Universitat Oberta de Catalunya
uoc.edu

Vitae: A Videogame Presentation Letter

Javier Riera Chirivella
Plan de Estudios del Estudiante
Área del trabajo final

Consultor:
Jordi Duch Gavaldà

Profesor responsable de la asignatura:
Joan Arnedo Moreno
Profesores colaboradores de la asignatura:
Helio Tejedor Navarro

Fecha Entrega:
6 de Junio de 2021

Dedicado a mi abuelo que, aunque no entienda del todo qué es esto que hago, me anima a seguir con ello cada vez que se presenta la ocasión

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-NoComercial 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-SinObraDerivada 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento-CompartirIgual 3.0 España de Creative Commons

Esta obra está sujeta a una licencia de Reconocimiento 3.0 España de Creative Commons

B) GNU Free Documentation License (GNU FDL)

Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU Free Documentation License".

C) Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Vitae: A Videogame Presentation Letter*
Nombre del autor:	*Javier Riera Chirivella*
Nombre del consultor/a:	*Jordi Duch Gavaldà*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega (mm/aaaa):	*06/2021*
Titulación::	*Máster Universitario en Diseño y Programación de Videojuego*
Área del Trabajo Final:	*Trabajo Final de Máster*
Idioma del trabajo:	*Castellano*
Palabras clave	

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Vitae: A Videogame Presentation Letter es un videojuego de corta duración realizado con la intención de ser mostrado como currículum y muestra de lo que es capaz su creador. Para ello, se utilizan mecánicas ya implementadas en varios juegos anteriores, pero se integran en un marco narrativo en el que el personaje principal es el propio creador del juego, cuyo objetivo es conseguir todas las habilidades necesarias para encontrar un trabajo como programador de videojuegos.

Siguiendo los consejos de Vitae, un hada que le guiará a través de su viaje, Javi conseguirá superar a todos los enemigos que un programador suele encontrarse en la industria, y así completar su formación.

Se trata de un juego pensado para ser rápido, entretenido y fácil de mostrar a posibles empleadores, siendo así jugable en web desde una dirección proporcionada por el creador, para minimizar la barrera de entrada de los empleadores a la hora de ver el currículum.

Abstract (in English, 250 words or less):

Vitae: A Videogame Presentation is a short videogame created with the intent of being used and shown as a CV, and shows what the developer is capable of creating. With that objective, several mechanics that are already implemented in previously created games are used, but they are integrated in a narrative frame in which the main character is the creator of the game himself, whose objective is acquiring all the needed abilities and skills required to find a job as a videogame programmer.

Following the advice from Vitae, a little fairy that will guide him on his journey, Javi will defeat all of the usual enemies that a programmer tends to find in the industry, an in doing so, will complete his learning experience.

This game is intended to be fast, entertaining and easy to show to potential employers. For this, it is playable on a web browser from a web address provided by the creator, minimizing the entry barrier for employers to experience the CV.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Al acabar los estudios, la mayoría de estudiantes nos encontramos de cara al mundo laboral. Es un mundo para muchos desconocido, y además, nos damos de bruces con una competencia por los puestos de trabajo más deseados.
Ya se tengan preferencias por estudios indies y pequeños o por grandes multinacionales, el sector de los videojuegos es uno que la mayoría de familiares y gente de nuestro entorno no suele conocer, así que encontrar la forma de conseguir trabajo se tiene que hacer sin ayuda prácticamente.

Sumándole a eso, tenemos el problema de que cada vez hay más y más gente cualificada que se pelea por esos puestos, y llegamos al momento en que un currículum aburrido se convierte en un currículum desechado. Nadie quiere que el documento en el que ha plasmado toda su experiencia profesional acabe en un cajón o papelera (en nuestro caso normalmente una papelera digital), siendo ignorado. Es por eso que, al encontrarme con la posibilidad de hacer un Trabajo de Fin de Máster de libre elección, desempolvé una vieja idea: Hacer un currículum sobre videojuegos, **en formato videojuego**.

1.2 Objetivos del Trabajo

A grandes rasgos, y de manera directa, este Trabajo tiene dos objetivos claros: Servir como Trabajo de Fin de Máster para aprobar el título y, eventualmente, conseguirme trabajo.

Para que cumpla estos dos objetivos, por tanto, debe plasmar gran parte del conocimiento aprendido a lo largo del tiempo en el campo de la programación de videojuegos, mostrar mi experiencia laboral previa, y destacar los puntos clave que se destacarían de la misma manera en un currículum tradicional.

1.3 Enfoque y método seguido

Hay que tener en cuenta que este acercamiento es más bien poco ortodoxo. Aunque es muy típico poner un enlace a la página de git preferida de la persona que intenta ser contratada, para que se vean sus proyectos hasta la fecha, estos proyectos no suelen estar hechos con la intención de sustituir un currículum tradicional.

En otros campos del sector sí que es más común ver este tipo de material. Muchos estudiantes en campos audiovisuales deciden hacer vídeo-currículums o similares, y varias veces me he cruzado personalmente con estudiantes de publicidad que hacen una campaña personalizada para su propia contratación.

Hay que tener en cuenta, por tanto, que este trabajo tiene que guiarse por unas decisiones de diseño muy diferentes:

- No basta con hacer un proyecto que se pueda jugar sin enterarte de cuál es su objetivo, porque entonces no se estaría consiguiendo informar al jugador de lo que se quiere realizar con dicho juego. Debe ser obvio que la intención del juego es presentarte como candidato al trabajo.
- Debe ser entretenido. Si jugar al juego es un esfuerzo mayor que el de leerse un currículum, se está haciendo algo mal. Recordemos que una de las funciones de este trabajo es destacar por encima del resto de opciones, así que hacer que se acuerden de ti por ser el candidato que les hizo aburrirse durante más tiempo del necesario no es algo que se busque ni desee.
- No ha de ser muy largo. Recordemos que tu trabajo es uno entre muchos otros posibles candidatos, si haces perder más tiempo del necesario, aunque sea divertido, es muy probable que se quede a medias por jugar y no se obtenga la suficiente información sobre ti.

Evidentemente, la manera de proporcionar un producto como este es generando uno desde cero, aplicándole toda la personalidad propia del creador para que se entienda de antemano, incluso antes de una posible entrevista, cuál es la manera de pensar del creador.

1.4 Planificación del Trabajo

Este punto se abordó en la primera entrega de las cuatro que se tuvieron que hacer a lo largo de la asignatura. Para ello, voy a referirme a lo que se planificó en su momento, en el cual se planteó una serie de hitos de avance basados en las entregas que se debían hacer de manera obligatoria. A cada entrega (PEC2, PEC3 y PEC Final) se le asigna un nivel de pulido diferente, y se establecen los parámetros que tienen que cumplirse para que dicho nivel sea alcanzado.
A continuación, se encuentra un extracto de la PEC 1, en el cual se puede ver la estimación original del proyecto y cómo se dividían las tareas en cada uno de sus puntos:

Indispensable (PEC2):

- *Gameplay*: Ser capaz de moverse, disparar, matar enemigos, recibir daño y morir, así como controlar las colisiones con el entorno es estrictamente necesario para poder avanzar con el resto de mecánicas. Enemigos de un solo tipo y sencillos.
- Gráficos: Unos gráficos sencillos, casi de *placeholder*, que hagan que el jugador pueda interpretar lo que sucede.
- Elementos extra: Menú de inicio y pausa.

Esto debería poder realizarse relativamente rápido, y sobre esto se construirá el resto de los elementos.

Secundario (PEC 3):

- *Gameplay*: Poder hacer *dash*, añadir elementos de combate extra como un *dash*, escudo, granadas o teletransporte entre salas. Enemigos extra con mecánicas algo más complejas.
- Gráficos: Personalizar algunos de estos gráficos sencillos iniciales, para poder dar una imagen más personal.
- Elementos extra: Menú de opciones, cuadros de diálogo sencillos.

Terciario (PEC Final):

- *Gameplay*: Puntos de reaparición y división por salas de los objetivos.
- Gráficos: *Portraits* de Javi y Vitae.
- Elementos extra: Cutscene de inicio y final.

Para facilitar su planificación temporal, añado a continuación un diagrama sencillo de Gantt en el que se observan las fechas límite de cada una de las partes de la entrega. Hay que notar que este diagrama no estaba presente en la primera entrega, sino que se ha realizado posteriormente para server como apoyo visual en este punto del document.

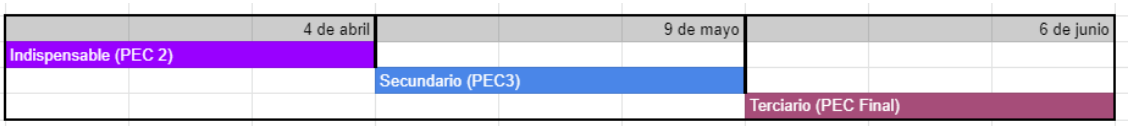

Figura 1 - Diagrama de Gantt con los hitos del Trabajo

1.5 Breve sumario de productos obtenidos

Podemos contar una serie de diferentes sistemas que se han creado para la realización del videojuego, aunque forman en su conjunto el juego completo. Por enumerarlos de manera sencilla y concisa, se podría hacer una lista similar a esta:

- Movimiento y aspecto del jugador
- Movimiento y aspecto de los enemigos
- Características diferentes de los enemigos
- Visualización de la vida del jugador
- Visualización de la vida de los enemigos
- Habilidad de disparo del jugador
- Habilidad de *dash* o acelerón del jugador
- Sistema de *checkpoints* o puntos de guardado
- Sistema de diálogos
- Sistema de habitaciones

1.6 Breve descripción de los otros capítulos de la memoria

En los capítulos posteriores nos centraremos en detallar diferentes partes claves del proyecto. A continuación, se adjunta una lista con una breve descripción de qué se detallará en cada uno de dichos capítulos:

- **Capítulo 2 – Estado del Arte:** Se hará una revisión del estado actual del género al que pertenece el juego, así como las plataformas de desarrollo que se han utilizado para que el juego se complete.
- **Capítulo 3 – Definición del Juego:** En este apartado se pretende presentar una descripción más detallada del juego. Sus elementos narrativos y jugables, y la forma en la que interactúa el jugador con ellos.
- **Capítulo 4 – Diseño técnico:** Para este punto se listarán y explicarán las diferentes herramientas utilizadas, así como todos aquellos *assets* que se hayan o creado o importado de alguna fuente exterior. También se justificará la estructura de software elegida y la Inteligencia Artificial de los enemigos.
- **Capítulo 5 – Diseño de niveles:** Este punto define la estructura en la que están dispuestos los niveles, y muestra cómo se han tomado las decisiones detrás de dicha disposición. Cualquier otro aspecto importante de diseño se describirá aquí de igual manera.
- **Capítulo 6 – Manual de usuario:** Aquí se describirán los requerimientos necesarios para poder jugar al juego, así como instrucciones previas necesarias.
- **Capítulo 7 – Conclusiones:** En este punto final, todo aquello que sea de especial importancia de entre las lecciones aprendidas durante el desarrollo, será explicado. Se trata de un punto más personal en el que se comenta una perspectiva subjetiva pero basada en la experiencia de la creación de este producto particular.

2. Estado del Arte

En este capítulo procederemos a ver cómo funcionan los géneros en los que se puede clasificar el juego, así como su evolución (a gran escala) y su estado actual en la industria.

2.1 Arcade de dos Sticks

Al jugar a Vitae, vemos que el modo de control no es precisamente novedoso. Se trata de una manera de controlar al personaje que ha sido probada en multitud de ocasiones a lo largo de los años en múltiples juegos.

La teoría es simple: Con un stick (o cruceta) se mueve al personaje, y con otro se selecciona la dirección en la que se mira y dispara. Normalmente, y por estándar de industria, estos roles los asumen el stick izquierdo y derecho, respectivamente.

Uno de los primeros juegos en utilizar este esquema [1], sino el primero, fue *Robotron: 2084*. Este juego de 1982 introduce este sistema de control, el cual permite una disociación del posicionamiento del personaje y su capacidad ofensiva. De esta manera, se permite a los diseñadores implementar mecánicas que requieran, simultáneamente, de prestar atención a la precisión del ataque y a la esquiva de enemigos.

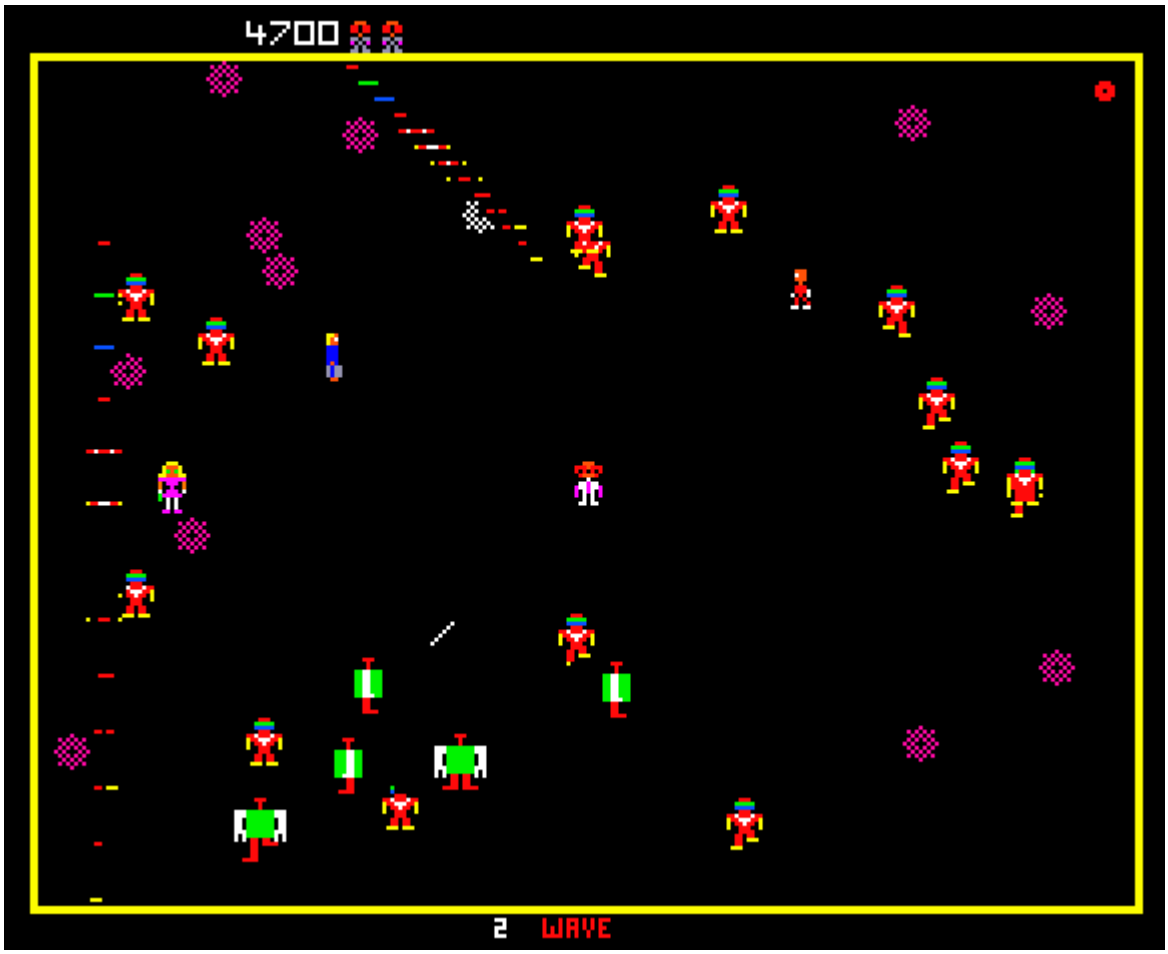

Figura 2 - Robotron: 2084 [1]

Muchos de estos juegos buscan, sin embargo, un punto medio entre estas dos maneras de jugar. Se suele buscar un balance que rechaza la precisión total en el movimiento o en el disparo, optando por rebajar la exigencia en ambos puntos y dejando más margen de error. Como en todos los juegos, y obras en general, este acercamiento se puede romper de forma deliberada y crear retos concretos que se centren mayormente en el movimiento o en la precisión en los disparos.

