The Prop Hunt

Daniel Túnez Bermejo
Ingeniería Informática
Área de videojuegos

Gus Marcos Ballester
Joan Arnedo Moreno

Diciembre de 2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	*The Prop Hunt*
Nombre del autor:	*Daniel Túnez Bermejo*
Nombre del consultor/a:	*Gus Marcos Ballester*
Nombre del PRA:	*Joan Arnedo Moreno*
Fecha de entrega (mm/aaaa):	01/2022
Titulación:	*Grado en Ingeniería Informática*
Área del Trabajo Final:	*Videojuegos*
Idioma del trabajo:	*Español e inglés*
Palabras clave	*Multijugador, casual, humor*

Resumen del Trabajo

El objetivo de este trabajo de final de grado es la programación de un videojuego multijugador 3D con un alto enfoque en el humor. El proyecto abarca desde la creación de una API externa para ofrecer apoyo al videojuego, hasta la programación de este, pasando por el diseño tanto a nivel práctico (mecánicas del juego) como visual.

El desarrollo se ha hecho con Unity 2021 y Visual Studio Code en un Macbook Pro, con una pequeña parte de Blender para el modelado 3D de algunos elementos. La elección de este motor gráfico frente a sus competidores se debe a mi familiaridad con el mismo así como con el lenguaje C#, con el cual llevo trabajando muchos años.

El mayor desafío de este proyecto ha sido el de hacer un videojuego en línea, pues cambia totalmente la forma de no solo programar, sino de entender cómo debe fluir cualquier procedimiento.

Como nota adicional, el resultado obtenido es una versión piloto del juego, para poder comprobar en círculos acotados las garantías de su éxito.

Abstract

The main purpose of this project is to develop an online 3D videogame, focusing on humor. The project consists in a API as an external source of support to the game, the game itself and, of course, designing all the necessary concepts of the videogame (mechanics, visuals, scheduling…).

Development has been made in Unity 2021 with Visual Studio Code with a Macbook Pro. Also, blender has been used to create some basic models. Unity have been chosen over others because I'm already familiar with the programming language C#.

Finally, the biggest challenge of this project was everything related to the online programming, which is different to traditional programming, not only in using the framework, but understanding the flow of a multiplayer game.

As an additional remark, result is a prototype version of the game. Is necessary to check its viability before further time investing.

Índice

Glosario

Término	Definición
Streamer	Persona dedicada a la creación de contenido, generalmente en directo y en plataformas como Youtube o Twitch.
Prop	Objeto. En el juego se denomina de esta forma a aquellos objetos en los que el jugador con rol “promorpher”.
Promorpher	Rol de jugador, el cual puede transformarse en props para esconderse del hunter.
Hunter	Rol de jugador, el cual debe atrapar a los promorphers.
Mod	Un mod, referente a un juego, es una forma de jugar alternativa. A menudo, son pequeñas modificaciones, pero pueden llegar a necesitar descargas externas completas o ser, directamente, un nuevo juego (que se instala en el original).
Standalone	Versión de un juego independiente. Generalmente, se llama a un juego como “standalone” cuando anteriormente usaba el motor de otro juego y dependía de su instalación para el funcionamiento (por ejemplo, DayZ, mod de Arma II, posteriormente con una versión standalone).
API	Del inglés “Application programming interface”, es un servicio online en el cual se pueden enviar y recibir datos desde cualquier lugar.
Matchmaking	Sistema de emparejamiento de jugadores, a menudo basado en la habilidad

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Los videojuegos nacieron principalmente con el único objetivo de divertirnos con ellos. Sin embargo, con el paso del tiempo, no solo ha crecido el número de géneros dentro de los videojuegos, sino también sus objetivos. Por ejemplo, un videojuego puede tener como objetivo principal la diversión, pero ser didáctico (como podría ser el videojuego *Imperium*, enseñando historia, o simuladores de vuelo como Flight Simulator). A lo largo de la historia han aparecido grandes éxitos que han servido de inspiración a muchísimas personas que, a su vez, han hecho que la industria progrese rápidamente (podemos ver, por ejemplo, como *Doom* usa prácticamente todo lo visto en *Wolfenstein 3D*, pero lo lleva a una versión muchísimo más enriquecida).

Sin embargo, no todos los videojuegos exitosos lo han sido desde el principio. En los últimos tiempos, hemos podido observar como numerosos videojuegos, cuyos lanzamientos fueron muy discretos, vieron la fama tiempo después gracias a la figura conocida como “streamer”. En general, estos juegos tenían varios elementos clave: Eran juegos online y de un carácter muy casual. Como un gran ejemplo de ello tenemos *Among Us*; un videojuego muy casual cuya salida pasó muy desapercibida y que, posteriormente y gracias a los streamers, llego a tener éxito.

Este proyecto es la creación de un videojuego que siga la caracterización *de Among* Us, *Fall* Guys y otros juegos similares; casuales, rápidos y con un toque humorístico; The Prop Hunt. El objetivo de The Prop Hunt es llevar una experiencia ya vista en modos de juego de otros videojuegos , a una versión standalone para así facilitar la incorporación de características jugables muy específicas, para favorecer los puntos fuertes de este tipo de juegos, además de reducir los puntos débiles. Call of Duty o Garry’s Mod ya tenían modos de juego de “Prop Hunt”, pero dependían del juego original (limitaciones del motor, reutilización de código…). Hacer un juego independiente permite poder añadir funcionalidades específicas para este modo de juego.

A pesar de que el videojuego es totalmente jugable, el resultado a obtener va más allá del TFG, puesto que se espera poder mejorar el apartado visual y sonoro del videojuego para así poder publicarlo en las tiendas digitales más populares e incluso trasladar la experiencia a otros dispositivos como consolas, teléfonos e incluso en la nube.

1.2 Objetivos del Trabajo

El objetivo del trabajo es la creación de una versión piloto de un videojuego, desde la fase de idealización y planificación hasta la finalización de la misma. El resultado deberá ser una versión jugable con las principales mecánicas del juego disponibles.

1.3 Enfoque y método seguido

Adaptar el modo de juego “Prop Hunt” a una versión simplificada pero más completa, y hacerlo accesible a todo el público.

Para llevar a cabo esta idea, se ha usado el motor Unity con el framework online Mirror. Ya he trabajado anteriormente con Unity y llevo toda mi vida laboral trabajando con el lenguaje C# (que es el que usa el motor), por lo que estoy familiarizado con el motor y el lenguajel. El mayor desafío ha sido aprender no solo Mirror, sino también cómo funciona un juego en línea.

1.4 Planificación del Trabajo

Las partes ajenas a la programación han sido relegadas a un segundo plano (por falta de tiempo principalmente, pero también por estar fuera del ámbito del proyecto). Así pues, el recurso principal es temporal, así como el de algunas personas para obtener feedback y poder probar el videojuego.

La planificación prácticamente no ha sufrido cambios desde el inicio del proyecto, siendo el único cambio la sustitución de algunas configuraciones de la partida, en favor de una mejora audiovisual general (modelos, sonidos, escenario…).

La planificación del trabajo aparece detalladamente en la siguiente página. Allí es posible identificar cada hito (entrega de cada PEC), así como las distintas versiones. Sería muy sencillo, a través del controlador de versiones GitHub, seguir qué se ha implementado exactamente en cada versión.

A medida que ha ido pasando el tiempo, el trabajo ya realizado ha sido marcado en verde, en azul el pendiente, en rojo los hitos y en naranja las sustituciones:

ACTIVIDADES	Sep.		Octubre				Noviembre				Diciembre				E.
	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1
Diseño del juego															
Determinar tecnología															
Planificación															
Prep. de las herramientas de trabajo y GITHUB															
PEC 1 – GDD		B													
0.1.1 - Movimiento personajes															
0.1.2 - Transformación del prop															
0.1.3 - Disparo del cazador															
0.2.1 - UI básica															
0.2.2 - Unirse a una sala online															
0.3 - Elección de bando															
0.4 - Escenario básico															
0.5 - Pruebas online (1 persona)															
PEC 2 – Primer prototipo						B									
0.6.1 - Diseño de elementos interactivos															
0.6.2 - Escenario mejorado															
0.7.1 - Fase pre-ronda															
0.7.2 - Final de la partida (objetivo)															
0.8.1 - Menú principal (jugar, opciones…)															
0.8.2 - Sistema de conexión online por código de sala															
0.8.3 – Modelos, animaciones y sonidos															
~~0.8.3 - ConfiguraHciones básicas de una sala~~ – descartado															
0.8.4 - Pruebas del juego (4-6 personas, varias salas)															
PEC 3 – Versión jugable											B				
0.9 - Ultimar detalles del escenario															
1.0 - Ultimar detalles de la interfaz gráfica															
Preparación memoria															
Preparación presentación															

1.5 Breve sumario de productos obtenidos

Una versión piloto de un videojuego, además de una API de simplificación de direcciones IP (para unirse a una partida online) que también podría ser usada y comercializada.

1.6 Breve descripción de los otros capítulos de la memoria

La estructura del resto de la memoria es la siguiente:

- Idea del juego: Breve explicación del videojuego y todas sus mecánicas.
- Motor del juego: Evaluación de motores, motivo de mi elección y explicación de sus características principales.
- Juego online y Mirror: Explicación del funcionamiento de un juego online, cómo se realiza con Mirror y cómo cambia el desarrollo tradicional.
- Análisis de costes: Descripción de los recursos empleados y de todo lo que será necesario de cara a la publicación.
- El desarrollo: Diario del desarrollo del videojuego, con los pasos seguidos y los problemas que han surgido.
- Juego en vivo: Capturas y explicaciones de los distintos elementos del juego.
- Planes futuros: Todo lo que habrá que hacer para llegar a la publicación del videojuego, así como una enumeración de ideas para implementar
- Conclusión

2. Idea del juego

2.1 Introducción

El videojuego está pensado para ser de un desarrollo a corto plazo pero con posibilidad de un mantenimiento posterior en caso de éxito. No obstante, el tiempo de desarrollo estimado es de aproximadamente un año. Por ello, es posible considerar la versión de este proyecto como una versión piloto, cuyo objetivo es obtener el feedback suficiente para saber si seguir adelante con él o no.

Además, está pensado para todos los públicos, con una especial énfasis en el grupo recién entrado en la adolescencia. Es un juego con una única modalidad online por lo que tiene un alto compontente social que, sumado a su carácter algo más infantil (tanto en gráficos como en jugabilidad), hace que ése sea el público ideal. Sin embargo, hago especial énfasis en que es un juego para todos los públicos.

2.2 Motivaciones

Llevo toda la vida jugando a videojuegos, y de todo tipo. Aunque tengo una predilección por los juegos en primera persona, también soy un amante de todo juego que me haga pasar un buen rato con los amigos. El último gran acierto en ese sentido fue el increíble Among Us; un juego que adapta mecánicas vistas en juegos de mesa (como el Hombres Lobo de Castronegro o similares) para llevarlas al monitor.

Después de probarlo durante varias partidas, me hice la siguiente pregunta: ¿Y si adaptara una experiencia anterior mía de la misma forma? Después de sopesar varias opciones, caí en un modo de un juego llamado Garry's Mod; el Prop Hunt. También seguía las mismas características: Basado sobre todo en el humor, y con una alta carga social. De hecho, ambos juegos tienen la misma premisa: Encontrar al que miente por un lado, y conseguir engañar a todos por el otro.

Sin embargo, mi intención no era hacer una versión standalone sin más, sino dotarla de características nunca vistas antes. Por eso, empecé a pensar qué mecánicas cambiarían el gameplay base, nutriéndolo de más funcionalidades.

2.3 Los dos bandos

Existen dos bandos: Los promorfos (en inglés, promorphs) y los cazadores (hunters). Los objetivos son esconderse y sobrevivir, o encontrar y matar a todos los del otro bando, respectivamente.

Promorphs

Este rol se basa en el camuflaje. Pueden transformarse en casi cualquier objeto que haya en el escenario para pasar desapercibido ante los cazadores.

Tienen mejor visión que su competencia, pero son ligeramente más lentos. Además, su transformación empieza a alterarse tras unos segundos si no se mueven lo suficiente, indicando a los demás jugadores su posición.

Su objetivo es sobrevivir el tiempo suficiente para poder escapar.

Hunters

El cazador tiene más velocidad que los promorfos, pero tiene una visión un poco más limitada. Es difícil que detecte a una presa, pero una vez detectada, sus posibilidades de atraparla son muy altas.

