Escuela
Politécnica
Superior

Elaboración de una guía sobre cómo desarrollar un GameEngine Entity-Component-System en C++20

Grado en Ingeniería Multimedia

Trabajo Fin de Grado

Autor:
Laureano Cantó Berná (alumno)
Tutor:
Francisco José Gallego Durán (tutor)

Universitat d'Alacant
Universidad de Alicante

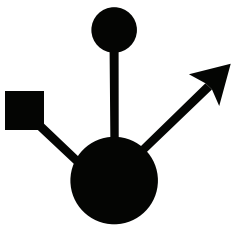
Mayo 2023

Elaboración de una guía sobre cómo desarrollar un GameEngine Entity-Component-System en C++20

Memoria del proyecto

Autor
Laureano Cantó Berná (alumno)

Tutor
Francisco José Gallego Durán (tutor)
Departamento de Ciencia de la Computación e Inteligencia Artificial



Grado en Ingeniería Multimedia

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Mayo 2023

Preámbulo

Actualmente los videojuegos nos ofrecen muchas horas de entretenimiento a lo largo de nuestra vida. Es por eso que cada vez se busca más avance y mejora en la tecnología para lograr los mejores títulos.

Este proyecto se centra en una pequeña parte dentro de la tecnología y arquitectura de la creación de videojuegos y concretamente en la arquitectura de software ”Entity Component System” (ECS).

Indagaremos de menos a más en esta arquitectura y la aplicaremos para crear estos motores de videojuegos, que mediante pequeños ejemplos y apartados teóricos, serán cada vez más completos. Pero el objetivo principal no será este, si no crear una guía del proceso de desarrollo de estos motores de videojuegos basados en la arquitectura ECS.

Agradecimientos

Este trabajo no habría sido posible sin el apoyo de todos mis seres queridos, ya que todos me llenan de palabras de ánimo y fuerza para seguir haciendo aquello que me gusta día a día.

Principalmente quiero agradecer a mi madre, María Victoria, y a mi padre, Laureano, que me siguen y apoyan en todas las decisiones que tomo, y sin ellos pude haberlo dejado en los momentos más difíciles. Llegar hasta aquí y presentar un proyecto al que he dedicado tanto esfuerzo y tiempo me llena de orgullo, pero se que a ellos dos todavía más, y me hace muy feliz, ya que es una forma directa de devolverles ese apoyo y amor recibido durante estos años en la Universidad, con un buen resultado.

También es importante nombrar a mis padrinos, Antonio y Carmen, que han sido durante toda mi vida, unos segundos padres, y que han apoyado de la misma forma toda mi trayectoria, en este proyecto también ellos merecen un hueco. Gracias por ese apoyo y amor constante.

Hay una persona, mi primo Rubén, que ha hecho este camino antes que yo, haciendo esta ingeniería, y siendo un gran apoyo en mi avance. A él también quiero dedicarle estas palabras ya que es mi ejemplo a seguir y una figura que admiro mucho.

Seguiré hablando de mi círculo académico cercano, donde hay dos personas principalmente que merecen un agradecimiento. En primer lugar, quiero agradecer a Jorge Marín, mi mejor amigo dentro de la carrera, que también ha apoyado todas mis decisiones y ha dado valor a todo lo que hago, con opiniones, ayuda y siempre buenos gestos. Me hace feliz que después de todo, sigamos teniendo una amistad fuera de la carrera, y quien sabe, ojalá trabajar juntos en un futuro. Desde aquí muchas gracias Jorge. Y no puede faltar, Alicia, mi pilar fundamental durante estos cinco años de carrera, mi pareja. Gracias a ti he conseguido todos mis logros, y digo gracias a ti por que eres la que ha conocido cada uno de mis movimientos, cada uno de mis fallos, y cada uno de mis éxitos, quiero agradecértelo de corazón, por que has sido fundamental, tanto en este proyecto, como en todos y cada uno de los momentos importantes durante todo este tiempo.

Agradecer también a mis dos mejores amigos, Manuel Vicente, y Nicolás, que gracias a ellos, nunca me ha faltado la sonrisa y la fuerza, gracias a su apoyo y amistad. Este proyecto también es por vosotros.

Y por último, quiero agradecer a Francisco José Gallego Durán, mi tutor en este último proyecto, y profesor en cuarto curso. Gracias a sus consejos y su conocimiento, he logrado estar donde quiero dentro del mundo de la ingeniería, y por supuesto todo mi reconocimiento, por tanta paciencia y dedicación que has puesto en este proyecto. Este resultado, no habría sido el mismo sin el esfuerzo que has dedicado a cada una de las reuniones que hemos tenido.

Es a ellos a quien dedico este trabajo.

Si quieres cambiar el mundo,
toma tu pluma y escribe.

Martín Lutero.

No hay nada que dé tanta vergüenza
como ver a alguien hacer algo
que uno dijo que era imposible hacer.

Sam Ewing.

Cuando las máquinas funcionan mal
es cuando nos recuerdan de verdad
cuánto poder tienen.

Clive James

Índice general

Índice de figuras

1. Introducción

Los videojuegos han sido una forma de entretenimiento popular durante décadas, y a medida que la tecnología ha evolucionado, también lo han hecho los videojuegos. Desde los simples juegos de arcade hasta los videojuegos en línea masivos y los juegos de realidad virtual, los videojuegos se han convertido en una parte importante de nuestra cultura y han sido influenciados por la tecnología que los rodea.

La industria del videojuego es una industria en constante evolución, con nuevas tecnologías y tendencias emergentes todo el tiempo. En la actualidad, existen entornos de desarrollo y tecnologías innovadoras para crear videojuegos, como Unreal Engine o Unity. Sin embargo, también es posible crear videojuegos desde cero. Para ello, es necesario comprender otras tecnologías para estructurar los datos y los diferentes actores de nuestro diseño.

Cuando hablamos de estructura, aparecen los patrones de diseño. Estos patrones rigen unas bases de estructura para nuestro código, creando una organización y optimización inmensa. Uno de los patrones de diseño más recientes y populares en el mundo de los videojuegos es el Entity Component System (ECS) para el desarrollo de motores de videojuegos.

La arquitectura ECS es un enfoque de diseño de software que aborda el problema de cómo manejar y administrar entidades complejas en un videojuego. A diferencia del enfoque de herencia clásico en la programación de videojuegos, que puede resultar rígido e inflexible, el ECS divide una entidad en sus componentes individuales y los administra de manera independiente.

Otorga un enfoque de diseño de software diferente a la hora de desarrollar videojuegos, mejorando el rendimiento, la escalabilidad, la flexibilidad y consiguiendo un código más independiente y reutilizable. Además, se adapta muy bien a la programación paralela, lo que permite aprovechar las capacidades de los procesadores multicore.

En este proyecto, abordaremos el desarrollo de una guía para motores de videojuegos que utilizan una arquitectura Entity Component System desde cero en C++. La guía tiene como objetivo proporcionar a los desarrolladores de videojuegos unos conocimientos y herramientas útiles para la creación de motores de videojuegos, logrando todas las cualidades mencionadas anteriormente.

Además del ECS, también abarcaremos otros temas importantes en el desarrollo de motores de videojuegos. Hablaremos de estructuras de datos complejas para guardar nuestros componentes en ellas. Los Slotmaps, o más conocidos como matrices dispersas que nos harán la vida más fácil.

También discutiremos el manejo de eventos, que es crucial en los videojuegos, y cómo podemos integrarlo en nuestro motor. Discutiremos cómo implementar diferentes eventos, como eventos de entrada de teclado, colisiones y mucho más.

2. Marco Teórico

En el apartado actual nos referiremos a toda la teoría que tenemos que conocer para comprender el proyecto. Hablaremos sobre videojuegos, sobre motores, sobre el modelo orientado a datos ECS y sus diferentes partes, estructuras de programación, las cuales son útiles para el desarrollo del propio motor, lenguajes de programación y otros temas de suma importancia a conocer antes del desarrollo.

Los dos siguientes apartados son un resumen teórico de lo que el libro en cuestión, "Cómo desarrollar un GameEngine Entity-Component-System en C++20", contiene, por lo tanto para el detalle completo mirar el documento (A).

2.1. El mundo de los Videojuegos

A día de hoy todos sabemos lo importante que han llegado a ser los videojuegos en nuestra sociedad. Han evolucionado desde pequeños juegos donde solo podíamos mover píxeles cuadrados y pequeños, a grandes obras de arte donde hasta el último detalle es importante.

Por ejemplo, uno de los juegos más importantes a nivel histórico fue el denominado "Pong". Un videojuego sencillo donde había poco contenido visual, pero el necesario para poder tener un objetivo a conseguir y unas mecánicas sencillas. En un juego de este estilo, a lo mejor no es tan necesaria una estructura tan marcada, por que el máximo de objetos que puedes tener son dos raquetas y una pelota, aunque siempre es recomendable.

En el otro lado, ya muy evolucionados, tenemos videojuegos más avanzados que requieren de estructuras y patrones de diseño ya que disponen de muchos objetos, y muchos factores distintos que modifican lo que vemos constantemente en la pantalla. Cuando el tamaño de objetos no se puede contar con las manos, puede volverse muy difícil el desarrollo del videojuego, además de sufrir tirones o errores por mala optimización. Es por eso que más adelante hablaremos de patrones de diseño y en concreto este proyecto trata sobre uno de ellos.

2.2. Desarrollo de videojuegos

Los videojuegos para muchos son la forma de escapar de la realidad, o emplear su tiempo libre en divertirse con ellos. Pero para gran cantidad de personas, ha llegado a ser su trabajo. Se utilizan en la enseñanza, e incluso algunos han dado pie a eventos y competiciones. Detrás de todo esto hay grandes equipos de desarrollo, los cuales se dividen el trabajo para conseguir grandes resultados en el menor tiempo posible.

Para desarrollar estos videojuegos se cuenta con muchos lenguajes de programación, que en sí son la herramienta base de trabajo de los programadores, pero el lenguaje más usado en la industria es C++. Un lenguaje multiparadigma, con el que podemos generar programas compactos de gran rapidez. Además cuenta con numerosos compiladores, y muchas actualizaciones y estándares, ya que aunque tiene más de 35 años, tiene detrás una comunidad que lo mantiene actualizado y muy cuidado.

En este mundo hace falta mucha organización y una buena estructuración del código, ya que muchas personas pueden utilizarlo a la vez. Además, con los avances tecnológicos se requiere de sistemas estructurales más complejos y para esto los desarrolladores crean motores donde se organizan los datos de una forma compleja y en bajo nivel, y así ofrecer una interfaz entendible para otros desarrolladores cuyo objetivo es crear videojuegos. A continuación hablaremos de estos motores.

2.2.1. Motores de videojuegos

Un motor es lo que entendemos como un conjunto de librerías que permite el diseño, creación y representación de un videojuego. Hay diversos tipos de motores, por ejemplo un motor gráfico, o uno de físicas. Éstos resuelven problemas difíciles y un programador los puede usar para facilitar el desarrollo y no tener que crearlos desde cero.

Otros programadores deciden desarrollarlos; en este caso nosotros desarrollamos el motor que gestiona todos los datos del videojuego, y a su vez usaremos un motor gráfico ya creado para representar visualmente los resultados.

Esta organización la basaremos en una arquitectura de software la cual mencionaremos más tarde, pero a continuación hablaremos de qué es la arquitectura de software.

2.2.2. Arquitectura de software

La arquitectura de software es una disciplina que se enfoca en la estructura, diseño y organización de los componentes de un sistema de software. Es un enfoque fundamental para desarrollar aplicaciones robustas, escalables y mantenibles. La arquitectura de software proporciona un marco conceptual que define cómo se organizan y relacionan los diferentes módulos o componentes del software, y cómo interactúan entre sí y con el entorno.

En esencia, la arquitectura de software establece la base para el desarrollo de software, definiendo los principios, patrones y estilos que guían el diseño y la implementación. Un buen diseño arquitectónico permite la creación de sistemas flexibles y adaptables a medida que evolucionan los requisitos y las tecnologías. Además, ayuda a gestionar la complejidad del software al descomponerlo en módulos más pequeños y manejables, lo que facilita el desarrollo colaborativo y la reutilización de componentes.

Si queremos desarrollar videojuegos robustos y fáciles de mantener, debemos cumplir una serie de reglas recomendables.

En nuestro caso, hablaremos del Entity Component System, la arquitectura de software sobre la que basaremos nuestros proyectos y esta guía.

2.2.2.1. Entity Component System

Un Entity Component System (ECS), es un tipo de arquitectura de software muy usado en videojuegos. Contiene entidades compuestas por componentes de datos, con unos sistemas que operan con estos componentes.

Sigue el principio de composición por herencia, lo cual significa que cada entidad está definida por los componentes que la forman, y los sistemas actúan de forma global sobre todas las entidades que posean el tipo de componente que el sistema solicita.

Una entidad pasa a ser un índice que le permite buscar componentes asociados con esa entidad. La mayoría de sus sistemas operan con subconjuntos de componentes, lo que nos genera una optimización enorme en el acceso a nuestros datos.

2.3. Estado del Arte

Antes de elaborar el proyecto, tenemos que tener en cuenta los trabajos actuales o antecedentes a la hora de explicar de forma guiada cómo hacer un ECS. Necesitaremos estudiar qué han hecho mal y qué han hecho bien otros, para que el nuestro sea novedoso, único y tenga menos fallos.

Una vez expuestos los diferentes tipos de proyectos actuales, tendremos que llegar a una conclusión final para realizar el nuestro. Esta conclusión está al final de esta sección y dejará clara la intención del proyecto.

2.3.1. Antecedentes o trabajos actuales

Esta sección del proyecto trata sobre trabajos existentes ya sean de ámbito académico, como otros trabajos de fin de grado o máster de otros estudiantes, o de proyectos con una visión más profesional.

Es difícil catalogar como iguales o similares a otros proyectos existentes basados en cómo hacer un videojuego ya que no se han encontrado trabajos similares.

Como referencia de proyectos similares anteriores podemos destacar alguna página web, blog o post que hable sobre como crear un ECS o videojuego sencillo, aunque no represente este proyecto al cien por cien.

A parte de las guías que haya a día de hoy, también hay que tener en cuenta los motores ECS que ya están creados, por eso comentaremos los motores "Entt" de Caini (2017) y "ECS" de Bloomberg (2016).

2.3.1.1. Guías sobre motores ECS

En este apartado nos referiremos solamente al contenido existente en menor medida, ya sean blogs o artículos explicando la creación o funcionalidades de motores ECS.

En primer lugar tenemos el trabajo de David Colson (2020), "How to make a simply ECS in C++". Este blog propone hacer un ECS del tipo que usa Unity, por ejemplo, usando estructuras para almacenar componentes y donde cada entidad tiene una máscara que los representa. El objetivo de esta pequeña guía es que entiendas y aprendas cómo funciona un simple ECS modificable para diferentes tipos de juegos.

Este segundo artículo habla también sobre la creación de un ECS cuyo enfoque es priorizar cómo se usa la memoria para almacenar los datos en C++. Otro de los temas que cubre es la metaprogramación. Según el autor, en este artículo no se pretende empezar y acabar un ECS completo si no explicar conceptos y obtener un resultado suficiente para su propósito, es decir, las explicaciones que otorga son en base a un proyecto existente propio. Este artículo fue escrito por un usuario de la página indiegamedev.net llamado, Deckhead (2020)

El último de los artículos que incluyo en este apartado es el trabajo de Morlan (2019) , el cual está pensado para ser un punto de partida para gente curiosa en cómo se hacen los videojuegos. El autor de este artículo enseña su proyecto personal, explicando algunos de los pasos más importantes, llegando a ser también una guía de cómo elaborar un motor sencillo de este estilo.

A mi parecer éste último es el mejor de los tres ejemplos propuestos, ya que muestra visualmente avances y tiene muy bien detallada la teoría, a la vez que muestra código de estos motores de entidades.

2.3.1.2. Otros motores ECS

A pesar de que el objetivo principal del proyecto no sea el motor (si no la guía de cómo crearlo), hablaremos de los motores existentes en internet similares al que vamos a crear. En este apartado trataremos con dos motores de entidades, EnTT y ECS, el primero de Michele Caini y el segundo de Sam Bloomberg.

El motor EnTT de Caini (2017) es un motor ECS de un solo archivo cabecera (.h) escrito en C++ de no gran tamaño y de código abierto. Es fácil de usar a la hora de programar videojuegos, un ejemplo de uno de ellos es el videojuego de Mojang "Minecraft".

El denominado ECS de Bloomberg (2016), es otro ECS simple escrito en C++. Usa C++11 por lo que para usarlo necesitarás esa versión. Este motor sirve para un prototipado rápido o como punto de partida para crear otros ECS más avanzados, u otras herramientas. Además no está optimizado para ser veloz.