Uno de los juegos actuales más famosos y que mayor triunfo ha alcanzado de este género es *The Binding of Isaac: Rebirth* y sus posteriores DLCs. En este juego se puede observar muy claramente esta ruptura de las normas, encontrándonos salas como las de la Figura 3. En esta sala, los pinchos hacen daño a nuestro avatar, pero los pinchos se meten bajo el suelo a intervalos constantes. De esta manera, y al no haber enemigos, se enfatiza la navegación por la sala en lugar de la precisión de los disparos.

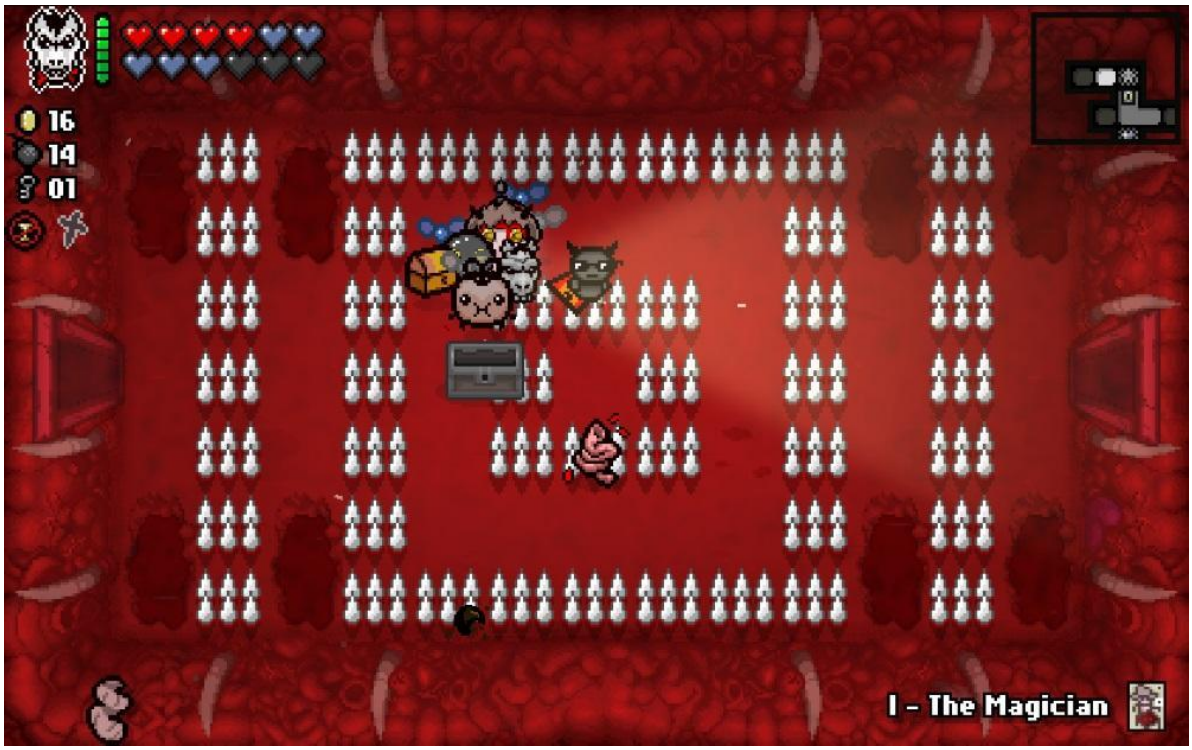

Figura 3 - Sala con énfasis en el movimiento en *The Binding of Isaac* [2]

Varios juegos más han hallado el éxito usando este esquema de controles y comportamiento similar, viéndose entre los más populares algunos títulos como *Nuclear Throne*, *Enter the Gungeon*, *Wizard of* Legendi o *Nova Drift*.

En el caso de este proyecto, los controles no serán de doble stick estrictamente, pues se jugará con ratón y teclado para mejorar su uso desde un ordenador en una web, pero se mantiene la idea de disociar el movimiento de los disparos. En este caso, las teclas WASD servirán para moverse y el ratón apuntará.

2.2 RPG

Dado que en *Vitae* asumes el control de Javi, un recién graduado de diseño y programación de videojuegos, y actúas desde su punto de vista, se puede asignar la etiqueta de RPG a este título. Esto es así porque RPG significa *Role Playing Game*, o juego de rol, en el que el jugador se pone en la piel de un personaje y vive la historia y el mundo de forma vicaria a través de dicho avatar.

Aunque a día de hoy esta etiqueta se asocia muchas veces a juegos en los que la personalidad y decisiones del personaje son elegidas directamente por el jugador, en lugar de venir predeterminadas por el equipo de desarrollo, no siempre ha sido el caso, ni lo sigue siendo en todas las ocasiones.

Uno de los ejemplos más famosos de juego RPG, en un nivel muy simple y sencillo del mismo, serían las diferentes entregas de los juegos de la rama principal de *Pokémon*.

En ellos, el jugador tiene poco más control sobre el personaje que su apariencia, con la famosa pregunta *“¿Eres chico o chica?”*.

En juegos guiados en mayor medida por la historia, como los *Final Fantasy*, las decisiones son normalmente también menores en importancia, pero la historia es mucho más compleja y los personajes están más desarrollados.

Un elemento que tienen en común todos estos juegos RPG es la capacidad de que los personajes que nos vamos cruzando tengan la capacidad de comunicarse con nuestro/s personaje/s y entre ellos. Muchas de las veces, esta comunicación se realiza a través de cuadros de diálogo, que pueden ir acompañados de dibujos o imágenes y nombres de los personajes que hablan, para que se sepa quién está diciendo la frase.
Muchas veces, además, estos dibujos o imágenes representan al personaje con una expresión facial que nos aporta información adicional sobre cómo leer el texto, dándonos a entender sus sentimientos de manera visual y no escrita.

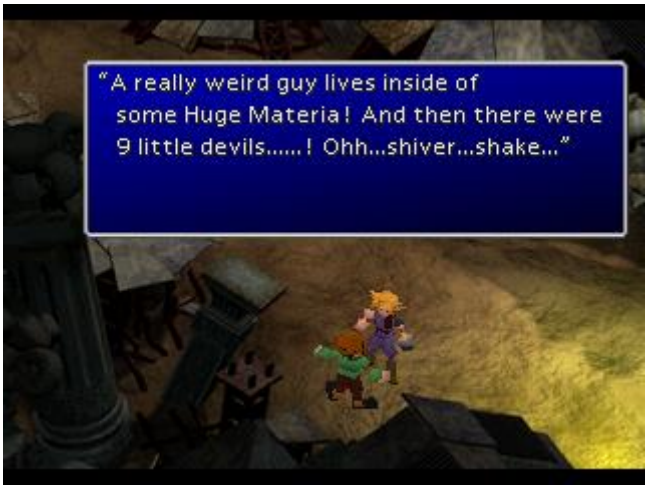

Figura 4 - Cuadros de diálogo en el *Final Fantasy VII* [3]

A día de hoy, los juegos RPG se han dividido entre juegos RPG Japoneses (JRPG) y Occidentales, centrándose en diferentes mecánicas y experiencias de juego, pero manteniendo el énfasis en la historia y cómo se desarrolla la misma alrededor del personaje principal y sus allegados.

Podemos destacar, de entre los últimos años:

- JRPG: Final Fantasy VII Remake, Final Fantasy XV, Dark Souls
- RPG Occidental: Skyrim, Pillars of Eternity, The Witcher 3

3. Definición del juego

3.1 Breve descripción del juego

Despiertas en un espacio desconocido, escuchando una voz desconocida. Esta voz pertenece a un hada que se presenta como *Vitae*, y que proclama que está ahí para ayudarte y entrenarte en tu búsqueda de trabajo en la industria. Tendrás que demostrar que has aprendido suficiente en tus años de formación, y aplicar este conocimiento en una versión *gamificada* de un entrenamiento para mejorar como creador de videojuegos.

Guiado por ella, y entablando conversación cada vez que algo es llamativo, conseguirás vencer a varios tipos de enemigos gracias a las habilidades que te aprendas a utilizar en esta realidad extraña.

3.2 Personajes

En este juego solo hay dos personajes, Javi y Vitae:

- **Javi**: Javi es un recién graduado que desea hacer videojuegos, pero al que le falta mucho por aprender. Se mueve por el entorno y lucha contra los enemigos y barreras que le impiden llegar a su objetivo, a la vez que aprende nuevas habilidades que le permiten superar esas barreras.
- **Vitae**: Vitae es un hada bastante molesta, quizá algo cliché y que roza la infracción de derechos de autor, pero ayuda a Javi a entender qué sucede y que sirve para saber hacia dónde está apuntando con sus proyectiles. Incluso si a veces ni siquiera ella misma sabe qué está pasando…

3.3 Interacción entre los personajes

La relación de estos dos personajes es lo que transmite la información al jugador. En cada conversación se puede aprender algo nuevo, a veces a nivel técnico y a veces a nivel de historia sobre el entorno en que se desarrolla el juego.

Hay cierto desdén al principio por parte de Vitae hacia Javi, pero conforme este va avanzando y mejorando, ella se suaviza y empieza a tener esperanza en que puede que sí consiga su objetivo.

Además de esto, dado que ambos son, en el fondo, una pareja de *frikis*, soltarán pequeñas referencias y guiños a la cultura popular de los videojuegos, porque no pueden evitarlo.

El jugador verá esta interacción a través de diálogos programados para ocurrir en diferentes puntos concretos del mapa y tras eventos específicos, y se mostrarán en cuadros de texto típicos de los juegos RPG.

3.4 Conceptos iniciales

Varias ideas se han quedado en el tintero. Diferentes tipos de disparos, enemigos y habilidades se han desechado por falta de tiempo, complejidad o incoherencia con la temática o la corta duración que se pretendía dar al juego.

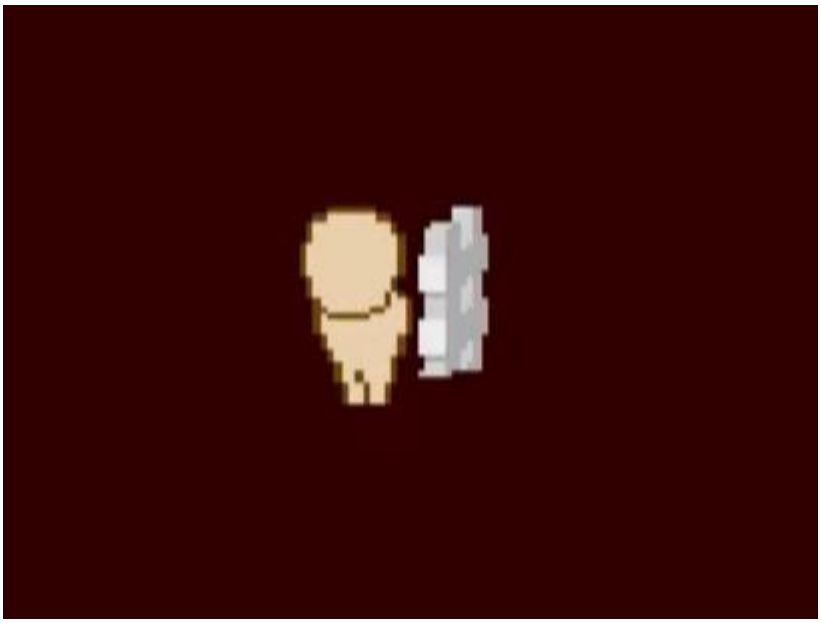

Figura 5 - Concept art en el que se ve usar la almohadilla como escudo, con maniquí de internet que acabo siendo definitivo [4]

En la Figura 5 se puede observar un prototipo de escudo que se creó cuando la posibilidad de enemigos que disparasen. La manera concreta en que iba a funcionar no estaba del todo claro, alternándose entre estar perpetuamente a la espalda o poder colocarse en el suelo. Esta última opción daba problemas con el *pathfinding* de los enemigos, así que se tuvo que desechar.
Muchas de las ideas que no se han llegado a implementar servirían para potenciar la parte de combate, como se puede ver en la Figura 6, en la que se puede ver como hay cantidad de armas, tipos de disparos y maneras de modificar la manera de hacer daño.

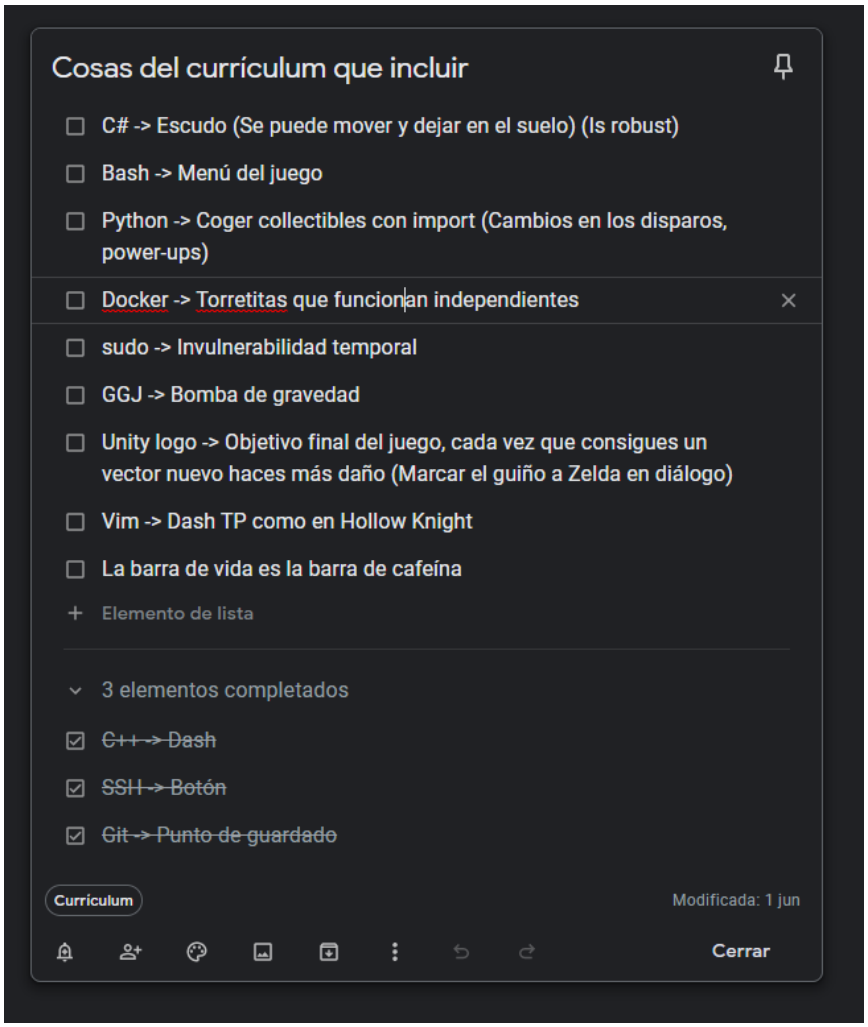

Figura 6 - Ideas guardadas durante el proceso de lluvia de ideas

Podemos ver incluso una de las ideas, la de la implementación de Docker, como pequeñas torretas. Para este caso en concreto incluso se buscó arte concreto, y se llegó a la decisión de que se utilizarían las cabezas de torreta que se ven en la Figura 7.

Figura 7 - Arte de torretas en fase de concepto, sacado de OpenGameArt [9]

Para el arte de los personajes, en concreto en los retratos que salen junto al texto en los cuadros de diálogo, se buscaba un estilo pixelart, y se utilizaron fotos de referencia, siendo muy notable la de Navi, de la saga The Legend of Zelda. Se usa esta referencia porque *Vitae* es una parodia de Navi, o al menos de la idea que tiene internet de Navi, basándose en lo repetitiva y hasta pesada que puede resultarle al jugador.

