentro de dos grandes categorías: formativos y de entretenimiento.

Los simuladores formativos son aquellos cuyo principal objetivo es transmitir al usuario ciertos conocimientos de una manera que en mayor o menor medida resulte agradable, entretenida y divertida. Algunos de estos simuladores son los de formación vial, de formación para licencias de vuelo, de control aéreo, de maquinaria pesada, formación académica de materias concretas, etc. Si bien este tipo de simuladores no sustituyen a la formación final en escenarios reales que en muchos casos se requiere, son un complemento muy valioso para que los alumnos ganen confianza en sí mismos e interpreten mejor la teoría.

De hecho, son muchos los casos en los que este tipo de simuladores ayudan a que profesionales sean entrenados para enfrentarse a situaciones que son difíciles de recrear en la realidad pero cuyas consecuencias pueden resultar fatales como en las que se pueden ver envueltos los pilotos en vuelo o el personal clínico y de seguridad en situaciones de emergencia. Otra faceta importante es la recreación de entornos con condiciones cambiantes lentamente presentadas comprimidas en el tiempo, siendo factible para el alumno

comprender las consecuencias a largo plazo de las decisiones tomadas en la actualidad.

En cambio, los simuladores de entretenimiento son aquellos que, por encima del contenido formativo, se enfocan en hacer que el usuario disfrute interactuando con ellos. En esta categoría podemos encontrar simuladores de carreras de coches, fútbol, temáticos, etc. Son concebidos como un pasatiempos y, pese a que la mayoría presenta una historia que se va desencadenando, no tienen como finalidad que el usuario reflexione ni analice las distintas situaciones que se le van presentando a lo largo de la misma.

Sea cual sea el marco en el que se encuentre un simulador, la premisa siempre es permitir al usuario experimentar situaciones reales en entornos controlados.

1.2. Motivación

No cabe duda de la utilidad de este tipo de software, pero su elaboración no es sencilla. Necesita de un equipo técnico para el desarrollo de software, un equipo especialista en la materia sobre la que trata el simulador, diseñadores gráficos y/o especialistas de modelado para crear los contenidos visuales, etc. Además, las especificaciones de cada simulador suelen ser muy concretas, lo que provoca que cada uno de ellos deba volver a la fase de desarrollo en mayor o menor medida por muy parecido que sea todo el entorno donde se va a desarrollar la acción del simulador.

¿Y no hay ninguna forma de reutilizar un simulador para realizar otro? Pues la respuesta es sí. Sí se puede reutilizar, y de hecho, se reutiliza. Pero si todo el simulador ha quedado herméticamente cerrado, inevitablemente esa reutilización va a tener que pasar por una árdua nueva fase de desarrollo para desligar el contenido y tener un nuevo simulador al que asociarle nuevo contenido, lo que implica involucrar una vez más a técnicos de desarrollo, con el consiguiente coste asociado.

Pero si se estudia y desgrana la estructura de un simulador, queda patente la repetición y similitud de ciertas partes entre distintas aplicaciones. La manera de poder rebajar costes en la generación de simuladores derivados de otros pasa por la modularización de la arquitectura para poder modificar ciertas partes de la misma sin ni siquiera tener que tocar otras.

1.3. Objetivos

El objetivo de este proyecto es la creación de un prototipo de simulador para la formación de jóvenes emprendedores cuya peculiaridad reside en que ser modificado tanto su flujo como su contenido tras la fase de desarrollo, una vez el juego se encuentre en producción. Esto puede conseguirse trabajando bajo la premisa de la modularización.

Es evidente que es necesario realizar un mayor esfuerzo en cada una de las fases de desarrollo del software para hacer un molde de simulador que resulte lo más útil posible en el futuro para la adaptación y generación de otros nuevos. Ese esfuerzo extra se ve ampliamente compensado cuando se explotan los beneficios que aporta una aplicación con contenidos desligados.

Para aquellos que estén familiarizados con el desarrollo en plataformas web, esta modularización en los simuladores puede asimilarse a la recomendación de separación bien diferenciada entre presentación y contenido en las páginas web. Esto implica que los desarrolladores que se encargan del código HTML[1] pueden trabajar independientemente de los que se encargan del CSS[2] y, éstos a su vez, del personal encargado de generar los contenidos de la página web.

Aunque para realizar una sóla página web, de contenidos sencillos, dicha separación de tareas también resulta recomendable, quedan especialmente patente sus beneficios conforme aumenta la complejidad y número de éstas. De hecho, prácticamente la totalidad de los dominios web del mundo se componen de varias, incluso decenas de páginas web, que además, por pertenecer al mismo dominio, presentan características muy similares. Es entonces cuando cobra gran importancia poder migrar la estructura y ciertos aspectos de la presentación desde una plantilla general a todas las páginas. En cambio, el contenido será totalmente diferente en cada una de ellas.

Cuando un usuario visita esas páginas web y todos los componentes se unen, percibe un conjunto en perfecta armonía aunque realmente haya sido desarrollada de forma modular. Ese es precisamente el objetivo que se quiere conseguir, extrapolando esa modularización web al sector de los simuladores.

[1] HyperText Markup Language: lenguaje de programación basado en etiquetas derivado del estándar XML -eXtended Markup Language- (véase página 18) que se emplea para generar la estructura de las páginas web

[2] Cascade Style Sheet: lenguaje basado en reglas que se utiliza para establecer la apariencia de los contenidos de una web

1.4. Fases del proyecto

El proyecto ha sido desarrollado en varias etapas que son presentadas a continuación por orden cronológico, aunque algunas de ellas se han solapado temporalmente:

Análisis del problema
Se ha analizado el entorno de los simuladores para emprendedores y se ha definido el alcance del proyecto teniendo en cuenta los requisitos que definen el mismo.

Pruebas con el motor de videojuegos
Se han llevado a cabo pruebas aisladas dentro del motor de videojuegos con el fin de conocer las posibilidades que ofrece éste. Se ha partido de pruebas muy simples y se ha ido aumentando la complejidad de éstas para obtener una perspectiva real del potencial de este software.

Modelado 3D
Una vez conocido el funcionamiento básico y las posibilidades del motor de videojuegos, se han generado los modelos 3D que formarán parte de los escenarios del simulador. Para la realización de algunos de estos modelos se ha partido de referencias reales tratando de reproducir fielmente cada uno de ellos, pero con la libertad suficiente como para realizar las modificaciones que se estimasen oportunas.

Importación de los modelos 3D al motor de videojuegos
Se trata de la exportación de los modelos desde el software de modelado 3D y la importación y adecuación de éstos al motor de videojuegos. El principal problema que se presenta en esta fase es la adecuación de las texturas al entorno de desarrollo de videojuegos, ya que a veces, incluso, éstas desaparecían por completo.

Definición de la arquitectura
Se procede a la definición de la arquitectura del sistema simulador y la relación entre los diferentes módulos. Cada módulo se encarga de proveer cierta funcionalidad al conjunto de los demás, formando en su totalidad la arquitectura del simulador.

Desarrollo del software y externalización de los recursos
Esta fase comprende tanto el desarrollo de la funcionalidad e interactividad del simulador como la externalización de recursos y su integracón con el sistema. Sin duda, la fase más compleja de todo el desarrollo fue la externalización de los recursos y su integración dentro del simulador como un módulo separado del resto.

Evaluación de la aplicación
Ejecución de pruebas de la aplicación en distintos Sistemas Operativos, que arroja que el software desarrollado es compatible con Microsoft Windows, Linux y Mac OS X.

Documentación
Generación de la documentación relativa a la memoria del proyecto y guía del usuario de la aplicación.

1.5. Estructura de la memoria

El objetivo de este documento es acercar al lector a una idea global del proyecto y ayudar a la comprensión de la solución técnica que subyace bajo el mismo. Para ello se ha establecido una estructura que comienza con una visión general y poco a poco se va centrando en las particularidades del proyecto que nos ocupa.

Esta memoria se inicia con *Estado del arte y planteamiento del problema*, donde se aborda un estudio de alguno de los simuladores para emprendedores existentes en el mercado y se pone de manifiesto el problema y los requisitos a cumplir; se continúa con *Diseño de la solución técnica*, abordando el desarrollo técnico completo de este proyecto y posteriormente se procede a la evaluación del software; finalmente se procede a presentar el *Presupuesto y planificación* del proyecto y las *Conclusiones y trabajos futuros*.

1.6. Recursos necesarios

Aunque más adelante, en el capítulo dedicado a *Herramientas empleadas* se ofrecerá una descripción más detallada y la justificación de la elección de algunos de ellos, a continuación se muestra una lista de los recursos específicos que se han empleado durante el desarrollo del proyecto.

- Sketchup [35]: aplicación para modelado 3D.
- Unity [37]: entorno de desarrollo de videojuegos.
- FBX Converter [11]: herramienta para la conversión de formatos de modelos 3D.
- GIMP [33]: software para la edición de imágenes.

- Inkscape [17]: software para la creación y edición de gráficos vectoriales.

Aparte del software anterior, evidentemente se necesita un ordenador con conexión a Internet y que cuente con buena capacidad gráfica para trabajar con este tipo de software. Se recomiendan tarjetas gráficas NVIDIA o ATI, especializadas en tareas de representación gráfica, para ahorrar a la CPU cálculos extra.

Capítulo 2

Estado del arte y planteamiento del problema

2.1. Análisis del Estado del Arte

2.1.1. Introducción

Los simuladores para emprendedores están en auge, sobre todo gracias a iniciativas de Ayuntamientos, Comunidades Autónomas e incluso Ministerios que apuestan por la formación de los jóvenes para crear empresas con una formación y conocimiento de los riesgos más completa.

En este apartado se va a proceder a realizar el análisis de dos de estos simuladores, prestando especial atención a las diferencias existentes entre éstos y el simulador de este proyecto, así como las ventajas y desventajas que presentan las diferentes tecnologías empleadas.

2.1.2. Simulador A: Crea y Compite

Crea y Compite: Simulador de Creación y Gestión de Empresas [13] es un simulador de plataforma web impulsado por la Comunidad de Madrid gracias a la ayuda del Fondo Social Europeo. Está desarrollado con tecnología Adobe Flash y es de tipo 2D. Es necesario crear una cuenta de usuario en la que se deben facilitar datos como el nombre de usuario, la contraseña de acceso, correo electrónico, sexo, nacionalidad y fecha de nacimiento. Una vez registrado, el usuario puede acceder al juego introduciendo sus credenciales.

Al entrar se presenta el escenario principal: una ciudad que contiene algunos de los más importantes iconos de la ciudad de Madrid, tal y como puede verse en la Figura 2.1.

Figura 2.1: Escenario principal del simulador Crea y Compite

La información para el usuario queda definida en dos barras de herramientas: la superior, que contiene las diferentes etapas del progreso del juego e indica cuáles han sido superadas; y la inferior, que a su vez se divide en dos submenús. El primero de ellos contiene la *Ayuda*, *Enviar a un amigo* el enlace al juego y *Accesibilidad*, que habilita los atajos de teclado. El segundo, referido al progreso del usuario en el simulador es detallado en los siguientes párrafos.

El botón de *Informes* permite acceder a los informes que le son de utilidad al usuario y que sirven para ir basando la historia del simulador, como puede verse en la Figura 2.2.

El botón denominado *Cuaderno* lleva al usuario a visualizar una lista de los documentos que ha obtenido y de los que debe conseguir. Al clicar sobre uno de los documentos obtenidos, éste es mostrado dentro de la aplicación, tal y como muestra la Figura 2.3. Aunque también se da la posibilidad de, pulsando sobre el icono de la impresora, ver el documento en formato PDF en una nueva pestaña del navegador.

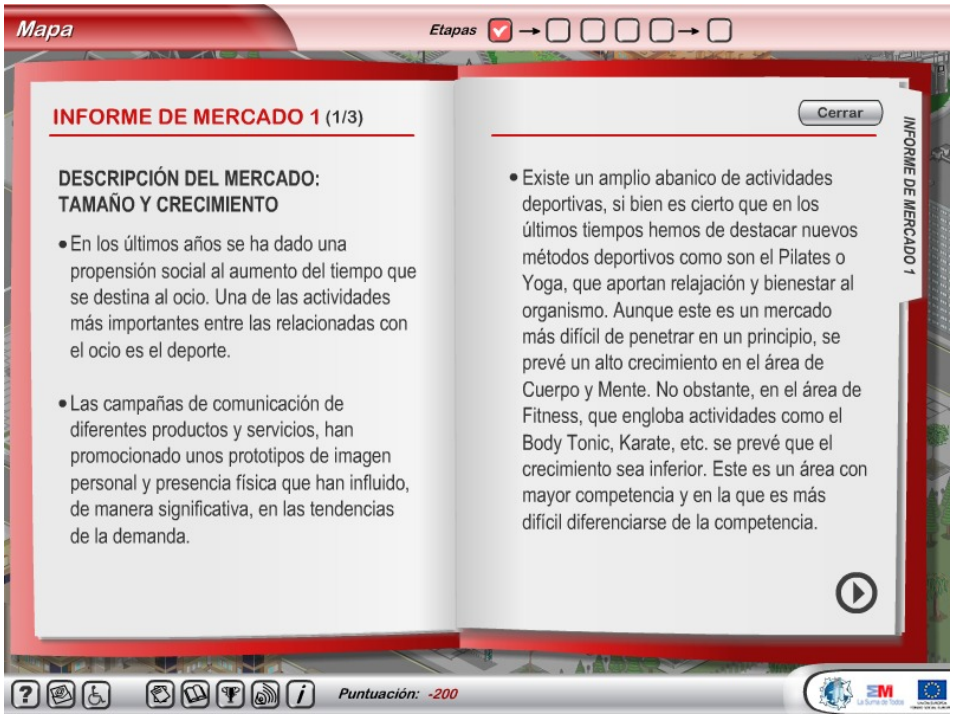


Figura 2.2: Uno de los informes del simulador Crea y Compite

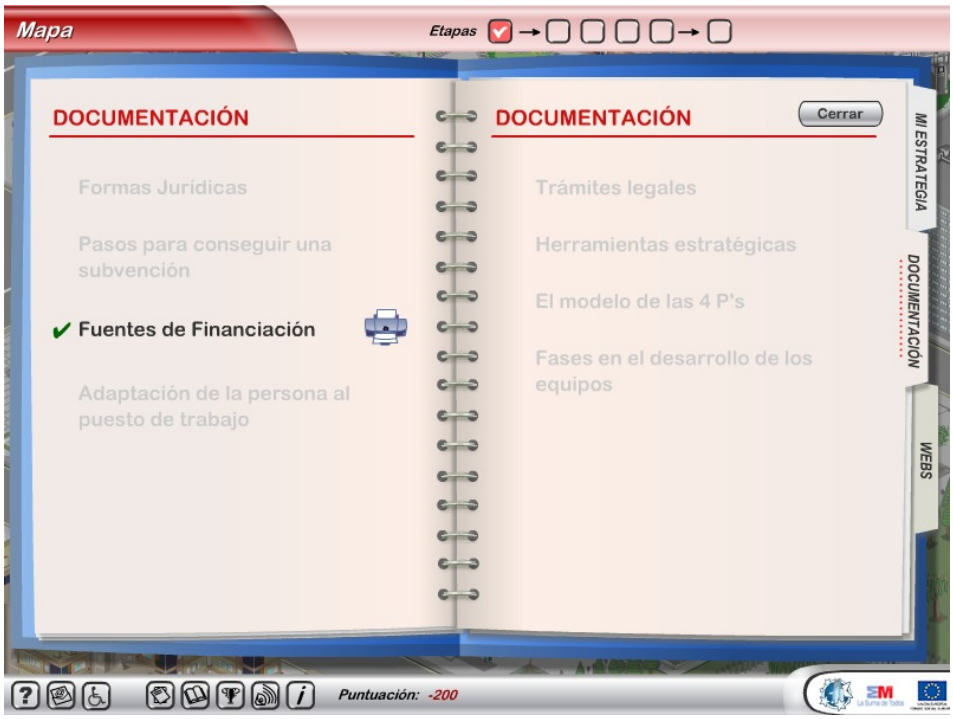


Figura 2.3: Menú Cuaderno del simulador Crea y Compite

El tercer botón es de *Ranking*, que muestra una tabla de los trece jugadores que mayor puntuación ostentan en su poder. El cuarto botón es el de *Apagar Sonido*, que como su propio nombre indica, silencia o no todos los sonidos emitidos por el simulador.

Finalmente, aparece el botón de *Inventario* que lleva al usuario a ver la lista que se muestra en la Figura 2.4 con la documentación que el usuario necesita ir consiguiendo para ir avanzando en el juego.

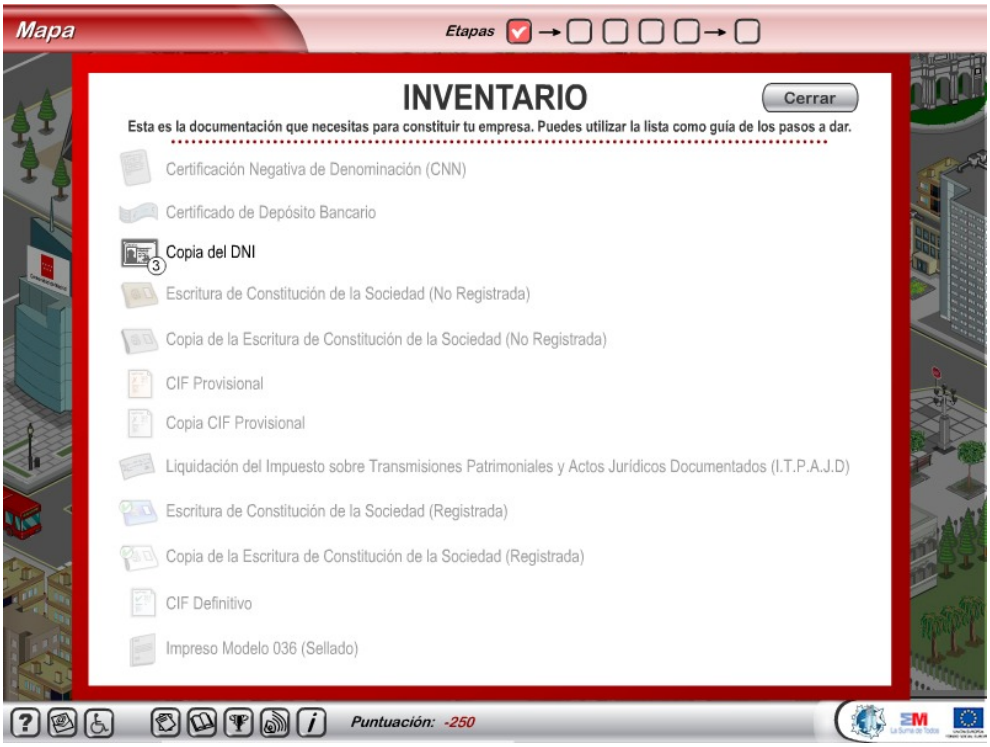


Figura 2.4: Inventario del simulador Crea y Compite

Adentrándonos un poco más en la parte técnica, el cambio de escenas se realiza al pulsar sobre alguno de los objetos interactivos que son aquellos que se iluminan o presentan algún otro tipo de cambio al pasar el cursor del ratón por encima. Como ya se comentó al detallar los botones, el usuario puede habilitar los atajos de teclado para acceder de manera rápida a los distintos componentes del escenario.

El juego posee una pista de audio con sonidos ambientales en el escenario principal y un hilo musical para los escenarios secundarios. Con ello se consigue un ambiente agradable para el usuario.

Por otro lado, el simulador presenta mucha documentación legislativa, algo que en ocasiones lo convierte en realmente una herramienta de aprendizaje y no de entretenimiento, ya que además a veces se hecha de menos cierta

personalización hacia el usuario. En cualquier caso, esto no puede convertirse de forma alguna en motivo de reproche hacia el simulador, puesto que su finalidad la cumple con creces.

En cuanto al feedback para el usuario, incorpora un sistema de puntuación numérica y ésta está presente en pantalla durante toda la partida. Esto supone una realimentación para el usuario sobre su capacidad emprendedora, aunque no hay que desanimarse si al principio la puntuación no es la esperada pues precisamente la razón existencial de estos simuladores es favorecer el aprendizaje.

En lo que se refiere a la continuidad, cuando el usuario sale del juego el progreso actual es guardado, manteniendo todos los progresos del jugador.

2.1.3. Simulador B: Simula

Se trata de un juego de simulación empresarial del Ministerio de Industria, Turismo y Comercio cuyo contenido es representado en 2D y la tecnología de desarrollo es Adobe Flash, al igual que el simulador analizado con anterioridad. Simula o Simul@ [25] también tiene como requisito registrarse para poder acceder al juego.

Figura 2.5: Configuración de la Empresa en Simula

Una vez se accede, el usuario debe introducir algunos datos relativos a la empresa y apariencia del jugador (véase Figura 2.5). La empresa puede

enmarcarse dentro de uno de los tres sectores que vienen predefinidos: Restauración, Confección textil o Comercio. Tras elegir estos datos, comienza la partida y éstos serán utilizados a lo largo de la misma para referirse al jugador (nombre de usuario facilitado).

Dependiendo del sector que el usuario haya elegido, el escenario de simulación es diferente. En caso de decantarse por el sector de Restauración el avatar del jugador será ubicado en un pequeño local, que puede verse en la Figura 2.6.

Figura 2.6: Escenario del sector de Restauración en Simula

Durante el progreso del juego al usuario le son requeridos ciertos datos que plasmen sus ideas para asesorarle. Estas peticiones se van realizando al pulsar sobre los objetos interactivos que van apareciendo en el escenario y que son mostrados bajo una flecha amarilla que los apunta, como se veía en la Figura 2.6. Un ejemplo de estas pantallas de introducción de datos se muestra en la Figura 2.7.

La ambientación del escenario se ayuda de algunos efectos sonoros como teléfonos sonando cuando el usuario debe atenderlos, lo que ayuda al usuario a localizar rápidamente cuál es el siguiente paso a dar a la vez que aporta algo de calidez al escenario para que no se muestre tan frío para el jugador.

En cuanto a la información que se le muestra al usuario, la pantalla se divide en cuatro zonas con menús. En la parte superior se muestra el menú referente al progreso de la partida y el punto actual de la misma, así como el nombre de la empresa creada y el sector en el que se enmarca. También

Figura 2.7: Una de las pantallas de petición de datos en Simula

incluye un submenú de utilidades entre los que se encuentra una calculadora propia del simulador, el botón de control de silencio del sonido, otro para recomendar el juego a un amigo y el botón de salir.

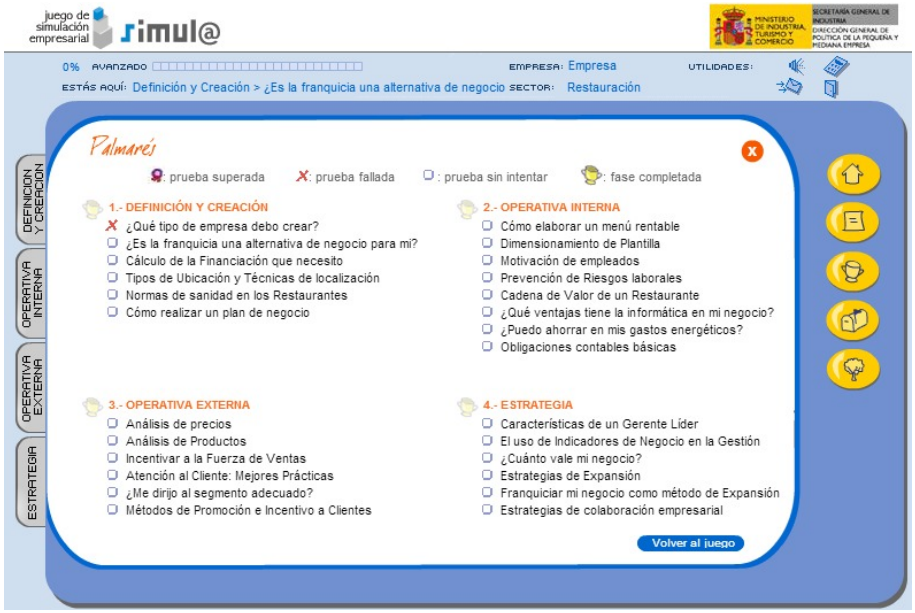


Figura 2.8: *Palmarés* de Simula

En el menú de la derecha aparecen los siguientes botones: *inicio*, que nos lleva a la página inicial que nos da la opción de modificar los datos de la

empresa o continuar jugando; *instrucciones*, que muestra información sobre el simulador y ayuda sobre el funcionamiento y uso del mismo; *palmarés*, que indica las fases e hitos de cada una de ellas que han de conseguirse y los que se han pasado, indicando claramente si han sido superadas satisfactoriamente o no (véase Figura 2.8).

El penúltimo botón del menú derecho, *buzón empresarial*, abre una nueva ventana del navegador con un formulario para enviar información y colaborar con la comunidad del simulador, como se puede ver en la Figura 2.9.

Figura 2.9: *Buzón empresarial* de Simula

Finalmente, el último botón se corresponde con *estación de conocimiento*, cuyo comportamiento es abrir documentación legal y de información empresarial útil para el jugador en una nueva pestaña del navegador. Esta documentación se encuentra organizada en tres niveles. El primero de ellos referente al sector empresarial de la empresa; el segundo nivel en la jerarquía, tras seleccionar el sector, es el de fases; y finalmente, la prueba con la que se asocia dicha documentación.

Así, en la Figura 2.11 puede verse la documentación relativa a la prueba de la Figura 2.10.