2.3.1.3. Libro "Data-Oriented Design" por Richard Fabian

Este libro de Fabian (2018) habla sobre el diseño orientado a datos. El diseño orientado a datos está inspirado en técnicas informáticas de alto rendimiento, diseño de bases de datos y valores de programación funcional. Proporciona una metodología práctica que reduce la complejidad al tiempo que mejora el rendimiento tanto de su equipo de desarrollo como de su producto.

Te hace comprender el objetivo, los datos, el hardware, etc, y por eso lo pongo como proyectos similares ya hechos, porque aunque no sea nuestro objetivo, este libro contiene muchos de los problemas de datos y memoria que vamos a tratar a lo largo del desarrollo, y brinda instrucción sobre los procesos de pensamiento involucrados al considerar los datos como el detalle principal de cualquier proyecto.

Además, aunque no sea una guía, es un documento escrito para ayudar a entender y aprender acerca de estos temas, por lo tanto lo hace similar a mi proyecto.

2.3.2. Conclusión: ¿Qué hay?¿Qué me gustaría hacer?

En general, en el primer apartado de esta sección los tres posts son demasiado cortos y no lo suficientemente guiados como a mi me gustaría. No son proyectos completos.

Si nos fijamos en los motores creados, EnTT es simple pero muy útil, ya que tiene numerosas funcionalidades y se puede crear todo tipo de componentes y sistemas. Sería el más completo entre todos los propuestos. A continuación, como hemos mencionado anteriormente, el motor "ECS" es lento, sirve para hacer prototipos rápidos y además esta programado en C++11 y hay que compilarlo con esta versión obligatoriamente. El libro de Richard Fabian, "Data-Object Design", es un libro completo sobre datos y estructura de datos, lo cual nos servirá también como referencia a la hora de consultar algo sólido a la hora de explicar cualquier tema.

En conclusión, visualizando el contenido actual, la guía propuesta será más explicativa que las pequeñas guías existentes, además de contemplar diversos proyectos pequeños, los cuales aumentarán de dificultad a medida que avancemos. Por último el motor a realizar en esta guía será lo más completo posible, teniendo en cuenta a motores anteriores como "EnTT" y "ECS" entre otros, que construiremos desde cero.

2.4. Análisis de las tecnologías

Aunque el objetivo principal del proyecto no sea el motor ECS, si no la guía de como realizarlo, tendremos que dejar claras cuales son las tecnologías que utilizaremos para desarrollar el motor, ya que la guía estará definida por éstas.

Tendremos en cuenta las diferentes tecnologías disponibles. En primer lugar hablaremos de los lenguajes de programación más utilizados en el mundo de los videojuegos, C++, Java, CSharp y Phyton entre otros. También expondremos cuál de ellos nos conviene utilizar más y la elección final del lenguaje.

Al final del apartado hablaremos de motores gráficos, ya que en esta guía lo que se pretende es crear un motor de entidades, no el motor gráfico del juego. Hablaremos de los diferentes motores que tenemos a mano para usar, como la librería TinyPTC, la interfaz SFML, OpenGL y RayLib, y realizaremos al igual que con el lenguaje de programación, la elección del que más nos conviene utilizar.

2.4.1. Lenguajes de programación

Los lenguajes de programación son ”la herramienta” que los programadores utilizan para dar forma a un programa, a las mecánicas de un videojuego, etc. Es decir, si nos referimos a videojuegos, es la maquinaria que hace posible que funcione de una forma determinada.

Cada lenguaje tiene diferentes secuencias, o código, y cada uno se emplea por ciertos motivos. Cada lenguaje es más óptimo para realizar ciertas tareas y tendremos que decidir cuál nos permite conseguir nuestros objetivos de la forma más eficiente y correcta.

2.4.1.1. C++

Este lenguaje de programación es uno de los más utilizados en el sector por profesionales. Es un lenguaje popular en los títulos AAA, se utiliza en videojuegos para PlayStation y Xbox, y en juegos independientes. Se trata del lenguaje más compatible con la mayoría de los motores de juego y tiene un tiempo de ejecución bastante rápido. Por otro lado, permite a los desarrolladores tener un control amplio sobre el hardware, la gestión de la memoria y los gráficos, y aunque al principio puede resultar complejo de utilizar, una vez te haces a él, podrás manejar cualquier otro lenguaje.

Pero si decidimos usar este lenguaje, tenemos que realizar otra elección, elegir el estándar que utilizaremos. El más utilizado hasta la fecha, es C++17, ya que muchas empresas y videojuegos la empezaron a utilizar hace años. Ésta tuvo mejoras respecto a la versión anterior, C++14, incorporando nuevas reglas para la deducción de auto, std::invoke, funciones plegables(folded expressions), etc.

Posteriormente en el año 2020 aparece C++20 que mejora el estándar anterior considerablemente añadiendo nuevas especificaciones de tipos y clases aceptadas en plantillas, alternativas a ”include” , etc. Es muy útil la mejora en las ”template” o plantillas ya que en un videojuego es algo que tener muy en cuenta y puede llegar a optimizar mucho nuestros programas.

Por último, el estándar C++23 que está por llegar en el año 2023 y aportará mejoras al lenguaje como por ejemplo bibliotecas de soporte para corrutinas, metaclases, una alternativa superadora para assert, ”executors” que permiten ejecutar código en diversos lugares (CPU,

GPU), etc.

Todas estas novedades tendremos que tenerlas en cuenta para decidir desde que versión partir.

2.4.1.2. C Sharp

En entornos Windows es uno de los más populares. Es un lenguaje mucho menos flexible que C++, pero por ejemplo, en Unity se puede programar con él, lo que lo hace atractivo. Es un lenguaje fácil de aprender, mucho más que C++ y es una opción para principiantes en el mundo de la programación y los videojuegos.

Las principales ventajas que presenta el uso C Sharp es su potencia como lenguaje. Soporta la mayoría de paradigmas, destacando el paradigma funcional que combinado con el orientado a objetos hacen del lenguaje uno de los más potentes.

La principal desventaja que puede presentar respecto a Java por ejemplo, radica en su portabilidad y en la dificultad que trae un desarrollo completo de un producto software empleando este lenguaje. En el mundo de los videojuegos, como hemos mencionado anteriormente, donde más se utiliza es en Unity.

2.4.1.3. Phyton

Phyton no es un lenguaje exclusivo en la creación de videojuegos, más bien es utilizado para hacer prototipos rápidos ya que permite plasmar ideas en pocas líneas de código. Tiene una ejecución mucho más simple que la de otros lenguajes. Dispone de un framework llamado Pygame, desarrollado exclusivamente para crear videojuegos, o prototipos como hemos mencionado anteriormente. Además funciona en prácticamente todas las plataformas y sistemas operativos.

La desventaja a destacar es que tiene un consumo de memoria alto que se debe a la flexibilidad de los tipos de datos, lo que lo hace más lento.

2.4.2. Elección del lenguaje

Como en el título del proyecto se indica, el lenguaje utilizado será C++.

El motivo principal es que es el lenguaje que más conozco entre todos los propuestos y

además permite tratar la memoria de forma sencilla. He realizado proyectos anteriores con C++, y en concreto con varias versiones de éste (la 17 y la 20), pero mi elección para esta guía es la versión 20, ya que obtuvo una mejora notable en las ”templates” entre otras cosas.

Además C++ es el lenguaje más usado en el mundo de los videojuegos y por lo tanto lo será en esta guía, ya que también me servirá como portfolio.

2.4.3. Motores gráficos

Aunque en este proyecto no vayamos a realizar una guía de como crear un motor gráfico, ni vayamos a hacer uno propio, tenemos que tener en cuenta que es un motor gráfico a grandes rasgos ya que el objetivo es crear un motor ECS e iremos viendo resultados de forma gráfica usando estos motores. Así que usaremos uno existente en nuestro proyecto. En la siguiente sección comentaremos por encima unos cuantos. Pero antes de avanzar a la siguiente sección, ¿qué es un motor gráfico y para qué sirve?.

Es el software capaz de dibujar en la pantalla de nuestro ordenador. Tambien podemos decir que se define como motor gráfico el framework de software diseñado para crear y desarrollar videojuegos. Un buen motor gráfico es el que traslada tus ideas creativas a tu pantalla.

2.4.3.1. TinyPTC

TinyPTC es una librería framebuffer creada para dibujar gráficos en una ventana que se puede crear de forma muy sencilla. La mayor de sus ventajas podría ser la claridad para dibujar ya que solo recorriendo los píxeles de la pantalla y llamando a su función update(), podemos dibujar ya sencillas formas.

Las desventajas parten de que no es una herramienta muy potente. Solo sirve para gráficos en dos dimensiones y requiere de conocimiento para pintar, ya que los colores en hexadecimal no todo el mundo los comprende, y cómo recorrer para dibujar diferentes elementos, puede ser costoso. Aun así no son grandes elementos a conocer, ya que con un vistazo o un ejemplo, podemos usarla decentemente.

2.4.3.2. SFML

SFML (Simple and Multimedia Library) es una API escrita en C++ con un enfoque orientado a objetos para el desarrollo de aplicaciones interactivas enfocada en el desarrollo de juegos 2D, y 3D.

Está compuesta en cinco módulos. El system que es donde se manejan los threads, tiempo, vectores... .El Window que es el módulo que te permite manejar la ventana de la aplicación. Graphics, que es utilizado para pintar sprites, texturas y demás objetos gráficos en una ventana como canvas. El de audio, manejador de audio. Y por último el Network, permitiendo conexiones con sockets y crear aplicaciones en red.

2.4.3.3. OpenGL

OpenGL (Open Graphics Library) es una especificación de una API para escribir aplicaciones multiplataforma que produzcan gráficos 2D y 3D. La interfaz de OpenGL tiene más de 250 funciones diferentes que se usan para dibujar escenas a partir de primitivas como puntos, líneas... Es una especificación y esto quiere decir que trata de un documento que describe un conjunto de funciones y el comportamiento que deben tener.

Estas librerías GLEW (OpenGL Extension Wrangeler Library), GLUT (OpenGL Utility Toolkit) y GLFW (Graphics Library Framework) son las que se utilizan para que OpenGL funcione y donde se recogen funcionalidades, carga de extensiones, e interacción de ventanas y la creación de un contexto OpenGL.

Es veloz, multiplataforma, tiene soporte para usarse en ámbito web (WebGL), compatible con diferentes Sistemas Operativos y ofrece una gran escalabilidad. Sus mayores ventajas es que reduce la sobrecarga de CPU asociada con la renderización y es más rápida que las de otras API, permite gran calidad visual de escenas texturizadas y tiene una interfaz estable.

2.4.3.4. RayLib

Según la página web de RayLib, esta es una librería para disfrutar de la programación de videojuegos. No dispone de ayudas visuales, ni interfáz gráfica. Dispone de una hoja con todas las funciones y su utilidad, lo cual es muy importante si el objetivo no recae en lo gráfico y se requiere de poco tiempo para esto.

Esta librería ha recibido diferentes premios de Epic Games y Google entre otros. Además se basa en OpenGL para diferentes funcionalidades, lo cual es interesante a tener en cuenta, ya que es más sencilla de utilizar y con una curva de aprendizaje menor y llevadera. Se puede usar tanto para hacer prototipos en 2D y 3D.

2.4.4. Elección del motor gráfico

El motor gráfico a utilizar en esta guía será RayLib, ya que es una librería ya creada la cual no tenemos que implementar, si no que tendremos que usar las funcionalidades que más nos gusten y nos sean útiles para nuestro propósito. En esto tenemos una gran ventaja al usar RayLib ya que dispone de un manual con todas las funciones separadas por funcionalidades y con la explicación de cada una de ellas.

En mi caso al querer hacer un motor ECS, y no dedicar mucho tiempo al aspecto gráfico de los diferentes proyectos, es de gran utilidad una librería fácil de entender y rápida de usar.

He usado tanto OpenGL como SFML, pero no TyniPTC o RayLib, pero el último me genera confianza ya que en ejemplos no le veo grandes dificultades de uso. Además, como mi objetivo es crear juegos en 2 dimensiones, SFML, RayLIB o TyniPTC, son librerías adecuadas para este propósito, pero usaré RayLIB en esta ocasión.

3. Objetivos

Este proyecto tiene unos objetivos claros y concisos que se exponen a continuación.

- Hacer comprender la arquitectura de software Entity Component System a los lectores.
- Una guía práctica intercalando apartados de teoría y programación.
- Desarrollar diferentes motores ECS durante la guía, iterando en su contenido y dificultad.

El primero de los objetivos reside en comprender la arquitectura Entity Component System. Cuenta con muchos detalles que requieren de unos apartados previos de introducción y toma de contacto antes de comenzar a desarrollar motores y videojuegos poniéndolo en práctica. Para esto, expondremos de forma teórica los conceptos y estructuras para lograr entenderlos antes de empezar con el desarrollo.

Esto me da pie a hablar del segundo objetivo del proyecto, que como ya he mencionado es una guía, por lo tanto, un objetivo principal es intercalar apartados de desarrollo con otros de teoría, buscando hacer un libro más entretenido y con diferentes explicaciones para entender los contenidos.

La guía estará compuesta por una parte principal teórica y una parte secundaria que estará formada por diferentes proyectos, aunque intercalados con pequeños apartados teóricos cuando lo requieran. Esto crea el objetivo de aprender realizando proyectos.

Otro de los objetivos principales será suministrar contenido en pequeñas dosis. Esto quiere decir que conforme avancemos en la guía y en sus proyectos, iremos aumentando los contenidos y dificultad de los motores y juegos que realicemos.

Finalmente aunque el proyecto trata de la creación de esta guía, dentro de ella habrá diferentes subproyectos como ya se ha mencionado, los cuales tienen como objetivo enseñar e ir mostrando un ECS desde principio a fin. Los objetivos principales de estos subproyectos se muestran en el siguiente apartado.

3.1. Objetivos específicos de los subproyectos

En la guía hay 5 subproyectos: ”Starfield”, ”Firefighter game”, ”Firefighter game (RayLib)”, ”Save the OVNI” y por último, ”The Final Room”.

3.1.1. Primer proyecto : Starfield

- Comprender la estructura ECS.
- Intentar hacer un prototipo inicial, de la forma más sencilla posible, para iterar sobre él.

Con este proyecto se busca conseguir una toma de contacto entre la guía y el lector. Para ello partimos de una teoría inicial muy básica. El objetivo es conseguir entender la estructura ECS y aplicarla a un proyecto sencillo, que será la base para los demás subproyectos.

3.1.2. Segundo proyecto : Firefighter Game

- Aplicar esta arquitectura con diferentes componentes, para observar cambios y formas de proceder.
- Al tener más componentes, necesitamos nuevos sistemas. Aprender a decidir para qué crear sistemas y componentes.

Con este pequeño juego, aumentamos el número y dificultad de los sistemas, aunque todavía será muy sencillo. El objetivo es seguir entendiendo la estructura con más contenido y comenzar a indagar en las Entidades, sin buscar optimización pero sí diferenciación entre componentes. Este es un primer paso, que mejora el subproyecto anterior.

3.1.3. Tercer proyecto : Firefighter Game (RayLib)

- Pasamos de hacer juegos en la terminal del sistema imprimiendo por pantalla, a usar RayLib y tener ventana gráfica. El objetivo base es observar cómo al empezar a usar esta librería, esto afecta a como se crea el componente de renderizado y algunos detalles más, pero no al motor que hemos creado.

- Introducir RayLib, ya que la usaremos de aquí en adelante. De esta forma, no tendremos que parar a explicar temas relacionados con la librería en subproyectos futuros.

Este proyecto es intermedio entre el anterior y el siguiente. El objetivo principal es entender la librería de dibujado y demostrar que con el mismo motor que hemos creado para el apartado anterior podemos simplemente cambiar componentes y sistemas y tener el mismo juego de diferentes formas. Como ya he mencionado, nos centramos en la parte de dibujado para que en futuros proyectos no haya que parar específicamente en cómo se dibuja, ya que se hará de forma similar.

3.1.4. Cuarto proyecto : Save the OVNI

- Aumentamos la dificultad, creamos sistemas nuevos más complejos.
- Mejoramos nuestro manejador de entidades, eliminando entidades cuando se requiere.
- Estados, observamos cómo manejar el flujo de entidades para cada estado.
- Entendemos que el orden importa para nuestros sistemas.

Este proyecto aumenta considerablemente la dificultad respecto a los anteriores, pues ya tiene sistemas y entidades distintas, que hacen tareas distintas. Es un primer paso hacia motores ECS más elaborados. El objetivo es seguir indagando en la forma de crear las entidades, evolucionando del proyecto anterior, a un paso por encima y analizar los sistemas y el orden de éstos para conseguir un buen resultado.