Figura 8 - Imagen original de Navi, del Zelda [5]

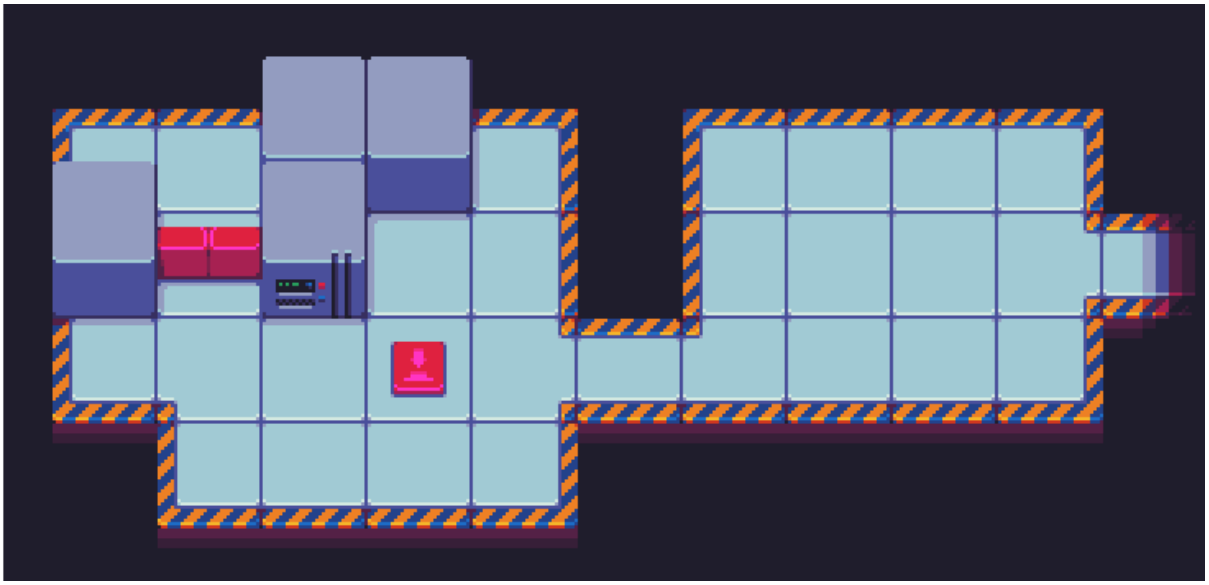

Figura 9 - Arte final para el entorno [6]

En cuanto a los tilesets, se decidió utilizar el arte de la Figura 9, el cual daba una estética de ciencia ficción que cuadraba mucho con la ambientación que se pretendía del juego. De esta manera, se consigue un entorno limpio, casi médico, y con una sensación de modernidad gracias a los ordenadores y terminales que terminan de dar ese aspecto de estar en *Matrix*.

4. Diseño técnico

4.1 Entorno elegido

Para la realización de este juego se ha escogido usar el motor gráfico de videojuegos Unity. Además de ser uno de los motores predominantes en la industria, es de acceso gratuito y permite hacer juegos sin cobrar royalties hasta cierto umbral de ganancias.

Además de eso, es una herramienta que conozco y llevo usando y estudiando años, y he aprendido a trabajar de manera adecuadamente eficiente como para avanzar a un ritmo al que me siento cómodo.

No solo eso, sino que además el hecho de usar un motor gráfico en primer lugar provee una agradecida abstracción, sobre todo en conceptos algo más abstractos o que domino menos como los efectos de partículas o las colisiones.

Aunque otra opción a considerar, en este aspecto, fue Godot, al final se desechó. Tenía una cantidad considerable de puntos a favor:

- Código abierto.
- Totalmente gratuito, sin tasas en ningún momento.
- Altamente compatible con Linux.
- Información disponible en internet muy actualizada.

Por desgracia, el simple hecho de no conocer la tecnología suficiente y no encontrar información sobre cómo hacer una *build* web, fue suficiente como para decantarme por no usarlo.

4.2 Requerimientos técnicos del entorno

4.2.1 Unity

La instalación de Unity es bastante sencilla y se recomienda realizarla a través de su lanzador Unity Hub, el cual facilita la instalación simultánea de varias versiones. Esto es especialmente útil para realizar el trabajo a la vez que se realizan los trabajos del resto de asignaturas, los cuales varias veces requieren de versiones específicas del editor.

A nivel *software*, el editor de Unity necesita, según su página web [7], los siguientes requisitos mostrados en la Figura 10:

Minimum requirements	Windows	macOS	Linux (Support in Preview)
Operating system version	Windows 7 (SP1+) and Windows 10, 64-bit versions only.	High Sierra 10.13+	Ubuntu 20.04, Ubuntu 18.04, and CentOS 7
CPU	X64 architecture with SSE2 instruction set support	X64 architecture with SSE2 instruction set support	X64 architecture with SSE2 instruction set support
Graphics API	DX10, DX11, and DX12-capable GPUs	Metal-capable Intel and AMD GPUs	OpenGL 3.2+ or Vulkan-capable, Nvidia and AMD GPUs.
Additional requirements	Hardware vendor officially supported drivers	Apple officially supported drivers	Gnome desktop environment running on top of X11 windowing system, Nvidia official proprietary graphics driver or AMD Mesa graphics driver. Other configuration and user environment as provided stock with the supported distribution (Kernel, Compositor, etc.)
	For all operating systems, the Unity Editor is supported on workstations or laptop form factors, running without emulation, container or compatibility layer.		

Figura 10 - Tabla de Requisitos de Unity Editor [7]

A nivel *hardware* no se especifica ningún requisito concreto, pero cabe destacar que una máquina más potente hará que los tiempos de compilación y carga de escenas sean más rápidos y, por tanto, el trabajo fluya mejor.

4.2.2 Sistema Operativo

El sistema operativo elegido fue Linux, en concreto la distribución de Arch. El sistema de distribución de paquetes por parte de la comunidad facilitó mucho la instalación de todas las herramientas que se usaron.

Se utilizaron los drivers gráficos proprietarios de Nvidia por su mayor compatibilidad y facilidad de uso e instalación.

4.2.3 *Hosting* online

Para el *hosting* de las versiones web se utilizó el propio proyecto git de GitHub en el que se guardan los cambios del proyecto, gracias a la utilidad de GitHub Pages que se ofrece como servicio gratuito.

Además, la manera de funcionar de GitHub Pages ha cambiado desde que se introdujo esta funcionalidad, siendo ahora mucho más fácil de utilizar y sin la necesidad de crear una rama específica en la que colocar los archivos que se quieran mostrar en la página.

4.3 Inventario de las herramientas empleadas

En este proyecto, al tratarse de la creación de un juego desde 0, han hecho falta varias herramientas diferentes. A continuación se ve una lista con las más importantes o de mayor notoriedad, así como una pequeña descripción de su uso y su utilidad:

- **Unity:** Esta herramienta ya se ha explicado en el punto 4.1, pero por resumir, es un motor gráfico orientado a la creación de videojuegos, y viene con su propio editor para realizar esta tarea.
- **Visual Studio Code:** VS Code, como se suele abreviar, es un editor de texto modular, con muchas extensiones posibles que le dan ciertas funcionalidades de IDE (Entorno Integrado de Desarrollo). Gracias a dichas extensiones, se ha modificado para que acepte autocompletado, *snippets* (trozos de código comunes preparados para escribirse con atajos), y señalización de errores sintácticas en C#.
  Además de eso, se ha añadido una extensión que permite el uso de los atajos de teclado y la manera de trabajar propia del editor de texto Vim, el cual permite editar código de manera mucho más rápida una vez el usuario se acostumbra a dicha forma de editar.
- **Git:** Como forma de almacenamiento persistente y aditivo, y como control de versiones, se ha usado Git. Aunque hay otras herramientas como SVN, Git sigue siendo la manera preferida por millones de usuarios para mantener sus proyectos seguros y a prueba de errores que un control de versiones pueda solucionar.
  En concreto, se ha utilizado el proveedor GitHub, tanto por ubiquidad en el mercado como por la herramienta de GitHub Pages que se ha explicado en el punto 4.2.3.
- **Gimp:** Gimp es un editor de imágenes muy potente de código abierto. Mucha gente lo describiría como un *Photoshop* gratuito, y en parte tendrían razón. Se ha utilizado principalmente a la hora de realizar sprites necesarios para el juego, a base de escalar imágenes disponibles en la web y retocar sus colores y transparencias para coincidir con la estética que se buscaba.
  También se ha utilizado para la edición general de cualquier imagen del juego que no coincidiese del todo con lo que se buscaba, como

fue el caso con los paneles de diálogo y su inclinación y color, que fueron alterados usando esta herramienta.

- **PixelMe:** Esta página web [8] se usó para la creación del *sprite* pixelado que se usa como retrato de Javi. Se probó a usarlo para otras imágenes pero parece ser que funciona principalmente con caras, y no con cualquier imagen, así que este uso fue el único que se le dio, aunque es notable debido a la calidad que se obtuvo. Los resultados se pueden ver en la Figura 11, más abajo.

Figura 11 - A la izquierda, la imagen original. A la derecha, la imagen procesada por PixelMe

4.4 *Assets* usados en el juego

No todos los recursos del proyecto se han creado para el mismo, ni han sido integrados en alguna herramienta que se haya nombrado ya, como el propio editor de Unity. Muchos de los *sprites* utilizados son originarios de internet, o varios sistemas de código han sido sacados de APIs gratuitas *online*.

Vamos a hacer una división entre recursos gráficos y recursos de código en dos puntos a continuación:

4.4.1 Recursos gráficos

Obtenidos de internet

Para hablar de los diferentes gráficos usados, podríamos comenzar por el propio logo. Dado que no soy un diseñador, y la tarea de diseñar logos es muy complicada, acudí a la búsqueda de imágenes de Google para encontrar logos sencillos pero modernos, los cuales tuviesen conexión con los Currículums, pues es al fin y al cabo lo que es este proyecto. Me crucé con el logo de la Figura 12:

Figura 12 - Logo de Vitae [10]

Este logo es moderno, simple, y muestra la C y la V de **C**urrículum **V**itae. Podría haberse pixelado, pero en el momento no me sentía aún cómodo con esa manera de trabajar y al final del proyecto simplemente estaba ya demasiado asentado como para cambiarse.

Muchos de los otros elementos gráficos que se han usado se han mostrado ya, pues a partir de la fase de concepto no hizo demasiada falta modificar lo que se

decidió. Se puede destacar el maniquí de 8 direcciones y animaciones tando de *idle* como de movimiento que se encuentran en la referencia [4], y el tileset utilizado de sci-fi de la referencia [6].

Se usó para los cuadros de diálogo una caja semitransparente a la que se le tuvieron que hacer unos pequeños cambios, pero que provenían de OpenGameArt [11] inicialmente.

Figura 13 - Cuadro de diálogo original

También se utilizó dicho cuadro para crear un fondo sobre el que colocar los retratos de los personajes, tal y como se ve en la Figura 14:

Figura 14 - Fondo de retrato para los diálogos

Para mostrar la vida del jugador, se utilizó arte de una barra de vida dividida en corazones [12], se modificó para que estuviesen espaciados de la misma manera, y se utilizaron para indicar tanto corazones llenos como vacíos.

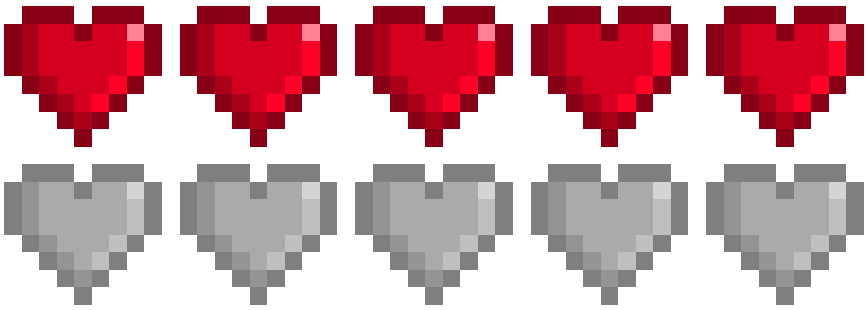

Figura 15 - Barra de vida del jugador, llena y vacía

Para la mayoría de los botones que se ven a lo largo del juego (Git, C++, C#, Unity) se obtuvieron imágenes de internet de las respectivas fuentes más fiables y se pixelaron, modificaron los colores y transparencias, hasta que fuesen arte de 32x32 píxeles la unidad. Estos gráficos se pueden ver en la Figura 16:

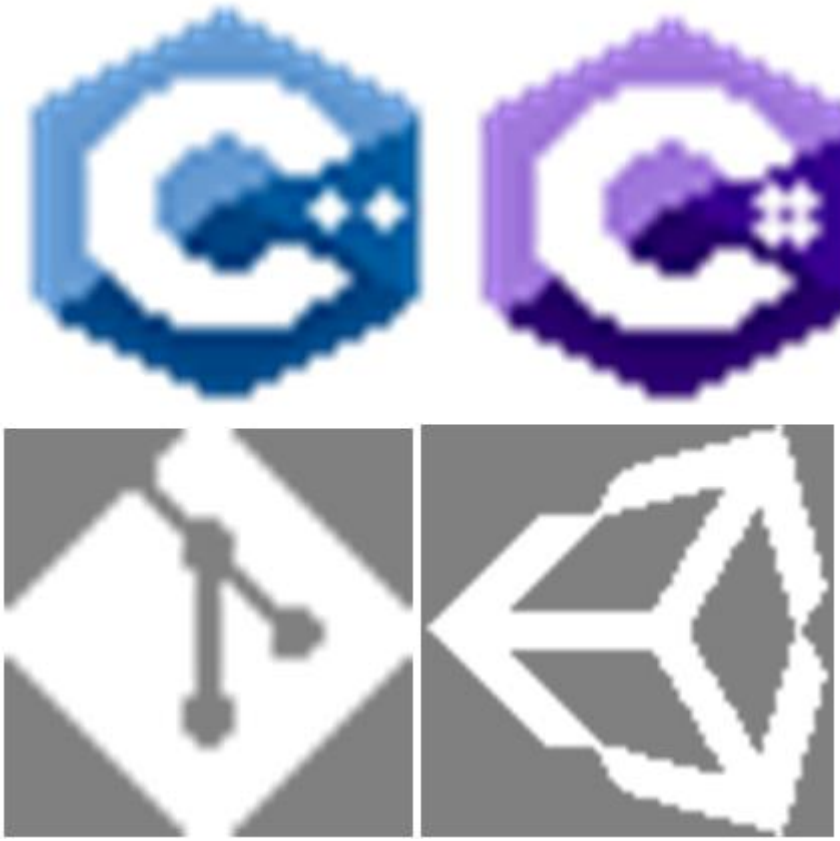

Figura 16 - Iconos pixelados

De la misma forma, esto mismo se hizo para los enemigos, dando como resultado los siguientes *sprites* de la Figura 17:

Figura 17 - *Sprites* de los enemigos

No todos los gráficos se obtuvieron de internet, eso sí.
Se utilizó Gimp para crear, desde cero, la almohadilla correspondiente al ataque de C#, la cual rota como si fuese una estrella ninja mientras avanza hacia el objetivo. Se puede observar el dibujo en la Figura 18 a continuación:

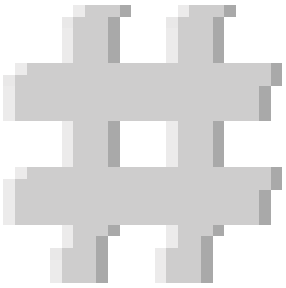

Figura 18 - Almohadilla de C#, que sirve de arma

También se creó simplemente utilizando este programa el icono que haría las veces de botón de SSH:

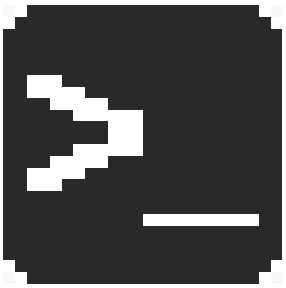

Figura 19 - Arte para el botón de SSH

A nivel fuente de texto, se ha utilizado una en particular para este proyecto: Retro Gaming. Esta fuente se puede encontrar en la entrada de bibliografía [13].