Atrapa a las presas absorbiéndolas con una potente succión de aire. Una vez empiece la atracción, es imposible escapar. Sin embargo, si no consiguen atrapar nada, sus mecanismos empezarán a deteriorarse, perdiendo un poco de salud.

Su objetivo es dar caza a todos los promorfos antes de que acabe el tiempo.

Fantasmas

No es un bando como tal, sino una especie de modo espectador. Ambos bandos tienen su propio fantasma: un ente que es capaz de moverse por el escenario y observar qué sucede. Sin embargo, al estar entre planos, no puede interactuar con nada. Tampoco puede atravesar paredes o puertas.

2.4 Los controles

Movimiento

El movimiento se realiza simplemente con las teclas W (arriba), A (abajo), S (izquierda) y D (derecha).

La rotación del personaje se realiza con el ratón, mirando siempre hacia el mismo.

Interacción

Cuando un objeto tenga un contorno azul, significa que podemos interactuar con él. Suelen ser puertas o botones.

Si tenemos el rol de promorfo, algunos objetos tendrán un contorno rojo. Esto significa que podemos transformarnos en él.

Si tenemos el rol de hunter, el click izquierdo o primario del ratón realizará el ataque. No podemos atacar a objetivos que se estén cubriendo detrás de paredes, puertas u otro tipo de obstáculo.

En el mapa hay una serie de portales de colores. Cada portal comunica con su contraparte del mismo color. Si pasamos por encima, apareceremos en el otro portal. Sin embargo, no podemos usar una misma comunicación de portales durante un tiempo.

2.5 Los sonidos

Los sonidos han sido realizados manualmente con la única habilidad que poporcionan las cuerdas vocales. De esta forma se logra un efecto bastante gracioso sin que desentone demasiado.

Por ahora no hay banda sonora, pero se realizará con la ayuda de una guitarra y algún tipo de software básico como *GarageBand*.

2.6 El aspecto

El aspecto es el apartado menos trabajado, y con un motivo de peso: Para un programador sin conocimiento de modelado, requeriría demasiado tiempo. Consecuentemente, no habría tiempo suficiente en el plazo de entrega de este proyecto.

El aspecto final será low-poly estilizado, para favorecer no solo un aspecto amable y sencillo, sino también la capacidad de funcionar en más dispositivos.

2.7 Las fases del juego

Lobby

En el lobby, el host dispone de un botón para iniciar la partida. Mientras tanto, también podrá observar una cadena de caracteres que puede compartir con sus amigos para que se unan a él en la sala.

Inicio

Tan solo 15 segundos de rigor para que todo el mundo esté preparado para el inicio. Una vez acabada esta cuenta atrás, empieza la fase de preparación.

Preparación

Los promorfos pueden salir, investigar el escenario y poder buscar el sitio ideal. Disponen de 30 segundos para esconderse, pasados los cuales, se inicia la fase de cacería.

Cacería

Los cazadores pueden salir. Esta fase es la principal y, por tanto, la que durará considerablemente más tiempo (300 segundos, concretamente). Cada hunter debe vagar por el escenario para intentar localizar a los promorphs (ligeros cambios en el nivel podrían delatar a un promorfo).

El juego acabará, por ahora, cuando la cacería termine o cuando todos los jugadores de un bando hayan muerto.

3. Elección del motor

Enumeración inicial

Una vez la idea del juego está clara, el siguiente paso es la elección del motor ideal para llevarla a cabo. Los principales contendientes han sido Unity 3D y Unreal Engine.

Cada motor tiene sus puntos fuertes y débiles. De forma resumida, Unity es más personalizable, pero eso hace que sea tenga un inicio más complejo. Por otra parte, en Unreal es muchísimo más sencillo hacer un primer prototipo jugable, pero su complejidad aumenta con el tiempo.

En cuanto a lenguajes, Unity usa el lenguaje C#, mientras que Unreal usa el C++, aunque en mayor medida todo se programa con “blueprints”, un método de programación visual por nodos.

Valoración económica

En Unity, los distintos planes están aquí [1]. El primer plan es gratuito hasta los 100.000$ de ingresos en los últimos 12 meses. Hasta los 200.000$, debemos pagar el plus, que son 400$/año, para después acabar en el pro, que son 1.800$/año.

A parte del precio, también hay diferencias en lo que puede hacer el motor. En general, se añaden herramientas de equipos y cloud. También se añade la posibilidad de cambiar la pantalla de inicio (que, en el plan gratuito, saldrá el logo de Unity).

Unreal Engine, por otro lado, es completamente gratuito hasta el primer millón que ganemos. A partir de ahí, cobrará un 5% de comisión, más información aquí [2].

En este apartado, Unreal Engine es mucho más favorable.

Compatibilidades con las plataformas

Ambos motoroes son totalmente compatibles con las principales plataformas y teléfonos móviles.

En este apartado, no hay ningún ganador.

Conclusión

Ambos motores son perfectamente capaces de llevar el juego a cabo. En este punto del análisis, aún no es posible decidir por cuál de los dos decantarse.

No obstante, como veremos en el siguiente apartado, la inclusión de un modo multijugador complica considerablemente el desarrollo, haciendo que sea más aconsejable escoger un lenguaje que resulte familiar.

Por este motivo, Unity ha sido escogido como motor para el videojuego, con el framework online Mirror.

4. Un juego online

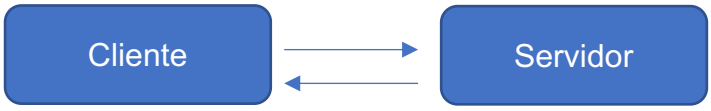
Introducción

Es imprescindible entender cómo cambia el desarrollo el hecho de hacer un videojuego online, respecto a uno tradicional. Por ello, este apartado ha sido clave a la hora de decidir el motor.

Interacción cliente-servidor

Generalmente, existen dos formas de operar: un cliente que a su vez haga de servidor, o bien un servidor dedicado. La primera opción permite reducir costes, pero puede crear desventajas (el cliente que haga de servidor no tendrá latencia alguna). La segunda necesita de un servidor (o servidores) disponibles permanentemente para que los jugadores puedan unirse a ellos.

Vamos a ver una explicación muy simplificada. El esquema de comunicación básico es el siguiente:



Es bastante simple. Sin embargo, lo que realmente sucede es lo siguiente:

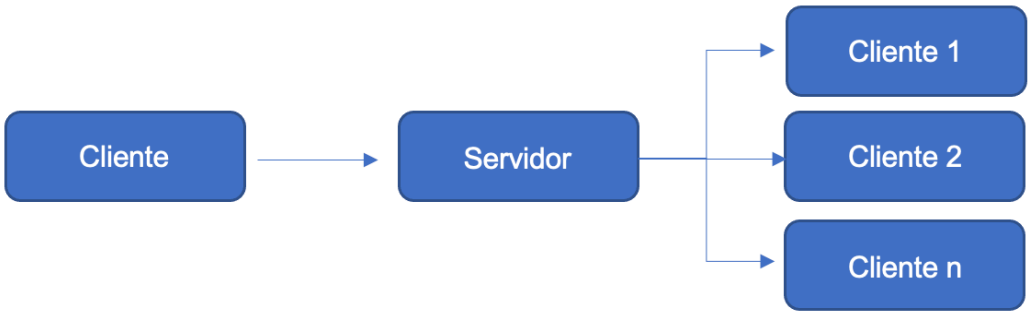


Ilustración 1 - Interacción cliente-servidor

Un cliente, cuando quiera realizar una acción, enviará una solicitud al servidor. Será el servidor quien decida si esa acción puede llevarse a cabo o no y, en caso afirmativo, replicará esa misma petición a todos los clientes.

En la primera opción, la comunicación entre el cliente propietario de la partida y el servidor es instantánea, creando una clara desventaja para los demás clientes. En la segunda opción, el tiempo de respuesta (latencia) depende de la conexión de cada cliente y, a priori, es una opción más justa.

A continuación, imaginemos que un cliente quiere abrir una puerta. En un juego tradicional, bastaría con realizar la acción. En un juego online, el cliente debe enviar una solicitud al servidor. El servidor decidirá si puede realizar esta acción o no. Si no puede, no sucederá nada (aunque el cliente que ha hecho la petición podría controlarlo, para saber el motivo por el cual no puede hacerlo). Si puede, el servidor enviará a todos los clientes esa acción. De esta forma, todos los clientes sabrán que esa puerta debe abrirse. La propia acción de abrir la puerta no ocurre en el servidor (aunque podría, pero no es óptimo), sino en el cliente. El servidor simplemente gestiona la petición, y la reenvía a quien sea necesario.

Imaginemos, por otro lado, que un cliente quiere enviar un mensaje (un mensaje literal, como "¡Hola!") a otro cliente. El cliente A envía el mensaje al servidor, indicando el destinatario. El servidor comprobará que el mensaje pueda ser enviado y que el destinatario exista, y luego lo enviará únicamente a ese cliente.

Existe un tercer caso. En esta ocasión, el cliente quiere moverse hacia arriba (recordemos, con la tecla W). La acción sería enviada al servidor, que decidirá si puede hacerlo o no. Responderá en consecuencia y el personaje se moverá, o no. En este caso, el movimiento se sincroniza automáticamente entre todos los clientes (cliente -> servidor -> clientes). Es la acción de moverse la que requiere de esa programación, en la que los demás clientes no sabrán nada del asunto (simplemente, recibirán la nueva posición una vez el cliente se mueva). Esto nos lo gestiona directamente el framework Mirror (aunque podría programarse de la misma manera que en el primer ejemplo).

No obstante, aunque cualquier acción debería pasar por el servidor, se ha considerado que este juego no necesita tal nivel de seguridad. Por ello, muchas acciones se decidirán directamente en el cliente, reduciendo así el tiempo de desarrollo. En juegos competitivos esto puede dar lugar a los famosos tramposos.

En el apartado 6 "El desarrollo" se explica con más detalle la programación de un juego multijugador.

Concepto de autoridad

En un juego multijugador, un concepto importantísimo es el de autoridad. Como su nombre indica, si un cliente tiene autoridad sobre un objeto, podrá realizar cambios en él. Éste es el caso cuando un jugador quiere mover su personaje.

Es tan importante, porque si un jugador tiene autoridad sobre lo que no debería, podría dar lugar, de nuevo, a los tramposos. Pero no solo eso; en Unity, dentro del nivel, coexisten todos los fragmentos de código de cada personaje. No tendría sentido que, si un cliente quiere mover su

personaje, mueva todos a la vez. Si no se comprueba la autoridad, eso mismo es lo que pasaría (aunque darían muchos errores antes, pero es más fácil de entender conceptualmente). Por ello, a la hora de enviar una acción al servidor (como la de moverse), antes deberemos comprobar: ¿Tiene autoridad sobre este objeto? Si hay 4 jugadores, 3 personajes contestarán “No”, y uno contestará “Sí”.

Frameworks online

Un framework online ayuda a hacer todas estas operaciones y mensajes, haciendo que sea más accesible. Unity, en el momento de empezar este proyecto, había discontinuado su framework de multijugador propio. Por ello, los posibles candidatos eran Mirror y Photon para Unity, y el propio de Unreal Engine. Sin embargo, como el desarrollo de un juego multijugador requiere de un amplio conocimiento del lenguaje y del motor, Unreal Engine ha sido descartado.

En cuanto a Unity, por un lado, Photon es un proyecto propietario y requiere de un pago por usuarios concurrentes (CCU). Esto hace que quede fuera del rango económico de este proyecto.

Por otro lado, Mirror es un framework open source. Podemos instalarlo en cualquier sitio y hacer que opere un cliente como servidor, o gestionar servidores dedicados, instalando una versión servidor del videojuego.

Conclusión: Motor y framework

Aquí llegamos a la conclusión. Hemos visto que un juego online requiere de bastantes cambios en la forma de desarrollar el juego. Por esto mismo, se ha decidido trabajar con Unity. Adentrarse en Unreal para un proyecto de un solo jugador podría haber sido interesante, sin embargo, se ha preferido no arriesgarse e ir a lo que ya resultaba familiar, por si surgía alguna dificultad (que han surgido bastantes, a lo largo de la programación).

5. Análisis de costes

El desarrollo del proyecto es en su totalidad gratuito. No es el caso para la publicación del mismo. A continuación se detalla cada apartado:

Publicación en tiendas digitales

Cada tienda digital tiene un sistema de publicación distinta. Se necesita una licencia, además de constar de unas comisiones.

