UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

D.A.N.G.E.R -

Disaster Prevention

Videojuego educativo para la gestión de emergencias en Unity 3D : IA, Agentes Inteligentes y Generación Procedural

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Pablo Querol Ballester

Tutor: Ramón Pascual Mollá Vayá

Segundo tutor: Teresa María Pellicer Armiñana

Curso 2020-2021

Resumen

En este TFG se propone realizar un videojuego de simulación social y estrategia en Unity 3D para ordenador con una posterior portabilidad a dispositivos móviles.

Este videojuego tendrá aspectos compartidos con Adrián Sanchez Lavarias donde cada uno desarrollaremos distintos aspectos del juego. Habrá aspectos que serán compartidos como, por ejemplo, el análisis de mercado, investigación en materias de seguridad y el documento de diseño del videojuego (GDD).

En este juego encontraremos personajes con distintas personalidades a los que deberemos darles órdenes con el fin de salvar al mayor número posible de la emergencia en la que se ven envueltos.

Las acciones del jugador tendrán repercusiones en el entorno y en los NPCs, que actúan como agentes a su entorno, que también se encuentra en constante cambio. Para ello, estos agentes inteligentes reactivos, serán capaces de percibir amenazas, obstáculos y otros NPCs y desplazarse por el mismo de forma autónoma interactuando con el mismo de forma acorde a su estado de estrés y condiciones físicas. Cada personaje presentará un nivel distinto de atributos físicos iniciales para proporcionar al juego de un mayor realismo.

Los niveles podrán ser generados tanto proceduralmente como a voluntad del jugador lo que fomentará la rejugabilidad del título y que se creen situaciones únicas.

Mi compañero se encargará de las interfaces de usuario, como los menús, una base de datos que recoja las estadísticas de los jugadores y el diseño de una página web, así como de un modo de juego para crear de forma manual las diferentes estancias.

El objetivo de este juego es usarlo como herramienta para utilizar estas tecnologías que se encuentran a la orden del día al mismo tiempo que se aprende acerca de las medidas y comportamientos que deben tomarse en caso de emergencia en un entorno interactivo, seguro y divertido.

Palabras clave: Simulación social, Unity 3D, IA, agentes inteligentes, generación procedural , videojuegos.

Summary

In this TFG we propose to develop a social simulation and strategy video game in Unity 3D for computers with a later portability to mobile devices.

This video game will have shared aspects with Adrián Sanchez Lavarias where each of us will develop different aspects of the game. There will be aspects that will be shared such as, for example, market analysis, security research and the game design document (GDD).

In this game we will find characters with different personalities to whom we will have to give orders in order to save as many as possible from the emergency in which they are involved.

The player's actions will have repercussions on the environment and on the NPCs, who act as agents to their environment, which is also constantly changing. To this end, these reactive intelligent agents will be able to perceive threats, obstacles and other NPCs and move through it autonomously interacting with it according to their state of stress and physical conditions. Each character will have a different level of initial physical attributes to provide the game with greater realism.

Levels can be generated either procedurally or at the player's will, which will encourage the replayability of the title and the creation of unique situations.

My partner will be in charge of the user interfaces, such as menus, a database to collect player statistics and the design of a web page, as well as a game mode to manually create the different rooms.

The objective of this game is to use it as a tool to use these technologies that are the order of the day while learning about the measures and behaviors to be taken in case of emergency in an interactive, safe and fun environment.

Keywords: Social simulation, Unity 3D, AI, intelligent agents, procedural generation, video games.

Tabla de contenidos

Índice de ilustraciones

1. Introducción

1.1 Motivación

Mi motivación para este Trabajo Fin de Grado proviene de que desde pequeño siempre me he sentido atraído por la tecnología y, particularmente, los videojuegos. Desde entonces me preguntaba cómo era posible que los personajes de un juego estuvieran dotados de inteligencia, que se anticipan a tus movimientos y compiten contigo o te ayuden. Más tarde conocí y me interesé por la robótica y automatización y es por todo ello que escogí la rama de Computación, para entender cómo funcionan, no solo en los videojuegos, sino cualquier comportamiento autónomo que parece sacado de una película de ciencia ficción. Siempre he pensado que este tipo de tecnologías son el futuro, y a día de hoy estamos en el inicio de este mundo de ciencia ficción que imaginaba con coches autónomos, asistentes virtuales y videojuegos donde puedes sumergirte dentro a través de la realidad virtual.

Este proyecto me ha brindado la oportunidad de aprender y poder programar mis propios personajes y entorno que actúan de forma autónoma y entender la magia (o la ciencia) que hay detrás de la cortina y que llevaba queriendo destapar desde hace años. En este proyecto se profundizará acerca de la creación de los diferentes aspectos del videojuego , pero sobre todo se hará hincapié en la parte relacionada con la generación procedural y el comportamiento inteligente o autónomo del entorno y los agentes que son los aspectos relacionados con mi rama del conocimiento dentro de la informática.

El proyecto está desarrollado junto a Adrián Sánchez Lavarias y, por lo tanto, tenemos en común los apartados de “Estado del arte” y el documento de diseño del juego situado en el anexo de la memoria.

1.2 Objetivo

El objetivo de este proyecto es el diseño y desarrollo de un videojuego en el que varios agentes inteligentes se tienen que poder desenvolver y reaccionar al entorno de forma tanto autónoma como guiada por el usuario.

El escenario que hemos escogido se desarrolla durante el transcurso de un incendio, de esta forma nos permite tener un entorno cambiante donde la presencia y el movimiento de elementos de riesgo como el fuego y el humo condicionan el comportamiento de los agentes tratando de generar una respuesta lo más humana y realista posible, con cambios a nivel tanto psicológico como físico. Estos escenarios serán generados de forma procedural, por lo que cada vez que se ejecute el juego los mapas serán completamente diferentes, lo que fomenta la rejugabilidad del título y aportará al jugador un mayor grado de dificultad y evaluará y ayudará a mejorar las capacidades de memoria y gestión del tiempo y los recursos del jugador.

La finalidad de este proyecto es la de que los jugadores se puedan exponer a diversas situaciones de riesgo de una forma segura y divertida a través de este juego con el que podrán desarrollar tanto comportamientos a tener en este tipo de situaciones y aumentar la concienciación y la precaución con este tipo de riesgos externos, como también a poder desarrollar conocimientos y habilidades de forma permanente para prevenirlas y gestionarlas de la forma más segura posible . También se le dotará al jugador de una mayor capacidad de liderazgo y capacidad resolutiva en situaciones de estrés donde los factores como el tiempo son de vital importancia , en donde pueden no tener toda la información del entorno que nos rodea y deban de ser en cierto modo adaptables o reactivos.

1.3 Estructura de la obra

La memoria del proyecto presenta una estructura dividida en 4 bloques principales:

- Introducción: Se explica la legitimidad de la realización de este proyecto a través de los objetivos enunciados en el mismo, así como la motivación para hacerlo.

- Análisis: Se realiza un breve análisis de mercado para comparar soluciones del mismo ámbito relacionadas con la gamificación que estén enfocadas para la prevención y gestión de emergencias.

- Diseño e implementación: Se explican las diferentes decisiones que han sido estudiadas y tomadas en la realización del proyecto, así como se explica la forma en la que han sido abordadas. Se desarrollará el uso y funcionamiento de los diferentes *scripts* y la relación entre los mismos.

- Conclusión: Evaluación del producto final, se estudia la relación del trabajo realizado con los estudios cursados y se plantean puntos para un futuro trabajo en el proyecto.

2. Estado del arte

2.1 Crítica al estado del arte

Los videojuegos se crearon con la finalidad de entretener, pero con el tiempo se ha expandido a mucho más. Hoy en día podemos encontrar videojuegos que sirven para transmitir ideas, como herramienta para expresar un sentimiento o con la intención de enseñar un concepto.

Si añadimos que la tecnología para jugarlos ha evolucionado con el paso de los años hasta convertirse en un producto accesible por la mayoría de gente, como son los teléfonos móviles.

Por esto y al ser nuestro objetivo que se pueda utilizar para enseñar en clases de alumnos u oficinas, utilizaremos ordenadores o smartphones para jugar. Aprovechando la tecnología para llegar a todo el mundo y un medio divertido para transmitir la información.

Al ser estudiantes y apasionados de los videojuegos nos interesan especialmente los juegos educativos como una nueva forma de enseñar [1]. Hay que destacar que este género de videojuegos no es el más popular, pero cuanto más avanza la tecnología más formas aparecen de crear mecánicas interactivas, innovadoras e interesantes con las que aprender mientras juegas.

A la vanguardia de las tecnologías actuales encontramos la realidad virtual o VR, un entorno en el que poder interactuar como si estuvieras físicamente dentro del videojuego. Lo que abre la posibilidad de simular lecciones o clases reales en el futuro, cuando está tecnología se popularice.

Junto con mi compañero Adrián analizamos el mercado de juegos y aplicaciones cuya temática se centra en la simulación de emergencias, se incluyen en este análisis no sólo la temática de las emergencias enfocadas a incendios, sino también emergencias médicas, o de carácter laboral, en las tres plataformas más accesibles actualmente: páginas web, dispositivos móviles y ordenadores.

- Al analizar el mercado de videojuegos de simulación de emergencias en la web encontramos numerosas aplicaciones creadas en flash, estos videojuegos se componen principalmente por juegos de memorización, juegos en los que relacionar palabras o incluso preguntas tipo test con los conceptos a enseñar.Cabe destacar

también que la tecnología de Flash Player ya no está soportada de forma nativa por los navegadores web y esta se encuentra obsoleta. Es posible descargar extensiones y programas, así como páginas web especializadas para poder reproducir este contenido, pero para el usuario promedio estos contenidos serán obsoletos e inaccesibles.

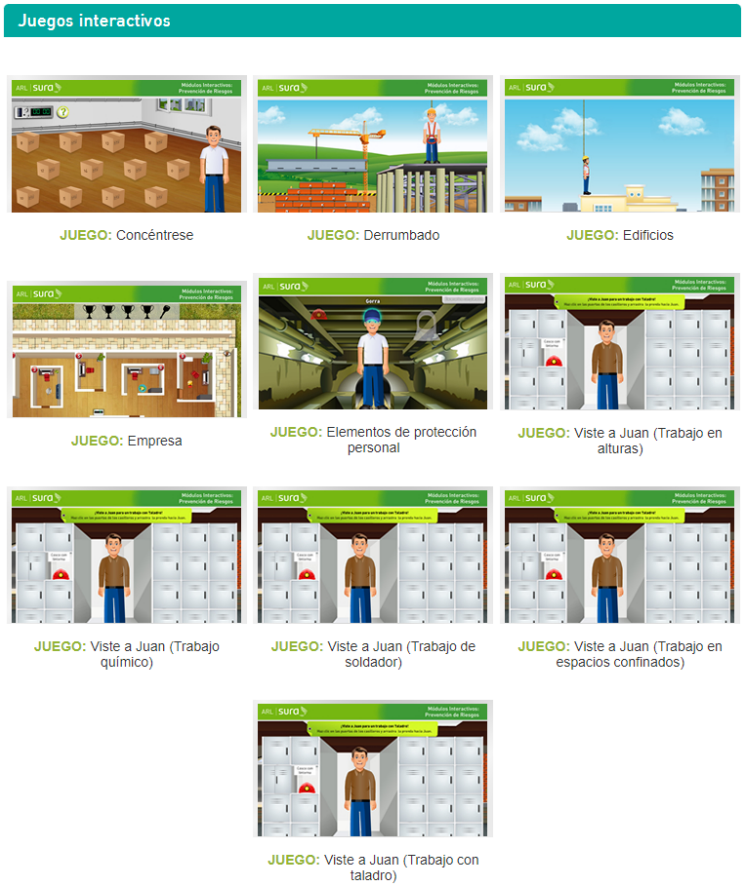


Figura 1: ARL Sura (1)

- En el mercado móvil principalmente se encuentran aplicaciones que sirven como manual con la información en formato texto. Cabe destacar que también encontramos videojuegos que no siguen esta norma, como por ejemplo basados en secuencias animadas en las que debemos interactuar para completar la escena. Dicho esto, la interacción no se desarrolla más allá de pulsar un botón repetidas veces o desplazar elementos en pantalla.

(1) ARL SURA: https://www.arlsura.com/index.php/aprenda-jugando-independientes

Figura 2: EMERGENCY HQ (2)

- Por último, en videojuegos de ordenador se pueden encontrar algunos de proyectos en 3D creados este año que enseñan, mediante simulación de casos reales, que hacer en caso de incendio en una oficina u otras emergencias. También encontramos juegos de estrategia como la saga de juegos EMERGENCY que nos muestran una ciudad con distintas emergencias, incendios o gente herida y debemos de gestionar los personajes para ir dirigiendolos hacia las mismas y solucionarlas:

Figura 3: EMERGENCY 20 (3)

(2) Emergency HQ:https://play.google.com/store/apps/details?id=com.sgs.emhq.android&hl=es&gl=US

(3) Emergency 20: https://store.steampowered.com/app/735280/EMERGENCY_20/

Encontramos una cuarta referencia en el mercado que son los juegos de mesa. Generalmente los juegos de mesa orientados a la prevención y actuación en casos de emergencia están destinados a memorizar una serie de comportamientos de riesgo o preventivos ya a trabajar la memoria pero no despiertan interés en los jugadores y todos los conceptos son representados como una situación abstracta que no aporta ningún grado de inmersión en la situación real.