4.4.2 Recursos de código

En este apartado se van a mostrar los recursos de código que se han utilizado, así como los que han servido de inspiración para la realización del proyecto.

Primero, podemos comenzar viendo una de las partes principales del juego: El ***Pathfinding***.

Dado que Unity ofrece una capacidad de ofrecer IA al movimiento de un ente **solamente** en 3D, hay que recurrir a otras opciones para realizar este tipo de movimiento en 2D. Para ello, una de las opciones a las que más se recurre es un *pack* llamado *"A* Pathfinding Project"* [14]. Este es un proyecto muy completo y con una versión gratuita que permite al desarrollador implementar varias zonas en las que moverse, varios tamaños de nodos, velocidades de los agentes para frenada, aceleración, giro… Y una gran cantidad de parámetros más.

Este ha sido un recurso muy completo, incluso en su versión de prueba, aunque para proyectos más grandes la versión gratuita no sería suficiente. Esto se debe a que muchas funcionalidades muy importantes se quedan fuera de esta versión, como la posibilidad de que los agentes se eviten entre ellos o el refresco dinámico de los obstáculos y por tanto de las zonas caminables.

Otro elemento que se planteó usar fue el mencionado en un artículo de Gamasutra [15], en el cual se exponía una manera de mostrar el texto de manera progresiva, similar a como se ha acabado haciendo en los cuadros de diálogo.

Por desgracia, no se pudo hacer que funcionase como era debido, así que se tuvo que abandonar en pos de un script propio, más sencillo, pero más eficaz.

4.5 Estructura de Software

Esta es la parte más técnica del proyecto. Gracias al uso de ScriptableObjects, Herencia y diseño funcional orientado al proyecto, se ha conseguido dividir el funcionamiento del juego en varios sistemas que funcionan en cohesión, pero que no son inherentemente necesarios entre ellos, excepto en algunos casos.

Comenzaremos hablando de uno de los más sencillos de explicar, pero que más técnicas utilizan: Los enemigos.

4.5.1 Enemigos

Los enemigos parten de una base muy sencilla: Una clase abstracta llamada *Enemy*. Esta clase tiene elementos que han de ser comunes para todos los enemigos, como un evento de Unity llamado OnDie(), una variable que indica su vida actual, elementos necesarios para moverse por el mapa, y funciones como Recibir Daño (Damage) o Morir (Die), así como una función especial llamada Spawn que permite poner todos los parámetros en orden antes de que se pueda utilizar el enemigo de manera correcta.

Cabe destacar que todos los métodos que hemos nombrado son de tipo *virtual*, y de visibilidad mímina *protected*. Esto asegura que todos los *scripts* que hereden de esta clase tengan posibilidad de utilizarlos tal cual funcionan, es decir, con el comportamiento predeterminado, o con su comportamiento propio. Esto es así porque un enemigo bomba no muere igual que un enemigo normal, sino que lo hace explotando y por tanto debe tener una implementación diferente de la función Die.

Además de esto, todo enemigo tiene asociado un ScriptableObject llamado EnemyData, en el cual se guardan instancias de diferentes tipos de cualidades según el enemigo al que hagan referencia. Un enemigo básico se moverá más lento y hará menos daño que uno avanzado. De esta manera, y utilizando ScriptableObjects, se pueden generar enemigos con comportamientos similares, pero cualidades diferentes de manera muy rápida y eficaz.

En la Figura 20 se puede ver un diagrama UML sencillo de cómo funciona esta estructura:

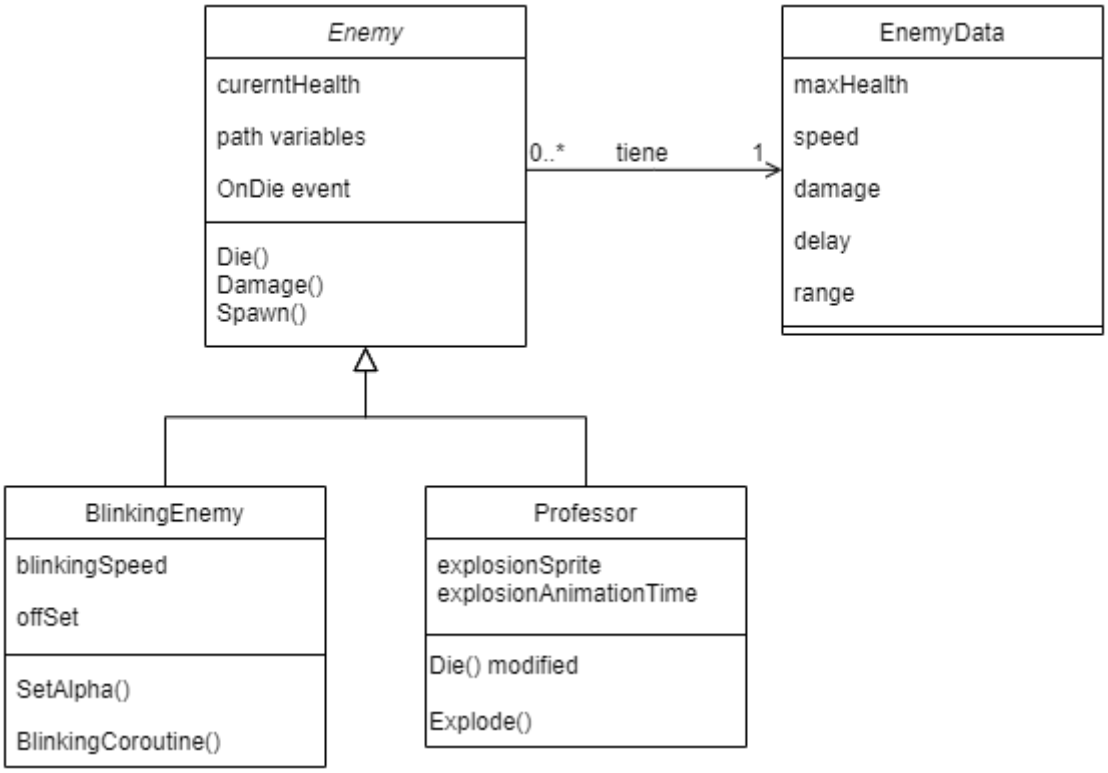

Figura 20 - Estructura de código de enemigos

4.5.2 Jugador

La forma en la que el jugador está creado es muy similar a la de los enemigos. Al igual que en el caso de los enemigos, se hace uso de ScriptableObjects para guardar su vida máxima, velocidad, y parámetros del *dash*. Sin embargo, también se usa de una forma particular. Para evitar la sobredependencia de unos *scripts* con otros, se guardan en este ScriptableObject una serie de parámetros que son modificados directamente por el usuario a través de un script de input. Estas variables son tanto la dirección en la que se mira como en la que se apunta.

Guardando estos valores aquí, se puede conseguir que, sin necesidad de una conexión directa entre el *script* de *dash*, el de movimiento y el de input, cada uno de ellos pueda trabajar de manera independiente sin problema. El *Dash* puede modificar la variable *isDashing* dentro del PlayerData para que el script de movimiento sepa que no tiene que aplicar movimiento, y de la misma manera el *dash* se efectúa en la dirección correspondiente (donde se mire o hacia donde se ande, dependiendo de si el jugador está parado o no) sin tener que hacer una *query* de información directa al movimiento.

Otra ventaja de este acercamiento es la posibilidad de añadir más de un jugador. Aunque no era la intención, esta manera de trabajar podría ser reutilizada en un futuro proyecto, y dado que no ralentizaba el proyecto, sino que simplemente le otorgaba más orden, se decidió hacerlo de esta manera.

No solo eso, sino que hay varios sistemas más que se hacen servir de esta estructura. Tanto el script de animación del jugador, como el de Vitae (que está orbitando apuntando al sitio donde irán los disparos), como de disparo, tienen una referencia a este ScriptableObject y así se evitan los típicos códigos de espagueti llenos de referencias nulas.

Un punto importante de este sistema es que se pueden desactivar tanto el *script* de *dash* como el GameObject de disparo, haciendo posible que estas habilidades se obtengan fácilmente y de manera permanente conforme se avanza en el juego. Otra forma de haber hecho esto habría sido mediante dos variables en el ScriptableObject, pero dado que solo eran dos habilidades y que no iban a quitársele al jugador en ningún momento, este acercamiento más rápido y sencillo funciona a la perfección.

A continuación, en la Figura 21, se muestra una imagen de cómo se estructura este objeto Player.

Figura 21 - Estructura del objeto Player

Podemos ver cómo todos los elementos que deben mantenerse en la misma posición relativa siempre con respecto al jugador descienden del mismo. De esta manera, al ahijarlos a él, se evita la creación innecesaria de *scripts* de seguimiento de posición, incluido para la cámara.

4.5.3 Habitaciones

Cada habitación tiene una serie de cosas:

- Un *tilemap* independiente para el suelo, los obstáculos y las partes que quedan por encima del jugador, respectivamente.
- Una serie de puertas tanto de entrada como de salida, que pueden ser ninguna, una o varias.
- Botón de activación (opcional).
- Enemigos (opcionales).
- Acciones a realizar al completar la habitación, como mostrar un diálogo, hacer aparecer un punto de guardado… etc (opcionales).

Veamos en la Figura 22 cómo se organiza esta habitación de ejemplo:

Figura 22 - Estructura de habitaciones y puertas

Se observa cómo hay un *SpawnPoint*, que corresponde al sitio donde aparece el jugador por defecto al comenzar el juego, un botón de inicio que hace que se abra la puerta y un objeto que es un botón modificado y que sirve para empezar una cutscene al llegar a un *collider*.

En las habitaciones en las que hay enemigos, además, hay un objeto llamado *Enemies* bajo el cual se anidan los enemigos que vaya a haber en la sala. De esta manera, el *script* de la habitación puede hacer clones de ellos cada vez que haya que resetear la habitación.

Cuando se resetea una habitación, se eliminan los enemigos *spawneados*, se cierran las puertas de salida y se espera a que se vuelva a activar la habitación para cerrar todas las puertas y hacer aparecer de nuevo a los enemigos.

Este comportamiento es así si la sala no ha sido completada, pues en caso contrario, el reseteo solo se asegura de que las puertas de entrada y salida hayan sido abiertas, y no se hacen aparecer enemigos de nuevo.

Control de las habitaciones

Hay que tener en cuenta que hay un controlador de juego, un GameController, que se encarga de que, cuando el jugador muera, todas las habitaciones se reseteen. Por cómo hemos visto que funciona esto en la parte anterior, las habitaciones que hayan sido ya completadas se abrirán por completo, mientras que aquellas que no harán desaparecer a todos sus enemigos *spawneados* y cerrarán sus puertas de salida.

4.5.4 Puertas

A cada puerta se le asigna un nombre, que normalmente son “Door”, seguido del nombre de la sala que precede a la puerta, un guion, y el nombre de la sala a la que lleva.

Cada GameObject de puerta tiene un *script* que permite que la puerta se cierre o abra, y esto lo hace a base de desactivar o activar los *tilemaps* de obstáculos y *foreground*, dejando así que el jugador pueda pasar o no.

4.5.5 Git y sistema de *Checkpoints*

Una de las maneras de asegurarnos de que el jugador puede pasarse el juego sin problemas es utilizar puntos de guardado. Estos vienen en forma de botones con el logo de Git, y el jugador reaparece en el último que haya tocado con la vida completa.

Para ello hay un sistema puesto en juego relativamente sencillo. Un controlador de Respawn del personaje tiene guardado en una variable la última posición de checkpoint que haya tocado el jugador, que se le manda a través de un evento que se activa en cada botón de git al ser tocado por el jugador.

Cuando el jugador muere y, por tanto, activa su evento OnDie(), el controlador de Respawn toma nota de esto y le hace reaparecer en el último checkpoint que tenga guardado.

4.5.6 Barras de Vida

Las barras de vida, aunque tienen el mismo propósito, se comportan de manera diferente en los enemigos que en jugador. Esto se debe principalmente a que la barra de vida del jugador pertenece a los elementos fijos de la UI, que se pueden colocar en un *Canvas* global, mientras que en el caso de los enemigos se usan sprites.

Para el jugador se usa un elemento *Image* que tiene un comportamiento de llenado de izquierda a derecha, y es precisamente esto lo que se encarga de actualizar un *script* propio que calcula el porcentaje entre la vida actual del jugador y su vida total.

Para los enemigos, en cambio, se usa un Sprite con forma cuadrada deformado y con anclaje en la parte izquierda, haciendo así que al variar su tamaño horizontal, crezca solo hacia la izquierda, dando sensación de llenado. Se usa un *script* muy similar al de la vida del jugador, pero que funciona independientemente con cada enemigo.

Hay que destacar que, en el caso de los enemigos que son punto y comas, las barras de vida debían desaparecer también, para no darle pistas al jugador de dónde se encuentra. De esta manera, en el *script* del enemigo parpadeante, se añaden referencias a los elementos de la barra de vida y se les cambia también la transparencia.

4.5.7 *Dash*

El *dash* es una habilidad muy importante del jugador. Le permite moverse a gran velocidad y ser invulnerable durante ese mismo tiempo. Para esto, cada vez que se inicia un *dash* se eliminan las interacciones entre las capas de enemigos y jugador, de tal manera que el jugador los pueda atravesar sin miedo a hacerse daño. También se activa una variable llamada *isDashing* en el

PlayerData del jugador, para que el resto de scripts sepan que se está en mitad de un *dash*.

Cuando el *dash* se acaba, o se choca el jugador con una pared, las colisiones entre las capas se vuelven a activar, haciendo que el jugador deje de ser invulnerable. También se actualiza el valor de la variable en PlayerData.

Dado que, una vez se estaba ya en contacto con la pared, se podía hacer un *dash* contra la misma sin que se parase, se añadió una comprobación sobre el ángulo en que se quería hacer. Si había una pared en menos de una distancia determinada en esa dirección, el *dash* no se hacía.

Cabe destacar también que el *dash* funciona de dos maneras diferentes:
Si se está moviendo el personaje, se hará en la dirección en la que se mueva. Si no, se hará en la dirección en que se mire.
Para esto, se hace una comprobación de si la velocidad de movimiento en el PlayerData es mayor que un umbral concreto y, en caso de que lo sea, se hace en esa dirección, y se hace en la dirección en la que se mira en caso contrario.

6. Diseño de niveles

En este apartado vamos a ver, una a una, las habitaciones que componen el mapa de *Vitae*, y vamos a explicar el porqué se han hecho de esta manera.

