YouRuleYouRole

Trabajo Fin de Grado
Curso 2020-2021

Autor
Rodríguez-Peral Bustos Álvaro

Director
Vázquez Poletti José Luis

Grado En Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

YouRuleYouRole

Trabajo de Fin de Grado en Desarrollo de Videojuegos
Departamento de Arquitectura de Computadores y Automática

Autor
Rodríguez-Peral Bustos Álvaro

Director
Vázquez Poletti José Luis

Convocatoria: Febrero/Junio/Septiembre 2021
Calificación: Nota

Grado en Desarrollo de Videojuegos
Facultad de Informática
Universidad Complutense de Madrid

DÍA de MES de AÑO

DEDICATORIA

A la gente de la Mazmorra del androide por esas largas noches en el local de Jaime y por esas noches que no han podido ser, pues han inspirado este proyecto.

AGRADECIMIENTOS

A todos los que me han soportado y apoyado durante tantos años cada vez que hablaba de mis ideas para algún juego que me gustaría hacer, si nadie me hubiera dado alas no habría podido llegar hasta aquí. Agradezco a mis padres por no rendirse conmigo ni con mi educación aunque haya sido difícil y también a mi tutor José Luis Vazquez-Poletti por el apoyo y la guía que me ha ofrecido durante este tiempo. Muchas gracias a todos.

Resumen

YouRuleYouRole

YouRuleYouRole (en adelante YR2 para abreviar) es un servicio online que pone en contacto a jugadores de rol clásico de mesa con directores de partidas o masters. Mediante la plataforma los jugadores pueden acceder a partidas a las que hayan sido invitados previamente por un master o director de juego y jugar a partidas dirigidas en tiempo real por dicho máster. Los masters por otro lado disponen de un pseudo motor de videojuegos fácil de manejar para poder montar sus propias partidas y modificarlas incluso en tiempo real. De esta manera se proporciona a los usuarios de la plataforma un sistema de reglas flexible para montar partidas de rol con el tipo de ambientación que prefieran.

Palabras clave

Rol, Rpg, Mesa, Tablero, Cloud, Online, Multijugador, Director, Jugador, Partida.

Abstract

YouRuleYouRole

YouRuleYouRole (hereinafter YR2 for short) is an online service that connects classic tabletop rpg players with game directors or masters. Through the platform, players are able to access games to which they have been previously invited by a master and play games in real time created and directed by said master. The masters, on the other hand, have an easy-to-use pseudo video game engine to be able to set up their own games and modify them even in real time. This way, users of the platform are provided with a flexible rule system to set up role-playing games with the type of setting they prefer.

Keywords

Role, Rpg, Table, Tabletop, Cloud, Online, Multiplayer, Master, Player, Game.

Índice de contenidos

Índice de figuras

Índice de tablas

Capítulo 1 - Introducción

Jugar a rol de mesa puede llegar a ser bastante complicado por varias razones. En muchas ocasiones es difícil coordinar a tanta gente para poder echar una partida (teniendo en cuenta que hacen falta un mínimo de 3 personas para poder jugar siendo lo recomendable entre 4 y 6), otras veces no existe un lugar adecuado donde poder reunirse para jugar y también existen en otras ocasiones barreras como la dificultad de ciertos sistema de juego, que pueden llegar a requerir de realizar diversas tiradas con distintos tipos de dados (de 4, 6, 8, 10 o 20 caras) para resolver a veces algo tan sencillo como un simple espadazo.

YouRuleYouRole pretende superar estas limitaciones proporcionando un espacio virtual en el que poder reunirse a celebrar partidas de rol mediante un sistema de reglas sencillo, proporcionando a la figura del master un pseudo motor de videojuegos donde desarrollar sus partidas sin necesidad de saber programar y donde los jugadores podrán ver de manera intuitiva los efectos de sus acciones en el juego sin necesidad de largos proceso de tiradas y comprobar tablas.

De este modo un grupo de amigos puede reunirse para echar una partida sin necesidad de descuidar otras obligaciones, siempre que puedan tener acceso a un ordenador con conexión a internet.

1.1 Motivación

Como se ha comentado lograr reunir la cantidad de gente necesaria para celebrar una sesión de rol de mesa, usualmente entre 4 y 8 personas (aunque el mínimo necesario sería de 3), puede ser complicado a veces. Estas complicaciones van a más según se van cumpliendo años y adquiriendo compromisos de los que se carece cuando se es más joven.

1.2 Objetivos

YR2 pretende facilitar este tipo de reuniones llevándolas al ámbito virtual, no obstante, hacer solamente eso no sería nada novedoso, puesto que ya existen plataformas como Roll20 que ofrecen este tipo de servicios.

La idea consiste en mejorar este tipo de experiencias proporcionando además un entorno 3D personalizado para los jugadores, donde ocurrirán sus aventuras. Esto será posible mediante un sencillo editor al que el máster tendrá acceso. La novedad reside en que la figura del máster tendrá además una serie de herramientas para automatizar ciertos eventos que tendrán lugar durante la partida, así como la posibilidad de influir directamente en esta en tiempo real proporcionando a los jugadores una experiencia completamente orgánica y diferente cada vez, en la que el mundo responde a sus acciones y ellos ven esto en la propia pantalla de su ordenador.

1.3 Plan de trabajo

Para el plan de trabajo se dividió en tareas el trabajo a realizar y se le dió a cada tarea una puntuación de 0 a 3, significando 0 que la tarea se realiza en pocos minutos y 3 que la tarea puede llevar hasta unas 10 - 12 horas. Si una tarea es demasiado grande, se ha dividido en varias subtareas.

Area de trabajo	**Tarea**	**Subtarea**	**Tiempo**	**Hecho**
Arquitectura de red	Base de datos	Crear estructura de la base de datos y desarrollarla teniendo en cuenta sus modificaciones mediante la API	2	X
	Funciones Lambda y API	Planteamiento del funcionamiento de la API y las	2	X

		llamadas que recibirá de parte de los clientes		
		Función creaUsuario	1	X
		Funcion creaPartida	1	X
		Función cargaPartida	2	X
		Función guardaPartida	2	X
		Función actualizaPartida	2	X
		Función play	1	X
		Función login	1	X
		Función logicaJuego	2	X
		Función cambio	2	X
Programa cliente	Llamadas a la API	Implementación de las llamadas a la API desde el cliente	1	X
	Lógica del cliente	Implementación del sistema de red para mandar peticiones a la API sobre el estado del juego y responder a los cambios del estado del juego	2	X
		Creación del editor de eventos	3	X
		Creación del editor de personajes	2	X
		Menú principal	3	X
		Creación de menús y UI del máster	3	X
		Creación de menús y UI de los jugadores	2	X
	Parte audiovisual de la aplicación	Implementación de efectos gráficos, efectos de partículas y efectos sonoros	2	X

Desarrollo de la memoria		Desarrollo del manual de usuario (Apéndice)	3	X
		Desarrollo de caso de uso	2	X
		Estudio de costes	3	X
		Desarrollo de la memoria	3	X
Pruebas		Prueba de la API	2	
		Prueba del editor de eventos	2	X
		Prueba del cliente y la interacción entre sus sistemas	2	X
Arreglos y ampliaciones		Arreglos 1	3	X
		Arreglos 2	3	X
		Redacción de posibles ampliaciones	1	X
		Subir proyecto a repositorio de mercurial	1	X

1.3.1 División de tareas

El resultado es de aproximadamente 220 horas de trabajo. Dedicando una media de 15 horas semanales el trabajo da un total de 14,5 semanas de trabajo, unos 3 meses y medio.

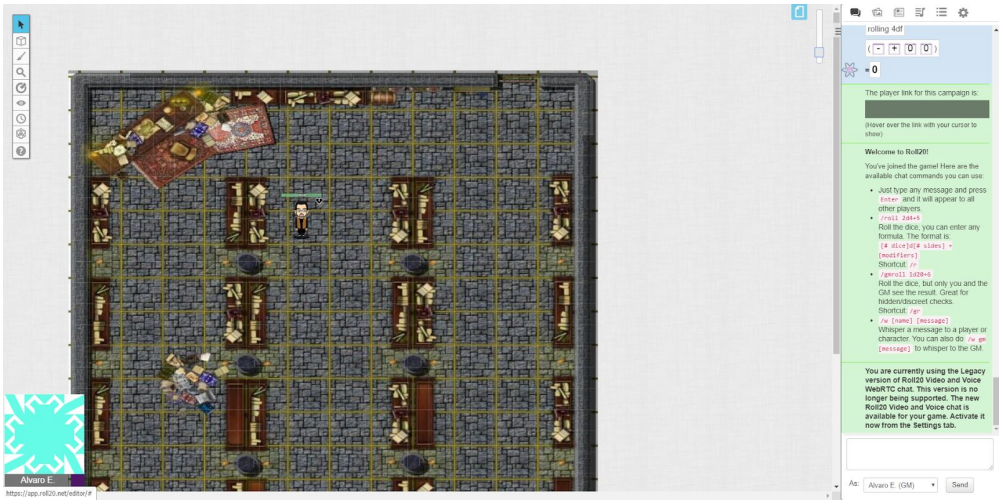
Capítulo 2 - Estado del arte

Existen ciertos servicios o alternativas que ofrecen algo parecido a la propuesta de YR2, siendo el principal de ellos Roll20[1], un servicio online que al igual que YR2 pone en contacto a jugadores y masters de rol para jugar partidas a través de internet, otra alternativa serían los motores de videojuegos RPG Maker[2], que son muy visuales e intuitivos y existen también ciertos videojuegos como Neverwinter Nights[3], (aunque no solo este), que ofrecen a sus jugadores acceso a un editor de niveles de manera que los jugadores pueden crear sus propios niveles y compartirlos con otros jugadores para que jueguen .

Es necesario mencionar las diferentes opciones que existen para la computación en la nube, así como algunos juegos o servicios que hacen uso de esta y hablar también de sus ventajas e inconvenientes así como su nivel de implantación en el mercado actual.

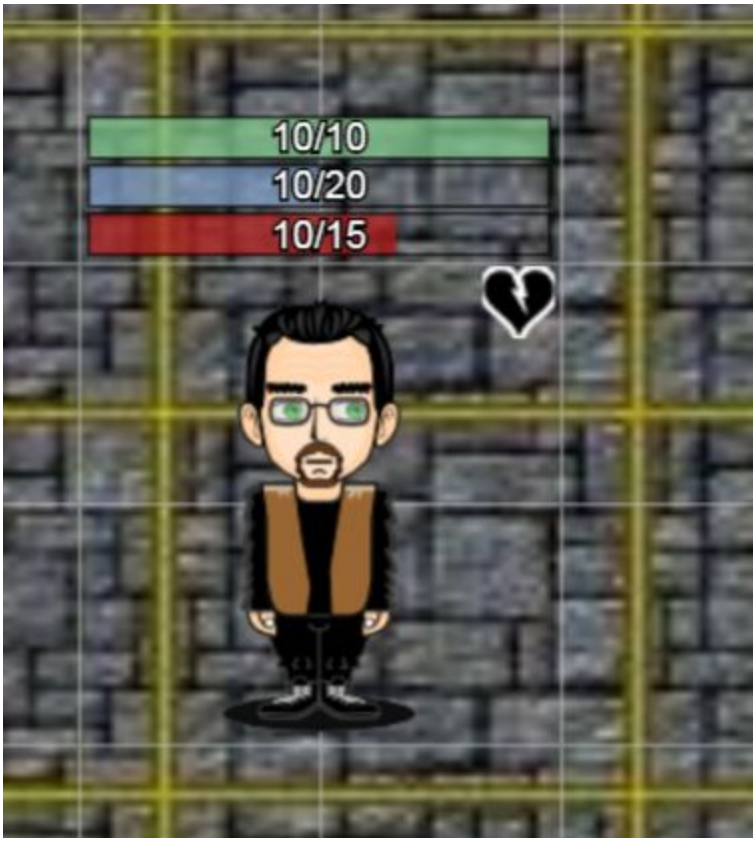
2.1 Roll20

Roll20, es un servicio web gratuito con funcionalidades de pago cuyo aspecto general se puede ver en la figura 2.1.1. La principal diferencia con YR2 radica en que, mientras que YR2 pretende ofrecer una experiencia más interactiva y audiovisual, Roll20 no ofrece esta posibilidad más allá de sprites inmóviles, algunos efectos simples y la posibilidad de usar músicas de fondo, y tampoco ofrece las posibilidades de automatización de eventos que ofrecerá YR2.



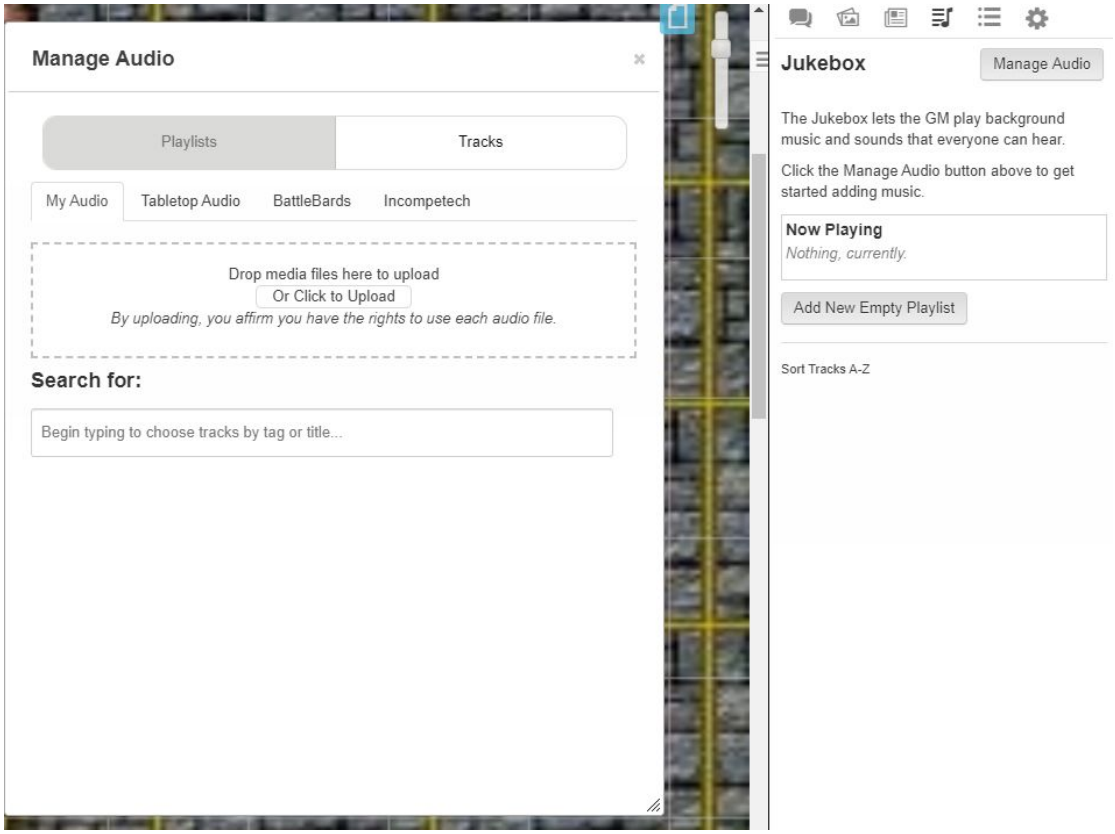
2.1.1 Aspecto general de Roll20.

El sistema ofrece la posibilidad de crear una serie de mapas a los que asignar una imagen de fondo y dividir esta en casillas que podrán ser usadas en caso de que la partida en cuestión lo requiera. Por otro lado ofrece la posibilidad de asignar un token a cada jugador que representará a su personaje y que puede tener diferentes barras para representar parámetros de manera gráfica, como se ve en la figura 2.1.2.



2.1.2 Muestra de las barras de parámetros.

También ofrece la posibilidad de crear fichas de personaje, e incluso ofrece modelos de fichas de personaje de fichas de personaje de distintos sistemas populares como pueden ser D20, Fate o D&D. Todo esto viene complementado con la posibilidad del máster de reproducir música y efectos de sonido mediante un menú a la derecha de la pantalla como se ve en la figura 2.1.3.



2.1.3 Manejo del audio en Roll20.

Roll20 ofrece la posibilidad de automatizar ciertos eventos sencillos que incluso tengan efecto en las fichas de jugador, pero las posibilidades de estos son limitadas y son complicados de manejar ya que implican el conocimiento de los códigos del chat. De hecho, si este servicio puede tener una crítica es precisamente que el sistema de chat es un sistema bastante complicado de aprender y al menos el master debería aprenderlo para poder sacarle todo el jugo al sistema, ya que este tiene una serie de códigos que amplían la funcionalidad básica del sistema y pretenden agilizarlo,

Este servicio se monetiza a través de una tienda que permite acceder a recursos gráficos y sonoros más allá de los proporcionados de base, permitiendo que cualquiera pueda publicar sus recursos y venderlos, llevándose Roll20 una parte de los beneficios.

Además ofrece una subscripción que amplía el servicio con ciertos elementos como la posibilidad de acceder a Roll20 en dispositivos móviles.

Es importante citar el valor de la comunidad en un servicio de esta índole y Roll20 ofrece por ello un foro en el que se organizan diferentes partidas y se discute acerca del juego.

2.2 Rpg Maker

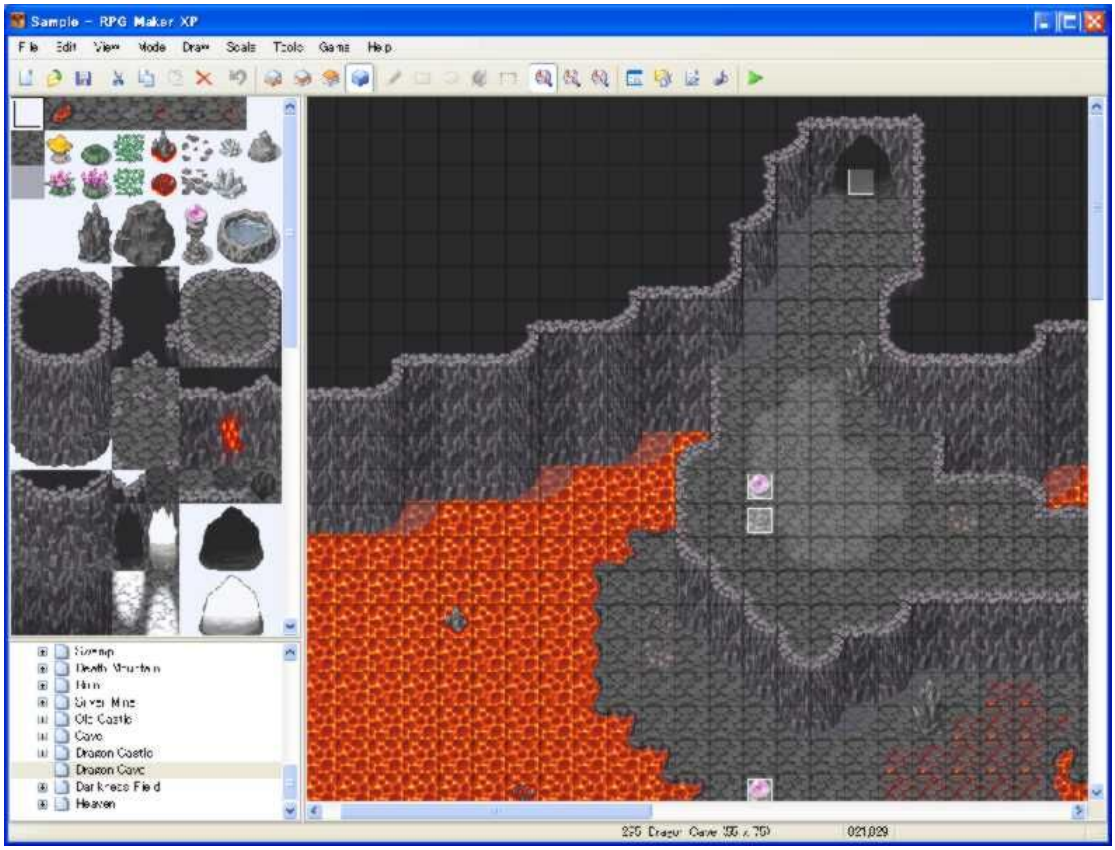
Otra influencia de la que bebe este proyecto serían los populares motores de videojuegos RPG Maker, que en sus múltiples versiones ofrecen la posibilidad de desarrollar videojuegos de rpg sin necesidad de saber programar.