En PC, tenemos tres tiendas candidatas: Steam, GOG y Epic Games.

En Steam, se debe pagar 100$ para publicar el juego, además de una comisión del 30%. Podemos encontrar toda la información aquí [3].

En GOG, es necesario enviar en un formulario [4] toda la información. Es algo más opaco, y además suelen ser bastante restrictivos.

En Epic Games, aún están en beta, pero podríamos subir el juego con una comisión desde un 12%. Toda la información está en esta página [5].

En App Store (iOS), debemos pagar una tasa anual de 100$ [6]. Existe una comisión por cada venta del 30% [7].

En Android Store, debemos pagar una tasa única[8] de 25$. También existe una comisión similar.

En consolas el proceso es algo más complejo (hay que adquirir una licencia además). El juego saldría en consolas y Stadia dependiendo del éxito, pero en ningún caso es una prioridad.

Servidor para la API

La API de traducción de IP necesita un servidor. Ahora mismo, está alojada en una app de Azure y su coste es de 25€ mensuales, aproximadamente. Por sus características, debería bastar para muchísimas solicitudes. En consecuencia, el factor delimitador es el propio servidor de juego. La referencia de precio es experiencia personal al tenerla publicada, ya que se paga por uso.

Servidor de juego

Aunque el juego puede ser jugado P2P (un cliente creando un servidor dentro del juego), para que otros jugadores puedan unirse, debe abrir el puerto 7777 en el router para TCP/UDP. Este proceso queda fuera del ámbito en el que se ha hecho tanto énfasis de “para todos los públicos”. Por ello, se necesita un servidor dedicado. Las primeras pruebas

consistían en una máquina virtual en Azure. Sin embargo, es un proceso algo complejo para el desarrollo del juego, y necesitará de más análisis en un futuro para saber qué configuración de máquina sería la ideal. El mantenimiento de estos servidores suele ser un gasto importante.

Página web

Cualquier servidor web puede satisfacer las necesidades de una web (con solo front end). No es necesario mucho más. Un servidor web simple con un dominio .es o .com, suele costar aproximadamente 50€ anuales.

Personal

En este proyecto no se ha contemplado ningún gasto relacionado con el personal. Sin embargo, de cara a la publicación, es posible que fuera necesario contratar a algún modelador 3D (como freelance).

Material

El material necesario para el desarrollo de este videojuego es simplemente un ordenador. Para este proyecto, se ha usado principalmente un Macbook Pro M1 con 16Gb de memoria RAM, con un precio oficial de 2.100€.

Adicionalmente, las pruebas del juego se han realizado con un PC Windows propio, que consta de un Ryzen 2600 como procesador, una tarjeta gráfica AMD VEGA 56 y 16Gb de memoria RAM. El coste aproximado es de 1.000€.

Finalmente, los dos monitores, que ascienden a un precio de 300€.

Se disponía de todo este material antes de empezar el proyecto, por lo que el coste final de este apartado es nulo.

6 - El desarrollo

6.1 Planificación

Como cualquier proyecto de IT, la planificación es una parte fundamental para la finalización de forma exitosa. Ya se ha visualizado el cronograma que, básicamente, es el resultado de esta fase. Sin embargo, es necesario entrar en más detalle, de cara a entender por qué se ha hecho de esa forma.

Recapitulando, el juego se basa en la lucha entre dos equipos. Cada equipo tiene un rol completamente distinto, pero también mucha jugabilidad en común: Se mueven igual, interactúan con los elementos dinámicos del escenario de la misma forma…

No obstante, la interacción no formaba parte de la base como tal; es un añadido al modo de juego, que puede jugarse perfectamente sin esta funcionalidad. Los elementos necesarios son: El movimiento, la transformación (promorph), el disparo (hunter) y una interfaz sencilla.

Estructura de carpetas

La estructura de carpetas de Unity puede ser algo caótica; cuando se añade un nuevo asset, se crea en la raíz. En consecuencia, se propone la siguiente estructura, con una carpeta con nombre "_Assets" para colocar allí dentro todo asset importado:

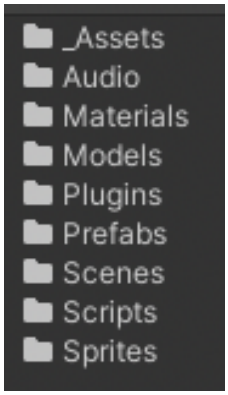


Ilustración 2 - Estructura del proyecto

6.2 Base

El movimiento

El pilar fundamental del juego es el movimiento. En este caso, se realiza con las teclas W, A, S y D, además de con el giro del ratón. Ha sido programado usando el último framework de entrada (input) de Unity. Lo

primero que debe realizarse es la importación (ya que, por ahora, no se instala por defecto). Después,debe ser configurado de la forma siguiente, donde se aprecian todas las acciones posibles:

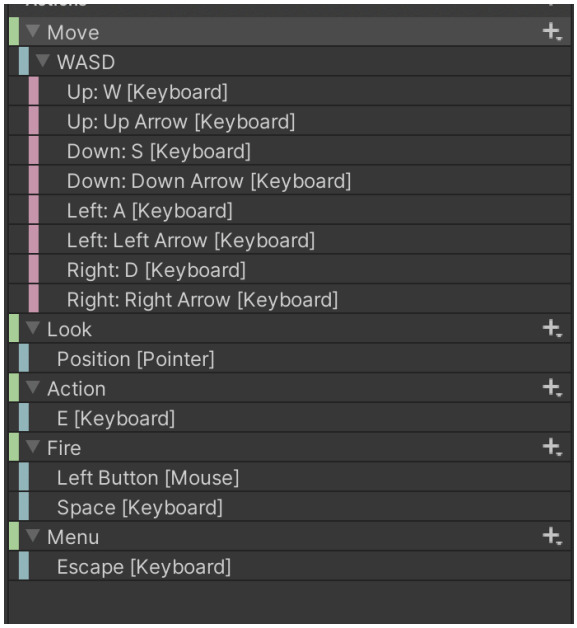


Ilustración 3 - Configuración de entrada

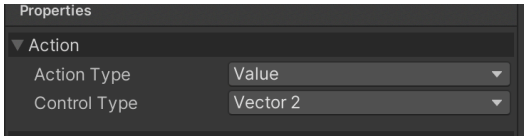


Ilustración 4 - Valor del movimiento

En el caso del movimiento, se indica que es un vector 2D (ilustración 3). Con los botones, solo se marca como “Button” y luego se especifica qué botón lo acciona.

Por la parte del código, se requiere crear un método con un formato concreto:

```
bnpfIc voId WOV6(IubnfACfIou.C9ffp9cKCOuf6Xf cOuf6Xf) => WOV6V6CfOL = cOuf6Xf.R69dV9fU6<V6CfOLS>();
```

Ilustración 5 - Ejemplo de evento de input

En este caso, se asigna un vector que será usado más tarde en los métodos de Update y FixedUpdate (estos métodos se ejecutan en cada frame).

```
[Client]
0 references
private void FixedUpdate()
{
    if (!hasAuthority)
        return;

    HandleMovement();
    HandleRotation();
}
```

Ilustración 6 - FixedUpdate del jugador

En el fragmento de código anterior, existen dos conceptos que lo diferencian de un juego tradicional. El primero es el atributo *Client*, que indica que el código que le sucede solo debe ejecutarse en en lado del cliente. De esta manera, el servidor no debe hacer más cálculos de lo necesario. El siguiente es una condición que comprueba si tiene autoridad. Como explicamos antes, el jugador solo tiene autoridad sobre su personaje, luego únicamente podrá indicar qué movimiento y rotación le pertenecen a él, y no a los personajes de los otros jugadores. En consecuencia, este código del cliente (las líneas relacionadas con el movimiento y rotación) solo será ejecutado en el cliente propietario de ese mismo objeto.

El movimiento y la rotación se realizan exactamente igual que en un juego tradicional (no pasan por el servidor en ningún momento del código). Para su sincronización, se añaden los siguientes componentes al objeto:

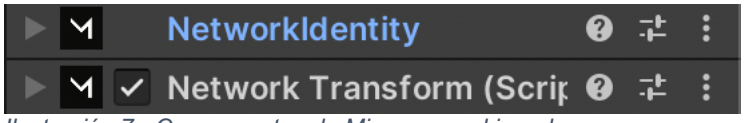


Ilustración 7 - Componentes de Mirror para el jugador

Estos componentes no requieren de ninguna configuración. Con su ayuda, todo el componente Transform (referente a la posición, rotación y escalado) se replica en todos los clientes.

A pesar de que esta forma de hacerlo no es la más segura (si un jugador consiguiera la manera de teletransportarse a través del escenario, su posición se replicaría sin problema), es la única manera de hacerlo viable para la duración establecida. Para arreglar esto, el movimiento debería ejecutarse a través del servidor, complicando considerablemente la programación.

No obstante, hay un detalle importantísimo: Aunque se puede comprobar la autoridad en cada método, es mucho más sencillo hacer lo siguiente: Si no se tiene autoridad sobre el personaje cuando éste se ha creado, entonces no se activa el input. Tampoco la cámara, por ejemplo. De esta

forma, todos los personajes tienen su script principal desactivado, a excepción de aquel que pertenece al jugador. Sin embargo, es importante destacar que el script sí está activo para todos los clientes y servidor.

El ataque del cazador

En este caso, el proceso sí pasa por el servidor:

Ilustración 8 - Flujo de la petición del disparo

Un cliente informa al servidor de que se debe ejecutar el ataque. Entonces, el servidor comunica a todos los demás clientes que ese personaje está ejecutando su ataque. En un juego competitivo o si se controlase de forma más extensa la seguridad e integridad, dentro del servidor existirían una serie de controles para determinar si se debe replicar o no.

A continuación, un cliente informa al servidor. Esta ejecución, virtualmente, no sale de ese script. Lo que hace el servidor es comunicar a todos los clientes, que ese script está ejecutando ese método. Es decir, es el mismo script del mismo personaje, pero cada cliente tiene su copia local.

Finalmente, al atacar, dentro del propio jugador local se detectan los objetos que colisionan. Si uno es un jugador, se manda al servidor una orden para que reciba daño. Un ejemplo con la sincronización visual de este mismo apartado, donde al ejecutar el ataque, el cliente indica al servidor que debe decirle a todos los clientes que se activen las partículas, es el siguiente:

```
public override void Fire(InputAction.CallbackContext context)
{
    CmdSetParticles(true);
}

[Command]
2 references
void CmdSetParticles(bool active)
{
    RpcSetParticles(active);
}

[ClientRpc]
1 reference
void RpcSetParticles(bool active)
{
    attackParticles.SetActive(active);
}
```

Ilustración 9 - Ejemplo Command y ClientRpc

El método “Fire” se ejecuta una vez se capta el input. Este método solo lo ejecuta el personaje local. A continuación, este método realiza una llamada al servidor (el atributo *Command* indica que es un método de servidor). Después, el servidor se comunica con los clientes a través de una llamada Rpc (el atributo ClientRpc indica que es un método de cliente, llamado desde el servidor). Así pues, un cliente le indica al servidor que replique la activación de partículas en todos los clientes.

Recapitulando; el método Fire, solo se ejecuta en el cliente. El método CmdSetParticles se ejecuta únicamente en el servidor. El método RpcSetParticles se ejecuta en todos los clientes, incluído el cliente originario de la llamada.

Importante: Todas las ejecuciones de estos tres métodos se ejecutan sobre el mismo personaje dentro del juego o, mejor dicho, sobre su misma *identidad*. Una identidad es lo que representa a un jugador (cliente) en la red. Es decir, el personaje cuya identidad tiene el identificador 3, tendrá ese mismo identificador en todos los clientes. Esto podría ser importante si programáramos un chat online, donde un jugador se comunica exclusivamente con otro jugador.

Transformación del prop

Este procedimiento es idéntico al ataque del cazador. En el caso anterior, el script local detectaba si había un objeto capaz de recibir daño (un jugador enemigo). En caso afirmativo, se enviaba un mensaje a todos los clientes para que replicasen este mensaje. La similitud está en las partículas; cuando se activan las partículas, se están modificando los elementos dentro del objeto padre (el personaje), haciendo visible un nuevo elemento.