Para comenzar, veamos un mapa numerado de todas las habitaciones:

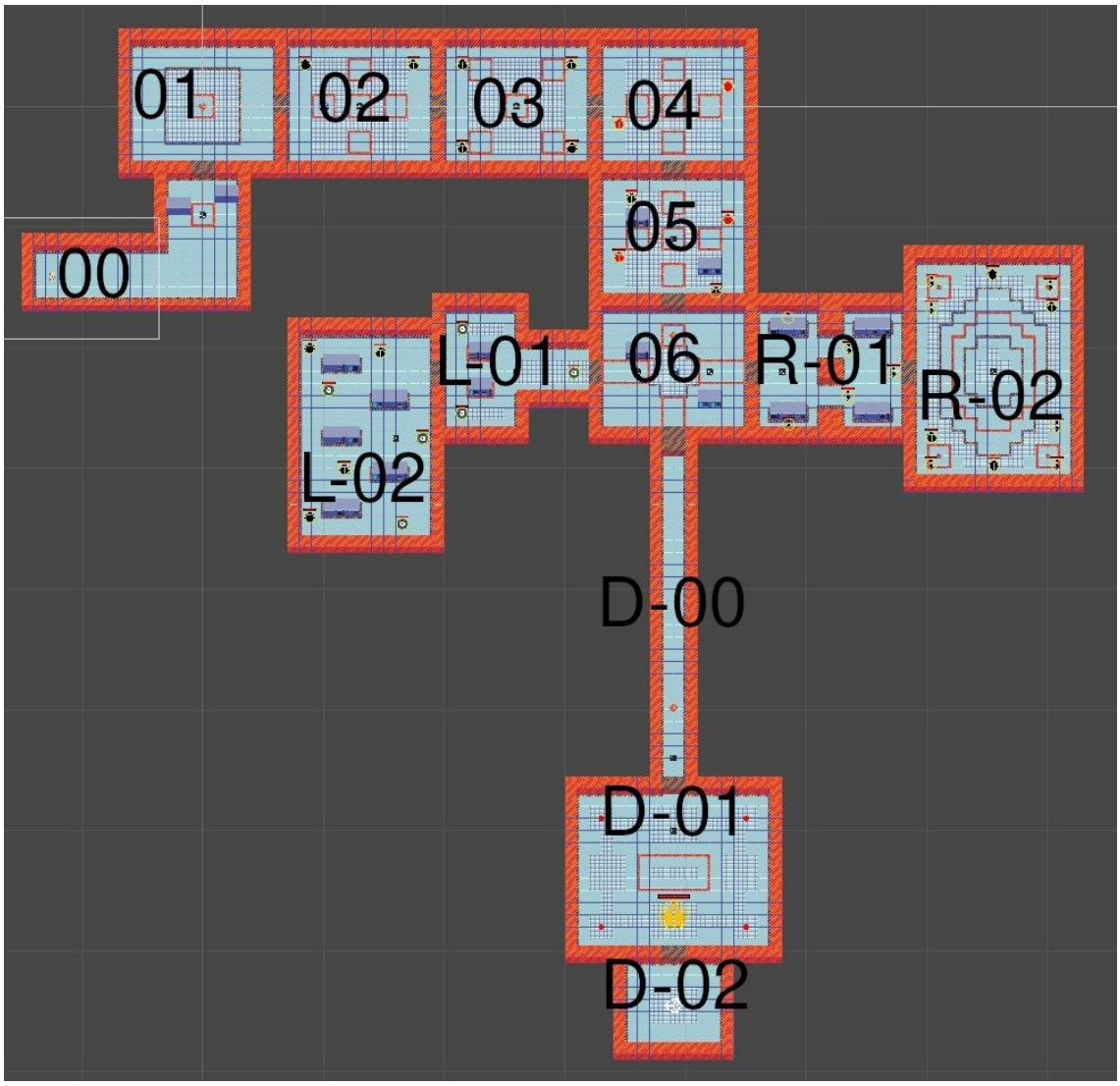

Figura 23 - Mapa completo

Como vista general, podemos observar que el jugador empieza en la habitación 00, en el lado izquierdo. Va avanzando hacia la derecha hasta que hay un giro hacia abajo, y finalmente se llega a una habitación que tiene la puerta hacia abajo cerrada, pero dos puertas hacia los lados que parecen abrirse con un botón normal. Cuando ambas alas del mapa se hayan completado, las dos puertas de abajo se abrirán, llevando a la habitación final con el jefe y, posteriormente, al objeto final del juego y a su fin.

6.1 Sala 00

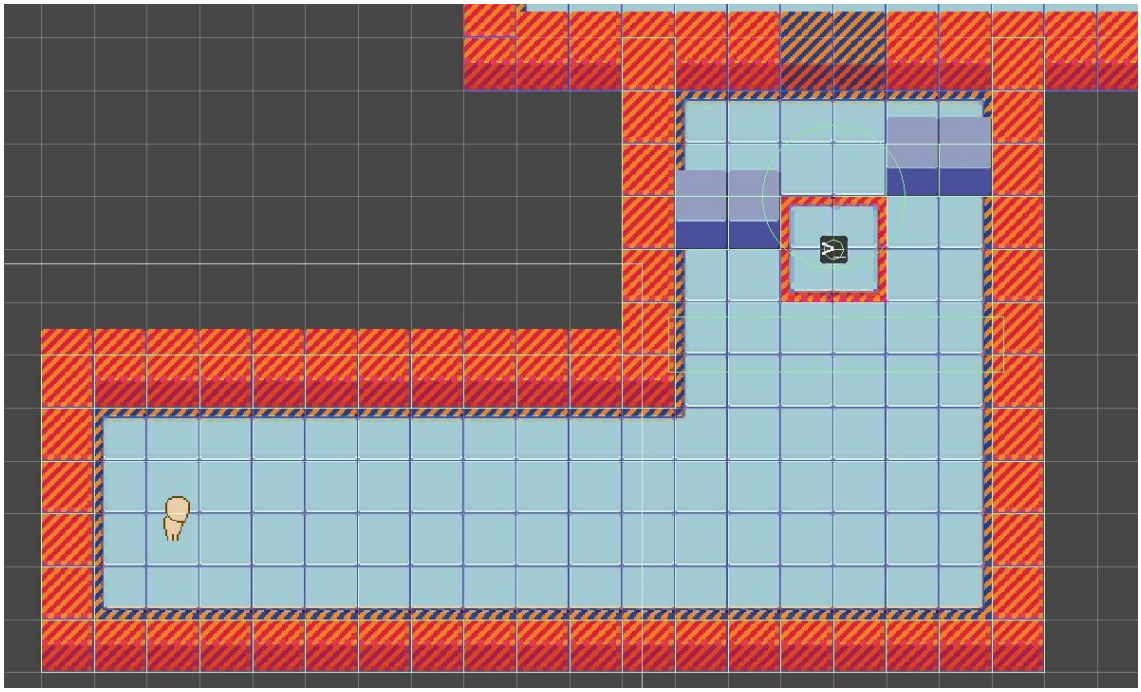

Figura 24 - Sala 00

Esta sala es muy sencilla. Es donde el jugador despierta y es introducido a Vitae. Ella le ayuda a moverse diciéndole los controles y, finalmente, le enseña a abrir puertas usando SSH.

6.2 Sala 01

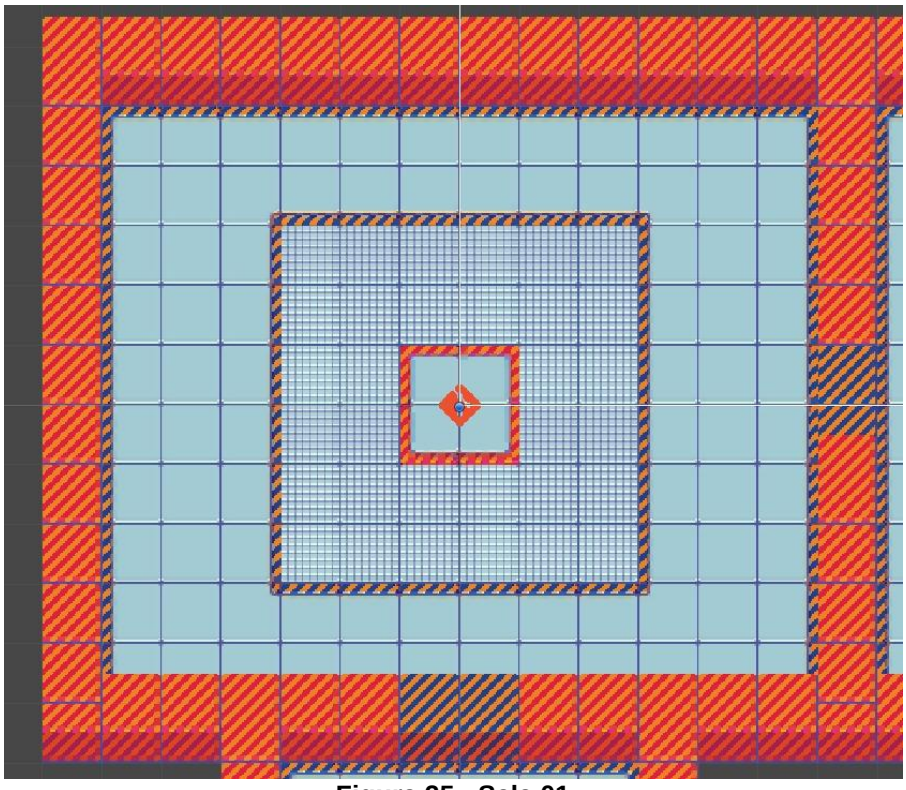

Figura 25 - Sala 01

Esta sala tiene un propósito muy sencillo: Enseñar al jugador cómo usar los puntos de guardado. Para ello, al acercarse al botón de git, salta una conversación que explica cómo funciona, y hasta que no se activa el punto de guardado, la siguiente puerta no se abre. De esta manera, se reduce la capacidad de error del jugador, a base de cerrarle posibles caminos que lleven a puntos de muerte sin posibilidad de recuperación.

6.3 Sala 02

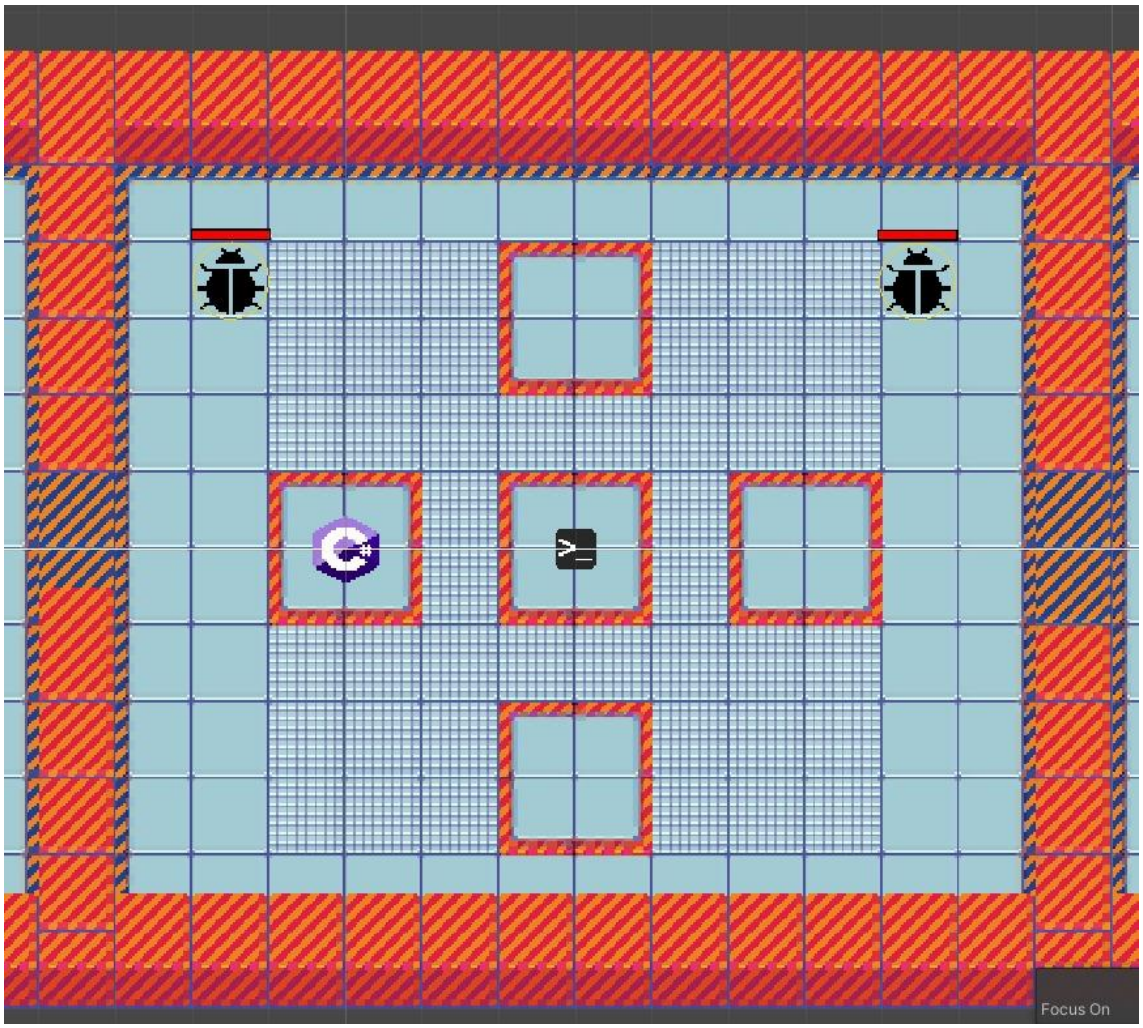

Figura 26 - Sala 02

En esta sala se introduce la mecánica general del juego: El combate y cómo avanzar entre habitaciones. Para empezar, el elemento más cercano al jugador es la habilidad de C#, que permite al jugador disparar. Además, a través del diálogo se le indica al jugador que prueba a disparar y se familiarice con la manera de jugar. Al pulsar más tarde el botón de SSH, aparecen dos enemigos, de los más básicos y débiles, que permiten tener una primera prueba de las mecánicas.

Al acabar con los enemigos, se abre la puerta y se puede avanzar.
En esta sala puede, sin embargo, utilizarse primero el botón de SSH y luego el de C#, si se rodea a propósito este último.

Para evitar este pensamiento, se coloca más cerca del jugador y se llama atención con el diálogo a que lo coja. Sin embargo, si el jugador decide ignorar el consejo, se verá obligado a cogerlo de todas formas, pues no podrá avanzar si no. De esta manera, nos aseguramos de que el jugador entiende que si el juego le indica algo, normalmente sería porque se le quiere enseñar cómo funciona el juego, y no debería ignorarlo.

6.4 Sala 03

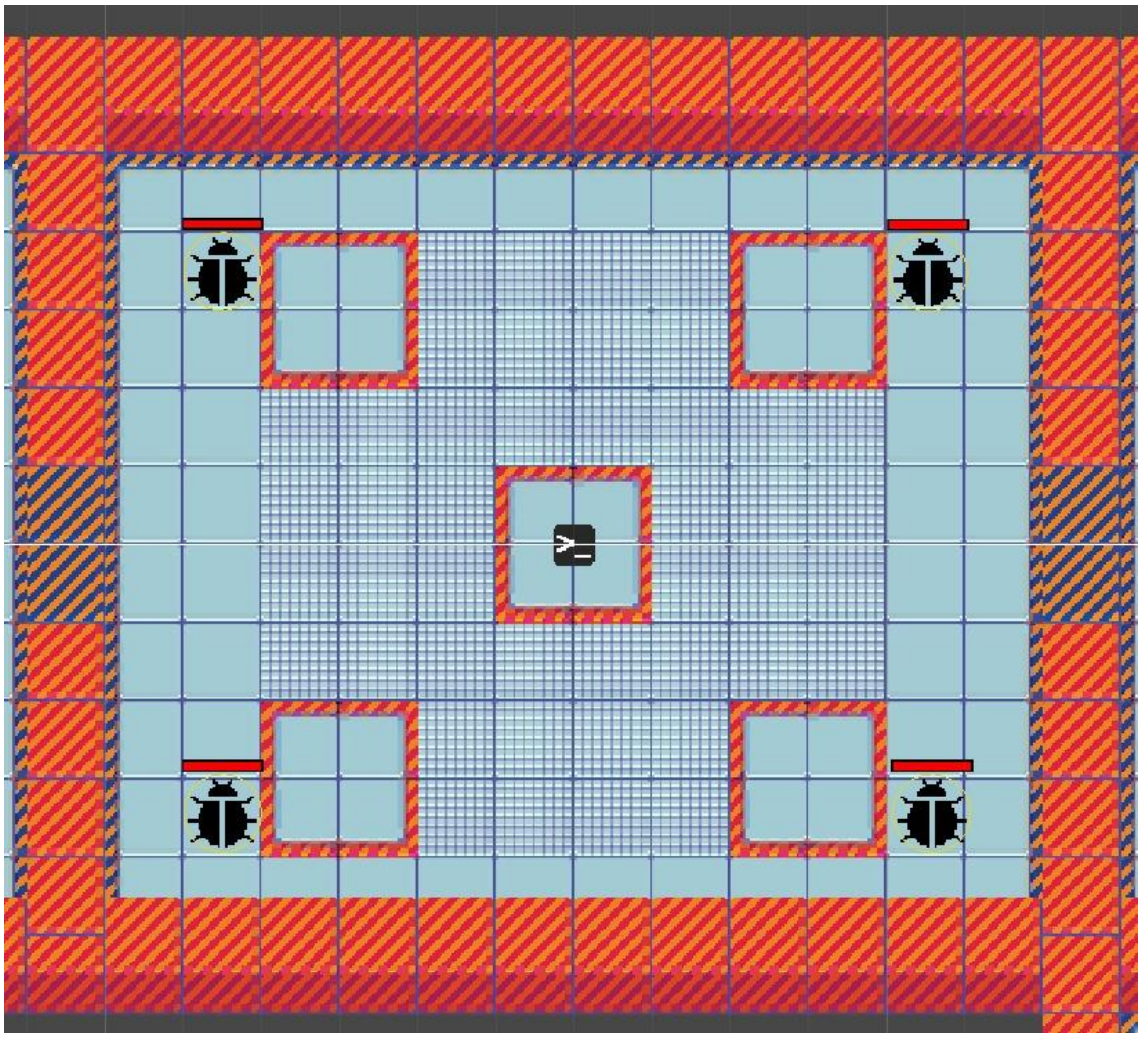

Figura 27 - Sala 03

Esta sala sirve para proporcionar un reto mayor al jugador, a base de mostrarle más enemigos de los que ya conoce. Cimenta la manera de moverse y disparar a la vez, mientras que hace entender al jugador que el reto va a ir aumentando.