- **PREVENCARD**

  Este juego de cartas tiene por objetivo inducir en la mente de los trabajadores la necesidad de identificar el riesgo, para poder prevenirlo y así aprender a protegerse.
  (Para jugar o ver más información haz Click aquí)

- **PREVENTE**

  En este juego de mesa, cada jugador deberá identificar correctamente los riesgos a que se expone en el trabajo y adoptar las medidas preventivas necesarias.
  (Para jugar o ver más información haz Click aquí)

- **9k1**

  En este juego deberás definir tu estrategia, cumplir tus metas, verificar si estás actuando correctamente y resolver las incidencias que te vayas encontrando, al final, conseguir cumplir tus objetivos de manera satisfactoria.
  (Para jugar o ver más información haz Click aquí)

- **ELIGE**

  Este juego trata de aventurarse hacia la toma de decisiones correctas. Ya que cada día tomamos multitud de decisiones, unas acertadas, otras no tanto. Si tuvieras la oportunidad de volver atrás cada vez que te equivocaras, si pudieras elegir, ¿qué harías?
  (Para jugar o ver más información haz Click aquí)

Figura 4: Información prevención (4)

(4) Información prevención:
https://informacionprevencion.com/formacion/juegos-de-prevencion-de-riesgos-laborales/

2.2 Análisis del problema

Tras el estudio realizado en el estado del arte podemos obtener las siguiente conclusiones:

Entre los juegos web encontrados había una gran cantidad de juegos educativos, pero con mecánicas muy repetitivas y simples con muy poca interacción y basadas en memorizar la información, además de estar obsoletos e indisponibles para la mayoría del público.

Por otro lado, en los dispositivos móviles las mecánicas son algo más interactivas para involucrar al jugador en un procedimiento real en el que aprender jugando. Sin embargo, la cantidad de juegos educativos actualmente en el mercado móvil es bastante limitada, sobre todo para un juego tan amplio y de mecánicas variadas como el que se propone, por lo que podría considerarse que es un mercado con potencial.

Por último, los juegos comercializados en ordenador son muy reducidos, pero exploran mecánicas interactivas e inmersivas. La tecnología de realidad virtual puede ser una herramienta muy visual y que proporcione una mayor inmersión, pero actualmente sería inviable utilizar esta tecnología en cualquier sala para enseñar a varias decenas de personas, por lo que se ha decidido desarrollar un videojuego en formato estándar que pueda ser portable y no tenga unos requisitos exigentes para llegar al mayor número de personas posible.

Por estas razones hemos decidido centrarnos en las plataformas de ordenador y móvil. Cabe añadir que el campo de los videojuegos educativos todavía debe actualizarse y desarrollarse ampliamente para actualizarse a las nuevas tecnologías, proporcionando una formación mucho más completa e interactiva que llame la atención de las personas que participan en ella y les proporcione de unos conocimientos y habilidades duraderas en el tiempo.

3. Diseño e implementación

En este bloque de la memoria del proyecto se desarrollarán de forma extensa tanto las decisiones que han sido tomadas a nivel de diseño como la posterior implementación y como han sido abordadas las mismas.Se desarrollará el uso y funcionamiento de los diferentes *scripts* y la relación entre los mismos.

3.1 Planteamiento

El planteamiento de este videojuego es que haya un número variable de agentes inteligentes que puedan percibir, interactuar y adaptarse al entorno en tiempo real, simulando un comportamiento lo más humano y realista posible en una situación de riesgo determinada.

El escenario donde se desarrolla toda la actividad del videojuego será generado de forma procedural en tiempo de ejecución. Este escenario estará dividido en salas, para emular un entorno real que pudiera ser el interior de un edificio. Estas salas además dispondrán de un mobiliario que tendrá que ser también colocado de manera procedural dando la posibilidad de poder alterar la probabilidad de que apareciese un determinado tipo de elemento, como pudiera ser un extintor de incendios u objetos decorativos menos comunes como un estante o un perchero. De este modo las estancias también estarán dotadas de un mayor realismo y se pueden adaptar a distintas las necesidades de los jugadores.

3.2 Relación entre las clases

En la siguiente imagen puede apreciarse de forma conceptual la relación entre los distintos scripts que componen el proyecto.

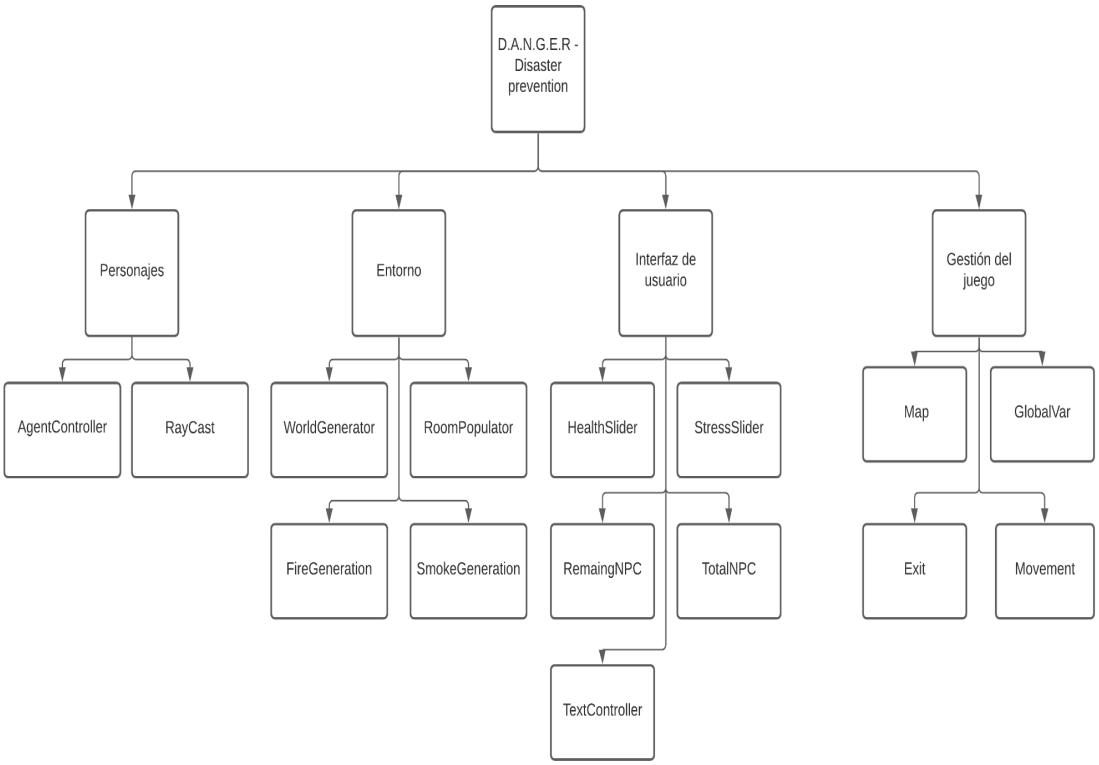


Ilustración 1: Bloques de scripts

Podemos agrupar los scripts en 4 bloques:

- Personajes: Donde se encuentra todo lo relativo al movimiento e interacción con el entorno de los NPCs.

- Entorno: Donde encontramos todos que hace referencia a la generación procedural de los niveles, así como la colocación del mobiliario y la aparición y propagación de los elementos peligrosos del fuego y el humo.

- Interfaz de usuario: En este apartado encontraremos los scripts que controlan los diferentes indicadores que tiene en pantalla el usuario. Estos serán la barra de salud, la barra de estrés, el texto que indica el estado psicológico del personaje y los contadores del número total de personajes y el número pendiente de los mismos que aún no han llegado hasta la salida.

- Gestión del juego: Aquí encontraremos varios scripts que se encargan de aspectos del juego que no entran dentro de los anteriores 3 bloques. Tendremos el modo de mapa para orientar a los jugadores, un script para almacenar las variables compartidas por los distintos scripts y para gestionar las distintas instancias de un mismo script, por ejemplo, de los agentes. Tendremos también el script de salida para controlar cuando un NPC llega a la salida del nivel y un script sencillo de movimiento para que el jugador pueda mover la cámara libremente por el mapa.

En cuanto a las relaciones entre los propios scripts:

A nivel de personaje, la clase “AgentController” es dependiente de la clase RayCast. Pues desde la clase RayCast , que se desarrollará más adelante su funcionamiento, se controla la visión del NPC, lo que afecta a su movimiento, animaciones y estado que se gestionan desde AgentController.

A nivel de entorno, la clase “WorldGenerator” genera el nivel base, es decir, crea la estructura de salas vacías. Una vez tenemos esta estructura construida se le pasa esta base a la clase “RoomPopulator” para que recorra cada una de estas salas e introduzca los distintos elementos que la componen, que son los elementos decorativos y los NPCs.

El resultado de ambas clases será también el entorno sobre el que se coloquen los elementos de riesgo , el fuego y el humo. Por lo tanto, una vez ya han sido las distintas estancias creadas y pobladas , pasamos esta información a la clase “FireGeneration”. Desde allí utilizaremos esa información para colocar el fuego correctamente en la escena y propagarlo por los límites adecuados del nivel. Por último, una vez hechos los cálculos necesarios y creado el fuego, pasaremos la posición del fuego a la clase “SmokeGeneration” para que tanto el fuego como el humo se propaguen a partir de la misma posición inicial.

A nivel de interfaz de usuario ,cada clase hace referencia a un elemento gráfico que ve el jugador en su pantalla y están colocados dentro del mismo panel. La relación entre

ellos más directa se encuentra para el nivel de estrés, ya que según el valor de estrés que aparece en la barra controlada por “StressSlider”, cambiará el valor del texto que se muestra en el texto que aparece encima del mismo entre los valores: “*Calm*”, “*Stress*” y “*Panic*”. Tanto la clase “StressSlider” cómo“HealthSlider” están relacionadas con “RayCast”, puesto que lo que vea el NPC afectará a su estrés y salud. Por último, dependiendo también de los niveles de estrés y salud afectará al movimiento y reacción del agente , por lo que ambas clases “StressSlider” y “HealthSlider” también estarán relacionadas con “AgentController”.

En cuanto al bloque de gestión del juego, la única clase que mantiene relación con otras es “GlobalVar”. En ella se almacena información compartida entre las diferentes clases o para que sean accesibles desde diferentes instancias de la misma clase. El funcionamiento de esta clase será descrito posteriormente pero desde ella se gestionan elementos de las clases “AgentController”, “FireGeneration”, “SmokeGeneration”, “WorldGenerator” , “RoomPopulator”, “RemainingNPC” , “TotalNPC”, “HealthSlider” y “StressSlider”. Se puede observar esto de forma detallada en la siguiente imagen:

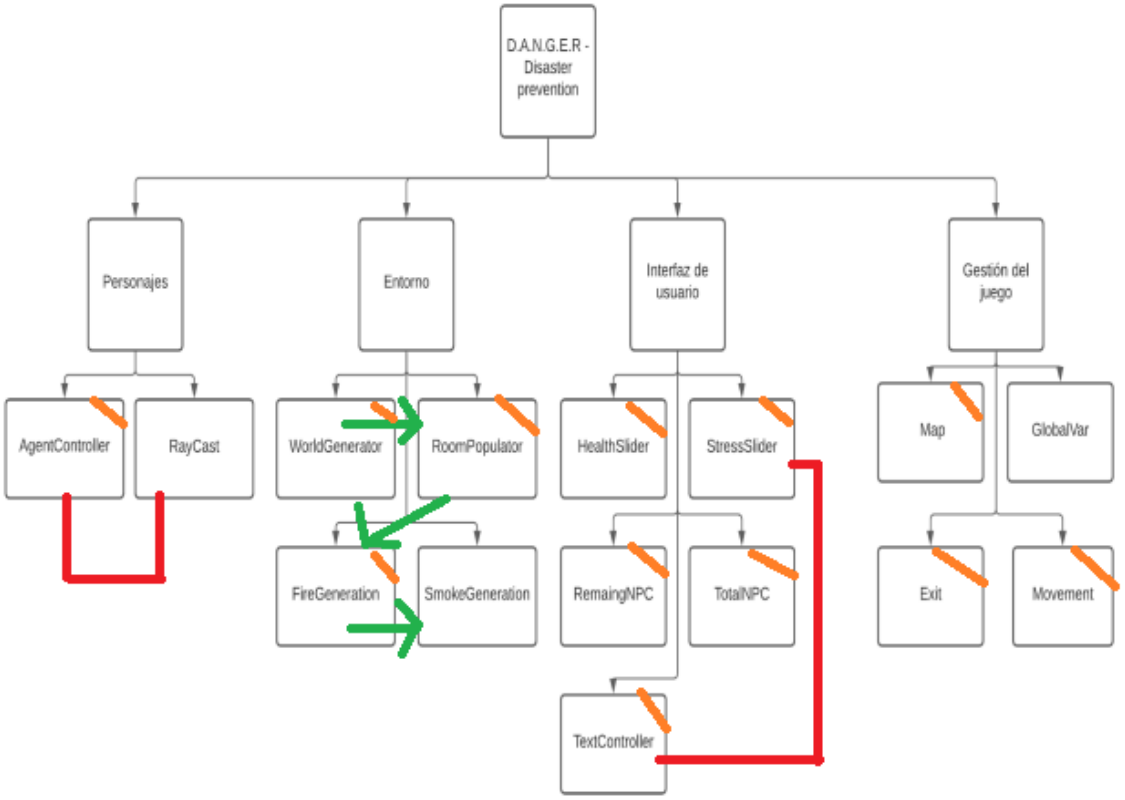


Ilustración 2: Relaciones entre scripts

Aquellos scripts relacionados por una línea roja tienen dependencia directa entre ellos mismos. Los que tienen una flecha verde significa que se pasa información y existe por lo tanto una dependencia de la clase que está es la punta (dependiente) con el script situado en el origen de la flecha. Por último, aquellos scripts que tienen una línea naranja en la esquina superior izquierda son aquellos que almacenan y extraen información del script GlobalVar.