En este caso, se esconde el cuerpo original del promorfo, y se muestra en su lugar una réplica del objeto seleccionado:

```
// enviamos solicitud al servidor para replicarlo el nuevo cuerpo
CmdChangeBody(obj.name, this.playerId);
```

Puede observarse cómo el método, de nuevo, comienza con el prefijo “Cmd”. Esto es un indicativo de que, a priori, se está llamando al servidor:

```
[Command]
1 reference
private void CmdChangeBody(string objId, uint parentId)
{
    //Hacemos el cambio de cuerpo en los demás clientes
    RpcHideBody(parentId);

    //clonamos el objeto
    RpcClone(objId, parentId);
}
```

```
[ClientRpc]
1 reference
private void RpcHideBody(uint id)
{
    Transform target = NetworkClient.spawned[id].transform;
    target.transform.Find("NewBody").gameObject.SetActive(true);
    target.transform.Find("BaseBody").gameObject.SetActive(false);
}
```

Efectivamente, el servidor indica a todos los clientes que esconda el modelo base del cazador, y active el modelo nuevo. A continuación debe asignarse este nuevo modelo. Esta operación se realiza en el método RpcClone, junto con los parámetros objId (nombre del objeto a clonar) y parentId (identificador del objeto del jugador en la red):

```
[ClientRpc]
1 reference
private void RpcClone(string objId, uint parentId)
{
    Transform parent = NetworkClient.spawned[parentId].gameObject.transform.Find("NewBody");

    //primero eliminamos lo que pueda tener antes
    foreach (Transform child in parent)
        GameObject.Destroy(child.gameObject);

    GameObject prop = GameObject.Find(objId);
    GameObject newObject = Instantiate(prop, parent, false);
    newObject.tag = "NewBody";
    newObject.transform.localPosition = new Vector3(0f, prop.transform.position.y - this.transform.position.y, 0f);
    newObject.transform.localRotation = Quaternion.identity;
}
```

En el código anterior puede observarse cómo todo se realiza ya de forma local: se busca el prop objetivo, se crea una nueva instancia (local), no antes de destruir todos los modelos anteriores (en caso de que este jugador ya estuviera transformado).

También es posible ver un detalle: El objeto nuevo creado no forma parte del juego original, ni tiene una identidad de red, luego el objeto que se crea en cada cliente no existe en los demás. Al trabajar con identificadores, no ocasionará problemas, pero es importante destacar que, efectivamente, no se crea un nuevo objeto en la red sincronizado sino que cada cliente crea de forma local un clon.

Para acabar, se ajustan las propiedades de posición y rotación para que todos los clientes tengan las mismas propiedades. El cálculo más destacable aquí es el de la posición Y del nuevo objeto; la altura del objeto y la del jugador son distintas y por ello, debe hacerse una pequeña diferencia para que quede en el nivel original.

Interfaz básica

Al añadir el componente “NetworkManager” de Mirror a la escena, se crea una interfaz para poder unirse a una sala. En su código (Mirror es un proyecto OpenSource), puede observarse cómo funciona y así replicarlo.

Para este proyecto, se añade el canvas al objeto *NetworkManager*, ya que éste está presente en todo momento (al cambiar de una escena a otra,

este script administrador se replicará en la nueva escena). De esta forma, el objeto de la interfaz (el canvas) está siempre disponible. La estructura es la siguiente:

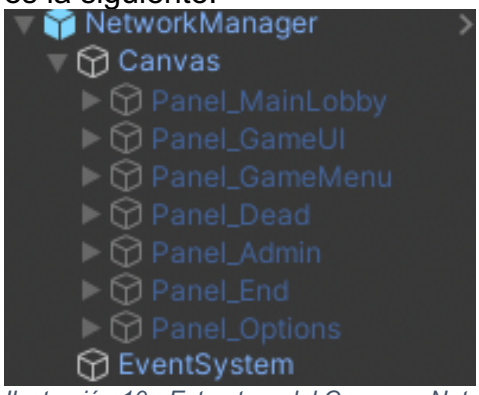


Ilustración 10 - Estructura del Canvas y NetworkManager

Dependiendo de dónde esté el jugador, se activa un panel u otro. Cuando entren al videojuego, se muestra el panel con nombre “MainLobby”:

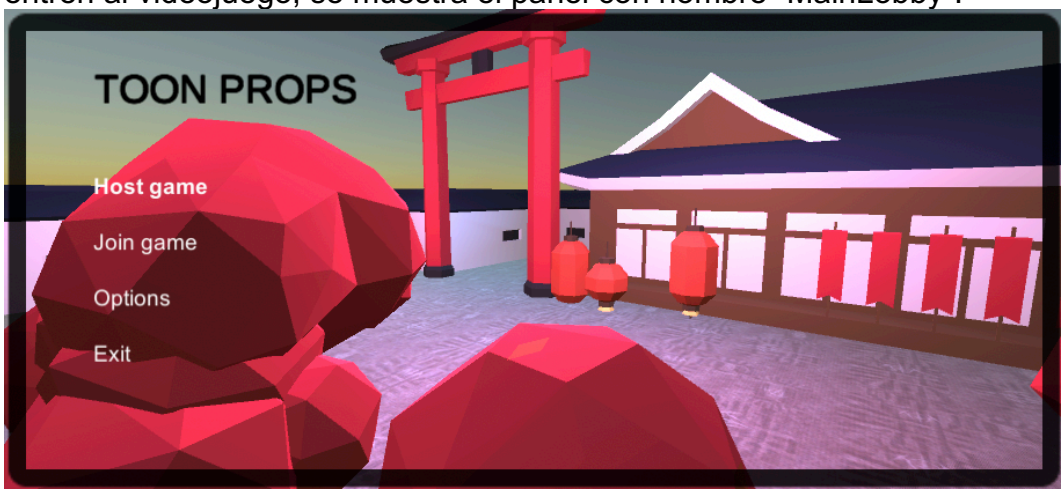


Ilustración 11 - Menú principal

Si el usuario acciona el botón “Host game”, crea una nueva sala. El botón “Join Game” activa el panel “JoinGame”, dentro del panel anterior:

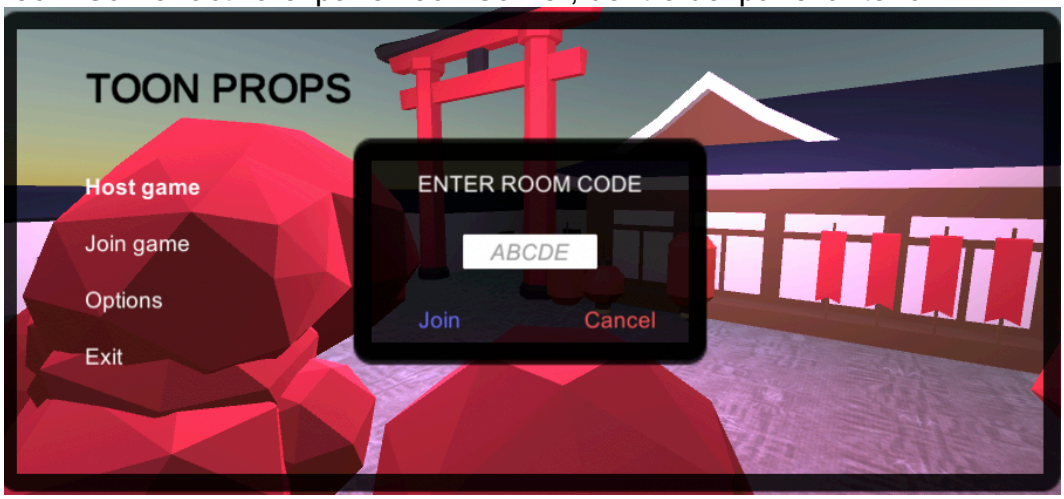


Ilustración 12 - Menú de unión a una sala

El menú de opciones es el siguiente:

Ilustración 13 - Menú de opciones

Este menú es operativo, pudiendo así cambiar los diferentes parámetros del juego.

Finalmente, dentro del juego hay elementos de interfaz muy básicos y orientativos (como botones para iniciar o salir de la partida).

El funcionamiento del código es muy simple: cuando se activa un botón, se esconde un panel y se muestra uno nuevo. A menudo es posible programar estas acciones directamente desde el inspector de Unity:

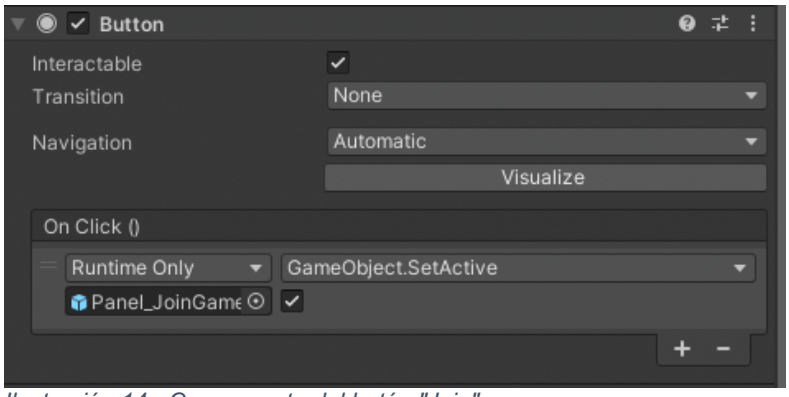


Ilustración 14 - Componente del botón "Join"

Puede observarse cómo en el evento *OnClick* se activará el panel de *JoinGame*.

Elección de bando

La selección de bando es automática (se ejecuta en el servidor) y semialeatoria. El juego está ideado para ser 4 promorfos contra 2 cazadores. Por ello, el algoritmo intenta estabilizar el número. El primer jugador tendrá una probabilidad del 50% de ser cazador. Hasta que no

haya 3 promorfos, la probabilidad de ser un cazador es nula. A partir de 3 (1vs3), tendremos una probabilidad del 50% de ser cazador. El último jugador siempre será del bando que se requiera completar.

Todos estos cálculos deben realizarse cuando un jugador se une a la sala. Mirror proporciona un método en el lado del servidor para ello llamado *OnServerAddPlayer* (si se requiere hacer un cálculo similar en el lado del cliente, se dispone del método *OnClientAddPlayer*). El algoritmo es el siguiente:

```
if (currentHunters < maxHunters)
{
    if (currentHunters == 0 && Random.value < 0.5f) //first player will have a 1/2 prob. to be a hunter
        spawnHunter = true;
    else if (currentHunters > currentProps) //if theres no props, then it is a prop
        spawnHunter = false;
    else
    {
        float probToBeHunter = (currentProps - currentHunters - 1) / 2;
        Debug.Log($"Probabilty to be a hunter: {probToBeHunter}");
        if (Random.value < probToBeHunter)
            spawnHunter = true;
    }
}
```

En función de si se ha calculado que el jugador debe ser un promorfo o un hunter, el código realizará una acción u otra. La principal diferencia es la posición de partida, pero para llevar un registro y así continuar con la ejecución correcta del algoritmo, se realizan operaciones distintas:

```
GameObject player;
//Transform startPos = GetStartPosition();
if (spawnHunter)
{
    Transform startPos = hunterSpawn;
    currentHunters++;
    player = startPos != null
        ? Instantiate(hunterPrefab, startPos.position, startPos.rotation)
        : Instantiate(hunterPrefab);
    listOfConnections.Add(conn.address, Enums.PlayerType.Hunter);
}
else
{
    Transform startPos = propSpawn;
    currentProps++;
    player = startPos != null
        ? Instantiate(propPrefab, startPos.position, startPos.rotation)
        : Instantiate(propPrefab);
    listOfConnections.Add(conn.address, Enums.PlayerType.Prop);
}
currentPlayers++;
```

Finalmente, existe un cálculo para cuando un cliente sale del juego, donde se aprovechan las propiedades del control de número de jugadores:

```
//check what type of player it was
switch (listOfConnections[conn.address])
{
    case Enums.PlayerType.Hunter:
        currentHunters--;
        break;
    case Enums.PlayerType.Prop:
        currentPlayers--;
        break;
}
currentPlayers--;
```

Cuando se desconecta un jugador, se resta uno al número de clientes totales. Adicionalmente, se resta uno también al bando al cual pertenecía.

El escenario

Para poder probar la dinámica del juego, es necesario un mínimo. En este caso, el diseño inicial es similar al actual. Lo más importante a tener en cuenta es que los objetos en los que un promorfo puede transformarse deben estar etiquetados como “prop”.