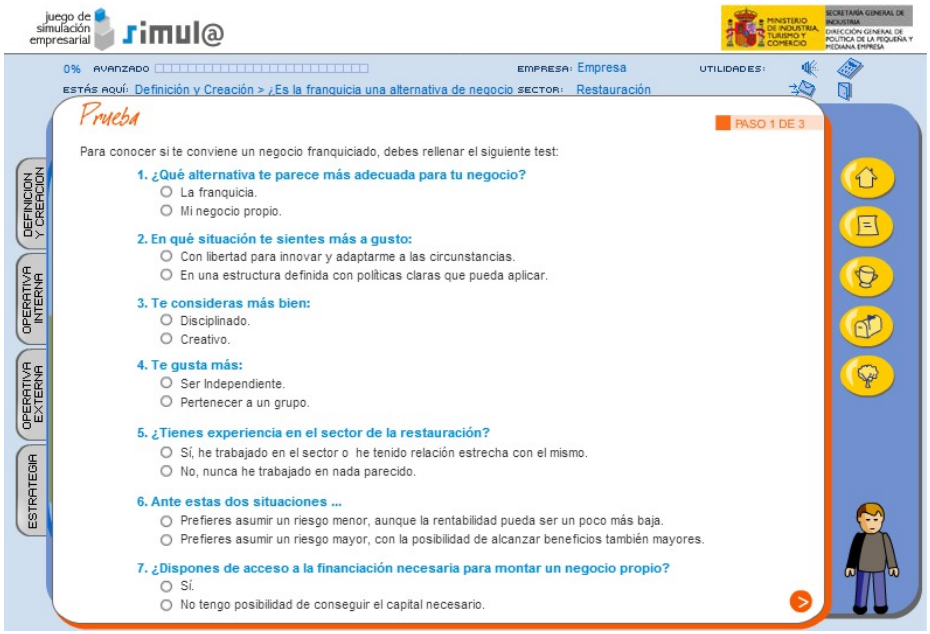


Figura 2.10: *Estación de conocimiento* de Simula

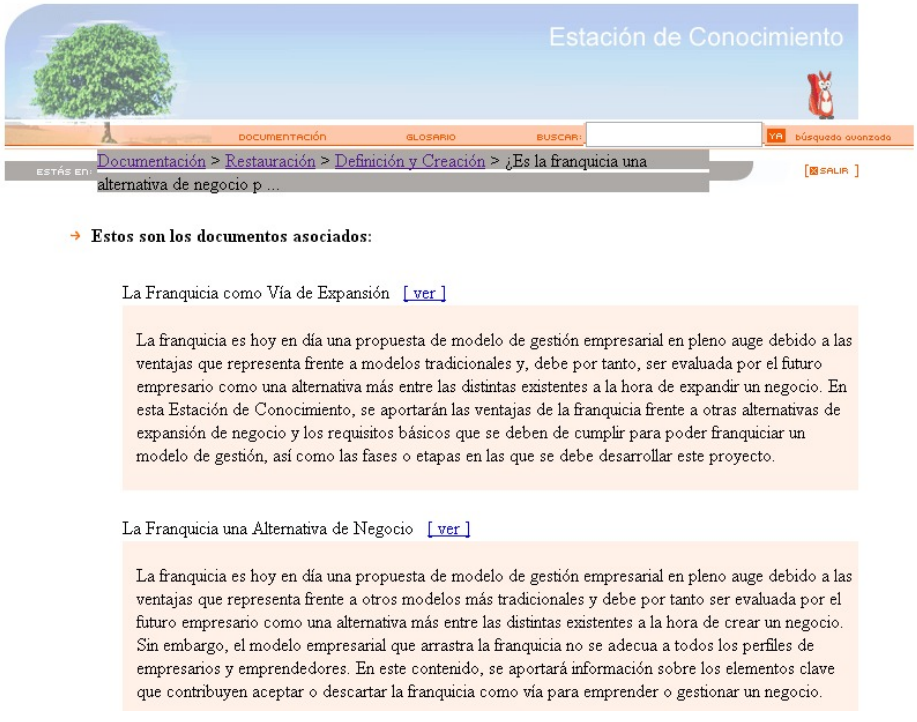


Figura 2.11: Una de las pruebas de Simula

Por otro lado, el menú izquierdo está compuesto por cuatro botones, correspondientes a cada una de las fases del juego. Al pulsar sobre ellos se despliega un menú contextual (ver Figura 2.12) con todas las pruebas que componen esa fase. Al pinchar sobre el nombre de una prueba, automáticamente el juego carga el escenario que la contiene y se lanza dicha prueba. Esto implica que el juego no tiene por qué seguir un orden secuencial definido.

Figura 2.12: Menú contextual de las pruebas para una fase de Simula

Por su parte, este simulador establece un sistema de puntuaciones no numérico, consistente en calificar como superada positivamente o negativamente una prueba. Éstas pueden repetirse para cambiar las calificaciones negativas y superarlas correctamente. Además, como es de esperar, el juego puede abandonarse y todo el progreso quedará guardado para la próxima vez que el usuario acceda al sistema.

2.1.4. Conclusiones

Tras el análisis de ambos simuladores, no cabe duda del nivel de éstos. El simulador de este proyecto, denominado *Entrepreneur Simulator 3D*, no pretende competir con ellos ni mucho menos, más si cabe partiendo de la premisa de que este simulador se trata de un prototipo, por lo que no contiene una historia completa.

Pero aún no siendo comparables cuantitativamente, sí podemos manifestar algunas de sus características en la Tabla 2.1 a modo de resumen.

	Crea y Compite	Simula	ES3D
Plataforma	2D	2D	3D
Tecnología	Adobe Flash	Adobe Flash	Unity 4
Métodos de entrada	Ratón y teclado (opcional)	Ratón y teclado	Ratón y teclado (teclas personalizables)
Interacción	Mediante ratón	Mediante ratón	Ratón y posición en el mundo virtual
Objetos interactivos	Sí, iluminados y cambio de cursor con *mouseover* [1]	Sí, flecha amarilla y cambio de cursor con *mouseover* [1]	Sí, cambio del cursor (2 tipos) con *mouseover* [1]
Recursos externos modificables	No	No	Sí, ficheros SCXML y JSON
Efectos sonoros	Sí	Sí	Ambientales
Documentación e información personalizada	Sí, genérica	Sí, genérica y referencias al usuario (nombre)	Sí, personalizada
Sistema de puntuaciones	Numérico	No numérico	Basado en consecución de objetivos
Reanudar partida	Sí	Sí	Sí
Idioma(s)	Español	Español	Español, inglés... (ampliable sin límite)
Escritorio virtual como atajo de aplicaciones	No	No (calculadora propia)	Sí

Tabla 2.1: Características de simuladores analizados y ES3D.

Aunque quizá para el tipo de simulador que nos ocupa la plataforma 2D sea suficiente para la presentación de los contenidos, un entorno 3D permite añadir nuevos tipos de interacción, como los de distancia o paso por cierto lugar del escenario. Además, acerca al jugador al entorno, ofreciendo otro tipo de experiencia más realista para el usuario.

Otro factor a tener en cuenta acerca de los tres simuladores es el uso de conexión a Internet. Mientras que *Crea y Compite* y *Simula* hacen un uso continuado de la conexión de datos, *Entrepreneur Simulator 3D* solamente lo realiza en el momento de obtener sus ficheros de instalación, siendo totalmente independiente de la conexión de red, a excepción de las aplicaciones externas que se acceden desde el propio juego.

[1] Generalmente así se denomina en programación al evento que ocurre cuando el cursor se sitúa encima de algún elemento.

2.2. Estándares empleados

2.2.1. XML

Se trata de un lenguaje basado en etiquetas cuya peculiaridad reside en que, a pesar de ser un estándar creado por el consorcio W3C -World Wide Web Consortium- [42], sus etiquetas no están limitadas y pueden definirse todas las que se deseen. Por la flexibilidad que ofrece, este lenguaje fue bautizado como eXtensible Markup Language, que da lugar a la siglas que lo definen [40].

Según el W3C, el XML deriva del SGML y, a su vez, de él descienden otros lenguajes específicos como el HTML -HyperText Markup Language-.

En la Figura 2.13 se muestra un ejemplo de la estructura de un documento XML (extraído de [14]).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
  <CD>
    <title>Eros</title>
    <artist>Eros Ramazzotti</artist>
    <country>EU</country>
    <company>BMG</company>
    <price>9.90</price>
    <year>1997</year>
  </CD>
  <CD>
    <title>When a man loves a woman</title>
    <artist>Percy Sledge</artist>
    <country>USA</country>
    <company>Atlantic</company>
    <price>8.70</price>
    <year>1987</year>
  </CD>
</catalog>
```

Figura 2.13: Código XML de ejemplo

Aunque este estándar en sí no es usado en el proyecto, se presenta dentro de la Sección *Estándares empleados* como presentación del siguiente de los estándares.

2.2.2. SCXML

Basado en el anterior, y desarrollado también por el W3C, el SCXML -State Chart XML: State Machine Notation for Control Abstraction- provee una estructura de máquina de estados con capacidad de definir estados, transiciones entre ellos, evaluadores condicionales, datos, contenido ejecutable y la posibilidad de indicar cuándo debe ser ejecutado el mismo (al entrar al estado o al salir de él), entre una larga lista de elementos [41].

Puede verse un ejemplo SCXML en la Figura 2.14 (extraído de [20]).

```
<?xml version="1.0"?>
<!-- Traffic light controller. -->
  <scxml initialstate="light">
  <state id="light">
    <initial><transition target="Red"/></initial>
    <state id="Red">
      <onentry>
        <send target="Self" event="goGreen" delay="1s"/>
      </onentry>
      <transition event="goGreen" target="Green"/>
    </state>
    <state id="Yellow">
      <onentry>
        <send target="Self" event="goRed" delay="500ms"/>
      </onentry>
      <transition event="goRed" target="Red"/>
    </state>
    <state id="Green">
      <onentry>
        <send target="Self" event="goYellow" delay="1s"/>
      </onentry>
      <transition event="goYellow" target="Yellow"/>
    </state>
  </state >
</scxml>
```

Figura 2.14: Código SCXML de ejemplo

Se ha empleado este estándar para definir el flujo de estados en el juego ya que, además de ser un lenguaje cuya estructura resulta más familiar por su descendencia de XML, existía la posibilidad de reutilizar la implementación básica de éste que fue desarrollada en [23], pues aunque no comprende la totalidad de la especificación, si lo hace de los estados, transiciones y eventos.

2.2.3. JSON

JavaScript Object Notation o Notación de Objetos de JavaScript presenta un formato sencillo y ligero para intercambio de datos que está ampliamente extendido en la comunicación con servidores web. Su facilidad de uso ha hecho que su empleo sea cada vez más popular para cualquier tipo de intercambio de datos. Y, aunque quede definido con sólo dos estructuras [2], pares clave-valor y colecciones tipo lista, la multitud de posibilidades de conjunción entre ambas provee a este formato de una flexibilidad extraordinaria.

En la Figura 2.15 vemos un ejemplo de este tipo de estructuras como una representación de los mismos datos que en la Figura 2.13.

```
{
  "catalog":
  {
    "CD":
    [{
        "title": "Eros",
        "artist": "Eros Ramazzotti",
        "country": "EU",
        "company": "BMG",
        "price": "9.90",
        "year": "1997"
      },
      {
        "title": "When a man loves a woman",
        "artist": "Percy Sledge",
        "country": "USA",
        "company": "Atlantic",
        "price": "8.70",
        "year": "1987"
      }]
  }
}
```

Figura 2.15: Estructura JSON de ejemplo

El empleo de este tipo de estructuras en el proyecto viene determinada por la ligereza y la amigable representación de los datos que posee. Esto permite que personas que no estén familiarizadas con el mundo de la programación no se sientan abrumadas por códigos con un corte más técnico.

2.3. Análisis de requisitos

Este proyecto nace con la idea de crear un simulador donde el usuario tuviera que pasar por distintos escenarios, los cuales se irían desbloqueando tras cumplir una serie de objetivos previos. En ningún caso se pretendía generar un guión completo, sino una pequeña porción que implementase todas las funcionalidades.

A continuación se presentan los requisitos a cumplir obligatoriamente y una serie de recomendaciones que, de ser posible, deberían cumplirse.

Requisito: REQ-00	
Descripción	El flujo de la aplicación debe estar definido en ficheros externos que puedan ser modificables para cambiar la secuencia de acontecimientos de la historia.
Obligatoriedad	Obligatorio.

Tabla 2.2: Requisito REQ-00.

Requisito: REQ-01	
Descripción	Externalización de los recursos tales como diálogos, botones, etc. Deben permitir su modificación y creación de tantos nuevos como se desee sin conocimientos de programación.
Obligatoriedad	Obligatorio.

Tabla 2.3: Requisito REQ-01.

Requisito: REQ-02	
Descripción	Personalización de los elementos de la interfaz. Se han de poder crear los elementos con propiedades de tamaño y posición deseados de manera fácil.
Obligatoriedad	Obligatorio.

Tabla 2.4: Requisito REQ-02.

Requisito: REQ-03	
Descripción	Persistencia. El progreso del usuario debe ser guardado y restaurarse cada vez que vuelva a la aplicación y así lo desee.
Obligatoriedad	Obligatorio.

Tabla 2.5: Requisito REQ-03.

Requisito: REQ-04	
Descripción	Idiomas. La aplicación debe contar al menos con los idiomas de español e inglés.
Obligatoriedad	Obligatorio.

Tabla 2.6: Requisito REQ-04.

Requisito: REQ-05	
Descripción	La transición entre estados de la historia y el acceso a nuevas opciones debe estar controlado mediante la consecución de objetivos.
Obligatoriedad	Obligatorio.

Tabla 2.7: Requisito REQ-05.

Requisito: REQ-06	
Descripción	Recolección de datos y generación de documentación personalizada tras completar el progreso del simulador.
Obligatoriedad	Obligatorio.

Tabla 2.8: Requisito REQ-06.

Requisito: REQ-07	
Descripción	Sistema Operativo. La aplicación del simulador debe correr bajo sistemas Windows como mínimo. Si puede extenderse a otros sistemas será bien acogido.
Obligatoriedad	Obligatorio y recomendable.

Tabla 2.9: Requisito REQ-07.

Requisito: REQ-08	
Descripción	Formato de los ficheros (externos) del flujo de simulación de la aplicación.
Obligatoriedad	Se recomienda formato SCXML (véase justificación en la pág. 19).

Tabla 2.10: Requisito REQ-08.

Requisito: REQ-09	
Descripción	Formato del resto de los ficheros de recursos externos de la aplicación.
Obligatoriedad	Libre.

Tabla 2.11: Requisito REQ-09.

Los requisitos establecidos en las tablas anteriores deja patente la necesidad de modularizar la arquitectura en tantos módulos como sea posible sin que se pierda la esencia de cada uno de ellos.

El resto de decisiones relacionadas con la aplicación y que no estén definidas dentro de los requisitos impuestos quedan a cargo del buen entender del desarrollador, siendo siempre debidamente justificadas. Éste, con el fin de adentrarse un poco más en el mundo del emprendedor leyó los primeros capítulos de *Empresa e iniciativa emprendedora* [12], donde se desarrollan las capacidades que debe tener un emprendedor.

2.4. Herramientas empleadas

2.4.1. SketchUp

Figura 2.16: Logo de Google SketchUp 8

SketchUp [35] es uno de los softwares de modelado 3D gratuitos más potentes del mercado, y sigue en desarrollo. También dispone de una versión de pago, denominada *Pro* que acompaña siempre a la versión básica. Su coste ronda los 475 € para la licencia *Single User* y su precio se ve rebajado hasta los 36,70 € anuales en la versión para estudiantes.

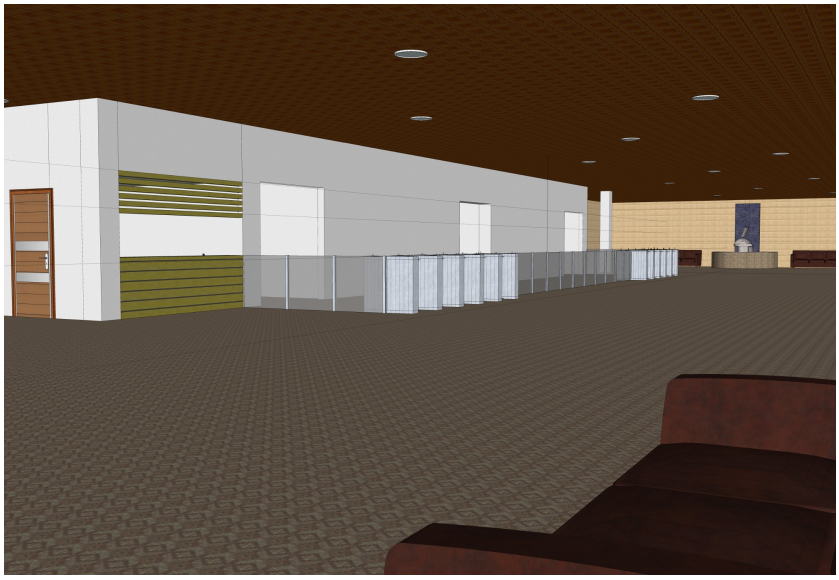


Figura 2.17: Escenario modelado con Google SketchUp 8

La historia de este programa se remonta al año 2000, cuando la empresa @Last Software lanza al mercado una aplicación de modelado 3D bajo el lema *"3D para todos" - "3D for everyone"*- y con la premisa de un uso fácil e intuitivo. En 2006, Gloogle Inc. adquirió la empresa, mejorando SketchUp y creando herramientas como Google 3D Warehouse, una base de datos online de modelos 3D creados con SketchUp, hasta el primer trimestre de 2012, cuando SkechUp y 3D Warehouse fueron comprados por la compañía Trimble, actual propietaria del software.

Durante el desarrollo de este proyecto se ha empleado la última versión disponible hasta el momento, Sketchup 8 (véase Figura 2.17), lanzada por Google en 2010; pero durante la finalización del mismo, una nueva versión fue lanzada al mercado. En mayo de 2013, la nueva propietaria Trimble, lanzó su primera versión del producto bajo la denominación de SketchUp 13 (no probada). Ahora la línea gratuita lleva el sobrenombre *Make*, mientras que la versión de pago sigue manteniendo su identificativo *Pro* [5].

La interfaz de Google SketchUp 8 es limpia y configurable. El modelado en este software se basa en formas geométricas rectangulares, circulares y polígonos, apoyada por trazado de líneas rectas, arcos y mano alzada. Se acompaña de herramientas como rotación, transportador de ángulos, medición, etiquetado, borrado, pintado, equidistancia, escalado, empujar/tirar e inserción de texto, que pueden verse en la Figura 2.18. Aunque posee algunas herramientas más complejas y sofisticadas que pueden obtenerse de [16], las citadas anteriormente son más que suficientes para crear infinidad de objetos y modelos.

Figura 2.18: Barra de herramientas de SketchUp 8

Además, dispone de multitud de herramientas de terceros integradas y que pueden conseguirse de [36], como exportación de la escena a programas de renderizado y postprocesado de imagen. Este es el caso de Kerkythea [3], aplicación que presentaremos a continuación.

2.4.2. Kerkythea

Se trata de un software de renderizado para modelos 3D. El plugin para SketchUp (véase Figura 2.20) realiza la exportación de la escena de modelado a un fichero XML que representa todos los modelos allí presentes. Una vez exportado, éste puede abrirse dentro de la aplicación Kerkythea, que es la encargada de realizar el renderizado de la escena.

Figura 2.19: Logo de Kerkythea

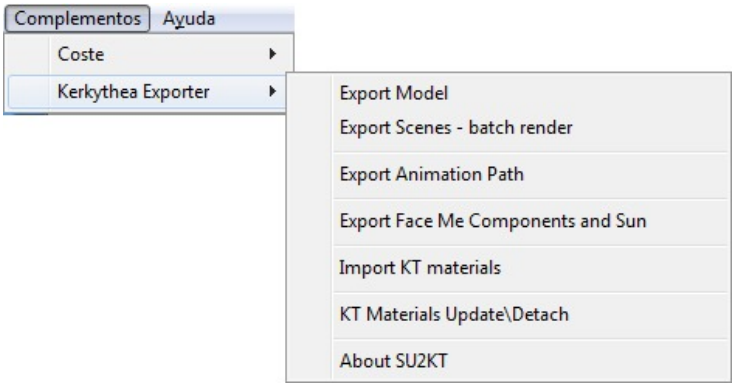


Figura 2.20: Plugin SU2KT SketchUp Exporter de Kerkythea

La principal diferencia encontrada entre este software y el resto, es que mientras que la mayoría exporta el encuadre que capta la cámara del programa de modelado, Kerkythea exporta toda la escena completa, pudiendo moverse libremente por ella una vez importada en Kerkythea. En la Figura 2.21 se muestra la escena de la Figura 2.17 una vez importada a Kerkythea.

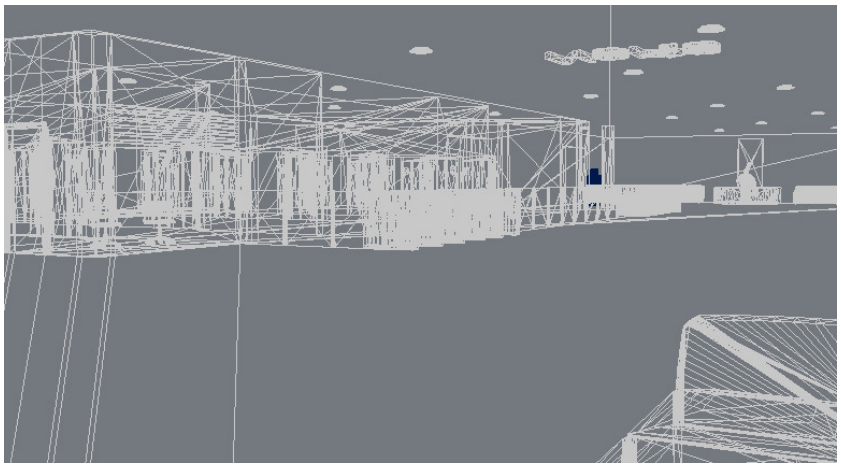


Figura 2.21: Escena exportada de SketchUp e importada Kerkythea

Otra diferencia que caracteriza a este software es que, a pesar de ser totalmente gratuito, no presenta limitaciones de uso, a diferencia de otros del mismo ámbito que limitan la resolución de la imagen renderizada a un tamaño predefinido que no suele ser elevado y debe abonarse una cantidad para poder desbloquear el límite de tamaño.

Y, aunque se trate de software gratuito, el nivel conseguido con sus renderizados es muy bueno. En la Figura 2.22 puede verse todo el potencial que puede desarrollar Kerkythea 2008 Echo, versión empleada en este proyecto (imagen obtenida de la web).

Figura 2.22: Escena renderizada con Kerkythea

Ha de aclararse que este software es prescindible en el proyecto, ya que su uso se ha basado en la generación de ciertas vistas renderizadas para tener una perspectiva más real de los escenarios y así ajustar mejor todos los componentes.

2.4.3. Autodesk FBX Converter

Una de las limitaciones que presenta Google Sketchup 8 gratuito es que no implementa la exportación a formato FBX (.fbx), que sí incorpora en la versión *Pro*. Sólo se tiene la posibilidad de exportar archivos COLLADA, un estándar con sintaxis XML [8] y cuya extensión de fichero es .dae, y archivos para Google Earth (.kmz). A pesar de que los ficheros COLLADA siguen un estándar, algunos programas manejan mejor los ficheros FBX.

Figura 2.23: Logo de Autodesk FBX Converter

Se trata de una herramienta muy útil y rápida para la conversión entre formatos, ya que no sólo realiza conversiones entre ficheros COLLADA y

FBX (como se ve en la Figura 2.24), sino que soporta ficheros propietarios de otras aplicaciones muy reconocidas como .3ds de Autodesk 3ds Max (más conocido como 3D Studio Max), .dxf de AutoCAD y .obj, estándar para modelos 3D. A pesar de todo su potencial, en este proyecto solamente se ha empleado para la conversión .dae → .fbx.

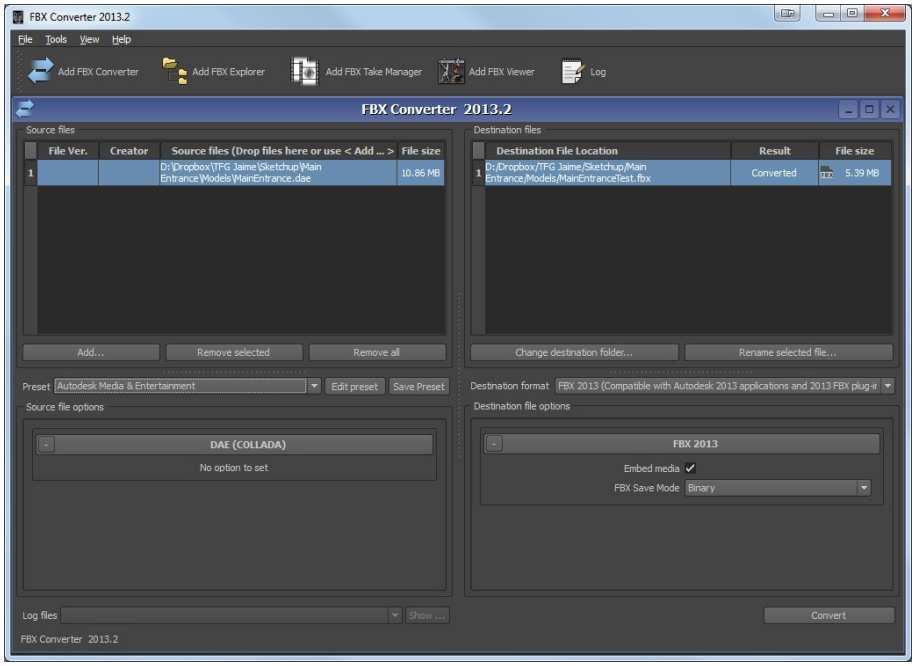


Figura 2.24: Autodesk FBX Converter transformando un fichero .dae a .fbx

2.4.4. Unity

Figura 2.25: Logo de Unity 4

El motor de videojuegos Unity, desarrollado por Unity Technologies es relativamente joven pues nació en 2004, fecha bastante más tardía que otros motores de videojuegos como Unreal Engine, cuya primera versión data de 1998 [6]. Pero esto no quiere decir que se trate de un software inmaduro, de hecho está creciendo a pasos agigantados a partir de su versión 3, y desde que se lanzase la versión 4 a mediados de 2012, las actualizaciones y mejoras son constantes. Actualmente está soportado para Windows y Mac.

Así, una de las grandes novedades que presenta Unity 4 es Mecanim, una tecnología avanzada para animación de avatares, además de añadir a Linux en la publicación para juegos de escritorio, junto con los ya existentes Windows y Mac OS X. En cuanto al precio de este software, existen dos tipos de licencia: Unity y Unity Pro.

La primera de ellas es gratuita y sus limitaciones están en cumplir con el pliego legal que la empresa exige (establece unos ingresos máximos por la empresa de 100.000 $ anuales) y algunas opciones como sombras en tiempo real que sólo están disponibles en la versión de pago.

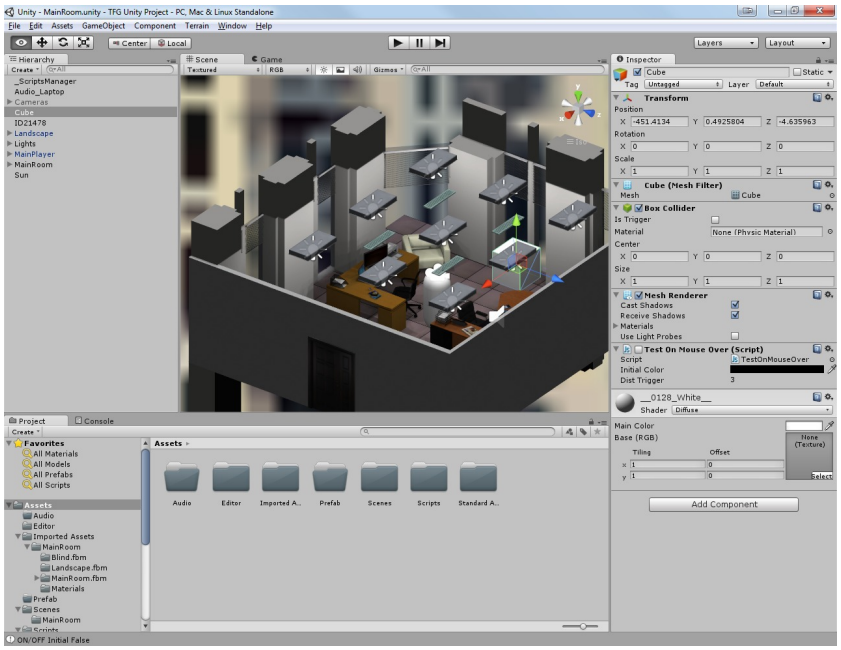


Figura 2.26: Entorno de desarrollo Unity 4

En cuanto a la versión Pro, ésta cuesta 1.500 $ para desarrollar juegos de escritorio y esa misma cantidad adicional para cada una de las plataformas móviles en las que se desea desarrollar (Android e iOS). Si bien es cierto que en la versión 4.1.3f3 se ha liberado la exportación gratuita a Android e iOS, con limitaciones similares a las versiones gratuitas de escritorio, evidentemente. En la Tabla 2.12 se muestra un resumen del tipo de licencias y en la Figura 2.27 pueden verse las opciones disponibles en la versión gratuita 4.1.3f3.

Licencia	Coste	Licencia	Coste
Unity (PC, Mac & Linux)	Gratuita	Unity Pro	1140 € 1500 $
Android	Gratuita	Android Pro	1140 € 1500 $
iOS	Gratuita	iOS Pro	1140 € 1500 $
Web Player	Gratuita	Flash Player	1140 € 1500 $
Google Native Client	Gratuita	Play Station 3	Contactar
Wii	Contactar	Xbox 360	Contactar

Tabla 2.12: Tipos de licencia de Unity.

Para el desarrollo de este proyecto, dado que se trata de simuladores educativos y los clientes potenciales no disponen de una financiación boyante en estos momentos, se ha tratado de emplear, siempre y cuando así lo permitiesen sus licencias de uso, software gratuito con el fin de abaratar costes y poder ofrecer un precio competitivo. Por ello, se ha empleado Unity 4 en su versión gratuita.

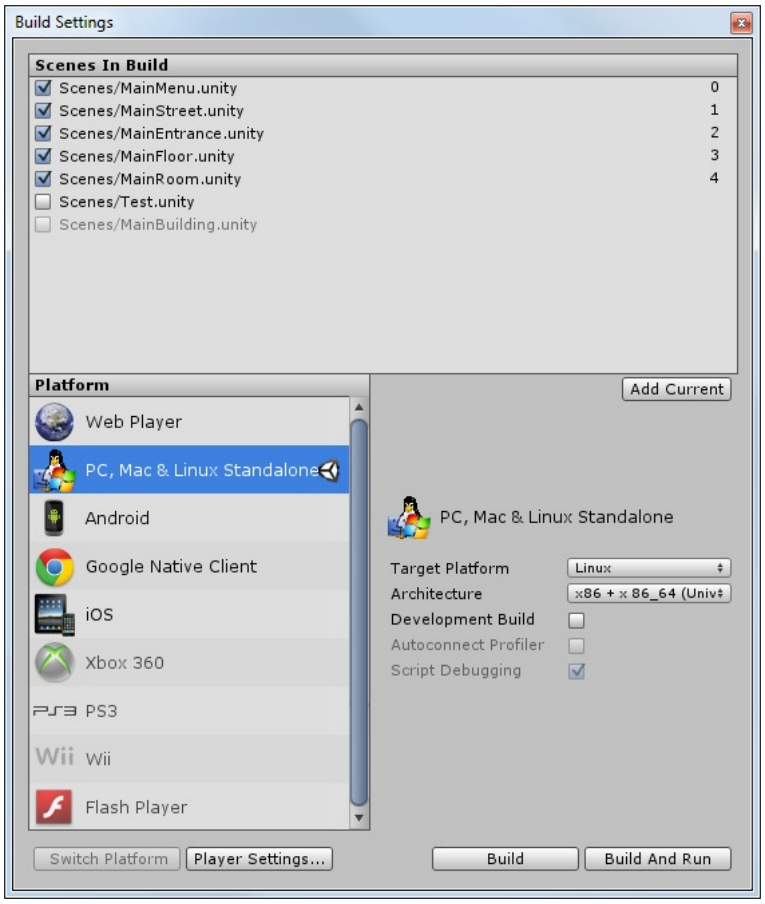


Figura 2.27: Opciones de exportación en Unity 4 gratuito

2.4.5. GIMP

GIMP, *the GNU Image Manipulation Program* es un software de edición de imágenes y retoque fotográfico gratuito y multiplataforma [33]. Su versión actual es la 2.8 y últimamente va ganando terreno a otros programas de edición fotográfica gracias a su simplicidad y alto potencial.

Figura 2.28: Logo de GIMP

Dispone de decenas de filtros preconfigurados y resulta muy cómodo para redimensionar imágenes y otro tipo de retoques básicos. Otros más avanzados, como correcciones por efectos de lente son tratados también con suma facilidad. En la Figura 2.29 se puede ver la interfaz de usuario del programa mientras se retocaba una de las texturas del videojuego.

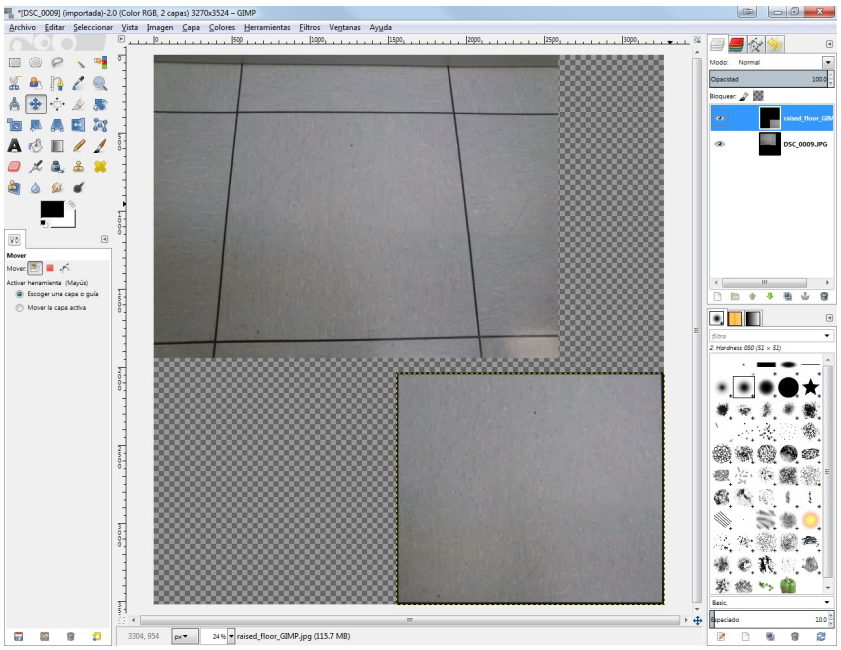


Figura 2.29: Retoque de texturas con GIMP

El retoque de texturas no es la única tarea realizada con GIMP. Se ha empleado para todo tipo de cometidos gráficos, como recortado, escalado y composición de imágenes.

2.4.6. Inkscape

Cuando se desea desarrollar logotipos o cualquier otro tipo de gráfico vectorial, esto es, que pueda ser redimensionado sin perder calidad, Inkscape puede ser una de las soluciones más rápidas y funcionales. Se trata de un editor de gráficos vectoriales de código abierto y gratuito que es compatible con el estántar de este tipo de gráficos, el SVG -Scalable Vector Graphics-.

Figura 2.30: Logo de Inkscape

En la Figura 2.31 podemos ver cómo se ha diseñado el logotido de *Entrepreneur Simulator 3D*, título del simulador de este proyecto.

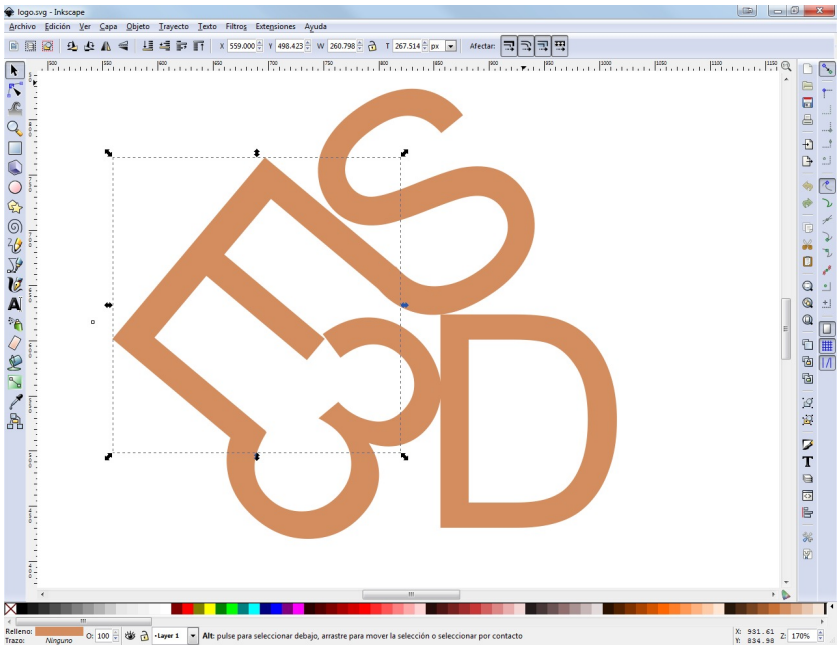


Figura 2.31: Logotipo de ES3D siendo editado con Inkscape

Capítulo 3

Diseño de la solución técnica

3.1. Arquitectura del simulador

El primer paso antes de ponerse a desarrollar cualquier aplicación debe ser reflexionar sobre la estructura que la vertebrará. Una buena estrategia siempre suele ser la de *divide y vencerás*, de tal forma que un problema se divida en pequeños problemas cuyas soluciones, en su conjunto, aporten una solución global.

Si bien esta forma de actuar es aplicable a todos los desarrollos de software y, en general, a cualquier contexto cotidiano, toma especial relevancia cuando la aplicación a desarrollar debe ser tan modular que permita modificar, crear y eliminar ciertos contenidos del mismo sin que afecte al buen funcionamiento de los demás módulos.

En el caso que nos ocupa, se ha divido el sistema en varios módulos independientes que se complementan unos a otros, dando forma en su conjunto a la funcionalidad completa del simulador. Así, a grandes rasgos, se puede dividir la aplicación en dos bloques bien diferenciados: el Subsistema Simulador y los Recursos Externos que dan sentido al mismo, tal y como puede verse en la Figura 3.1 junto a los módulos más importantes de cada subsistema.

El Subsistema Simulador abarca todos los recursos embebidos y empaquetados con la aplicación, no siendo modificables por el usuario y que, a su vez, dotan de funcionalidad y capacidad de interpretación al simulador. En cambio, Recursos Externos engloba todos aquellos archivos que se encuentran en el directorio `ExtResources`, siendo modificables por el usuario y aportando el contenido y el flujo de los diferentes escenarios de la aplicación.

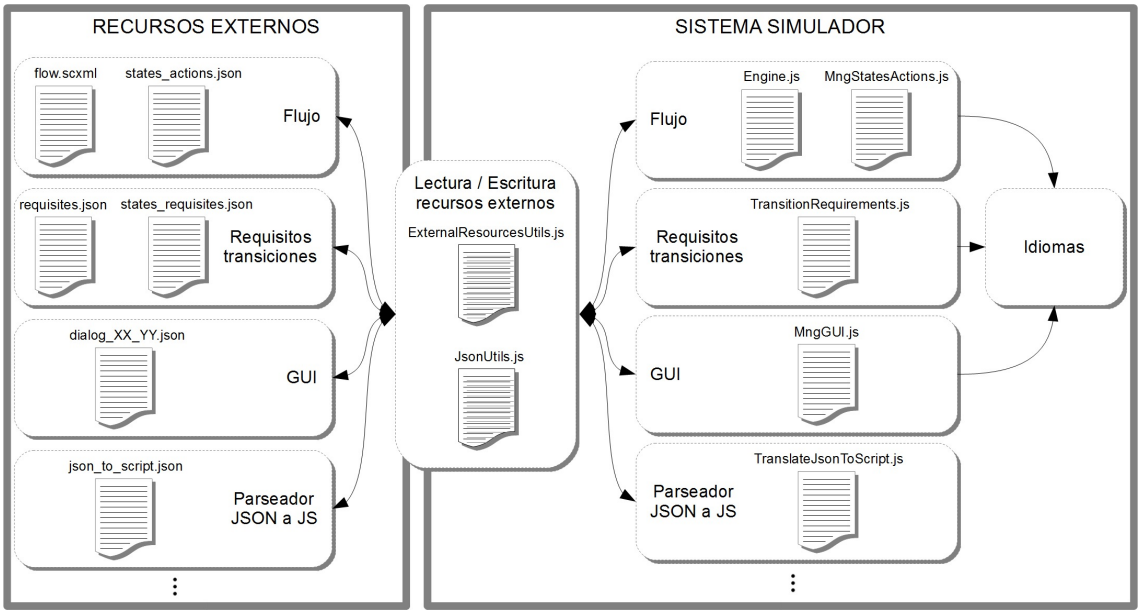


Figura 3.1: Módulos más importantes de la arquitectura del simulador

Para obtener una visión global de los ficheros que componen el simulador, la Figura 3.2 muestra la estructura completa de directorios y archivos.

Figura 3.2: Organización de ficheros externos (izda.) y scripts internos (dcha.)

Por otro lado, la clasificación de los módulos que se ha realizado en dicha arquitectura debe entenderse como una organización de los ficheros según su funcionalidad. Por ello aparecen los módulos representados en ambos subsistemas, aunque la capacidad real resida sobre los archivos de la parte del simulador, que son los que poseen la lógica para tratar dinámicamente los ficheros externos.

A continuación se describen cada una de las unidades modulares que componen el sistema global y se acompaña de porciones de código de algunos de los ficheros que los componen.

Módulo de Lectura/Escritura de recursos externos

Es la puerta de enlace entre el Subsistema Simulador y los Recursos Externos, pues es el que posee las funcionalidades necesarias para escribir y leer ficheros que no estén embebidos en la aplicación.

De hecho, se encarga de copiar los recursos externos a un directorio alcanzable por el usuario la primera vez que la aplicación se ejecuta. Posteriormente, cada vez que ésta arranca, comprueba si existen todos los ficheros externos predefinidos y, de no ser así, vuelve a regenerarlos.

Esto es especialmente útil si el usuario quiere volver a la versión original del simulador, pues basta con eliminar la carpeta `ExtResources` y la aplicación volverá a crearla con todos los ficheros tal y como fueron dispuestos en la fase de desarrollo.

Módulo de Flujo

Se trata de la parte encargada de leer los ficheros *flow.scxml* que se encuentran dentro de cada escenario y que definen la máquina de estados y sus transiciones de la historia que seguirá el simulador.

La definición de los estados y sus transiciones queda reflejada en la Figura 3.3, donde se han omitido partes del código debido a su extensión. En ella pretende ilustrarse la relación existente entre los eventos, el estado objetivo de dicho evento (*target*) y los propios estados.

En ella puede verse que estando en el estado inicial, cuando el evento 'traffic_light_trigger' es lanzado, el flujo debe pasar al estado 'state_hud_traffic_light_show', ya que es el estado objetivo de la transición asociada al evento disparado.

También se encarga de manejar y asociar todas las acciones que deben ser llevadas a cabo en cada uno de los estados, definidas en el fichero *states_actions.scxml*. Así, cuando el simulador dispare un evento y este módulo lo recoja y coteje con las transiciones definidas para el estado actual, pondrá en disposición el salto al siguiente estado.