3.1.5. Quinto proyecto : Final Room

- Entendemos cómo se usan los Slotmaps y cómo se programan.
- Las entidades pasan a ser máscaras y llaves, ya no tienen componentes en su interior.
- Creamos un gestor de componentes que nos ayudará a ordenar nuestro código, y se exponen opciones para mejorar.
- Creamos un sistema de estados más complejo, teniendo en cuenta las entidades de las que disponemos y cómo manipularlas.

- Aparecen las ”Tags” que son etiquetas para las entidades, que nos ayudan a elegirlas o diferenciarlas de forma más sencilla.

- Comprendemos en un proyecto final de guía todos los conceptos aprendidos juntos.

Este juego tiene como objetivo principal separar los componentes de las Entidades, optimizando la memoria. Tendremos que desarrollar estructuras propias, los Slotmaps, para conseguir una optimización mayor. En este proyecto final de guía se busca optimizar al máximo sin dar un salto muy brusco de contenido para el lector, y dejar una estructura final para el motor ECS fácil de entender, utilizar y reutilizar.

3.2. Futuros objetivos y proyectos

Aunque he realizado cinco subproyectos para esta guía, la idea es en un futuro hacer una segunda parte donde poder seguir observando el avance de estos motores usando la arquitectura de software Entity Component System para lograr mayores proyectos y resultados, buscando todavía más optimización.

Es un proyecto muy largo, por lo que me veo obligado a hacerlo en dos tomos. El siguiente tomo incluirá plantillas, metaprogramación y recursos de optimización mayores, pero de la misma forma que he hecho este primer libro, paso a paso y proyecto a proyecto.

El objetivo final de la primera guía ha sido comprender, describir y mejorar la estructura del ECS, y finalmente optimizarla para lograr mejores resultados de velocidad, flexibilidad y optimización.

Una vez terminada la guía, será ofrecida a los futuros alumnos de cuarto curso de Ingeniería Multimedia, con el objetivo de recibir más feedback para mejorarla y que les sirva de ayuda para llevar a cabo una asignatura extensa y con muchos conceptos, que en ese momento son nuevos para todos ellos.

4. Metodología

Para gestionar este proyecto hacen falta unas pautas o una metodología. Necesitaremos unas reglas para que el producto evolucione con la máxima calidad posible.

Podemos definir una metodología como el conjunto de procesos, herramientas y técnicas que se utilizan para planificar, ejecutar y controlar un proyecto con el objetivo de alcanzar sus metas de manera eficiente. La elección de una metodología adecuada es esencial para el éxito de un proyecto, ya que puede afectar en gran medida al rendimiento, al coste y al plazo del proyecto.

Existen numerosas metodologías de gestión de proyectos, cada una con sus propias ventajas e inconvenientes. Algunas metodologías, como la cascada, se centran en la planificación y ejecución de un proyecto de manera secuencial, mientras que otras, como Scrum, se basan en el enfoque iterativo e incremental. La elección de una metodología dependerá de diversos factores, como el tamaño y complejidad del proyecto, los recursos disponibles, el equipo de proyecto y las preferencias del cliente.

4.1. Metodologías Analizadas

A continuación expondremos algunas de las diferentes metodologías existentes más famosas y elegiremos finalmente la que mejor se adapte a nuestro proyecto, teniendo en cuenta de dónde partimos y lo que queremos lograr.

4.1.1. Cascada

Es una metodología secuencial y lineal que se centra en la planificación detallada del proyecto desde el principio hasta el final. Las diferentes fases del proyecto se completan antes de pasar a la siguiente. Esta metodología es adecuada para proyectos con requisitos bien definidos y pocos cambios a lo largo del ciclo de vida del proyecto.

Tienen que estar establecidos los requisitos y prever todas las preguntas pertinentes antes de

comenzar el desarrollo. Los requisitos deben ser lo más completos posibles porque el equipo trabaja basándose en este diseño inicial.

En la figura 4.1 se aprecia de forma visual este modelo, dando a comprender que no avanzamos hasta el siguiente paso antes de haber completado el anterior.

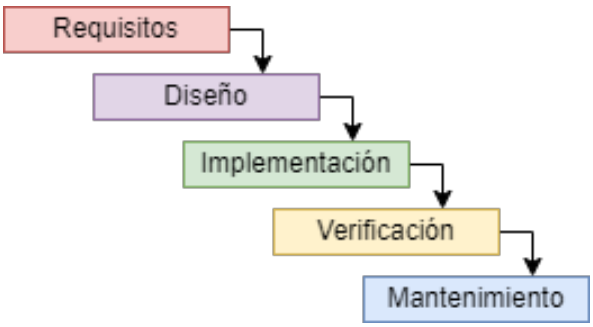


Figura 4.1: Esquema del "modelo cascada"

4.1.2. Scrum

Es una metodología ágil que se centra en el trabajo en equipo, la comunicación continua y la adaptación a los cambios. La metodología Scrum se basa en ciclos de trabajo cortos, llamados sprints, en los que se entregan incrementos funcionales del producto. Esta metodología es adecuada para proyectos con requisitos cambiantes y que requieren una mayor flexibilidad y adaptabilidad.

El método Scrum tiene un proceso de planificación detallado, en el que se seleccionan las tareas a realizar en el siguiente sprint y se estiman las horas necesarias para completarlas.

Es una de las más usadas ya que puede ser muy adaptable, entregas un valor por tu producto desde el principio, presentando resultados. Estimula el trabajo en equipo y consigue una mejora continua a través de las reuniones de retrospectiva, en las que el equipo reflexiona sobre lo que funcionó bien y lo que no funcionó durante el sprint anterior, y cómo pueden mejorar en el siguiente sprint. Esto ayuda a mantener el equipo enfocado en la mejora constante y a evitar la complacencia.

4.1.3. Kanban

Es una metodología visual que se utiliza para gestionar y optimizar el flujo de trabajo. Esta metodología se basa en la visualización del proceso de trabajo en un tablero Kanban, en el que se muestran las tareas y los estados de cada tarea. Esta metodología es adecuada para

proyectos donde se requiere una mayor transparencia en el proceso de trabajo.

No tiene ciclos de trabajo definidos si no que utiliza un flujo continuo para entregar el valor al cliente. Además, Kanban no posee roles definidos lo que permite a los equipos organizarse según sus necesidades.

4.1.4. Método basado en prototipos

Esta metodología de desarrollo se basa en la idea de que la mejor manera de entender y especificar los requisitos del software es a través de la construcción de prototipos que se puedan probar y mejorar antes de crear la versión final.

El método basado en prototipos tiene varias ventajas, como la capacidad de obtener rápidamente comentarios de los usuarios finales, la reducción de costos al detectar errores y problemas en una etapa temprana del proceso de desarrollo, y la mejora de la calidad del software al permitir la retroalimentación continua. Sin embargo, también tiene algunas limitaciones, como el riesgo de crear prototipos poco representativos de la versión final y la posibilidad de que los desarrolladores se centren demasiado en la construcción del prototipo en lugar de en la construcción del software final.

4.2. Metodología a emplear

Las metodologías anteriores son las principales y las que predominan en el mundo de la ingeniería de software, por lo tanto basaré mi elección en ellas pero no por ello será una de éstas.

La metodología a seguir será una metodología híbrida iterativa. Esta decisión se debe a que cada explicación y novedad de la guía, estarán reflejadas en proyectos distintos entre sí pero con un motor en común, por lo tanto veremos una similitud con la metodología por prototipos.

Cada iteración durará un tiempo variable pero de aproximadamente tres semanas o un mes. En ese momento habrá una reunión con el tutor en el que expondré mi prototipo, qué quiero hacer para la siguiente iteración y qué novedades tendrá ese nuevo proyecto. Entonces, después del feedback conseguiremos la versión final de ese prototipo y nos centraremos en el siguiente proyecto para seguir con la guía. Esto genera una similitud con la metodología Scrum, posicionando al tutor del proyecto como Product Owner y sprints de reunión a reunión.

Es por eso que no podemos encasillar la metodología seguida en una u otra, si no que he escogido los elementos más importantes y que más se adaptan a mi proyecto de algunas de ellas. Finalmente la llamaremos iterativa, ya que nos basamos en iteraciones, y cada iteración elimino, modifico o añado nuevas partes al proyecto.

En cada iteración tendré un prototipo resultado, y de esta forma se irá componiendo la guía a lo largo del tiempo.

5. Desarrollo

Este apartado trata sobre el proceso de creación del proyecto, desde el principio, hasta el final. Describiré cómo he realizado cada una de las partes, por iteraciones. En ellas se aclararán los objetivos principales de la iteración y el proceso para conseguirlos.

Podemos comenzar a desarrollar esta sección tras haber realizado un trabajo previo para exponer y analizar. Es un apartado que se verá modificado y aumentado cada vez que se trabaje o investigue sobre el proyecto.

El final de cada sesión de trabajo lo dedicaré a redactar todo lo acontecido y avances. Estos avances e iteraciones se ven encuadrados en la siguiente tabla (5.1).

	Título	Referencia
Iteración 0	El comienzo	5.1
Iteración 1	Contacto teórico	5.2
Iteración 2	Nace el esqueleto	5.3
Iteración 3	Retoques teóricos y estructurales	5.4
Iteración 4	Primeros pasos con ejemplos de código	5.5
Iteración 5	Proyecto con diferenciación de componentes	5.6
Iteración 6	Reorganización e introducción a RayLib	5.7
Iteración 7	Memoria y elección de tecnología	5.8
Iteración 8	Último proyecto y detalles finales	5.9
Iteración 9	Resultado final	5.10

Tabla 5.1: Tabla de iteraciones

Además, aparecerán diferentes bocetos y diagramas hechos a mano como muestra y representación de lo que explico posteriormente, que son los que he necesitado para estructurar mis ideas. Estos bocetos se pueden encontrar en un estado pulido y también en el libro "Cómo desarrollar un GameEngine Entity-Component-System en C++20" junto a la explicación pertinente en cada caso.

5.1. Iteración 0 - El comienzo

Esta es la primera de las iteraciones. En este punto del proyecto aún no hay nada sobre lo que trabajar. En la iteración 0 mis objetivos principales han sido analizar el contenido anterior a mi proyecto y analizar los vídeos del canal de YouTube de Francisco Gallego, Retroman (2014), para así tener una base sobre la que empezar, saber a qué me enfrento y tener una estructura más clara del proyecto.

En primer lugar, en un documento de texto, apunté cuales de estos vídeos me resultan importantes para mi proyecto, para así tener un acceso directo a éstos en el momento de llegar a necesitarlos. Muchos de estos vídeos pertenecen a fragmentos de clases académicas a las que he asistido y por lo tanto conozco el contenido de los vídeos a grandes rasgos.

Como medida, decidí hacer un diagrama muy sencillo de un motor ECS, donde tener ciertas partes y estructuras diferenciadas, para así saber en qué parte colocar cada vídeo. Es decir, si hablo de Entidades, tendré asociados varios vídeos que traten de esto.

El diagrama es el de la figura 5.1:

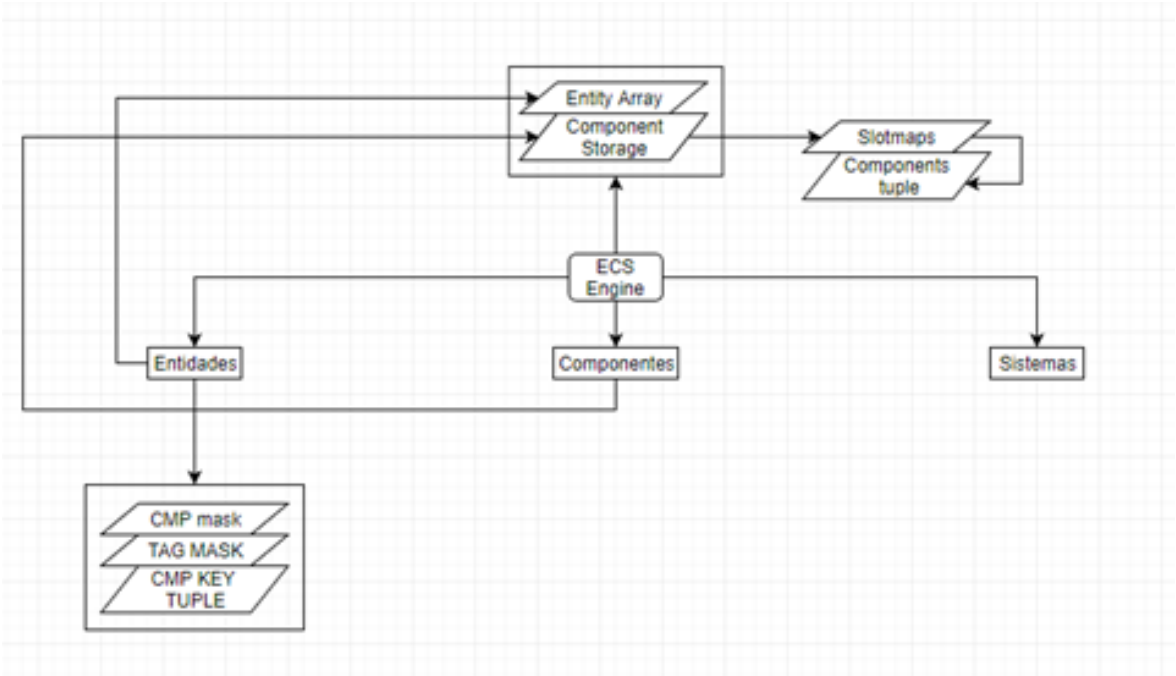


Figura 5.1: Esquema básico del ECS.

Para esta selección recorrí todo el catálogo de vídeos recordando cuáles iban primero cronológicamente, porque como ya he comentado yo he asistido a estas clases. De cada vídeo que he considerado útil para mi proyecto he estudiado su contenido en profundidad, ya que

muchos de ellos tratan más temas de los que se muestran en el título.

Además con este conjunto de vídeos, podía empezar a hacerme la idea de la posible estructura del proyecto. En la siguiente figura 5.2, se puede apreciar una pequeña parte de lo que fue al principio esta selección.

```
VIDEOS INTERESANTES ····································································

.ECS SOBRE IRRLICHT:                    https://www.youtube.com/watch?v=yYBsPNT9UBo&t=16s
·HEAP ARRAY VECTOR (conceptos) ECS(pp*): https://www.youtube.com/watch?v=c9Tow7Wx7qI
·TAG DISPATCHING:                       https://www.youtube.com/watch?v=JtAYbpExmgo&t=11s

·TUPLAS DE CMPS:                        https://www.youtube.com/watch?v=vKf6zL48bvQ&t=227s
·COMPONENTES, MASCARAS, IDs, CS*:       https://www.youtube.com/watch?v=sJ_clEW1Eok
·IMGUI 1:                               https://www.youtube.com/watch?v=cyPPDtrDTcY&t=41s
·IMGUI 2:                               https://www.youtube.com/watch?v=uTskEwkv7Bo
·SLOTMAP TEORIA                         https://www.youtube.com/watch?v=GKfRDAUvFoE
·SLOTMAP PROGRAMACIÓN                   https://www.youtube.com/watch?v=GKfRDAUvFoE

·TEMPLATES EXPLICACIÓN:                 https://www.youtube.com/watch?v=5CW3wHEeSuw
·TEMPLATES PROG:                        https://www.youtube.com/watch?v=qwgEmuHEamI&t=1s
·METAPROGRAMING:                        https://www.youtube.com/watch?v=4NKbmCfZ9OI
```

Figura 5.2: Primeros contenidos ordenados.

Por otro lado en esta iteración principal, consulté numerosas fuentes teóricas acerca de qué es un ECS, para tener los conceptos más claros y contrastados. Ya tenía un conocimiento previo sobre esta temática, ya que dediqué gran parte del cuarto curso académico a desarrollar un ECS propio, y son esos los conocimientos que vengo a ofrecer en mi guía.

Al final de esta iteración principal, tenía los vídeos importantes preparados en el esquema de vídeos anterior, y el conocimiento necesario acerca del proyecto que iba a realizar. Este fue el final de la iteración 0.

5.2. Iteración 1 - Contacto teórico

En la iteración 1, los objetivos aumentaron, dando paso a escribir apartados principales del proyecto, como por ejemplo el marco teórico, el estado del arte, el análisis de tecnología, e incluso a comenzar con el producto final.

Fue en esta iteración también cuando comencé a escribir este apartado, el desarrollo del proyecto, al darme cuenta de que cada cosa que escribía o analizaba, era importante de conocer.

Lo primero que realicé, fue escribir el marco teórico, en un documento, para ver toda la información de la que disponía y conocía, por lo tanto tras analizar diferentes fuentes que

hablaban sobre el tema, decidí escribir yo acerca del ECS.