El diseño inicial del escenario es el siguiente:

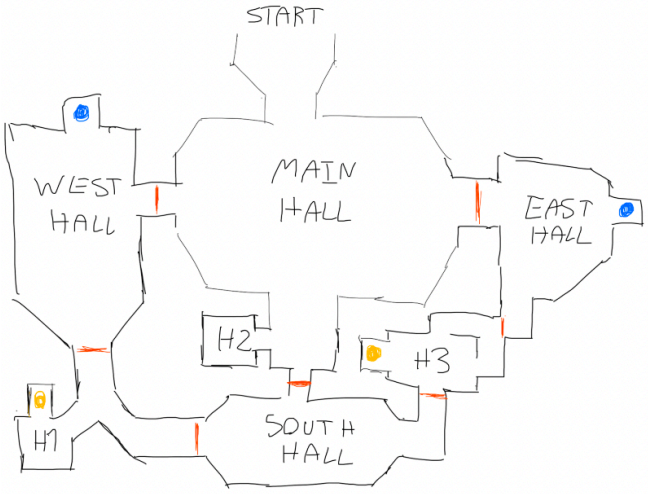


Ilustración 15 - Diseño del escenario inicial

Cada parte del escenario está tematizada:

- El main hall es un laboratorio.
- El west hall es un área militar.
- El south hall, es un área de “Far West”.
- El east hall, es un área asiática.
- Cada habitación tiene una temática también:
    - H1 es un repositorio de armas.
    - H2 es una habitación.
    - H3 es una sala de música / conciertos.

- El área inicial (start) no tiene ninguna temática; simplemente es un punto de partida.

El pensamiento inicial para hacer el escenario es que siempre existan un mínimo de tres salidas, ya que el juego está pensado para un máximo de dos cazadores (consecuentemente, no podrían bloquear todas las salidas de una zona).

No obstante, en las primeras pruebas del videojuego se pudo comprobar que la habitación H2 era muy problemática, puesto que no hay ninguna salida más que la propia entrada. Además, H1 tiene una finalidad muy reducida; únicamente servía para usar el portal. Por ello, era necesaria una fusión de las salas H2 y H1 y dotar a esa zona de más utilidad. El nuevo escenario es el siguiente:

Ilustración 16 - Diseño del escenario final

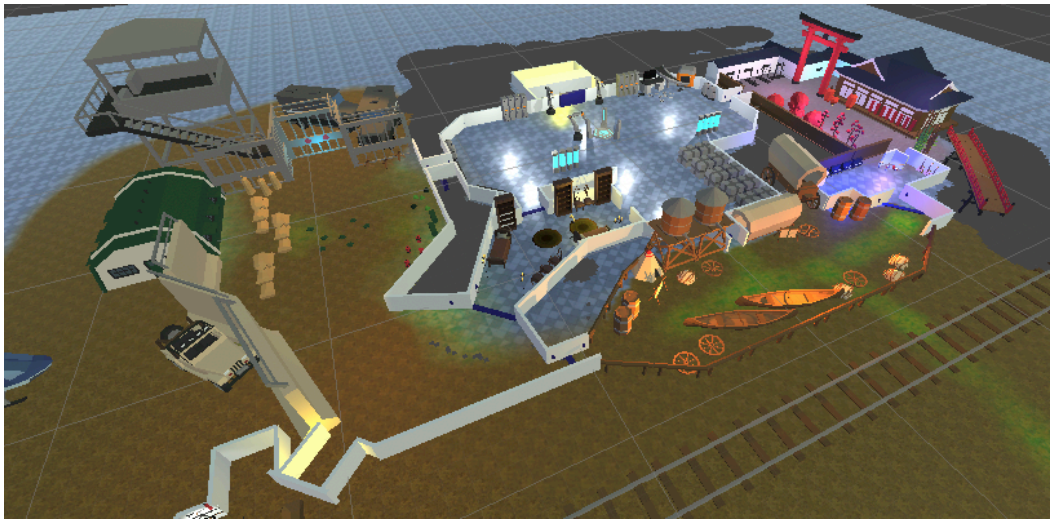


Ilustración 17 - Diseño del escenario final, vista con perspectiva

En conclusión, en esta nueva versión no hay puntos muertos; cada sala y cada hall tiene varias entradas y salidas.

6.3 Versión jugable

La versión final del proyecto requiere de tres apartados: los elementos dinámicos del escenario, las fases de una partida y facilitar la conexión.

Elementos dinámicos

Las puertas (con sus botones) y los portales son los únicos elementos dinámicos de esta versión del juego. Los portales son relativamente sencillos: funcionan de forma local, ya que la posición se sincroniza automáticamente. Las puertas funcionan de una forma muy parecida a las acciones del jugador, salvo que el control está en un objeto completamente independiente.

La sincronización se realiza idénticamente a, por ejemplo, el ataque del cazador:

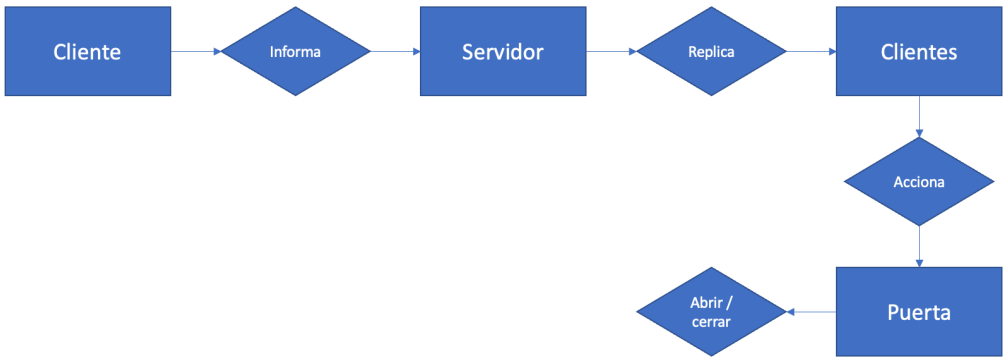


Ilustración 18 - Flujo del uso de una Puerta

Es una ampliación de lo anterior: cada cliente llama a su puerta local. Es decir, el script de la puerta controla detalles como el tiempo de reutilización, o si está abierta o cerrada, pero no tiene absolutamente ningún control relacionado con Mirror.

Aunque también podría controlarse con una sincronización del componente transform, en el desarrollo se ha tenido muy presente que cuantos menos recursos estén en el servidor, mejor. Aunque ahora mismo es un cliente el que hace de servidor, la intención es llevarlo a Azure para que todo el mundo pueda acceder sin necesidad de abrir puertos. Reducir la carga necesaria hará que el coste sea considerablemente menor. Sin

embargo, sí que tiene una identidad de red, y en el siguiente código se observa el motivo:

```
[Command]
1 reference
private void CmdAction(uint netId)
{
    Debug.Log("CmdAction id: " + netId);
    RpcAction(netId);
}
[ClientRpc]
1 reference
private void RpcAction(uint netId)
{
    Debug.Log("RpcAction id: " + netId);
    NetworkClient.spawned[netId].gameObject.GetComponent<Interactive>().Action();
}
```

Ilustración 19 - Código para accionar una Puerta

Una vez llega al cliente la acción (RpcAction), cada cliente busca el ID que se pasa por parámetro para encontrarlo en su escena local. Es decir, el cliente que realiza la acción le dice a todos los demás que el objeto con un identificador X, debe ser activado.

Facilitar la conexión

Eliminar el uso de la dirección IP era imprescindible por dos motivos principalmente: por seguridad, y por comodidad. Para lograrlo, se ha creado una API pública que traduce direcciones IP a cadenas de pocos caracteres y viceversa. De esta forma, es posible la conexión a un servidor mediante una cadena de caracteres como “X5L3”.

Esta API tiene métodos para consultar todas las direcciones y sus traducciones, crear, eliminar o recuperar una traducción y reiniciar por completo la lista. Para facilitar el desarrollo, se ha usado la extensión *Thunder Client [9]* para *Visual Studio Code*, Es posible hacer una biblioteca con las peticiones, y consta de una interfaz limpia y simple:

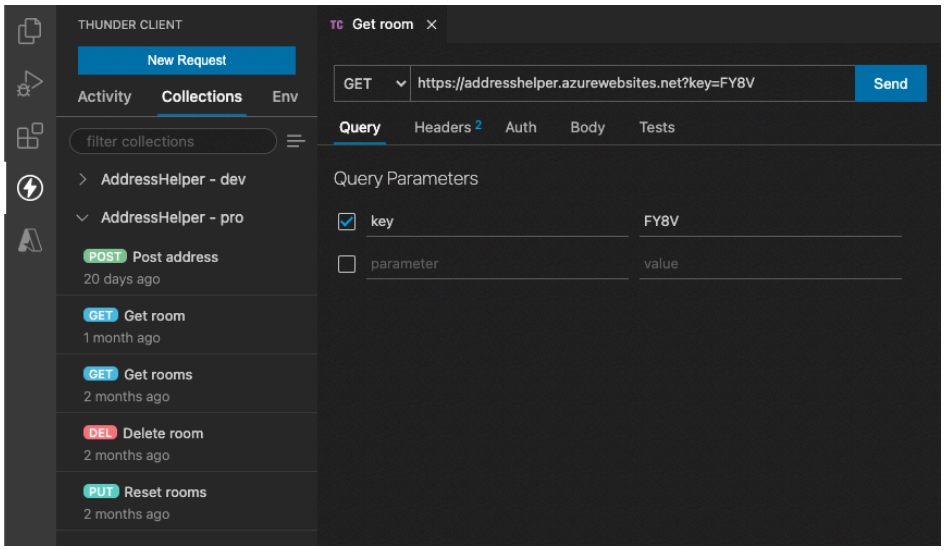


Ilustración 20 - Thunder Client

Finalmente, la API se ejecuta sobre una aplicación en Azure (https://addresshelper.azurewebsites.net). Para la publicación se han usado las extensiones propias de Microsoft:

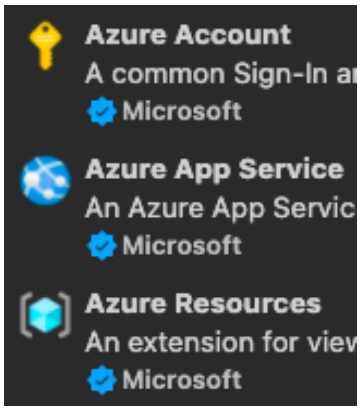


Ilustración 21 - Extensiones Azure

De esta manera, la publicación se vuelve muy sencilla; tan solo es necesario un botón:

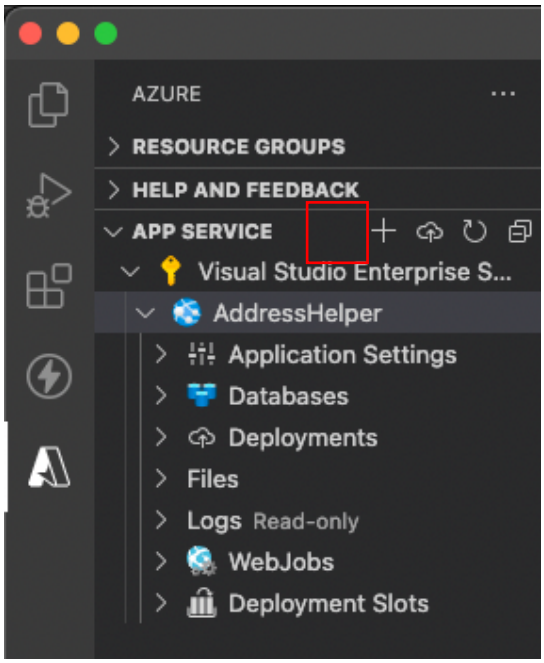


Ilustración 22 - Publicación en Azure

Después de la publicación, la API ya está disponible para cualquiera.

Fases de la partida

Para hacer las fases de la partida, existen dos elementos principales: Una indicación en la interfaz, y una acción física.

La primera parte es una especie de lobby, donde el creador de la partida puede ver el código para unirse y, además, dispone de un botón para empezar la partida. Una vez se aprieta el botón, empieza un procedimiento local en cada cliente:

1. Durante 15 segundos, no sucede nada (es una preparación).
2. Se abren las puertas de los promorfos para que puedan salir por el escenario. Se reinicia el contador a 30.
3. Una vez se acaban los 30 segundos, se abren las puertas de los cazadores. Se reinicia el contador a 300.
4. Una vez se acaba este contador, se acaba la partida.