Estos motores, de videojuegos en 2d ofrecen un sencillo editor de mapas y un sistema de objetos personalizables que pueden ser programados visualmente para responder de diferentes maneras, ofreciendo la mayoría de elementos jugables de un RPG tradicional; como un sistema de inventario y equipamiento, un sistema de niveles, un sistema de combate por turnos y un sistema de diálogos, todos ellos muy sencillos de manejar mediante una interfaz gráfica.

Estos motores de videojuegos están divididos en tres partes diferenciadas:

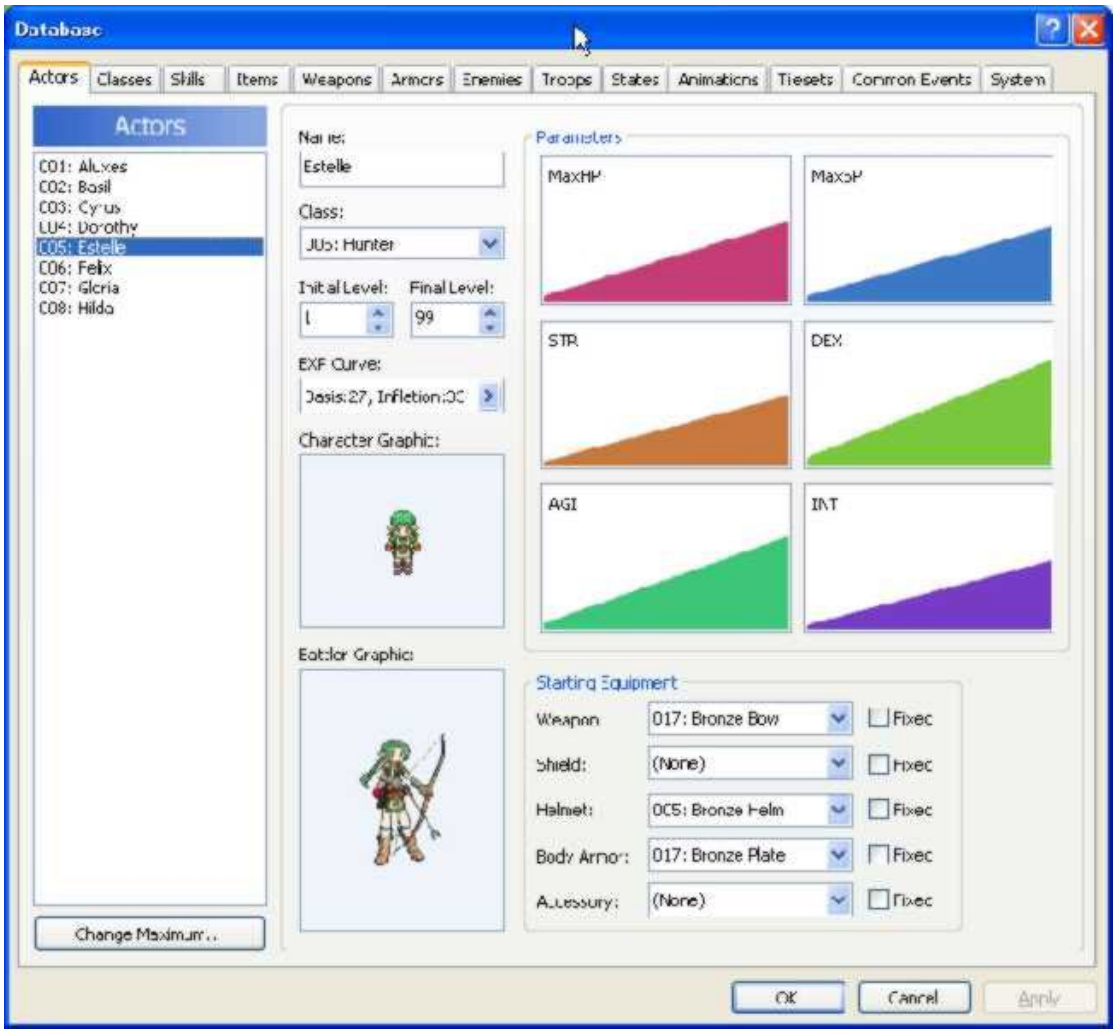
- Editor de mapas → Desde el que se pueden crear nuevos mapas, editarlos y crear objetos interactivos (eventos), se puede ver el editor de mapas en la figura 2.2.1. El editor de mapas es lo primero que se ve al abrir el motor. Este editor se divide generalmente en 3 capas, aunque algunas versiones del programa tienen más:
    - Inferior → Se pintará debajo de los personajes.
    - Superior → Se pintará por encima de todos los demás objetos a no ser que se trate de un objeto de evento configurado específicamente para ser mostrado por encima.

- Eventos → Se trata de la capa intermedia a la que pertenecen los personajes y el resto de objetos interactuables del juego (a no ser que se indique lo contrario).



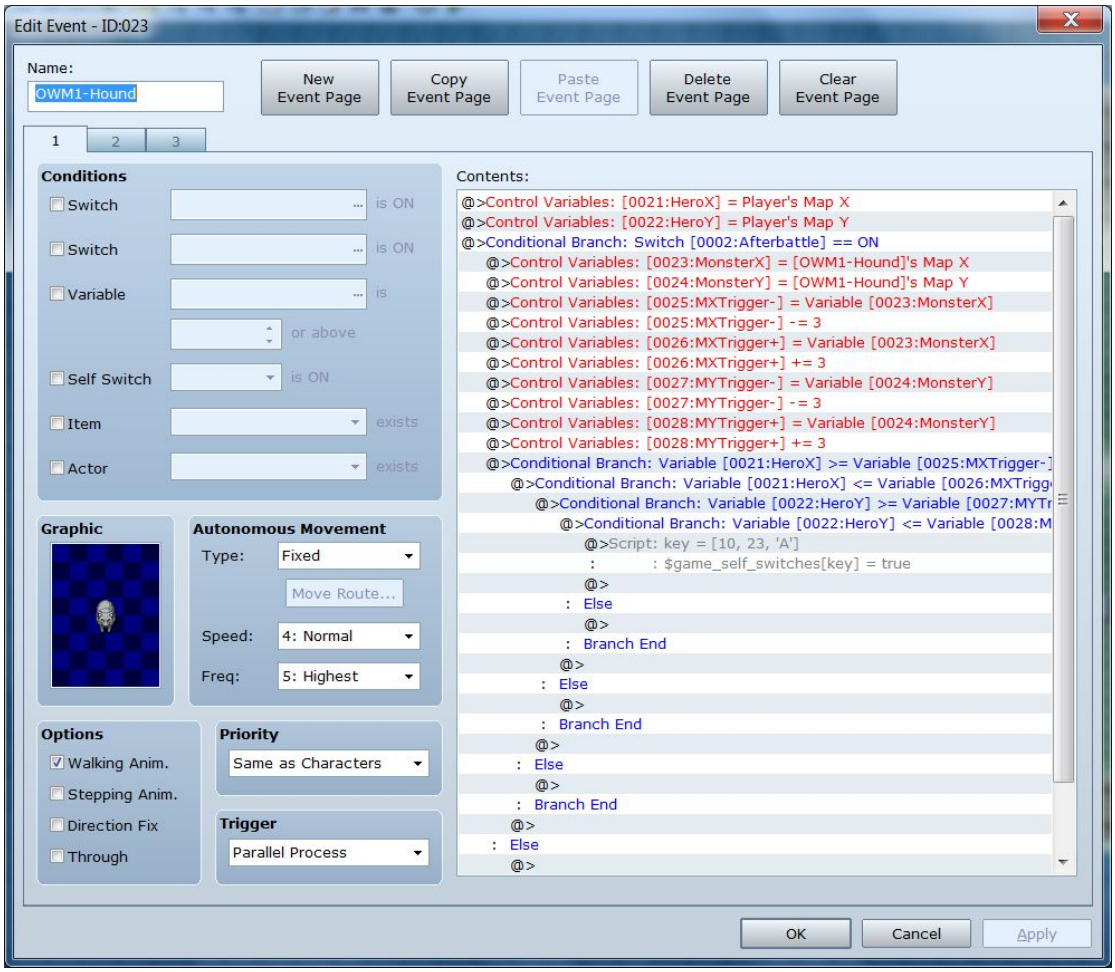
2.2.1 Editor de mapas de RPG Maker XP.

- Base de datos → Desde donde se pueden modificar los atributos de los personajes como sus sprites asociados, sus parámetros de ataque, defensa, etc… A la base de datos también pertenecen los datos de los enemigos que se podrán encontrar durante el juego, las animaciones de combate, los objetos tanto equipables como no equipables y un largo etcétera. La base de datos es accesible desde la barra de herramientas del motor y tiene el aspecto que se muestra en la figura 2.2.2.



2.2.2 Base de datos de RPG Maker XP.

- Editor de eventos → Mostrado en la figura 2.2.3, desde este editor se puede programar y cambiar la lógica del juego. Los eventos pueden ser automáticos o responder a condiciones programables (tales como el valor de una variable o que se pulse el botón de acción junto a él). Se accede a él desde el editor de mapas haciendo doble click en un evento.



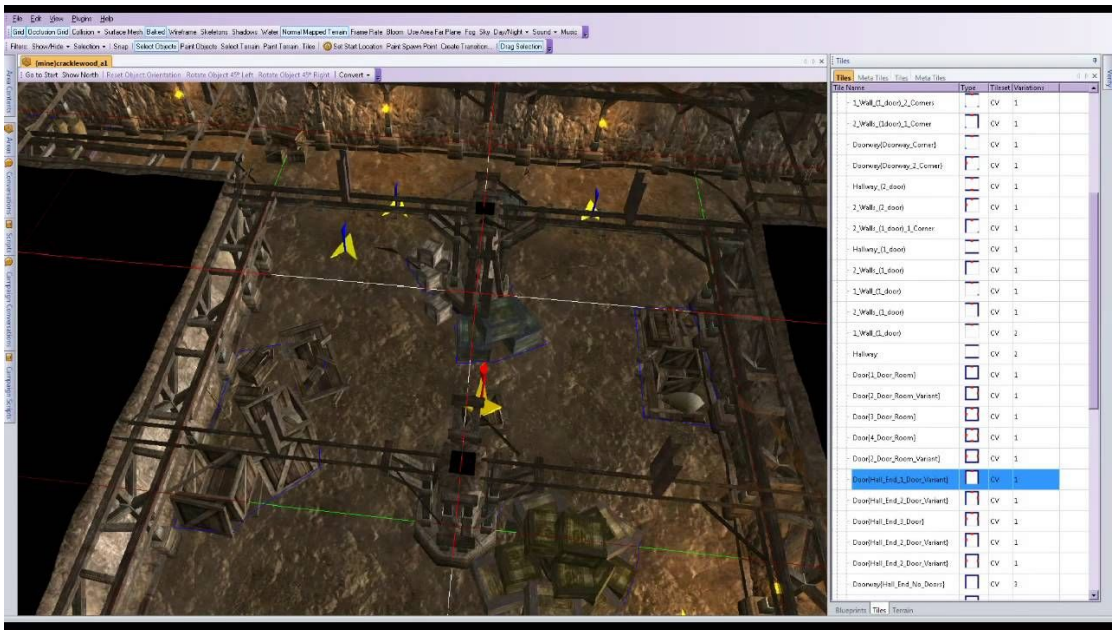
2.2.3 Editor de eventos de RPG Maker.

2.3 Neverwinter Nights

También hay que resaltar que existen ciertos videojuegos comerciales como Neverwinter Nights, que ofrecen un editor de niveles con el que los jugadores pueden realizar sus propias partidas y compartirlas con otros jugadores.

No obstante la figura del máster pierde importancia aquí pasando a ser un mero diseñador de niveles, pués una vez acabada su labor de diseño no tiene control sobre lo que pueda pasar en las partidas que haya diseñado. La diferencia y ventaja de este videojuego respecto a los servicios anteriores como Roll20 y los motores como RPG Maker es que incluye un editor de personajes y su sistema de juego es más dinámico al

tratarse de un action-RPG en tres dimensiones. Puede verse el editor de mapas en la figura 2.3.1.



2.3.1 Editor de mapas de Neverwinter Nights 2.

2.4 Cloud gaming

En cuanto a las tecnologías en la nube aplicadas a videojuegos, estas se han aplicado sobre todo al streaming de juegos con poco éxito hasta el momento, debido sobre todo al problema de la latencia, que deriva de la necesidad de recibir el input para el juego por red antes de poder procesar toda la información y enviarla al jugador. En este tipo de sistemas el juego y su información se procesan en servidores remotos y se manda al dispositivo en el que se juega tan solo la información audiovisual necesaria, recibiendo de este dispositivo la información de entrada para poder interactuar con el videojuego. Ejemplos de estas plataformas de streaming de juegos pueden ser algunos como G-Cluster, OnLive y GameFly, o más recientemente el servicio Google Stadia padeciendo todos en general el mismo problema. Como

puntos positivos que ofrecen las tecnologías en la nube a los videojuegos podemos citar la promesa a los desarrolladores de no tener que preocuparse en el futuro de que el dispositivo en el que se va a jugar sea capaz de mover sus juegos, y la dificultad de piratear un videojuego que no se ha distribuido en el sentido tradicional de la palabra.

Por otro lado, existen diversos servicios para la computación en la nube que es necesario mencionar, ya que este tipo de servicios han sido utilizados para dar forma a este trabajo. Los servicios más utilizados y conocidos son los que ofrecen Google con Google Cloud computing, Microsoft con Microsoft Azure y Amazon con Amazon Web Services o AWS.

En cuanto a las tendencias de la computación en la nube se puede ver por la encuesta llevada a cabo por RightScale en 2017[4] y la llevada a cabo por IDG en 2020[5], que las empresas tecnológicas cada vez utilizan más la computación en la nube y la tendencia para la computación en la nube es de crecimiento para los próximos años.

Aunque algunas opciones son mejores que otras dependiendo del servicio que se vaya a utilizar, el coste del servicio de soporte difiere en gran medida siendo el coste del servicio de soporte de Google más barato de 150$ al mes, por lo que en caso necesitar hacer uso de los servicios de soporte Amazon es más barato y además está más extendido.

Respecto al uso de estos servicios de computación en la nube podemos mencionar grandes títulos por parte de Google como Apex Legends o Fornite por parte de Amazon, con lo cual queda de manifiesto que las 3 opciones son válidas para el uso que se le quiere dar.

De entre las 3 posibles opciones se ha elegido AWS debido a que es la más utilizada y por tanto se supone más sencillo encontrar información sobre ella y la

solución a posibles problemas que se hubieran podido encontrar durante el desarrollo, sin necesidad de pagar ningún servicio de soporte aparte.

Capítulo 3 - Descripción de YR2

De la mezcla de los conceptos mencionados en el apartado anterior nace YR2. En YR2, a través de un simple sistema de objetos a imagen del sistema de eventos ofrecido por RPG Maker, el máster podrá colocar en una cuadrícula en 3d diferentes objetos y programarlos para que respondan de diferentes maneras a una serie de comandos que puedan realizar los jugadores sobre cada objeto. La figura del máster contará también con un sencillo editor de escenarios para colocar objetos que no sean interactuables. El manejo del juego se realizará a través de teclado y ratón, siendo el ratón utilizado para ajustar a donde mira la cámara y utilizar las habilidades del jugador, y el teclado para el movimiento del personaje, escribir en el chat y ciertos atajos.

Los elementos básicos que conforman el concepto de YR2 son los siguientes:

- Sistema de juego.
- Sistema de chat.
- Interfaz de jugador.
- Interfaz de máster.
- Editor de mapas.
- Editor de eventos.

3.1 Sistema de juego

Estableciendo un paralelismo con los sistemas de reglas del rol tradicional, siendo este el que ayuda a determinar los resultados de las acciones del juego, cada jugador podrá manejar un personaje que puede ser creado desde un editor de personajes. Los siguientes parámetros que se corresponden a las acciones básicas que cualquier personaje puede realizar durante una partida:

- Atacar
- Recoger
- Provocar
- Engañar
- Carisma
- Robar

Cada jugador puede utilizar su personaje realizando cualquiera de estas acciones y estas serán notificadas al máster. Será tarea del máster programar las reacciones o responder el mismo a estas acciones una vez vea que se han realizado. Será también decisión del máster tener en cuenta o no los valores de estos parámetros al actuar durante una partida o al automatizar eventos, no obstante es recomendable seguir las reglas.

3.2 Sistema de chat

Mediante él los jugadores y el propio máster estarán en contacto y podrán comunicarse entre sí, este estará disponible en la parte inferior de la pantalla, tanto para los jugadores como para el máster. Para el TFG el sistema de chat será exclusivamente un chat de texto y en caso de querer añadir un chat más elaborado se utilizará alguna herramienta ya disponible a tal efecto. La posible inclusión de un chat de voz se tratará de una ampliación de cara a futuro.

3.3 Interfaz de jugador

Utilizando un sistema de cámara en 1ª persona, se utilizará el ratón para orientar la cámara, elegir las habilidades a usar (con la rueda del ratón) y usarlas (click izquierdo), con el botón central se puede elegir interactuar con una acción por

defecto con el objeto en cuestión. El teclado se utilizará para el movimiento del personaje.

En la parte superior de la pantalla el jugador podrá editar el aspecto de su personaje.

3.4 Interfaz de máster

Al crear una nueva partida el máster podrá ver un terreno plano La partida empezará en modo **stop**, por lo que algunos comandos no estarán disponibles para el máster en este modo. Con una vista en 1ª persona el máster podrá editar este terreno y afectar a algunos aspectos de la partida con los siguientes comandos elegibles a través de un menú al hacer clic derecho con el ratón:

Con **crear / modificar evento** se cambia el modo de edición al modo de edición de eventos. Existen otros modos de edición como el modo borrar eventos que se activa con el botón **borrar evento** o el modo de edición del mapeado que se activa con el botón **elementos mapeado**.

Con el botón **instanciar evento** se puede crear un nuevo **evento** en base a los **eventos predefinidos** que ya existan.