Compuse un marco teórico de 4 apartados, donde se expone una pequeña introducción, el por qué de estructurar la memoria, sobre lo que realmente es un ECS y sus partes, y por último sobre la estructura de datos que voy a utilizar, los Slotmaps. Esto lo hice apoyándome de contenido existente, para saber cómo expresar o ver cómo otras personas expresaban la información.

Seguidamente, para hablar sobre los Slotmaps, repasé el vídeo detalladamente de los Slotmaps del canal de YouTube de Profesor Retroman, y a partir de esto, dibuje en papel un pequeño diagrama donde se ve clara la composición de una de estas estructuras.

Ese diagrama es el de la figura 5.3.

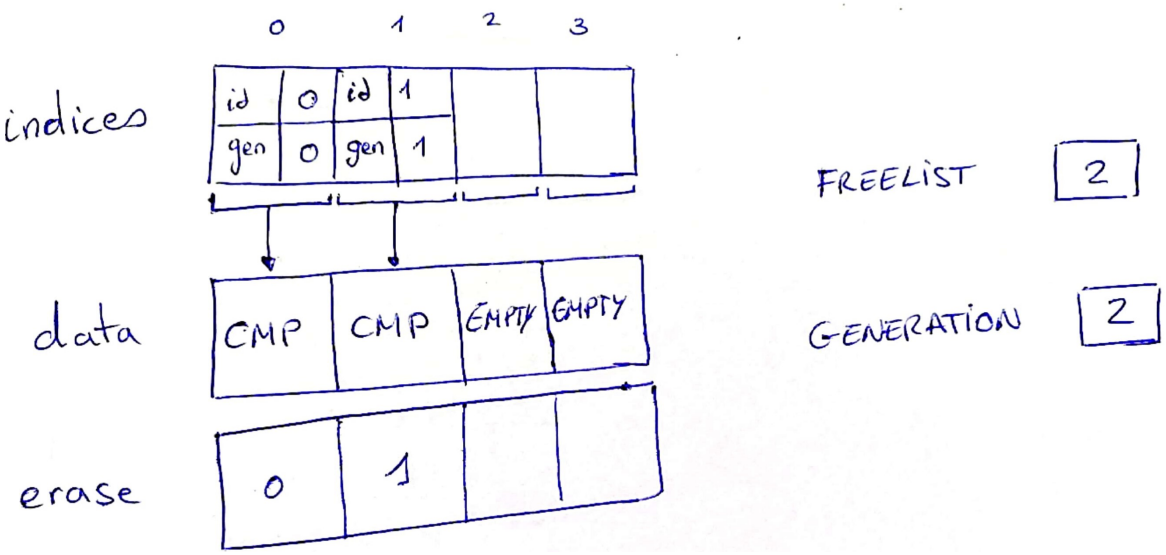


Figura 5.3: Estructura de un slotmap.

A continuación, en el apartado de estado del arte, realicé el mismo flujo de trabajo. En primer lugar, exploré trabajos anteriores que traten sobre el mismo tema que el mío, pero no tuve mucho éxito, ya que no encontré guías claras y adecuadas de cómo realizar un motor ECS.

Decidí hablar acerca de tres páginas que encontré, que detallan lo que es un ECS, pero la implementación de estos es simple y no tan legible como una guía. Aun así, leí los documentos para ver de que trataban y los expuse de forma breve en este apartado.

Otro apartado que he redactado ha sido sobre otros motores ya existentes, porque si bien no encontré proyectos de guías anteriores, sí que encontré motores anteriores, los cuales analicé indagando en sus carpetas, documentación y código para así poder exponerlos también como

arte anterior a mi proyecto.

Por último, en este apartado escribí acerca del libro ”data-oriented design”, que me ayudará en el futuro a entender ciertos pasos a la hora de estructurar los datos. Analicé su índice y apartados más importantes y más parecidos a lo que yo voy a tratar en mi proyecto.

El siguiente tema que traté en esta iteración fue el análisis de tecnologías. En primer lugar, separé este apartado en dos bloques, el lenguaje que voy a utilizar y el motor gráfico que usaré ya que no voy a desarrollar uno propio en esta guía. Busqué los mejores lenguajes de programación actuales y más usados en el desarrollo de videojuegos, y hablé de estos lenguajes, fijándome en sus ventajas y desventajas, y así escribir una conclusión sobre cuál usar. Aunque ya lo tenía elegido desde el principio, explico el por qué de esta decisión.

Realicé el mismo proceso acerca de los motores gráficos que podemos encontrar ya creados, elegí uno de ellos basándome en mi experiencia y alguna de sus ventajas y desventajas. Además decidí hacer una guía de uso corta de este motor también con las funciones que utilizaré en mi guía. Al principio iba a incluirlo como anexo, pero finalmente decidí que haría un capítulo introductorio a este motor gráfico.

Así es como termina la iteración 1, con los principales apartados del proyecto escritos, aunque habrá que profundizar más y redactar mejor, es el comienzo y un gran paso en el proyecto.

5.3. Iteración 2 - Nace el esqueleto

Al principio de la iteración comenzó el paso a LaTeXde toda la documentación ya realizada en editores de texto, ya que el contenido iba a resultar más ordenado y editable a lo largo del desarrollo.

Antes de este paso, tuve que aprender a usar bien la plantilla del grado para TFG que había disponible en GitHub y configurarla de forma correcta para su funcionamiento. Fue difícil ya que tuve que resolver errores que eran problema de la licencia gratuita del editor Overleaf de la cual disponía.

Una vez tuve la plantilla y el editor en orden, comencé a transcribir a limpio partes teóricas que tenía escritas en sucio, intentando darle un nivel estructural, para así a su vez, comenzar a tener un índice. Redacté de mejor forma el estado del arte completo junto con el análisis y decidí las tecnologías con las que iba a trabajar durante el desarrollo.

Finalmente en cuanto a teoría respecta, separé el proyecto en dos partes. Una será la memoria del trabajo y la otra la guía de como desarrollar el motor ECS que será el resultado principal del proyecto. En este punto, dispongo de dos diferenciados documentos, por lo que la teoría

más pura de mi proyecto se encuentra en la parte del ”libro” a desarrollar y en la memoria decidí hacer referencia a esta teoría, para no escribir dos veces el mismo contenido. Me puse a pensar en el libro y en su estructura, así que basándome en mis esquemas e índices anteriores, escribí un primer índice para este documento.

Antes de la separación, estructuré el contenido en los apartados principales de un TFG los cuales la memoria incluía para así obtener la base del índice de la memoria. Además de seguir investigando acerca de la teoría y tecnologías para un mayor grado de perfección a la hora de incorporar los contenidos al proyecto. Algunas de las novedades en este punto son por ejemplo la división del apartado análisis para diferenciar las tecnologías actuales, hablar de todas un poco y elegir una final. Primero elegí las más usadas. Seguidamente tuve en cuenta las ventajas y desventajas de éstas y en base a esto y a mi objetivo, elegí.

5.4. Iteración 3 - Retoques teóricos y estructurales

En esta iteración, el objetivo ha sido principalmente, añadir nuevo contenido teórico a la memoria del proyecto y producto final.

En primer lugar, disponía de una teoría ya avanzada en el libro, es decir, empezaba hablando directamente de Entity Component System. He introducido un primer apartado hablando del mundo de los videojuegos previamente, el mundo del desarrollo de videojuegos, motores, etc. A su vez he cambiado la estructura principal del libro debido a estos cambios al inicio del proyecto. El objetivo de esta parte previa a la teoría de mi proyecto, es dejar claro lo que voy a hacer, para que todo lector sepa desde dónde partimos y a dónde queremos llegar.

En cuanto a la memoria, se ha añadido una sección de marco teórico en formato resumen. Este formato resumen proviene del libro, para no tener el mismo contenido dos veces. Además he incluido un pequeño comentario donde remarco esto, y redirijo al libro para tener en cuenta toda esa teoría en su plenitud. Además de esto, en el análisis de la tecnología también ha habido modificaciones. Ahora hay otras tecnologías de gráficos como RayLib y he llegado a la conclusión de que esta será la que voy a utilizar para el proyecto. El cambio se debe a que es más simple que Irrlicht Engine, que era la tecnología a utilizar hasta este punto, y para mi propósito es mucho más eficiente.

Respecto a la estructura del proyecto, también ha habido cambios. El estado del arte y análisis ahora han pasado a formar parte del Marco teórico, ya que teniendo en cuenta su contenido, veo más lógico que estén situados en este lugar.

Una vez tenía toda la nueva teoría escrita en ambos documentos, me dispuse a cambiar todas las referencias que había escrito hasta ahora, ya que las tenía escritas de forma manual, y el objetivo es automatizarlas para que al ser pulsadas, redirijan a la fuente original. Además de

esto he estado mirando el programa JabRef, que es un lector de bibliografías el cual todavía no he utilizado por su desconocimiento, pero que posteriormente usaré. No solo se han cambiado referencias a otras fuentes, si no también referencias a figuras.

Otro avance a remarcar, es la aparición de la tabla de iteraciones en este mismo apartado, donde he nombrado a cada iteración con un título, dependiendo de su contenido, y clasificado en esta tabla con referencias a cada una de ellas. Con esto se logra de un vistazo ver todas las iteraciones de mi proyecto y poder navegar entre todas ellas.

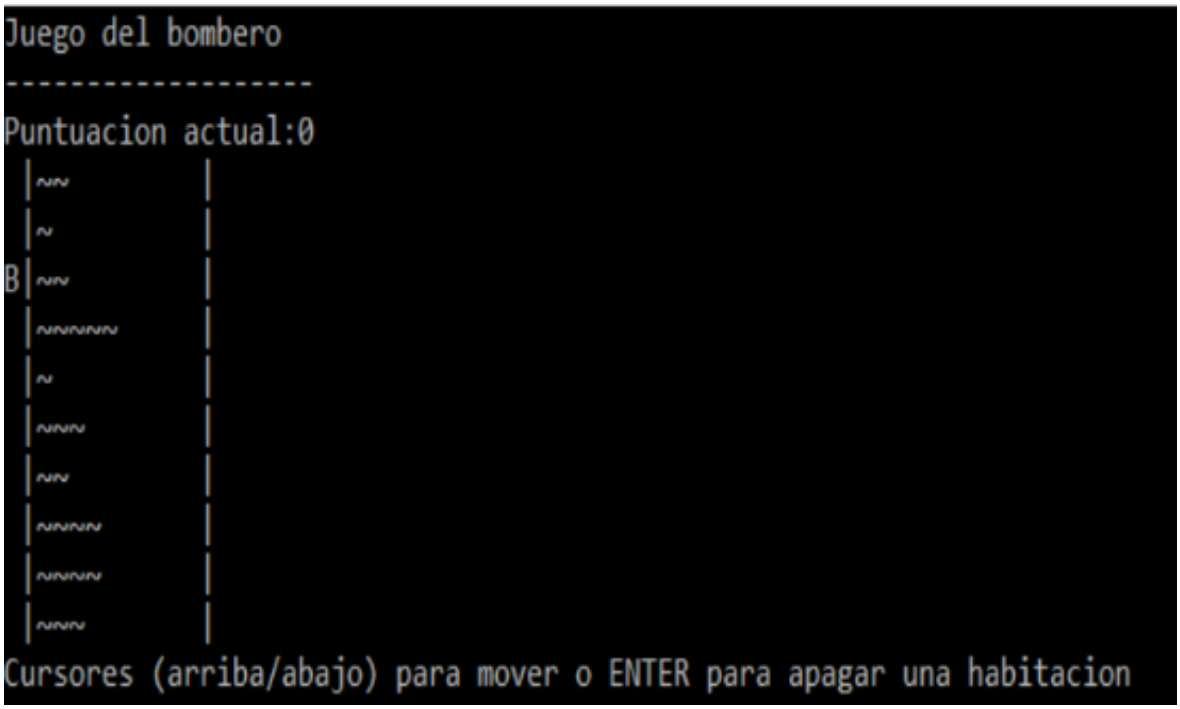


Figura 5.4: Prototipo juego del bombero.

Por último, he comenzado a prototipar un pequeño ECS, donde se muestre por consola, un producto mínimo viable, el cual hasta la fecha no he conseguido terminar. Trata de un juego simple llamado el juego del bombero, donde el bombero tiene que moverse de arriba a abajo para ir apagando fuegos en las diferentes habitaciones. Adjunto unas imágenes de lo que sería este prototipo en la figura 5.4

Esta idea proviene de una práctica que hice cuando estaba estudiando el grado de Ingeniería en Telecomunicaciones en la asignatura de Fundamentos de la Programación II, y para llevarla a cabo simplemente he retomado la memoria de esta práctica. Era una práctica programada en C, por lo tanto la he tomado como referencia y cambiado la estructura ya que el objetivo de esta era plantear un ECS.

Posteriormente a estar intentando este prototipo, he llegado a la conclusión de que tal vez,

un prototipo más sencillo sin input de teclado, sería mejor como producto mínimo viable, y he pospuesto esta idea para más tarde.

5.5. Iteración 4 - Primeros pasos con ejemplos de código

En primer lugar en esta iteración, he hecho retoques en la plantilla de esta memoria, ya que tenía demasiados errores conceptuales, espacios puestos a mano y demás pequeños fallos que a la larga podían ser grandes problemas.

Revisé el vídeo grabado de la tutoría anterior con mi tutor y a raíz de ésta hice una lista con detalles a cambiar que habíamos comentado, además de otra lista de tareas para la siguiente iteración. Todo esto lo coloqué en "Trello" para trabajar de forma más ordenada, cosa que hago y repito en cada iteración.

Como paso previo al código, también retoqué la teoría en el libro, colocando imágenes para complementar la explicación. Estas imágenes están hechas por mi manualmente. Además de retocar lo realizado hasta ahora, revisé todo el contenido de mi guía para detectar errores y mejorarla.

Una vez tenía el producto y memoria de forma ordenada, comencé a pensar en la forma de programar los ejemplos que iban a complementar mi explicación. Al final de la iteración anterior, comenté que el ejemplo que empecé a programar era complejo y decidí comenzar a programar algún ejemplo previo, como fue "Starfield".

Empiezo con el "Starfield" ya que veo en él una clara simpleza, todas las entidades tienen los mismos componentes y es muy buen ejemplo para comprender la estructura del Entity Component System además de una buena base para partir.

"Starfield" consta de un montón de estrellas con diferentes velocidades, moviéndose por la pantalla de forma horizontal. Era un poco difícil hacerlo de golpe, así que decidí hacer un esquema o diagrama, donde separar las entidades y componentes de mi proyecto, dejando clara la estructura del problema. Ese diagrama es el de la figura 5.5. Teniendo este esquema delante, comencé a programar estas entidades y componentes, y una vez teniendo los datos almacenados en los componentes, fue fácil diferenciar los sistemas que iba a necesitar para componer mi Entity Component System. Fueron dos, los sistemas de Render y Físicas. El proceso para conseguir tener los sistemas y la parte del gestor de entidades y componentes, el "EntityManager", fue sencillo dado que ya tenía los conocimientos necesarios adquiridos en cuarto curso. Aun así, revisé alguno de los vídeos del canal de YouTube "Profesor Retroman" de Francisco Gallego, Retroman (2014), para asegurarme de algunos conceptos.

Para dibujar el resultado y conseguir detectar el teclado por la terminal del sistema, utilicé

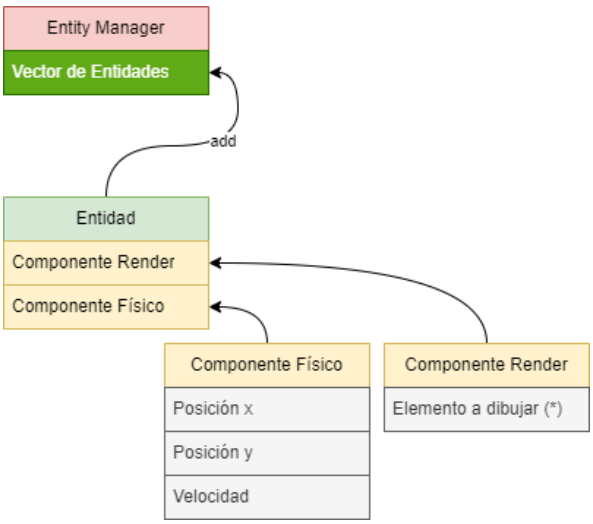


Figura 5.5: Diagrama de Entidades y Componentes de "Starfield".

una librería de uso de terminal de Gallego-Durán (2021), la cual permite dibujar con colores de forma muy sencilla, ya que tiene los códigos de los colores predefinidos. También permite controlar la entrada por teclado de una forma sencilla.