El administrador es el único capaz de visualizar e interactuar con el botón. Una vez accionado, se disparará todo el proceso, empezando aquí:

```
public void StartGame()
{
    foreach (var player in NetworkServer.connections.Values)
    {
        player.identity.gameObject.GetComponent<PlayerController>().gameStarted = true;
    }
}
```

De esta manera, se indica a cada cliente que el juego ha comenzado. Esta propiedad booleana tiene una peculiaridad; se sincroniza en la red y, además, lanza un evento:

```
1 reference
[HideInInspector, SyncVar(hook = "OnGameStarted")] public bool gameStarted;
```

Una vez se cambia el booleano, se llamará al método *OnGameStarted*:

```
void OnGameStarted(bool oldValue, bool newValue)
{
    menuScript.NewPhase(Enums.RoundType.Starting, networkManager.initialTime);
}
```

A partir de aquí, los jugadores ya no deben realizar nada; el proceso seguirá su marcha. Este nuevo método simplemente actualiza el contador y el valor del enumerable del tipo de fase del script del menú:

```
4 references
public void NewPhase(Enums.RoundType roundType, float time)
{
    this.currentRoundType = roundType;
    this.countdown = time;
}
```

Finalmente, el propio método *Update* del menú se encargará de controlar el tiempo y seguir llamando este método cuando sea necesario. Por ejemplo, el paso de la fase de preparación (Starting) a la del inicio de la partida (izquierda), y del cambio en la interfaz (derecha) serían los siguientes:

```
    if (countdown <= 0)
    {
        switch (currentRoundType)
        {
            case Enums.RoundType.Starting:
                playerController.StartPreround();
                break;
```

```
if (countdown > 0)
{
    string text = "";
    switch (currentRoundType)
    {
        case Enums.RoundType.Starting:
            text = "Starting in... " + (int)countdown;
            break;
```

6.4 Extras

Además de todo lo relacionado con la programación, hay dos apartados con mucha relevancia: El visual, y el sonoro. Aunque requiere de muchísimo trabajo (y fuera del ámbito de este proyecto), era conveniente hacer un modelo básico para el cazador (parecido a un robot-coche) y uno para el promorfo (parecido a un aspirador inteligente):

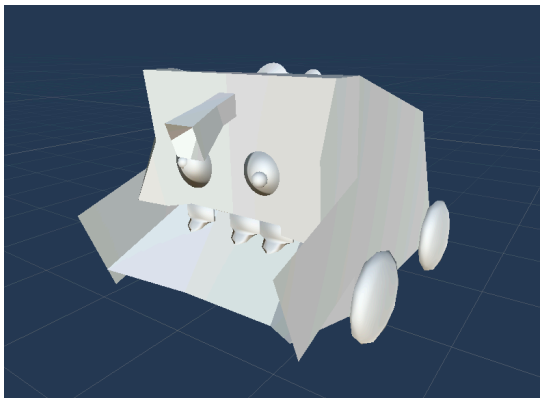


Ilustración 23 - Modelo del cazador

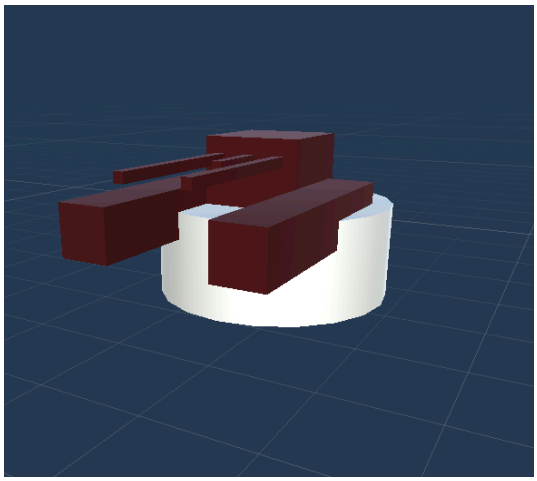


Ilustración 24 - Modelo del promorfo

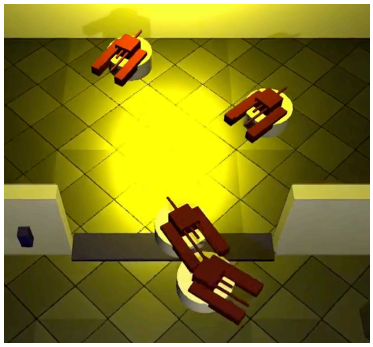
También se han diseñado los efectos de sonido para la transformación, la activación de puertas, el indicador de movimiento y el ataque del cazador.

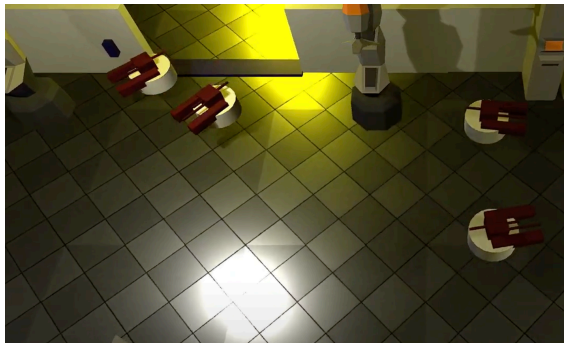
6. Juego en vivo

A continuación, se detalla con capturas de pantalla cada parte importante del videojuego. Por el carácter intrínseco de un videojuego online, siempre será más recomendado ver el juego en funcionamiento en tiempo real. En los anexos están disponibles los vínculos a varias de las partidas de prueba (que se corresponden con la versión 0.8.4).

6.1 Sincronización de movimiento

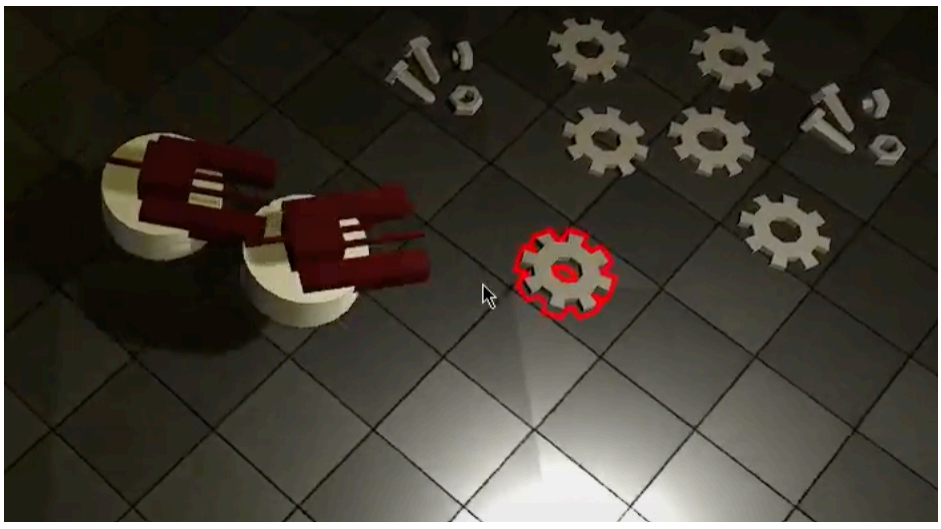
Al inicio de la partida, los promorfos pueden salir y camuflarse. Estas dos capturas marcan esta secuencia:



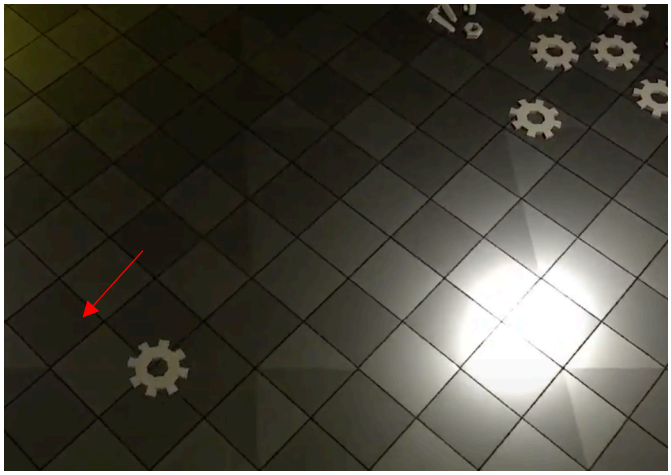


6.2 Transformación del promorfo

La primera parte, es una selección del objeto en el que transformarse:

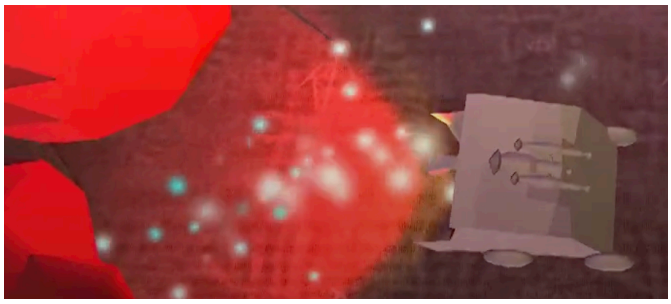


Una vez existe un objeto marcado con contorno rojo, podemos transformarnos:



6.3 Ataque del cazador

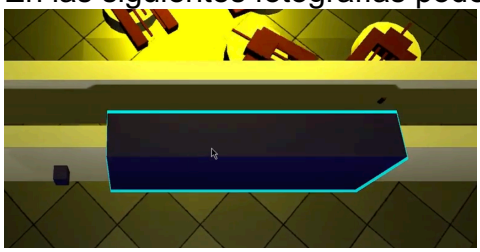
Cuando el cazador acciona el ataque, empieza un proceso de “succión” de partículas. Todo promorfo que esté al alcance es eliminado:

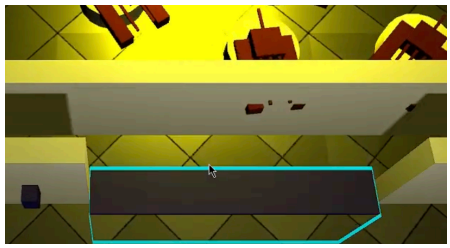


6.4 Puertas

De la misma forma que se marca un objeto para transformarse, una puerta (o cualquier objeto dinámico, en el futuro) tiene un contorno azul. Apretando la tecla de acción, cambiaremos el estado de la puerta de abierta a cerrada, o viceversa.

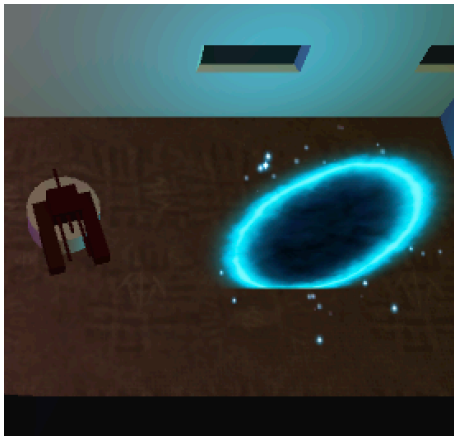
En las siguientes fotografías podemos ver estos dos estados:

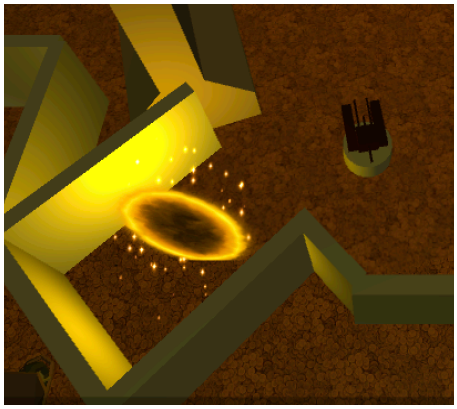




6.5 Portales

Los portales comunican dos puntos del mapa. En el escenario actual, existen dos colores de portales enlazados entre sí; el azul y el amarillo.