También es la primera sala sin diálogo, así que ayuda a mantener un ritmo rápido en comparación con las salas anteriores que estaban llenas de explicaciones.

6.5 Sala 04

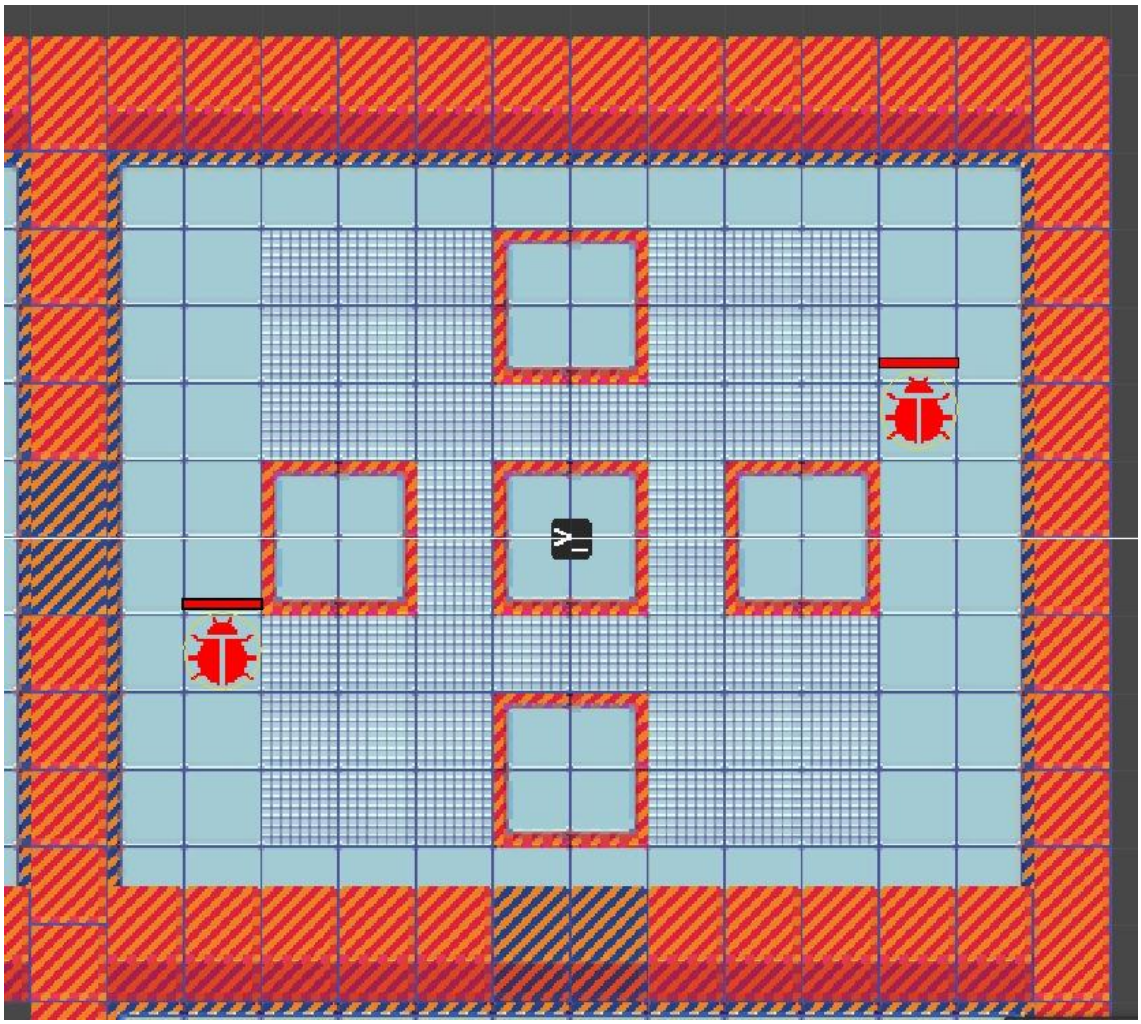

Figura 28 - Sala 04

Una vez el jugador ha dominado el reto anterior, se le presentan por primera vez un tipo diferente de enemigo. En este caso, una variación del original, para que no sea demasiado difícil de entender cómo se comportan. Son más difíciles, y suponen un reto mayor, pero dado que conservan la misma forma se asume que su comportamiento es igual, pero se les añade más peligro al usar el color rojo, típicamente asociado con peligro.

6.6 Sala 05

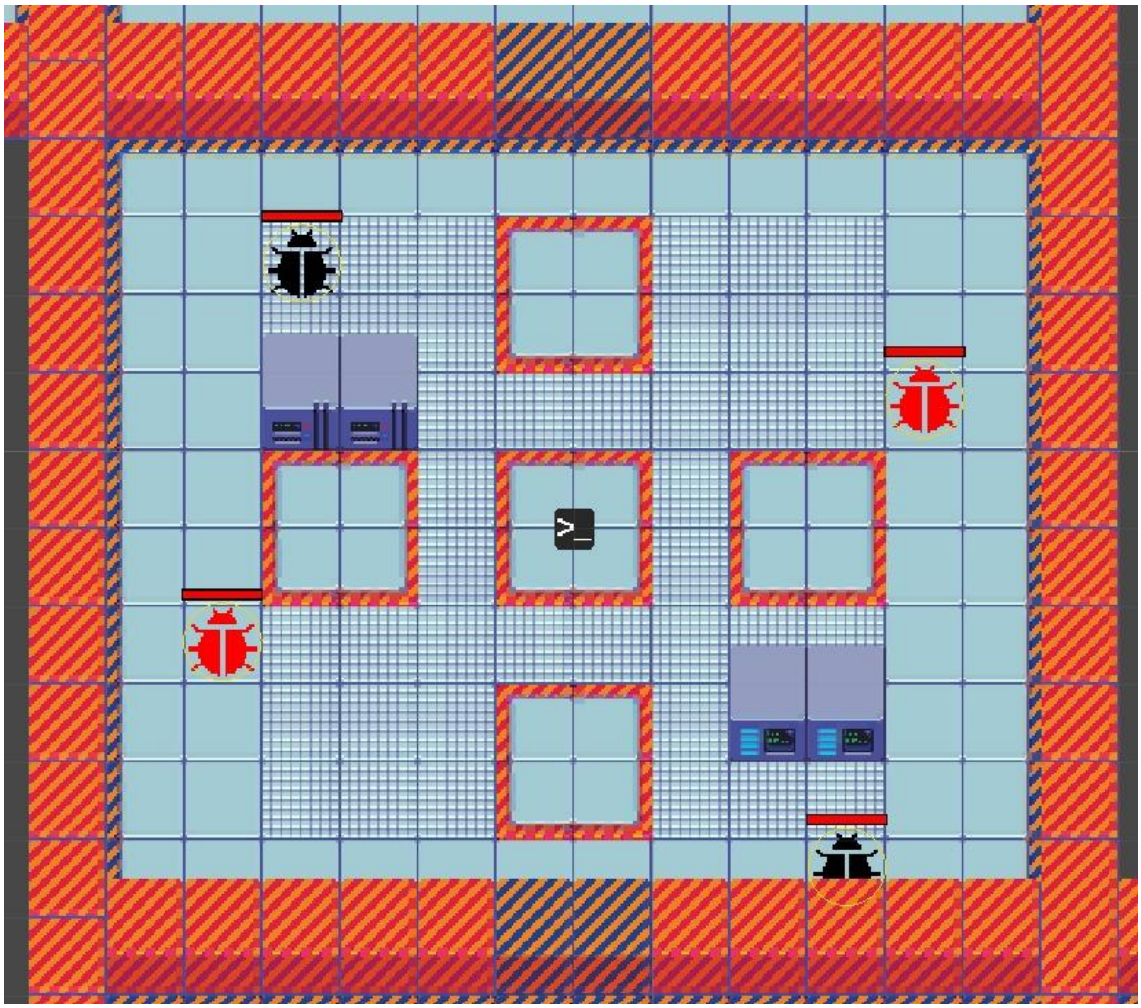

Figura 29 - Sala 05

Es al llegar a este punto que se introducen los obstáculos, y se permite al jugador usarlos para esquivar y confundir a los enemigos. Se añaden dos enemigos sencillos a los dos difíciles que se presentaron en la sala anterior, y se refuerza la idea de que el reto va a continuar aumentando.

6.7 Sala 06

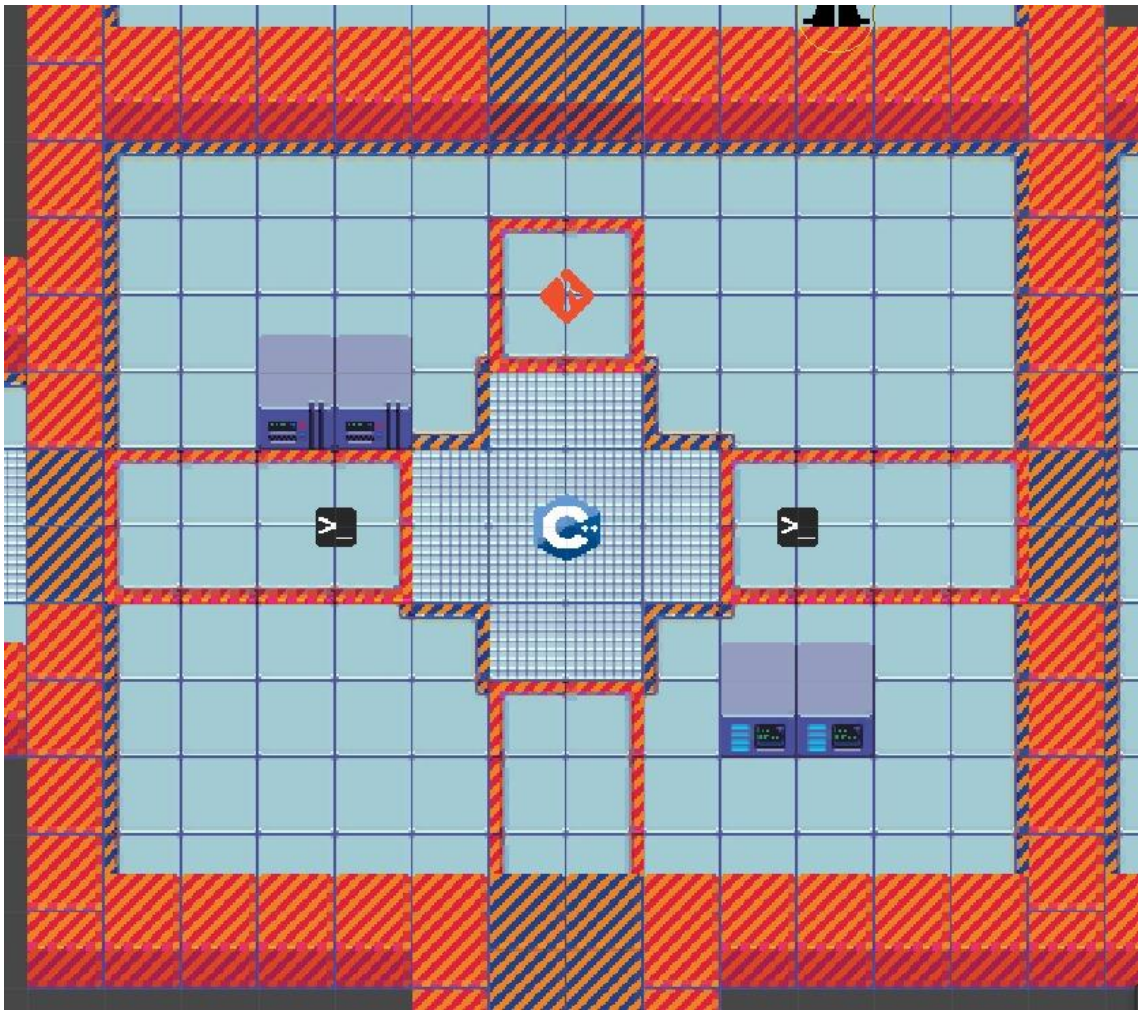

Figura 30 - Sala 06

Esta es la primera vez que se rompe la dinámica de un único camino, siendo una sala con 3 posibles salidas. Se refuerza la importancia de esta sala al estar llena de cosas en comparación con las salas anteriores, y aparecen en el suelo una serie de guías que llevan hacia las tres puertas, marcando la importancia de seguir hacia adelante.

Además, aquí se añade la última mejora del jugador: El *dash*. Es una habilidad muy útil que se maneja de forma diferente a lo que se ha hecho anteriormente, así que se insta al jugador a probarla en esta sala, la cual no es peligrosa. Se utilizan obstáculos en mitad de sala para que el jugador pueda ver de primera mano cómo interactúan los obstáculos y el *dash* y se permite al jugador elegir por qué puerta ir primero. Nosotros continuaremos primero hacia la izquierda.