Se pueden cambiar directamente los atributos de un personaje con la opción **cambiar atributo**.

Se pueden modificar valores de las variables mediante los comandos **generar aleatorio** y **variables**. La primera genera un número aleatorio que se guarda en la variable RAMDOM y la segunda cambia directamente el valor de una variable.

A través de este menú se puede también afectar al ambiente del juego con opciones como **efecto climático**, que cambia la hora y el tiempo climático entre soleado y lluvioso, **sonido / música ambiente**, que hace lo que su propio nombre

indica, **fx sonido**, que reproduce un efecto de sonido o **efectos especiales**, que reproduce un efecto de partículas.

Por otro lado, se puede ayudar a ambientar la partida mostrando a los jugadores videos o imágenes directamente mediante la opción **video / imagen** y se puede afectar directamente a los jugadores ofreciéndoles toma de decisiones con **encuestas**, aplicando fuerzas a los objetos que les representan con **fuerza**, moviendo a los jugadores a otros mapas o posiciones con **teletransporte**, y crear nuevos mapas mediante la opción **crear / cambiar mapa**.

Algunas de estas opciones también están disponibles como comandos configurables para los **eventos**.

3.4.1 Editor de Mapas

Es la interfaz por defecto del máster, mediante la cual se da forma al mapa, permite añadir y modificar los dos tipos de objetos que componen los mapas, siendo estos los **elementos** y los **eventos**.

Los **elementos** son objetos no interactuables que tan solo tienen una entidad física, sirven para limitar escenarios y como parte de la escenografía de cada mapa propiamente dicho.

Así mismo, los **eventos** son objetos programables y permiten a los jugadores interactuar con ellos mediante alguno de los 6 tipos de acciones que tienen a su disposición o mediante la acción de interactuar por defecto. Una de las reacciones programables es la reacción por defecto, para los casos en los que el máster prefiera que el evento reacciones de la misma manera sea cual sea la interacción con el evento, la página por defecto es la que se reproducirá si el jugador a interactuado con el **evento** y la página correspondiente a la interacción del jugador está vacía o el

jugador usó la interacción por defecto. Se amplía la información respecto a los eventos en la sección 3.4.2.

3.4.2 Editor de Eventos

Se puede acceder al editor de eventos desde el editor de mapas al crear un evento o modificar una ya existente. Mediante el **editor de eventos** se pueden programar las respuestas de los eventos a las acciones de los personajes, programando incluso una respuesta diferente según el tipo de interacción. Se puede también modificar el aspecto del objeto del evento y cambiar el tipo de disparador del evento para que se reproduzca al pasar sobre cierta zona o cuando un jugador entre en el mapa por ejemplo.

Para programar la respuesta o comportamiento de un evento se le pueden introducir comandos desde una interfaz visual, pudiendo dividir estos comandos en 3 categorías diferentes; **comandos del sistema**, **comandos del juego** y **comandos de personaje**.

3.4.2.1 Comandos del sistema

Son los comandos que afectan directamente al flujo de ejecución de su evento o de otros eventos. Estos comandos son: **ejecución condicional**, **bucle**, **instanciar evento**, **borrar evento**, **variables** y **guardar atributo.**

Los comandos de **ejecución condicional** son 2, marcando uno el punto de inicio del código que se debe ejecutar en caso de que se cumpla la condición y otro el punto final de dicho comando.

Con respecto a los comandos de **bucle** pasa lo mismo que con los comandos de **ejecución condicional**, existiendo 2 comandos separados para marcar el inicio y el final del código que debe ejecutarse en bucle hasta que la condición de entrada deje de cumplirse.

Existe en YR2 la figura del **objeto predefinido**, siendo este un objeto de tipo evento que ha sido preparado con anterioridad y guardado para poder ser usado de manera rápida en diferentes ocasiones. Mediante esta opción es posible crear una instancia de un **objeto predefinido** en el punto del mapa que se le indique.

El comando **borrar evento** permite borrar un evento existente elegible desde un desplegable.

También es posible cambiar el valor de una variable dentro del juego desde el comando **variables**. Las variables dentro de YR2 solo pueden tener valores enteros, esto es así con el objetivo de mantener la sencillez buscada en su manejo.

Por último existe la posibilidad de guardar el valor de un atributo de algún personaje como una variable en caso de querer usarlo como valor de comparación para algún **bucle** o **ejecución condicional** mediante el uso del comando **guardar atributo.**

3.4.2.2 Comandos del juego

Los **comandos del juego** son aquellos que están más relacionados con el desarrollo más inmediato de la partida, se utilizan para afectar directamente a los hechos que tienen lugar en la propia partida.

El comando **texto** escribe un mensaje en el chat, pudiendo utilizarse para comunicarse con los jugadores de parte de un personaje no jugador por ejemplo, o para describir situaciones que podrían ser difíciles de crear con el propio motor de YR2 si el máster prefiere mantener las cosas sencillas.

Con el comando **encuesta** muestra a los jugadores la posibilidad de elegir entre varias opciones, la opción tomada por cada jugador se guardará en una variable nombra *OPTION* seguido del nombre del jugador.

Mediante el comando **teletransporte** el jugador especificado será movido a la coordenada elegida del mapa indicado.

También existe el comando **fuerza** que aplicará una fuerza sobre un objeto en específico, aunque este comando sólo puede afectar a **jugadores** o a **eventos** del tipo adecuado(PNJ).

Para cambiar la música o el efecto de sonido de ambiente que suene en ese momento, o ambas pistas a la vez se utilizará el comando de **sonido ambiente / música**, y si lo que se desea es reproducir algún efecto de sonido en específico se usará el comando **fx sonido**. De momento los valores de sonido ambiente y música son globales para toda la partida, aunque en un futuro sería recomendable separarlos y hacer que sean parte de la información del mapa, o incluso que sean individuales para cada jugador.

Mediante el comando **video / imagen** se permite mostrar a todos los jugadores una imagen o un video, que pueden ser utilizados con fines de ambientación y para poner en situación.

Por otro lado, si se quiere definir el camino que debe seguir un personaje no jugador en cierto momento, se puede utilizar el comando **ruta,** al usar este comando se permite configurar una serie de instrucciones indicando la dirección en la que debe moverse el **evento**, que luego se introducirán como comandos individuales en la página de comandos **.**

También se puede cambiar los atributos de un personaje mediante el uso del comando **cambiar atributos**, de esta manera se introduce la posible mecánica de responder de una manera diferente a las acciones de un personaje dependiendo de su nivel de habilidad en la tarea que se propone llevar a cabo.

Por último existen un par de comandos utilizados también para la ambientación de las partidas como son el comando de **ambiente / clima**, que permite cambiar la iluminación del mapa en función de la hora del día que sea, así como cambiar el tipo de clima entre despejado y lluvioso, y el comando de **efectos especiales**, que reproducirá un efecto de partículas en la posición indicado de entre los disponibles.

3.4.2.3 Comandos de personaje

Hacen referencia a las acciones que pueden realizar los jugadores, de esta manera se pueden utilizar los objetos de tipo **evento** para simular acompañantes o personajes no jugadores mediante los comandos que realizan estas acciones, que son los siguientes:.

- Atacar
- Recoger
- Provocar
- Engañar
- Carisma
- Robar

Capítulo 4 - Arquitectura de YR2

YR2 se ha desarrollado en base al motor de videojuegos Unity, pero las características que lo diferencian viene de su arquitectura de red que viene detallada más adelante.

4.1 Arquitectura Serverless

Para desarrollar YR2 se partirá de una arquitectura serverless aprovechando el ecosistema AWS, mediante estos servicios se desarrollarán varias funciones lambda gracias al servicio AWS Lambda[6]. El servicio AWS Lambda permite ejecutar código, en nuestro caso en javascript, sin tener que preocuparse de pagar ni administrar servidores. Hay que tener en cuenta que este código de las funciones lambda se ejecuta sin ningún contexto por lo cual éste debe ser proporcionado por los parámetros que le llegan a la propia función, a través de una base de datos, o desde ambas fuentes como es el caso.

Para poder llamar a estas funciones lambda, dado que no podemos llamarlas directamente es necesario implementar una API REST que ejercerá de intermediaria entre la función y los clientes que realicen esas llamadas, para ello se utiliza el servicio AWS API Gateway[7], que nos permite crear dicha API y conectarla a las funciones lambda creadas con AWS Lambda. A través de las llamadas a esta API que recibirán la información de contexto necesaria para llevar a cabo sus funciones en JSON, se maneja el input del juego y de los jugadores y el estado de la partida es guardado en una base de datos. En el punto 4.2 se especifica más el uso que se hace del servicio API Gateway para YR2.

La base de datos hace uso del servicio Amazon Simple Storage Service[8], en adelante S3, almacenando en un bucket de S3 la información como archivos de texto planos, un bucket es la unidad básica de almacenamiento que ofrece S3, para el caso

de YR2 solo es necesario utilizar un bucket. Los archivos dentro del bucket de S3 estarán organizados con cierta estructura de archivos y carpetas específica para simplificar su uso y hacerlo más eficiente, esta estructura se explicará más adelante en la sección 4.3 de este capítulo.

La razón de utilizar archivos de texto planos como bases de datos en lugar de usar directamente un servicio de bases de datos que ya te ofrezca la plataforma como puede ser DynamoDb, es en aras de poder independizarse de la plataforma en un futuro si es preciso.

4.2 API Rest

Con el objetivo de mantener conectado el cliente con la base de datos y de gestionar la lógica de cada partida es necesario mantener una API, que es la que se encargará de actualizar la base de datos ejecutando ciertas funciones lambda. Estas funciones son las que determinan en última instancia el coste del servicio de YR2, razón por la cual es aconsejable mantenerlas lo más cortas y eficientes que sea posible. Por esta razón se ha dividido cada posible petición que se pueda hacer a la base de datos como una función separada a fin de mantener este coste bajo.

Las llamadas a la API son las siguientes:

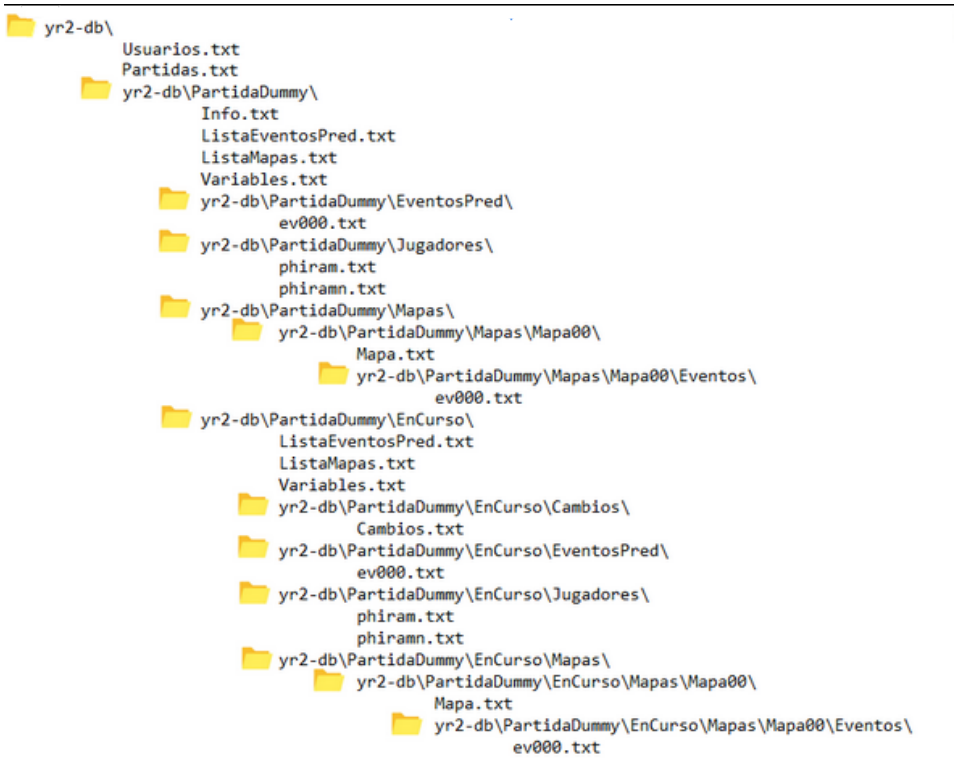
- creausuario → Crea y añade un nuevo usuario del servicio a la base de datos
- creapartida → Crea una nueva partida vacía y la añade a la base de datos con su creador como master
- cargapartida → Recopila los dato de una partida existente en la base de datos y los proporciona al cliente que ha pedido la partida
- guardapartida → Recopila los datos actuales de la partida en el cliente y los manda a la base de datos para sustituir a los que tiene
- actualizapartida → Devuelve al cliente los cambios que ha habido desde la última vez que actualizó su estado

- play → Cambia el estado de la partida en la base de datos y se asegura de que esta está preparada para poder ser jugada
- login → Comprueba si un usuario existe en la base de datos, y si la contraseña que se le ha pasado es correcta devuelve la información de dicho usuario.
- logicaJuego → Ejecuta un paso de lógica de la partida

En los apéndices aparecerá un documento explicando los pormenores del uso de la API y su relación con la base de datos.

4.3 Base de datos

Cómo utilizamos una API para gestionar la lógica del juego, no existe un servidor como tal y toda la información de la partida debe ir guardada en la base de datos, que se irá actualizando constantemente. La base de datos está organizada de la siguiente manera:



4.1 Estructura de carpetas de la base de datos.

La razón para organizar así la base de datos viene de compartimentar lo más posible la información, evitando que cuando se realicen peticiones de algún dato de la base de datos se tengan que cargar grandes archivos lo que podría subir el coste de las funciones lambda por un lado y también podría afectar al tiempo de procesamiento de cada función provocando de esta manera posibles problemas como la aparición de lag.

De esta manera la mayor parte de las funciones se ejecutarán en pocos milisegundos, aunque algunas otras funciones como la de cargar la partida podrían tardar más ya que debe recopilar todos los datos de la partida antes de devolverlos al cliente que haya realizado la llamada.

Por otro lado se mantienen duplicados los datos de la partida con la inclusión de la carpeta **EnCurso** para permitir la opción de guardar el estado de la partida en cualquier momento y poder volver a ese punto de guardado ya que los cambios que se realizan en la partida se ven reflejados en la base de datos en tiempo real debido a la naturaleza de las funciones lambda y su no persistencia.

Por último el fichero usuarios permite que estos sean independientes de las partidas en las que puedan ser incluidos.

Capítulo 5 - Casos de uso

A continuación se definen las diferentes acciones que se pueden realizar en YR2, diferenciadas según el rol que tome el usuario. También se muestra un diagrama de las acciones necesarias a tomar para jugar una partida y quien toma cada una de estas acciones, todo ello mostrado en las figuras de la 5.1 a la 5.6.

Actor: Usuario jugador
Descripción: Representa a un usuario de YR2 que va a tomar el rol de jugador dentro de una partida. No puede actuar sobre la partida más que controlando las acciones de un único personaje.

Actor: Usuario master
Descripción: Representa a un usuario de YR2 que tomará el rol de master dentro de una partida, puede controlar cómo reacciona el entorno y los personajes no jugadores frente a las acciones de los usuarios jugadores.

5.1 Tipos de usuarios del sistema YR2.

Nombre: Cambiar tipo edición
Descripción: Permite cambiar la forma de afectar a la partida por parte del master
Actores: Usuario master
Precondiciones: Usuario existente, Usuario identificado, Partida existente, Partida cargada
Flujo: 1. Hacer click derecho con el ratón en el cualquier punto fuera del GUI. 2. Seleccionar una de las opciones para cambiar el modo de edición entre “Crear / Modificar evento”, “Borrar evento” o “Elementos mapeado”.
Flujo alternativo:
Postcondiciones: Edición Eventos / Edición borrar eventos / Edición mapeado

Nombre: Crear modificar evento
Descripción: Permite modificar directamente un evento dentro del mundo del juego y sus comportamientos.
Actores: Usuario master
Precondiciones: Usuario existente, Usuario identificado, Partida existente, Partida cargada, Edición eventos
Flujo: 1. Hacer click izquierdo en un punto vacío del mapa para crear un nuevo evento o en un punto donde exista un evento para modificarlo.
Flujo alternativo:
Postcondiciones: Editando evento

5.2 Acciones dentro del sistema YR2 parte 1.

Nombre: Login
Descripción: Permite a un usuario de cualquier tipo identificarse dentro del juego YR2.
Actores: Usuario jugador y usuario master
Precondiciones: Usuario existente
Flujo: 1. Hacer click en el botón "Entrar". 2. Escribir nombre de usuario y contraseña. 3. Hacer click en el botón "Login". 4. Si los datos son correctos se muestra la pantalla de selección de partida.
Flujo alternativo: 4. A El sistema indica que el nombre de usuario o contraseña son inválidos Si los datos son incorrectos permite volver a introducirlos, volviendo al punto 2.
Postcondiciones: Usuario identificado

Nombre: Crear usuario
Descripción: Permite crear un nuevo usuario que no existe en el sistema.
Actores: Usuario jugador y usuario master
Precondiciones: Ninguna
Flujo: 1. Hacer click en el botón "Registrarse". 2. Rellenar los datos para el usuario. 3. Hacer click en el botón "Crear". 4. Si los datos son correctos se muestra la pantalla de selección de partida.
Flujo alternativo: 4. A El sistema indica que alguno de los datos es incorrecto o que el usuario ya existe en la base de datos, tras ello permite volver a introducir los datos, volviendo al punto 2.
Postcondiciones: Usuario identificado, Usuario existente

Nombre: Crear partida
Descripción: Permite crear una nueva partida dentro del sistema YR2.
Actores: Usuario master
Precondiciones: Usuario existente, Usuario identificado
Flujo: 1. Hacer click en el botón "Crear una nueva partida". 2. Escribir el nombre de la partida. 3. Hacer click en el botón "Crear". 4. Si la partida no existe en la base de datos, muestra un mensaje de confirmación.
Flujo alternativo: 4. A El sistema indica que la partida ya existe o el nombre es inválido y permite introducir el nombre de nuevo, volviendo al paso 2.
Postcondiciones: Partida existente

Nombre: Entrar partida
Descripción: Entra a una partida previamente existente.
Actores: Usuario jugador y usuario master
Precondiciones: Partida existente, Usuario existente, Usuario identificado
Flujo: 1. Seleccionar una partida desde el desplegable que muestra las diferentes partidas 2. Hacer click en el botón "Acceder".
Flujo alternativo:
Postcondiciones: Partida cargada

5.3 Acciones dentro del sistema YR2 parte 2.

Nombre: Borrar evento
Descripción: Permite cambiar la forma de afectar a la partida por parte del master
Actores: Usuario master
Precondiciones: Usuario existente, Usuario identificado, Partida existente, Partida cargada, Edición borrar eventos
Flujo: 1. Hacer click izquierdo en el punto del mapa donde se encuentre el evento que deseas borrar".
Flujo alternativo:
Postcondiciones:

Nombre: Crear cambiar elemento mapa
Descripción: Permite modificar directamente un evento dentro del mundo del juego y sus comportamientos.
Actores: Usuario master
Precondiciones: Usuario existente, Usuario identificado, Partida existente, Partida cargada, Edición mapeado
Flujo: 1. Seleccionar el tipo de elemento a crear mediante la rueda del ratón. 2. Hacer click en un punto vacío del mapa para crear un nuevo elemento del tipo seleccionado o en un punto donde exista un elemento para modificarlo.
Flujo alternativo:
Postcondiciones:

5.4 Acciones dentro del sistema YR2 parte 3.

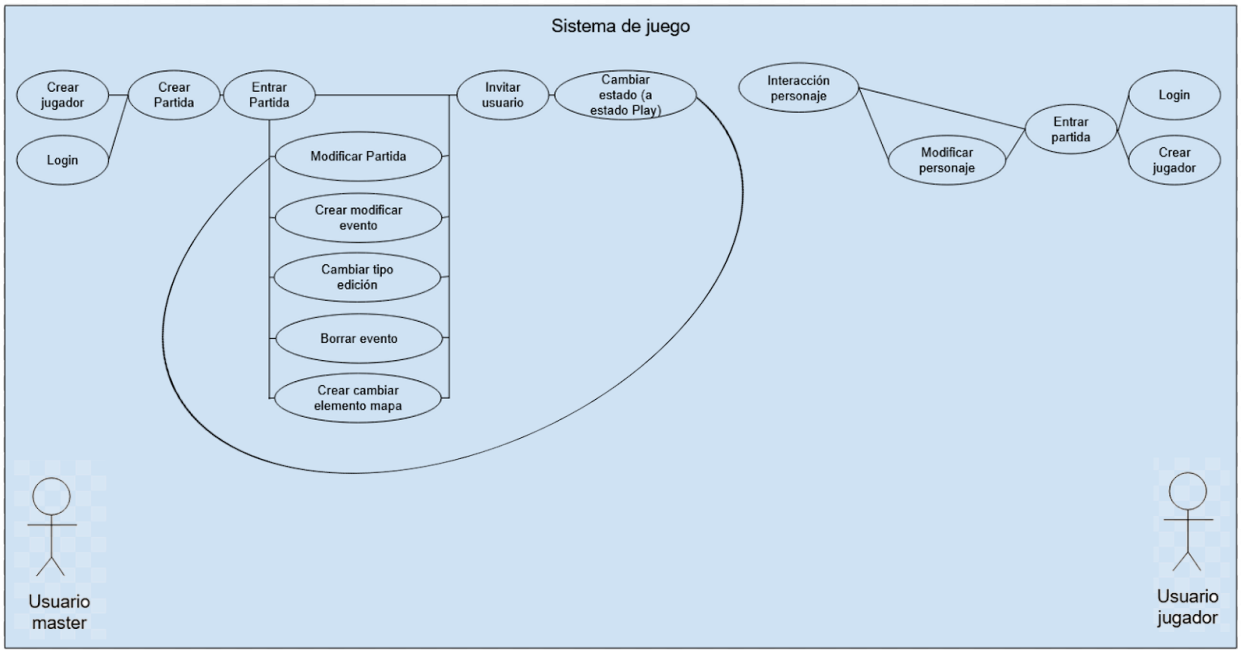
Nombre: Modificar personaje
Descripción: Permite cambiar el aspecto de un personaje
Actores: Usuario jugador
Precondiciones: Usuario existente, Usuario identificado, Partida existente, Partida cargada
Flujo: 1. Hacer click en el botón "Editar". 2. Utilizar las flechas y lo selectores de color para cambiar el aspecto del personaje del jugador. 3. Hacer click en el botón "Aceptar".
Flujo alternativo:
Postcondiciones:

Nombre: Interacción personaje
Descripción: Permite interactuar con el mundo del juego.
Actores: Usuario jugador
Precondiciones: Usuario existente, Usuario identificado, Partida existente, Partida cargada, Partida en play
Flujo: 1. Moverse con las teclas "AWSD". 2. Controlar la cámara con el movimiento del ratón. 3. Seleccionar el tipo de interacción con la rueda del ratón 4. Interactuar con el click izquierdo del ratón.
Flujo alternativo:
Postcondiciones: Evento en play (dependiendo de las condiciones de la interacción)

Nombre: Cambiar estado
Descripción: Permite cambiar el estado en el que se encuentra la partida
Actores: Usuario master
Precondiciones: Usuario existente, Usuario identificado, Partida existente, Partida cargada
Flujo: 1. Hacer click en alguno de los botones de estado de la parte superior izquierda de la pantalla botón "Play", "Stop", "Pause".
Flujo alternativo:
Postcondiciones: Partida en play(Botón play) / Partida en pause(Botón pause) / Partida en stop (Botón stop)

Nombre: Modificar partida
Descripción: Permite modificar directamente el mundo del juego.
Actores: Usuario master
Precondiciones: Usuario existente, Usuario identificado, Partida existente, Partida cargada
Flujo: 1. Hacer click derecho con el ratón en el cualquier punto fuera del GUI. 2. Seleccionar una de las opciones para modificar el mundo entre "Variables", "cambiar atributo", "Generar aleatorio", "Instanciar evento", "Efecto climático", "Crear mapa", "Encuesta", "Teletransporte", "Aplicar fuerza", "FX sonido", "Efectos especiales" o "Video / Imagen". 3. Rellenar los datos del formulario y pulsar en "Aceptar".
Flujo alternativo:
Postcondiciones:

5.5 Acciones dentro del sistema YR2 parte 4.



5.6 Acciones en una partida de YR2.

Capítulo 6 - Estudio de costes y posible modelo de negocio

En este capítulo se trata acerca de los costes que supondría un servicio como YR2 en un escenario de explotación real y se propone un posible modelo de negocio para hacerlo rentable.

6.1 Estudio de costes

Los costes medios de cada función en tiempo son los siguientes:

actualizaPartida	100 ms
logicaJuego	100 ms
login	100 ms
cargaPartida	1.000 ms
play	1.000 ms
guardarPartida	1.000 ms
creaPartida	1.000 ms
cambio	100 ms

6.1.1 Costos en tiempo de las funciones de YR2.

Para jugar una partida es necesario mínimo que se realicen las llamadas detalladas a continuación, estos cálculos se realizan suponiendo una partida de 4

jugadores y 1 master, existiendo grupos de rol más grandes y más pequeños parece un compromiso aceptable:

- Coste por partida
    - 4 jugadores y 1 master son 5 llamadas a actualizaPartida cada segundo
    - 1 Llamada a logicaJuego cada segundo
    - 5 Llamadas a cambio cada segundo aproximadamente, suponiendo que cada jugador actúa una vez cada segundo.
    - 5 llamadas a login al inicio de la partida
    - 5 llamada a cargaPartida al inicio de la partida
    - 1 llamada a play al inicio de la partida
    - El master debe preparar la partida, lo que conlleva otros gastos
        - 1 llamada al login
        - 1 llamada a guardarPartida
        - 1 llamada a cargaPartida o creaPartida

Teniendo esto en cuenta, el coste medio de una partida suponiendo que se trata de una sesión de unas 4 horas sería:

Llamadas = Total de llamas que se realizan en 4 horas = 3600 * 4
100 ms * 5 * Llamadas + 100 ms * Llamadas + 100 ms * 5 + 1.000 ms * 5 + 1.000 ms + 100 ms * 5 * llamadas = 15.846.500 ms → Coste de la partida
100 ms + 1.000 ms + 1.000 ms = 2.100 ms → Coste de preparación de la partida
2.100 ms + 721.446.500 ms = 721.448.600 ms → Coste por partida

6.1.2 Cálculo del tiempo y costes de una partida promedio.

Si el coste por cada 100 ms de procesamiento es 0,0000002083$ (ya que cada función tiene asignados 128 MB de memoria), nos da un coste de 3,3 dólares por cada partida jugada.

Como hemos calculado los costes de esta partida con una base de 5 jugadores, nos sale como gasto necesario por cada usuario de la plataforma para que esta sea viable 0,66$ por partida. Aún teniendo en cuenta que estos cálculos son el mínimo de llamadas necesario para una partida, si doblamos el coste aún nos da un coste de 6,6$ por partida en total o 1,32$ por cada jugador.

Para poder hacer frente a estos gastos lo adecuado sería implementar una tienda en la que ofrecer partidas ya preparadas y material específico para las partidas, para que estas puedan ser más personalizadas que si se realizaran con el material por defecto. Incluso sería recomendable permitir crear sus propias tiendas dentro de la plataforma a los usuarios, para que vendan su propio material a cambio de una comisión del 30%. Todo esto estaría dentro del apartado conclusiones y trabajo futuro.

También habría que tener en cuenta el coste de la base de datos en AWS S3, siendo este de 0,024$ por GB al mes, al estar guardando nada más que datos en texto, podemos suponer que 50 GB son más que suficientes para todo el sistema significando un gasto de 1,2$ al mes, lo cual es casi despreciable.

Por último habría que sumar a esto el coste de las llamadas a la API aparte del coste de la función lambda en sí misma, siendo este coste de 1,17$ al mes en caso de no superar los 300 millones de llamadas a la API y de 1,05$ en caso de superar dicho límite.

Todos estos precios se han consultado en las páginas de AWS Lambda[9], AWS S3[10] y AWS API Gateway[11].

6.2 Modelo de negocio

Como modelo de negocio surgen 4 alternativas de monetización, siendo la primera de ellas la de generación de recursos para YR2 más allá de los recursos base que poder vender en la tienda del juego, otra la colaboración con empresas externas que quieran traer sus ambientaciones a YR2, otra la venta de recursos creados por terceros específicamente para YR2 y por último también la venta del juego en tiendas digitales como Steam o similares.

Las 3 primeras líneas de negocio estarían condicionadas por la inclusión de una tienda dentro de la aplicación, o en alguna plataforma asociada, preferiblemente una página web a la que acceder mediante la misma cuenta que al juego (esto entraría dentro del apartado de trabajo futuro).

En esta tienda los usuarios podrían crear su propia tienda donde poner a la venta sus propios recursos creados para ambientar partidas, o partidas ya preparadas para ser jugadas, siendo esta la primera línea de negocio, de forma que otros usuarios puedan jugar directamente a YR2 sin necesidad de haber preparado alguna partida con anterioridad. A cada usuario se le cobraría una comisión del 30% por cada venta. Para el mantenimiento de esta línea de negocio es importante la creación de una comunidad y una herramienta para ello sería la creación de un foro en una página web de YR2, donde la gente pueda compartir sus dudas consejos y apoyarse entre ellos.

Otra línea de negocio sería llegar a acuerdos con otras empresas dedicadas a juegos de rol, ya sean juegos de mesa o videojuegos propiamente dichos para llevar sus ambientaciones y mundos a YR2. Mediante esos acuerdos se realizarían escenarios y partidas específicas para dichas empresas a cambio de una cantidad pactada. Para ello es necesario la existencia de un equipo dedicado específicamente a ello

con al menos 1 grafista, 1 persona ocupada del sonido y 1 persona ocupada del guión.

Una tercera línea de negocio sería la creación de recursos y partidas para YR2 aparte de los que vendrían por defecto en el juego. Estos recursos se venderían en la misma tienda que los demás recursos creados por usuarios de YR2 y tendrían un equipo dedicado a su creación de manera específica, aunque ese no sería su único cometido, ya que deberían ocuparse también del desarrollo de material propio de YR2 como se menciona en la línea de negocio explicada más arriba.

Independientemente de las vías antes mencionadas para obtener beneficios, vendiendo el juego a 6€ en steam se cubren los gastos para al menos 7 partidas en las que los jugadores no realicen ninguna compra (hay que tener en cuenta que Steam se queda con un 30% de cada venta que realiza, dejando estos 6€ en 4,2€, que se traducen en 5,14$). Aunque un precio más razonable, en vista de los precios manejados en tiendas digitales de videojuegos sería de 17€, lo que se quedaría en 12€ (14,69$ y 22 partidas) después de quitar el porcentaje para el marketplace. El motivo de vender el producto a un precio mayor viene dado al efecto que pueda tener vender a un precio demasiado bajo, pudiendo ocasionar que se perciba una falta de fe el el producto en sí y generando de este modo una mala impresión del producto mismo. Para las conversiones de moneda se ha utilizado la herramienta proporcionada por Google a través de su propio buscador[12].

Para mantener estas líneas de negocio sería necesario un equipo de 5 personas. De ellas, 3 dedicadas a la creación de nuevo contenido, tanto original como basado en IPs ajenas, otra persona para ocuparse del mantenimiento y desarrollo de YR2 y sus posibles ampliaciones de cara a futuro. Por último sería necesaria otra persona para ocuparse de las negociaciones con otras empresas para poder cerrar acuerdos con ellas y poder utilizar IPs ajenas dentro de la plataforma YR2 con las que llamar la

atención de nuevos jugadores que se unan a la plataforma provenientes de las IPs mencionadas.

En cuanto a cómo llegar a los usuarios, teniendo en cuenta la media de edad del público al que se quiere llegar, lo más adecuado sería la publicidad en plataformas como facebook y enviar notas de prensa a medios especializados con cada hito alcanzado (acuerdos importantes, etc...). En caso de no conseguir aparecer de manera orgánica en dichos medios existe la opción de pagar artículos patrocinados.

Sobre el coste de YR2, se habló en el pasado apartado 6.1 del coste del servicio en sí, pero para realizar un modelo de negocio habría que añadir el coste de los empleados y el coste tener y mantener una página web (unos 100€ anuales) donde alojar información de YR2, ofrecer tal vez la posibilidad de realizar las compras para la plataforma desde la propia página web y tener unos foros para el mantenimiento de la comunidad.

En la siguiente tabla 6.2.1 se detallan los costes de personal.

Coste de personal:

Grafista / 3d	20.000€ anuales
Técnico de sonido / músico	25.000€ anuales
Guionista	20.000€ anuales
Desarrollador	30.000€ anuales
Comercial	A comisión

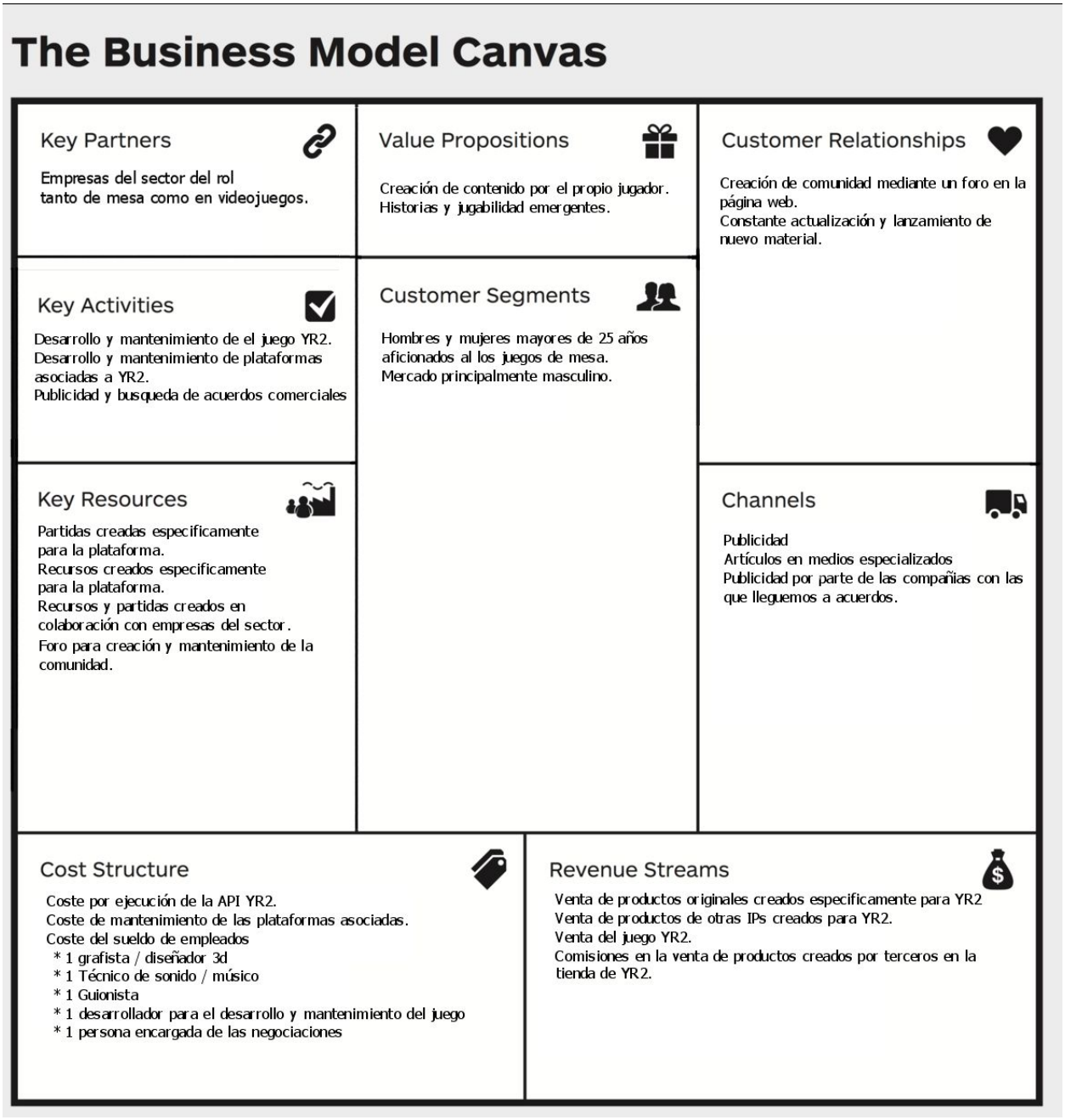
6.2.1 Costes de personal.

Todo sumado nos da un coste de aproximadamente 95.100€ anuales en infraestructura. Ya que el uso de los servicios de YR2 por parte de los usuarios no supone que este suponga un coste hasta la partida número 22 o hasta las 88 horas de uso, límite que no se espera que superen los jugadores menos propensos a realizar compras en la tienda de YR2.

Teniendo todo esto en cuenta sería necesario tener una base de 8.000 nuevos jugadores o compras por un valor equivalente dentro de la tienda del juego para ser rentable cada año.

Por otro lado, si bien es cierto que los servicios de AWS son baratos y permiten prestar el servicio manteniendo un margen de beneficio, depender de Amazon como proveedor de estos servicios únicamente es una fuente posible de amenazas, al existir la posibilidad de que cambie las condiciones del servicio en cualquier momento, motivo por el que la base de datos se ha mantenido en texto plano, por si pudiera darse el caso de tener que migrar el servicio a servidores propios o de otra compañía más adelante.

A continuación se puede ver una imagen que resume en sí misma el modelo de negocio propuesto en la figura 6.2.1.



6.2.1 Business model canvas o modelo de negocio.

Capítulo 7 - Conclusiones y trabajo futuro

Podemos concluir, en vista a lo expuesto anteriormente, que el modelo serverless es viable para la arquitectura de red de un videojuego. Con servicios como AWS incluso da igual el tamaño del juego y la cantidad de peticiones que se vayan a recibir, ya que en caso de que pudiera ser un problema bastaría con aumentar la memoria asignada a las funciones de la API para que puedan soportar mayor carga de trabajo.

Por otro lado, en cuanto a la seguridad, esta alternativa ofrece la certeza de la estabilidad de los servidores frente a un modelo cliente-servidor en el que el master debería realizar las labores de servidor, ya que no se depende de su conexión a internet y se evita así un posible cuello de botella, sacrificando eso sí, las ventajas económicas que ofrece el modelo cliente-servidor clásico en el que no habría coste alguno por el servicio, ya que la comunicación durante las partidas sería principalmente entre los propios jugadores.