```
<state id="state_initial">
    <transition event="traffic_light_trigger"
        target="state_hud_traffic_light_show" />
    <transition event="mainbuilding_entrance_ext"
        target="mainbuilding_entrance_ext" />
    <transition event="object_entrepreneuradvertisement"
        target="ap_00_00" />
</state>

<state id="state_hud_traffic_light_show">
  <transition event="cross_road" target="state_traffic_light_go_red" />
  <transition event="cancel" target="state_hud_traffic_light_hide" />
</state>
                                ...
<state id="ap_00_00">
  <transition event="option_A" target="ap_00_01" />
  <transition event="option_B" target="state_initial" />
</state>

<state id="ap_00_02">
  <transition event="option_A" target="ap_00_03" />
  <transition event="option_B" target="ap_00_04" />
</state>
```

Figura 3.3: Código de *flow.scxml* del escenario *MainStreet*

```
{
"state_hud_traffic_light_show":"API.ShowGUI(\"hud_traffic_light\");"
,
"mainbuilding_entrance_ext":"API.ChangeToLevel(\"MainEntrance\");"
,
"ap_00_00":"API.RequisitePassed(\"req_00_00\");
                API.ShowGUI(\"dialog_00_00\");"
,
"ap_00_01":"API.PlayerAction(\"call\");"
,
                                ...
"ap_00_06":"API.PlayerAction(\"hang_up\"); API.SendEvent(\"initial\");"
}
```

Figura 3.4: Código de *states_actions.json* del escenario *MainStreet*

Módulo de Requisitos entre transiciones

Una vez está preparado el siguiente estado, la tarea a realizar es la comprobación de los requisitos que deben cumplirse para poder acceder al mismo. Los objetivos de cada uno de los escenarios se encuentran definidos en los ficheros *requisites.json* de cada nivel y están asociados a cada estado de la historia en los archivos *states_requisites.json*, como puede verse en las Figuras 3.5 y 3.6.

```
{
"requisite":
[
    {
    "name": "req_00_00",
    "desc_en": "Get an incentive to think about undertake",
    "desc_es": "Conseguir un aliciente para pensar en emprender"
    },
    {
    "name": "req_00_01",
    "desc_en": "Get a contact to help you",
    "desc_es": "Conseguir un contacto que te ayude"
    }
]}
```

Figura 3.5: Código de *requisites.json* del escenario *MainStreet*

De esta forma, si el usuario ha conseguido los objetivos necesarios, se accederá al nuevo estado; de lo contrario, el juego permanecerá en el estado actual hasta que otro evento sea lanzado y la comprobación de sus requisitos resulte satisfactoria.

```
{
"state":
[
    {
    "name": "mainbuilding_entrance_ext",
    "requisites":
            ["req_00_00",
            "req_00_01"]
    }
]}
```

Figura 3.6: Código de *states_requisites.json* del escenario *MainStreet*

Módulo GUI

Se encarga de la Interfaz Gráfica de Usuario -Graphical User Interface- y su misión es, con la ayuda del *Parseador JSON a JS* representar la información que aparece en los ficheros *dialog_XX_YY.json*[1] como interfaces gráficas en el juego.

En la Figura 3.7 se presenta uno de estos ficheros de ejemplo con el fin de ilustrar cómo se definen las interfaces gráficas del simulador.

```
{ "GUI":
    [   { "Box": {
                "width":"240",
                "height":"180",
                "posX":"center",
                "posY":"center",
                "text_en":"Traffic Light Controller",
                "text_es":"Controlador del Semáforo"
            }
        },{ "Button": {
                "width":"220",
                "height":"70",
                "posX":"center",
                "posY":"center-30",
                "text_en":"Press button and wait green",
                "text_es":"Presionar botón y esperar verde",
                "action": "API.SendEvent(\"cross_road\");"
            }
        },{ "Button": {
                "width":"220",
                "height":"70",
                "posX":"center",
                "posY":"center+50",
                "text_en":"Cancel",
                "text_es":"Cancelar",
                "action":"API.SendEvent(\"cancel\");"
            }
        }    ]
}
```

Figura 3.7: Código de *hud_traffic_light.json* del escenario *MainStreet*

[1]Nombre del fichero donde XX denota el número de escenario del simulador y donde YY representa el número de interfaz de ese nivel. El nombre de estos archivos puede ser cualquiera que sea representativo del contenido que incluye.

La representación de éstas viene determinada por el estado actual del flujo del simulador y la interacción del usuario con ellas repercute sobre el sistema. Así, si al usuario se le presentan dos botones, los cuales llevan asociados un evento relacionado a su vez con una transición de estado, cuando éste presione uno de ellos estará eligiendo por qué rama de la bifurcación del flujo quiere ir.

Por ejemplo, en la Figura 3.8 se presenta la interfaz gráfica que genera el código anterior (Figura 3.7). En ella se le presenta al usuario una bifurcación en el flujo de la historia. Si pulsa el botón superior, se cambiará al estado que controla los semáforos, mientras que si pulsa el botón inferior, el juego volverá al estado anterior.

Figura 3.8: Interfaz del controlador de los semáforos

Módulo Idiomas

Este módulo no lleva asociado ningún fichero en sí, ya que su funcionalidad se halla repartida entre varios scripts. Su función es representar en pantalla la versión correspondiente al idioma seleccionado por el usuario de los contenidos que se encuentran como recursos externos.

El idioma de la aplicación puede ser seleccionado desde el menú principal del juego, donde se podrá elegir entre los diferentes idiomas definidos en *menu_translations.json*, fichero que se muestra en la Figura 3.9.

```
{
"SupportedLangs": {
    // Must be separated by '-' (i.e. "AvailableLangs":"en-es-fr")
    "AvailableLangs":"en-es"
    },

"MainMenu_Texts": {
    //"ButtonName_lang":"text",
    "ResumeGame_en":"Resume Game",
    "ResumeGame_es":"Continuar Juego",

    "StartNewGame_en":"Start New Game",
    "StartNewGame_es":"Comenzar Juego Nuevo",

    "Exit_en":"Exit",
    "Exit_es":"Salir",

    "ChangeLang_en":"Change Language",
    "ChangeLang_es":"Cambiar Idioma",

    "ConfirmNewGame_en":"All current progress will be deleted.
                        Are you sure?",
    "ConfirmNewGame_es":"Todo el progreso actual será eliminado.
                        ¿Está seguro?"
    }
}
```

Figura 3.9: Código de *menu_translations.json*

Así, antes de que, por ejemplo, una GUI sea mostrada en pantalla, se ha escogido cuál es el contenido adecuado para la misma en función del idioma de la aplicación. Esto es muy útil, ya que permite añadir tantos idiomas como se deseen y el sistema buscará el contenido correspondiente.

Módulo Parseador JSON a JS

Su función es traducir las claves y valores definidos en los ficheros con estructura JSON a código JavaScript (UnityScript[2] concretamente) que pueda ser interpretado por el sistema simulador. Estas traducciones se encuentran en el fichero *json_to_script.json* (véase Figura 3.10).

Se emplea exclusivamente por el módulo de interfaces gráficas ya que es el único módulo que necesita traducir etiquetas de JSON a scripts completos. El resto de módulos se basan en el almacenamiento de las etiquetas y valores de las estructuras JSON para su empleo cuando sea necesario, ya que contienen contenido y no código que necesite ser interpretado (esto no quiere decir que no sean procesados).

A continuación se muestra el código que, una vez interpretado por el script correspondiente dentro del simulador, genera las Interfaces Gráficas de Usuario. Para comprender mejor este código se recomienda consultar en la Figura 3.31 el modo en que se definen dentro de Unity dichas interfaces gráficas.