3.3 Algoritmo de barajado de Fisher-Yates

El algoritmo de ordenación o barajado de Fisher-Yates es uno de los algoritmos empleados para este propósito más extendido. Esto es así ya que este algoritmo tiene una serie de ventajas:

- Alto rendimiento y calidad de los resultados. En la práctica, más de un 11% mejor que otro de los algoritmos más usados, el generador lineal congruencial.

- Bajo coste temporal y espacial, generalmente tendrá un coste lineal O(n), ya que se hace un único recorrido por el conjunto de datos a barajar.
- Sencillez de implementación
- No repetición y duplicidad. Cada vez que se utiliza el algoritmo proporciona un resultado distinto para un mismo elemento de entrada.

Pseudocódigo:

```
Fisher-Yates_Shuffle( array a)

    para (i = a.longitud -1 ; i > 0 ; i-- )

        numAle = número aleatorio entre 0 e i;

        temp = a[i];

        a[i] = a[numAle];

        a[numAle] = temp;

    finpara
```

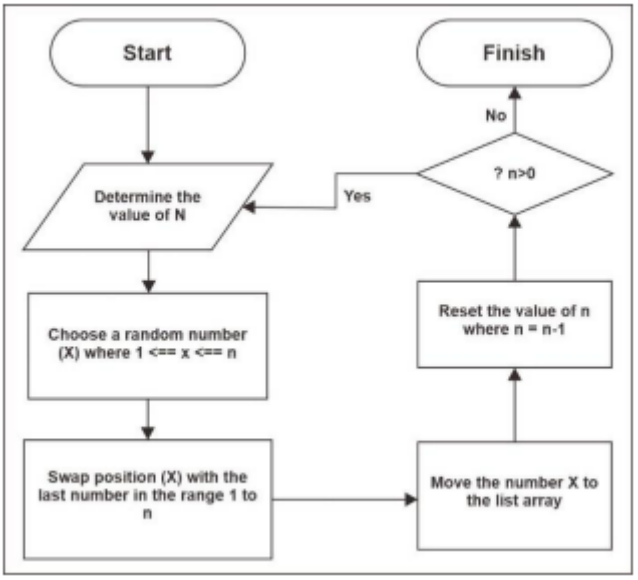


Figura 5: Implementation of the Fisher-Yates Shuffle Algorithm in Exam-Problem Randomization on M-Learning Application [2]

Descripción del algoritmo:

El algoritmo consiste en recorrer de forma descendente todas las posiciones de la estructura de datos a barajar. Para la primera iteración se tomará el último elemento de la estructura, para la siguiente el penúltimo y así sucesivamente. En cada una de estas iteraciones, tomaremos un valor al azar entre 0 y el número de posiciones que quedan por barajar. Por ejemplo, si tenemos un array de 6 elementos y estamos en la segunda iteración, escogeremos un valor entre 0 y 4. Este valor será la posición de la estructura de donde vamos a hacer el intercambio. Tomamos el valor de la posición donde nos encontramos (número de elementos totales - iteración actual) y lo guardamos en una variable temporal. A continuación guardamos en esta posición el valor de la posición del número aleatorio que hemos escogido anteriormente. Por último restauramos el valor de la variable temporal en la posición aleatoria que habíamos escogido anteriormente.

3.4 NavMesh

NavMesh es una funcionalidad de Unity que permite la creación de una malla sobre la cual se mapean las zonas por donde se pueden desplazar los diferentes agentes o personajes del videojuego. Para ello, cuenta con diferentes componentes.

- NavMesh Surface: Genera la superficie por la cual pueden caminar los agentes. Se pueden especificar las capas sobre las que se quiere permitir que estos se desplacen, además de permitir ajustar las características físicas del agente como su tamaño, altura, velocidad y la capacidad que tienen para subir peldaños o la inclinación máxima de las cuestas que pueden subir.
- NavMesh Agent: Se sitúa sobre el agente inteligente. Dota al agente de la capacidad de calcular las rutas entre los diferentes puntos de la superficie y de desplazarse entre los mismos dado un punto que pertenezca a la misma. Permite configurar

- NavMesh Obstacle : Se sitúa sobre un objeto. Permite que el agente trate de esquivar el objeto que incorpora este componente. Esto es especialmente útil para objetos dinámicos donde no podemos calcular la posición del objeto para esquivarlo a partir de la malla que genera “NavMesh Surface”.

La finalidad de esta herramienta es buscar el camino más corto entre el punto actual y otro punto presente en la malla.

Funcionamiento:

La búsqueda del camino más corto entre un punto A y un punto B consiste en dividir el terreno en polígonos convexos. A partir del primer punto, buscamos el camino más óptimo conectando los diferentes vértices de los polígonos por los que se van pasando hasta llegar al mismo polígono donde se encuentra el punto B. Una vez ambos puntos están en el mismo polígono , el camino restante será igual a una línea recta entre los puntos A y B.

3.5 Raycast

Citando el “Estudio y simulación de un vehículo autopilotado en Unity 5 haciendo uso de algoritmos de aprendizaje automático” [5] : “El raycasting es un proceso consistente en disparar un rayo invisible desde un punto hacia alguna dirección con la idea de detectar los objetos que colisionan con dicho rayo. ”

Este proceso nos permite obtener información sobre los objetos con los cuales ha colisionado, ya que podemos obtener elementos como su nombre, etiqueta u otra información que podamos obtener a partir del collider del objeto.

La información de la colisión se almacenará en un objeto del tipo “RaycastHit”. Este objeto contiene la información sobre la distancia, el punto de impacto (coordenadas en forma de vector) y el componente “Rigidbody”, si tiene uno.

Para lanzar un rayo se hace empleando la función “Raycast” de la librería de físicas de Unity. Esta función recibe como parámetros 2 vectores y un número. El primer vector

hace referencia al origen, es decir, el punto en el espacio desde el cual se dispara el rayo. En el segundo se indicará el vector de dirección. El último parámetro se corresponde con la distancia máxima que puede alcanzar este rayo (si no se proporciona este parámetro el rayo se alargará indefinidamente hasta que colisione con un objeto).

3.6 Personajes

En este punto se detalla el manejo tanto de forma autónoma como de los personajes que participan en el juego, así como la percepción de su entorno y la consecuente reacción.

3.6.1 Control de los personajes

En este apartado se detalla la navegación autónoma de los personajes en el nivel del juego, así como la gestión del estrés, la salud y la interacción con el entorno. Estos comportamientos son los que se detallan en el script “AgentController”.

En primer lugar, para lo referente a la navegación de los personajes hacemos uso de los NavMesh de Unity explicados anteriormente. Utilizando el componente del agente en nuestro prefab del personaje este complemento nos calcula la ruta óptima para alcanzar cualquier punto que se encuentre dentro del NavMesh.

El juego está planteado como un juego de estrategia en tercera persona donde puedes seleccionar y controlar a los distintos agentes del juego utilizando el cursor del ordenador.

Para desplazar un personaje de forma manual, se situará el cursor sobre uno de los personajes y, pulsando el clic izquierdo este quedará seleccionado. El personaje

seleccionado aparecerá resaltado en un color más oscuro que el resto y sus características de estrés y salud aparecerán en las barras de la interfaz de usuario.

Ilustración 3: Personaje sin selección

Ilustración 4: Personaje seleccionado

Respecto al desplazamiento de los personajes de forma autónoma el planteamiento principal consiste en tener un temporizador que se activa cada vez que un personaje deja de estar en movimiento, es decir, llega a su destino. Durante este tiempo el personaje se encuentra en un estado de reposo esperando. Una vez termina el tiempo se elige un punto aleatorio de una sala elegida también de forma aleatoria y se le indica al personaje de desplazarse hasta allí. Una vez el personaje llega hasta el punto elegido vuelve a empezar el temporizador. El jugador puede intercalar entre ambos modos, es decir, el personaje por defecto se encontrará investigando el mapa en modo autónomo, pero el jugador puede seleccionarlo e indicarle el lugar donde quiere dirigirlo.

En el momento que el personaje llegue al lugar donde el jugador le había designado el personaje volverá a entrar en modo autónomo y comenzará el temporizador. Cualquier temporizador que ya hubiera sido activado volverá a empezar desde el principio si un jugador selecciona a un personaje y le da un destino de forma manual.

Desde este script del controlador del agente también se gestionan los niveles de estrés y de salud. La detección del fuego, por medio de trazado de rayos se hace desde la visión del personaje, sin embargo es desde este script donde se afectará al comportamiento del personaje.

Por ejemplo, en cuanto se detecte el fuego, se pondrá al personaje en un modo autónomo especial cuyo objetivo es imitar el pánico humano al verse en una situación donde se encuentre rodeado de fuego. En este modo el personaje ignorará todos los temporizadores y buscará continuamente una ruta y un punto del mapa al que pueda desplazarse tratando de esquivar el juego.

Este comportamiento imita una reacción instintiva que tendría una persona, sin embargo, lo más probable es que desvíe al jugador del objetivo de sacar a los distintos personajes del nivel en la salida y el personaje se dirija a un lugar completamente diferente mientras no se encuentre fuego en su camino, tratando de imitar lo que haría una persona desorientada tratando de escapar del mismo.

Desde aquí también se aumentará el nivel de estrés una vez sea detectado. Además, el nivel de estrés de los personajes irá disminuyendo gradualmente con el paso del tiempo, siempre y cuando no se encuentren de nuevo con elementos de riesgo como son el fuego y el humo. El nivel de salud, sin embargo, no será regenerativo.

Por último, desde este script también se controlan las animaciones de los personajes. Una vez un personaje comienza a desplazarse el personaje activa una animación de caminar y cuando se detiene activa una animación de "*idle*" (reposo). Para averiguar cuando un personaje se encuentra realizando un desplazamiento lo que tendremos que hacer es consultar si tiene un camino asignado en el componente de agente del NavMesh.

3.6.2 Visión de los personajes

La visión de los personajes ha sido implementada con el uso de raycasts. Para ello cada personaje tendrá un determinado número de rayos, que puede ser modificado y parametrizable. Cuánto mayor sea el número de rayos mayor será la precisión a la hora de detectar objetos pero habrá un mayor consumo de recursos.

Los rayos actúan como el campo de visión del agente. Cuando un objeto o elemento colisiona con uno de estos rayos el agente recibe información acerca de la distancia a la que está de él . De este modo, por ejemplo, puede detectar el fuego o el humo y conocer la distancia a la que está de él para esquivarlo. Para ello, una vez el rayo colisiona con un objeto recuperamos su etiqueta. Si esta es igual a la etiqueta correspondiente del fuego y el humo aumentamos el nivel de estrés y provocamos una reacción en el personaje. El ángulo de visión también es configurable. De este modo se puede hacer que el personaje tenga un mayor o menor grado de visión periférica. En el caso de aumentar este valor hay que tener en cuenta que los rayos quedan más dispersos y por lo tanto se puede estar perdiendo precisión si no se aumenta el número de los mismos.

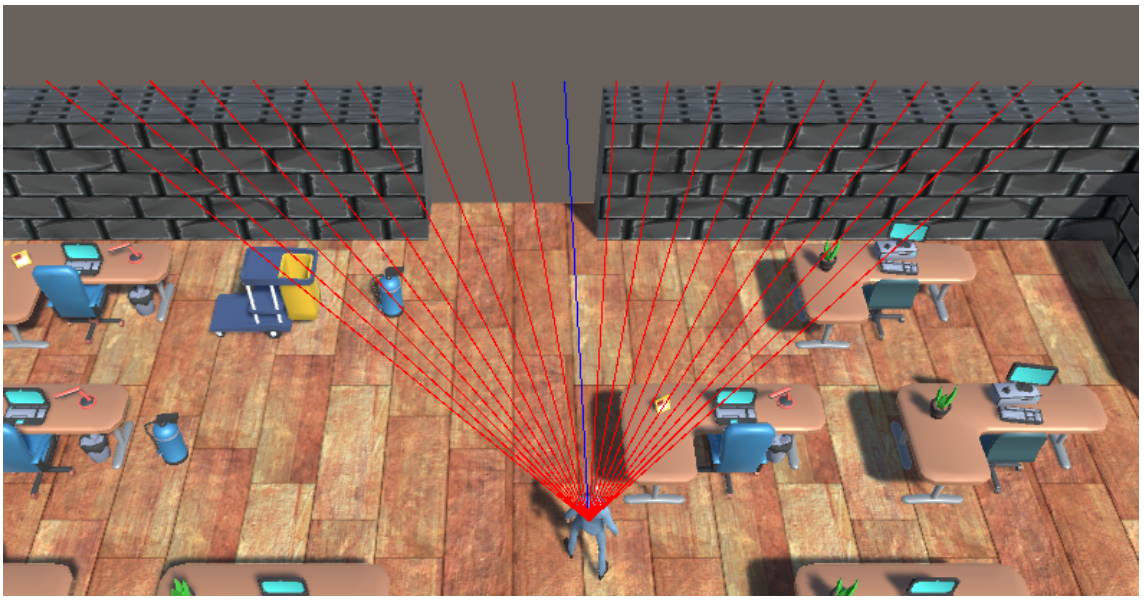


Ilustración 5: Visión del personaje

3.7 Entorno

En este punto se detallarán los distintos aspectos de la generación de los niveles y población de las salas.