Como punto negativo, esta alternativa nos obliga a depender de un agente externo que no podemos controlar. Amazon, como prácticamente cualquier empresa que ofrece estos servicios se reserva el derecho a cambiar las condiciones de su servicio y podría decidir cambiar sus tarifas cualquier día, haciendo que los cálculos realizados en base a estas tarifas no sirvan para nada. Si los servidores de Amazon se caen no se podría ofrecer servicio tampoco, si bien es cierto que esto no es algo que suceda muy a menudo, es una posibilidad real que hay que tener en cuenta. Es por ello que la base de datos se encuentra guardada en texto plano, en lugar de utilizar alguno de los servicios de bases de datos específicos que ofrece la plataforma, ya que estos exigen el uso de un lenguaje específico propio para ellas y sería más difícil de migrar en un futuro si fuera necesario.

Como posibles mejoras y ampliaciones al trabajo realizado lo dividiremos en 5 puntos: la inclusión de un chat de voz, la comprobación por parte de los clientes de las órdenes que reciben para evitar ejecutar órdenes repetidas, comprobación en las funciones de la API de los mensajes recibidos para evitar enviar órdenes innecesarias o contradictorias a los clientes, realizar test de estrés de las diferentes funciones de la API para ver cómo se comportan bajo condiciones no idóneas, la creación de una tienda in-game de productos donde los propios usuarios puedan subir sus productos digitales para YR2 y el desarrollo de una página web con un foro utilizando los mismos usuarios de la base de datos del juego.

Para el chat de voz se utilizaría el servicio Vivox[13], ya que tiene un SDK para Unity. Debido a las características del proyecto actual no se ha planteado y tanto el chat como su historial están guardados en la base de datos, pero el uso de este servicio permitiría plantearse cambiar el chat por escrito para usar este sistema. En cuanto al precio del servicio es gratuito para desarrolladores independientes hasta los 5.000 usuarios simultáneos, pero no hay unas tarifas públicas y habría que llegar a un acuerdo específico con Vivox en caso de superar dicho límite.

La comprobación de los mensajes por parte de la API permitiría hacer más estable y predecibles los comportamientos del juego, ya que uno de los problemas específicos del juego en red es precisamente mantener la coherencia entre lo que pasa en los diferentes clientes que ejecutan el juego, ya que las órdenes a ejecutar pueden no llegar en el mismo orden en el que se han producido. La red es impredecible, pero de esta manera se puede controlar un poco y hacer más coherente entre sí el comportamiento de los diferentes clientes. La idea es comprobar cuando se envíen órdenes a la API las ordenes que ya tiene guardadas para enviar a los diferentes clientes y borrar las que no sean necesarias o sean contradictorias entre sí, para esto ya se envía un timestamp en estos mensajes a la API, aunque por el momento es una comprobación que no se realiza.

El cliente podría guardar las órdenes recibidas para su ejecución en base a su timestamp. De esta manera cuando le llegue una nueva orden debería comprobar su timestamp y si existe una orden con ese timestamp comprobar si la orden es la misma para evitar ejecutar varias veces una misma orden, ya que esto podría provocar comportamientos inesperados.

También sería importante comprobar el comportamiento de YR2 bajo situaciones de estrés, para detectar vulnerabilidades o problemas de rendimiento y poder actuar en consecuencia. A este fin existen servicios como Chaosmonkey[14] de Netflix.

Como parte del modelo de negocio se ha hablado de la creación de una tienda digital para de YR2 en la que poner a la venta tanto productos "oficiales" de YR2, como productos creados por los usuarios (recursos gráficos o sonoros, o incluso partidas ya preparadas), por lo tanto una ampliación necesaria sería el desarrollo de esta tienda utilizando alguna pasarela de pago como Ayden[15] o Stripe[16] o utilizando un servicio como Woocommerce (suponiendo en este caso que la tienda queda completamente fuera del juego, y las compras se realizarían en una página web de YR2). Dicha tienda deberá permitir la creación de tiendas dentro de ella por parte de los usuarios, probablemente gestionadas desde una página web.

Por último, sería necesario para la puesta en práctica real de todo lo mencionado anteriormente la creación de una página web con un foro y una tienda como se menciona en el párrafo anterior. Para ello la opción más conveniente es utilizar Wordpress[17]. La creación de dicha página sería un tema sencillo, siendo la única complicación el poder utilizar dentro del juego los usuarios creados a través de la página web y viceversa.

Chapter - Introduction

Tabletop role-playing can be quite difficult for a number of reasons. On many occasions it is difficult to coordinate so many people to be able to play a game (taking into account that it takes a minimum of 3 people to be able to play, being the recommended between 4 and 6), other times there is no suitable place where to meet to play and there are also barriers on other occasions such as the difficulty of certain game systems that may require several rolls with different types of dice (4, 6, 8, 10 or 20 faces) to sometimes solve something as simple as a simple blow.

YouRuleYouRole aims to overcome these limitations by providing a virtual space in which to meet to celebrate role-playing games through a simple system of rules, providing the figure of the master with a pseudo video game engine where to develop their games without needing to know how to program and where players can intuitively see the effects of their actions in the game without the need for long process of spins and checking tables.

In this way a group of friends can get together to play a game without having to neglect other obligations, as long as they can have access to a computer with an internet connection.

Chapter - Motivation

Tabletop role-playing can be quite difficult for a number of reasons. On many occasions it is difficult to coordinate so many people to be able to play a game (taking into account that it takes a minimum of 3 people to be able to play, being the recommended between 4 and 6), other times there is no suitable place where to meet to play and there are also barriers on other occasions such as the difficulty of certain game systems that may require several rolls with different types of dice (4, 6, 8, 10 or 20 faces) to sometimes solve something as simple as a simple blow.

As has been mentioned, managing to gather the amount of people necessary to hold a table role session, usually between 4 and 8 people (although the minimum necessary would be 3), it can be difficult at times. These complications are greater as we get older and acquire commitments that we lack when we are younger.

In this way a group of friends can get together to play a game without having to neglect other obligations, as long as they can have access to a computer with an internet connection.

Chapter - Objectives

YR2 aims to facilitate this kind of meetings by taking them to the virtual environment, however, doing just that would not be anything new, since there are already platforms like Roll20 that offer this type of services.

The idea is to improve this kind of experience by also providing a personalized 3D environment for the players, where their adventures will take place. This will be possible through a simple editor to which the master will have access. The novelty is that the figure of the master will also have a series of tools to automate certain events that will take place during the game, as well as the possibility of directly influencing it in real time, providing players with a completely organic and different experience each time , in which the world responds to their actions and they see this on their own computer screen.

Chapter - Work plan

For the work plan below, the work to be done was divided into tasks and each task had been given a score from 0 to 3, with 0 being that the task is completed in a few minutes and 3 that the task can take up to 10 - 12 hours. If a task turned out to be too long, it has been divided into subtasks.

Work area	Task	Subtask	Time	Done
Net architecture	Database	Create database structure and develop it taking into account its modifications through the API	2	X
	Lambda functions and API	Approach to the operation of the API and the calls it will receive from clients	2	X
		Function creaUsuario	1	X
		Function creaPartida	1	X
		Function cargaPartida	2	X
		Function guardaPartida	2	X
		Function actualizaPartida	2	X
		Function play	1	X
		Function login	1	X
		Function logicaJuego	2	X
		Function cambio	2	X
Client	Calls to the API	Implementation of API calls from the client	1	X
	Client logic	Implementation of the network system to send requests to the API about the state of the game and respond to changes in the state of the game	2	X
		Create the event editor	3	X
		Creation of the character editor	2	X
		Main menu	3	X
		Master UI and Menu Creation	3	X

		Player UI and Menu Creation	2	X
	Audiovisual part of the application	Implementation of graphic effects, particle effects and sound effects	2	X
Report writing		Development of the user manual (Appendix)	3	X
		Case of use	2	X
		Cost study	3	X
		Report writing	3	X
Test		API test	2	
		Event editor test	2	X
		Client test and interaction between systems test	2	X
Fixes and extensions		Fixes 1	3	X
		Fixes 2	3	X
		Writing of possible extensions	1	X
		Upload project to a mercurial repository	1	X

- 8.3.1 Task division

Chapter - Conclusions and future work

We can conclude, in view of the above, that the serverless model is viable for the network architecture of a video game. With services like AWS, it doesn't even matter the size of the game and the number of requests that will be received, since in case it could be a problem, it would be enough to increase the memory allocated to the API functions so that they can bear a greater load. of work.

On the other hand, in terms of security, this alternative offers the certainty of the stability of the servers compared to a client-server model in which the master should perform the server tasks, since it does not depend on its internet connection and A possible bottleneck is thus avoided, sacrificing the economic advantages offered by the fact that the main cost of the serverless architecture, the main cost of which comes from the API calls during the game itself.

As a negative point, this alternative forces us to depend on an external agent that we cannot control. Amazon, like practically any company that offers these services, reserves the right to change the conditions of its service and could decide to change its rates on any day, making the calculations made based on these rates useless. If Amazon's servers go down, no service could be offered either, although it is true that this is not something that happens very often, it is a real possibility that must be taken into account. That is why the database is stored in plain text, instead of using any of the specific database services offered by the platform, since these require the use of a specific language of their own and it would be more difficult to migrate it in the future if necessary.

As possible improvements and extensions to the work carried out, we will divide it into 5 points: the inclusion of a voice chat, the verification by the clients of the orders they receive to avoid executing repeated orders, verification in the API functions of the

messages received to avoid sending unnecessary or contradictory orders to customers, perform stress tests of the different API functions to see how they behave under unsuitable conditions, the creation of an in-game store of products where users themselves can upload their digital products for YR2 and the development of a website with a forum using the same users from the game database.

For voice chat, the Vivox service that has an SDK for unity would be used. Due to the characteristics of the current project, it has not been considered and both the chat and its history are stored in the database, but the use of this service would allow us to consider changing the chat in writing to use this system. As for the price of the service, it is free for independent developers up to 5,000 simultaneous users, but there are no public rates and a specific agreement with Vivox would have to be reached in case of exceeding this limit.

The verification of the messages by the API would make the behavior of the game more stable and predictable, since one of the specific problems of network gaming is precisely maintaining coherence between what happens in the different clients that run the game, since the orders to be executed may not arrive in the same order in which they were produced. The network is unpredictable, but this way you can control a bit and make the behavior of different clients more consistent with each other. The idea is to check when orders are sent to the API the orders that you already have saved to send to the different clients and delete those that are not necessary or are contradictory to each other, for this a timestamp is already sent in these messages to the API, although at the moment it is a check that is not carried out.

The client could save the orders received for execution based on their timestamp. In this way, when a new order arrives, you should check its timestamp and if there is an order with that timestamp, check if the order is the same to avoid executing the same order several times, as this could cause unexpected behavior.

It would also be important to check the behavior of YR2 under stress situations, to detect vulnerabilities or performance problems and to be able to act accordingly. To this end, there are services such as Netflix's Chaosmonkey.

As part of the business model, there has been talk of the creation of a digital store for YR2 in which to sell both "official" YR2 products, as well as products created by users (graphic or sound resources, or even games already prepared), therefore a necessary extension would be the development of this store using a payment gateway such as Ayden or Stripe or using a service such as Woocommerce (assuming in this case that the store is completely out of the game, and purchases would be made on a YR2 website). This store must allow the creation of stores within it by users, probably managed from a web page.

Finally, it would be necessary for the real implementation of everything mentioned above to create a website with a forum and a store as mentioned in the previous paragraph. For this the most convenient option is to use Wordpress. The creation of said page would be a simple matter, the only complication being to be able to use the users created through the web page within the game and that that works the other way around too.

Bibliografía

[1] https://roll20.net/ [Último acceso: 21 01 2021].

[2] https://www.rpgmakerweb.com/ [Último acceso: 21 01 2021].

[3] https://en.wikipedia.org/wiki/Neverwinter_Nights_(2002_video_game) [Último acceso 21 01 2021].

[4] https://web.archive.org/web/20201102093103/https://kinsta.com/es/blog/google-cloud-vs-aws/ [Último acceso: 20 01 2021, versión en caché].

[5] https://www.infoworld.com/article/3561269/the-2020-idg-cloud-computing-survey.html [Último acceso: 20 01 2021].

[6] https://aws.amazon.com/es/lambda/ [Último acceso 21 01 2021].

[7] https://aws.amazon.com/es/api-gateway/ [Último acceso 21 01 2021].

[8] https://aws.amazon.com/es/s3/ [Último acceso 21 01 2021].

[9] https://aws.amazon.com/es/lambda/pricing/ [Último acceso: 20 12 2020].

[10] https://aws.amazon.com/es/s3/pricing/?nc=sn&loc=4 [Último acceso: 20 01 2021].

[11] https://aws.amazon.com/es/api-gateway/pricing/ [Último acceso: 20 01 2021].

[12] Datos de equivalencia de monedas de Morningstar [Último acceso: 28 12 2020, consultado a través de Google].

[13] https://unity.com/products/vivox/ [Último acceso: 30 12 2020].

[14] https://netflix.github.io/chaosmonkey/ [Último acceso: 30 12 2020].

[15] https://www.adyen.com/es_ES/ [Último acceso: 30 12 2020].

[16] https://stripe.com/es [Último acceso: 30 12 2020].

[17] https://wordpress.com/es/ [Último acceso: 30 12 2020].

Anexos

A - Anexo: Manual de usuario

Manual de usuario de YR2

Acceder a una partida

Para acceder a una partida de YR2 tienes que estar logueado en el sistema, para lo cual debe existir un usuario válido en la base de datos. Si es la primera vez que usas YR2 es posible que no tengas un usuario en la base de datos, en cuyo caso deberás crear un usuario primero. En el caso de que tengas un usuario de YR2, para acceder a una partida bastará con hacer login como se especifica más adelante. En cualquier caso, nada más iniciar YR2 aparecerá una pantalla como la siguiente.

Figura A.1. Pantalla principal de YR2.

No tengo usuario de YR2

En caso de no tener un usuario puedes crearlo Desde el menú principal de YR2 pulsando sobre el botón "Registrarse". Al hacerlo aparecerá un formulario como el de la figura A.2 y una vez hayas rellenado los campos necesarios para el registro que aparecerán tras hacer click en el botón tan solo deberás hacer click en el botón "Crear" y el usuario se añadirá a la base de datos de YR2, iniciando sesión en el proceso.

Figura A.2. Formulario de creación de usuario de YR2.

Ya tengo un usuario de YR2

En este para identificarte como usuario y acceder a tus partidas debes pulsar el botón "Entrar" desde el menú principal de la aplicación para acceder al menú de Login (figura A.3). Acto seguido se te preguntará por el nombre de usuario único y tu contraseña. Rellena estos campos y haz click en el botón "Login" para entrar al sistema y ver tus partidas.

Figura A.3. Menú de Login de YR2.

Una vez identificado en el sistema, tendrás la opción de unirte a una partida existente o crear una nueva partida a través de una interfaz como la de la figura A.4.

Figura A.4. Menú de partidas de YR2.

Unirte a una partida existente

Para unirte a una partida existente debes elegir la partida a la que quieres unirte desde el desplegable que aparece en la pantalla de selección de partidas. Al seleccionar una partida, la ficha situada debajo del desplegable mostrará el rol que tienes en esa partida (máster o jugador). Una vez hayas elegido la partida que quieres abrir, basta con hacer click en el botón "Acceder" para acceder a esa partida.

Crear una nueva partida

Para crear una nueva partida debes hacer click en el botón "Crear una nueva partida" que aparece en el selector de partidas. Una vez hagas click en dicho botón, podrás rellenar unos campos con la información básica de la partida. Rellena estos campos y haz click en el botón "Crear". Una nueva partida aparecerá en el desplegable del selector de partidas en la que tendrás el rol de máster al ser tú el creador de dicha partida.

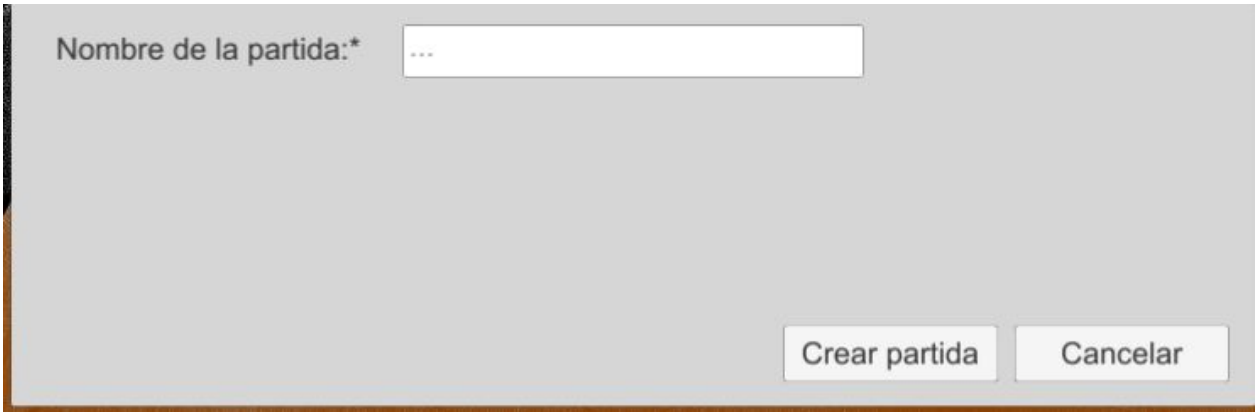


Figura A.5. Formulario de creación de partida de YR2.

Opciones de máster

Al iniciar una partida como máster por primera vez lo que verás será un mapa vacio. Cómo máster las opciones para modificar este y las partidas son variadas.

Editor de mapas

Movimiento:

Desde una vista en primera persona el master puede moverse en las 3 direcciones del espacio utilizando las teclas WASD para moverse en el eje izquierda/ derecha y adelante/ atrás. El movimiento en la dimensión correspondiente a la altura dependerá de lo que enfoque la cámara del máster. Es decir que si miras al sol y pulsas la tecla W, irás directo hacia él.

Crear o modificar eventos:

Desde el editor de mapas puedes crear y borrar objetos de evento que luego podrás programar. Para ello debes asegurarte de estar en el modo de edición de eventos (activado por defecto). Con este modo activado se mostrará un cubo transparente frente al master en todo momento, en la posición en la que se crearía un evento al hacer click izquierdo.

Al hacer click izquierdo en este modo, se creará un evento en ese punto si la zona del mapa que ocupa el cubo transparente no tiene ningún evento. Si existiera un evento en esa posición se modificaría el evento existente. En cualquier caso se abriría la ventana del editor de eventos.

Borrar eventos:

Para acceder a esta opción debes cambiar el modo de edición al modo borrar eventos, cosa que se puede hacer haciendo click derecho en cualquier parte del mapa. Se te mostrará un menú contextual en el que entre otras opciones existe la de borrar eventos. Haz click en esa opción y habrás cambiado el modo de edición. Acto seguido muévete por el mapa de forma

que el cubo transparente en frente de la cámara del master ocupe la posición de evento que quieres borrar y haz click izquierdo para borrar dicho evento.