```
// element : unityscript code
{
    "box":"GUI.Box(Rect($posx, $posy, $width, $height), \"$text\");",
    "button":"if(GUI.Button(Rect(...), \"$text\")){$action};",
    "label":"GUI.Label(Rect(...), \"$text\");",
    "textfield":"textFieldVar = GUI.TextField(Rect(...), textFieldVar);",
    "textarea":"textAreaVar = GUI.TextArea(Rect(...), textAreaVar);",
    "repeatbutton":"if(GUI.RepeatButton(Rect(...), \"$text\")){$action};"
}
```

Figura 3.10: Código de *json_to_script.json*

Puede observarse que los campos *textfield* y *textarea* son distintos a los demás en cuanto a que existe una palabra que precede a '*GUI.Text...*'. Esto se debe a que son campos de texto y para cumplir con el requisito de la Tabla 2.8 deben definirse estas variables para guardar temporalmente los datos hasta que sean almacenados en las preferencias de la aplicación.

[2]Es un lenguaje de programación propio del entorno de desarrollo Unity basado en JavaScript y con opciones de tipado. Es un lenguaje precompilado, lo que hace que su ejecución sea mucho más rápida que JavaScript tradicional. Además, acepta el uso de librerías de .NET, haciéndolo mucho más versátil y completo para trabajar con el propio Sistema Operativo.

3.2. Creación de los escenarios

Una vez queda definida la arquitectura del simulador hay que comenzar a darle forma. Todo simulador debe tener contenidos visuales, ya sean 2D, 3D, simples textos, etc. Como ya se comentó en la Sección 2.4, para la creación de los modelos 3D se ha empleado Google Sketchup 8.

Una de las primeras consideraciones a tener en cuenta son los escenarios que van a aparecer en el simulador. Dado el carácter empresarial del mismo, se pensó en crear una oficina, habitación cuyo principal objetivo sería que el usuario pudiera interactuar con los objetos para proporcionarle información rápida. Pero esta habitación debería situarse en un contexto espacial. Para ello se pensó realizar un edificio entero donde ubicar la oficina dentro de una de las plantas del mismo.

A su vez, para llegar a dicha planta, el jugador debería pasar antes por la planta baja del edificio, con lo que surge el escenario del hall. Por último, para dar mayor envergadura a los escenarios, se completarían con un escenario de la calle donde se ubicaría el edificio.

Así, dentro de esta sección se presentará el desarrollo de cada uno de los escenarios que conforman el simulador.

3.2.1. Edificio de oficinas

Para modelar el edificio de oficinas que será el principal foco de atención en el escenario global, se quiso recrear algún edificio real que fuese, a la par que simbólico para una ciudad, fácil de reproducir y con suficiente documentación arquitectónica para poderla modelar con exactitud.

Dado el contexto geográfico donde se estaba llevando a cabo del desarrollo del simulador, y tras algunas búsquedas de información en la web, se eligió la Torre Picasso de Madrid, inaugurada a finales de 1988 y diseñada en 1974 por el arquitecto Minoru Yamasaki [4]. Gracias a la información recopilada de [1] acerca de los materiales y dimensiones de las estructuras externas y a los planos e información técnica ofrecidos por [19] y [18], se procedió al modelado de la misma.

En la Figura 3.11 puede verse la base del edificio, el núcleo principal del mismo, la zona de ascensores y los pilares exteriores. Para hacerse un idea de las dimensiones del edificio, basta con detallar que las partes centrales de los pilares exteriores presentan una separación de 3 m.

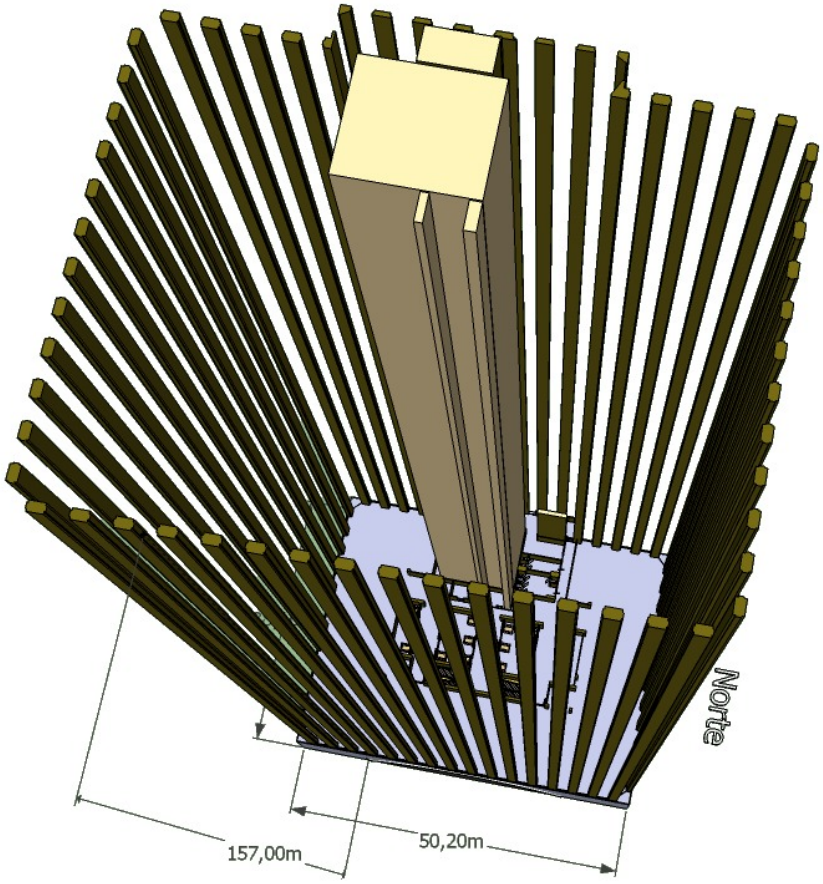


Figura 3.11: Estructura principal de la Torre Picasso

Aunque se ha querido ser minucioso en la reproducción del edificio (véase comparación de áreas en la Figura 3.13), siendo consciente de lo que representa el modelado dentro de la totalidad del proyecto, es obvio que solamente podría desarrollarse una planta para que el jugador anduviese por ella. Por ello, y debido a que los planos de la Torre Picasso nos muestran tres zonas con diferente distribución espacial (plantas 2 a 18, plantas 19 a 32 y plantas 33 a 43), se optó por modelar una planta representativa basándose en el plano de las plantas 2 a 18.

Tras realizar el modelado de toda la estructura, comenzaba el trabajo de texturizado del modelo. Éste se basa en aplicar pintura o imágenes sobre las caras de los objetos, como muestra la Figura 3.12 con la textura de la fachada, obtenida de imágenes de otros edificios revestidos exteriormente con el mismo material. La técnica consiste en recortar la imagen a la zona de interés y definir el tamaño de la textura para que ésta se repita cada cierto espacio. Finalmente, si es necesario, debe desplazarse la textura para hacerla coincidir con el modelo.

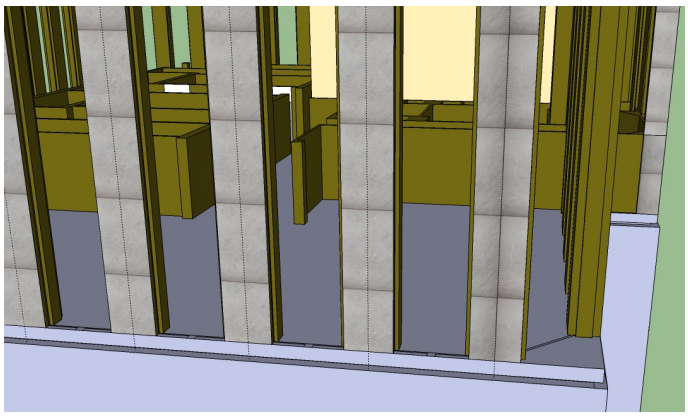


Figura 3.12: Aplicación de texturas sobre la fachada

Una vez retocados todos los pequeños detalles que pudieran hacer que el modelo perdiese similitudes respecto al verdadero, en la Figura 3.14 puede verse una previsualización del modelo acabado junto al real.

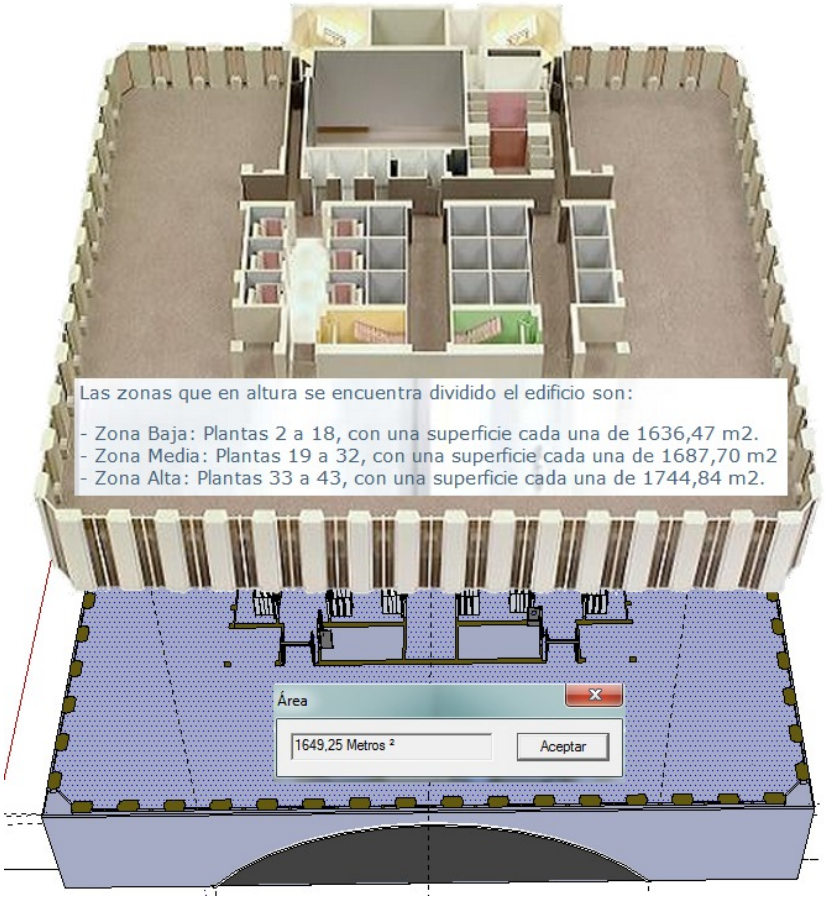


Figura 3.13: Áreas real (arriba) y modelada (abajo) (similitud del 99.2 %)

Figura 3.14: Edificio modelado (izda.) y Torre Picasso fotografiada (dcha.)

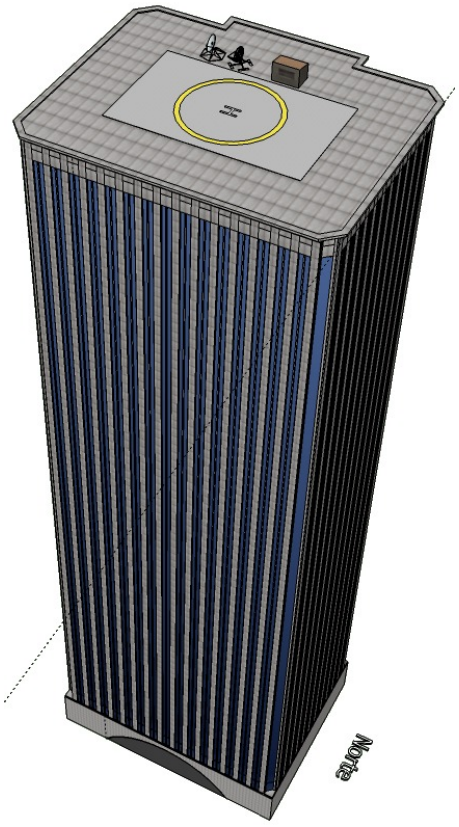


Figura 3.15: Modelado de la Torre Picasso finalizado

El modelo completo del edificio es el de la Figura 3.15, donde, a modo de información legal, todos los grandes espacios y texturas de éstos son propios pero los modelos del mobiliario del edificio se han obtenido desde 3D Warehouse, no pudiendo atribuirse la propiedad al autor de este documento.

3.2.2. Oficina

Una vez completado el modelo del edificio, se utilizó éste como plantilla para realizar una habitación, pues ya se tenía la estructura exterior y solamente había que levantar paredes interiores. Pero esto no quiere decir que fuese una tarea fácil, pues este escenario es de los más complejos en cuanto a mobiliario se refiere.

Una vez más, se ha modelado la habitación y se ha empleado la base de datos gratuita de 3D Warehouse para amueblar la oficina, a excepción de ambos escritorios, cuyos modelos son propios basados en [27] y [28], tal y como puede verse en la Figura 3.16. Y, aunque trabajar con modelos ya creados obviamente descarga de mucho trabajo, aún así se requirió tiempo para modificar y adaptar algunos modelos al contexto y para colocarlos correctamente.

Figura 3.16: Modelado del escritorio A a semejanza del catálogo

Una vez colocados todos los objetos para cumplir cada uno con una funcionalidad, la oficina presenta la distribución de la Figura 3.17, aunque posteriormente sufrió ligeros cambios mientras se trabajaba con ella en el motor de videojuegos.

Figura 3.17: Oficina una vez modelada

3.2.3. Planta del edificio

Al igual que sucediera anteriormente, aprovechando que ya se tenía modelado el edificio completo, se usó como plantilla y se reutilizó para modelar este escenario. Básicamente se seccionó la estructura de la torre a 3 m desde el suelo que había encima de la base y se aplicó un techo de unos 20 cm de grosor (véase Figura 3.18).

Aunque la fase de modelado fue sencilla, no resultó fácil ni rápido eliminar todos los hilos sobrantes de los pilares exteriores. Además, hubo que recolocar las texturas de la fachada interior para ajustarla al centro de los pilares, tal y como se muestra en la Figura 3.19.

Se trata del escenario con mayor número de tareas repetitivas y, por tanto, el que resultó más laborioso.

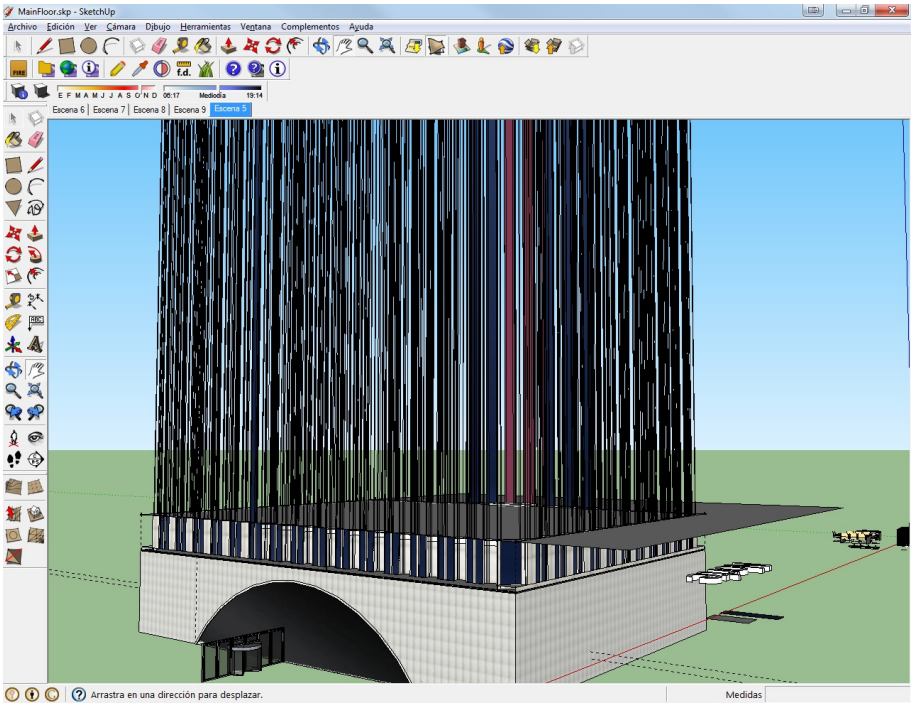


Figura 3.18: Modelado de la planta de oficinas

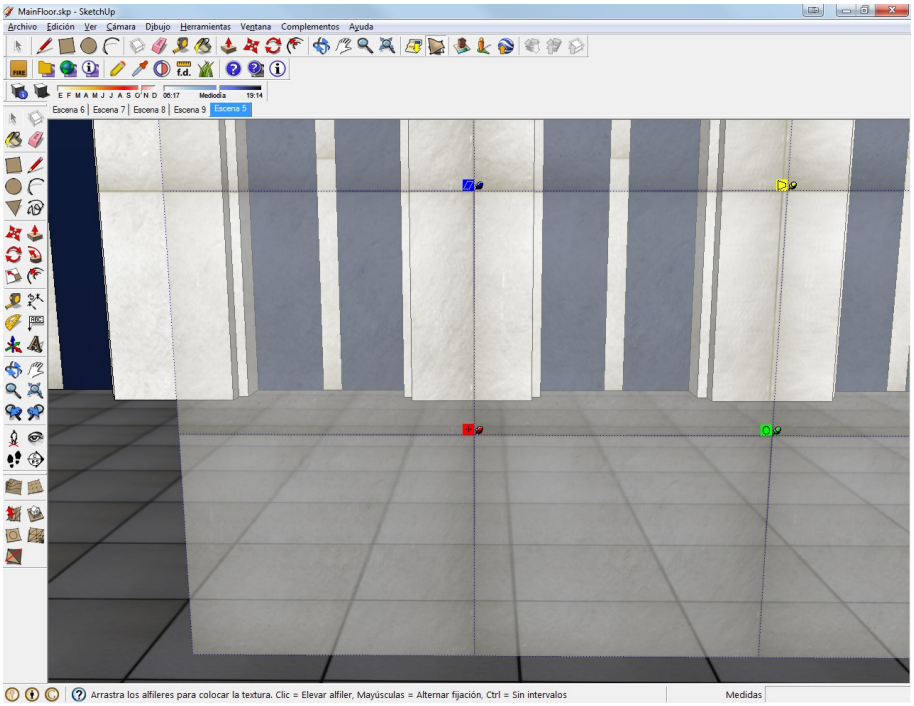


Figura 3.19: Ajuste de las texturas

El resultado final de este escenario modelado aparece en la Figura 3.20.

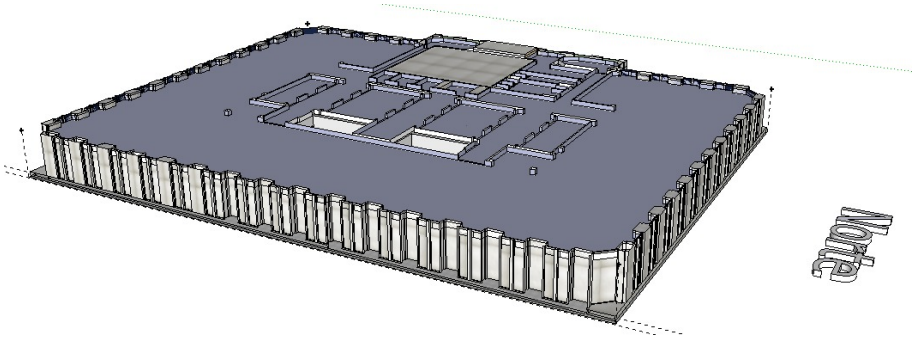


Figura 3.20: Vista exterior de la planta del edificio

3.2.4. Hall del edificio

Antes de llegar a cualquiera de las plantas del edificio, se debe pasar obligatoriamente por la entrada del mismo. Esa es la función que tiene este escenario: servir de paso al jugador entre la calle y la oficina.

Otra vez más, teniendo como plantilla la parte baja del edificio modelado con anterioridad, se procedió a dar forma al hall de éste. Para asemejarse a cualquier entrada de un edificio de oficinas, debía cumplir, al menos, con la existencia de una recepción y varios tornos para acceder al recinto interior. En la Figura 3.21 pueden verse los objetos modelados para esta ocasión.

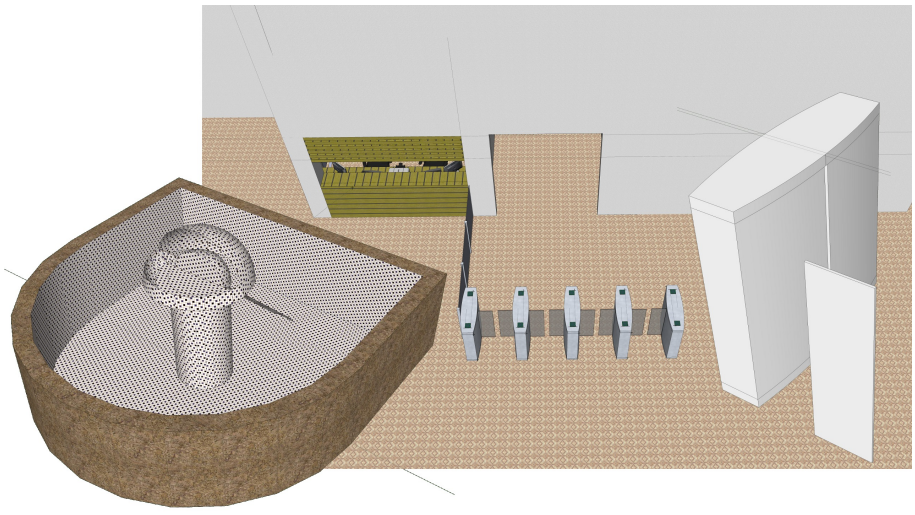


Figura 3.21: Objetos modelados para este escenario

Los tornos se modelaron a semejanza del catálogo disponible en [9], mientras que las mamparas de cristal fueron inventadas. Se completó el mobiliario

con sillones, papeleras, ordenadores y la puerta giratoria descargados desde 3D Warehouse, luciendo el acabado que se presenta en la Figura 3.22.

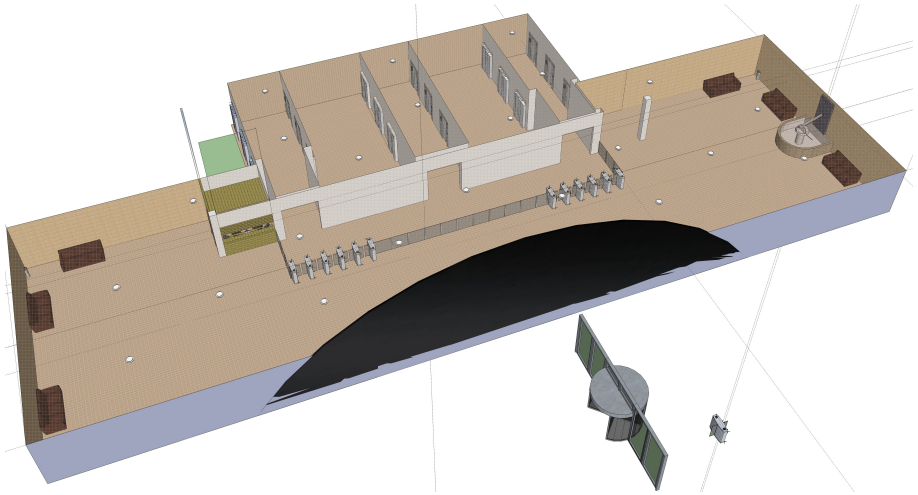


Figura 3.22: Entrada principal del edificio

3.2.5. Calle

El último de los escenarios a modelar era el espacio más amplio pero fácil de todos: la calle. El mayor trabajo de este escenario fue el texturizado, ya que para el modelado bastaba con realizar cubos para los edificios y aceras.

Pero uno de los mayores retos no tenía relación con las tareas de modelado y texturizado, si no con qué límites le iban a ser impuestos al jugador para contenerle dentro del escenario. En un principio se pensó implementar el escenario que se esbozó a tal fin (véase Anexo B: *Esbozo del escenario de calle*), pero fue cambiado dinámicamente por el escenario de la Figura 3.23, donde los límites vienen determinados por dos túneles exclusivos para vehículos.

Las calles, aceras, pasos de peatones, boca de metro, túneles y edificios son de modelado propio, mientras que las farolas, papeleras y resto de mobiliario urbano fueron obtenidos nuevamente de 3D Warehouse.

Por otro lado, este escenario fue uno de los que más problemas dio al importarlo al motor de videojuegos, tarea que será detallada en la Sección 3.3.

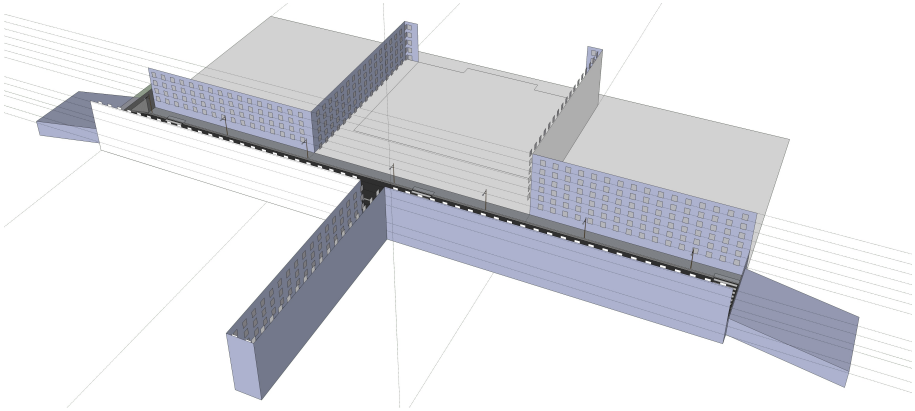


Figura 3.23: Borrador del escenario de calle

3.3. Importación de los escenarios

Una vez terminados todos los escenarios, el siguiente paso es darles utilidad. En general, la finalidad de la creación de modelos 3D es emplearlos principalmente de tres formas distintas: realizar renderizado de imágenes, renderizado de secuencias de vídeo o incluirlos como escenarios de videojuegos, como es el caso.

Para ello, hay que ver qué compatibilidad tiene el software de destino y qué posibilidades de exportación posee la herramienta de modelado empleada. Google Sketchup 8, en su versión gratuita solamente permite exportar ficheros COLLADA (.dae) y Google Earth (.kmz), aparte del formato propietario de proyectos Sketchup (.skp).

A su vez, Unity es compatible con la importación de ficheros .fbx, .dae, .3DS, .dxf y .obj, por lo que no debería haber problemas si se exportan los modelos como ficheros COLLADA, como se muestra en la Figura 3.24.

Cuando se procede a realizar la tarea, es cierto que en cuanto a la geometría de los modelos no existe problema alguno, pero la importación de las texturas resulta complicada. El problema reside en que los ficheros .dae llevan asociada una carpeta `Textures` que al importar a Unity no se lee correctamente.

El procedimiento para solucionarlo debe ser el siguiente:

1. Crear/Abrir un modelo 3D con Sketchup.
2. Exportar en formato .dae y seleccionar las opciones de exportación que convengan, pero siempre activar la opción *Exportar mapas de textura.*

3. Abrir Unity y crear una nueva carpeta dentro de la pestaña de *Assets* para guardar los modelos.

4. Arrastrar/Importar dentro la carpeta anterior tanto el fichero .dae como la carpeta *Textures* generada al exportar el modelo.

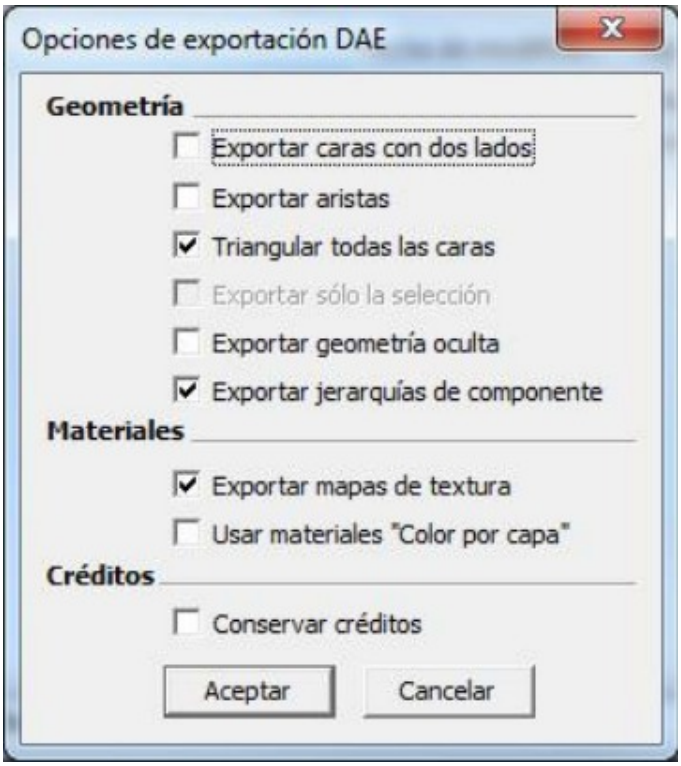


Figura 3.24: Opciones de exportación para ficheros .dae en Sketchup 8

La estructura de ficheros debería resultar como la de la Figura 3.25.

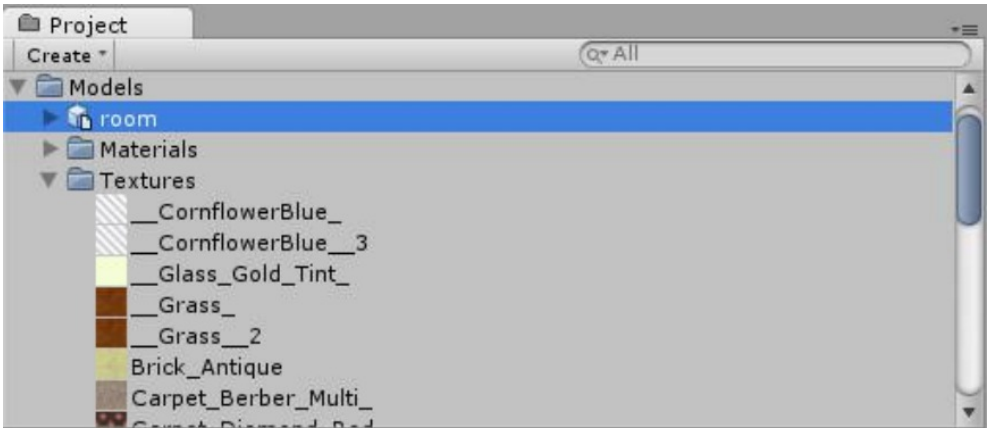


Figura 3.25: Estructura tras importar el modelo 3D

Pero existe otra forma de realizar esta misma operación si el modelo lo importamos en Unity como .fbx. Para ello, primero debe instalarse el programa Autodesk FBX Converter, del que se habló más detalladamente en la Sección 2.4, y después hay que seguir los pasos que se enumeran a continuación:

1. Crear/Abrir un modelo 3D con Sketchup.
2. Exportar en formato .dae y seleccionar las opciones de exportación que convengan, pero siempre activar la opción *Exportar mapas de textura*.
3. Convertir el fichero .dae a .fbx con Autodesk FBX Converter (ver Figura 2.24), teniendo seleccionada la opción *Embed media* para que incluya las texturas dentro del propio fichero.
4. Abrir Unity y arrastrar/importar el fichero .fbx generado.

Aunque el número de pasos resultantes, obviando la instalación del conversor, es el mismo, resulta más práctico y cómodo al tener un solo fichero sin tener que preocuparse de la estructura de directorios.

Pero esta no es la principal razón para acometer el segundo de los procedimientos. Los ficheros .fbx, aún embebiendo las texturas, ocupan mucho menos que los .dae (en modelos del proyecto en torno 3 a 4 veces menos).

Además, como ya comentamos en el Apartado 2.4.4, Unity 4 incorpora una nueva herramienta: Mecanim. Ésta sólo es compatible con la importación de ficheros .fbx, pudiendo establecer animaciones al modelo, dotándolo de movimientos humanos.

Texturas y animaciones aparte, algo muy importante a tener en cuenta es la escala de los objetos que se importan para que todos coincidan. Por defecto, cuando se importa un modelo, Unity le asigna el valor de escala 0.1 pero hay que comprobar que esa es la escala que necesitamos. Una buena forma es comparar el tamaño de nuestro modelo con el de un objeto primario creado en Unity, como por ejemplo un cubo, cuyas lados miden 1 unidad.

Esa unidad puede tomarse equivalente a lo que más se desee, ya sean centímetros, metros, pulgadas, pies... pero debe ser la referencia para todos los modelos de la escena. Así, si se crean modelos en Sketchup con la plantilla de éste configurada en metros, la sorpresa al importarlos en Unity es que son bastante más grandes de lo esperado.

La explicación reside en las unidades de medida en Unity equivalen a pulgadas [22], por lo que para adecuar los modelos creados en metros debe aplicarse la escala de la Ecuación 3.1 en el campo *Scale Factor* de las opciones de importación (ver Figura 3.26). De esta forma se consigue que el lado del cubo por defecto pueda suponerse de 1 m de lado y los modelos que se habían importado estén referenciados como múltiplos de esa magnitud.

$$Escala = Escala\ por\ defecto \times \frac{2.54\ cm}{1”} \times \frac{1\ m}{100\ cm} = 0.0254\ (m/”) \qquad (3.1)$$

En cuanto al rendimiento, en la documentación oficial de Unity [38] aconsejan que los objetos no se reescalen, pero si ha de hacerse, el que menor impacto presenta es el ajuste del factor de escala al importar el modelo, después el ajuste de escala simétrico de todos los ejes de la transformación del objeto y, finalmente, el escalado asimétrico de cada eje en la transformación del objeto es el que supone mayor impacto en el rendimiento.

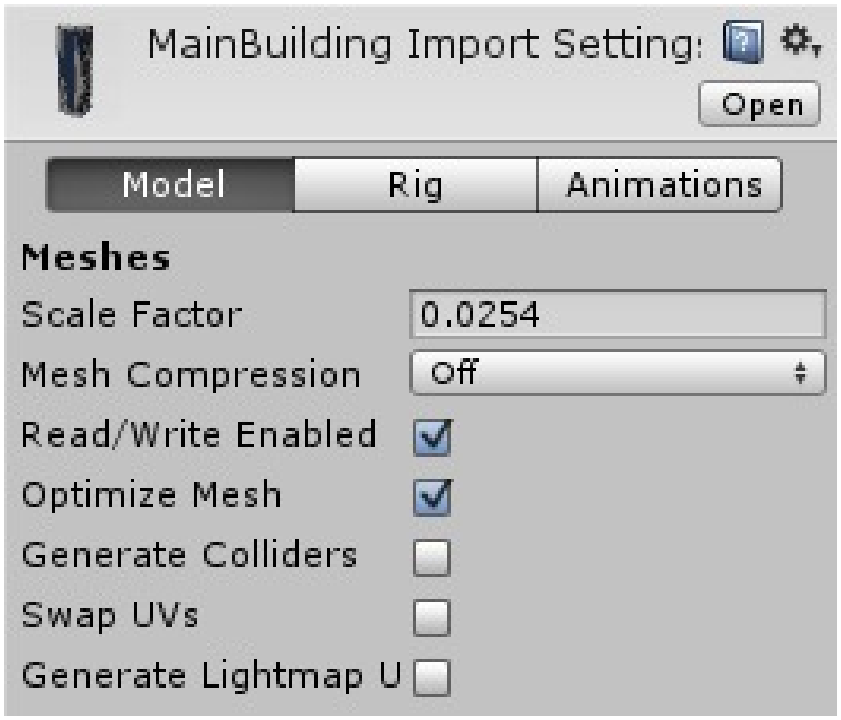


Figura 3.26: Algunas de las opciones de importación de modelos en Unity 4

3.4. Desarrollo en Unity

3.4.1. Ajustes y mejoras visuales de los escenarios

Hasta el momento todas las labores se habían centrado en preparar todo lo necesario para el trabajo con Unity. Llegados aquí, fue hora de emplear Unity para el desarrollo de la aplicación.

Lo primero de todo fue crear un nuevo proyecto e importar los escenarios, cada uno de ellos en una nueva escena. Aún siguiendo todos los paso descritos en la Sección 3.3, el proceso de importación no fue sencillo y hubo que dedicar bastante tiempo a solucionar diferentes inconvenientes.

Unos modelos tuvieron que exportarse aparte de la escena e importarlos por separado, como fue el caso del suelo de la oficina (véase Figura 3.27) que hubo que volverse a modelar aparte e importarlo como un objeto separado. A otros modelos hubo que aplicarles las texturas desde el propio Unity, como ocurrió a todos los edificios del escenario de calle, ya que al importarlos presentaban todos la fachada gris que por defecto se asigna a los objetos sin texturas.

Figura 3.27: Algunos modelos perdieron sus texturas

Incluso, en el escenario de la calle, hubo que eliminar muchas partes del mismo y regenerar el escenario desde Unity con objetos primarios como cubos y planos, aplicando posteriormente las texturas adecuadas a cada uno de ellos.

Siguiendo un poco más con la parte gráfica del simulador, los escenarios se notaban lejanos, fríos, debido principalmente a la falta de sombras. Tras una breve investigación sobre cómo implementar sombras en Unity, se encontró la solución: lightmapping.

El lightmapping es una técnica por la cual el motor gráfico calcula cómo incide la luz sobre los objetos y las reflexiones de ésta y genera texturas modificadas donde las luces y las sombras aparecen proyectadas (véase Figura 3.28). De este modo se consigue tener sombras estáticas manteniendo la carga de CPU al renderizar la escena a costa de requerir un poco más de memoria para almacenar texturas más complejas.

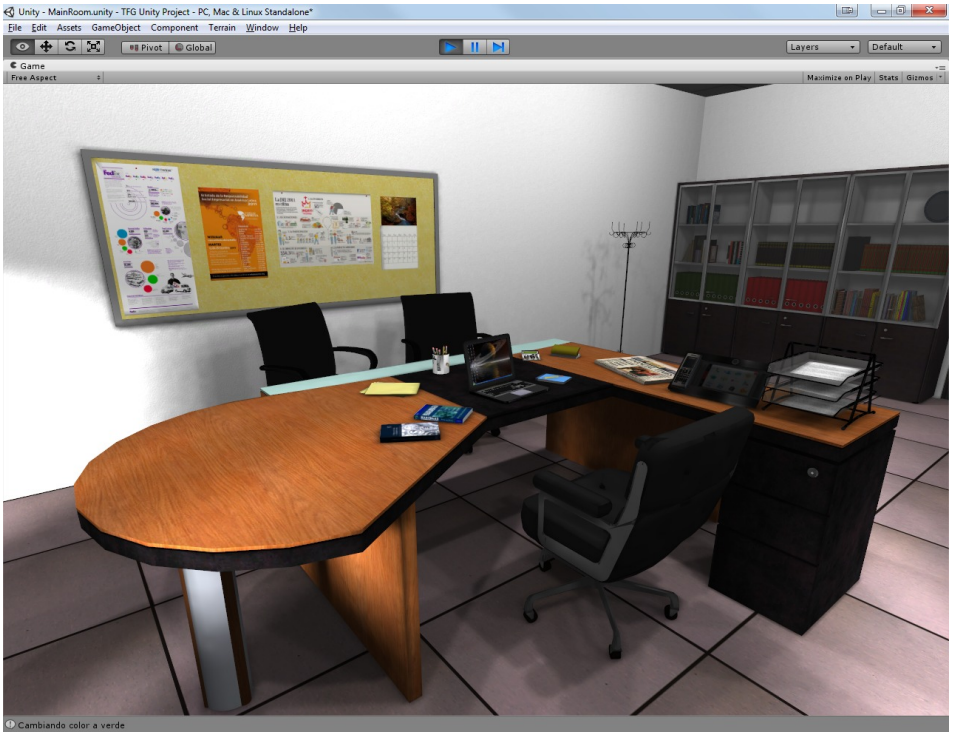


Figura 3.28: Sombras estáticas gracias al lightmapping

En Unity Pro existen las llamadas sombras dinámicas, que no son otras que las sombras proyectadas en tiempo real sobre los objetos de la escena. Aunque para la versión gratuita no estén disponibles, no supone un gran inconveniente, puesto que significa una carga adicional para el procesado de la escena y los escenarios no contienen demasiados elementos en movimiento.

3.4.2. Avatares e interactividad

Además de los escenarios físicos, lo que caracteriza a cualquier motor de videojuegos es poder implementar funcionalidad e interactividad de distintos personajes con la escena. Dado que iba a ser necesario el empleo de avatares en el simulador, comenzó la documentación acerca de éstos en Unity.

Buscando en el Asset Store, una plataforma de recursos para Unity, similar a 3D Warehouse para Sketchup, se encontró un paquete de avatares creados por la empresa Mixamo [26]. Se obtuvieron dos paquetes: *MaleCharacterPack* y *FemaleCharacterPack*. Cada uno de ellos contiene tres ficheros FBX, correspondientes a los avatares.

Una vez estos son importados en Unity 4, puede configurarse cada una de las partes móviles del avatar, como se representa en la Figura 3.29. En el Anexo C: *Máquina de estados del avatar* se muestra el diagrama de la máquina de estados del controlador del avatar.

Este diagrama de estados se desarrolla creando un *Animator Controller* para el avatar. En primer lugar se configura la animación de tipo 'humanoide'. Posteriormente, abriendo la ventana del editor *Animator* se puede crear el árbol de estados del controlador, añadiendo tantos estados, transiciones y parámetros como se deseen.

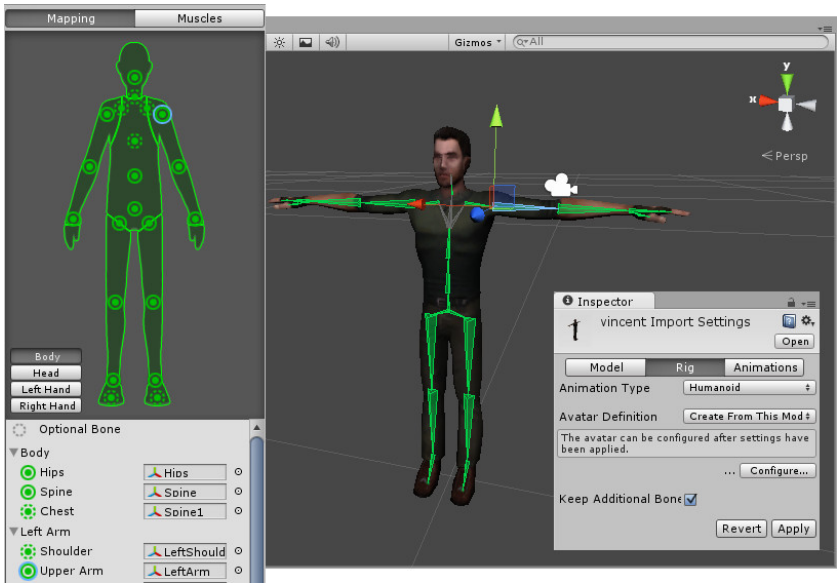


Figura 3.29: Configuración del avatar en Unity 4

Así, a cada estado se le puede asignar una animación, cuya creación es la tarea realmente difícil. Pero gracias a los paquetes gratuitos del propio equipo de Unity [31] y [32] se puede disfrutar de animaciones listas para usar.

Por otro lado, dado que se necesitaba cierta interacción con el escenario y movilidad de los objetos, se creó una nueva escena, esta vez de test, para realizar pruebas con los avatares, las animaciones y los cambios de textura de los objetos a través de interactividad con los personajes y, por ende, con el usuario.

La primera tarea fue probar la interactividad a través del cursor del ratón manejado por el usuario. Se realizaron test tanto al pasar el cursor por encima de un objeto como al clicar sobre él. Se diseñaron cursores diferentes para indicar al usuario dicha funcionalidad [21].

Aprovechando una de las bondades que ofrecen las plataformas en tres dimensiones, se hicieron test sobre los 'objetos disparadores' -trigger objects-. Éstos son objetos cuya funcionalidad es lanzar eventos cuando otro elemento choca, penetra, se encuentra dentro o sale de él. Por supuesto, son elementos que pueden hacerse no visibles para el usuario y que se emplean para desencadenar acciones cuando el jugador llega a cierto punto físico del escenario. En la Figura 3.30 puede verse el test que se realizó con estos objetos y el cambio de texturas de los semáforos a través de código UnityScript.

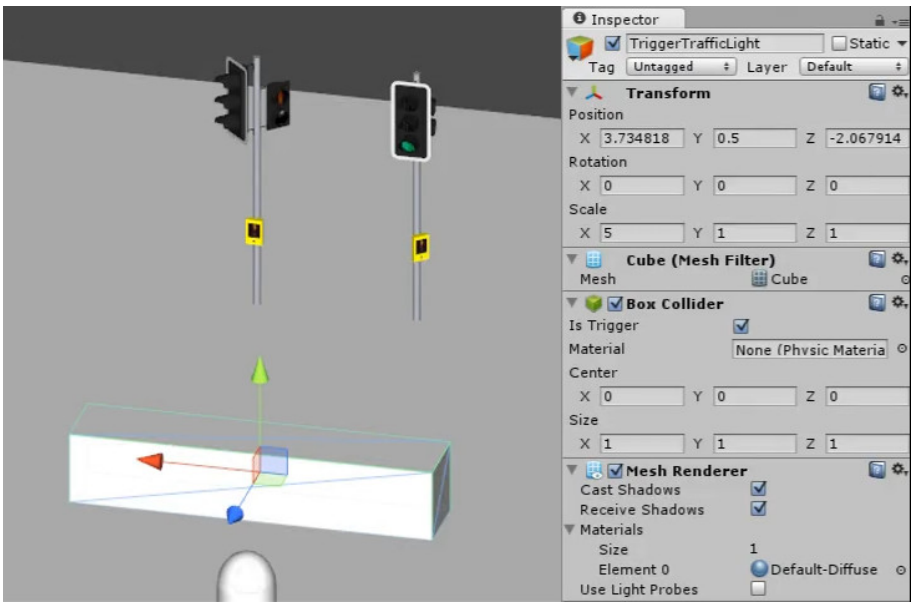


Figura 3.30: Configuración del avatar en Unity 4

Para crear estos objetos especiales basta con crear un objeto, ya sea primitivo o vacío, dependiendo de si se quiere que sea visible o no (o controlando el campo *Mesh Renderer* del objeto). Hace falta añadirle un *Collider*[3] para que el objeto sea capaz de lanzar eventos de colisión. Es importante marcar

[3]Se trata de una estructura que permite detectar colisiones. Generalmente se aplica sobre los objetos para que éstos no puedan ser atravesarlos.

la casilla *Is Trigger* para que el objeto se comporte como tal, permitiendo ser atravesado y lanzando los eventos correspondientes. La realización de las acciones asociadas debe realizarse capturando dichos eventos.

A raíz de las estructuras *Collider*, una vez más el equipo de Unity da consejos sobre su utilización y el rendimiento. Así, de mejor a peor rendimiento se enumeran: *Sphere Collider*, *Box Collider*, *Capsule Collider* y *Mesh Collider*. El último hace referencia a la adaptación perfecta de la estructura al modelo 3D del objeto, y es la opción por defecto que se encuentra activa cuando se importa un nuevo modelo en Unity.

Es muy importante desactivar esta opción (*Generate Colliders*) si no es estrictamente necesario cuando se importa un modelo. Es mejor, aunque más laborioso, aplicar posteriormente otras estructuras primitivas con mejor rendimiento, como los *Box Collider*.

3.4.3. Scripts

Aunque a lo largo de este documento se ha ido haciendo referencia a la palabra 'script', éste es el momento de definirla. Un script es un fragmento de código cuya funcionalidad es realizar diversas tareas como combinar componentes, interactuar con el sistema o con el propio usuario.

La primera tarea de esta fase de desarrollo era clara: adaptar la implementación del motor intérprete SCXML desarrollado en el proyecto SimEmergencias [23]. Tras una larga dedicación para la comprensión y alguna que otra reestructuración del código, quedó integrado en este proyecto. Esta sería la única parte que se tendría como referencia durante todo el desarrollo, por lo que comenzaba una aventura a la par que difícil, satisfactoria.

El siguiente y más importante objetivo era la externalización de todos los recursos. Esto es, extraer toda aquella funcionalidad susceptible de ser modificada por el usuario final. Esta parte fue la que más tiempo demoró y más complicada resultó, ya que en un principio no había demasiados problemas con leer ficheros de texto desde la carpeta `Resources` con la función *Resources.LoadAll()* del API de Unity. pero la actualización de estos ficheros dentro del simulador no era correcta, ya que no se reflejaban los últimos cambios como se esperaba.

Esto llegó a solucionarse parcialmente mediante la función *AssetDatabase.Refresh()*, que actualiza todos los ficheros de la estructura del proyecto Unity. Pero existía un problema: esa función sólo estaba disponible para el

Editor[4], no pudiendo ser empleada en la versión de escritorio -*Standalone version*-. La cosa no acababa ahí, pues existía otro inconveniente. La sorpresa fue que al exportar el simulador como versión de escritorio estos ficheros quedaban empaquetados en un fichero propio de Unity, siendo imposible su modificación.

Por ello, tras muchas pruebas insatisfactorias, se abandonó la idea de leerlos como recursos y se implementó la lectura de ficheros externos gracias a funcionalidades del .NET Framework 2.0 [24]. De este modo se consiguió leer y escribir ficheros externos, teniéndolos actualizados cada vez que la aplicación arrancaba. Ya podían implementarse funcionalidades asociadas a los recursos externos.

Pero existía una de las funcionalidades que no podía ser implementada tal y como se hacían las demás. Se trataba de las Interfaces Gráficas de Usuario. Para comprender el problema, primero hay que comprender cómo Unity las maneja.

En el API de Unity, existe una función específica para mostrar las GUI, denominada *OnGUI()*. Ésta puede ser implementada en cualquiera de los lenguajes que este entorno de desarrollo admite: JavaScript (UnityScript), C# y Boo. Obviamente, en consonancia con todo el código de este proyecto, se implementó en UnityScript.

En la Figura 3.31 se muestra el ejemplo del propio API donde se crea un botón de dimensiones 150 píxeles de ancho por 100 píxeles de alto, cuya esquina superior izquierda coincide con el décimo píxel tanto horizontal como vertical y presenta el texto *I am a button*. Al clicar sobre él, se imprimirá el texto *You clicked the button!* por consola.