La segunda parte de esta iteración viene dada por otro ejemplo, donde realizaremos, ahora sí, el ejemplo de la iteración anterior, el juego del bombero. Para desarrollar este prototipo el proceso fue el mismo, y la estructura del "Entity Manager" también es la misma, ya que la podemos reutilizar para definir el proyecto que deseemos, pero en este caso ampliamos la complejidad en la estructura del ECS, ya que podemos diferenciar entidades según los componentes que tienen, aún de forma sencilla e ineficiente.

Antes de comenzar, el diagrama de entidades y componentes de nuevo fue necesario, ya que se aprecia mejor lo que se necesita, y es un punto clave para conseguir ver qué sistemas necesitaremos. Es el diagrama de la figura 5.6.

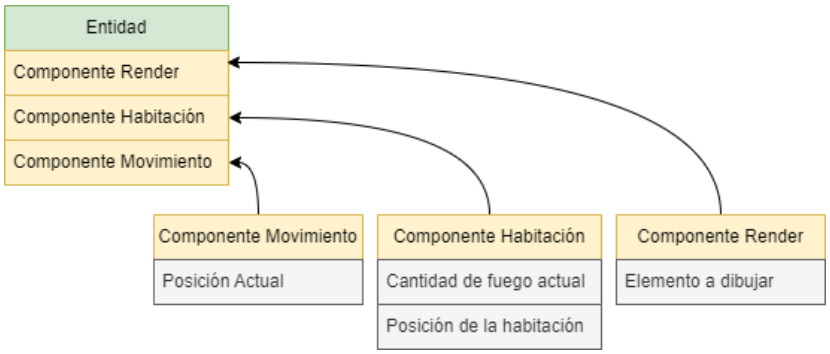


Figura 5.6: Diagrama de Entidades y Componentes de "Firefighter Game".

Con el diagrama delante, de nuevo, comencé a desarrollar los elementos que se muestran y seguidamente dos nuevos sistemas; el de renderizado, usando de nuevo la librería comentada anteriormente, y el sistema de movimiento, teniendo input y definiendo diferentes acciones para cada tecla usada.

Una vez con los prototipos terminados y pulidos, comencé a definirlos en la guía. Dividí por partes el Entity Component System y fui explicando con mis propias palabras lo que había ido haciendo, pero antes de estas explicaciones, coloqué una introducción a la estructura del ECS, ya que estos dos prototipos son los primeros y servirán para entenderla.

Ambos códigos se pueden encontrar en las siguientes referencias: Cantó-Berná (2023b), y Cantó-Berná (2023c).

5.6. Iteración 5 - Proyecto con diferenciación de componentes

Tras los pequeños proyectos de la iteración anterior, pensé en comenzar a hacer un proyecto más grande. Pensando en esto, me surgió la idea del pequeño juego del dinosaurio de Google, pero no quería hacer el mismo. Es por esto que he creado un juego que he llamado ”Save the OVNI”, partiendo de la idea inicial comentada anteriormente.

La pequeña diferencia es que en mi prototipo, el jugador podrá moverse en todas direcciones, y los enemigos serán misiles procedentes de derecha a izquierda, los cuales habrá que esquivar, sumando puntuación cuanta más distancia recorramos. En la figura 5.7 se puede apreciar el resultado de este proyecto. Pero antes de comenzar a programar esto, tuve que hacer como

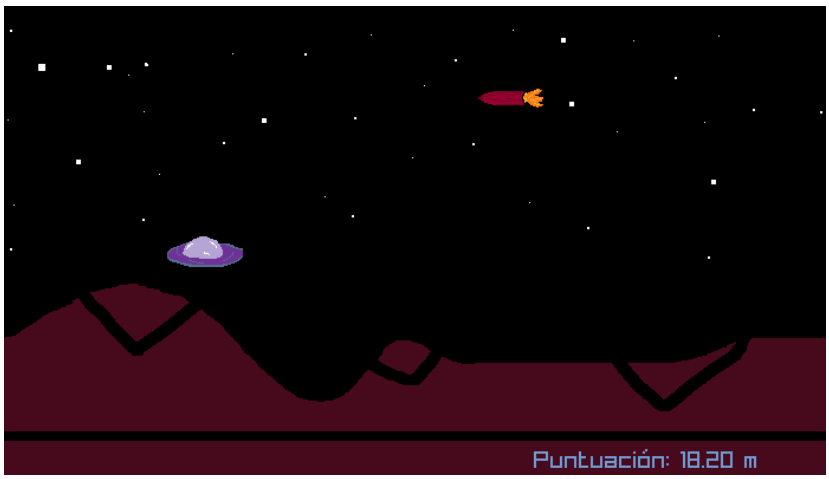


Figura 5.7: Juego ”Save the OVNI”.

en ejemplos anteriores, un pequeño diagrama para tener una primera visión principal del

problema. Esto nos beneficia mucho, ya que el objetivo principal de este proyecto es aplicar el ECS, diferenciando componentes. Este diagrama es el de la figura 5.8.

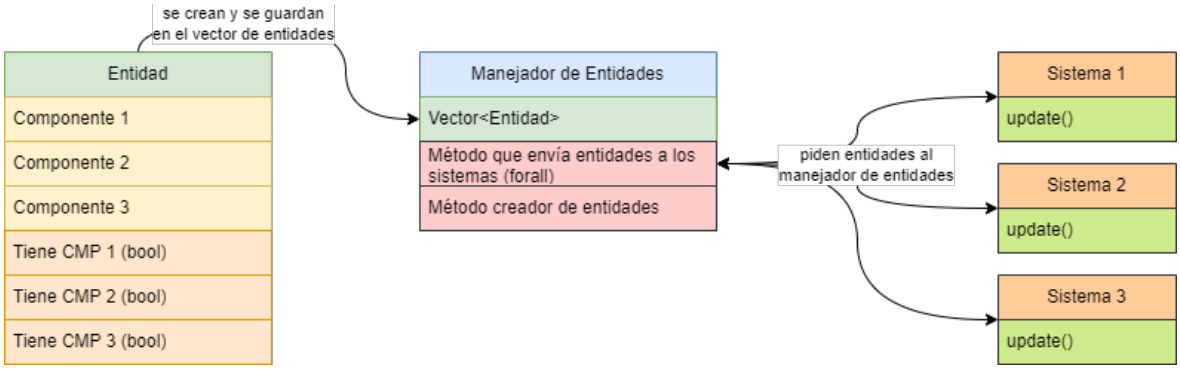


Figura 5.8: Diagrama de Entidades y Componentes de "Save the OVNI".

Como en otros proyectos anteriores, una vez con el diagrama delante procedo a programar. En primer lugar los componentes y entidades, y seguidamente el manejador de entidades, que retomamos del proyecto anterior, pero al que voy añadiendo funciones necesarias en cada momento. De esta forma se consigue observar un avance en esta estructura. Por último defino y programo los sistemas necesarios según los componentes que requieran de actualización y creo la clase del juego "Game" donde todo lo que hemos programado se une.

Además de este nuevo proyecto, en esta iteración he cambiado la plantilla del libro, ya que no me convencía la que usaba para el tipo de documento que estoy creando. Por lo tanto, pasé un buen tiempo eligiendo de nuevo una plantilla y modificándola para conseguir un mejor producto.

Al hacer el cambio de plantilla, tuve en cuenta la teoría que ya tenía escrita, y aproveché para releer y reescribir partes que podían estar mejor, incluso explicar las cosas de formas diferentes para que sea más fácil de entender y contrastar.

Otros cambios de esta iteración fueron incluir el código en "archive.org" en lugar de tenerlo todo al final del libro en un anexo, ya que se hacía difícil de comprender todo el código junto y sin separación.

En conclusión, esta iteración la he utilizado para reestructurar la plantilla y comenzar el desarrollo de un nuevo proyecto, el cual ya utiliza RayLib para dibujado y colisiones. Esto hace que haya un gran salto entre mis anteriores proyectos y este, por lo tanto, para la siguiente iteración el objetivo es encontrar un proyecto intermedio, donde haya una toma de contacto con RayLib.

5.7. Iteración 6 - Reorganización e introducción a RayLib

En esta iteración, en primer lugar comencé a hacer cambios en la estructura del documento, principalmente en la organización del contenido. Esto lo hice debido a que todo se veía muy denso, por lo tanto establecí unos apartados y subapartados para cada sección del documento.

Este cambio vino dado tras la última reunión con mi tutor en la cual participó un compañero, Rodrigo Guzmán. Con él y con Francisco Gallego estuvimos hablando de reorganizar el código para que no fuera tan denso, y surgió la idea de separar el contenido por apartados más pequeños a los que había en aquel momento.

Además añadí figuras sombreadas las cuales me son útiles para, a la hora de explicar, hacer énfasis en la parte de los diagramas a la que me refiero. Unido a esto también he comenzado a utilizar cajas dentro del código, para seleccionar en un fragmento extenso, a qué me estoy refiriendo. Similar a las figuras, pero con código.

Una vez realicé todos estos cambios en el contenido que ya tenía, comencé a modificar un pequeño detalle en el proyecto que realicé en la iteración anterior, Save the OVNI. El problema venía a la hora de morir, cuando te impactaba un cohete, no se veía con claridad el momento de la muerte y se cambiaba a una pantalla final.

El cambio viene tras haber observado ese problema, por lo tanto, incluí un apartado en este proyecto dedicado a mejorar el resultado. Haciendo que la colisión permanezca en la pantalla y ésta sea una especie de pantalla final, 5.9

Figura 5.9: Pantalla final una vez colisionado en el proyecto "Save the OVNI".

Después de haber realizado esta última modificación al proyecto del OVNI, y tras la reunión con mi tutor y compañero, llegamos a la situación de que el salto de nivel y de contenido entre el juego del bombero y el juego del OVNI era muy grande, por lo que decidí hacer un proyecto intermedio, donde se introdujese la librería RayLib a la par que creamos un primer proyecto con gráficos anterior al del OVNI. Este proyecto intermedio será el juego del bombero de nuevo, pero haciendo los cambios pertinentes para emplear gráficos sencillos.

Esto sirve de introducción a los gráficos para ejemplos futuros, donde no se le dará tanta importancia. Por lo tanto es necesario un apartado para comentar como he hecho todos esos apartados gráficos y se conozca su origen.

En cuanto al proyecto, simplemente modifiqué el componente de renderizado para uso de terminal a un componente gráfico con textura. En lugar de imprimir por pantalla, he usado RayLib, modificando cada impresión por una llamada al método de dibujado. Es importante que se aprecie que simplemente cambiando un componente y un sistema, el juego sigue funcionando igual pero de forma gráfica.

Con otro proyecto terminado (5.10), creé su espacio en la web "Archive.org" y subí todo el contenido. Por último modifiqué los códigos que he cambiado en esta iteración para que estén actualizados.

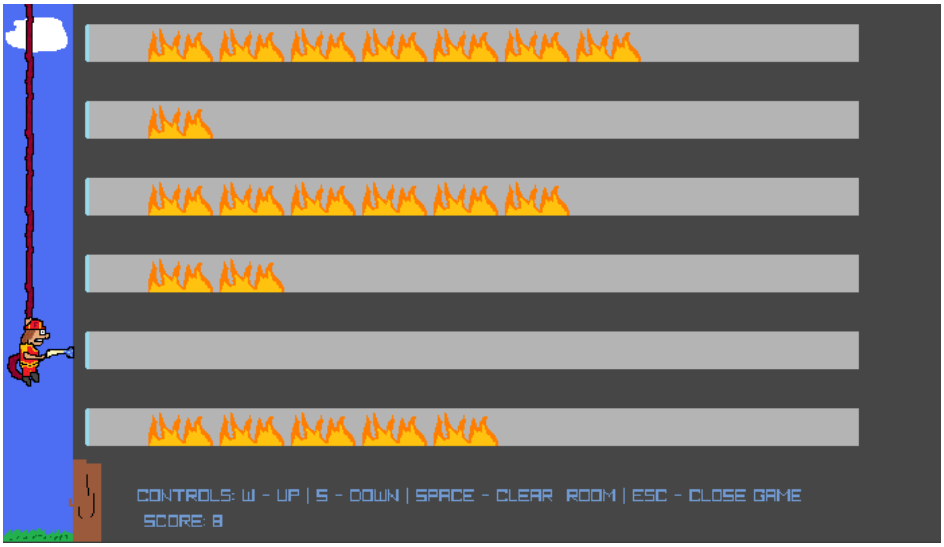


Figura 5.10: Proyecto "Firefighter Game" con RayLib.

5.8. Iteración 7 - Memoria y elección de tecnología

La iteración 7 es una iteración planificadora, más que de obtención de resultados, ya que en ella se han llevado a cabo muchos momentos de decisiones importantes que contaremos más adelante.

Esta iteración comienza por el desarrollo de la memoria del trabajo. Hasta este punto del proyecto solo tenemos partes concretas, como el marco teórico y todas las iteraciones anteriores, pero nada más.

Comencé buscando cómo desarrollar los diferentes apartados de objetivos y metodología. Respecto al primero, con el resultado final casi en mano, definí una serie de puntos listados al principio de este apartado, para que de un vistazo se puedan observar los objetivos de la guía. En cuanto a la metodología, tuve que hacer una búsqueda de los diferentes tipos de metodología que existen para recordarlas. Una vez expuestas, me di cuenta de que ninguna seguía mi método, por lo tanto lo defino como una metodología iterativa, híbrida entre Scrum y por prototipos, sin llegar a encasillarla en ninguna de las existentes.

Una vez con estos dos apartados finalizados, completé todos los demás puntos de la memoria dejando únicamente vacías las partes que todavía no se pueden redactar, como el último proyecto.

La mayor parte de la iteración fue para la puesta a punto de la memoria, aun así, el producto también tuvo modificaciones. Creé un apartado final que se titula ”resumen de lo aprendido”, donde se expone lo que el subproyecto busca enseñar a los lectores y se muestran los resultados de este.

Además en la guía cambié todos y cada uno de los códigos que aparecen, usando un estilo y paquete diferente, para que el resultado quede más profesional, y se compongan de colores más fáciles de diferenciar y leer. Junto al cambio anterior, también eliminé las cajas que encapsulaban partes del código para enfocar la atención en esos fragmentos. Lo he sustituido por recortes de código donde solo aparece el fragmento del que hablo en cada momento. De esta forma queda un resultado más fácil de leer y entendible.

Llegados a este punto, con la memoria preparada, el siguiente paso es pensar en cuál va a ser el próximo proyecto y pensar en qué contendrá.

En primer lugar, el objetivo es hacer un juego con diferentes componentes, cinco o seis, y sacar a estos de las entidades, para formar nuevas estructuras para almacenar estos datos, los Slotmaps. Para ello, hay que pensar que necesitaremos añadir, eliminar, obtener y modificar componentes.

El avance va a ser muy amplio, por lo tanto primero desarrollaremos un Slotmap para un tipo de componente y después basándonos en todo lo que tenemos creado, necesitaremos tantas estructuras como componentes tengamos. Esto es lo que hace que tenga sentido crear un Slotmap genérico para todos los tipos de componentes utilizando plantillas.

Lo único que dejaremos sin hacer plantilla es la forma de tratar a los componentes en el "component storage", ya que supondría un avance de contenido muy amplio para los lectores.

Por último las entidades tendrán llaves, que necesitaremos en el "Component Storage" para recoger el componente de dicha entidad. Esto queda así tras mucho tiempo de investigación y medición de dificultades, ya que al principio la idea era hacer que las entidades simplemente estuviesen compuestas por un id. Con ese elemento la idea era poder hacer referencia a todos sus componentes, pero esto hace que la dificultad de desarrollo aumente considerablemente, y quedará pendiente para un futuro proyecto.

5.9. Iteración 8 - Último proyecto y detalles finales

Esta iteración tiene un objetivo claro y es desarrollar el último juego del proyecto, "The Final Room". Para ello, contamos con los avances del apartado anterior, para comenzar a preparar lo que tenemos que programar.

Para que sea más llevadero, al igual que en otros proyectos, decidí hacer un diagrama de la estructura principal de componentes, entidades y la parte de datos del ECS. Este diagrama es el de la figura 5.11

Además de este diagrama, diseñe un segundo diagrama para hacer una idea al lector de que estoy haciendo y cual es la idea a seguir. El diagrama de la figura 5.12, muestra una estructura de como será la interfaz del juego.

La estructura a seguir es clara, ya que tenemos componentes y entidades separados, y los componentes se guardan en Slotmaps, que a su vez se guardan en el "Component Storage". He decidido utilizar esta estructura para los datos por que optimiza mucho el rendimiento y la memoria, además de ser un buen paso para dejar de tener todas las entidades con todos los componentes.