3.7.1 Generación procedural de las salas

En este apartado se detalla la generación procedural de las salas, es decir, como se genera la estructura base de salas de forma rectangular de forma que estas acaben siendo conexas y teniendo un aspecto y una distribución coherentes y lo más realista posible. El comportamiento que se explica aquí es el que se desarrolla en el script “WorldGenerator.cs” .

Para la creación de una sala necesitaremos únicamente 2 tipos de objetos distintos: las paredes y el suelo. Las paredes de crearán a partir de un objeto de forma rectangular (creado a partir del objeto base de Unity: “Cubo”) con ancho y largo de tamaño 1 de forma obligatoria para después poder redimensionarlo fácilmente a nivel de código, la altura es completamente independiente y no afecta al escalado por lo que en este caso se optó con un objeto de una altura de 4 para que, en proporción a los personajes y el mobiliario quedase a una escala realista. El suelo será un plano (creado a partir del objeto de Unity : “Plano”) de ancho y largo a 1 también para poder redimensionarlo.

El planteamiento de la solución es crear para cada una de las salas 4 puntos , uno en cada lateral de forma rectangular. Estos puntos serán los “generadores”. La función de un punto generador será dar a entender al algoritmo que a partir de esa posición se puede crear (o no) una nueva sala.

Para definir estos “puntos generadores” se ha creado una nueva estructura llamada “generatorPoint”. En este objeto almacenaremos las coordenadas de este punto generador y la dirección que tiene el mismo, para saber si se trata de una sala que se crearía en la parte superior, inferior, derecha o izquierda de la sala actual.

El script tendrá la condición de que mientras sigan quedando habitaciones por generar tratará de generar una habitación a partir de cada uno de los puntos generadores de las anteriores salas.

Para hacer esto crearemos una cola de prioridad en la que almacenaremos los puntos generadores de cada una de las salas que creemos. De esta forma, una vez se cree una sala, añadiremos sus 4 puntos generadores a esta cola. Al añadir una nueva sala sus 4 generadores se añadirán después de los de la sala anterior, de este modo

hacemos que las salas se expandan de una forma relativamente uniforme y más semejante a la realidad, evitando distribuciones anómalas.

Para crear cada una de las salas tomaremos un “generador” de la cola de prioridad y probaremos a crearla a partir del mismo. Dependiendo de la dirección de la sala (arriba, abajo, derecha o izquierda) obtendremos las coordenadas del centro de la sala que vamos a tratar de crear. Si la sala se está creando en un lateral de la anterior le sumamos (o restamos) a las coordenadas del generador la mitad del ancho de la nueva sala. Si la sala está creándose arriba o abajo le sumamos (o restamos) la mitad del largo de la nueva sala.

Una vez que tenemos la posición central de nueva la sala lo que tendremos que hacer es comprobar que la sala que estamos tratando de crear no se solapa con ninguna otra que ya haya sido creada previamente. Para ello se define la estructura para el objeto “occupiedArea”. Crearemos un objeto de este tipo para cada sala que creemos y guardaremos todos ellos en una lista. Este objeto almacenará los valores máximos y mínimos de las coordenadas en X y en Z de una sala. Tener estos valores máximos y mínimos nos permite conocer qué área va a ocupar la sala en el espacio.

Como se puede apreciar en la siguiente ilustración, si almacenamos al crear una sala, a partir de las esquinas inferior izquierda y superior derecha, las coordenadas correspondientes a las máximas y mínimas X y Z podemos después extrapolar el área que ocupará una sala.

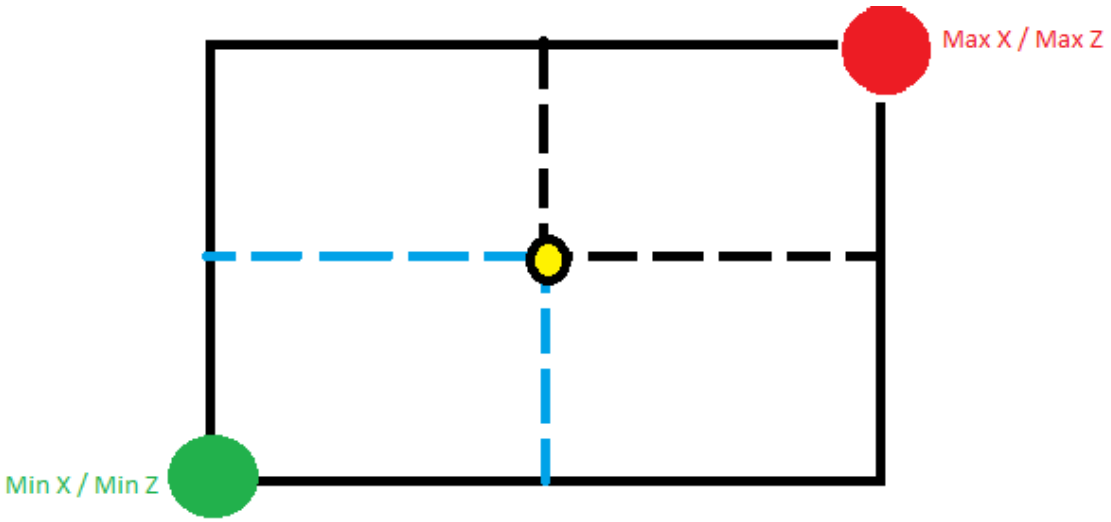


Ilustración 6 : Coordenadas generación procedural

Por lo tanto, para evitar solapamientos, lo que haremos será comprobar que ninguna posición (X y Z) de la sala nueva que se está tratando de crear se encuentra dentro de los intervalos:

- Mínimo X [sala i] > X > Máximo X [sala i]
- Mínimo Z [sala i] > Z > Máximo Z [sala i]

Dónde [sala i] hace referencia a cada una de las salas creadas previamente.

Si esta condición se cumple entonces instanciamos los objetos de las paredes y el suelo de la sala. Para crear las paredes utilizaremos el definido previamente, heredado del cubo. Colocaremos 8 cubos en las posiciones que se pueden apreciar en la siguiente ilustración:

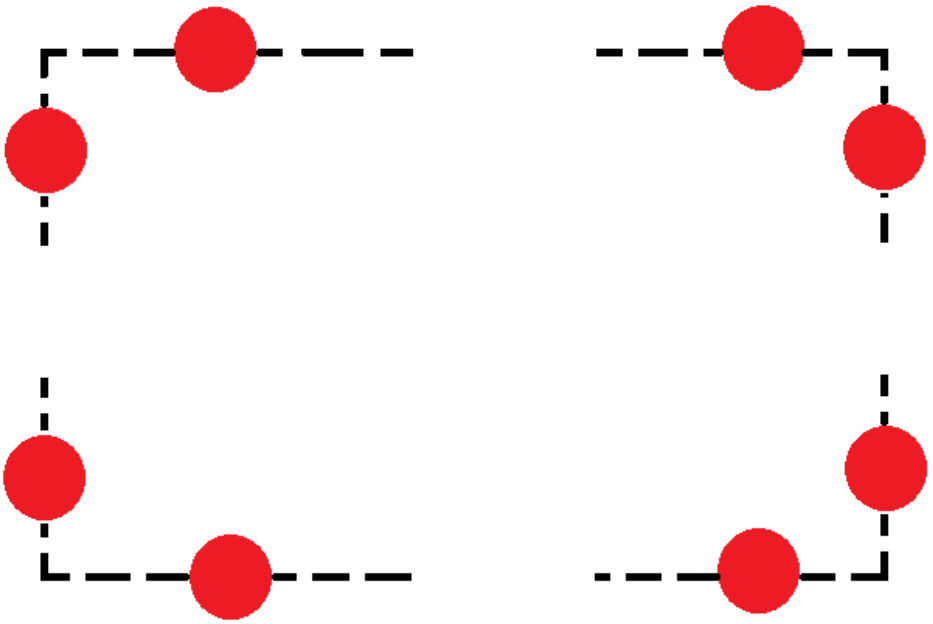


Ilustración 7: Posiciones cubos, generación procedural de salas

Estas posiciones se corresponden a la mitad entre la abertura dejada para la puerta (pudiendo modificarse el tamaño de esta abertura) y de cada una de las esquinas de la sala.

Una vez tenemos los cubos colocados, los redimensionamos con el tamaño entre la esquina y la abertura, al tener ancho y largo igual a uno quedarán ajustados a este espacio quedando como la siguiente ilustración:

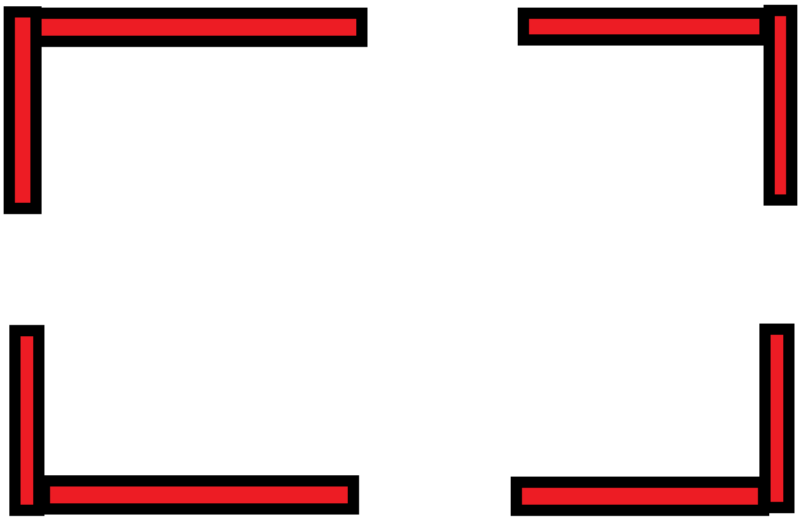


Ilustración 8: Creación de las paredes de las salas

A continuación añadimos el plano que actuará como el suelo de la estancia en el centro de la misma, y lo redimensionamos escalando en anchura y altura con las dimensiones de la sala al tener también las proporciones de ancho y alto con tamaño igual a 1.

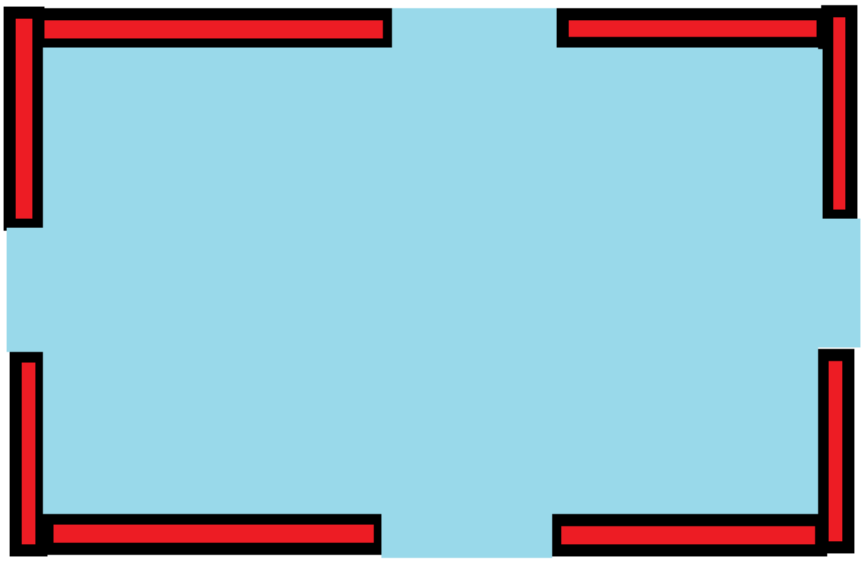


Ilustración 9: Creación del suelo de las salas

Una vez ya tenemos la sala como tal generada, queda únicamente añadir los puntos generadores en el centro de cada uno de las puertas, es decir, en la mitad de cada

lado de la sala. Sin embargo, como cuando creemos estos puntos será para añadirlos a la cola , si fijamos el orden en el que se crean estos, por ejemplo en sentido horario: arriba, derecha, abajo, izquierda , las salas se crearán siempre en estructuras muy similares.

Como lo que queremos conseguir generando estos niveles de forma procedural es realmente lo contrario justamente para fomentar la originalidad de los niveles y fomentar la rejugabilidad y aparición de diferentes escenarios y situaciones, lo que haremos será cambiar el orden en el que se crean estos puntos generadores. Para ello, introduciremos el algoritmo de barajado de Fisher Yates.

Barajando el orden en el que se añadirán a la cola los puntos generadores utilizando el algoritmo de Fisher Yates conseguimos una pseudo-aleatoriedad de forma que las estructura de salas que se genera finalmente es mucho más variada, y, teniendo en cuenta que se añaden todos los puntos de la sala a la cola antes de crear nuevas salas, nos aseguramos de seguir manteniendo una estructura coherente de forma que se vaya expandiendo poco a poco de forma relativamente equilibrada y que no nos van a surgir formas anómalas.