```
function OnGUI()
{
    if(GUI.Button(Rect(10,10,150,100), "I am a button"))
        print("You clicked the button!");
}
```

Figura 3.31: Función *OnGUI()*

¿Y en qué difiere esta funcionalidad del resto? Mientras el resto de los ficheros externos guardan información referente al contenido y de una manera u otra puede procesarse a conveniencia para asignarse a variables internas, en este caso, el botón o cualquier otro elemento gráfico se representa a partir de código de programación.

[4]Versión referida a la ejecución del juego dentro del entorno de desarrollo Unity.

Tras una larga y tediosa búsqueda de documentación al respecto se llegó a la conclusión de que no era posible compilar código de scripts externos. Esto se debe a que Unity los precompila en el momento de exportar la aplicación a cualquier plataforma, quedando empaquetados y no pudiendo modificarse. Que sea precompilado implica que Unity no admite compilación en el momento de la ejecución -runtime compilation-.

Para solucionarlos se hizo uso de la función *eval()*, que permite ejecutar código JavaScript en tiempo de ejecución. Tras varias pruebas se logró representar interfaces gráficas de este modo por lo que se creó una especie de traductor de pares clave-valor de estructuras JSON a UnityScript (véase Figura 3.10), lo que supuso un gran avance.

Aún quedaba por resolver otro pequeño inconveniente, y es que solamente se representaba la última de las interfaces, por lo que hubo que agudizar el ingenio y crear una nueva *Coroutine*[5] que fuese añadiendo o eliminando código a la variable que existía dentro de la función *OnGUI*. En la Figura 3.32 se muestra un ejemplo más detallado del código.

```
function OnGUI () {
    StartCoroutine(OnGUIAux());
}

function OnGUIAux(){
    if(code != null && code.Trim() != String.Empty){
        eval(code);
    }
}
```

Figura 3.32: Porción de código de *MngGUI.js*

Donde la variable *code* contiene, si no está vacía, código estructurado de la forma de la Figura 3.33.

De este modo, cuando una interfaz quiere dejar de mostrarse, basta con buscar dentro del contendido dónde empieza la línea que coincide con su nombre y dónde finaliza la siguiente coincidente, eliminando el contenido albergado entre ambos puntos. De forma análoga, para mostrar una interfaz se añade al final del contenido la línea con el nombre de la misma, el contenido en sí y de nuevo la línea con el nombre.

Así, se definieron varios elementos gráficos para la interacción con el usuario, los cuales van desde simple representación como cajas y texto a botones,

[5]Similar a los hilos -threads- de otros lenguajes de programación, se trata de una manera de realizar tareas simultáneas sin bloquear la ejecución principal del programa.

```
// nombre_del_fichero_al_que_pertenece_la_GUI_1
    CÓDIGO CON LOS ELEMENTOS GRÁFICOS A MOSTRAR DE LA GUI 1
// nombre_del_fichero_al_que_pertenece_la_GUI_1

// nombre_del_fichero_al_que_pertenece_la_GUI_2
    CÓDIGO CON LOS ELEMENTOS GRÁFICOS A MOSTRAR DE LA GUI 2
// nombre_del_fichero_al_que_pertenece_la_GUI_2
                              ...
// nombre_del_fichero_al_que_pertenece_la_GUI_N
    CÓDIGO CON LOS ELEMENTOS GRÁFICOS A MOSTRAR DE LA GUI N
// nombre_del_fichero_al_que_pertenece_la_GUI_N
```

Figura 3.33: Estructura del contenido de la variable *code*

áreas y campos de texto. Éstos últimos tienen la propiedad de poder almacenar lo que el usuario introduzca en ellos para usarlos posteriormente. En la Figura 3.34 se muestran identificados cada uno de estos elementos.

Figura 3.34: Elementos gráficos definidos en el simulador

En cuanto al resto de contenido externo, se generaron scripts internos para interpretarlos. Para ello se crearon funciones complejas para leer los datos provenientes de los ficheros .json externos y almacenarlos en estructuras de datos como diccionarios y listas, que a su vez podían contener a otras. Para conseguir analizar las estructuras JSON se hizo uso del parser [34].

3.4.4. Idiomas

Al igual que con todo el contenido externo, existen scripts internos que manejan el contenido de idiomas y lo interpretan. Así, se definió el fichero *MainMenu.js* para realizar las tareas de presentación del menú principal de la aplicación. Todo su continente y acciones es interno y no admite modificación externa ya que se consideró que debía de ser robusto y, al fin y al cabo, no tiene sentido modificar las acciones de los botones de este menú.

En cualquier caso, el contenido sí es modificable para poder cambiar el idioma. En lugar de presentar los textos de los botones como algo adherido a éstos, se crearon funciones capaces de modificar el texto que contienen. En la Figura 3.35 se muestra el menú de la aplicación en diferentes idiomas, mientras que la parte de código que lo hace posible se presenta en la Figura 3.37.

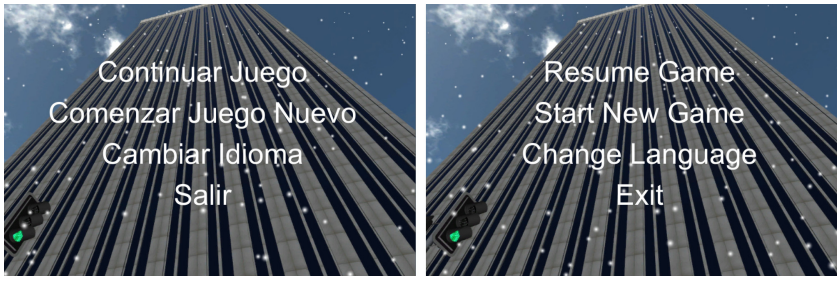


Figura 3.35: Menú de la aplicación en español (izda.) e inglés (dcha.)

Dado que la aplicación está preparada para cambiar el idioma de los contenidos, implica que alguno de los elementos que se muestran en los escenarios deban ser modificados debidamente, al igual que ocurre con el menú.

Figura 3.36: Carteles en español (izda.) e inglés (dcha.)

Siguiendo la misma metodología se consiguió personalizar, por ejemplo, los carteles de la planta de oficinas que representan el departamento al que pertenece cada estación de trabajo (véase Figura 3.36). De este modo, puede modificarse desde el exterior, a través del fichero *mainfloor_signboards.json*, cuyo estructura se especifica en la Figura 3.38.