6.8 Sala L-01

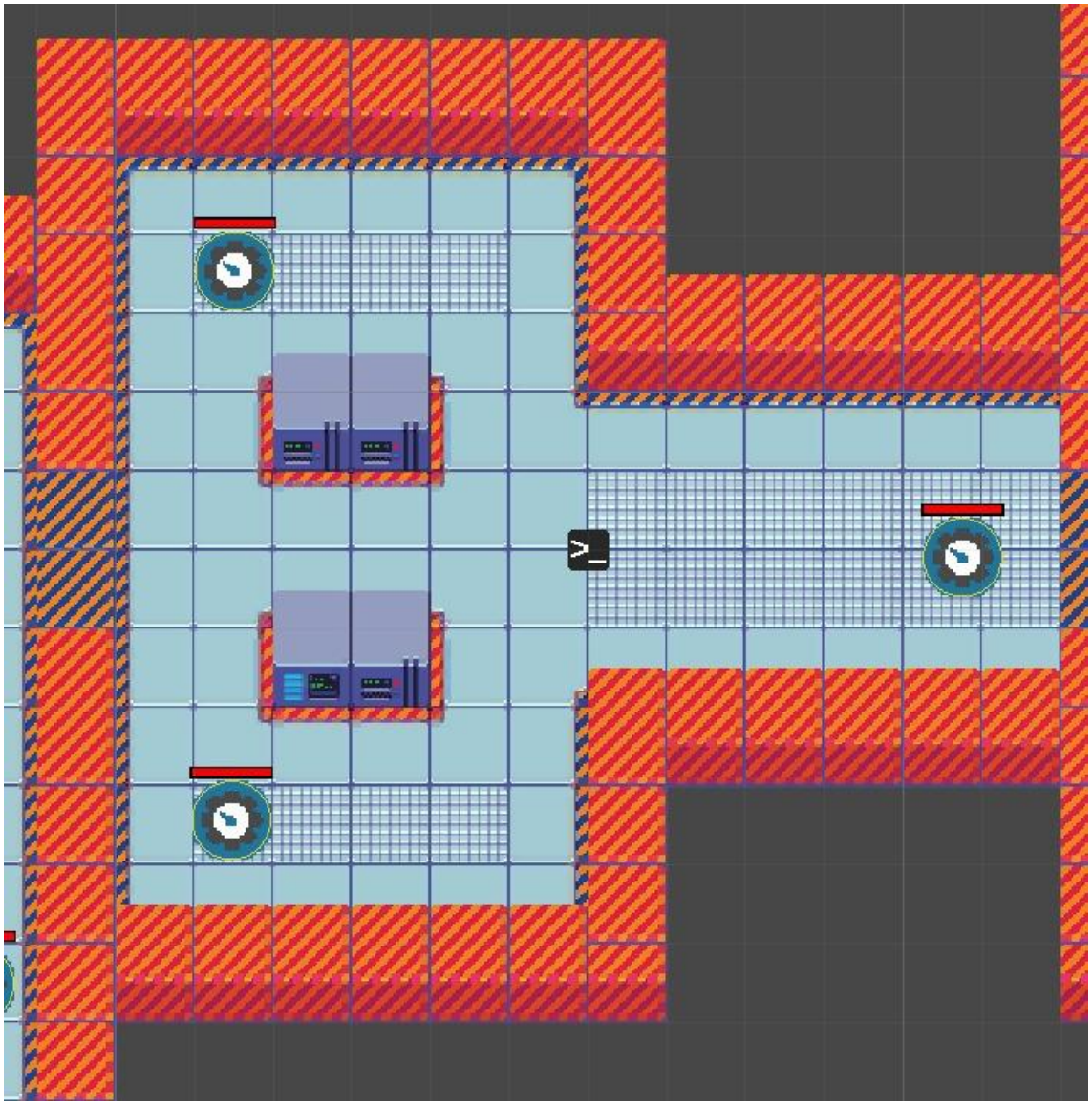

Figura 31 - Sala L-01

Aquí hallamos una particularidad con respecto al resto de las salas: es la primera en la que empezamos a la derecha y tenemos que avanzar a la izquierda. Aunque pueda parecer que no es del todo importante, los jugadores solemos tener inculcado que el avance es de izquierda a derecha, debido a nuestra forma habitual de escritura. Además, es la primera habitación con una forma no cuadrada en la que va a haber combate.

El botón está en el centro, y cuando aparecen los enemigos bomba, el diálogo explica qué es lo que hacen y por qué están ahí. La colocación hace que el jugador casi seguro detone una de las bombas habiendo un muros entre ambos, y verá así que los obstáculos no paran las explosiones.

6.9 Sala L-02

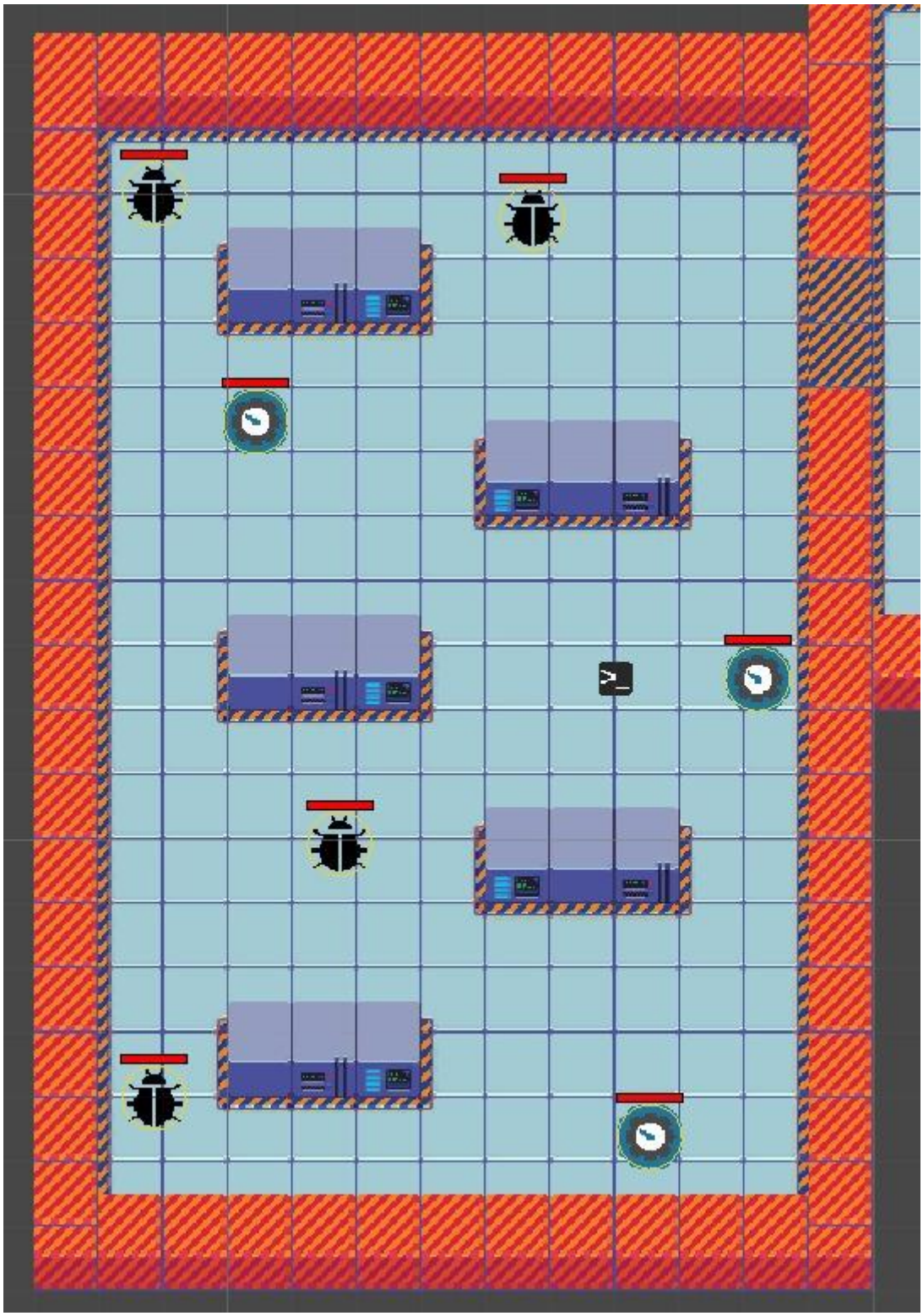

Figura 32 - Sala L-02

Aquí se halla una sala vertical, con cantidad de muros y enemigos. Aunque hay muchos, estos enemigos mueren pronto y se hacen daño entre ellos, gracias a las bombas, así que en realidad no es demasiado complicada pero sirve para afianzar la posibilidad de habitaciones llenas de enemigos. Los muros en medio de la sala ayudan a guiar a los enemigos y, en caso de que en la sala anterior

no lo haya aprendido, es muy difícil que el jugador no vea en esta sala cómo las explosiones atraviesan paredes.

6.10 Sala R-01

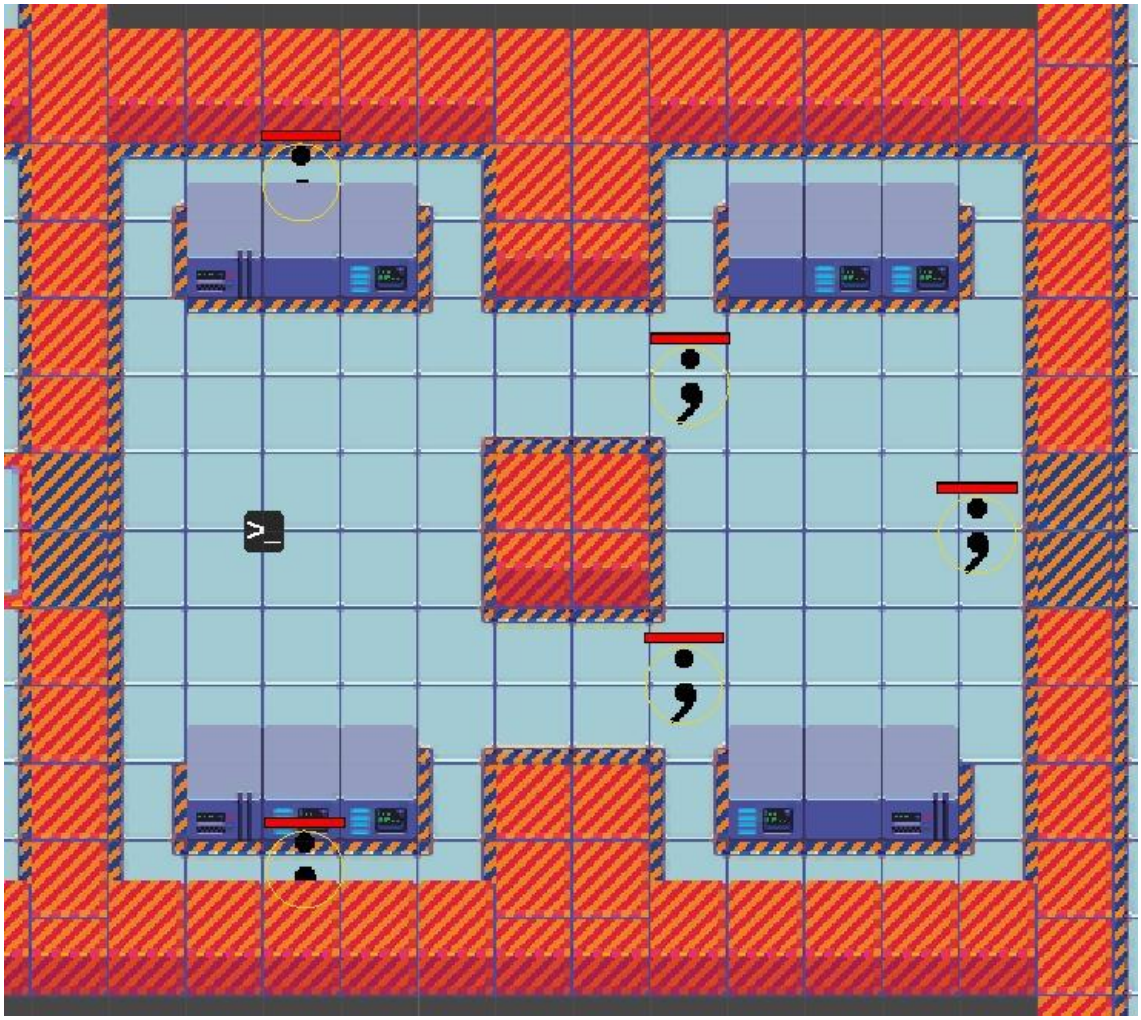

Figura 33 - Sala R-01

En esta habitación se introducen los enemigos punto y coma, que aparecen y desaparecen. Se da la explicación de este comportamiento y se permite al jugador que los elimine. Además, se usa un *layout* extraño que no se ha utilizado hasta el momento, y que permite que algunos enemigos se escondan mucho más de lo que lo harían en una sala vacía.

6.11 Sala R-02

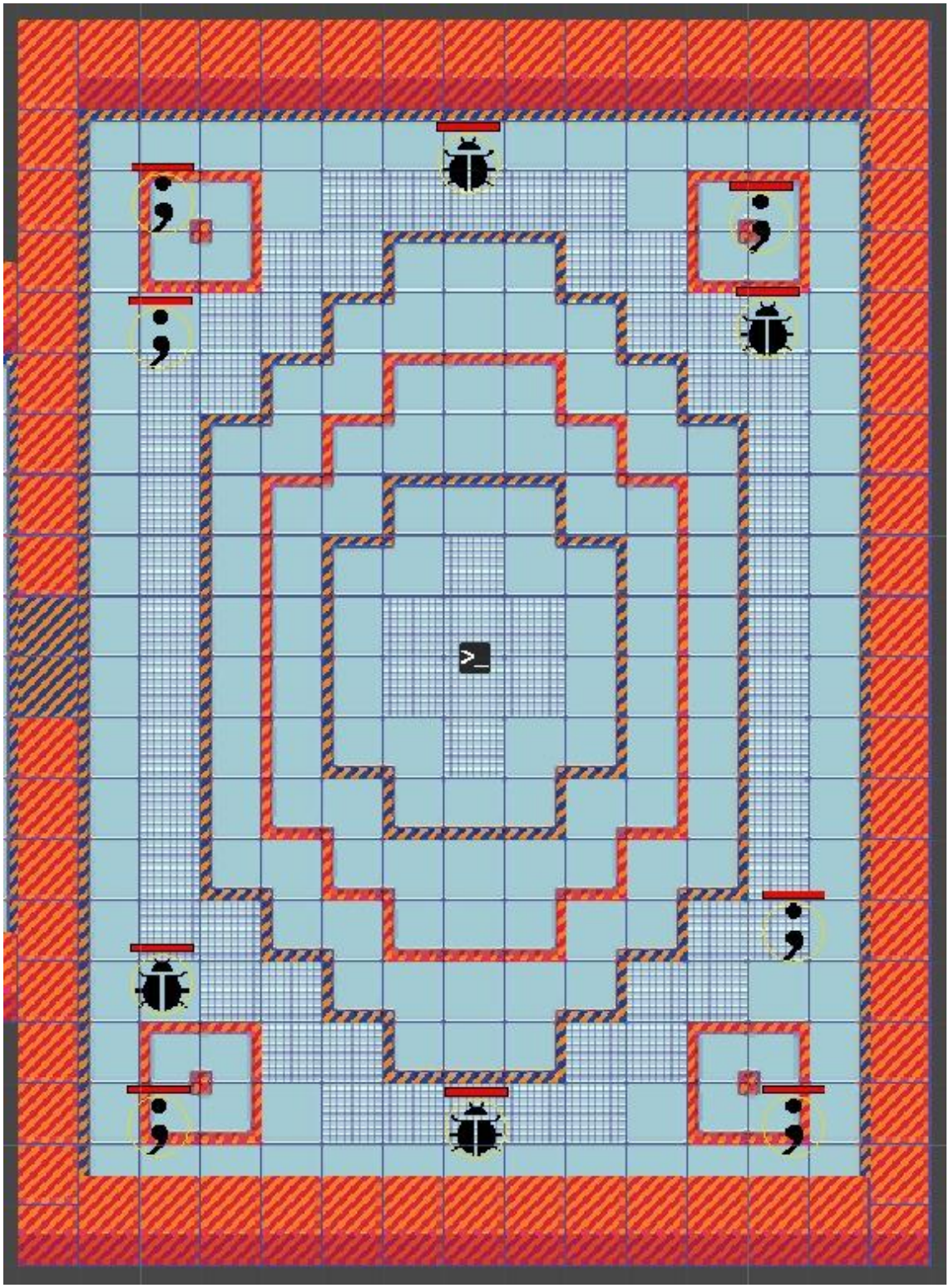

Figura 34 - Sala R-02

En esta sala se ejecuta el mayor reto hasta el momento, haciendo que el jugador aprenda a moverse en salas grandes con muchos enemigos, principalmente dando vueltas y circulando alrededor de los enemigos.

6.12 Sala D-00

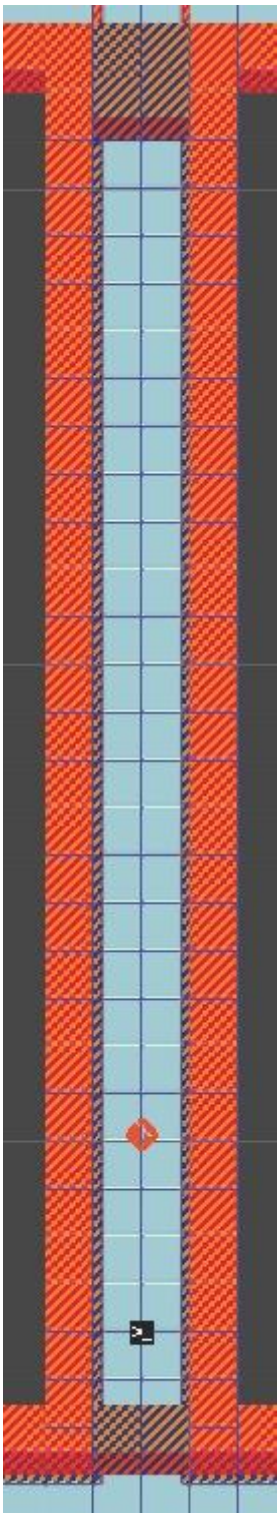

Figura 35 - Sala D-00

Esta sala no es realmente una sala. Hace la función de pasillo y sirve para generar suspense sobre lo que viene adelante. Es largo y lleva más tiempo de atravesar que cualquier sala que hayamos visto hasta el momento, y tiene un punto de guardado justo antes, reforzando la idea de que lo que viene es diferente a lo que se haya experimentado al principio.