Crear o modificar elementos del mapeado:

Para acceder a esta opción debes cambiar el modo de edición al modo de edición de elementos. Para cambiar a este modo haz click con el botón derecho del ratón en cualquier parte del mapa y elige la opción elementos de mapeado. Una vez en este modo puedes seleccionar el elemento que quieres crear o modificar con la rueda del ratón. Para crear o modificar un elemento, haz clic con el botón izquierdo del ratón y el elemento se creará en el lugar que ocupe el elemento transparente frente a la cámara de máster.

Editor de eventos

Esta pantalla está compuesta por varios elementos que conviene explicar por separado. Su aspecto general se puede ver en la figura A.6.

- Tipo de evento: Consta de un desplegable y un botón de editar situados en la parte superior izquierda de la ventana de edición de evento y permite cambiar el aspecto gráfico del evento. Si el evento es del tipo personaje no jugador el botón editar mostrará el editor de personajes, cuyo uso se definirá más adelante, en cualquier otro caso mostrará una ventana con la opción de elegir entre los elementos de mapeado disponibles.
- Tipo de ejecución: Consta de un desplegable debajo de tipo de evento que permite seleccionar la política de ejecución del evento entre 3. Ejecución automática (se ejecutará automáticamente cuando un jugador entre en el mapa en el que se encuentra el evento), ejecución por trigger (se ejecutará cuando un jugador ocupe su espacio físico) o

ejecución por activación del jugador (se ejecutará cuando un jugador interactúe activamente con el evento).

- Pestañas del editor: Situadas en la parte superior de la ventana del editor, permiten seleccionar y mostrar los eventos que se ejecutarán como respuesta a las interacciones de los jugadores con un evento. En caso de que el evento no esté configurado con el tipo de ejecución por activación del jugador los comandos que ejecutará serán los de la ventana por defecto.
- Ventana de comandos: En la parte derecha de la ventana de edición de eventos aparece una ventana que estará en blanco hasta que la llenes de comandos a ejecutar con los botones que aparecen debajo de esta. En esta ventana podemos movernos de posición haciendo clic sobre los comandos para cambiar la posición en la que se añaden o se borran comandos y añadir o borrar comandos con los botones inferiores.

Figura A.6. Editor de eventos de YR2.

Comandos directos

- Instanciar Evento: Crea un evento previamente definido en la posición indicada.
- Variables: Crea una variable o cambia el valor de una existente.
- Cambiar Atributo: Cambia el valor de un atributo de un personaje.
- Generar Aleatorio: Genera un número aleatorio que será guardado en una variable con el nombre RANDOM.
- Efecto Climático: Cambia los valores de clima y hora del mapa actual.
- Sonido / Música Ambiente: Cambia las pistas que está reproduciendo actualmente de música o efectos sonoros de ambiente.
- Crear / Cambiar Mapa: Crea un nuevo mapa y coloca al master en su posición 0,0,0, en caso de que el nombre indicado sea el de un mapa existente simplemente lleva al máster a dicho mapa.
- Encuesta: Muestra una encuesta a los jugadores, la opción votada por cada jugador se guardará en la variable SURVEY seguido del nombre del jugador.
- Teletransporte: Mueve un jugador a la posición indicada del mapa indicado.
- Fuerza: Aplica una fuerza sobre el evento o jugador indicado.
- FX Sonido: Reproduce un efecto de sonido.
- Efectos Especiales: Reproduce un efecto de partículas.
- Video /Imagen: Muestra un video o una imagen a los jugadores.

Opciones de jugador

Como jugador existen una serie de acciones básicas que puedes realizar dentro de YR2 que se pueden resumir en moverte por el mapa e interactuar con sus elementos.

Para moverse por el mapa se utiliza la vista en 1ª persona usando el ratón para enfocar la cámara y las teclas WASD para moverse.

Por otro lado, como jugador puedes utilizar la rueda del ratón para elegir el tipo de interacción que quieres llevar a cabo y el botón izquierdo del ratón para llevar a cabo dicha interacción.

También tienes la opción de escribir en el chat y editar el aspecto de tu personaje, pero al ser estas funciones comunes al jugador y al máster se definen en el siguiente apartado. En la siguiente imagen se puede ver el aspecto de la interfaz del jugador.

Figura A.7. Interfaz de jugador de YR2.

Opciones comunes al jugador y al master

Chat

En la parte inferior de la pantalla aparece una ventana de chat donde como jugador y como máster puedes escribir y leer los mensajes de los demás miembros de la partida.

Figura A.8. Panel del chat de YR2.

Editor de aspecto

Como jugador puedes editar tu aspecto pulsando sobre el botón editar en la parte superior derecha de la pantalla. Cómo máster por otro lado puedes editar el aspecto de ciertos eventos con el mismo haciendo click en el botón editar dentro de la sección Tipo de evento del editor de eventos.

Una vez abierto el editor te mostrará cuatro opciones (Cabeza, Cuerpo, Brazos y Piernas) y cada una de estas opciones tendrá dos botones para cambiar el aspecto de dicha extremidad y un selector de color Una vez estés satisfecho con el aspecto del personaje haz click en Aceptar, o haz click en Cancelar si no quieres que se guarden los cambios.

El aspecto del editor es el siguiente.

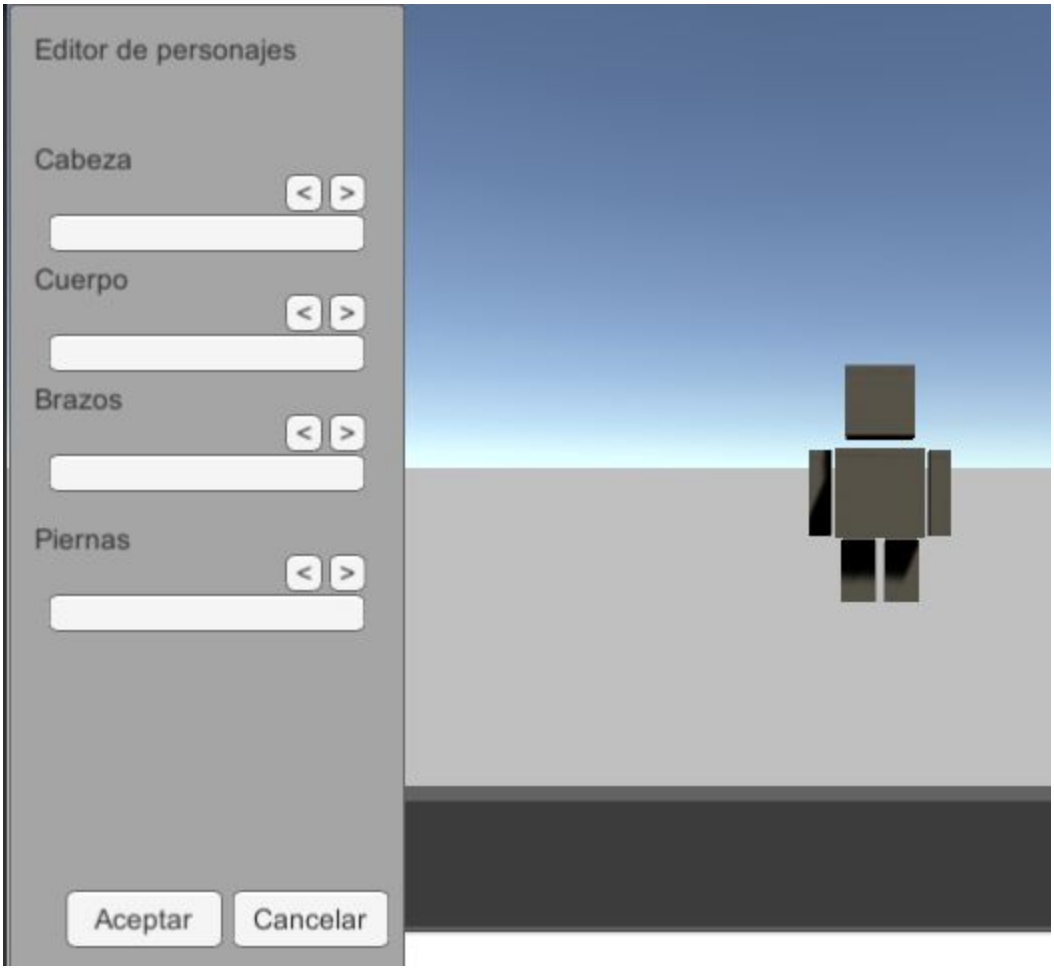


Figura A.9. Editor de aspecto de YR2.

Estado del juego

En la parte superior izquierda de la pantalla del master se muestran 4 opciones que le son exclusivas, que sirven para cambiar el estado actual de ejecución del juego. Estas opciones son Play, Stop, Pause y Guardar partida.

En estado Play los eventos se ejecutarán y se llamará cada segundo a la función de actualización de la partida por parte del máster, ocasionando que el estado de los objetos en esta cambie y se avise de estos cambios a los jugadores.

Durante el estado Pause dejarán de ejecutarse las funciones que aseguran la actualización de la lógica de la partida.

En estado Stop no se ejecutará ninguna función de actualización de la partida y ni el máster ni los jugadores serán conscientes de las acciones que realice cada uno dentro de esta.

La opción de Guardar partida guarda todos los datos de la partida en su estado actual en la base de datos.

Figura A.10. Panel del estado de partida de YR2.

B - Anexo: API de YR2

Cuando se hace un cambio por cualquier usuario de la partida se llama a la función que lo lleva a cabo y se registra en la carpeta "EnCurso" y en el fichero de cambios si procede.

Si el master guarda partida se guardan todos los datos en la carpeta base y en la carpeta "EnCurso", excepto la información de Info.txt (el chat, jugadores de la partida, si la partida está ejecutando o no y el id del máster).

Para cargar partida hay que mirar el archivo Info.txt, si la partida está en estado play coge los archivos de la carpeta "EnCurso" en caso contrario carga los datos de la carpeta base.

Funciones lambda

guardarPartida()

Recibe el id de partida, sobreescribe los archivos de la carpeta base y los de la carpeta "EnCurso" con la información que tenga el master en la partida actualmente.

```
exports.handler = async (event, context) => {

    const jsonPrev = JSON.parse((event.body));
    var statusCodeReturn = 200;
    var bodyReturn = "{}";
    var headerReturn = "";

    try {


      var info = JSON.stringify(jsonPrev['Info'], null, 4);
      var destparams = {
        Bucket: 'yr2-db',
        Key: jsonPrev['Partida'] + '/Info.txt',
```

```
            Body: info
        };
        await s3.putObject(destparams).promise();

        for (var i = 0; i < jsonPrev['Info']['jugadores'].length; i++)
        {
            info = JSON.stringify(jsonPrev['Jugadores'][jsonPrev['Info']['jugadores'][i]], null, 4);
            destparams = {
                Bucket: 'yr2-db',
                Key: jsonPrev['Partida'] + '/Jugadores/' + jsonPrev['Info']['jugadores'][i] + '.txt',
                Body: info
            };
            await s3.putObject(destparams).promise();
        }
        var listaMapas = {
            'ListaMapas': jsonPrev['ListaMapas']
        };
        info = JSON.stringify( listaMapas, null, 4);
        destparams = {
            Bucket: 'yr2-db',
            Key: jsonPrev['Partida'] + '/ListaMapas.txt',
            Body: info
        };
        await s3.putObject(destparams).promise();

        for (var i = 0; i < jsonPrev['ListaMapas'].length; i++)
        {
            info = JSON.stringify(jsonPrev['Mapas'][jsonPrev['ListaMapas'][i]][jsonPrev['ListaMapas'][i]], null, 4);
            info = info.substring(0 , info.indexOf('"EventosInfo"')-6) + "\}", null, 4;
            destparams = {
                Bucket: 'yr2-db',
                Key: jsonPrev['Partida'] + '/Mapas/' + [jsonPrev['ListaMapas'][i]] + '/Mapa.txt',
                Body: info
            };
            await s3.putObject(destparams).promise();
            for (var j = 0; j < jsonPrev['Mapas'][jsonPrev['ListaMapas'][i]][jsonPrev['ListaMapas'][i]]['eventos'].length;
j++) {
                info =
JSON.stringify(jsonPrev['Mapas'][jsonPrev['ListaMapas'][i]][jsonPrev['ListaMapas'][i]]['EventosInfo'][jsonPrev['Mapas'][jso
nPrev['ListaMapas'][i]][jsonPrev['ListaMapas'][i]]['eventos'][j]], null, 4);
                destparams = {
                    Bucket: 'yr2-db',
                    Key: jsonPrev['Partida'] + '/Mapas/' + [jsonPrev['ListaMapas'][i]] + '/Eventos/' +
jsonPrev['Mapas'][jsonPrev['ListaMapas'][i]][jsonPrev['ListaMapas'][i]]['eventos'][j] + '.txt',
                    Body: info
                };
                await s3.putObject(destparams).promise();
            }
        }

        var listaEventosPred = {
            'ListaEventosPred': jsonPrev['ListaEventosPred']
        };
        info = JSON.stringify(listaEventosPred, null, 4);
        destparams = {
```

```
                Bucket: 'yr2-db',
                Key: jsonPrev['Partida'] + '/ListaEventosPred.txt',
                Body: info
            };
            await s3.putObject(destparams).promise();

            for (var i = 0; i < jsonPrev['ListaEventosPred'].length; i++)
            {
                info = JSON.stringify(jsonPrev['EventosPred'][jsonPrev['ListaEventosPred'][i]], null, 4);
                destparams = {
                    Bucket: 'yr2-db',
                    Key: jsonPrev['Partida'] + '/EventosPred/' + jsonPrev['ListaEventosPred'][i] + '.txt',
                    Body: info
                };
                await s3.putObject(destparams).promise();
            }

            var variables = {
                'Variables': jsonPrev['Variables']
            };
            info = JSON.stringify(variables, null, 4);
            destparams = {
                Bucket: 'yr2-db',
                Key: jsonPrev['Partida'] + '/Variables.txt',
                Body: info
            };
            await s3.putObject(destparams).promise();

            return {
                statusCode: statusCodeReturn,
                headers: {
                    "Content-Type": "application/json",
                    "access-control-allow-origin": "*",
                    "header" : headerReturn
                },
                body: bodyReturn
            };

        } catch (err) {
            console.error(err);
            return {
                statusCode: 400,
                headers: {
                    "Content-Type": "application/json",
                    "access-control-allow-origin": "*",
                    "header" : headerReturn
                },
                body: "{}"
            };
        }
};
```

login()

Recibe el id de usuario y las contraseña codificada, valida el nombre de usuario y contraseña y devuelve la información de la cuenta.

```
exports.handler = async (event, context) => {

        const {id, password} = JSON.parse((event.body));
        var statusCodeReturn = 0;
        var bodyReturn = "";
        var headerReturn = "";

        try {

           const response = await getObject('Usuarios.txt');
           var objectStr = response.Body.toString();
           var jsObj = JSON.parse(objectStr);
           if (jsObj[id])
           {
              if (jsObj[id]['password'] == password)
              {
                 statusCodeReturn = 200;
                 headerReturn = "Login Correcto";
                 bodyReturn = '{"jugador":[';
                 for (var i = 0; i < jsObj[id]['partidas']['jugador'].length; i++) {
                    bodyReturn += "" + jsObj[id]['partidas']['jugador'][i] + "";
                    if ( i < jsObj[id]['partidas']['jugador'].length -1)
                       bodyReturn += ',';
                 }
                 bodyReturn += '],"master":[';
                 for (var i = 0; i < jsObj[id]['partidas']['master'].length; i++) {
                    bodyReturn += "" + jsObj[id]['partidas']['master'][i] + "";
                    if ( i < jsObj[id]['partidas']['master'].length -1)
                       bodyReturn += ',';
                 }
              bodyReturn += ']}';

              }
              else
              {
                  statusCodeReturn= 401;
                 headerReturn = "Error, password incorrecto";
              }

           }
           else
           {
              statusCodeReturn= 404;
              headerReturn = "Error, usuario no encontrado";
           }
```

```
            return {
                statusCode: statusCodeReturn,
                headers: {
                    "Content-Type": "application/json",
                    "access-control-allow-origin": "*",
                    "header" : headerReturn
                },
                body: bodyReturn
            };

        } catch (err) {
            console.error(err);
            return {
                statusCode: 400,
                header: "Error al pedir la información",
                body: ""
            };
        }
};
```

cambio()