```
static function LoadMenuTranslation(){
    var jsonObj = FileUtils.LoadJsonFile(
        GeneralScript.EXT_RES_MENU_DIR + "menu_translations.json");
    JsonUtils.ProcessJson(jsonObj, JsonUtils.MENU_STORE, false);
    GeneralScript.SetAvailableLangs();
}

static function ChangeMainMenuLang(){
    var dict = JsonUtils.GetStore(JsonUtils.MENU_STORE);
    var auxLang = GeneralScript.GetLanguage();

    var val : String;
    for(key in dict.Keys){
        if(key.Substring(key.IndexOf("_")+1,
                key.length-key.IndexOf("_")-1) == auxLang){
            dict.TryGetValue(key, val);
            key = key.Substring(0,key.IndexOf("_"));

            var go = GameObject.Find(key);
            if(go != null && go.activeSelf){
                var textMesh : TextMesh = go.GetComponent("TextMesh")
                     as TextMesh;
                textMesh.text = val;
            }//if null
        }//if lang
    }//for
}
```

Figura 3.37: Estructura del contenido de la variable *code*

```
{
    "MainFloor_Signboards":
    {
        "Signboard04_en":"HR",
        "Signboard04_es":"RRHH",

        "Signboard05_en":"Sales",
        "Signboard05_es":"Ventas",
    }
}
```

Figura 3.38: Definición de los textos de los carteles de la Figura 3.36

Por último, otro de los elementos que es sensible al cambio de idiomas es el texto mostrado al pasar el ratón por encima de los objetos interactivos encargados de lanzar las aplicaciones externas, como muestra la Figura 3.39.

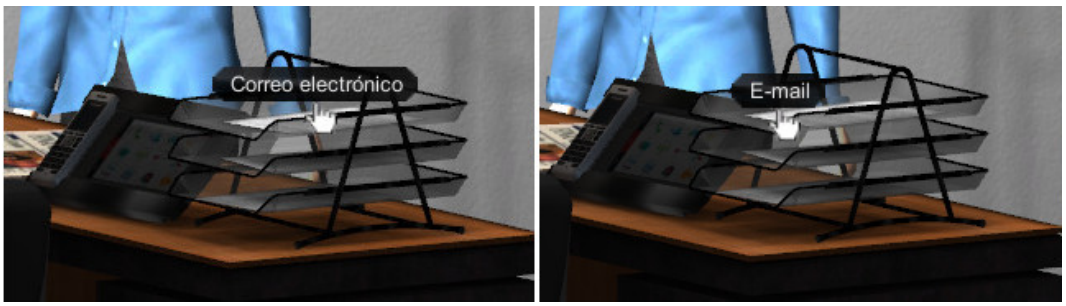


Figura 3.39: Tooltip en español (izda.) e inglés (dcha.)

3.4.5. Shader

Los *Shader* son pequeños scripts que permiten configurar cómo el hardware gráfico renderizará los objetos. El Shader que se aplica por defecto a los objetos en Unity solamente tienen una cara para reducir carga de procesamiento. Pero hay veces que es necesario que los objetos presenten dos caras, como es el caso de la bandera que puede verse en la Figura 3.40.

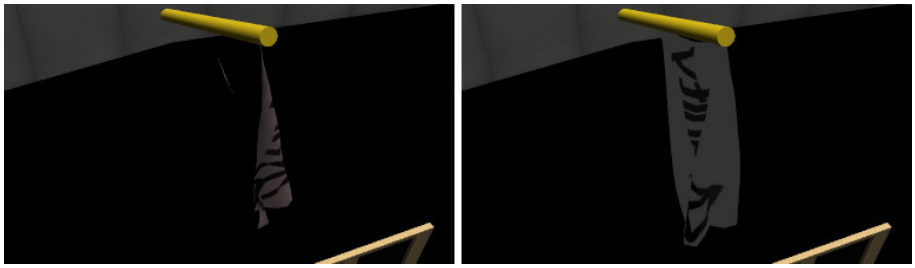


Figura 3.40: *Shader* por defecto (izda.) y *Shader* personalizado (dcha.)

Para corregir este inconveniente, gracias a la información de [30] se creó un *Shader* (Figura 3.41) que se aplicó al tipo de material de la bandera.

Además, hubo que crear otro *Shader* para los carteles que se muestran en la Figura 3.36, ya que por defecto el material de los textos 3D de Unity hace que se vean a través de los objetos que tenga delante, es decir, se comporta como si los objetos que debían taparlo fueran transparentes.