En primer lugar, implemento un Slotmap para un solo componente, para ello sigo una lógica, basada en el funcionamiento de las matrices de dispersión (Los Slotmaps y matrices de dispersión, son el mismo elemento). Esto lo hago como primer paso, para que los lectores entiendan cómo funciona para un solo componente. De esta forma se aprecia la cantidad de código que se requiere repetir para más componentes y de esta forma, dar pie a la automatización del Slotmap para el componente que sea.

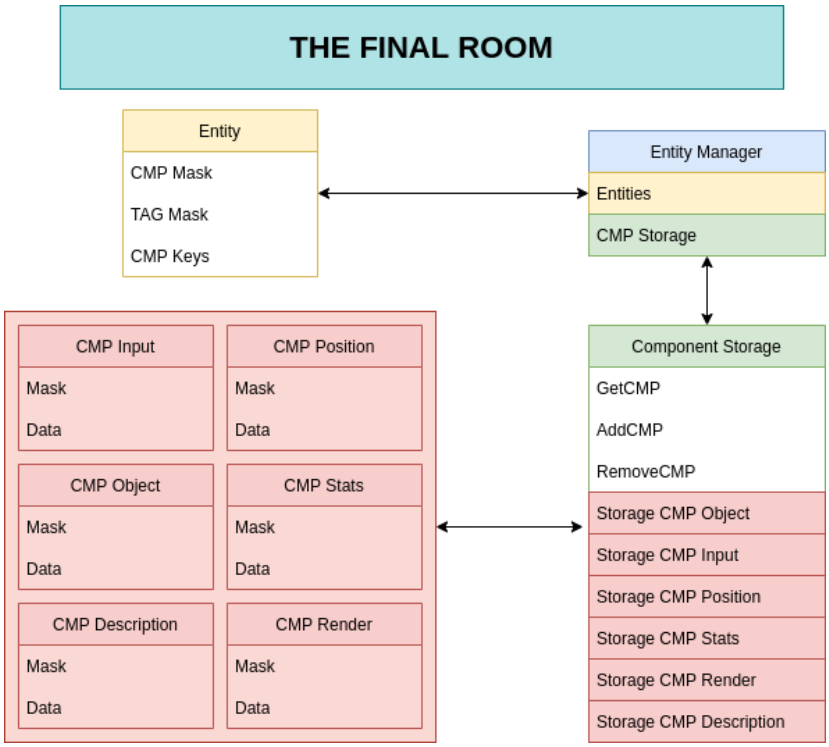


Figura 5.11: Diagrama de la estructuración de los datos del proyecto "The Final Room".

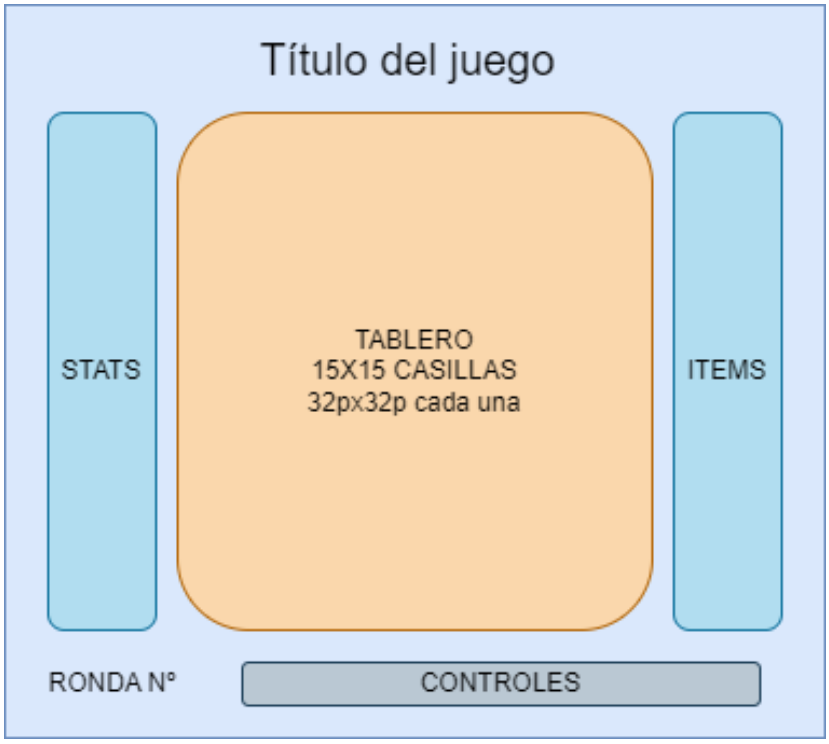


Figura 5.12: Diagrama de la interfaz de "The Final Room"

En el ”Component Storage”, pasa algo similar. El ”Component Storage” guarda todos los Slotmaps en su interior, uno por componente. El problema es que necesitamos funciones para añadir, eliminar, y obtener los datos de estas estructuras.

No podemos dar el paso para automatizar estas funciones y simplemente tener una para el componente que sea, por lo tanto, tendremos que hacer estas funciones para cada componente distinto. Además en el libro se explica que esto puede mejorarse y que en futuros proyectos se implementará, pero para el actual, será más que suficiente.

Esa es la idea que me ha llevado a este resultado, que es muy bueno como paso siguiente al proyecto anterior. Además se han implementado estados, sistemas nuevos y etiquetas para diferenciar las entidades que queramos. Esto nos sirve para por ejemplo seleccionar solo aquellas que decida que sean enemigos.

Todos estos detalles estarán plasmados en el libro para el lector y además he implementado ejercicios extra, para que también tenga actividades, si quiere, ya que también estarán los resultados de estos ejercicios.

Figura 5.13: Proyecto ”The Final Room”. Captura del juego

Una vez con el resultado en la mano, decidí cambiar el orden de estructuración de este proyecto en el libro, por que al tener un motor ECS tan grande, decidí separarlo en dos. Por un lado tengo el motor explicado, con el que hacer el juego en el segundo apartado del punto seis del libro.

El resultado de este proyecto se puede consultar en el apartado de la memoria ”Resultados”

(6), pero aquí se puede ver una captura del juego en ejecución, 5.13

Además de este nuevo proyecto, he mejorado los contenidos de la memoria, como la introducción al proyecto, haciéndola más extensa, y los objetivos, añadiendo un listado, para que sean más fáciles de identificar.

5.10. Iteración 9 - Resultado final

Esta es la última iteración.

Los últimos retoques han comenzado por revisar los códigos de los diferentes proyectos, arreglando errores y modificando en el libro los mismos. Los códigos una vez limpios, han sido subidos a la web de "Archive.org", para que permanezcan guardados para futuros lectores de esta guía.

También he creado los diferentes ejecutables para Linux y he escrito archivos "Readme" para explicar cómo lanzarlos y qué hacer en caso de tener un sistema Windows, que será usar el subsistema Linux.

El siguiente paso ha sido revisar la estructura del libro y modificarla en caso del último proyecto, el creado en la iteración anterior. El motivo recae en que ha quedado un proyecto muy largo, y como lo he dividido en dos partes, por un lado motor de entidades y por otro implementación del juego, los he separado en dos capítulos.

Además de este cambio, he añadido un nuevo apartado al capítulo tres, donde hablábamos de nuestros dos primeros proyectos, ambos creados para lanzarlos en terminal. El nuevo apartado habla del uso de la librería "Básic Linux Terminal" de Gallego-Durán (2021), para que los usuarios no vayan a ciegas durante el desarrollo de ambos proyectos. He incluido y explicado las funcionalidades que utilizo.

Como último capítulo del proyecto, he añadido un apartado de recopilación para exponer el resultado también en el libro y un apartado de futuro aprendizaje, para hacer saber a los interesados que habrá segunda parte de este proyecto.

Por último he revisado tanto este documento, como el libro y he repasado cada uno de los apartados, en busca de errores, del tipo que sean, corrigiéndolos.

Ha sido una última etapa para pulir el resultado, donde me he dado cuenta de puntos con falta de explicación que he rellenado de contenido, buscando la mejor experiencia para el lector.

6. Resultados

En este apartado trataremos los diferentes tipos de resultados que hemos obtenido a lo largo del desarrollo de nuestro proyecto.

Principalmente estarán separados en dos bloques. El primero representará el producto, que al fin y al cabo es el objetivo de este trabajo, la guía sobre cómo elaborar un motor usando la arquitectura ECS desde 0 en C++. En segundo lugar repasaremos más detalladamente cada uno de los resultados de los proyectos internos de la guía.

Por último y antes de comenzar, quiero mostrar mi orgullo al haber conseguido dichos resultados, en el tiempo propuesto.

6.1. Resultados de la guía

Comenzaremos hablando de uno de los objetivos principales del proyecto, que es la comprensión del Entity Component System de la mejor forma posible. Para ello, hemos introducido una parte principal completamente teórica, donde se explican y muestra con pequeños fragmentos de código qué partes tiene un ECS y cómo se comporta, para una vez comprendidos esos conocimientos poder comenzar a desarrollar los pequeños proyectos posteriores.

Esta primera parte teórica se muestra en el segundo capítulo de la guía. En la figura 6.1 se puede observar un pequeño apartado de este capítulo donde se comienza a hablar de este tema.

Como resultado, ha quedado un documento final muy cuidado, gracias a que hemos colocado teoría y partes de código intercaladas, y con explicaciones de cada tramo del proceso. Por lo tanto, se ha logrado uno de los objetivos principales a la hora de desarrollar las explicaciones y contenido. Prueba de ello puede ser por ejemplo alguna de las siguientes imágenes, donde se pueden apreciar estos temas. En la figura 6.2, se puede observar en el índice del proyecto la aparición de apartados teóricos intercalados con desarrollo, lo cual se puede observar también en la figura 6.3, directamente en el documento.

Algunas de las explicaciones que se dan a lo largo del documento, las podemos encontrar de

2 Entity Component System

En el mundo de los videojuegos, la optimización del código, los datos y la memoria es muy importante, de esto depende la potencia de tu creación. Uno de los problemas a la hora de hacer un juego es la flexibilidad de tu código, ya que si no se sigue ninguna estructura u orden será un desastre el resultado.

Necesitaremos unas rutinas de programación que permitan el diseño, creación y funcionamiento de un videojuego. Esta rutina, la conseguiremos desarrollando un motor Entity Component System (ECS), el cual nos permite tener entidades relacionadas con sus componentes y de un correcto flujo entre los sistemas.

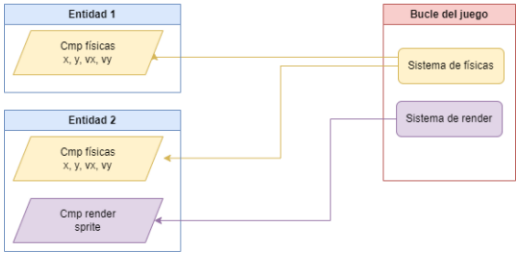
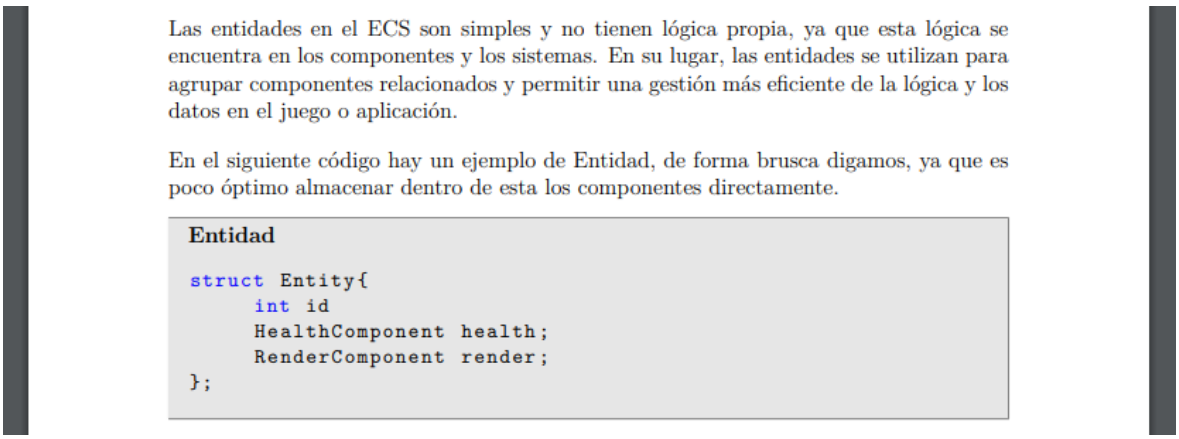


Figura 7. Diagrama simple del patrón Entity Component System

Figura 6.1: Comienzo del apartado de explicación y toma de contacto con el ECS.

Figura 6.2: Índice del proyecto donde se muestra la intercalación de teoría y desarrollo de código.



Las entidades en el ECS son simples y no tienen lógica propia, ya que esta lógica se encuentra en los componentes y los sistemas. En su lugar, las entidades se utilizan para agrupar componentes relacionados y permitir una gestión más eficiente de la lógica y los datos en el juego o aplicación.

En el siguiente código hay un ejemplo de Entidad, de forma brusca digamos, ya que es poco óptimo almacenar dentro de esta los componentes directamente.

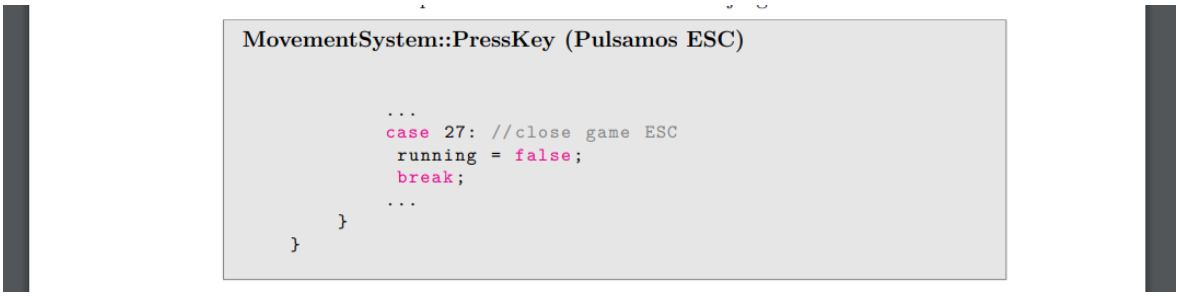
Entidad

```
struct Entity{
     int id
     HealthComponent health;
     RenderComponent render;
};
```

Figura 6.3: Fragmento del documento donde se muestra la intercalación de teoría con códigos.

diferentes formas. Hay apartados difíciles de entender, los cuales he expresado de diferentes formas, haciendo que sea más fácil de asumir ese conocimiento. Esto lo hace también más completo.

A lo largo del documento, también hay explicaciones largas, las cuales se muestran por trozos, para centrarnos en la explicación de esa pequeña parte. Esto lo podemos observar en la figura 6.4.



MovementSystem::PressKey (Pulsamos ESC)

```
            ...
            case 27: //close game ESC
             running = false;
             break;
            ...
        }
    }
```

Figura 6.4: Fragmento de código recortado

Por último, otro de los detalles introducidos para la ayuda de la lectura, es la representación de las situaciones con diagramas propios, que a su vez he ido recuperando a la vez que avanzamos, sombreando las partes ya realizadas. Esto hace que el lector se enfoque solo en la parte iluminada. De esta forma puede hacerse una idea rápida sobre lo que llevamos hecho, por donde vamos y que nos queda por hacer. Un ejemplo de estos diagramas es el de la figura 6.5.