Recibe el id del objeto del juego, mapa, evento o lo que sea que cambia, el id del usuario que hace el cambio, el id de partida, el id del mapa(puede ser redundante pero prefiero mantener todas las llamadas iguales), el tipo de cambio(mapa, evento, eventoPred, jugador), el timestamp y la información del cambio en json, guarda esta información en el bucket dándole un identificador de tiempo y unos destinatarios.

```
exports.handler = async (event, context) => {


        const objEv = JSON.parse((event.body));
        var statusCodeReturn = 200;
        var bodyReturn = "{}";
        var headerReturn = "";

        try {
            var response = await getObject(objEv['partida']+ '/EnCurso/Cambios/Cambios.txt');
            var objectStr = response.Body.toString();
            var jsCambios = JSON.parse(objectStr);


            response = await getObject(objEv['partida']+ '/Info.txt');
```

```
objectStr = response.Body.toString();
var jsInfo = JSON.parse(objectStr);
var jugadores = jsInfo['jugadores'];
var dirEv;

// Guarda en Cambios.txt la info y los destinatarios que son todos menos el que envio la información
if (objEv['tipo'] == 'jugador')
{
   dirEv = objEv['partida'] + 'EnCurso/Jugadores/' + objEv['id'] + '.txt';
   var destparamsJugador = {
      Bucket: 'yr2-db',
      Key: dirEv,
      Body: objEv['msg']
   };
   await s3.putObject(destparamsJugador).promise();

   if(jsInfo['jugadores'].includes(objEv['id']) )
   {
      jugadores.splice(jugadores.indexOf(objEv['jugador']),1);
   }
   else
   {
      jsInfo['jugadores'].push(objEv['id']);

      destparamsJugador = {
         Bucket: 'yr2-db',
         Key: objEv['partida']+ '/Info.txt',
         Body: jsInfo.toString()
      };
      await s3.putObject(destparamsJugador).promise();

   }

}
if (objEv['tipo'] == 'mapa')
{
   dirEv = objEv['partida'] + 'EnCurso/Mapas/' + objEv['id'] + '/Mapa.txt';
   response = await getObject(objEv['partida'] + '/EnCurso/ListaMapas.txt');
   objectStr = response.Body.toString();
   var jsMapa = JSON.parse(objectStr);
   if (!jsMapa['ListaMapas'].includes(objEv['id']))
      jsMapa['ListaMapas'].push(objEv['id']);

   var destparamsMapa = {
      Bucket: 'yr2-db',
      Key: objEv['partida'] + '/EnCurso/ListaMapas.txt',
      Body: jsMapa.toString()
   };
   await s3.putObject(destparamsMapa).promise();

   destparamsMapa = {
      Bucket: 'yr2-db',
      Key: dirEv,
      Body: objEv['msg']
   };
```

```
        await s3.putObject(destparamsMapa).promise();
        jugadores.splice(jugadores.indexOf(objEv['jugador']),1);
    }
    if (objEv['tipo'] == 'evento')
    {
        dirEv = objEv['partida'] + 'EnCurso/Mapas/' + objEv['mapa'] + '/Eventos/' + objEv['id'] + '.txt';
        response = await getObject(objEv['partida'] + '/EnCurso/Mapas/' + objEv['mapa'] + '/Mapa.txt');
        objectStr = response.Body.toString();
        var jsEvento = JSON.parse(objectStr);
        if (!jsEvento['eventos'].includes(objEv['id']))
            jsEvento['eventos'].push(objEv['id']);

        var destparamsEvento = {
            Bucket: 'yr2-db',
            Key: objEv['partida'] + '/EnCurso/Mapas/' + objEv['mapa'] + '/Mapa.txt',
            Body: jsEvento.toString()
        };
        await s3.putObject(destparamsEvento).promise();

        destparamsEvento = {
            Bucket: 'yr2-db',
            Key: dirEv,
            Body: objEv['msg']
        };
        await s3.putObject(destparamsEvento).promise();
        jugadores.splice(jugadores.indexOf(objEv['jugador']),1);
    }
    if (objEv['tipo'] == 'eventoPred')
    {
        dirEv = objEv['partida'] + 'EnCurso/EventosPred/' + objEv['id'] + '.txt';
        response = await getObject(objEv['partida'] + '/EnCurso/ListaEventosPred.txt');
        objectStr = response.Body.toString();
        var jsEventoPred = JSON.parse(objectStr);
        if (!jsEventoPred['ListaEventosPred'].includes(objEv['id']))
            jsEventoPred['ListaEventosPred'].push(objEv['id']);

        var destparamsEventoPred = {
            Bucket: 'yr2-db',
            Key: objEv['partida'] + '/EnCurso/ListaEventosPred.txt',
            Body: jsEventoPred.toString()
        };
        await s3.putObject(destparamsEventoPred).promise();


        destparamsEventoPred = {
            Bucket: 'yr2-db',
            Key: dirEv,
            Body: objEv['msg']
        };
        await s3.putObject(destparamsEventoPred).promise();

        jugadores.splice(jugadores.indexOf(objEv['jugador']),1);
    }
    if (objEv['tipo'] == 'variable')
    {
```

```
                dirEv = objEv['partida'] + 'EnCurso/Variables.txt';
                var destparamsVar = {
                    Bucket: 'yr2-db',
                    Key: objEv['partida'] + '/EnCurso/Variables.txt',
                    Body: objEv['msg']
                };
                await s3.putObject(destparamsVar).promise();
                jugadores.splice(jugadores.indexOf(objEv['jugador']),1);
            }
            if (objEv['tipo'] == 'info')
            {
                destparamsVar = {
                    Bucket: 'yr2-db',
                    Key: objEv['partida'] + '/Info.txt',
                    Body: objEv['msg']
                };
                await s3.putObject(destparamsVar).promise();
                jugadores.splice(jugadores.indexOf(objEv['jugador']),1);
            }

            //subir información a cambios.txt
            var timeStamp = objEv["timeStamp"];
            for (var i = 0; i < jugadores.length; i++)
            {
                while(jsCambios['destino'] && jsCambios['destino'][jugadores[i]] &&
jsCambios['destino'][jugadores[i]][timeStamp])
                {
                    timeStamp++;
                }
                if (!jsCambios['destino'])
                {
                    jsCambios['destino'] = {};
                }
                if (!jsCambios['destino'][jugadores[i]])
                {
                    jsCambios['destino'][jugadores[i]] = {};
                }
                jsCambios['destino'][jugadores[i]][timeStamp] = dirEv;

                jsCambios['cambios'][dirEv] = objEv['msg'];
            }

            var destparamsCambios = {
                Bucket: 'yr2-db',
                Key: objEv['partida'] + '/EnCurso/Cambios.txt',
                Body: objEv['msg']
            };
            await s3.putObject(destparamsCambios).promise();

            statusCodeReturn = 200;
            headerReturn = "Cambios realizados";
            bodyReturn = "{}";

            return {
                statusCode: statusCodeReturn,
```

```
            headers: {
               "Content-Type": "application/json",
               "access-control-allow-origin": "*",
               "header" : headerReturn
            },
            body: bodyReturn
         };

      } catch (err) {
         console.error(err);
         return {
            statusCode: 400,
            header: "Error al pedir la información",
            body: "{}"
         };
      }
};
```

pideArchivo()

Recibe el id de un archivo con información del juego y devuelve ese archivo.

```
exports.handler = async (event, context) => {


      const {id} = JSON.parse((event.body));
      var statusCodeReturn = 0;
      var bodyReturn = "";
      var headerReturn = "";

      try {
         const response = await getObject(id);
         var objectStr = response.Body.toString();

         statusCodeReturn = 200;
         headerReturn = "Archivo recuperado";
         bodyReturn = objectStr;


         return {
            statusCode: statusCodeReturn,
            headers: {
               "Content-Type": "application/json",
               "access-control-allow-origin": "*",
               "header" : headerReturn
            },
            body: bodyReturn
         };
```

```
        } catch (err) {
            console.error(err);
            return {
                statusCode: 400,
                header: "Error al pedir la información",
                body: ""
            };
        }
};
```

play()

Cambia el estado de la partida a Play.

```
exports.handler = async (event) => {

        const infoPrev = JSON.parse((event.body));

        try
        {

            var info = getObject(infoPrev['Partida'] + '/Info.txt');

            var destparams = {
            };

            for (var i = 0; i < infoPrev['ListaJugadores'].length; i++) {
                info = await getObject(infoPrev['Partida'] + '/Jugadores/' + infoPrev['ListaJugadores'][i] + '.txt');
                destparams = {
                    Bucket: 'yr2-db',
                    Key: infoPrev['Partida'] + '/EnCurso/Jugadores/' + infoPrev['ListaJugadores'][i] + '.txt',
                    Body: info.Body.toString()
                };
                await s3.putObject(destparams).promise();
            }

            for (var i = 0; i < infoPrev['ListaMapas'].length; i++) {
                info = await getObject(infoPrev['Partida'] + '/Mapas/' + infoPrev['ListaMapas'][i] + '/Mapa.txt');
                destparams = {
                    Bucket: 'yr2-db',
                    Key: infoPrev['Partida'] + '/EnCurso/Mapas/' + infoPrev['ListaMapas'][i] + '/Mapa.txt',
                    Body: info.Body.toString()
                };
                await s3.putObject(destparams).promise();
            }

            for (var i = 0; i < infoPrev['ListaEventosPred'].length; i++) {
                info = await getObject(infoPrev['Partida'] + '/EventosPred/' + infoPrev['ListaEventosPred'][i] + '.txt');
                destparams = {
```

```
                Bucket: 'yr2-db',
                Key: infoPrev['Partida'] + '/EnCurso/EventosPred/' + infoPrev['ListaEventosPred'][i] + '.txt',
                Body: info.Body.toString()
            };
            await s3.putObject(destparams).promise();
        }
        for (var i = 0; i < infoPrev['ListaEventos'].length; i++) {
            var str = infoPrev['ListaEventos'][i].toString().split('/');
            console.log(infoPrev['Partida'] + '/Mapas/' + str[1] + '/Eventos/' + str[0] + '.txt');
            info = await getObject(infoPrev['Partida'] + '/Mapas/' + str[1] + '/Eventos/' + str[0] + '.txt');
            destparams = {
                Bucket: 'yr2-db',
                Key: infoPrev['Partida'] + '/EnCurso/Mapas/' + infoPrev[str[1]] + '/Eventos/' + str[0] + '.txt',
                Body: info.Body.toString()
            };
            await s3.putObject(destparams).promise();
        }
        info =  await getObject(infoPrev['Partida'] + '/Variables.txt');
        destparams = {
            Bucket: 'yr2-db',
            Key: infoPrev['Partida'] + '/EnCurso/Variables.txt',
            Body: info.Body.toString()
        };
        await s3.putObject(destparams).promise();

        var cambios = '{"destino":{},"cambios":{}}';
        destparams = {
            Bucket: 'yr2-db',
            Key: infoPrev['Partida'] + '/EnCurso/Cambios/Cambios.txt',
            Body: cambios
        };
        await s3.putObject(destparams).promise();


        return {
            statusCode: 200,
            headers: {
                "Content-Type": "application/json",
                "access-control-allow-origin": "*",
                "header" : "Todo correcto"
            },
             body: "{}"
        };
    }
    catch(err)
    {
        console.error(err);
        return {
        statusCode: 400,
        header: "Error al pedir la información",
        body: "{}"
        };
    }
};
```

actualizaPartida()

Comprueba si la partida está en marcha y devuelve los cambios, manda id de partida e id de usuario. Se debe llamar una vez por segundo por cada jugador, el tiempo no tendría porque ser estricto pero cuanto más tiempo pase entre una llamada y otra más fácil es que los jugadores acusen el lag, si se llama demasiado a menudo puede causar problemas de incoherencia entre unas llamadas y otras porque alguna tarde más en procesarse y además aumentaría mucho el coste del servicio.

```
exports.handler = async (event, context) => {


        const {id, partida} = JSON.parse((event.body));
        var statusCodeReturn = 200;
        var bodyReturn = "{}";
        var headerReturn = "";

        try {
           var response = await getObject(partida + '/info.txt');
           var objectStr = response.Body.toString();
           var objJson = JSON.parse(objectStr);

           if (objJson['estadoPartida'] != 'stop')
           {
              response = await getObject(partida + '/EnCurso/Cambios/Cambios.txt');
              objectStr = response.Body.toString();
           }
           statusCodeReturn = 200;
           headerReturn = "Archivo recuperado";
           bodyReturn = objectStr;


        return {
            statusCode: statusCodeReturn,
           headers: {
              "Content-Type": "application/json",
              "access-control-allow-origin": "*",
              "header" : headerReturn
           },
           body: bodyReturn
        };

        } catch (err) {
           console.error(err);
           return {
```

```
                statusCode: 400,
                header: "Error al pedir la información",
                body: "{}"
            };
        }
};
```

logicaJuego()

Ejecuta la lógica del juego, únicamente en los mapas que tiene jugadores, esta función debe ser llamada por el master aproximadamente 1 vez cada segundo.

```
exports.handler = async (event, context) => {


    const {id, partida, mapas, master} = JSON.parse((event.body));
    try {
        var strMapas = mapas.split('/');
        var idObj = partida + '/Info.txt';

        var response = await getObject(idObj);
        var objectStr = response.Body.toString();
        var jsObjInfo = JSON.parse(objectStr);
        var prefijo = partida + "/";

        if (jsObjInfo['estadoPartida'] != "stop")
        {
            prefijo += "EnCurso/";
            var arJugadores = jsObjInfo['jugadores'];
            for (var i = 0; i < strMapas.length; i++)
            {
                idObj = prefijo + 'Mapas/' +  strMapas[i] + '/Mapa.txt';
                response = await getObject(idObj);
                objectStr = response.Body.toString();
                jsObjInfo = JSON.parse(objectStr);
                var v1 = 0;
                var v2 = 0;
                var auxDate = Date.now();

                for (var j = 0; j < jsObjInfo['eventos'].length; j++)
                {
                    idObj = prefijo + 'Mapas/' +  strMapas[i] + '/Eventos/' + jsObjInfo['eventos'][j] + '.txt';
                    response = await getObject(idObj);
                    objectStr = response.Body.toString();
                    var jsObjInfoEv = JSON.parse(objectStr);
                    var destparamsComando = {};
                    var auxCondiciones = {};
                    //var strToPlayer = "";
```

```
//La información de si está en play nos llega por parte de los jugadores
if (jsObjInfoEv['ejecutando'].toString() == 'true')
{
   //Codigo de ejecutar
   var strEjecucion = jsObjInfoEv['ejecucionActiva'].toString().split('/');
   //var ejecutor = strEjecucion[0];
   if (strEjecucion[1] == 'Defecto' || strEjecucion[1] == 'Automatico' || strEjecucion[1] == 'Trigger')
   {
      strEjecucion = jsObjInfoEv['comandosDefecto'].toString().split('/');
   }
   else if (strEjecucion[1] == "Atacar")
   {
      strEjecucion = jsObjInfoEv['comandosAtacar'].toString().split('/');
   }
   else if (strEjecucion[1] == "Provocar")
   {
      strEjecucion = jsObjInfoEv['comandosProvocar'].toString().split('/');
   }
   else if (strEjecucion[1] == "Robar")
   {
      strEjecucion = jsObjInfoEv['comandosRobar'].toString().split('/');
   }
   else if (strEjecucion[1] == "Enganar")
   {
      strEjecucion = jsObjInfoEv['comandosEnganar'].toString().split('/');
   }
   else if (strEjecucion[1] == "Carisma")
   {
      strEjecucion = jsObjInfoEv['comandosCarisma'].toString().split('/');
   }
   else if (strEjecucion[1] == "Recoger")
   {
      strEjecucion = jsObjInfoEv['comandosRecoger'].toString().split('/');
   }

   var lineaEjecucion = Number.parseInt(jsObjInfoEv['lineaEjecucion'].toString(), 10);
   if (lineaEjecucion < strEjecucion.length)
   {
      var strAux = JSON.parse(strEjecucion[lineaEjecucion]);
      var comando = '';
      var comando2 = '';
      var cambio = '';
      var cambio2 = '';
      if(strAux['uniqueId'] == 'If')
      {

         v1 = Number.parseInt(strAux['v1'].toString(), 10);
         v2 = Number.parseInt(strAux['v2'].toString(), 10);
         if (strAux['c'] == 'Mayor que' && v1 > v2)
         {
            lineaEjecucion++;
         }
         else if (strAux['c'] == 'Menor que' && v1 < v2)
        {
```

```
            lineaEjecucion++;
        }
        else if (strAux['c'] == 'Cierto' && v1 > 0)
        {
            lineaEjecucion++;
        }
        else
        {
            lineaEjecucion++;
            auxCondiciones = JSON.parse(strEjecucion[lineaEjecucion]);
            while (auxCondiciones['uniqueId'] != 'Endif')
            {
                lineaEjecucion++;
                auxCondiciones = JSON.parse(strEjecucion[lineaEjecucion]);
            }
        }
    }
if(strAux['uniqueId'] == 'Endif')
{
    var lineaAux = lineaEjecucion;
    lineaAux--;
    auxCondiciones = JSON.parse(strEjecucion[lineaAux]);

    while (auxCondiciones['uniqueId'] != 'If')
    {
        lineaAux--;
        auxCondiciones = JSON.parse(strEjecucion[lineaAux]);
    }

    v1 = Number.parseInt(auxCondiciones['v1'].toString(), 10);
    v2 = Number.parseInt(auxCondiciones['v2'].toString(), 10);
    if (auxCondiciones['c'] == 'Mayor que' && v1 > v2)
    {
        lineaAux++;
        lineaEjecucion = lineaAux;
    }
    else if (auxCondiciones['c'] == 'Menor que' && v1 < v2)
    {
        lineaAux++;
        lineaEjecucion = lineaAux;
    }
    else if (auxCondiciones['c'] == 'Cierto' && v1 > 0)
    {
        lineaAux++;
        lineaEjecucion = lineaAux;
    }
    else
    {
        lineaEjecucion++;
    }
}
if(strAux['uniqueId'] == 'Bucle')
{

    v1 = Number.parseInt(strAux['v1'].toString(), 10);
```

```
    v2 = Number.parseInt(strAux['v2'].toString(), 10);
    if (strAux['c'] == 'Mayor que' && v1 > v2)
    {
       lineaEjecucion++;
    }
    else if (strAux['c'] == 'Menor que' && v1 < v2)
    {
       lineaEjecucion++;
    }
    else if (strAux['c'] == 'Cierto' && v1 > 0)
    {
       lineaEjecucion++;
    }
    else
    {
       lineaEjecucion++;
       auxCondiciones = JSON.parse(strEjecucion[lineaEjecucion]);
       while (auxCondiciones['uniqueId'] != 'EndBucle')
       {
          lineaEjecucion++;
          auxCondiciones = JSON.parse(strEjecucion[lineaEjecucion]);
       }
    }
 }
 if(strAux['uniqueId'] == 'EndBucle')
 {
    lineaAux = lineaEjecucion;
    lineaAux--;
    auxCondiciones = JSON.parse(strEjecucion[lineaAux]);

    while (auxCondiciones['uniqueId'] != 'Bucle')
    {
       lineaAux--;
       auxCondiciones = JSON.parse(strEjecucion[lineaAux]);
    }

    v1 = Number.parseInt(auxCondiciones['v1'].toString(), 10);
    v2 = Number.parseInt(auxCondiciones['v2'].toString(), 10);
    if (auxCondiciones['c'] == 'Mayor que' && v1 > v2)
    {
       lineaAux++;
       lineaEjecucion = lineaAux;
    }
    else if (auxCondiciones['c'] == 'Menor que' && v1 < v2)
    {
       lineaAux++;
       lineaEjecucion = lineaAux;
    }
    else if (auxCondiciones['c'] == 'Cierto' && v1 > 0)
    {
       lineaAux++;
       lineaEjecucion = lineaAux;
    }
    else
    {
```

```
            lineaEjecucion++;
        }
    }
    //Despues de comprobar las expresiones condicionales ejecutamos el codigo correspondiente
    strAux = JSON.parse(strEjecucion[lineaEjecucion]);

    if(strAux['uniqueId'] == 'VidImg')
    {
        if(strAux['isVideo'] == 'true')
        {
            comando = '{"vid":"' + strAux['vidImg'] + '"}';
        }
        else
        {
            comando = '{"img":"' + strAux['vidImg'] + '"}';
        }
        cambio = ' VidImg';
        lineaEjecucion++;
    }
    else if(strAux['uniqueId'] == 'Var')
    {
        var idObjVar = prefijo + '/Variables.txt';
        var responseVar = await getObject(idObjVar);
        var objectStrVar = responseVar.Body.toString();

        objectStrVar['Variables'][strAux['nombre']] = strAux['valor'];
        comando = objectStrVar['Variables'].toString();
        cambio = 'Variables.txt';

        lineaEjecucion++;

        destparamsComando = {
            Bucket: 'yr2-db',
            Key: prefijo + '/Variables.txt',
            Body: objectStrVar.toString()
        };
        await s3.putObject(destparamsComando).promise();
    }
    else if(strAux['uniqueId'] == 'Transp')
    {
        var idObjT = prefijo + 'Jugadores/' +  strAux['player'] + '.txt';
        var responseT = await getObject(idObjT);
        var objectStrT = responseT.Body.toString();

        objectStrT['mapa'] = strAux['mapa'];
        var pos = strAux['posX'] +'/' + strAux['posY'] +'/' + strAux['posZ'];
        objectStrT['pos'] = pos;

        comando = objectStrT.toString();
        cambio = 'Jugadores/' +  strAux['player'] + '.txt';
        lineaEjecucion++;

        destparamsComando = {
            Bucket: 'yr2-db',
            Key: prefijo + 'Jugadores/' +  strAux['player'] + '.txt',
```

```
                        Body: objectStrT.toString()
                    };
                    await s3.putObject(destparamsComando).promise();
                }
                else if(strAux['uniqueId'] == 'EventInst')
                {
                    var idObjI = prefijo + 'Mapas/' +  strMapas[i] + '/Mapa.txt';
                    var responseI = await getObject(idObjI);
                    var objectStrI = responseI.Body.toString();

                    var idObjI2 = prefijo + 'EventosPred/' +  strAux['objeto'] + '.txt';
                    var responseI2 = await getObject(idObjI2);
                    var objectStrI2 = responseI2.Body.toString();

                    objectStrI['eventos'].push(strAux['objeto']);
                    var posI = strAux['posX'] +'/' + strAux['posY'] +'/' + strAux['posZ'];
                    objectStrI2['pos'] = posI;


                    cambio = 'Mapas/' +  strMapas[i] + '/Mapa.txt';
                    cambio2 = 'EventosPred/' +  strAux['objeto'] + '.txt';
                    comando = objectStrI.toString();
                    comando2 = objectStrI2.toString();
                    lineaEjecucion++;

                    destparamsComando = {
                        Bucket: 'yr2-db',
                        Key: prefijo + 'Mapas/' +  strMapas[i] + '/Mapa.txt',
                        Body: objectStrI.toString()
                     };
                     await s3.putObject(destparamsComando).promise();

                    var destparamsComando2 = {
                        Bucket: 'yr2-db',
                        Key: prefijo + 'EventosPred/' +  strAux['objeto'] + '.txt',
                        Body: objectStrI2.toString()
                    };
                    await s3.putObject(destparamsComando2).promise();
                }
                else if(strAux['uniqueId'] == 'Random')
                {
                    var idObjVarR = prefijo + '/Variables.txt';
                    var responseVarR = await getObject(idObjVarR);
                    var objectStrVarR = responseVarR.Body.toString();
                    objectStrVarR['Variables']['Random'] = (Math.random() * (Number(strAux['max']) -
Number(strAux['min'])) + Number(strAux['min'])).toString();

                    comando = objectStrVarR['Variables'].toString();
                    cambio = 'Variables.txt';
                    lineaEjecucion++;

                    destparamsComando = {
                        Bucket: 'yr2-db',
                        Key: prefijo + '/Variables.txt',
                        Body: objectStrVarR.toString()
```

```
    };
    await s3.putObject(destparamsComando).promise();
}
else if(strAux['uniqueId'] == 'Force')
{
    var idObjVarF = prefijo + '/Mapas/' + strMapas[i] + '/Eventos/' + strAux['evento'] + '.txt';
    if (strAux['isEvento'] != 'true')
        idObjVarF = prefijo + '/Jugadores/' + strAux['evento'] + '.txt';
    var responseVarF = await getObject(idObjVarF);
    var objectStrVarF = responseVarF.Body.toString();

    var velF = strAux['x'] +'/' + strAux['y'] +'/' + strAux['z'];
    objectStrVarF['vel'] = velF;

    comando = objectStrVarF.toString();
    cambio = '/Mapas/' + strMapas[i] + '/Eventos/' + strAux['evento'] + '.txt';
    lineaEjecucion++;

    if (strAux['evento'] != jsObjInfo['eventos'][j])
    {
        destparamsComando = {
            Bucket: 'yr2-db',
            Key: prefijo + '/Mapas/' + strMapas[i] + '/Eventos/' + strAux['evento'] + '.txt',
            Body: objectStrVarF.toString()
         };
        await s3.putObject(destparamsComando).promise();
    }
    else
    {
        objectStrVarF = jsObjInfoEv;
     }
}
else if(strAux['uniqueId'] == 'AttributeCh')
{
    var idObjA = prefijo + '/Jugadores/' + strAux['player'] + '.txt';
    var responseA = await getObject(idObjA);
    var objectStrA = responseA.Body.toString();

    objectStrA[strAux['atributo']] = strAux['valor'];

    comando = objectStrA.toString();
    cambio = '/Jugadores/' + strAux['player'] + '.txt';
    lineaEjecucion++;

    destparamsComando = {
        Bucket: 'yr2-db',
        Key: prefijo + '/Jugadores/' + strAux['player'] + '.txt',
        Body: objectStrA.toString()
    };
    await s3.putObject(destparamsComando).promise();
}
else if(strAux['uniqueId'] == 'EventEr')
{
    var idObjEE = prefijo + '/Mapas/' + strAux['mapa'] + '/Mapa.txt';
    var responseEE = await getObject(idObjEE);
```

```
    var objectStrEE = responseEE.Body.toString();

    var k = 0;
    while (k < objectStrEE['eventos'].length && k > -1)
    {
      if (objectStrEE['eventos'][k] == strAux['nombre'])
      {
        objectStrEE['eventos'].splice(k,1);
        k = -1;
      }
      k++;
    }

    comando = objectStrEE.toString();
    cambio = '/Mapas/' + strAux['mapa'] + '/Mapa.txt';
    lineaEjecucion++;

    destparamsComando = {
    Bucket: 'yr2-db',
      Key: prefijo + '/Mapas/' + strAux['mapa'] + '/Mapa.txt',
      Body: objectStrEE.toString()
    };
    await s3.putObject(destparamsComando).promise();
  }
  else if(strAux['uniqueId'] == 'Weather')
  {
    var idObjW = prefijo + '/Mapas/' + strAux['mapa'] + '/Mapa.txt';
    var responseW = await getObject(idObjW);
    var objectStrW = responseW.Body.toString();

    objectStrW['clima'] = strAux['clima'];
    objectStrW['hora'] = strAux['hora'];

    comando = objectStrW.toString();
    cambio = '/Mapas/' + strAux['mapa'] + '/Mapa.txt';
    lineaEjecucion++;

    destparamsComando = {
      Bucket: 'yr2-db',
      Key: prefijo + '/Mapas/' + strAux['mapa'] + '/Mapa.txt',
      Body: objectStrW.toString()
    };
    await s3.putObject(destparamsComando).promise();
  }
  else if(strAux['uniqueId'] == 'FxSon')
  {

    comando = '{"fxSnd":"' + strAux['FxSon'] + '"}';
    cambio = 'FxSon';
    lineaEjecucion++;
  }
  else if(strAux['uniqueId'] == 'SonMus')
  {

    comando = '{"sndAmbiente":"' + strAux['ambiente'] +  '", "sndMusica":"' + strAux['musica'] + '"}';
```

```
                        cambio = 'SonMus';
                        lineaEjecucion++;
                    }
                    else if(strAux['uniqueId'] == 'Fx')
                    {

                        comando = '{"fx":"' + strAux['fx'] + '"}';
                        cambio = 'Fx';
                        lineaEjecucion++;
                    }
                    else if(strAux['uniqueId'] == 'Survey')
                    {

                        comando = '{"op1":"' + strAux['op1'] + '","op2":"' + strAux['op2'] + '","op3":"' + strAux['op3'] +
'","op4":"' + strAux['op4'] + '"}';
                        cambio = 'Survey';
                        lineaEjecucion++;
                    }
                    else if(strAux['uniqueId'] == 'Route')
                    {
                        //var route = strAux['route'];
                        if (strAux['route'] == 'Forward')
                        {
                        jsObjInfoEv['vel'] = '0/0/1';
                        }
                        else if (strAux['route'] == 'Backward')
                        {
                            jsObjInfoEv['vel'] = '0/0/-1';
                        }
                        else if (strAux['route'] == 'Left')
                        {
                            jsObjInfoEv['vel'] = '-1/0/0';
                        }
                        else if (strAux['route'] == 'Right')
                        {
                            jsObjInfoEv['vel'] = '1/0/0';
                        }
                        lineaEjecucion++;
                        //jsObjInfoEv['lineaEjecucion'] = lineaEjecucion.toString() + '/' + lineaEjecucionRoute.toString();
                        comando = jsObjInfoEv.toString();
                    }
                }

                //Subimos comandos y cambios de evento, los cambios que impliquen la DB se tienen que hacer
más arriba
                if (comando != '' || comando2 != '')
                {
                    var idObjCambios = prefijo + '/Cambios/Cambios.txt';
                    var responseCambios = await getObject(idObjCambios);
                    var objectStrCambios = responseCambios.Body.toString();


                    if (comando != '')
                    {
                        objectStrCambios['cambios'][cambio] = comando;
```

```
                        for (var i = 0; i < arJugadores.length; i++)
                        {
                            auxDate = Date.now();
                            //var strToPlayer = "" + auxDate.getTime().toString() + "':'" + cambio + "";
                            objectStrCambios['destino'][arJugadores[i]].push();
                        }
                    }
                    if (comando2 != '')
                    {
                        objectStrCambios['cambios'][cambio2] = comando2;
                        for (var i = 0; i < arJugadores.length; i++)
                        {
                            auxDate = Date.now();
                            //var strToPlayer = "" + auxDate.getTime().toString() + "':'" + cambio2 + "";
                            objectStrCambios['destino'][arJugadores[i]].push();
                        }
                    }

                    var destparamsCambios = {
                        Bucket: 'yr2-db',
                        Key: prefijo + '/Cambios/Cambios.txt',
                        Body: objectStrCambios.toString()
                    };
                    await s3.putObject(destparamsCambios).promise();
                }
                var destparamsEvento = {
                    Bucket: 'yr2-db',
                    Key: prefijo + 'Mapas/' +  strMapas[i] + '/Eventos/' + jsObjInfo['eventos'][j] + '.txt',
                    Body: jsObjInfoEv.toString()
                };
                await s3.putObject(destparamsEvento).promise();

            }
        }
    }
    return {
        statusCode: 200,
        headers: {
            "Content-Type": "application/json",
            "access-control-allow-origin": "*",
            "header" : "Ejecutada logica"
        },
        body: "{}"
    };
}
catch (err) {
    console.error(err);
    return {
        statusCode: 400,
        header: "Error al pedir la información",
        body: "{}"
    };
}
};
```