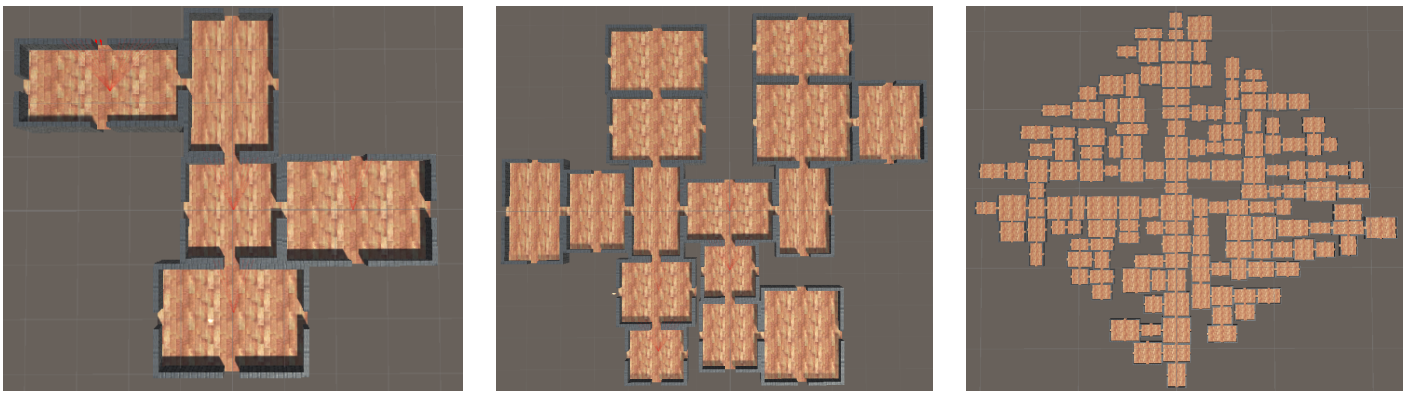


Ilustración 10: Ejemplos de generación para niveles de 5, 15 y 150 salas

3.7.2 Población de las salas

En este apartado se detalla el comportamiento del script “RoomPopulator”, en el que, tras recibir la estructura de salas vacías descritas en el anterior paso será el encargado de insertar los diferentes elementos decorativos y de mobiliario en las diferentes salas.

El planteamiento consiste en que a partir de una lista de objetos proporcionada y de las salas previamente creadas insertarlos, dejando entre ellos un margen establecido, de forma que respeten la probabilidad de aparición de cada objeto.

Para ello, en primer lugar, recuperamos de la clase “WorldGenerator” 3 listas. La primera contiene las coordenadas de cada una de las estancias, la segunda las anchuras y la tercera los largos.

A continuación, recorremos cada una de las estancias y recuperando la estructura de “occuppiedArea” crearemos 2 “pasillos” de forma que las entradas de las salas queden siempre accesibles y no haya nunca ningún problema a la hora de pasar entre diferentes salas y se pudiera dar un bloqueo. Esto lo haremos creando 2 “occupiedAreas” (estructura que utilizamos para almacenar los máximos y mínimos en cada coordenada X y Z para una sala) que tendrán en una dimensión (X o Z) el tamaño del margen de la puerta y en la otra el largo o el ancho de la sala. Añadiremos estos pasillos a una lista de “occupiedAreas” llamada “restrictedAreas”en donde guardaremos todas aquellas áreas que están ocupadas por otros objetos (o pasillos).

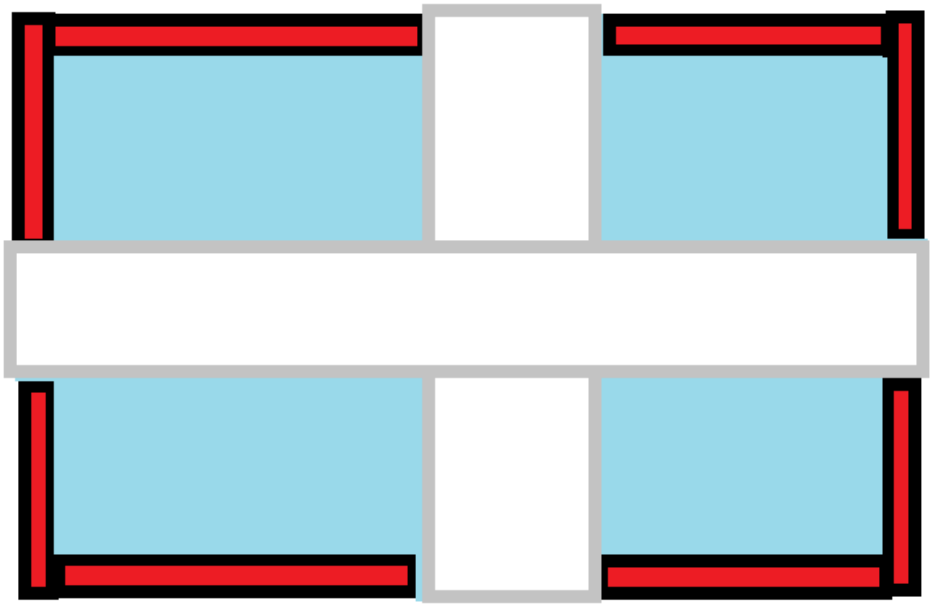


Ilustración 11 : Creación de los pasillos

También guardaremos la información de los máximos y mínimos de la sala para poder tener disponible el área entera de la misma y crear las restricciones al tratar de introducir los objetos.

Una vez tenemos toda la información de las salas y el espacio de los “pasillos” reservado comenzamos a introducir objetos de la lista que nos ha sido proporcionada. Cabe recordar que el algoritmo está preparado para aceptar cualquier objeto que se quiera introducir y es completamente adaptable, el único requisito es que el objeto tenga una malla de renderización y un colisionador, pudiendo este ser de malla, caja, cápsula, etcétera.

La forma de introducir los objetos será, en primer lugar, escoger el objeto a introducir. Esto lo haremos utilizando el algoritmo de barajado de Fisher Yates. Cada vez que vayamos a introducir un nuevo objeto se escogerá el primer elemento de la lista de objetos que habrá sido reordenada con este algoritmo. A continuación, obtenemos las dimensiones del objeto y calculamos sus 4 esquinas o, mejor dicho sus coordenadas máximas y mínimas hacia cada dirección de los ejes X e Y ya que el objeto puede no tener forma rectangular.

Una vez disponemos de toda esta información lo que deberemos hacer es comprobar si podemos situar el objeto en la sala.Para esto iremos recorriendo la sala como si se

tratase de una matriz dividiendo el espacio de la sala en celdas de 1x1 que serán las posiciones que avanzaremos como filas y columnas. Recorremos las salas desde la esquina superior izquierda hasta la esquina inferior derecha. Tendremos 2 tipos de comprobaciones:

1. Comprobaremos que todas las coordenadas máximas y mínimas (esquinas) del objeto se encuentran efectivamente dentro del área de la sala. Para ello, recuperaremos la información de los máximos y mínimos de la sala y comprobaremos que las coordenadas de cada una de las esquinas se encuentran dentro del intervalo permitido por los máximos y mínimos de la sala.
2. Comprobaremos que máximas y mínimas (esquinas) del objeto no se encuentran dentro de ninguna área restringida. Para ello recorreremos la lista de áreas restringidas, donde se han ido introduciendo las áreas de los objetos colocados en el nivel y comprobaremos que las esquinas del objeto no se encuentran dentro de ninguna de estas áreas mirando si están en los mismos intervalos de coordenadas X y Z.

Si alguna de estas 2 comprobaciones se incumplen no colocaremos el objeto. Si por lo contrario, se satisfacen ambas condiciones, colocaremos el objeto en el nivel y añadiremos el área que está ocupando a la lista de “restrictedAreas” para futuras comprobaciones.

En el recorrido de la sala no avanzaremos de posición en posición, sino que, una vez hayamos colocado un objeto y elegido mediante el algoritmo de barajado de Fisher Yates el siguiente objeto, calcularemos el tamaño de este nuevo objeto y nos trasladaremos en el espacio de la sala lo suficiente como para tratar de que sea una ubicación potencial para el objeto.

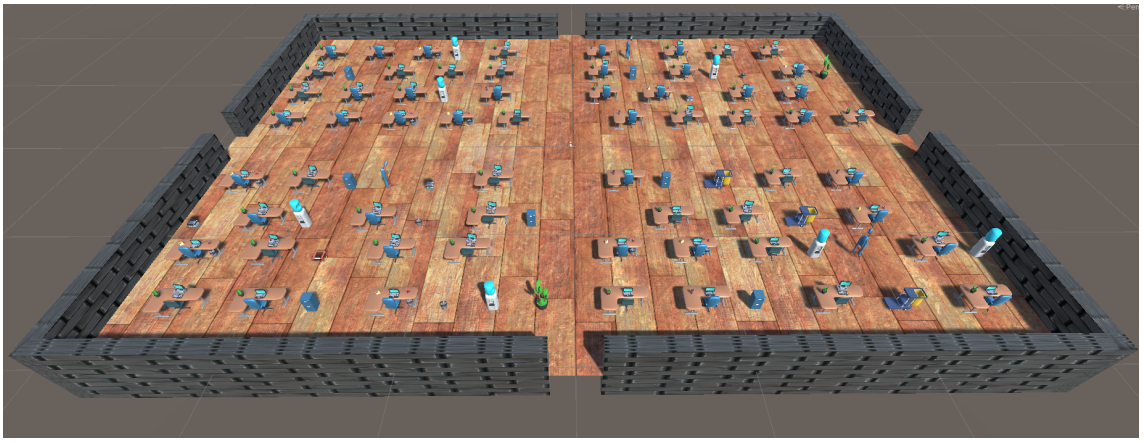


Ilustración 12: Ejemplo de población de sala

3.7.3 Generación del fuego

En este apartado se detalla el comportamiento del script “FireGeneration”, en el que, tras haber creado el nivel completo con todas las estancias, se añadirá el fuego y se controlará su comportamiento y propagación.

La solución planteada consiste en la creación de una matriz booleana, cuyas posiciones puedan ser trasladadas al nivel real, donde se indique en que posiciones puede tanto aparecer como propagarse el fuego para dar el comportamiento más realista posible y, a partir de ese punto, generar los distintos fuegos con unos intervalos de tiempo parametrizables.

Para resolver esto, en primer lugar , crearemos la matriz que representará el nivel. Las dimensiones de esta matriz serán:

- Filas: Diferencia entre la máxima y la mínima largura (coordenada Z) entre todas las salas.
- Columnas: Diferencia entre la máxima y la mínima anchura (coordenada X) entre todas las salas.

Para calcular estas , simplemente calcularemos los puntos más y menos anchos y altos de cada una de las salas, o para no tener que volver a calcularlo, podemos hacer este cálculo también a la hora de crear las salas en “WorldGenerator”.

Una vez tengamos esto, inicializamos la matriz booleana a falso y la recorreremos para indicar con verdadero aquellas posiciones que se corresponden con una posición donde exista una sala. No solo eso, sino que además el fuego no debe de aparecer en las paredes de las habitaciones y debe propagarse entre las diferentes estancias únicamente a través de las puertas que hay entre las mismas. Para ello, transformaremos las posiciones de la matriz en posiciones reales del nivel y comprobaremos que las posiciones se corresponden con los intervalos reales de las salas, y excluimos aquellas posiciones sobre las que se vayan a colocar las paredes.

Una vez tenemos la matriz completamente inicializada, intentaremos escoger una posición aleatoria de la misma hasta que esta tenga valor verdadero. Esta posición será la posición a partir de la cual comenzará a propagarse el fuego por el escenario.

Para crear el fuego se llamará a una función recursiva llamada “generarFuego”. A esta función se le darán como parámetros la posición en la que se intentará crear el fuego. Para que el fuego se cree tendrá que satisfacer 2 condiciones, la primera es que los valores de las coordenadas X y Z estén dentro los valores de la matriz y la segunda será que la el valor de la matriz para matriz[X,Z] sea verdadero, es decir, que la posición real en el nivel que se corresponde con esa posición en la matriz caiga dentro de una parte de la sala que no sea una pared. Si cumple estas condiciones crearemos el fuego en esa posición y además lanzaremos 4 llamadas recursivas para avanzar una posición en todas las direcciones,es decir:

- generarFuego (x+1, z)
- generarFuego(x-1, z)
- generarFuego(x, z+1)
- generarFuego (x, z-1)

Tras cada una de estas llamadas esperaremos unos segundos para recrear una propagación más realista del fuego. Este tiempo se puede configurar para ajustar la dificultad del juego. También se podrían añadir varios focos de incendio y que se propaguen desde 2 puntos del mapa distintos. Mientras que las posiciones adyacentes pertenecen también las salas se seguirán creando.

3.7.4 Generación del humo

La generación del humo se hace de manera paralela al fuego. Partiendo un duplicado de la matriz booleana creada en el script “FireGeneration”, el comportamiento de “SmokeGeneration” es el mismo que el del fuego, a excepción de que variamos los tiempos de propagación para que estos sean más rápidos y el humo se expanda por las salas más rápido que el fuego, además de que cambiaremos el objeto prefabricado del fuego por el de partículas de humo gris. De esta forma el humo llegará primero hasta los personajes como haría en la vida real y creará un entorno mucho más realista.

3.8 Interfaz de usuario

En este apartado se detallarán los diferentes elementos de la interfaz de usuario, cuya utilidad es proporcionar información relevante al jugador durante el transcurso del juego

3.8.1 Nivel de estrés y salud

Los niveles de estrés y salud son la representación en el juego, para dotarlo de un mayor realismo, del estado mental y físico de cada uno de los personajes.