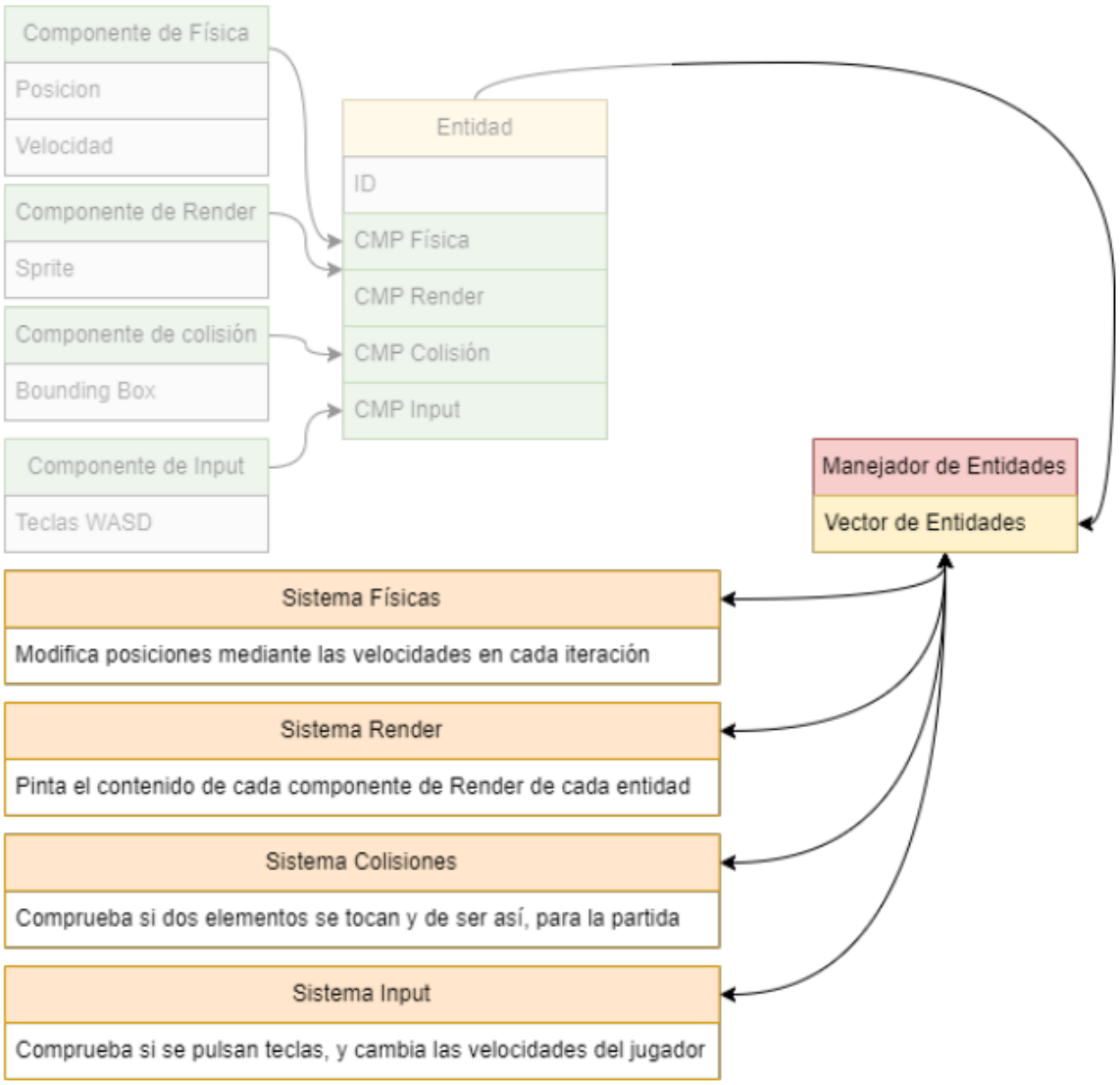


Figura 6.5: Diagrama con sombreado de la parte ya realizada.

6.2. Resultados de los subproyectos

He decidido separar este apartado como resultados de los proyectos que incluye la guía como diferentes a los resultados generales del documento.

En primer lugar describiré los resultados obtenidos en el ámbito explicativo de cada uno de los subproyectos y posteriormente expondré los resultados de estos, con imágenes finales de estos.

6.2.1. Starfield

Este es el primer proyecto de la guía por lo tanto el lector no tiene que saber absolutamente nada más que la introducción anterior.

Es un proyecto donde los objetivos son ponerse en contacto con la estructura del ECS y comprender como representar una entidad. Es por eso que elegí este proyecto para ser el primero. Es muy sencillo ya que solo requiere dos sistemas y dos componentes. No tiene interacción con el usuario.

El resultado es un campo de estrellas moviéndose de izquierda a derecha desapareciendo por un lado y reapareciendo por el otro.

La estructura no es demasiado óptima respecto a cómo lo será en futuros ejemplos, pero como he mencionado anteriormente, sirve para observarla y entenderla.

A continuación se muestra en la figura 6.6, los resultados finales de este proyecto.

Figura 6.6: Proyecto Starfield terminado.

En este proyecto solo tenemos 2 tipos de componentes el de físicas y el de renderizado, y sus respectivos sistemas. Cada asterisco es una entidad. La estructura final es sencilla y fácil

de entender, ya que todas las entidades tienen incluidos todos los componentes.

6.2.2. Firefighter Game

Este es el segundo proyecto de la guía, que requiere los conocimientos del proyecto anterior. Comenzamos definiendo los componentes necesarios para formar nuestras entidades. Esto lo hemos hecho mediante un diagrama (figura 6.7) que representa la estructura de la composición de entidades del proyecto. De esta forma será fácil saber que sistemas necesitaremos. Una vez

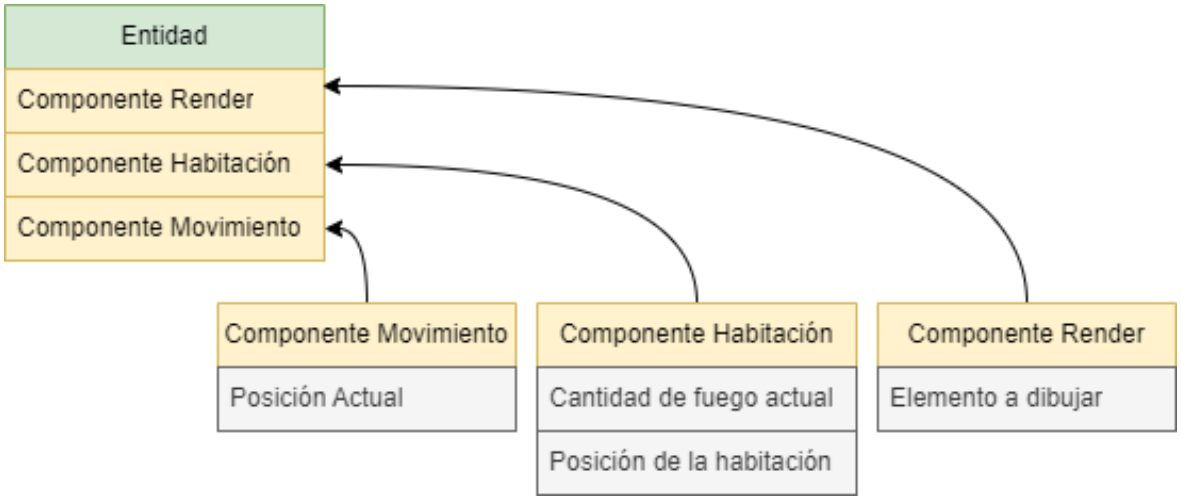


Figura 6.7: Diagrama de la estructura del Entity Component System del juego "Firefighter Game".

con la estructura presente fue fácil comenzar a desarrollar lo que necesitamos. En este caso fueron tres componentes diferentes y dos sistemas. Tendremos los componentes de físicas, dibujado y habitación, y por otro lado los sistemas de dibujado y movimiento.

Este fue el segundo proyecto con una finalidad, representar una pequeña diferenciación de componentes inicial. Este fue el escogido debido a su simpleza, ya que solo necesita dos elementos diferentes, habitaciones con fuego en su interior y el bombero. Esta separación en dos entidades es la que nos hace ver lo simple que es, ya que todas las entidades tienen los componentes de físicas y dibujado exceptuando al bombero que tiene el de movimiento también.

La diferencia reside en la forma de crear las entidades, ya que ahora cada componente se verá acompañado de un booleano que representará si se tiene o no dicho componente.

Otro de los objetivos que se refleja en el resultado es la estructura del ECS, ya que forma parte del mismo apartado principal que el proyecto anterior. Ambos tienen como objetivo representar la estructura principal de la arquitectura ECS.

A continuación muestro una imagen final del juego, donde se pueden apreciar las dos entidades, el bombero ”B” y cada habitación como otra entidad con componente ”room” donde un número indicará la cantidad de fuego a representar dentro de cada una de ellas.

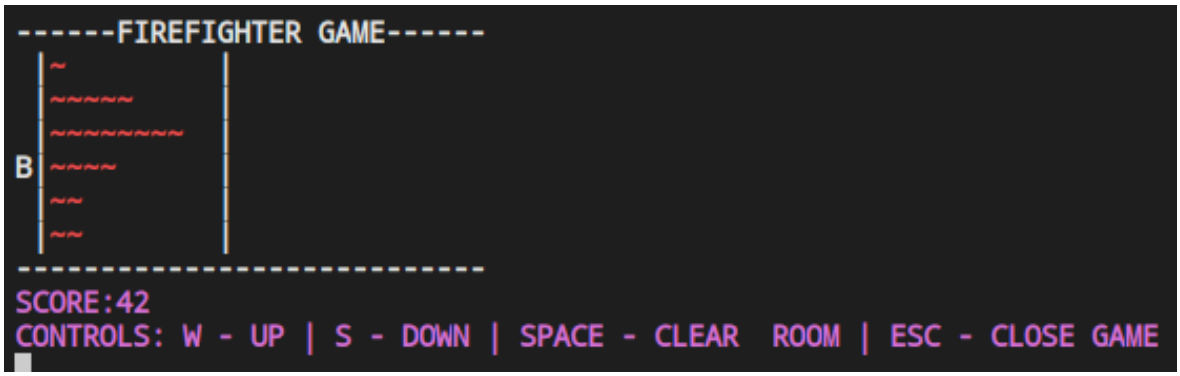


Figura 6.8: Juego del bombero.Resultado visual del Proyecto 2.

6.2.3. Firefighter Game (RayLib)

El resultado de este proyecto es una clara muestra de que simplemente cambiando el componente de renderizado podemos conseguir un juego con gráficos si usamos una librería gráfica.

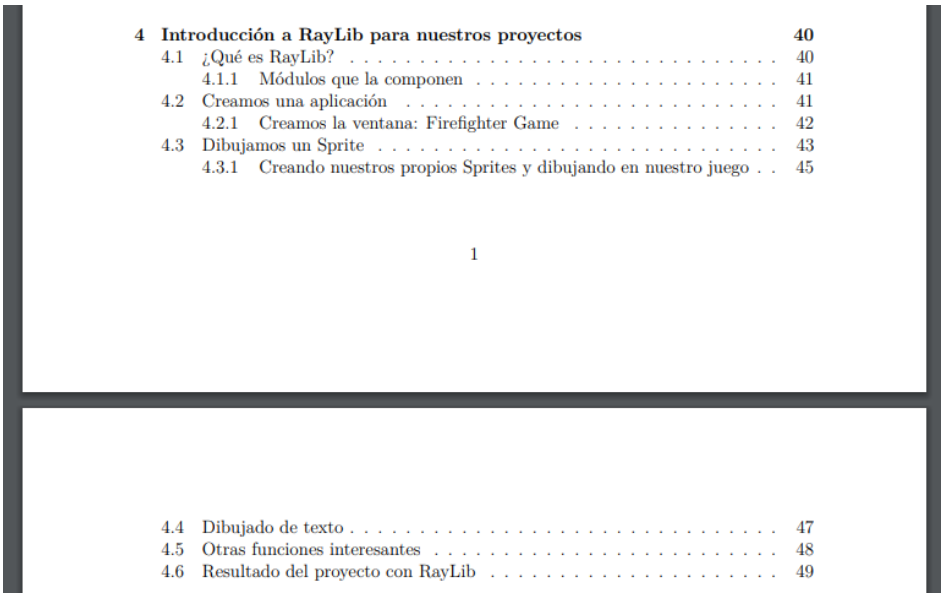


Figura 6.9: Índice del apartado de introducción a RayLib

El objetivo de este proyecto era explicar y llevar a cabo el paso de un juego que utiliza un motor con Entity Component System pero no utiliza librería gráfica a uno donde sí que

ocurra. Para ello hemos utilizado RayLib.

Como resultado, en la figura 6.9 se puede apreciar el índice del proyecto, donde cada apartado representa una parte del dibujado y uso de la librería, como las funciones más básicas y necesarias para conseguir lo que queremos. En los subapartados llevamos a cabo los cambios entre el proyecto "Firefighter Game" al nuevo.

El ECS es muy parecido al que hemos realizado en el proyecto anterior, pero con los cambios necesarios para poder dibujar los "sprites" en una aplicación en lugar de en la terminal del sistema.

Los cambios más contundentes los podemos encontrar en los componentes de renderizado y en el sistema de renderizado, y en la forma de detectar la entrada por teclado, ya que anteriormente usábamos la librería para uso de terminal Gallego-Durán (2021).

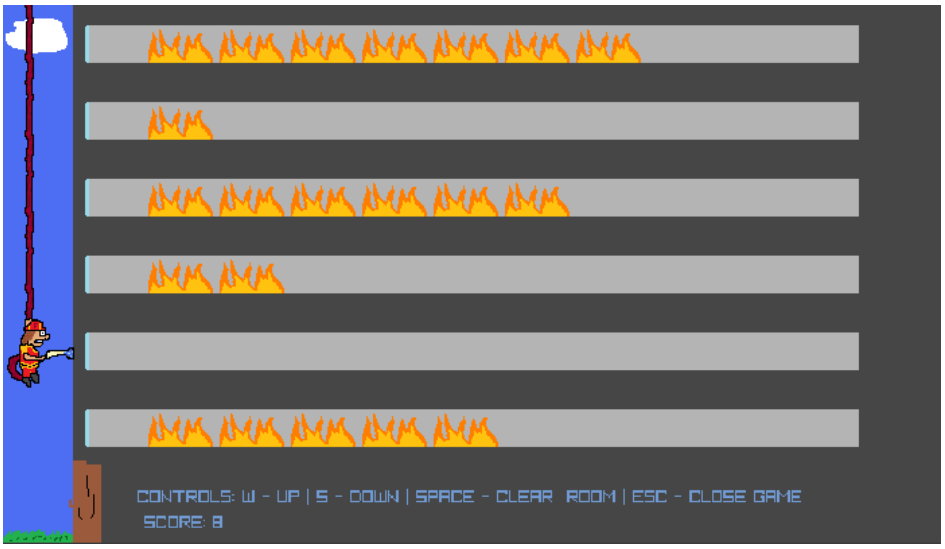


Figura 6.10: "Firefighter Game" usando RayLib para el dibujado.

Por último y como muestra final del resultado de este proyecto, en la figura 6.10, aparece el juego "Firefighter game" usando RayLib. Los "sprites" que he usado para mi prototipo son propios, y aunque no tengo grandes capacidades para dibujar, son representativos y servirán para mostrar el resultado.

6.2.4. Save the OVNI

Seguimos con los resultados de "Save the OVNI" un juego bastante más avanzado que los anteriores ya que comenzamos a parte de nuestro objetivo principal, a usar estados. Esto nos permite crear un menú donde se puede escoger dificultad y colisiones.

El objetivo de este proyecto, era avanzar en la estructura de nuestra entidad, teniendo en cuenta de donde venimos, es decir, modificando las entidades con un booleano por componente por elementos de la clase ”std::optional” , figura 6.11. De esta forma en el ejemplo anterior hemos introducido a este, ya que lo que ”std::optional” esta compuesto por la estructura que nosotros usábamos para el apartado anterior. Ahora tenemos entidades más sencillas de entender, donde podemos saber fácilmente si tenemos o no componentes.

Entidad

```
struct Entity{
    Entity();
    int id;
    std::optional<CollisionCMP> coll;
    std::optional<InputCMP> inp;
    std::optional<RenderCMP> rend;
    std::optional<PhysicCMP> phy;
    private:
    inline static int nextID{1};
};
```

Figura 6.11: Entidades usando ”std::optional” para el proyecto de ”Save the OVNI”.

También hemos introducido nuevos sistemas y componentes, siendo este el que más número tiene. Dispone de cuatro componentes, el de físicas, el de input, el de dibujado y el de colisión. También tiene cuatro sistemas que coinciden con los componentes. Un sistema de físicas, sistema de colisiones, sistema de movimiento o input y por último el de dibujado.

En este proyecto, se ha intercalado un apartado de teoría que habla sobre las colisiones, explicando que tipos existen, cómo funcionan y donde hemos puesto un ejemplo de uso de el tipo que vamos a usar, las AABB (Axis-Aligned Bounding Box).

En este proyecto, además de haber creado este juego, propongo una mejora al final del desarrollo. Trata de mejorar el orden de los sistemas y cambiar la pantalla final del juego, entrando un poco en el terreno de la mejora visual y mejora de funcionamiento. Otro de los apartados teóricos trata sobre la organización de los sistemas y su orden en el bucle del juego, un apartado muy importante para conseguir buenos resultados.

Como en el anterior, los ”sprites” utilizados son propios.

A continuación en la figura 6.12 se puede apreciar lo que hemos comentado anteriormente, el apartado de la guía donde proponemos una mejora para el lector a modo de ejercicio, como parte opcional, pero que también he desarrollado.