cargaPartida()

Recibe el id de la partida, reúne toda su información de la base de datos y la devuelve en formato json.

```
exports.handler = async (event, context) => {


        const {id} = JSON.parse((event.body));
        var statusCodeReturn = 200;
        var bodyReturn = "{}";
        var headerReturn = "";

        try {
            bodyReturn = '{"Info":';
            var response = await getObject(id + '/Info.txt');
            var objectStr = response.Body.toString();
            var jsObjInfo = JSON.parse(objectStr);

            bodyReturn += objectStr + ',';
            var prefijo = id + "/";

            if (jsObjInfo['estadoPartida'] != "stop")
                prefijo += "EnCurso/";

            bodyReturn += '"Jugadores":{';
            for (var i = 0; i < jsObjInfo['jugadores'].length; i++)
            {
                //bodyReturn += "" + jsObjInfo['jugadores'][i] + '":{';
                response = await getObject(prefijo + 'Jugadores/' + jsObjInfo['jugadores'][i] + '.txt');
                bodyReturn += "" + jsObjInfo['jugadores'][i] + '":' + response.Body.toString();
                // bodyReturn += '}';
                if (i < jsObjInfo['jugadores'].length -1)
                {
                    bodyReturn += ',';
                }
            }

            response = await getObject(id + '/ListaMapas.txt');
            objectStr = response.Body.toString();
            var jsObjListaMapas = JSON.parse(objectStr);
            bodyReturn += '},' + objectStr.slice(1,objectStr.length-2) + ',';


            bodyReturn += '"Mapas":{';
            for (var i = 0; i < jsObjListaMapas['ListaMapas'].length; i++)
            {
                console.log(prefijo + 'Mapas/' + jsObjListaMapas['ListaMapas'][i] + '/Mapa.txt');
                bodyReturn += "" + jsObjListaMapas['ListaMapas'][i] + '":{';
```

```
                response = await getObject(prefijo + 'Mapas/' + jsObjListaMapas['ListaMapas'][i] + '/Mapa.txt');
                var jsObjListaEventos = JSON.parse(response.Body.toString());
                var salidaEventos = '"EventosInfo":{';
                bodyReturn += '"' + jsObjListaMapas['ListaMapas'][i] + '":' + response.Body.toString();
                for (var j = 0; j < jsObjListaEventos['eventos'].length; j++)
                {
                   salidaEventos += '"' + jsObjListaEventos['eventos'][j] + '":';
                   response = await getObject(prefijo + 'Mapas/' + jsObjListaMapas['ListaMapas'][i] + '/Eventos/' +
jsObjListaEventos['eventos'][j] + '.txt');
                   salidaEventos += response.Body.toString() + '}';
                   if (i < jsObjListaEventos['eventos'].length -1)
                   {
                      salidaEventos += ',';
                   }
                }
                //salidaEventos += '}';
                bodyReturn = bodyReturn.slice(0, bodyReturn.length-2) + ',' + salidaEventos +
bodyReturn.slice(bodyReturn.length-2);
                bodyReturn += '}';
                if (i < jsObjListaMapas['ListaMapas'].length -1)
                {
                   bodyReturn += ',';
                }
             }
             bodyReturn += '},';

             console.log(id + '/ListaEventosPred.txt');
             response = await getObject(id + '/ListaEventosPred.txt');
             objectStr = response.Body.toString();
             var jsObjListaEventosPred = JSON.parse(objectStr);
             bodyReturn += objectStr.slice(1,objectStr.length-2) + ',';

             bodyReturn += '"EventosPred":{';
             for (var i = 0; i < jsObjListaEventosPred['ListaEventosPred'].length; i++)
             {
                console.log(prefijo + 'EventosPred/' + jsObjListaEventosPred['ListaEventosPred'][i] + '.txt');
                response = await getObject(prefijo + 'EventosPred/' + jsObjListaEventosPred['ListaEventosPred'][i] +
'.txt');
                bodyReturn += '"' + jsObjListaEventosPred['ListaEventosPred'][i] + '":' + response.Body.toString();
                if (i < jsObjListaEventosPred   ['ListaEventosPred'].length -1)
                {
                   bodyReturn += ',';
                }
             }
             bodyReturn += '},';

             response = await getObject(prefijo + 'Variables.txt');
             objectStr = response.Body.toString();

             bodyReturn += objectStr.slice(1);




             statusCodeReturn = 200;
```

```
            headerReturn = "Archivo recuperado";
            return {
                statusCode: statusCodeReturn,
                headers: {
                    "Content-Type": "application/json",
                    "access-control-allow-origin": "*",
                    "header" : headerReturn
                },
                body: bodyReturn
            };

        } catch (err) {
            console.error(err);
            return {
                statusCode: 400,
                header: "Error al pedir la información",
                body: "{}"
            };
        }
};
```

creaUsuario()

Recibe id y password (ya debe estar encriptada para evitar pasar información sensible por red) como mínimo, aparte de otra información de usuario, crea un nuevo usuario.

```
exports.handler = async (event) => {
        const {id, password, nombre, apellidos, email} = JSON.parse((event.body));
        var statusCodeReturn = 0;
        var bodyReturn = "{}";
        var headerReturn = "";

        try {

            const response = await getObject('Usuarios.txt');
            var objectStr = response.Body.toString();
            var jsObj = JSON.parse(objectStr);
            if (jsObj[id])
            {

                statusCodeReturn= 400;
                headerReturn = "Error, usuario existente";

            }
            else
            {
```

```
                var usuario = {
                    'nombre': nombre,
                     'apellidos':apellidos,
                     'e-mail':email,
                     'partidas':{
                         'jugador':[],
                         'master':[]
                     },
                     'password':password
                 };
                 jsObj[id] = usuario;
                var destparams = {
                    Bucket: 'yr2-db',
                    Key: 'Usuarios.txt',
                    Body: JSON.stringify(jsObj, null, 4)
                };
                await s3.putObject(destparams).promise();

                statusCodeReturn= 200;
                headerReturn = "Usuario creado";
            }

            return {
                statusCode: statusCodeReturn,
                headers: {
                    "Content-Type": "application/json",
                    "access-control-allow-origin": "*",
                    "header" : headerReturn
                },
                body: bodyReturn
            };

        } catch (err) {
            console.error(err);
            return {
                statusCode: 400,
                header: "Error al pedir la información",
                body: "{}"
            };
        }
};
```

creaPartida()

Recibe id de partida y usuario, crea una nueva partida y le asigna al creador el rol de master.

```
exports.handler = async (event) => {
```

```
const {id, master} = JSON.parse((event.body));
var statusCodeReturn = 0;
var bodyReturn = "{}";
var headerReturn = "";

try {

   const response = await getObject('Partidas.txt');
   var objectStr = response.Body.toString();
   var jsObj = JSON.parse(objectStr);
   if (jsObj[id])
   {

      statusCodeReturn= 400;
      headerReturn = "Error, partida existente";

   }
   else
   {
      //const nuevaPartida = ',"' +  id + '":';
      //var output = objectStr.substring(0, objectStr.length - 1) + nuevaPartida + '}';
      jsObj[id] = "partida";
      var destparams = {
         Bucket: 'yr2-db',
         Key: 'Partidas.txt',
         Body: JSON.stringify(jsObj, null, 4)
      };
      await s3.putObject(destparams).promise();

      const responseUser = await getObject('Usuarios.txt');
      var objectStrUser = responseUser.Body.toString();
      var jsObjUser = JSON.parse(objectStrUser);
      if (jsObjUser[master])
      {
         jsObjUser[master]['partidas']['master'].putObject(id);
         var destparamsUser = {
            Bucket: 'yr2-db',
            Key: 'Usuarios.txt',
            Body: jsObjUser.toString()
         };
         await s3.putObject(destparamsUser).promise();


         var info = {
            'master':  master,
            'estadoPartida':'stop',
            'chat':'',
            'jugadores':[]
         };
         destparamsUser = {
            Bucket: 'yr2-db',
            Key: id + '/Info.txt',
            Body: info.toString()
         };
         await s3.putObject(destparamsUser).promise();
```

```
    var listaEventosPred = {
        'ListaEventosPred': []
    };
    destparamsUser = {
        Bucket: 'yr2-db',
        Key: id + '/ListaEventosPred.txt',
        Body: listaEventosPred.toString()
    };
    await s3.putObject(destparamsUser).promise();

    var listaMapas = {
        'ListaMapas': [
            'Mapa00'
        ]
    };
    destparamsUser = {
        Bucket: 'yr2-db',
        Key: id + '/ListaMapas.txt',
        Body: listaMapas.toString()
    };
    await s3.putObject(destparamsUser).promise();

    var variables = {
        'Variables': []
    };
    destparamsUser = {
        Bucket: 'yr2-db',
        Key: id + '/Variables.txt',
        Body: variables.toString()
    };
    await s3.putObject(destparamsUser).promise();

    var map = {
        'id':'Mapa00',
        'clima':'sol',
        'hora':'8',
        'elementos':[],
        'eventos':[]
    };
    destparamsUser = {
        Bucket: 'yr2-db',
        Key: id + '/Mapas/Mapa00/Mapa.txt',
        Body: map.toString()
    };
    await s3.putObject(destparamsUser).promise();

    var cambios = '{}';
    destparamsUser = {
        Bucket: 'yr2-db',
        Key: id + '/EnCurso/Cambios/Cambios.txt',
        Body: cambios.toString()
    };
    await s3.putObject(destparamsUser).promise();
}
```

```
            }

            return {
                statusCode: statusCodeReturn,
                headers: {
                    "Content-Type": "application/json",
                    "access-control-allow-origin": "*",
                    "header" : headerReturn
                },
                body: bodyReturn
            };

        } catch (err) {
            console.error(err);
            return {
                statusCode: 400,
                header: "Error al pedir la información",
                body: "{}"
            };
        }
};
```

C - Anexo: Repositorio del proyecto

Se trata de un repositorio de mercurial, para mirarlo recomiendo usar Tortoisehg.

ssh://phiram@hg.code.sf.net/p/yr2/code