```
Shader "Custom/TwoSides" {
    Properties {
        _MainTex ("Base (RGB)", 2D) = "white" {}
    }
    SubShader {
        Tags { "RenderType"="Opaque" }
        LOD 200
        Cull Off
        CGPROGRAM
        #pragma surface surf Lambert
        sampler2D _MainTex;
        struct Input { float2 uv_MainTex; };
        void surf (Input IN, inout SurfaceOutput o) {
            half4 c = tex2D (_MainTex, IN.uv_MainTex);
            o.Albedo = c.rgb;
            o.Alpha = c.a;
        }
        ENDCG
    }
    Fallback "Diffuse"
}
```

Figura 3.41: *Shader* personalizado para mostrar las dos caras del objeto

3.4.6. Seguridad

Debido a la exposición de los recursos externos a cualquier usuario, para evitar código malicioso se ha creado un filtro que solamente permite realizar llamadas al API público de este simulador (véase Apartado 3.4.7) o a la clase de Interfaces Gráficas de Unity, obviando cualquier otro tipo de código.

3.4.7. API

Puesto que el presente software dispone de miles de líneas de código, resulta imprescindible que éste goce de una estructura limpia y ordenada, y con abundantes comentarios para el resto de desarrolladores. Con el fin de crear un API en formato web, se ha empleado la herramienta de generación de documentación YUIDoc [7].

Ésta, a partir de una breve configuración y siguiendo el etiquetado correcto en los comentarios, se genera a través de un script toda una serie de páginas web con el API de los ficheros que se deseen.

Se han generado dos tipos de API, uno público compuesto por un sólo módulo y una sola clase para el público en general, y uno privado para los desarrolladores formado por trece módulos y treinta y nueve clases. Junto a este documento se facilita el de ámbito público. En la Figura 3.42 puede verse una imagen del API para desarrolladores.

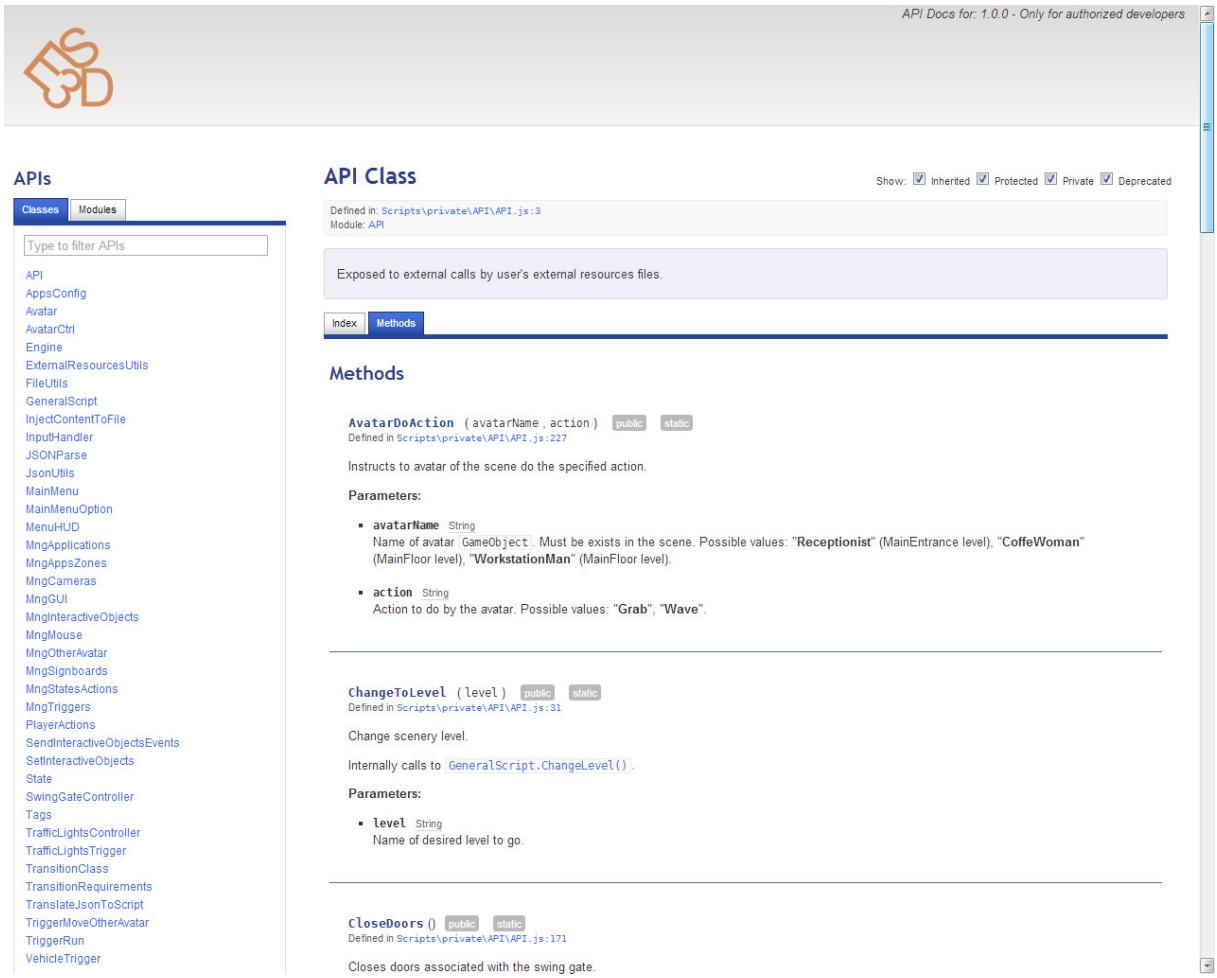


Figura 3.42: Captura de pantalla del API para desarrolladores

3.4.8. Efectos sonoros

Todos los sonidos ambientales que se perciben en los distintos escenarios han sido obtenidos de [10], [15], [39] y [29] y establecidos en Unity como *AudioSource*.

3.4.9. Aplicaciones externas

Como ya se comentó en la Tabla 2.1, una de las funcionalidades añadidas a este simulador es permitir el acceso desde el mismo a aplicaciones externas. Para ello se ha creado un despacho físico que actúa como un portal que mediante un click en ciertos elementos da acceso al usuario a lanzar aplicaciones virtuales de escritorio.

En la Tabla 3.1 puede consultarse el listado completo de aplicaciones disponibles para cada Sistema Operativo.

App/Recurso	Plataforma		
	Windows	Linux	MacOSX[6]
Ofimática			
Calculadora	calc	gnome-calculator	Calculator.app
Calendario	http://www.timeanddate.com/calendar/		iCal.app
Notas	notepad	gedit	TextEdit.app
MO Word	winword	-	Microsoft Word.app/Contents/MacOS/Microsoft Word[7]
MO Excel	excel	-	Microsoft Excel.app/Contents/MacOS/Microsoft Excel[7]
MO PowerPoint	powerpnt	-	Microsoft PowerPoint.app/Contents/MacOS/Microsoft PowerPoint[7]
OO Writer	swriter	soffice -writer	OpenOffice.org.app/Contents/MacOS/swriter
OO Calc	scalc	soffice -calc	OpenOffice.org.app/Contents/MacOS/scalc
OO Impress	simpress	soffice -impress	OpenOffice.org.app/Contents/MacOS/simpress
LO Writer	swriter	soffice -writer	LibreOffice.app/Contents/MacOS/swriter
LO Calc	scalc	soffice -calc	LibreOffice.app/Contents/MacOS/scalc
LO Impress	simpress	soffice -impress	LibreOffice.app/Contents/MacOS/simpress
Correo electrónico			
MO Outlook	outlook	-	Microsoft Outlook.app/Contents/MacOS/Microsoft Outlook[7]
Gmail	https://mail.google.com/		
Thunderbird	thunderbird	thunderbird	Thunderbird.app
Multimedia			
Reproductor	wmplayer	totem	iTunes.app
Navegadores			
Internet Explorer	iexplore	-	-
Google Chrome	chrome	google-chrome	Google Chrome.app/Contents/MacOS/Google Chrome
Mozilla Firefox	firefox	firefox	Firefox.app/Contents/MacOS/firefox
Safari	safari	-	Safari.app/Contents/MacOS/Safari[8]
Opera	opera	opera	Opera.app/Contents/MacOS/Opera
Comunicación			
Skype	Skype.exe[9]	skype	Skype.app
Gtalk	https://mail.google.com/		
Redes Sociales			
Twitter	https://twitter.com/		
Facebook	https://www.facebook.com/		
Tuenti	https://www.tuenti.com/		
LinkedIn	http://www.linkedin.com/		
Noticias			
Google News	https://news.google.com/		
Almacenamiento			
Dropbox	https://www.dropbox.com/home		
Google Drive	https://drive.google.com/		
Motores de búsqueda			
Google	https://www.google.com/		
Yahoo	http://www.yahoo.com/		
Bing	http://www.bing.com/		
Altavista	http://www.altavista.com/		

Tabla 3.1: Aplicaciones externas capaces de ser lanzadas por el simulador.

[6]Por motivos de espacio se ha omitido el prefijo `/Applications/`, necesario para todos los comandos de este Sistema Operativo que no sean URLs.

[7]Estos comandos tienen como prefijo `/Applications/Microsoft Office 2011/`.

[8]No abre la página solicitada sino la que tenga configurada por defecto.

[9]El comando completo es `C:\Program Files (x86)\Skype\Phone\Skype.exe`, siendo la ruta por defecto del programa para computadoras con Sistema Operativo de 64 bits. Para Sistemas Operativos de 32 bits el programa debe ser instalado en esta ruta, ya que la directorio por defecto es diferente y no podría lanzarse desde el simulador.

Esto se consiguió realizar gracias al comando del Framework .NET 2.0 *System.Diagnostics.Process.Start()* y a un complejo tratamiento de los comandos que deben ser lanzados dependiendo del Sistema Operativo donde esté corriendo el simulador. Asimismo, es capaz de detectar qué programas se encuentran instalados dentro de una lista predefinida de aplicaciones. Esta versión solamente es compatible con sistemas Windows.

Como se puede ver en la Figura 3.43, los programas disponibles así como sus iconos dependen del Sistema Operativo.

Figura 3.43: Aplicaciones en Windows (izda.) y Mac OS X (dcha.)

Cabe destacar que las aplicaciones homólogas de OpenOffice.org y LibreOffice solamente son distinguidas en Sistemas Operativos MacOSX. Esto es debido a que, tanto en Windows como en Linux, estas aplicaciones análogas son lanzadas mediante el mismo comando. Por tanto, si en un equipo coexisten los paquetes OpenOffice.org y LibreOffice, será ejecutada la aplicación del paquete office que fuese instalado con posterioridad, ya que habrá sobrescrito el valor en el sistema.

También es importante mencionar que en los equipos con Windows las aplicaciones que se reconocen son mostradas con su logo a todo color y las que no con su logo casi transparente, imposibilitando la selección de ésta. Sin embargo, tanto en equipos Linux como MacOSX todos son mostrados a todo color y, por tanto, el usuario no puede saber a priori qué aplicaciones

han sido reconocidas y las que no. Se trata de un hecho conocido y que será resuelto en futuras versiones de la aplicación.

3.4.10. Información personalizada

El simulador ha sido diseñado para que la información introducida por el usuario en tiempo de ejecución pueda utilizarse posteriormente en diferentes elementos de la interfaz. Esto permite dar feedback personalizado como puede verse en la Figura 3.44 o utilizar documentos plantilla para generar resultados personalizados del proceso de formación, como por ejemplo el plan de negocio del emprendedor.

Figura 3.44: Información personalizada en formato de página web

En la Figura 3.10 se muestra cómo algunos de los elementos gráficos que serán mostrados llevan asociadas unas variables para almacenar datos. Esas variables pueden guardarse en las preferencias de la aplicación mientras la interfaz se está mostrando invocando a la función del API público *API.SavePreference("companyMainActivity", textFieldVar);* como acción dentro de una GUI que contenga alguno un campo de texto (*textFieldVar* asociada o un área de texto con *textAreaVar*).

Cuando se quieran exportar esos datos a algún documento plantilla basta con llamar a *API.InjectPlayerPrefsVarsToFile("gameCollectedInfo.html");* pasándole como parámetro el nombre del documento, que debe encontrarse dentro de `ExtResources/filesToFill`. Además, se ha implementado una lógica por la cual puede distinguirse entre idiomas para escribir en distintos documentos.

Así, en el comando anterior se invoca a *gameCollectecInfo.html*, pero si se inspecciona la estructura de ficheros de la Figura 3.2 se verá que este fichero no existe. En su lugar, existen *gameCollectecInfo_ en.html* y *gameCollectecInfo_ es.html*. Cuando la aplicación no encuentra el fichero solicitado, intenta buscar otro con el mismo nombre y sufijo '_xx', donde xx denota el código del idioma actual de la aplicación.

El contenido que será sustituido en los documentos plantilla debe ir encerrado entre caracteres $$ y $$. Por ejemplo: Congratulations $$userName$$! .

3.5. Evaluación de la aplicación

Llegando a las fases finales de desarrollo, la aplicación fue probada en diferentes equipos con diferentes configuraciones y Sistemas Operativos. A continuación, en la Tabla 3.2 se muestra un resumen de compatibilidad con los test realizados.

Sistema Operativo	Compatible	Tarjeta Gráfica	Compatible
Windows 7 (x86 | x64) (32|64-bit)	✓	Intel	✓
Windows XP (32-bit)	✓	Intel HD	✓
Linux (x86 | x64) (32|64-bit)	✓	NVIDIA	✓
Mac OS X (x86) (32-bit)	✓	ATI	✓

Tabla 3.2: Pruebas del software

Obviamente, dependiendo de la potencia de la CPU, de la tarjeta gráfica y, en general, de las especificaciones de cada equipo, la aplicación era ejecutada con menor o mayor fluidez, pero siendo siempre jugable gracias a las diferentes configuraciones gráficas que permite la aplicación.

Durante esta fase se fueron encontrando *bugs* que se fueron corrigiendo. Uno de ellos tuvo que ver con el posicionamiento del jugador en el escenario al reanudar una partida. Mientras en unos Sistemas Operativos se empleaba el carácter ',' para separar los decimales de la posición en los ejes del espacio, en otros se empleaba el carácter '.'. Otros, en cambio, se debían a los caracteres especiales con las fuentes que no permitían codificación UTF-8. Y así un largo etcétera de pequeños fallos cuya solución era sencilla.

Capítulo 4

Presupuesto y planificación

4.1. Planificación

Si bien desde el principio del proyecto han existido unos objetivos y unas fases bien definidas, no ha existido una planificación temporal estricta debido a la coincidencia del desarrollo de este proyecto con una beca de Prácticas en Empresa a lo largo de todo el periodo y con clases y prácticas de una asignatura durante la primera mitad del mismo, conllevando dedicaciones semanales y diarias muy dispares.

Por ello, la duración, en días, de las tareas presentadas en las Figuras 4.1 y 4.2 no debe entenderse como una representación del esfuerzo de cada una de ellas, sino como mera información del comienzo, fin y la simultaneidad entre ellas. La dedicación a cada grupo de tareas puede verse en detalle en la Tabla 4.5. Cabe aclarar, que entre las distintas reuniones con la tutora, tanto presenciales como online, se ha mantenido contacto vía correo electrónico para el seguimiento del proyecto.

Además, hay que apuntar que existió un parón del proyecto desde el 13 de noviembre de 2012 hasta el 18 de enero de 2013, ambos inclusive, debido a la dedicación plena a actividades académicas. Esta pausa se muestra como una franja roja en el Diagrama de Gantt de la Figura 4.2. También se muestra el camino crítico[1] mediante las tareas que figuran rellenas mediante líneas oblicuas negras.

Otro diagrama, el de Pert, se adjunta en el Anexo A por su extensión. En dicho diagrama las tareas críticas son indicadas mediante recuadros amarillos.

[1] Ruta que enlaza las tareas críticas, que a su vez llevan a la duración mínima del proyecto.

Nombre	Fecha de inicio	Fecha de fin	Duración
Trabajo Fin de Grado	9/07/12	20/06/13	345
Análisis del problema	9/07/12	14/07/12	6
Estudio del estado del arte	9/07/12	12/07/12	4
Análisis de los requisitos del proyecto	12/07/12	14/07/12	3
Pruebas motor videojuegos	15/07/12	24/07/12	10
Comprensión de la interfaz de usuario	15/07/12	16/07/12	2
Pruebas de objetos, cámaras y luces	16/07/12	20/07/12	5
Pruebas con UnitScript	20/07/12	24/07/12	5
Modelado 3D	2/08/12	24/05/13	294
Modelado del despacho	2/08/12	13/09/12	43
Modelado del edificio principal	7/08/12	27/08/12	21
Modelado de la calle	25/02/13	27/02/13	3
Modelado de la entrada del 'hall'	6/04/13	10/04/13	5
Otros	4/08/12	24/05/13	292
Importación modelos 3D	16/07/12	28/05/13	315
Pruebas de texturas	16/07/12	22/07/12	7
Importación modelos finales	16/07/12	28/05/13	315
Definición de la arquitectura	7/11/12	12/11/12	6
Análisis de los módulos necesarios	7/11/12	8/11/12	2
Definición de cada módulo	9/11/12	12/11/12	4
Desarrollo de software y externalización de los recursos	13/10/12	21/05/13	219
Funcionalidades del simulador	14/10/12	21/05/13	218
Interacciones básicas con el simulador	13/10/12	13/10/12	1
Externalización básica	9/11/12	9/11/12	1
Interacciones avanzadas con el simulador	26/02/13	18/05/13	82
Externalización avanzada	19/01/13	17/03/13	58
Evaluación de la aplicación	18/10/12	9/06/13	233
Evaluación en Windows	18/10/12	9/06/13	233
Evaluación en Linux	6/05/13	6/06/13	32
Evaluación en Mac OS X	20/05/13	31/05/13	12
Documentación	31/10/12	20/06/13	231
Redacción de la memoria	31/10/12	18/06/13	229
Generación de la guía de usuario	19/06/13	20/06/13	2
Reuniones	22/10/12	7/06/13	227
Reunión online 1	22/10/12	22/10/12	1
Reunión presencial 1	23/10/12	23/10/12	1
Reunión online 2	12/02/13	12/02/13	1
Reunión presencial 2	20/03/13	20/03/13	1
Reunión presencial 3	10/04/13	10/04/13	1
Reunión presencial 4	19/05/13	19/05/13	1
Reunión presencial 5	7/06/13	7/06/13	1

Figura 4.1: Tareas definidas en el poyecto

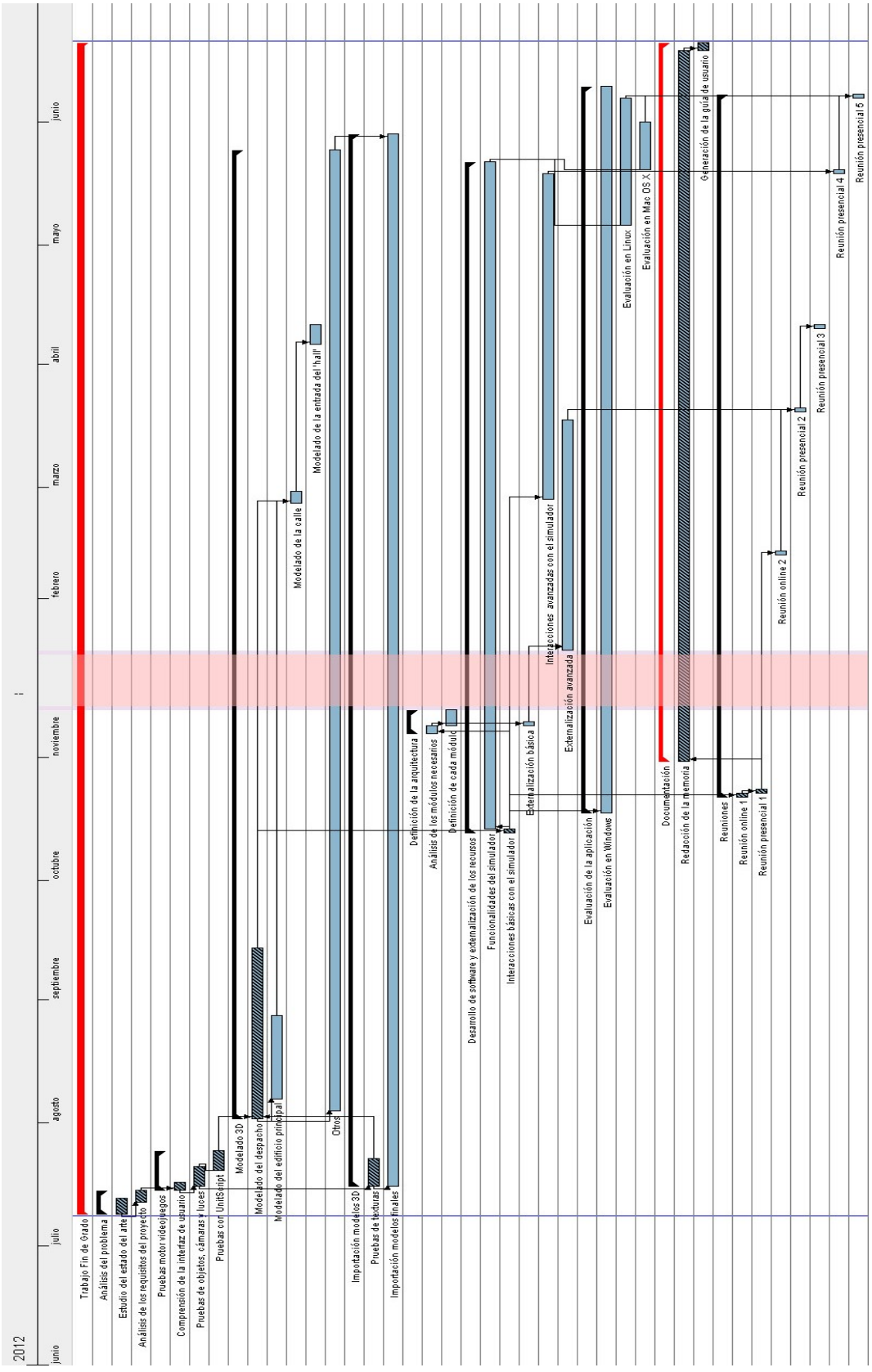


Figura 4.2: Diagrama de Gantt del proyecto

4.2. Presupuesto

El coste de recursos humanos se ha calculado a partir de una retribución bruta anual de 34.500 € distribuida en 14 pagas. El mensual devengado se ha calculado como el bruto anual dividido entre el número de pagas. A su vez, la base de cotización se obtiene sumando al mensual devengado la cantidad correspondiente al prorrateo de las pagas extra, tal y como se define en la Ecuación 4.1.

$$Base\ cotización = Devengado\ mensual + \frac{2 \times Devengado\ mensual}{12} \tag{4.1}$$

Se tiene en cuenta que el trabajador está en plantilla, por lo que las retenciones de la Seguridad Social por desempleo se sitúan en el 1,55 % frente a los 1,60 % para los empleados temporales. La empresa deberá abonar por este hecho el 5,50 % al mismo concepto en lugar del 6,70 % que tendría que hacer frente en caso de tratarse de un empleado temporal.

Asimismo, no existen horas extra ni otros conceptos que atribuir a las contingencias profesionales de la Seguridad Social y todos los conceptos salariales cotizan. Es decir, no existen conceptos tales como dietas, plus de empresa ni ningún otro, por lo que el se considera que el mensual devengado coincide con el salario base.

Bruto anual	34.500,00 €	Neto anual	26.444,25 €
Mensual devengado	2.464,29 €	Retenciones mensuales	671,31 €
Neto mensual	1.792,97 €	Retenciones pagas extra	418,93 €
Neto pagas extra (2)	2.045,36 €	Pagas	14
Base cotización	2.875,00 €	Laborable anual	1800 h

Tabla 4.1: Detalles sobre la nómina general del empleado.

Tal y como se mostraba en la Tabla 4.1, al empleado le son retenidos 671,31 € en concepto de contribución a la Seguridad Social y pago del IRPF. Es importante aclarar que al salario mensual le son retenidos ambos conceptos, pero a las pagas extraordinarias solamente se les aplica el IRPF, puesto que las contribuciones sociales le son restadas mes a mes en el prorrateo mensual. En la Tabla 4.2 se muestran con detalle los conceptos aplicables a ambos gravámenes para el trabajador y de Seguridad Social para la empresa.

Contribuciones mensuales				
Coste para la empresa			Retenciones al empleado	
678,50 €	23,60 %	Contingencias comunes	4,70 %	135,13 €
-	-	Contingencias profesionales	-	-
158,13 €	5,50 %	Desempleo	1,55 %	44,56 €
5,75 €	0,20 %	FOGASA	-	-
17,25 €	0,60 %	Formación profesional	0,10 %	2,88 €
859,63 €	29,90 %	Total Seguridad Social (sobre base cotización)	6,35 %	182,56 €
-	-	IRPF (sobre devengo mensual)	19,40 %	418,93 €
-	-	Total retenciones	-	671,31 €

Tabla 4.2: Contribuciones a la Seguridad Social e IRPF.

A continuación, en las Tablas 4.3 y 4.4, se muestran en detalle la retribución y los costes tanto anuales como mensuales y por hora del trabajador y la empresa. Para obtener las tarifas y costes por horas se han tomado como referencia 1800 h laborables al año.

Concepto	Anual	Mensual [2]	Por hora [3]
Salario bruto	34.500,00 €	2.875,00 €	19,17 €
Retenciones S.S.	2.190,75 €	182,56 €	1,22 €
Retenciones IRPF mensuales	5.027,14 €	418,93 €	2,79 €
Retenciones IRPF pagas extra	837,86 €	69,82 €	0,47 €
Retenciones totales	8.055,75 €	671,31 €	4,48 €
Salario neto	26.444,25 €	2.203,69 €	14,69 €

Tabla 4.3: Salarios y contribuciones detalladas del empleado.

Concepto	Anual	Mensual [2]	Coste por hora [3]
Salario bruto del empleado	34.500,00 €	2.875,00 €	19,17 €
Costes S.S.	10.315,50 €	859,63 €	5,73 €
Salario + Costes S.S.	44.815,50 €	3.734,63 €	24,90 €

Tabla 4.4: Costes detallados de la empresa.

[2]Los valores de esta columna son valores anuales prorrateados durante 12 meses.
[3]Los valores de esta columna son valores anuales prorrateados durante 1800 h.

Una vez obtenido el coste del trabajador a la empresa por hora, se procede a calcular la dedicación y el coste de cada de las fases en la Tabla 4.5.

Etapa	Descripción	Tiempo	Importe
Análisis problema	Análisis de estado del arte y alcance del proyecto	20,1 h	500,44 €
Pruebas Unity	Fase de pruebas con el motor de videojuegos	29,5 h	734,48 €
Modelado 3D	Creación de modelos 3D para el simulador	72,9 h	1.815,03 €
Imp. modelos 3D	Resolución de problemas con los modelos 3D	24,6 h	612,48 €
Arquitectura	Definición y diseño de la arquitectura	11,2 h	278,85 €
Desarrollo	Funcionalidad del simulador y externalización	399,4 h	9.944,06 €
Evaluación	Pruebas de la aplicación	18,3 h	455,62 €
Reuniones	Reuniones de seguimiento	10,2×2 h	253,95×2 €
Documentación	Redacción de la memoria y guía del usuario	151,5	3.771,97 €
Total		737,7	18.620,84 €

Tabla 4.5: Tiempo consumido en cada una de las etapas.

Recurso	Coste	Dedicación	Coste imputable
Sketchup	Gratuito	9 meses	0 €
Unity	Gratuito	9 meses	0 €
FBX Converter	Gratuito	9 meses	0 €
GIMP	Gratuito	9 meses	0 €
Inkscape	Gratuito	9 meses	0 €

Tabla 4.6: Costes directos de software.

Componente	Coste	Vida útil	Dedicación	Imputable
PC				
Intel i3 550 2x3.2 GHz	160,00 €	5 años	9 meses	30,00 €
ASRock H55 Pro	112,00 €	5 años	9 meses	21,00 €
Gigabyte NVIDIA® GeForce GTS250	134,00 €	5 años	9 meses	25,13 €
Kingston 12 GB @ 1333 PC3 10600 CL9	172,00 €	10 años	9 meses	21,50 €
Seagate Barracuda 500 GB 7200.12	56,40 €	5 años	9 meses	10,58 €
Fuente alimentación Unyka 51842 600W	32,00 €	5 años	9 meses	6,00 €
Unidad óptica LG GH22N550	24,00 €	6 años	N/A	- €
Lector tarjetas + DNIe CoolBox CR600	14,90 €	8 años	N/A	- €
Caja Xigmatek Asgard	40,90 €	10 años	9 meses	3,83 €
Periféricos entrada				
Teclado + Ratón inalámbricos NGS	50,00 €	4 años	9 meses	9,38 €
Periféricos salida				
Doble monitor LG FLATRON L1715S	2x180,00 €	6 años	9 meses	45,00 €
Sistema Operativo				
Windows 7 Professional 64 bits SP1	125 €	8 años	9 meses	11,72 €
Total	1.281,20 €			184,13 €

Tabla 4.7: Costes de hardware y software asociado.

Ha de aclararse que para calcular el coste asociado a las reuniones se ha supuesto un salario equivalente para la tutora de este proyecto, quedando así detallado en la Tabla 4.5.

Concepto	**Importe**
Recursos Humanos	18.620,84 €
Recursos Materiales y Software	184,13 €
Costes Indirectos (10 %)	1.880,50 €
Base imponible	20.685,47 €
I.V.A. (21 %)	4.343,95 €
Importe total	**25.029,42 €**

Tabla 4.8: Costes totales del proyecto.

El coste de los equipos informáticos empleados para las evaluaciones intermedia y final de la aplicación imputables a este proyecto quedan contenidos dentro de los costes indirectos presupuestados.

Capítulo 5

Conclusiones y trabajos futuros

5.1. Conclusiones

En este Trabajo Fin de Grado se ha realizado el proyecto completo del prototipo de un simulador para emprendedores. Sus característica diferenciadora es el inmenso control que se tiene de la aplicación desde el exterior gracias a que todos sus recursos están externalizados.

Para ello se ha empleado el estándar SCXML para el control de flujo de la historia y estructuras JSON para el resto de recursos externos. La implementación ha sido costosa, pero supone un abaratamiento de costes a medio y largo plazo debido a la posible reutilización del software sin necesidad de volver a la fase técnica de desarrollo.