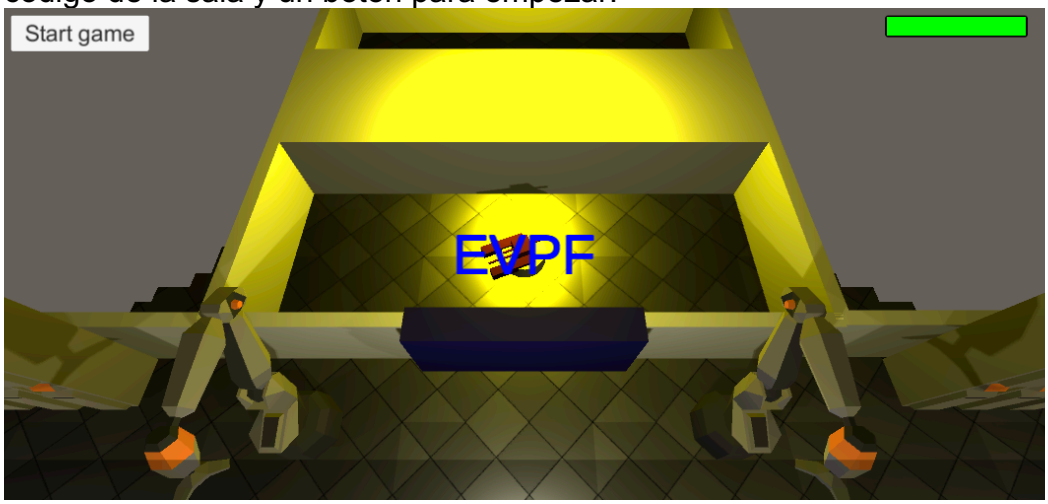




Los portales tienen un tiempo de reutilización por persona, por lo que un mismo jugador no podría entrar varias veces sucesivamente. Sin embargo, otros jugadores pueden entrar simultáneamente o justo después (por ejemplo, en una persecución).

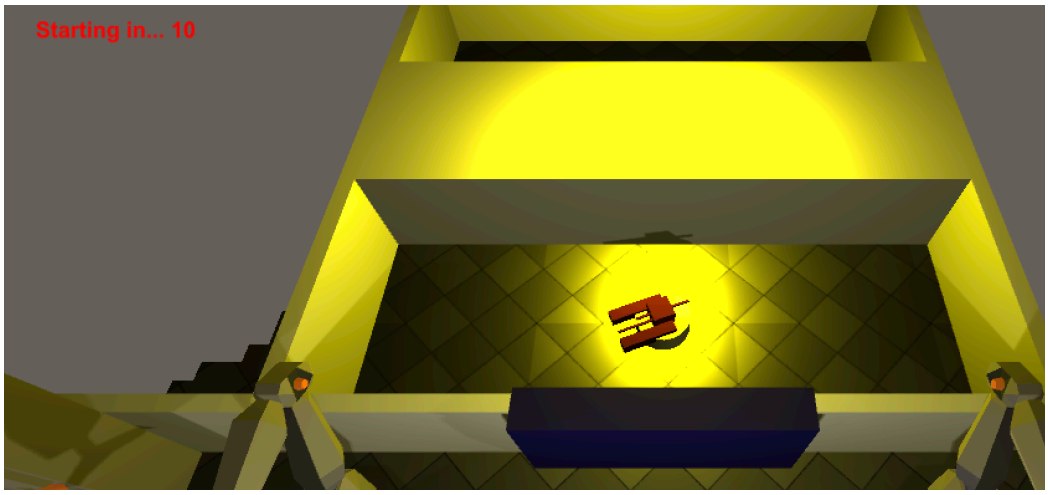
6.6 Fases de la partida

En primer lugar, al crear una nueva partida el administrador puede ver el código de la sala y un botón para empezar:



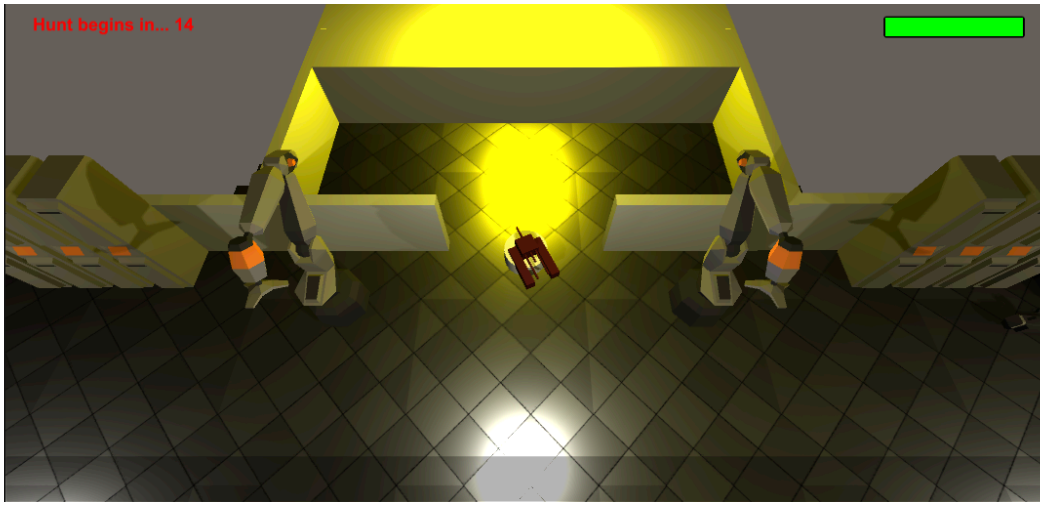
A partir de aquí, se inician las fases como tal.

Iniciando



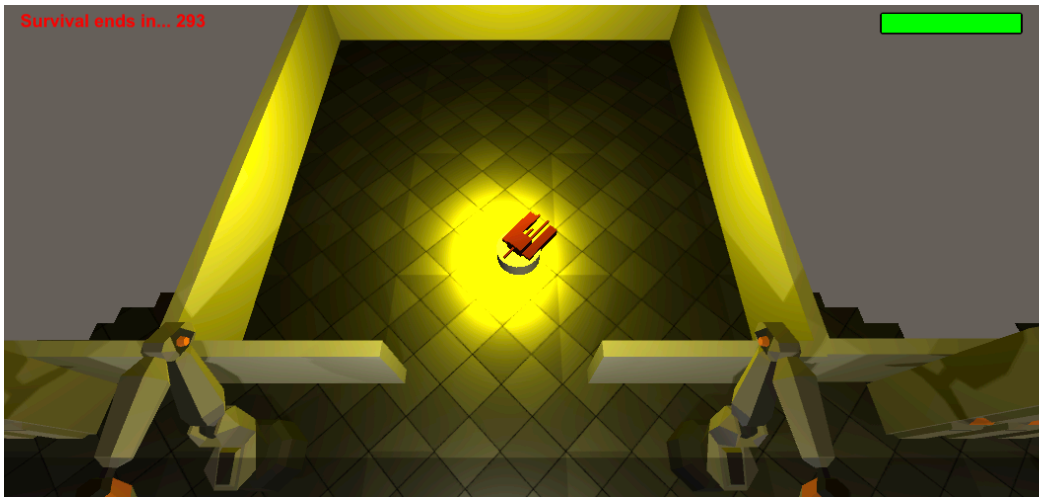
Se está iniciando la partida. Los jugadores disponen de 15 segundos para prepararse.

Preparación



La puerta de los promorfos se abre, y pueden salir. Durante 30 segundos, deben encontrar el mejor camuflaje.

Cacería



Los cazadores ya pueden salir, y empieza la fase principal del juego.

6.7 Modo fantasma

Con el modo fantasma, se le da un motivo al jugador para poder seguir jugando. Las reglas, por ahora, deben hacerla los propios jugadores, indicando así si los fantasmas pueden participar activamente en el juego o no (por ejemplo, dando indicaciones a su compañero).

Pueden moverse libremente, pero no pueden atravesar objetos físicos ni interactuar con ningún elemento dinámico (incluyendo portales):

6.8 Fin del juego

Al acabar la cuenta atrás, la partida acaba, dejando únicamente la opción de salir y volver al menú. En el siguiente apartado se detallan algunas mejoras y alternativas.

7. Planes futuros

El videojuego presentado en este proyecto es una versión prototipo. Como la intención es publicarlo, hay una serie de tareas a realizar antes de tener una versión “gold”.

7.1 Interfaz gráfica

La interfaz debe ser modificada drásticamente, sobre todo la que forma parte de la partida (gestión de rondas, tiempo de reutilización…). Es preferible indicar los tiempos de reutilización como “hechizos disponibles”, similar a otros juegos de clase MOBA o MMORPG pero, lógicamente, de una forma muy simplificada:

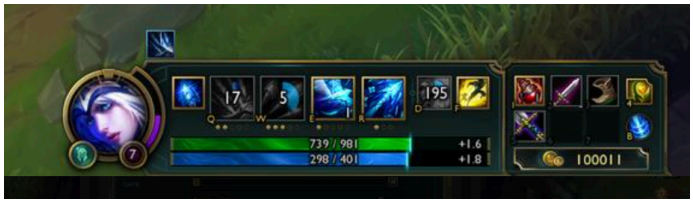


Ilustración 25 - Interfaz de League of Legends

Ilustración 26 - Interfaz móvil de Among Us

El menú principal cumple con su objetivo, de manera que los posibles cambios a realizar quedan relegados a mejoras y efectos de visualización. Adicionalmente, se añadiría un objeto moviéndose y un hunter persiguiéndolo de vez en cuando, haciendo así del menú un elemento muy vivo y representador del juego. Este menú está inspirado en el juego Half Life 2, donde el propio menú era una escena dentro del juego:

Ilustración 27 - Menú de Half Life 2

No obstante, el menú de opciones es mejorable. Consecuentemente, para facilitar su uso, es conveniente quitar el fondo blanco y hacer un diseño similar al visto en la ilustración 12 (menú de unión a sala).

Finalmente, se creará una pantalla final de juego con una tabla de puntuaciones, donde se podrán ver los detalles de cada jugador. En esta misma pantalla será posible volver a jugar con las mismas personas, o salir al menú principal.

7.2 Servidor

En un juego online pensado para todas las plataformas, la idea de que un jugador deba abrir puertos para otras personas puedan conectarse es totalmente incompatible.

En primer lugar, se requiere de unos servidores disponibles todas las horas del día para que los clientes puedan conectarse. Microsoft Azure y Amazon Web Services son las principales opciones.

En segundo lugar, se requiere una API intermedia que sirva de balanceo de carga, gestionando así en qué servidor se creará una nueva sala.

Para acabar, se requiere de una programación adicional en Unity para hacerlo compatible con este nuevo sistema. Adicionalmente, se debe tener en cuenta la opción de añadir un *matchmaking* para poder jugar una partida rápida.

7.3 Página web

Se requiere de una página web básica que sirva como explicación del videojuego, así como redirigir a las distintas formas de obtenerlo.

Esta página web se diseñará puramente con HTML y CSS, usando el lenguaje Javascript en caso de que sea necesario. No se prevé la

necesidad de programación adicional, como podrían ser la gestión de usuarios o foros.

Para acabar, existirá la opción de suscribirse a una newsletter. En principio, se usará un servicio de terceros como MailChimp.

7.4 Traducciones

El videojuego actualmente está en inglés. No obstante, se traducirá el juego a los idiomas más populares (español, catalán, alemán, francés…). No se descarta añadir idiomas adicionales según se requiera (si la población del juego de un lenguaje en concreto crece).

7.5 Ideas del juego no implementadas

Existen una serie de ideas comentadas desde el principio del proyecto que siempre figuraron en una lista de "posibles en un futuro". Otras ideas han sido creadas gracias a las múltiples sesiones de pruebas realizadas.

Sistema de habilidades

Se creará un *pool* de habilidades para cada rol (cazador o promorfo). Al empezar la partida, se escogerá una entre tres habilidades escogidas aleatoriamente.

Estas habilidades tendrán un tiempo de reutilización propio y harán que el juego sea mucho más dinámico. Entre otras, podrán encontrarse habilidades de invisibilidad, velocidad, trampas, portales…

Este sistema es el reemplazo al sistema de héroes. De esta forma, el juego permanece siendo muy simple, y añadir variabilidad se volverá muy sencillo (no hace falta hacer modelos nuevos, por ejemplo).

El sistema sería muy parecido a la elección de habilidades definitivas del modo de juego "Ultimate Spellbook" de League of Legends:

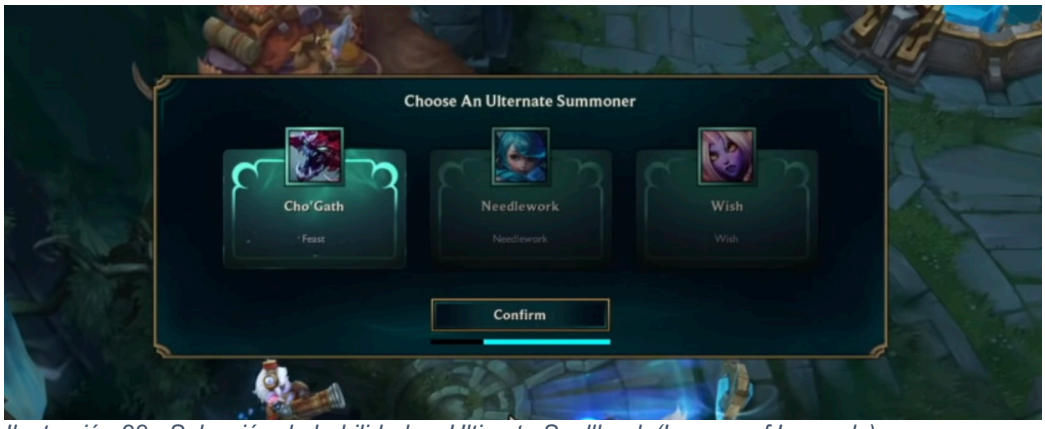


Ilustración 28 - Selección de habilidades, Ultimate Spellbook (League of Legends)

Sistema de movimiento frecuente

Actualmente, el juego dispone de un mecanismo para que un promorfo no pueda estar quieto toda la partida. Sin embargo, es un sistema algo forzado que será reemplazado por un sistema de misiones y eventos.