Respecto al estrés, este representa el estado mental del personaje. Puede visualizarse en la barra amarilla de la esquina inferior derecha. El jugador tendrá 2 elementos visuales que le ayudarán a identificar el estrés. En primer lugar la barra que indica el estrés de forma visual y en segundo lugar el texto, que irá alternando entre “Calm”, “Stress” y “Panic” según incrementa o decrementa el nivel de estrés. El nivel de estrés aumentará cuando un NPC detecte fuego o humo en su campo de visión y disminuirá cuando lleve un periodo de tiempo sin ver ninguno de estos elementos. Cuando el nivel de estrés alcanza el estado de “Stress” disminuirá el tiempo de espera para navegar autónomamente de un lugar a otro en 1 segundo y se aumentará la velocidad del personaje un 25%. Cuando el nivel de estrés alcanza el estado de “Panic”

disminuirá el tiempo de espera para navegar autónomamente de un lugar a otro en 1 segundo adicional, (para un total de 2 segundos menos respecto del estado de calma) y se aumentará la velocidad del personaje un 50% respecto con la que tienen los personajes en calma.

Ilustración 13: Medidor de estrés

Respecto a la salud, representa el estado físico del personaje. Esta tiene forma de barra y puede visualizarse en la barra roja de la esquina inferior izquierda. Cuando el personaje se encuentre cercano o en contacto al humo y fuego su salud disminuirá como haría en un caso real por inhalación de humo o tras sufrir quemaduras. Cuando lo haga, esta barra disminuirá en proporción al daño que ha recibido.

Ilustración 14: Medidor de salud

3.8.2 Indicadores de NPCs restantes y totales

Con la finalidad de que el jugador pueda conocer el estado de la partida, se han añadido 2 contadores. El primero de ellos indica el número total de NPCs que hay presentes en la escena cuando se crea el nivel. De esta forma el jugador puede

localizarlos desplazándose con el mapa para tenerlos localizados e ir organizándolos. El segundo contador es el de NPCs restantes, con este contador el jugador puede gestionar cuantos de los NPCs no han llegado hasta la salida del nivel. De esta forma puede ver si hay alguno que se haya extraviado y hacer un seguimiento de la partida de una forma más cómoda y rápida.

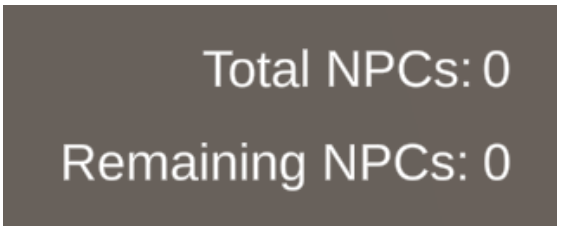


Ilustración 15: Contadores de personajes

3.9 Gestión del juego

En este apartado se detallarán una serie de comportamientos que no se pueden clasificar dentro de un único bloque, estos aportan funcionalidad adicional al juego.

3.9.1 Mapa

El mapa es una funcionalidad adicional del juego que permite al jugador 2 acciones adicionales : posicionarse en el escenario y localizar la salida.

Estos dos aspectos son clave para la jugabilidad, ya que, de generarse un número relativamente grande de salas comienza a ser complicado hacer un seguimiento de donde se encuentra en cada momento ya que los entornos pueden ser similares. Para ello, como si se tratase de un plano de emergencia situado en una de las paredes del edificio en donde está ocurriendo la emergencia, el jugador puede visualizar este plano dando así a conocer tanto su ubicación dentro del escenario señalada con un punto rojo como la salida de emergencia hasta la cual tiene que dirigir a los NPCs señalada con un cuadrado verde.

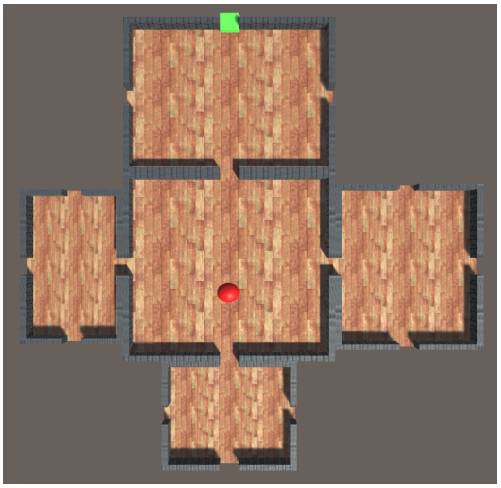


Ilustración 16:Mapa

3.9.2 Variables globales

Como se especifica en el apartado de relaciones entre clases, hay muchos scripts que dependen de otros o, simplemente, se quiera manejar varias instancias de un mismo script y hacer alguna operación o control entre ellos. Es por ello que se dispone de la clase “GlobalVar” donde se gestionan estos aspectos.

En primer lugar, se hace uso de una variable para almacenar el ID del agente que se encuentra seleccionado. De esta forma se puede hacer una sección crítica donde se puede gestionar que únicamente el agente seleccionado se vea afectado por la próxima acción del jugador.

A la hora de comunicar los niveles de estrés y salud de los distintos personajes, estos serán almacenados en una variable global que será la que se esté leyendo en la interfaz de usuario para mantener actualizadas las barras y los textos correspondientes.

Para poder generar el entorno de forma correcta, hay información, como las coordenadas de las salas y sus dimensiones que son requeridas por “RoomPopulator”, “FireGeneration” y “SmokeGeneration”. Como estos datos se van pasando en el entre las clases mencionadas anteriormente desde “WorldGenerator” una vez se creen se almacenarán en estas variables globales para que puedan ser accedidas por todas las

instancias de estas clases sin necesidad de una comunicación contínua entre ellas que pudiera modificar su valor.

También almacenaremos aquí las coordenadas centrales del nivel para poder generar correctamente el mapa y gestionaremos del mismo también si está activo o no para poder apagar, encender y controlar el movimiento de las distintas cámaras que se están empleando.

Por último, se almacenarán para mantener actualizados también los datos del total de personajes y personajes restantes a mostrar posteriormente en la interfaz de usuario.

3.9.3 Movimiento de cámara

Con la finalidad de que el jugador pueda gestionar de una forma mucho más comoda y visual, la cámara está situada de forma cercana a las salas en lugar de encontrarse alejada y tratar de ver el mapa completo al mismo tiempo (es para esto que está el mapa). Esto introduce un problema que es que el jugador solo puede ver una fracción de mapa al mismo tiempo. Para solucionar esto, la cámara es controlada por el jugador y puede moverse de manera vertical y horizontal utilizando las teclas W (arriba),A (izquierda),S (abajo) y D (derecha) o las flechas de dirección del teclado.

La cámara dispone de una velocidad base uniforme, sin embargo, si el jugador desea desplazarse más rápido, puede mantener pulsada la tecla shift para incrementar la velocidad del desplazamiento de la cámara al doble de la velocidad original.

3.9.4 Salida de los personajes

El script “Exit”, cuyo comportamiento se detalla a continuación, define el cumplimiento del objetivo principal de la jugabilidad, que es el que un personaje llegue hasta la salida de emergencia que se puede visualizar en el mapa.

Una vez un personaje haya llegado hasta la posición asignada, este desaparecerá y se restará 1 al valor que se encontraba previamente para los NPCs restantes de salir.

Una vez todos los personajes hayan salido, el juego terminará y se cambiará a la escena de créditos.

4. Conclusiones

En este apartado vamos a proceder a analizar los resultados obtenidos de la realización del proyecto , lo relacionamos con estudios cursados durante el transcurso del Grado en Ingeniería Informática y comentaré las posibles ampliaciones que se podrían haber realizado al proyecto de haber tenido más tiempo disponible para realizarlo.

4.1 Análisis de los resultados

El resultado obtenido de la realización del proyecto puede considerarse exitoso o satisfactorio ya que se ha conseguido desarrollar un videojuego en tres dimensiones utilizando el motor de Unity completamente funcional.No solo eso, sino que además el producto resultante, aunque no perfecto, es visualmente llamativo y apela a su uso.

Además, se han conseguido desarrollar y aplicar todos los objetivos principales o puntos claves del juego como son la generación procedural de los niveles y el control autónomo de los personajes y el entorno. De forma personal, este proyecto me ha dado la oportunidad de profundizar mucho más mis conocimientos en el desarrollo de videojuegos y en este motor a partir de las nociones básicas que teníamos de la carrera. Del mismo modo, este proyecto también ha servido para aprender a gestionar mejor el tiempo y llevar una planificación, así como a manejar herramientas colaborativas y de sincronización como GitHub que me ayudarán en el futuro mundo laboral.

4.2. Relación con los estudios cursados

El proyecto sobre el que estoy realizando el TFG está íntimamente relacionado con mis estudios académicos. Esto se debe a que el proyecto está enfocado principalmente a los aspectos de comportamiento autónomo y generación procedural de niveles que son aspectos que se trabajan en la rama de Computación del grado en Ingeniería Informática.

La lógica que se ha utilizado ha estado inspirada en asignaturas cursadas anteriormente tales como “Sistemas Inteligentes” y “Agentes inteligentes”. En la propia asignatura de “Agentes Inteligentes” programamos unos NPCs para que compitan en un juego de capturar la bandera y, de hecho, hacer ese proyecto fue una de las razones por las cuales se me ocurrió el realizar este nuevo proyecto.

Durante el transcurso del proyecto he aplicado conocimientos ya no exclusivamente de la rama de Computación, sino se han aplicado todos los conocimientos vistos de programación y desarrollo de software. Se ha seguido un planteamiento de un proyecto de ingeniería, con un planteamiento previo, planificación, desarrollo, pruebas y preparación del producto final. Además se han aplicado también conocimientos vistos en asignaturas de algorítmica y estructuras de datos.

Se han utilizado por ejemplo algoritmos de ramificación y poda para la generación y propagación de los fuegos. He utilizado también, y sobre todo, los conocimientos adquiridos sobre el motor y el funcionamiento de Unity que se dieron en la asignatura de “Introducción a la Programación de Videojuegos” como base para muchos aspectos del juego. Por último también se han aplicado los conocimientos de las máquinas de estados vistos en la asignatura de teoría de autómatas y lenguajes y en a la hora de crear los distintos estados físico-psicológico de los personajes entre los que deben de ir alternando.

4.3. Trabajo futuro

Aunque se han conseguido satisfacer todos los objetivos , hay ciertos aspectos del juego que me habría gustado añadir para aportar funcionalidad extra al mismo que por razones de tiempos no han sido posible incluir.

En primer lugar, una mejora interesante sería la capacidad de los NPCs de coger objetos ya presentes en la sala, como son los extintores, y poder hacer uso de los mismos. Mi idea era que una vez obtengan el objeto y se aproximen a los fuegos puedan utilizarlo para poder abrirse paso y hacer un pequeño recorrido para llegar hasta una sala que ya es inaccesible por el fuego, o simplemente para que el jugador ordene un NPC retener el fuego mientras el resto evacua la sala hacia la salida.

Otra mejora interesante sería aportar una mayor variedad de NPCs, tanto a nivel de su apariencia visual sino de su comportamiento, dotándolos de una personalidad diferenciada haciéndolos más cobardes, valientes o colaborativos según esta.

Por último, me habría gustado añadir efectos especiales y de sonido para dotar al juego de un mayor realismo e inmersión del jugador.

5.Agradecimientos

Quiero agradecer el apoyo recibido durante estos meses sobre todo a mi novia por aguantarme y sacrificar el tiempo este verano para ayudarme y por enseñarme a redactar una bibliografía en condiciones.

También quiero agradecerles a mi familia el apoyo que me han dado y el haberme preparado las comidas para que pudiera dedicarle mi tiempo íntegro a este proyecto.

Por último, me gustaría agradecer a mis amigos su apoyo como beta testers del juego cuando tuve que llevarme el ordenador durante el viaje de este verano para terminar de redactar esta memoria.

6.Bibliografía

1. Kirana C., Wijaya B., Holil A. Implementation of the Fisher-Yates Shuffle Algorithm in Exam-Problem Randomization on M-Learning Application. Khazanah Informatika [Internet]. 2021 [Consultado 12 agosto 2021]; Vol. 7 No. 2, ISSN: 2477-698X. Disponible en: https://journals.ums.ac.id/index.php/khif/article/view/11761/6657

2. Panca F., Arie H., Sylfania D.Y. Performance Comparison of Linear Congruent Method and Fisher-Yates Shuffle for Data Randomization. Journal of Physics: Conference Series [Internet]. 2019 [Consultado 4 septiembre 2021]. Disponible en: https://iopscience.iop.org/article/10.1088/1742-6596/1196/1/012035/pdf

3. He Z. Research and Application of Path-finding Algorithm Based on Unity 3D. School of Computer Science: Communication University of China [Internet]. [Consultado el 17 agosto 2021]. Disponible en: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7550934

4. Zikky M. Review of A* (A Star) Navigation Mesh Pathfinding as the Alternative of Artificial Intelligent for Ghosts Agent on the Pacman Game. EMITTER InternationL Journal of Engineering Technology [Internet]. 2016 [Consultado 20 agosto 2021]; Vol. 4, No. 1, ISSN: 2443-1168. Disponible en: https://emitter2.pens.ac.id/ojs/index.php/emitter/article/view/117/56

5. Antón J., Zhu X. Estudio y simulación de un vehículo autopilotado en Unity 5 haciendo uso de algoritmos de aprendizaje automático [Internet]. Universidad Computense de Madrid. 2017. Disponible en: https://eprints.ucm.es/id/eprint/50223/1/032.pdf

6. Fernández A. Implementación de situaciones de riesgo en un simulador de accidentes de tráfico mediante motor de videojuegos Unity [Internet]. Universidad Politécnica de Cartagena. 2015. Disponible en: https://repositorio.upct.es/bitstream/handle/10317/5196/pfc6284.pdf?sequence=1&isAllowed=y

7. Sánchez S. Posibilidades de la Realidad Virtual para mejorar la accesibilidad en desarrollos realizados con Unity [Internet]. Escuela Politécnica Superior de la Universidad de Alcalá. 2018. Disponible en: https://ebuah.uah.es/xmlui/bitstream/handle/10017/33862/TFG_Sanchez_Lopez_2018.pdf?sequence=1&isAllowed=y

7.Anexo

7.1 Documento de diseño del videojuego(GDD)

GAME DESIGN DOCUMENT

D.A.N.G.E.R

Adrián Sánchez Lavarias
Pablo Querol Ballester

SECCIÓN I: Visión General Del Proyecto

Nombre del juego

D.A.N.G.E.R - Disaster Prevention

Personal de equipo

A continuación definiremos el personal al completo que forma parte del videojuego Danger, con sus respectivos correos para contactar con ellos.