Además, permite añadir y actualizar contenidos sin necesidad de poseer conocimientos técnicos acerca del desarrollo de videojuegos, de tal forma que un especialista en la materia del simulador pueda mejorar los contenidos y adaptarlos en cualquier momento.

En cuanto a los requisitos que inicialmente se planteaban en la Sección 2.3, en las siguientes tablas se muestran las actuaciones llevadas a cabo para su consecución.

Solución: SOL-00	
Actuación	El flujo de cada escenario está definido en un fichero SCXML que se encuentra bajo la raíz del directorio del escenario en cuestión.
Observaciones	Véase Figura 3.3.

Tabla 5.1: Solución SOL-00.

Solución: SOL-01	
Actuación	Cada una de las Interfaces de Usuario que son mostradas quedan definidas dentro de un fichero. Puede contener tantos elementos gráficos como se desee.
Observaciones	Véase Figura 3.7.

Tabla 5.2: Solución SOL-01.

Solución: SOL-02	
Actuación	Dentro de cada fichero de la Tabla 5.2, cada elemento dispone de propiedades de tamaño y posición, además del contenido a mostrar.
Observaciones	Véase Figura 3.7.

Tabla 5.3: Solución SOL-02.

Solución: SOL-03	
Actuación	La partida queda guardada, tanto el progreso conseguido como el escenario actual y la posición del jugador dentro del mismo. En el menú se da la opción al usuario de continuar con la partida existente o comenzar una nueva.
Observaciones	Debe mostrarse un aviso si los datos van a ser eliminados.

Tabla 5.4: Solución SOL-03.

Solución: SOL-04	
Actuación	Todos los contenidos están externalizados, incluso los de aquellos textos que pertenecen a partes no modificables en cuanto a interactividad se refiere, por lo que pueden incluirse tantos idiomas como se consideren oportunos.
Observaciones	Véase Figura 3.9.

Tabla 5.5: Solución SOL-04.

Solución: SOL-05	
Actuación	Se ha implementado un sistema de definición y comprobación de objetivos.
Observaciones	Véase Figuras 3.5 y 3.6.

Tabla 5.6: Solución SOL-05.

Solución: SOL-06	
Actuación	Se ha implementado un sistema que almacena datos que son introducidos por el usuario y que genera documentos de cualquier tipo, mientras estén basados en texto, con la información que fue almacenada.
Observaciones	Véase Apartado 3.4.10.

Tabla 5.7: Solución SOL-06.

Solución: SOL-07	
Actuación	El simulador presenta versiones compatibles con Windows, Linux y Mac OS X.
Observaciones	Desde el primer momento se tuvo en cuenta este aspecto, programando siempre que fuese posible abstrayéndose de la plataforma objetivo.

Tabla 5.8: Solución SOL-07.

Solución: SOL-08	
Actuación	El flujo de cada uno de los escenarios queda albergado como recurso externo dentro de un fichero .scxml.
Observaciones	

Tabla 5.9: Solución SOL-08.

Solución: SOL-09	
Actuación	Elección de estructuras JSON encapsuladas en ficheros con extensión .json debido a las razones que se justificaron en la página 20.
Observaciones	

Tabla 5.10: Solución SOL-09.

Personalmente, este Trabajo Fin de Grado ha supuesto una dura, intensa pero interesante, grata e incluso adictiva dedicación por su acercamiento al sector de los simuladores y al desarrollo de videojuegos en general.

Durante el recorrido de este proyecto no sólo se ha aprendido modelado 3D y programación de videojuegos, también algo mucho más general e importante, como es aprender a ver las cosas desde otro punto de vista y apostar siempre por la mejora de lo que ya se ha realizado.

5.2. Trabajos futuros

Aunque se han cumplido todos los requisitos del proyecto, siempre puede mejorar y ampliarse. Por ello, se proponen las siguientes líneas de trabajo futuro por orden de prioridad:

Corregir pequeños bugs con aplicaciones externas

Antes de acometer cualquier novedad, deberían corregirse los problemas presentados en el Apartado 3.4.9. La dificultad prevista para esta tarea es media.

Integración con redes sociales

Autenticación y autorización mediante OAuth con las principales redes sociales (Facebook, Twitter, etc). Es un proceso sencillo. Así podrían mostrarse en el escenario objetos personalizados, como por ejemplo que una fotografía encima del escritorio contenga una textura que es la foto de perfil de Facebook del usuario, descargada mediante la invocación al API de la red social. Puede conllevar una dificultad moderada, pero existen muchos ejemplos y documentación oficial en la web.

Base de datos con usuarios

Añadir al simulador una base de datos MySQL y manejarla con PHP para guardar usuarios de la aplicación y datos asociados (contactos, información personal...). Incluso podrían mostrarse alguno de estos datos en la página web (especie de ranking o algo así).

Además, el API de Unity3D incorpora clases como *WWW*, *WWWForm* y *Social* que podrían hacer esta tarea más fácil, aunque a priori se presenta como una tarea de complejidad alta teniendo en cuenta la parte de Unity, ya que PHP y MySQL no representarían dificultades significativas.

Página web del simulador

Crear una página HTML5 + JavaScript + CSS3 con información y promoción del simulador. Si se consigue exportar el juego a plataforma web, puede incluso embeberse en la propia página web. La web se haría pública situándolo en el servidor Apache. En este caso, se estima que se trataría de una tarea fácil.

Exportar a plataforma web y móvil

Flashplayer o Nativo de Chrome y Android e iOS. Parece que los únicos problemas que se presentan para que la actual versión no haya sido exportada a las versiones web es que hay que cambiar el método para cargar los ficheros externos, puesto que la actual forma no está permitida por cuestiones de seguridad.

La plataforma móvil sería más complicada puesto que el problema es que las plataformas móviles no soporta la función *eval()* puesto que todo su código debe ser precompilado.

Aplicación para Android que muestre la web con la base de datos

Consiste en una aplicación Android con Cordova (Phonegap) con una Activity que lanza un WebView que carga la web HTML5. La interacción entre el JavaScript de la web y el Java de Android y viceversa es posible, pudiendo incluso a acceder a servicios nativos como cámara, agenda, estado de la batería, GPS, sensores, preferencias de la aplicación (guardado de datos), etc.

Se indican esos permisos de acceso en Android y el usuario es notificado al instalar la aplicación. De este modo se consigue una aplicación con front-end HTML5 y JavaScript multiplataforma, pues para desarrollarla en otros sistemas como BlackBerry o iOS, basta con generar lo equivalente a una Activity de Android para cargar la web. Bastaría con colocar en la web un pequeño código JavaScript que, dependiendo del sistema operativo desde el que ha sido realizada la petición, cargue uno u otro CSS.

El "look & feel"de la aplicación puede hacerse nativo teniendo diferentes carpetas de recursos de imágenes y redirigiendo a una u otra desde el JavaScript de detección de sistema operativo. De hecho, dependiendo del tamaño del dispositivo móvil (tablet, Smartphone...) el CSS puede ajustarse.

Podría aprovecharse para extender la funcionalidad del sitio web tradicional, estimándose su complejidad en media.

Anexos

Anexo A

Diagrama Pert

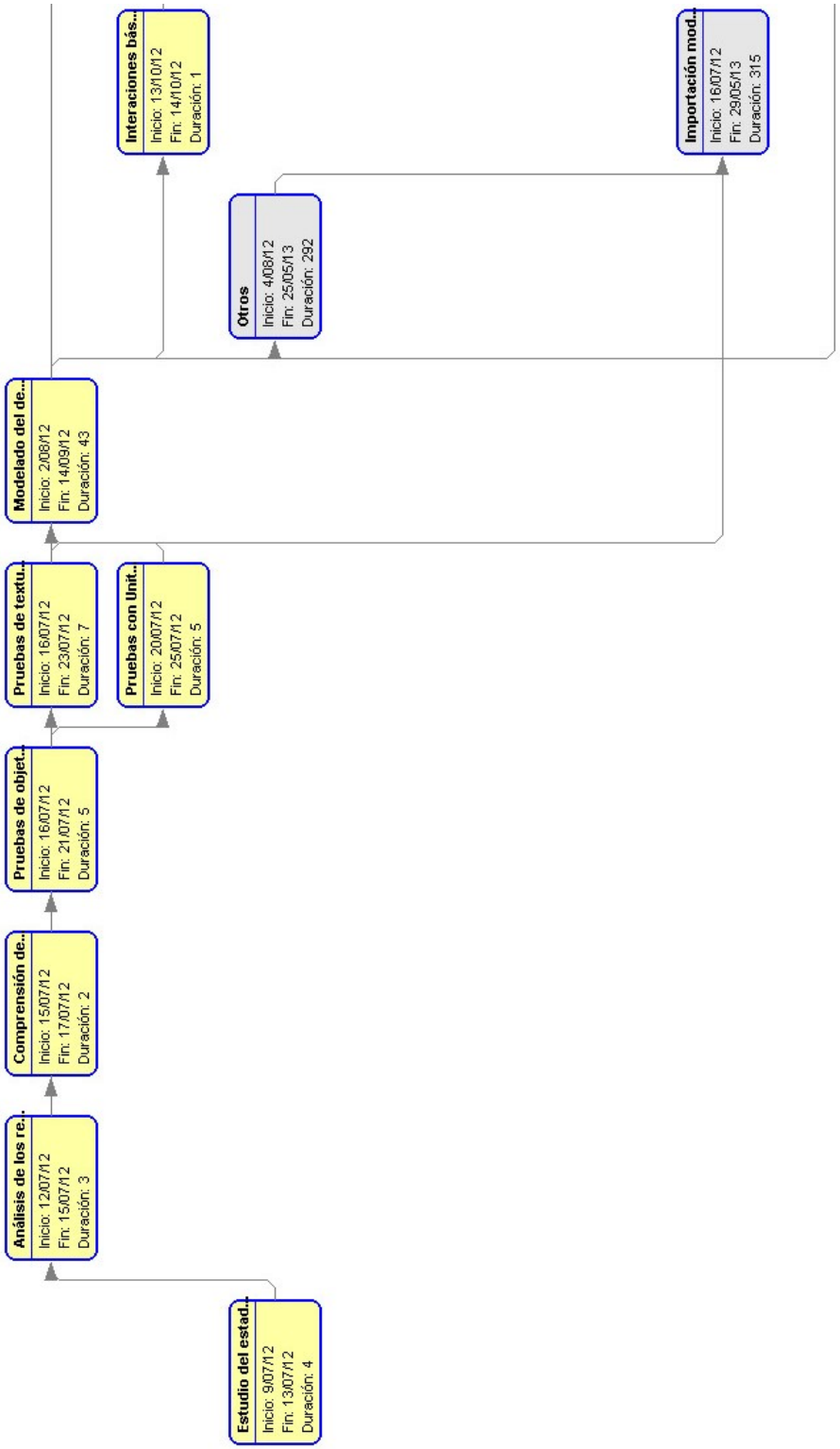


Figura A.1: Diagrama Pert del proyecto

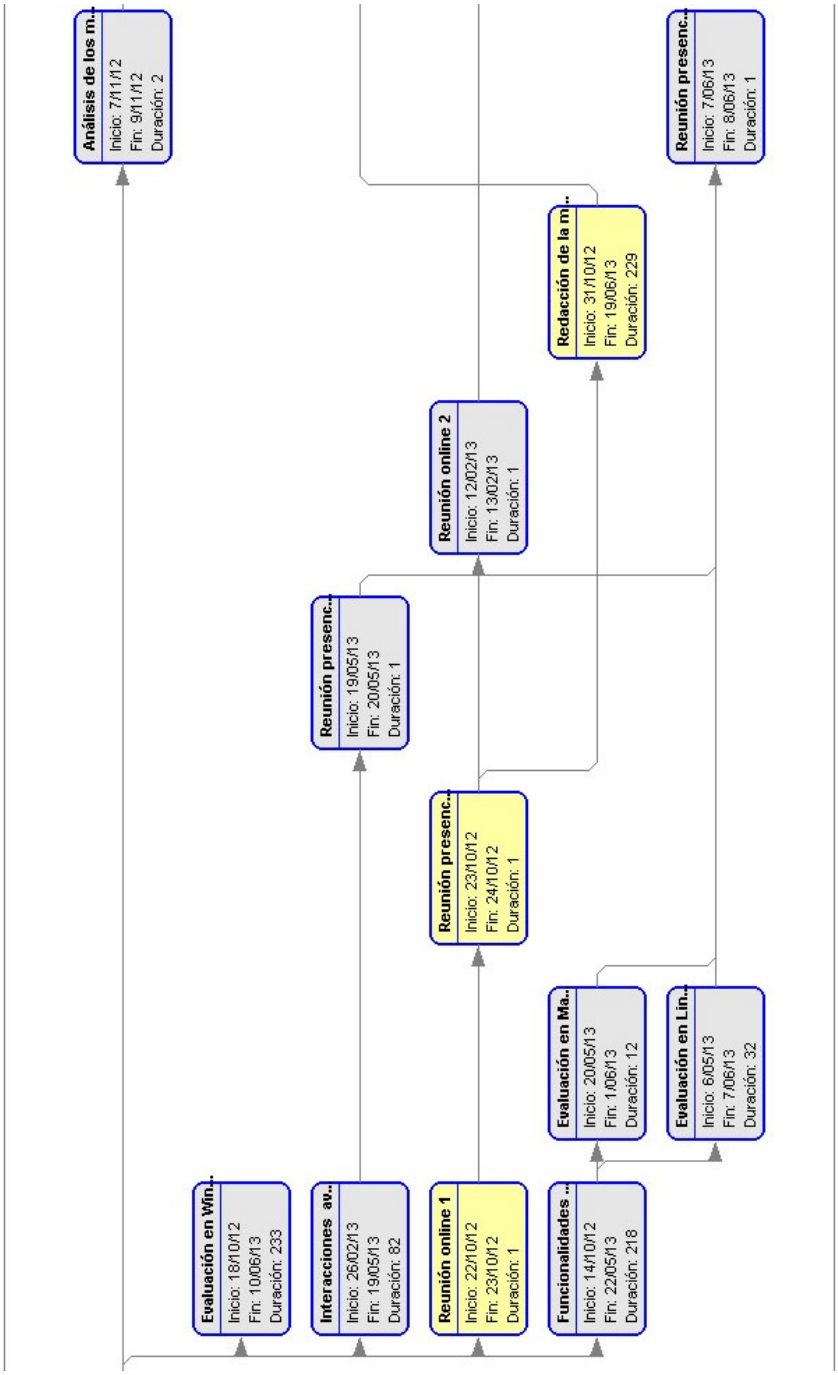


Figura A.2: Diagrama Pert del proyecto

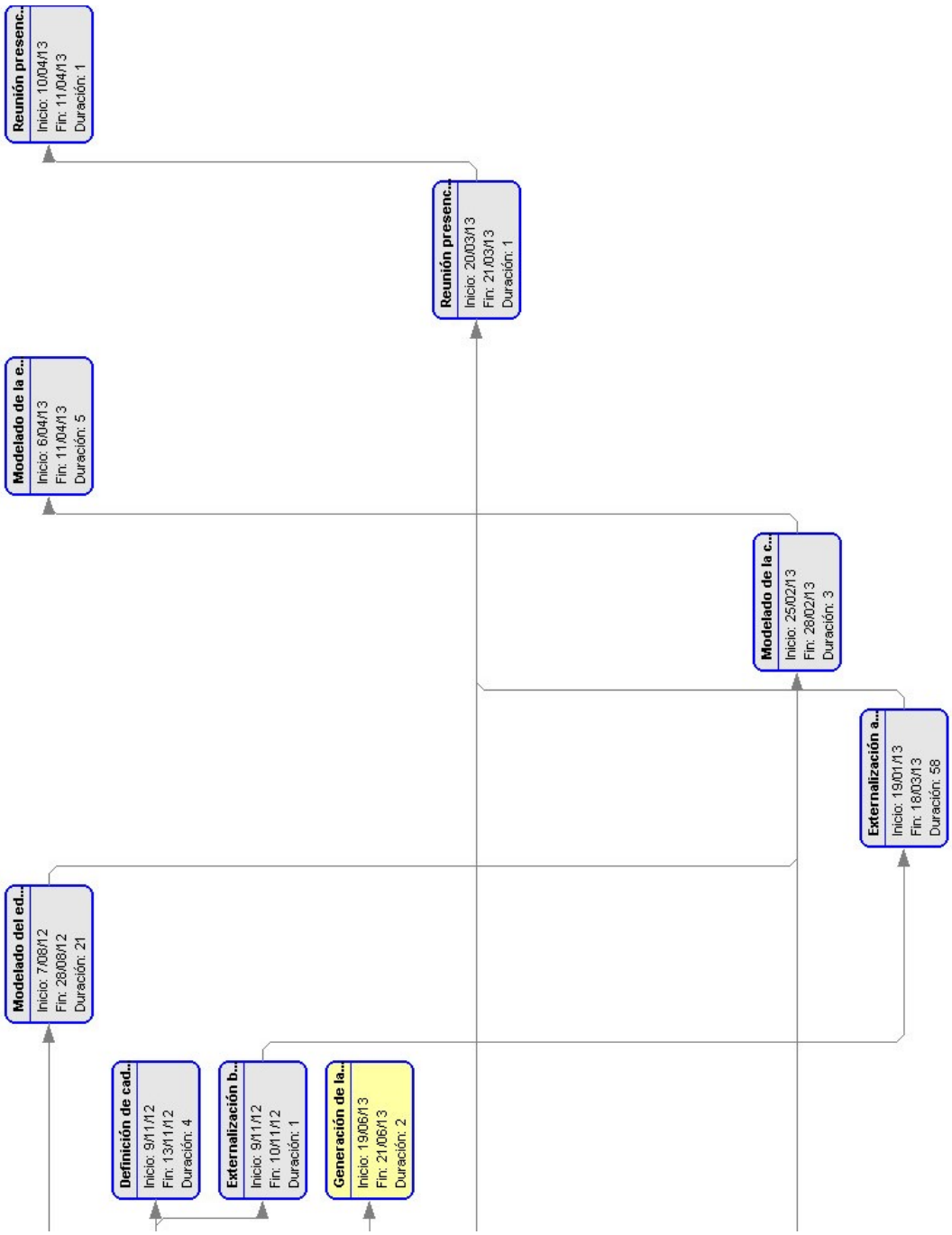
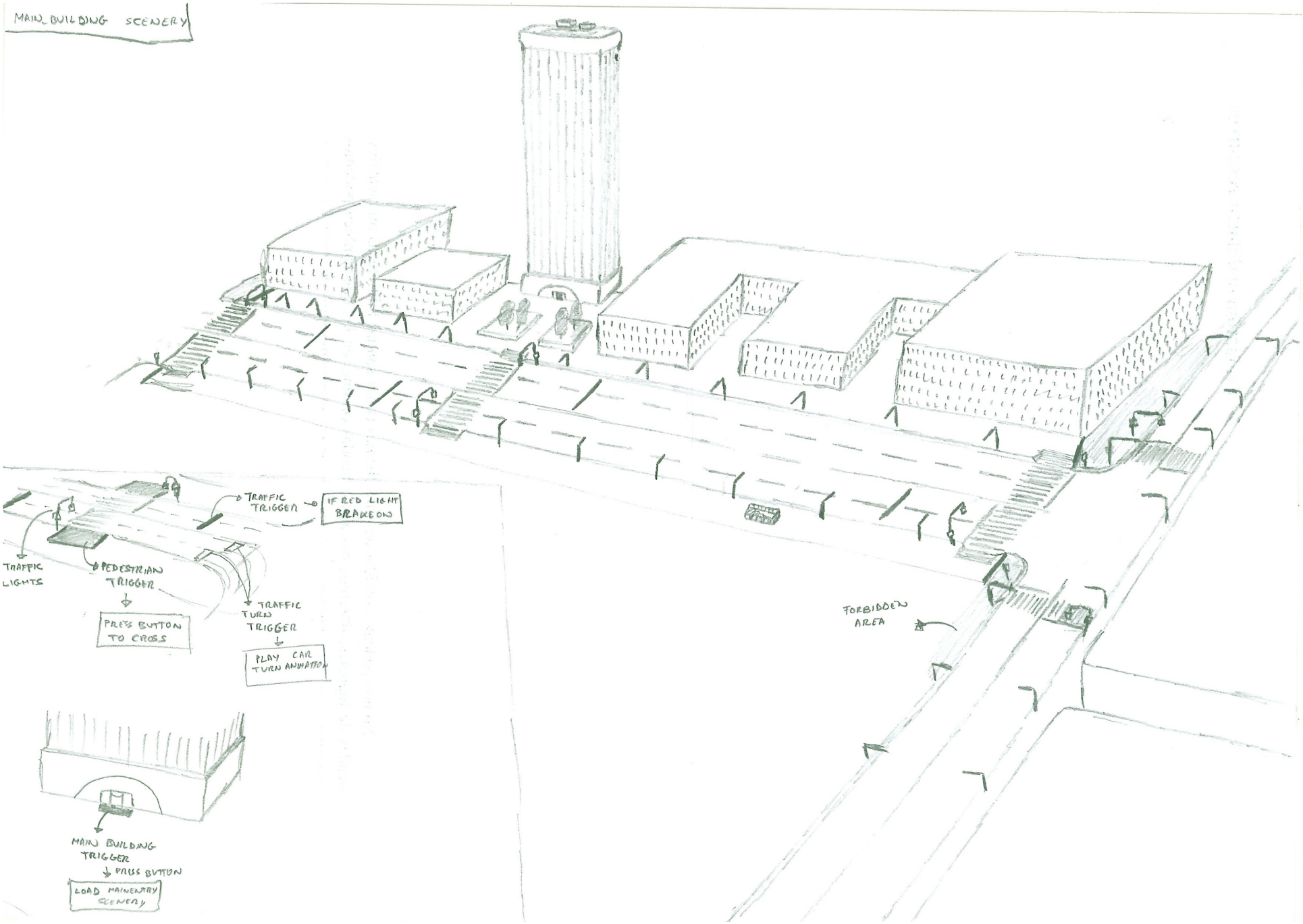


Figura A.3: Diagrama Pert del proyecto

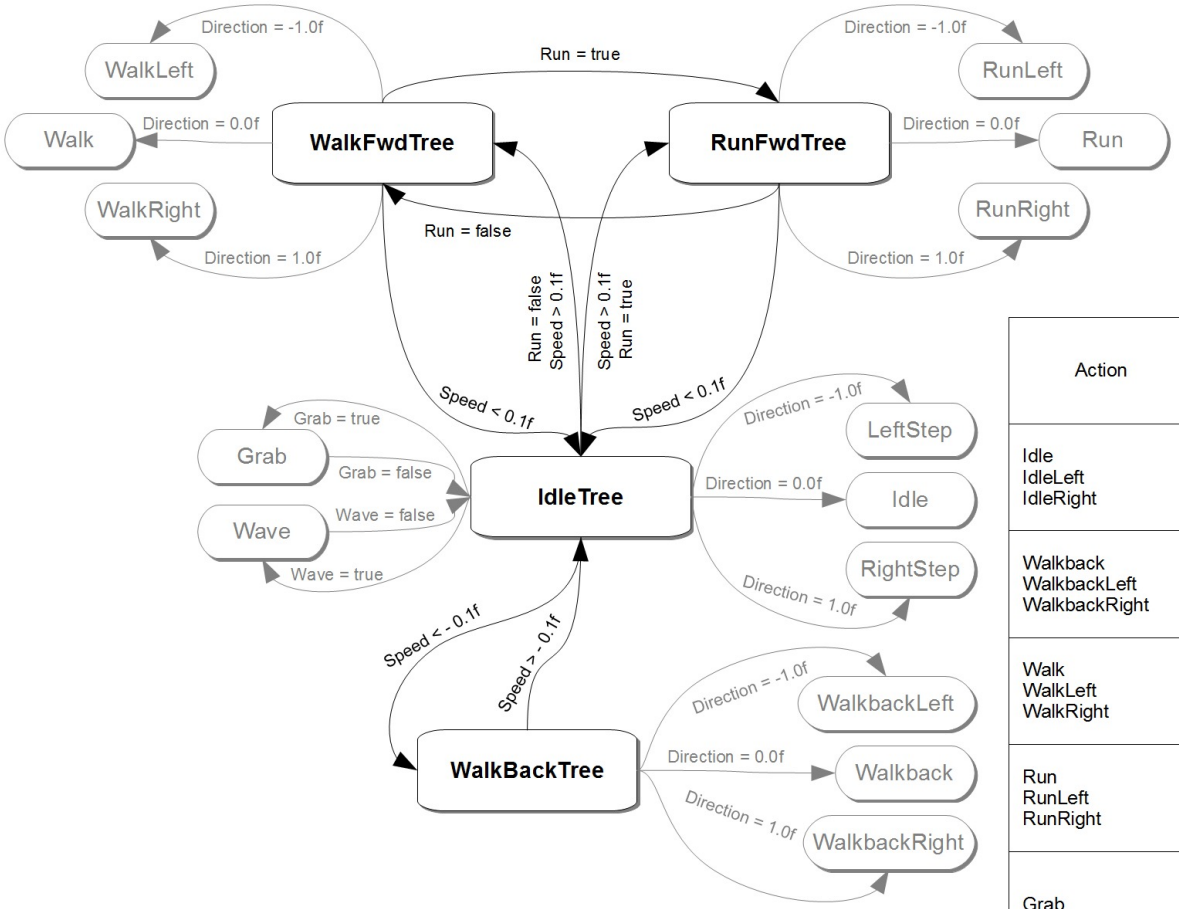
Anexo B

Esbozo del escenario de calle


MAIN_BUILDING SCENERY
TRAFFIC TRIGGER
IF RED LIGHT BRAKE ON
TRAFFIC LIGHTS
PEDESTRIAN TRIGGER
PRESS BUTTON TO CROSS
TRAFFIC TURN TRIGGER
PLAY CAR TURN ANIMATION
FORBIDDEN AREA
MAIN BUILDING TRIGGER
PASS BUTTON
LOAD MAINENTRY SCENERY

Anexo C

Máquina de estados del avatar



Action	Parameter				
	Direction	Speed	Run	Grab	Wave
Idle IdleLeft IdleRight	0 -1 +1	-0.1<x<+0.1	X	X	X
Walkback WalkbackLeft WalkbackRight	0 -1 +1	< -0.1	X	X	X
Walk WalkLeft WalkRight	0 -1 +1	> +0.1	false	X	X
Run RunLeft RunRight	0 -1 +1	> +0.1	true	X	X
Grab	X	X	X	true	X
Wave	X	X	X	X	true

Anexo D

Guía de usuario

D.1. Instalación

Una vez obtenida alguna de las versiones de Entrepreneur Simulator 3D, ya sea para Windows, Linux o Mac OS X, copiar la carpeta raíz a cualquier directorio del sistema, como se muestra en la Figura D.1.

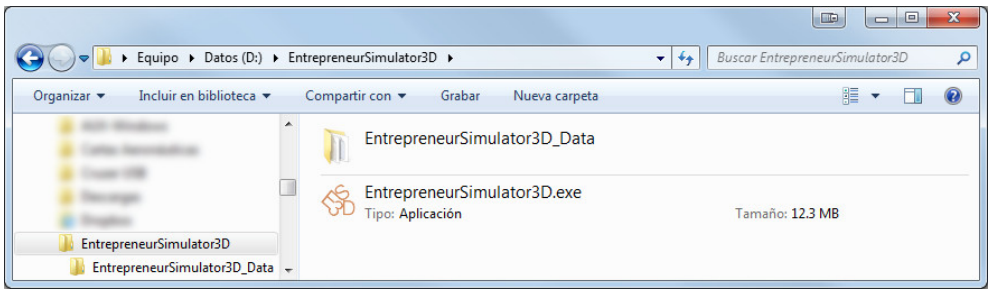


Figura D.1: Copiar la carpeta de la aplicación a cualquier directorio

D.2. Controles

Los controles pueden ser modificados por cualesquiera que el usuario quiera. En la Figura D.2 se muestran los controles por defecto de la aplicación.

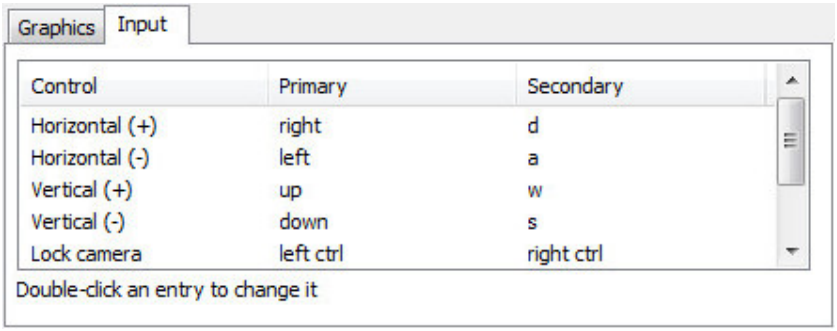


Figura D.2: Controles por defecto

Aparte, el ratón servirá para desplazar la vista en los ejes X e Y de tal forma que simulen el movimiento de la cabeza del jugador.

D.3. Inicio

Si es la primera vez que accedes a la aplicación y todavía no has empezado la partida, el menú que se mostrará será el de la Figura D.3.

Figura D.3: Menú del juego la primera vez

En caso contrario, verás el menú de la Figura D.4

Figura D.4: Menú del juego una vez se haya comenzado una partida

D.4. La pantalla de juego

Existen dos tipos de HUD (Heads-Up Display): el genérico, como el de la Figura D.5, y el extendido, como el de la Figura D.6

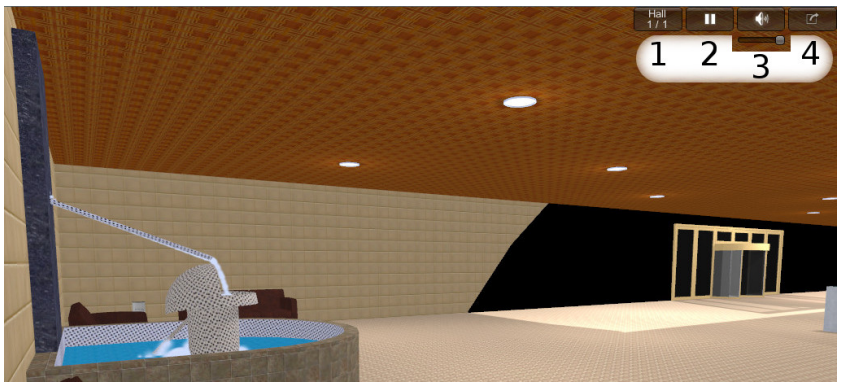


Figura D.5: Pantalla de juego genérica

Figura D.6: Pantalla de juego en su versión extendida

1. Botón de objetivos
   Muestra el nombre del escenario actual junto al número de objetivos conseguidos respecto del total del escenario. Pulsando este botón se muestra una lista con la descripción de los distintos objetivos.

2. Botón de pausa
   Si se pulsa durante la partida el juego queda pausado. Volviendo a pulsarlo se reanuda la partida.

3. Control de volumen
   El botón silencia o no el sonido del juego, mientras que la barra deslizante ajusta el nivel de éste.

4. Botón de salir
   Este botón lleva al menú principal del juego.

5. Botón de configuración de aplicaciones
   Este botón será mostrado en escenarios que contengan objetos interactivos capaces de lanzar aplicaciones externas y mostrará un menú de configuración de las mismas.

6. Botón de cámaras
   Efectúa el cambio de cámara en escenarios que dispongan de más de una cámara.

D.5. Modificación de los recursos externos

Este simulador tiene la característica de que sus contenidos pueden modificarse editando los ficheros que se encuentran en la carpeta `ExtResources` dentro del directorio de la aplicación, `EntrepreneurSimulator3D_Data`. La estructura de directorios se muestra en la Figura D.7 y se detalla más abajo.

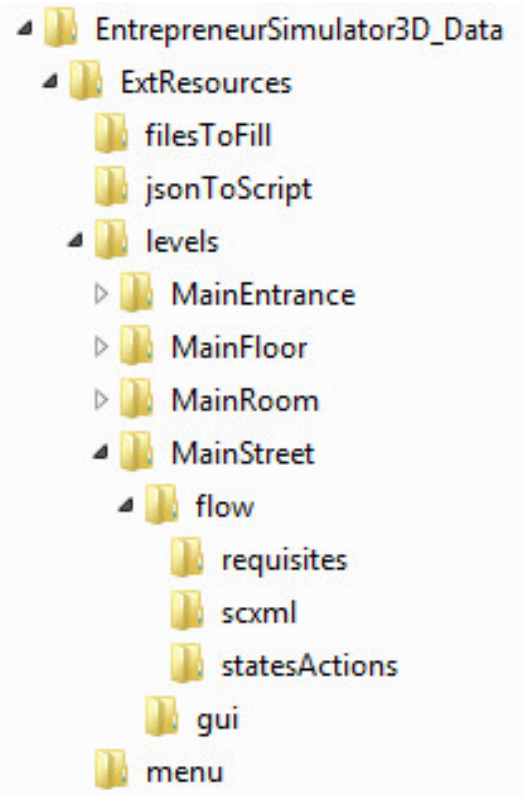


Figura D.7: Estructura de directorios de los recursos modificables

filesToFill
Contiene los ficheros que van a servir para ofrecer al usuario información personalizada. Pueden ser páginas web o cualquier otro tipo siempre y cuando sean archivos de texto.

El contenido que será sustituido debe ir encerrado entre caracteres $$ y $$. Por ejemplo: $$texto_a_reemplazar$$.

jsonToScript
Contiene el fichero que sirve para generar las Interfaces Gráficas de Usuario (GUI) a partir de etiquetas en los ficheros existentes en el directorio `levels/levelName/gui`.

levels/levelName/gui
Aquí se encuentran los ficheros encargados de generar las Interfaces Gráficas de Usuario en el juego. Dentro de cada uno de los ya creados existen instrucciones más específicas para modificarlo, como las etiquetas que son aceptadas, los atajos de escritura como cambiar *'Screen.width/2'* por *'center'*, *'Screen.width'* por *'fill'*, etc.

levels/levelName/flow/scxml
Contine el fichero `flow.scxml`, que proporciona el flujo de la historia del escenario. Así, define los diferentes estados, las transiciones entre ellos y los eventos encargados de que las transiciones ocurran.

levels/levelName/flow/requisites
Contiene el fichero `requisites.json`, encargado de definir todos los objetivos necesarios de ese nivel y sus descripciones, y el fichero nombrado `states_requisites.json`, donde se asignan los objetivos imprescindibles para acceder a los diferentes estados del flujo de la historia.

levels/levelName/flow/statesActions
Contiene el fichero `states_actions.json`, donde se encuentran definidos todos las acciones que deben ejecutarse cuando el flujo cambia de estado.

menu
Contiene tres ficheros. El primero de ellos, `menu_translations.json`, contiene los idiomas que va a soportar el juego y la traducción de los botones del menú principal del juego. El segundo, `hud_translations.json`, define las traducciones a los distintos idiomas de todos los textos que guardan relación con las interfaces gráficas y los HUD. Finalmente, `mainfloor_signboards.json` contiene las traducciones de los carteles que aparecen en el escenario de la planta de oficinas.

Para ver una descripción más detallada de la estructura de los ficheros y sus peculiaridades, abrir estos con cualquier editor (se recomienda Notepad++, Gedit o similar) e inspeccionar su cabecera.

Es MUY IMPORTANTE que todos los ficheros modificados mantengan las propiedades originales: Codificación UTF-8 (sin BOM) y Fin de línea con formato Windows.

D.6. Requisitos mínimos

Sistema Operativo	Windows XP o superior / Mac OS X 10.6 o superior / Linux
DirectX	DirectX 9
Procesador	2 GHz o superior
RAM	512 GB
Tarjeta gráfica	128 MB compatible con shader model 2.0
Espacio en disco	261 MB (282 MB para Linux)

Tabla D.1: Requisitos mínimos del sistema

Sistema Operativo	Windows XP o superior / Mac OS X 10.6 o superior / Linux
DirectX	DirectX 10
Procesador	Arquitectura de doble o cuádruple núcleo a 2.4 GHz o superior
RAM	4 GB
Tarjeta gráfica	512 MB compatible con shader model 2.0
Espacio en disco	261 MB (282 MB para Linux)

Tabla D.2: Requisitos recomendados del sistema

Bibliografía

[1] *Madrid: Torre Picasso, 157m, 44p (Minoru Yamasaki) - Página 3 (Detalles de la estructura).* [En línea], (2007). Recuperado el 7 de agosto de 2012 desde http://www.urbanity.es/foro/rascacielos-y-highrises-mad/103-madrid-torre-picasso-157m-44p-minoru-yamasaki-3.html#post6248.

[2] *Introducción a JSON.* [En línea], (2012). Recuperado el 8 de noviembre de 2012 desde http://www.json.org/json-es.html.

[3] *Kerkythea.* [En línea], (2012). Recuperado el 2 de agosto de 2012 desde http://www.kerkythea.net/joomla/.

[4] *Torre Picasso - WikiArquitectura.* [En línea], (2012, octubre 4). Recuperado el 7 de agosto de 2012 desde http://es.wikiarquitectura.com/index.php/Torre_Picasso.

[5] *SketchUp - Wikipedia, the free encyclopedia.* [En línea], (2013). Recuperado el 18 de junio de 2013 desde http://en.wikipedia.org/wiki/SketchUp.

[6] *Unreal Engine - Wikipedia, la enciclopedia libre.* [En línea], (2013). Recuperado el 18 de junio de 2013 desde http://es.wikipedia.org/wiki/Unreal_Engine.

[7] *YUIDoc - Javascript Documentation Tool.* [En línea], (2013). Recuperado 1 de mayo de 2013 desde http://yui.github.io/yuidoc/.

[8] *COLLADA - Wikipedia, the free encyclopedia.* [En línea], (2013, abril 29). Recuperado el 18 de julio de 2013 desde http://en.wikipedia.org/wiki/COLLADA.

[9] Accesor Applications and Services. *PASILLO MOTORIZADO HIDDEN GATE-ACR - Productos de Torniquetes en Accesor.* [En línea], (2013). Recuperado el 30 de abril de 2013 desde http://www.accesor.com/esp/detail_product.php?id_article=142.

[10] Aprendelo.com. *Banco de sonidos.* [En línea], (2013). Recuperado el 14 de junio de 2013 desde http://www.aprendelo.com/rec/banco-imagenes-sonidos.html.

[11] Autodesk, Inc. *FBX - Platform-Independent 3D Data Interchange Technology - Autodesk.* [En línea], (2012). Recuperado el 17 de julio de 2012 desde http://usa.autodesk.com/fbx/.

[12] María Eugenia Caldas Blanco, Reyes Carrión Herráez, and Antonio J. Heras Fernández. *Empresa e iniciativa emprendedora.* Editex, 2011.

[13] Comunidad de Madrid. *Crea y Compite: Simulador de Creación y Gestión de Empresas.* [En línea], (2012). Recuperado el 10 de julio de 2012 desde https://gestiona.madrid.org/esim_pub/run/j/Login.icm.

[14] Refsnes Data. *W3Schools Online Web Tutorials.* [En línea], (2012). Recuperado el 13 de julio de 2012 desde http://www.w3schools.com/.

[15] Ministerio de Educación - Instituto de Tecnologías Educativas. *Banco de Imágenes y Sonidos.* [En línea], (2013). Recuperado el 25 de febrero de 2013 desde http://recursostic.educacion.es/bancoimagenes/web/.

[16] Google Inc. *SketchUp - Paquetes extra.* [En línea], (2011). Recuperado el 2 de agosto de 2012 desde http://www.google.com/intl/es/sketchup/bonuspacks.html.

[17] Inkscape. *Inkscape. Dibuja Libremente.* [En línea], (2013). Recuperado el 31 de mayo de 2013 desde http://inkscape.org/.

[18] Pontegadea Inmobiliaria. *· · · TORRE PICASSO · · · (Características).* [En línea]. Recuperado el 7 de agosto de 2012 desde http://www.per-gestora.com/ing/conoce_curiosidades.htm.

[19] Pontegadea Inmobiliaria. *· · · TORRE PICASSO · · · (Planos).* [En línea]. Recuperado el 7 de agosto de 2012 desde http://www.torrepicasso.info/esp/comercial_caracteristicas_planos.htm.

[20] SCXML Web Laboratory. *Synergy SCXML Web Laboratory.* [En línea], (2012). Recuperado el 13 de julio de 2012 desde http://www.ling.gu.se/~lager/Labs/SCXML-Lab/.

[21] lcn75. *change mouse cursor on mouseover - Unity Answers.* [En línea], (2012, enero 4). Recuperado el 19 de mayo de 2013 desde http://answers.unity3d.com/questions/42135/change-mouse-cursor-on-mouseover.html.

[22] lukasaurus. How to - Import a Sketchup (.skp) Model into Unity (with Textures). [Mensaje en línea], (2012, mayo, 29). Recuperado 17 julio de 2012 desde http://forum.unity3d.com/threads/99965-How-to-Import-a-Sketchup-(-skp)-Model-into-Unity-(with-Textures).

[23] Álvaro García Uzquiano. *SimEmergencias: simulador para formación en atención de emergencias extra-hospitalarias.* Proyecto Fin de Carrera, junio 2011.

[24] Microsoft. *.NET Framework 2.0.* [En línea], (2013). Recuperado el 19 de enero de 2013 desde http://msdn.microsoft.com/en-us/library/aa139623.aspx.

[25] Ministerio de Industria Turismo y Comercio. *:: Simul@ :: Juego de Simulación Empresarial.* [En línea], (2012). Recuperado el 10 de julio de 2012 desde http://servicios.ipyme.org/simulador/loginWeb.aspx.

[26] Mixamo. *Mixamo: Production-quality 3d character animation in seconds.* [En línea], (2013). Recuperado el 8 de febrero de 2013 desde http://www.mixamo.com/.

[27] Ofiprix. *Kuorum.* [PDF en línea]. Recuperado el 2 de agosto de 2012 desde http://www.ofiprix.com/descargas/kuorum.pdf.

[28] Ofiprix. *Manager.* [PDF en línea]. Recuperado el 2 de agosto de 2012 desde http://www.ofiprix.com/descargas/manager.pdf.

[29] SoundJay.com. *SoundJay.com - Free Sound Effects.* [En línea], (2013). Recuperado el 14 de junio de 2013 desde http://www.soundjay.com/.

[30] Unity Technologies. *Unity - ShaderLab syntax: Culling & Depth Testing.* [En línea], (2012, septiembre 5). Recuperado el 6 de abril de 2013 desde http://docs.unity3d.com/Documentation/Components/SL-CullAndDepth.html.

[31] Unity Technologies. *MecanimTute.* [En línea], (2013). Recuperado el 9 de febrero de 2013 desde http://files.unity3d.com/will/MecanimTute.zip.

[32] Unity Technologies. *MecanimTuteStarterPack.* [En línea], (2013). Recuperado el 9 de febrero de 2013 desde http://u3d.as/content/unity-technologies/raw-mocap-data-for-mecanim/3Bt.

[33] The GIMP Team. *GIMP - The GNU Image Manipulation Program.* [En línea], (2012). Recuperado el 2 de agosto de 2012 desde http://www.gimp.org/.

[34] tonioloewald. *tonioloewald/jsonparse · GitHub*. [En línea], (2013). Recuperado el 21 de febrero de 2013 desde https://github.com/tonioloewald/jsonparse.

[35] Trimble. *Trimble SketchUp*. [En línea], (2012). Recuperado el 9 de julio de 2012 desde http://sketchup.google.com/intl/es/.

[36] Trimble. *Trimble SketchUp: SketchUp Plugins*. [En línea], (2012). Recuperado el 2 de agosto de 2012 desde http://sketchup.google.com/intl/en/download/plugins.html.

[37] Unity Technologies. *Unity - Game Engine*. [En línea], (2012). Recuperado el 9 de julio de 2012 desde http://unity3d.com/unity/.

[38] Unity Technologies. *Unity - Unity Manual*. [En línea], (2012). Recuperado el 18 de octubre de 2012 desde http://docs.unity3d.com/Documentation/Manual/.

[39] Universal-Soundbank. *Ambiances sonores gratuites*. [En línea], (2013). Recuperado 14 de junio de 2013 desde http://www.universal-soundbank.com/ambiances-sonores.htm.

[40] W3C. *eXtensible Markup Language*. [En línea], (2012). Recuperado el 12 de julio de 2012 desde http://www.w3.org/XML.

[41] W3C. *State Chart XML (SCXML): State Machine Notation for Control Abstraction*. [En línea], (2012). Recuperado el 12 de julio de 2012 desde http://www.w3.org/TR/scxml/.

[42] W3C. *World Wide Web Consortium*. [En línea], (2012). Recuperado el 12 de julio de 2012 desde http://www.w3.org/.