6.13 Sala D-01

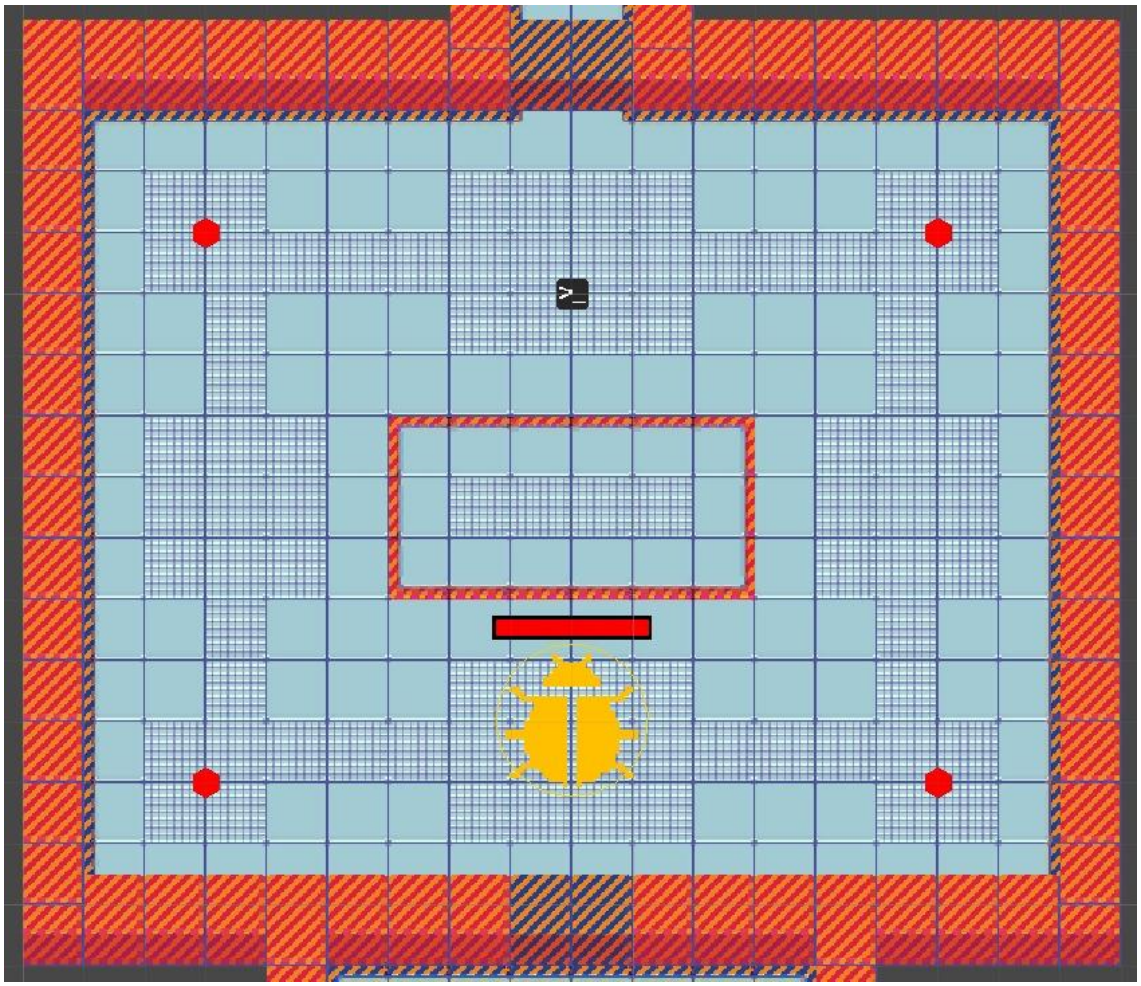

Figura 36 - Sala D-01

Esta es la sala del combate final. Inmediatamente nada más pulsar el botón aparecen cantidad de enemigos, todos ellos desde los puntos rojos que hay en las esquinas, y seguirán apareciendo cada poco. Además, el jugador enseguida ve cómo un enemigo gigante, más grande que ninguno hasta el momento, aparece justo frente a él.

Este es un combate en el que hay que derrotar al enemigo principal para poder avanzar, y para ello hay que saber aprovechar el *dash*, los enemigos que explotan, y saber apuntar al enemigo principal.

6.14 Sala D-02

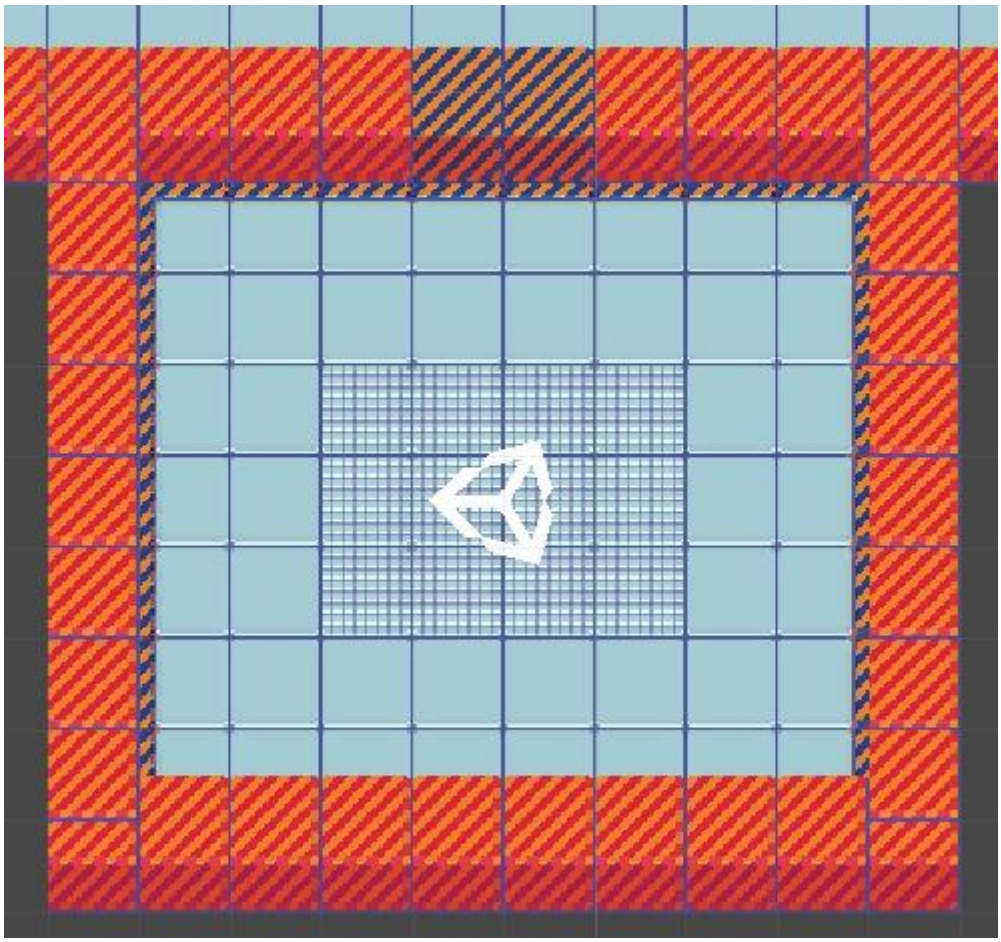

Figura 37 - Sala D-02

Este es el momento en que Javi consigue su objetivo. Un logo de Unity girando frente a él, una despedida emotiva por parte de Vitae y de él, y el juego se acaba, haciendo una última referencia a la cultura pop de videojuegos.

7. Conclusiones

Como trabajo de final de Máster, he de decir que me siento orgulloso de él. Evidentemente hay problemas y errores, se habría podido ampliar las mecánicas y el pulido, prestar mayor atención a la jugabilidad… Pero todo trabajo merece ser considerado por lo que es, y este es, personalmente, una buena carta de despedida al máster.

De entre la cosas que se han desarrollado menos de lo que me gustaría puedo contar varias:

- El movimiento de los enemigos por el mapa no es perfecto. Se mueven directamente al jugador sin ningún tipo de patrón extra, y muchas veces se quedan enganchados en los obstáculos y no encuentran las rutas adecuadas. Para poder solucionar esto, se podría haber utilizado un sistema propio de *pathfinding*, o haber investigado en mayor profundidad qué es lo que causa este comportamiento. También se podría haber intentado crear un juego 2.5D, de tal manera que se usasen las herramientas de Unity NavMesh por poder colocarse en 3D, aunque eso habría derivado en una gran cantidad de errores y posiblemente hubiese ampliado el uso de hardware hasta un punto en que no sería viable jugarlo en web.
- Los golpes no tienen suficiente feedback, ni visual ni sonoro. La mayor indicación de que algo ha sido golpeado es un cambio en la barra de vida, que es fácil de identificar en los enemigos pero no en el propio jugador.
- Ausencia total de sonido. Aunque no sea parte de mi abanico de habilidades, utilizar sonidos o música de internet sin copyright habría aumentado la inmersión en el juego.
- La falta de personalización del avatar. Para tratarse de un juego con aspectos de RPG, se ha perdido la oportunidad de dotar al personaje principal de una figura más personalizada. Aunque esto se puede arreglar en cierta medida con los retratos usados en los diálogos, no deja de sentirse como que falta algo.
- La cantidad de ideas en el tintero. Muchos, muchos conceptos se han quedado sin incluir, algunos para bien y otros para mal. Una manera de poder lanzar bombas, o de añadir cargadores en lugar de disparos infinitos habría llevado este juego a un punto más allá en cuanto al *gunplay*.

En cambio, se puede decir que el cumplimiento de los hitos ha funcionado bien. Las exigencias eran realistas por mi parte, y era así debido al tiempo que se le podría dedicar. Esto ha sido de gran ayuda y ha conseguido evitar demasiado *crunch* en las horas finales de cada entrega, aunque eso no quita que se le haya dedicado tiempo extra en estos momentos.

Si este juego se pretendiese hacer de nuevo, posiblemente haría una estructura mucho más clara de los aspectos a incluir y a obviar, creando un GDD mucho más detallado en el que poder basarse para generar una

estructura de código adecuada que permita la inclusión de estos elementos mucho más fácilmente.

Puestos a hablar de estructura de código, aunque me he sentido cómodo usando técnicas como la herencia y los ScriptableObjects, las cuales desde luego proporcionan una muy necesitada mejora de estructura de código, noto que algunas referencias han quedado a medio hacer.

Hay algunos *Controllers* que se usan siguiendo un patrón de *Singleton*, pero cuyo uso no está explorado del todo. De la misma manera, otras técnicas como la utilización de eventos de juego mediante ScriptableObjects, como se explica en la charla de la fuente [16], hacia el minuto 27:50. También se puede encontrar una guía de cómo implementarlos en la fuente [17]. Esto habría ayudado a hacer un entorno mucho más centrado en el editor y menos en el código, facilitando el avance rápido en la creación de nuevas habitaciones y niveles.

8. Glosario

- **Unity:** Motor de videojuegos que permite la creación de los mismos.
- **Input:** Entrada de información al programa. Esto puede suceder mediante mandos, ratones, teclados, o controladores diversos.
- **Dash:** Acelerón rápido del jugador en el que se ve propulsado en una dirección, normalmente sin poder cambiar la dirección en la que se mueve.
- **Bug:** Error de un programa que evita que se ejecute correctamente.
- **Script:** Series de instrucciones que el ordenador procesa y sigue para conseguir un resultado deseado. Los *scripts* pueden generar todo tipo de comportamientos dentro de un software, como movimientos de enemigos o procesado de información de formularios.
- **Controller:** *Script* al que se suelen atribuir tareas de control de aspectos más grandes de un programa, como la manera en que fluyen los turnos, cómo se muestran los diálogos, o cómo se cambia de niveles.
- **ScriptableObject:** Script cuya función principal es guardar información y hacer procesos pequeños con ella. No se usa directamente en la escena y puede haber varias instancias del mismo con diferentes datos.
- **Tilemap:** Objeto que muestra información gráfica en pantalla, normalmente dividida en cuadrícula, como en los juegos antiguos en los que cada cuadrado debía tener solo un tipo de imagen.
- **Spawn:** Significa aparecer. Cuando algo *spawnea* significa que aparece en la escena, y cuando algo *despawnea* es que desaparece. Si hace *respawn* es porque ha vuelto a aparecer, puede que en el mismo sitio en el que apareció la última vez o en otro diferente.
- **Checkpoint:** Punto de guardado. Se suele utilizar este término para definir los puntos en los que el progreso se guarda, y a los que se vuelve si se falla en el futuro.
- **Sprites:** Recursos gráficos, imágenes que se pueden utilizar en la creación de un juego.
- **Assets:** Recursos en general de un juego. Pueden ser tanto visuales, como sonoros, como de código y cualquier otra cosa que pueda servir dentro de la creación directa de un juego.
- **Placeholder:** Elementos provisionales que pueden utilizarse para hacerse una idea de cómo se verá el juego final. Una táctica muy utilizada de este tipo en otros sectores es el *Lorem Ipsum*.

9. Bibliografía

- [1] Juegos célebres de doble stick (28/5/2021): https://www.gamasutra.com/view/news/302732/7_twinstick_shooters_that_game_developers_should_study.php

- [2] Imagen de la sala de pinchos del *Binding of Isaac* (28/5/2021): https://moddingofisaac.com/mod/997/the-binding-of-isaac-cleanbirth

- [3] Imagen de los cuadros de diálogo del *Final Fantasy VII* (29/5/2021): https://thelifestream.net/ffvii-the-original/final-fantasy-vii-the-unused-text-series/part-3

- [4] Maniquí de 8 direcciones con animaciones (15/3/2021): https://opengameart.org/content/8-directional-character-template

- [5] Navi, de la Saga The Legend of Zelda (1/4/2021): https://zelda.fandom.com/es/wiki/Navi

- [6] Tileset Sci-Fi: https://opengameart.org/content/sci-fi-interior-tiles

- [7] Requisitos de Unity Editor (18/5/2021): https://docs.unity.cn/Manual/system-requirements.html

- [8] PixelMe, pixelador de retratos online (20/4/2021): https://pixel-me.tokyo/en/

- [9] Torretas de concepto (13/3/2021): https://opengameart.org/content/vertical-shmup-set-2-m484-games

- [10] Logo de Vitae (2/4/2021): https://www.coolfreecv.com/resume-icon-free

- [11] Cuadros de diálogo (15/5/2-21): https://opengameart.org/content/4-svg-message-boxes

- [12] Barra de vida del jugador dividida en corazones (1/6/2021): https://opengameart.org/content/hearthealth

- [13] Retro Gaming Font (20/5/2021): https://www.dafont.com/retro-gaming.font

- [14] *A* Pathfinding Project* (30/3/2021): https://arongranberg.com/astar/

- [15] *Script* de revelación de texto (17/5/2021): https://www.gamasutra.com/blogs/MiguelSantirso/20180129/313803/Better_text_reveals_Unity_code_inside.php

- [16] Charla *“Unite Austin 2017 - Game Architecture with Scriptable Objects”* (5/6/2021): https://youtu.be/raQ3iHhE_Kk

- [17] Tutorial *“Three ways to architect your game with ScriptableObjects”* (5/6/2021): https://unity.com/how-to/architect-game-code-scriptable-objects