Desarrolladores:

- Adrián Sánchez Lavarias (email: adsanla@inf.upv.es)
- Pablo Querol Ballester (email: pabqueba@inf.upv.es)

Resumen ejecutivo

Concepto general

Danger es un juego con propósito pedagógico con el que aprender normativas y procedimientos aplicables a la prevención de incendios. Formado por 2 modos de juego, en el primero podrás construir salas y ponerlas a prueba para comprobar si cumplen con la normativa. Mientras que en el segundo podrás ponerte en la piel de las personas que se encuentran en un incendio y guiarlas para que logren escapar antes de que les alcance el fuego.

El Gancho

En Danger podrás desatar tu imaginación creando distintas salas o desarrollar tus propias estrategias para salvar la vida de las personas que han quedado atrapadas.

Género y extensión del juego

Género: Construcción y estrategia en tiempo real (RTS)

Extensión: Dispone de 2 modos de juego.

Estilo visual

3D en 3ª persona.

Motor

Unity y Visual Studio

Núcleo del Gameplay

Mezclará los géneros de construcción, puzzle y estrategia en tiempo real en tercera persona, enfatizando en la toma de decisiones a lo largo del videojuego. Las plataformas principales son PC y dispositivos móviles.
Es un juego pensado para un solo jugador, sin distintos niveles de dificultad. Enfocado en la temática principal de incendios, puede ser rejugado para explorar diferentes situaciones.

Características de juego

Innovaciones del Gameplay

El objetivo es hacer un juego interactivo en 3D con el fin de enseñar la normativa y metodología en caso de incendio. Para ello se han creado 2 modos de juego de los géneros construcción con puzzle y estrategia en tiempo real. Lo que permite desarrollar la creatividad del usuario y colocarlo en una simulación de una situación real.

Características diferenciadoras

El factor diferenciador del juego es la dualidad de modos de juego que desarrollan la creatividad, la capacidad de estrategia y reacción del jugador. Ambos modos de juegos, además, fomentan la rejugabilidad del título.

Alcance de proyecto

Descripción de niveles

1. Creación

En el modo de juego “build” encontramos un nivel con un solo bloque base desde donde el jugador comenzará a crear el conjunto de salas deseado. Estás salas tendrán unos requisitos dependiendo del tamaño total de las mismas que deben cumplirse para finalizar el nivel.

2. Estrategia

Los niveles del modo “escape” se generan de forma procedural, por lo que cada vez que se ejecuta el juego es diferente. Sin embargo, todos comparten que son salas conexas entre sí con forma rectangular pobladas con mobiliario de oficina.

Público objetivo

Danger está dirigido a todo tipo de público (PEGI 7). Esto es debido a que el jugador deberá afrontar tomas de decisiones en un entorno controlado y creativo donde aprender.

Plataformas

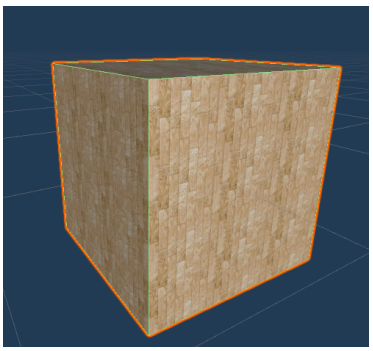
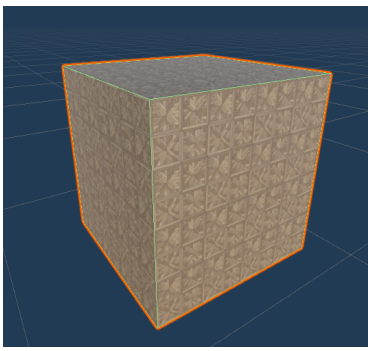
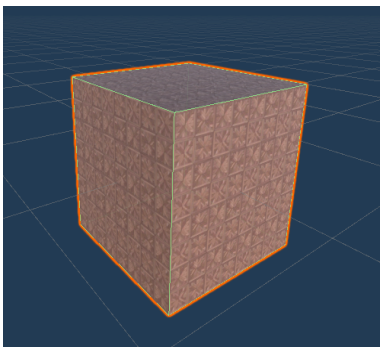
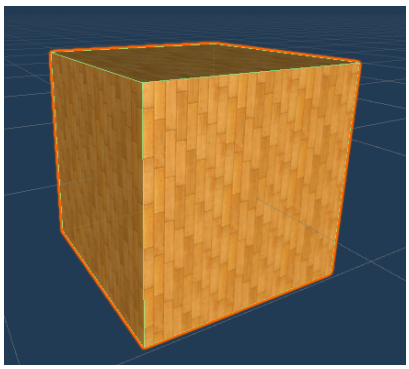
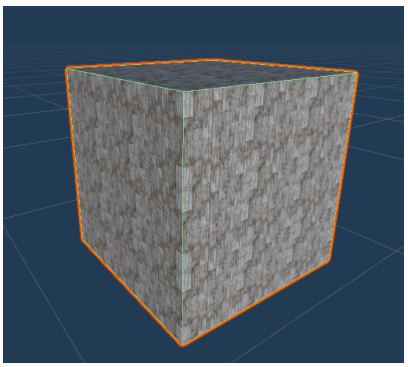
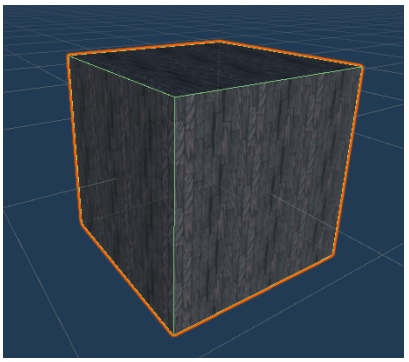
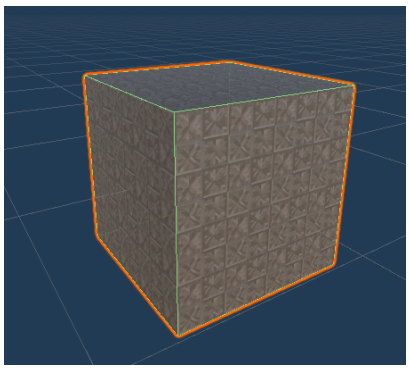
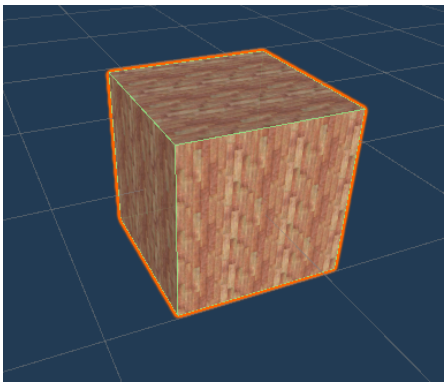
PC

SECCIÓN II: Assets

Modo construcción

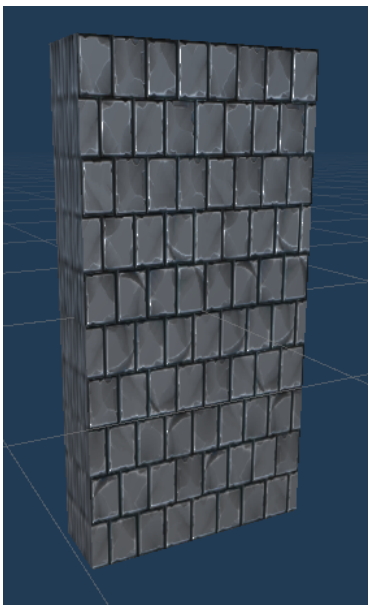
A continuación se detallarán los distintos objectos o assets utilizados para el modo construcción a lo largo del proyecto:

- Bloques:

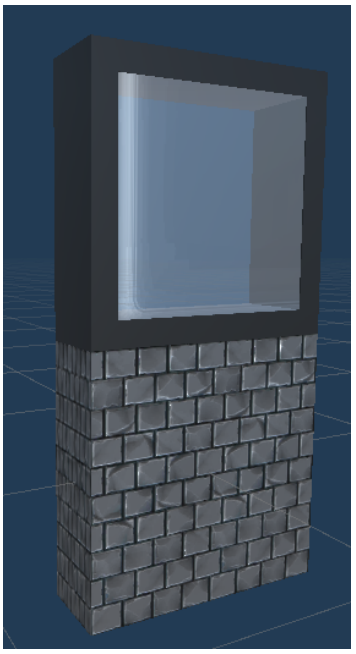









- Puerta:

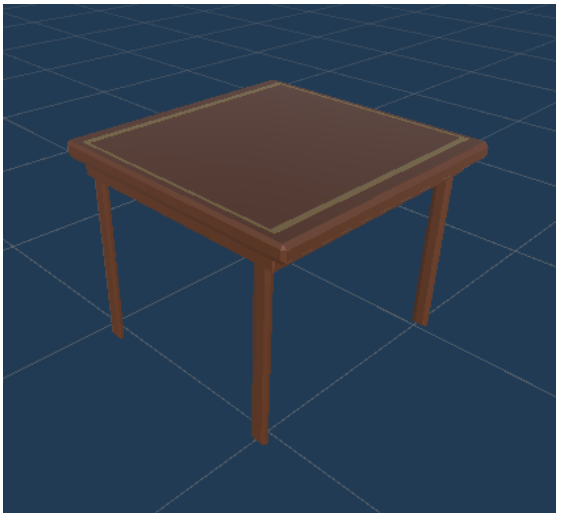
- Muro:



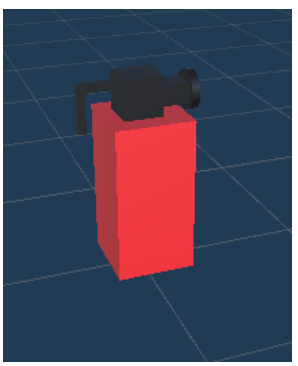
- Ventana:



- Mesa:



- Extintor:

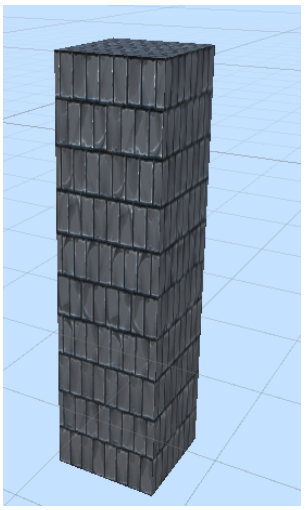


Modo estrategia (escape)

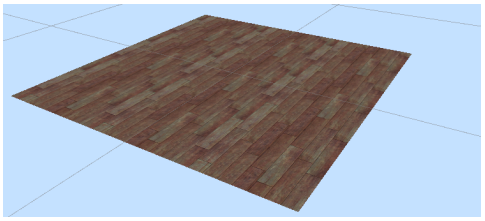
A continuación se detallarán los distintos objetos o assets utilizados para el modo de estrategia (escape) a lo largo del proyecto:

- NPC:

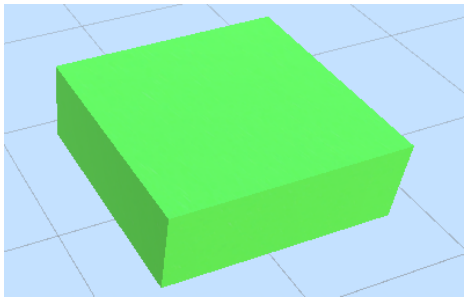
- Pared:



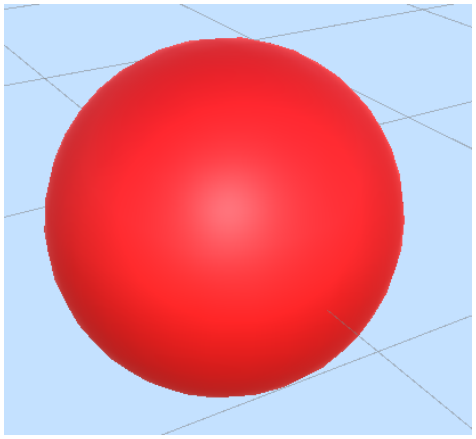
- Suelo:



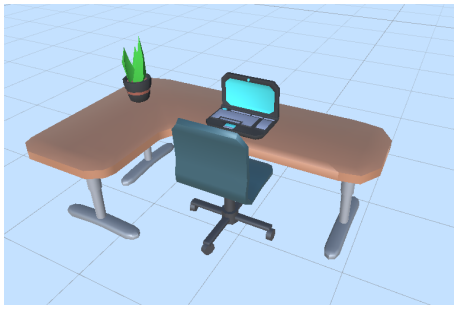
- Salida:

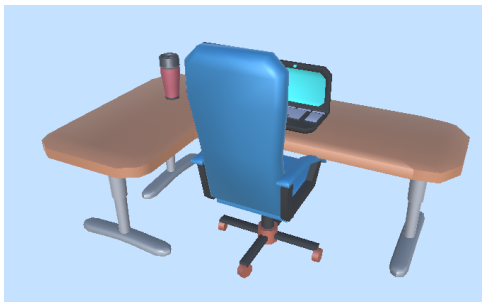


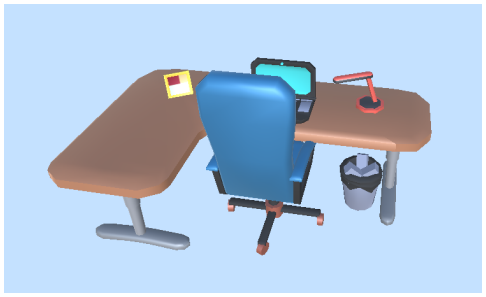
- Localizador jugador:



- Mesas:

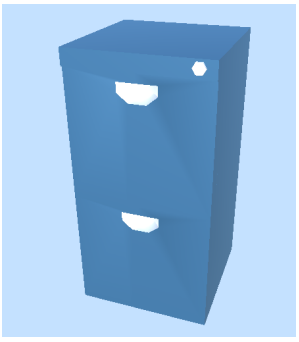




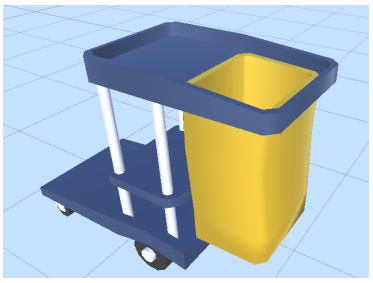


- Papelera:

- Armario:

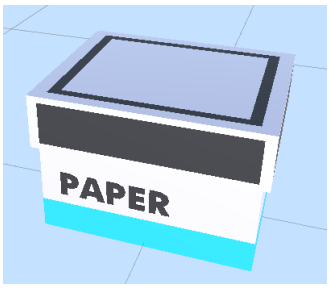


- Carro de limpieza:



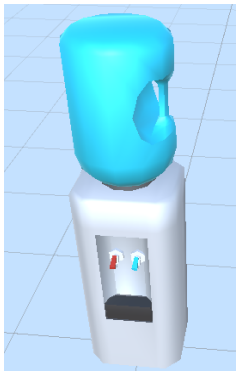
- Extintor:

- Caja de papeles:

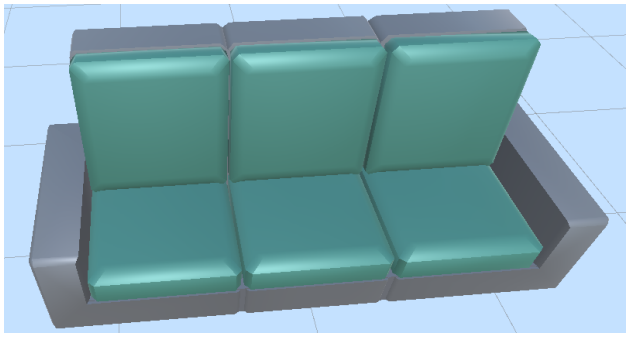


- Planta:

- Máquina de agua:



- Sofá:



SECCIÓN III: Controles

En este apartado se tratará de profundizar en detalle en todo lo referido a los controles que usaremos, para así poder realizar las distintas acciones que estarán definidas dentro del videojuego.

Tenemos la intención de lanzar nuestro videojuego en PC y dispositivos móviles, sin embargo los controles de la versión de móvil no están

implementados todavía.

Órdenes de Ratón/Teclado de PC

A continuación definiremos los controles básicos que manejaremos a lo largo del videojuego. Algunas de las siguientes configuraciones, como son por ejemplo el caso del desplazamiento de la cámara y los personajes.

Teclas por defecto para el modo de juego construcción (Build)

- Desplazamiento de la cámara: Control Izq + Click Izquierdo.
- Rotación de la cámara: Control Izq + Click Derecho.
- Zoom de la cámara: Rueda del ratón.
- Colocar un objeto: Click Derecho.
- Eliminar un objeto: Click Izquierdo.

Teclas por defecto para el modo de juego de estrategia (Escape)

- Desplazamiento de la cámara: teclas WASD
- Seleccionar personaje: Click izquierdo .
- Seleccionar destino:Click izquierdo .
- Aumentar velocidad del movimiento: Shift .
- Abrir/cerrar mapa: tecla M

Menú de Acceso

- Acceso al menú: Escape.

SECCIÓN IV: Interfaz

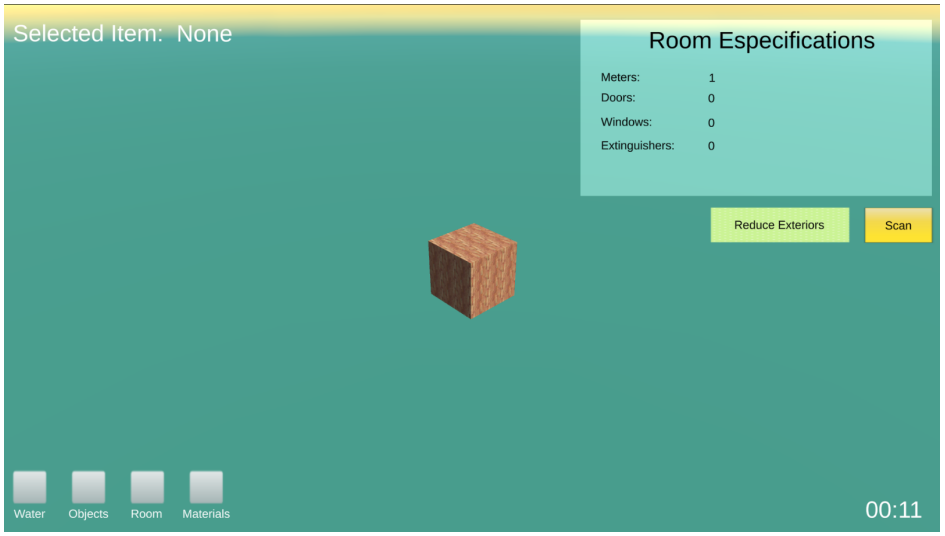
En esta sección concretamos la interfaz del jugador en detalle, especificaremos los distintos menús y cómo moverse entre ellos. Dentro de esta parte encontraremos el menú principal con sus distintas opciones, el HUD y el inventario.

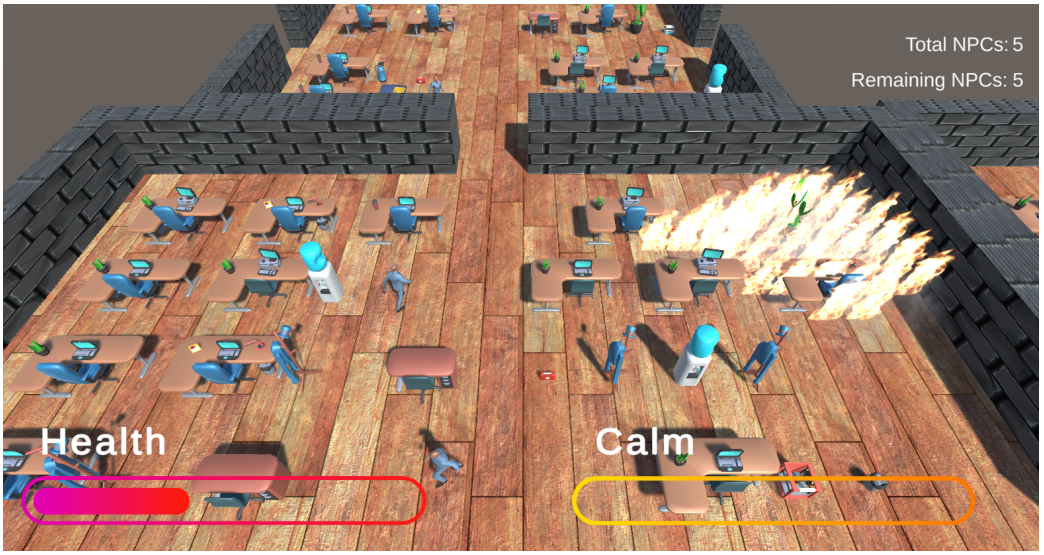
Cámara

Respecto a la cámara tendremos una vista en 3ª persona controlable por el jugador, de manera que el la sitúe en la dirección y ángulo que quiera en cada momento.

Vista estándar

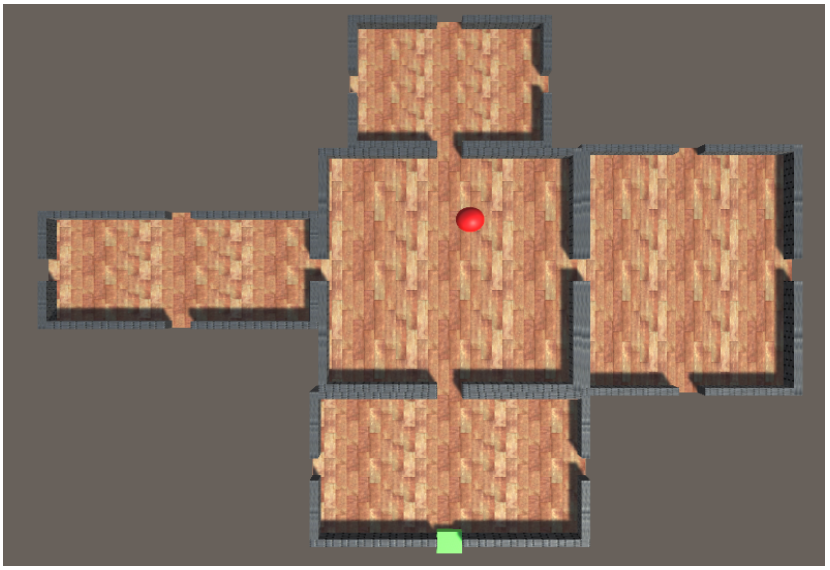
De manera estándar la cámara se situará en una posición concreta.





Vistas Alternativas

En el modo de estrategia (Escape) disponemos de una vista alternativa en forma de mapa, desde esta vista es la única donde pueden verse la posición actual del jugador en el escenario (punto rojo) y la salida del edificio donde tienen que llevar los personajes para ganar (cuadrado verde)



Opciones controlables por el Jugador

Utilizando el ratón el jugador podrá mover la cámara a voluntad.

HUD

Vista del jugador

- Modo de construcción


Red
Black
Red 2
White
Black 2
Grey
Orange
White 2
Water
Objects
Room
Materials

- Modo de estrategia

Información de estado

En la vista del jugador podemos observar los distintos estados e información que le damos al jugador:

- Modo escape
    - Salud

Visible en la esquina inferior izquierda del HUD y administrada mediante una barra de color gradiente entre rosa y rojo. Cuando el fuego te alcanza al personaje el tamaño de la barra disminuye. Si te quedas sin salud pierdes el juego.

    - Estrés

Visible en la esquina inferior izquierda del HUD y administrada mediante una barra de color gradiente entre amarillo y naranja. Cuando el personaje detecta fuego o humo este nivel aumenta. Cuando el personaje lleva un tiempo sin ver elementos peligrosos comienza a disminuir gradualmente. Cuanto más alto es este nivel mayor será la velocidad del personaje y menor será el tiempo de espera del personaje antes de pasar al modo autónomo. También aparecerá un texto sobre la barra que se intercambiará entre: “Calm”, “Stress” y “Pánic” según vaya aumentando o disminuyendo el nivel de estrés.

    - NPCs totales y restantes

Contadores visibles en la esquina superior derecha con el número de NPCs restantes y totales.

- Modo build
    - Selected Item

En la esquina superior izquierda encontraremos este elemento que nos indicará qué objeto concreto tiene el jugador seleccionado en cada momento.

    - Room Especifications

Este elemento se encuentra en la esquina superior derecha, se compone por un panel con la información relativa a todas las salas creadas por el jugador en la escena.

    - Item selectors

Conjunto de botones donde los diferentes objetos se clasifican, al pulsar estos botones abrimos sus páginas correspondientes para poder seleccionar el objeto deseado.

Menús

Menú principal

Menús de opciones

SECCIÓN V: Planificación

Planificación			
Tareas	**Encargado**	**Comienzo**	**Final**
Desarrollo			
Diseño	Pablo y Adrián	01/06/21	06/06/21
Concepto	Pablo y Adrián	01/06/21	03/06/21
Level Mechanics	Pablo y Adrián	03/06/21	06/06/21
Art	Pablo y Adrián	06/06/21	20/08/21
Modo estrategia	Pablo	06/06/21	20/08/21
Modo construcción	Adrián	06/06/21	16/08/21
UI	Adrián	06/06/21	20/08/21
Implementación	Pablo y Adrián	08/06/21	31/08/21
Modo estrategia	Pablo	08/06/21	31/08/21
Modo construcción	Adrián	08/06/21	31/06/21
Audio	Pablo y Adrián	01/08/21	31/08/21
Sound Design	Pablo y Adrián	01/08/21	31/08/21
MVP	Pablo y Adrián	15/08/21	25/08/21

Testing			
Test Plan	Pablo y Adrián	29/08/21	29/09/21
Beta Testing	Pablo y Adrián	29/08/21	03/09/21
Fase de lanzamiento			
Listo para su uso	Pablo y Adrián	04/09/21	04/09/21