En este nuevo sistema, un jugador deberá moverse por el mapa para poder lograr la misión y, de esta manera, ganar la partida.

Cada partida tendría un evento distinto; recoger paquetes, operar máquinas… Todas las opciones se basarán en lo mismo: llenar un indicador para poder escapar.

Este sistema no reemplaza el sistema de fases de rondas, sino que lo alimenta; no solo habrá una cuenta atrás, sino que los promorfos deberán cumplir un objetivo (y los cazadores, evitarlo). Es probable que el mapa deba ser ampliado para ello.

Modos de juego adicional: Prop Nightmare

Siguiendo la estela del modo prop hunt, se contemplará la idea de alternar el modo de juego.

El primer modo adicional será el “Prop Nightmare”, donde se vuelven las tornas; los promorfos darán caza a los cazadores. El mapa se volverá oscuro, la distribución de equipos cambiará ligeramente (ya no será un 2 vs 4, sino, por ejemplo, un 3 vs 6).

Para estabilizar los bandos, los promorfos seguirán siendo vulnerables. Las habilidades nativas tampoco cambian: se pueden seguir transformando en objetos. Sin embargo, se crearán habilidades específicas para este rol, que les permitan matar a los cazadores, ya sea por medio de trampas (como crear un clon en movimiento que explote al morir) o por daño directo.

El objetivo del mapa se invierte: los cazadores son los que ahora deben sobrevivir durante un tiempo. Como el mapa está oscuro, su linterna les será de vital importancia. Los promorfos ven perfectamente en la oscuridad.

En conclusión, añadir un modo de juego como este supone unos cambios mínimos, pero podrían alargar la vida útil del videojuego, así como mejorar la diversión de los jugadores.

Nutrir la fase de preparación

Uno de los problemas actuales del juego es que una persona puede memorizar el mapa a lo largo de las partidas. Para esto, se plantean las siguientes características de la fase de preparación:

- Se amplia el tiempo considerablemente (de 30 segundos a, como mínimo 60)
- Se dotan de habilidades a todos los promorfos. Entre otras habilidades, tendrían la opción de mover, rotar o clonar objetos del escenario.

No ha sido posible implementar estas funcionalidades porque se necesita añadir una distracción para los cazadores (que estarán esperando todo este tiempo). En el siguiente punto se describe una opción.

Fase de huida

Al acabar el tiempo y/o objetivo, se abrirán unas puertas que estarán colocadas alrededor del escenario (mínimo 3 puertas, para que los cazadores no puedan bloquearlas). Los promorfos deberán conseguir llegar a un punto exterior del escenario para poder finalizador con éxito la partida.

Adicionalmente, el cazador podrá preparar unas trampas o señuelos en la fase de preparación. Consecuentemente, las salidas estarán comunicadas por unos pasillos exteriores, rodeando el mapa.

8. Conclusiones

8.1 Lecciones aprendidas

A lo largo del desarrollo del proyecto, no todo lo aprendido tiene relación con la propia programación. Se han adquirido habilidades de planificación y valoración, que pueden ser decisivas para el éxito de un proyecto. Sin una correcta planificación y valoración de las herramientas, el proyecto podría sufrir demasiados cambios durante su desarrollo, en ocasiones dificultando su continuidad. Esto ha afectado a numerosas empresas internacionales, publicando juegos que la comunidad de jugadores ha considerado como no acabados, o simplemente dando la impresión de engaño al cliente (publicitando unas características que no se corresponden con el producto final).

Adicionalmente, se ha partido desde un conocimiento prácticamente nulo cuanto a programación de videojuegos online. Se considera que, de ahora en adelante, el desarrollo de nuevos proyectos en esta modalidad o la continuación de este, tendrán una repercusión mucho más llevadera, así como unos tiempos de desarrollo más cometidos.

Por último, también se ha mejorado considerablemente la habilidad de poder unos informes continuados y, en consecuencia, redactar toda la información necesaria en caso de tener que llevar el proyecto a terceras personas.

8.2 Logro de los objetivos y reflexiones

Considerando la planificación inicial y el resultado obtenido, el grado de satisfacción del proyecto es muy elevado. Hablando de carga de trabajo, se ha cumplido con más de un 90% de los objetivos propuestos. Solo ha habido un cambio necesario en la planificación inicial, y se ha propuesto una alternativa que, si bien no cumple con el mismo objetivo, se ofrece otra funcionalidad igualmente atractiva para el consumidor final.

Concretamente, se eliminó del proyecto la posibilidad de configurar una partida por parte del jugador (por ejemplo, número de jugadores y tiempos). Esta funcionalidad habría requerido de bastante tiempo, y no era absolutamente necesaria; el juego está configurado por defecto con las opciones que se consideran óptimas para su disfrute. En su lugar, se ofrece una mejora visual de los modelos de los dos personajes básicos, así como la incorporación de sonidos. De esta manera, el juego parece mucho más cercano a una versión final. Este es mucho más sustancial y con menor coste de tiempo, por lo que se considera que ha sido un cambio no solamente necesario, sino también acertado.

Para acabar, no se está conforme con el resultado de todos los apartados audiovisuales (algunos dentro del juego, pero, sobre todo, externos). Se considera que el tráiler es muy mejorable. Para defender este punto, llegados a una versión comerciable del producto, se contrataría a un experto para llegar al resultado adecuado. Se ha llegado a la conclusión de que no era el foco principal de este proyecto.

8.3 Seguimiento de la planificación y metodología

Se considera que el seguimiento ha sido adecuado y, salvando la excepción mencionada anteriormente, la planificación del proyecto no ha sufrido cambios importantes.

Es importante mencionar que, aunque todas las funcionalidades propuestas para cada hito de seguimiento del proyecto han llegado a tiempo, el orden cronológico de las mismas ha cambiado ligeramente. Esto se hizo para facilitar la programación y no interrumpir dos funcionalidades a priori parecidas, con una característica o funcionalidad completamente distinta. Estos cambios se dieron, sobre todo, entre el primer seguimiento (PEC1) y el segundo (PEC2). El resultado, no obstante, es el mismo en, supuestamente, menor tiempo.

9. Bibliografía

1. Unity Store Plans, diciembre de 2021
2. Unreal Engine pricing, diciembre de 2021
3. Steam Direct, diciembre de 2021
4. GOG Submit Game, diciembre de 2021
5. Epic Games Publishing, diciembre de 2021
6. Apple Developer membership, diciembre de 2021
7. Apple Developer – What's included, diciembre de 2021
8. Android Developer signup, diciembre de 2021
9. Thunder Client, octubre de 2021
10. Unity Manual, consulta frecuente
11. Mirror Networking, 20 de septiembre de 2021 en adelante
12. 3DJuegos, consulta frecuente (información sobre lanzamientos)
13. Reddit Unity3D, consulta frecuente (foro de desarrolladores)
14. Gamasutra (ahora GameDeveloper), consulta frecuente
15. GameCareerGuide, consulta previa al TFG

10. Anexos

Descarga e instalación

La descarga se encuentra en el archivo Readme de la página de GitHub, aquí. Para la ejecución, simplemente extraer y hacer click en el archivo ejecutable. Disponible para PC Windows o Mac.

Controles del videojuego

- Movimiento: W, A, S y D
- Acción: E
- Transformación (promorfo): E
- Ataque (cazador): click primario del ratón
- Menú: ESC

Partidas de prueba

- Partida 1 (promorfo)
- Partida 2 (cazador)

Las partidas de prueba se realizaron en la versión 0.8.4 y sirvieron para arreglar varios errores críticos de cara a la versión final, referentes a los límites del escenario e interfaz.

Los usuarios que participaron fueron de todo tipo (técnicos, jugadores o más casuales). La acogida fue buena; tanto, que la sesión de prueba se alargó una hora y media (de la media hora inicial) para seguir probándolo. Fue todo un éxito, no solo por la buena acogida, sino también por el aporte de ideas, muchas de ellas reflejadas en el apartado 7.5.

Videojuegos tomados como inspiración

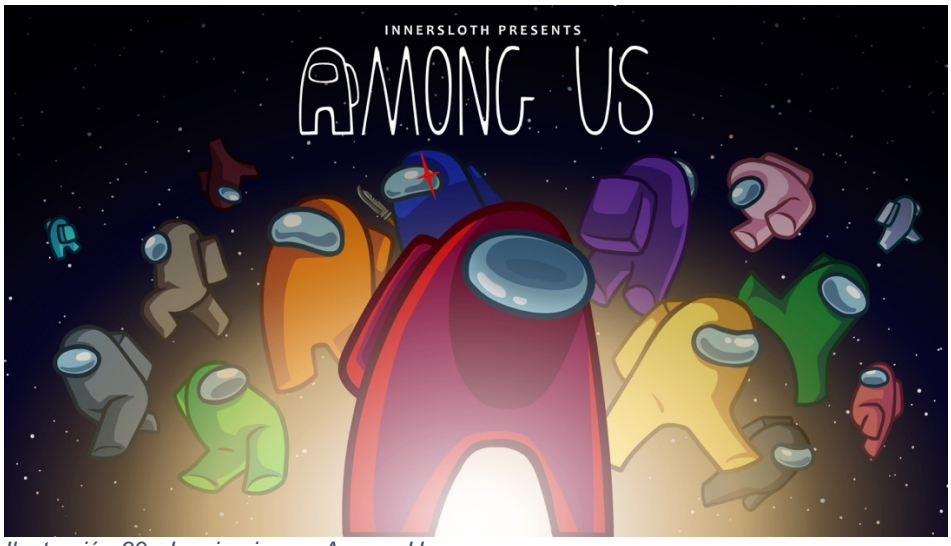


Ilustración 29 - Inspiraciones: Among Us

Por su simpleza, estilo visual simple y efectivo y su capacidad de reunir a varias personas durante un buen rato, Among Us es una de las mayores inspiraciones para este proyecto.

Ilustración 30 - Inspiraciones: Borderlands 3

El estilo de arte de Borderlands 3 podría ser muy acertado para este proyecto, combinándolo el estilo “cómic” con modelados low-poly.

Ilustración 31 - Inspiraciones: Valorant

El estilo cell-shading de Valorant también es un buen candidato para el proyecto. No sería imposible hacer una combinación de ambas, manteniendo los contornos del estilo cómic con las texturas más planas del cell-shading.

Ilustración 32 - Inspiraciones: Half Life 2

El menú principal de Half Life 2 es una obra maestra en este aspecto. No es solo un menú dinámico, sino que la escena irá cambiando al nivel en el que nos encontremos actualmente. El juego tiene muchísimas más virtudes, pero no aplican para este proyecto.

Ilustración 33 - Inspiraciones: Counter Strike

Aunque pueda parecer que Counter-Strike no tenga absolutamente nada que ver con este videojuego, su diseño de niveles es excepcional. No en cuanto al visual, sino la estructura. Se ha intentado llevar esta misma experiencia a The Prop Hunt, haciendo una estructura del escenario que sea justa para ambos bandos.

Ilustración 34 - Inspiraciones: League of Legends

League of Legends lleva más de 10 años en el mercado y se podría decir que ahora vive su mejor momento. El sistema de personajes y habilidades propia de los MOBA ha influenciado directamente en la planificación (a futuro, pues era difícil de implementar en un periodo tan acotado) del proyecto. Además, uno de sus últimos modos de juego “Ultimate Spellbook” provocó un cambio en el sistema inicial de héroes del videojuego, en favor de un sistema de elección de habilidades. Aunque no es posible ver la influencia en la versión piloto del juego más allá de una vista aérea similar, si lo sería en una posible versión gold.