Finalmente este es el resultado del juego, donde la estructura del ECS es mucho más compleja y extensa pero seguimos funcionando de la misma forma, ya que esta arquitectura nos ofrece gracias a la forma de estructurarla, mucha facilidad para añadir y hacer nuestro proyecto mucho más grande. En las figuras 6.13, 6.14, 6.15, 6.21, queda reflejado el resultado final del juego propuesto para este apartado.

5.1.12 Proponemos una mejora visual

Para que el jugador tenga una mejor experiencia, vamos a proponer una mejora. La experiencia de juego es uno de los factores más importantes de este, ya que de esto dependerá que el jugador quiera seguir jugando o abandone el juego.

Una de las cosas a mejorar es ver como el jugador muere, y es lo que proponemos cambiar en este apartado. Actualmente al colisionar con un misil, automáticamente el juego cambia de estado a una pantalla final y no se aprecia el choque del todo.

Es por eso que nuestro objetivo a continuación es hacer que cuando muera el jugador se siga viendo esa colisión en la pantalla con el juego como si estuviese pausado, y salga el mensaje final en esa pantalla con la colisión de fondo. De esta forma el jugador siempre podrá ver cómo ha perdido.

Figura 6.12: Apartado a modo de ejercicio para mejorar el juego desarrollado.

Figura 6.13: "Save the OVNI": Pantalla principal y menú del juego

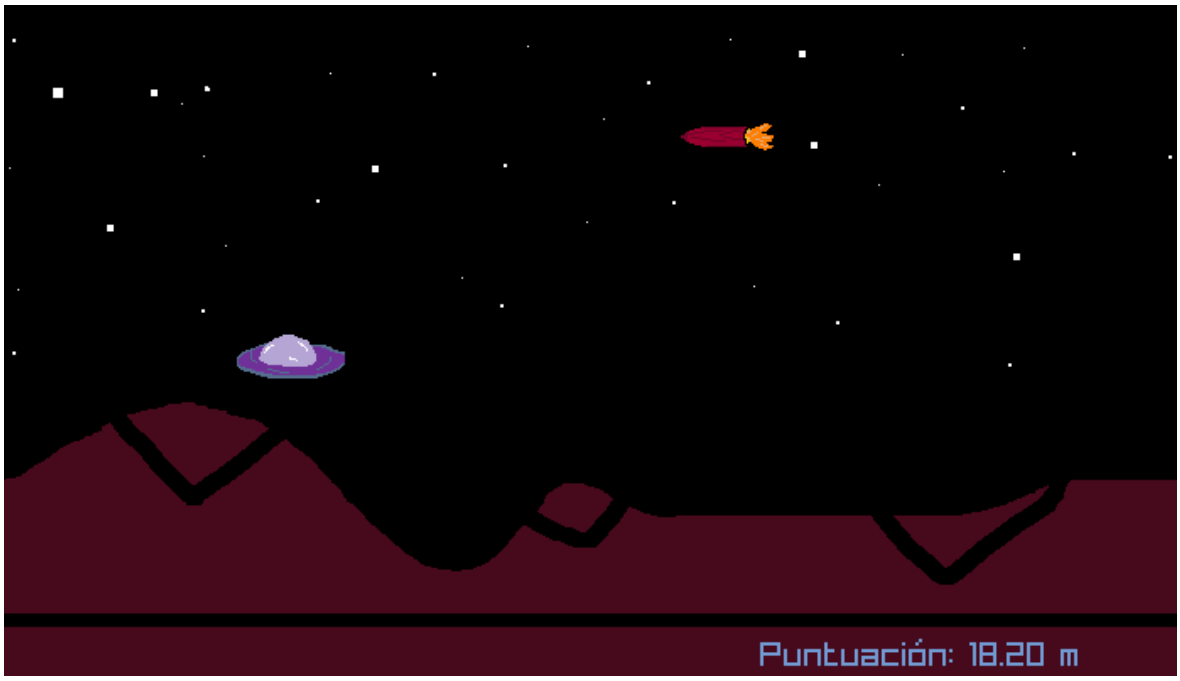


Figura 6.14: "Save the OVNI": Pantalla durante el juego

Figura 6.15: "Save the OVNI": Pantalla de muerte

Figura 6.16: "Save the OVNI": Pantalla final del juego

Como podemos apreciar en las figuras anteriores, podemos diferenciar entidades como por ejemplo el OVNI que tendrá los componentes de render, físicas, movimiento y colisión. Por otro lado, podemos apreciar enemigos (primer juego donde aparecen enemigos) representados por entidades con componente de render, físicas y colisión.

6.2.5. The Final Room

"The final Room" es el juego más completo de la guía. Disponemos por ejemplo de enemigos, objetos a recoger por el mapa, llaves para abrir puertas, diferentes niveles y generación aleatoria de niveles de forma infinita, entre otras cosas.

Esto lo hace un juego con muchas entidades, muchos componentes y datos. Por lo que hemos implementado unas estructuras de datos (Slotmaps) para estructurar bien nuestros elementos del juego.

Cumplimos con todos los objetivos propuestos para este juego, y además con un buen resultado. Tiene funcionalidades como coger objetos, moverse por turnos, matar enemigos, recibir daño, velocidad de movimiento y muchos más aspectos interesantes que hay que tener en cuenta, que en otros proyectos no hacía falta.

Como resultado es un buen ejemplo de como construir un motor Entity Component System,

cuando necesitamos mucho espacio o tenemos muchos elementos diferentes a crear.

El ECS creado, responde a todas las necesidades del usuario, creando entidades configurables. Puedes añadirle los componentes que quieras y etiquetas para categorizar cada entidad.

A continuación expongo en forma de lista los elementos que hemos conseguido en este último proyecto, antes de seguir con las imágenes del resultado.

- En primer lugar hemos implementado Slotmaps para almacenar componentes fuera de las entidades y facilitar el acceso, modificación y borrado.
- Nuestras entidades ahora tienen simplemente sus máscaras y sus llaves para recoger componentes, son estructuras sencillas y mucho más manejables.
- Hemos creado la interfaz "Component Storage", para que sea más fácil de usar de cara al usuario.
- Control del flujo de datos, creamos, eliminamos y gestionamos las entidades cada nivel, reutilizando el espacio que ya tenemos.
- Hemos creado todos los elementos que puede necesitar un juego a partir de este motor, que ahora usa etiquetas para separar entidades y elegirlas en el momento oportuno. Esto es posible gracias a las modificaciones realizadas al manejador de entidades.
- Por último hemos logrado un producto final al que poder jugar, y hemos propuesto mejoras al lector para animarlo a mejorarlo.

Estos son los principales aspectos a resaltar de este proyecto, que repito es el último de la guía y el más completo. Hemos logrado una gran optimización de la memoria y del flujo del juego.

A continuación, se pueden observar las diferentes figuras que representan capturas de nuestro juego, los diferentes niveles y distintas pantallas.

Figura 6.17: "The Final Room": Menú del juego

Figura 6.18: "The Final Room": WIKI o biblioteca

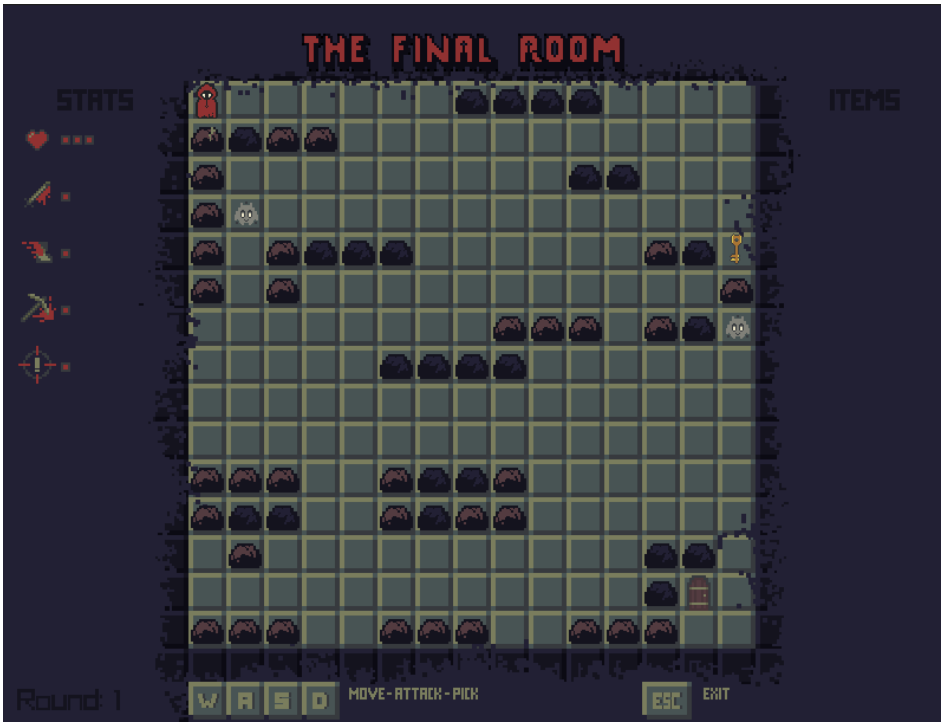


Figura 6.19: "The Final Room": Nivel 1 creado a mano

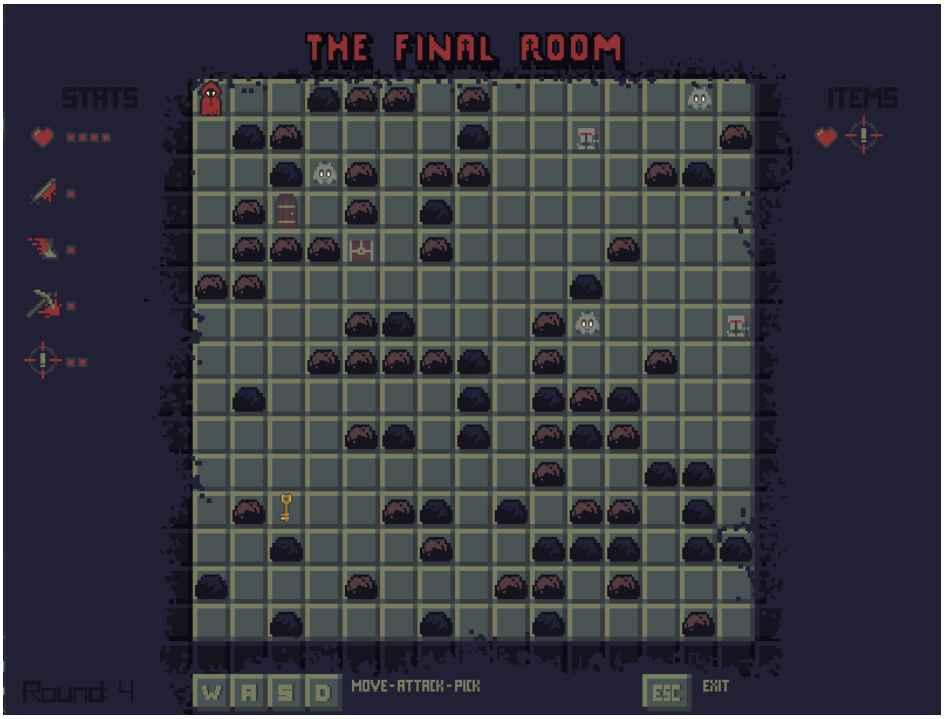


Figura 6.20: "The Final Room": Nivel 4 creado de forma automática

Figura 6.21: "The Final Room": Pantalla final

7. Conclusiones

En este proyecto, hemos abordado el mundo de los videojuegos, entrando de lleno en el mundo de la creación de estos, creando una guía para que los lectores puedan crear un motor propio de videojuegos desde 0 usando la arquitectura de software Entity Component System. Para ello he tratado de realizar una primera etapa para recoger diferentes perspectivas con el objetivo de conseguir un documento lo más agradable para el lector y sólido en conocimientos nuevos.

En esta conclusión, resumiremos los principales puntos de nuestro proyecto y destacaremos lo que este ofrece.

A lo largo de este trabajo, hemos encontrado algunas dificultades como la forma de expresión o de escribir para llegar de forma directa y sin rodeos a los lectores, que tengo que aclarar que no es una tarea nada fácil. Pero hemos empleado técnicas para dirigir el foco de atención a ciertos apartados, como los sombreados en diagramas entre otros.

La guía contiene un total de cinco proyectos internos, que desarrollan un motor de videojuegos ECS, de menor a mayor dificultad, y donde se exponen los cambios entre unos y otros casos. El resultado es un motor final válido y con el que estoy orgulloso. Aun así, mi intención era seguir con más proyectos y más avance en mi guía, pero el tiempo disponible para realizarlo lo ha puesto muy difícil. Esto quiere decir, que con el tiempo disponible el proyecto resultante es muy bueno, ya que comenzamos desde cero y hemos llegado a una optimización muy buena a la hora de manejar las entidades.

Para el futuro, mi idea es hacer un segundo tomo de la guía, partiendo por los conocimientos finales de esta, donde se toquen temas como plantillas o programación estática. En este tipo de programación, el código se compila antes de su ejecución y se genera un archivo ejecutable que contiene todo lo necesario para que el programa funcione correctamente. En otras palabras, todas las dependencias y librerías que utiliza el programa se incorporan en el archivo ejecutable en tiempo de compilación, en lugar de en tiempo de ejecución. Esto hace que el programa sea más rápido y eficiente en su ejecución, ya que no necesita buscar ni cargar librerías adicionales en tiempo de ejecución. Además, la programación estática también permite detectar errores de programación antes de la ejecución del programa, lo que facilita la depuración y la corrección de problemas.

Otro tema a abarcar en el futuro, es la metaprogramación. La metaprogramación es un

paradigma de programación que consiste en escribir programas que generan otros programas o que manipulan el código fuente de los programas en tiempo de compilación o ejecución. En otras palabras, la metaprogramación es una técnica que permite a los programadores escribir programas que puedan modificar su propio comportamiento o el comportamiento de otros programas. Se utiliza para automatizar tareas repetitivas o para facilitar el desarrollo de software complejo y nos puede venir bien para ciertos temas futuros mejorando nuestros motores.

Para concluir, comentar de nuevo que mi guía ha quedado para mí muy completa para el tiempo del que se disponía y me gusta que además de pie a un futuro proyecto, ya que he disfrutado mucho con este. Se investiga sobre el ECS, y se desarrolla todo paso a paso, entendiendo desde pequeños detalles, hasta grandes estructuras. Espero con este proyecto, poder ayudar a futuros alumnos de ingeniería multimedia en su paso por la carrera y concretamente por las asignaturas de videojuegos.

Bibliografía

Bloomberg, S. (2016). *Ecs.* Descargado de https://github.com/redxdev/ECS

Caini, M. (2017). *Entt (ecs engine).* Descargado de https://github.com/skypjack/entt

Cantó-Berná, L. (2023a, mayo). *Cómo desarrollar un gameengine entity-component-system en c++20.* Descargado de https://archive.org/details/TFGLaureanoECS20

Cantó-Berná, L. (2023b, febrero). *Ecs project: Starfield.* Descargado de https://archive.org/details/starfield_202302

Cantó-Berná, L. (2023c, febrero). *Entity component system project: Firefighter game.* Descargado de https://archive.org/details/firefighter-game

Cantó-Berná, L. (2023d, marzo). *Entity component system project: Firefighter game with raylib.* Descargado de https://archive.org/details/firefighter-rl

Cantó-Berná, L. (2023e, marzo). *Entity component system project: Save the ovni.* Descargado de https://archive.org/details/save-the-ovni

Cantó-Berná, L. (2023f, mayo). *The final room game.* Descargado de https://archive.org/details/froom

Colson, D. (2020). *How to make a simple entity-component-system in c++.* Descargado de https://www.david-colson.com/2020/02/09/making-a-simple-ecs.html

Deckhead. (2020). *An entity component system in c++ with data locality.* Descargado de https://indiegamedev.net/2020/05/19/an-entity-component-system-with-data-locality-in-cpp/

Fabian, R. (2018). *Data-oriented design.* Descargado de https://www.dataorienteddesign.com/dodbook/

Gallego-Durán, F. J. (2021, mayo). *Basic linux terminal library.* Descargado de https://archive.org/details/bltl_20210507

Morlan, A. (2019). *A simple entity component system (ecs) [c++].* Descargado de https://austinmorlan.com/posts/entity_component_system/

Retroman, P. (2014). *Youtube channel.* Descargado de https://www.youtube.com/@ProfesorRetroman

Santamaria, R. (2013). *Raylib: A simple and easy-to-use library to enjoy videogames programming.* Descargado de `https://www.raylib.com/index.html`

A. Anexo I: Cómo desarrollar un GameEngine Entity-Component-System en C++20

Este es el producto de esta memoria, el libro "Cómo desarrollar un GameEngine Entity-Component-System en C++20". Puedes leerlo o descargarlo de la siguiente referencia (Cantó-Berná (2023a))