**Proyecto Final. Redes y Videojuegos en Red**

**Instrucciones para la entrega**

* **Rellenar esta memoria** con la memoria del proyecto.
* Si el proyecto se realiza en **pareja** sólo es necesario **rellenar una memoria del proyecto**.
* El **código** del proyecto se entregará usando [www.github.com](http://www.github.com/).
* La **evaluación** del proyecto tendrá en cuenta el último commit antes de la fecha de entrega. No se considerarán commits posteriores.
* El **proyecto deberá defenderse en una presentación con el profesor**. En el acto de presentación los alumnos responderán preguntas sobre el proyecto realizado.

**Formulario de entrega**

|  |  |  |
| --- | --- | --- |
| **Título** | Magic Maze | |
| **Alumno** | Apellidos y Nombre | Eva Sánchez Muñoz |
| Usuario github | evsanc07@ucm.es |
| **Alumno** | Apellidos y Nombre | Emile de Kadt |
| Usuario github | edekadt@ucm.es |
| **URL del repositorio** | | <https://github.com/edekadt/totally-not-magic-maze> |

**Memoria del Proyecto**

**Breve Descripción:**

Magic Maze es un juego multijugador cooperativo, donde dos jugadores tendrán que conducir un grupo de personajes hacia las salidas del edificio donde están situados.

El proyecto está basado en un juego de mesa llamado Magic Maze[1] donde una serie de jugadores, cada uno con diferentes tipos de acciones, deben mover a los personajes por un laberinto hasta sus respectivas salidas sin comunicarse de forma verbal entre ellos. En esta versión, hemos decidido simplificar las reglas originales, eliminando el tiempo límite del que disponen los jugadores para finalizar la partida, las casillas especiales (Como por ejemplo, los portales o las escaleras) o la composición del mapa, con una mecánica similar a la presente en el dominó.[2]

Por turnos, cada jugador deberá utilizar los comandos que tenga asignados para desplazar a todos los personajes en una dirección o, si la situación lo requiere, saltar el turno. Los comandos estarán divididos en dos, correspondiendo los del primer jugador al eje vertical, y los del segundo al eje horizontal. Ambos jugadores, además, dispondrán del comando para "saltar" de turno.

Por una decisión de diseño, se decidió que los comandos de teclado moverían a todos los personajes a la vez, similar a la mecánica de desplazamiento que podemos encontrar en juegos como el 2048[3] , en contraste con los movimientos individuales que podemos realizar en el juego original.

Para terminar una sala, los jugadores deberán conducir a cada personaje a la casilla de salida del mismo color, siendo el máximo número de personajes en una misma sala cuatro y el mínimo dos. Tras finalizar, el juego generará una nueva sala diferente, con un nuevo número de personajes aleatorio.

**1. Arquitectura del Juego:**

**1.1 Modelo del juego:**

En la versión original del proyecto planteamos un modelo basado en la topología de **Cliente - Servidor**[5], donde dos clientes estarían conectados a un servidor que controlaría todos los aspectos del juego. Sin embargo, durante el desarrollo del proyecto, se decidió que sería mejor solución implementar una de las variaciones de susodicha topología, siendo el propio servidor uno de los dos clientes.

Este modelo implica que el jugador que posea el rol de host (Es decir, el cliente encargado de funcionar como un servidor) deberá poseer una buena capacidad de red, ya que poseerá el conjunto de sistemas que desarrollarán la lógica interna del juego, además de los sistemas que compartirá con el jugador invitado (En este caso, el cliente conectado al servidor creado por el host)

**1.2 Estado del juego:**

El estado del juego vendrá dado por el número de jugadores conectados.

En el caso de haber dos jugadores conectados al mismo servidor, el estado del juego será el estado de **ejecución,** donde ambos jugadores podrán desplazar a los personajes a través del mapa con una serie de comandos establecidos según su rol (Host o invitado) Por otro lado, si solo hay un jugador conectado, dependiendo del rol que posea, podremos encontrarnos con una partida en el estado de **pausa** o inexistente.

Tendremos dos ejecutables, uno para el cliente y otro para el servidor. Cuando se lanza el ejecutable del servidor, este abre un servidor en el puerto y la dirección que reciba como argumentos. Gracias a estos dos ejecutables, se decidirá qué jugador realizará el rol de host (es decir, el jugador que jugará con el cliente-servidor) y cuál el rol de invitado (el jugador que jugará como cliente).

//Posibilidad de inflar más esto

**1.3 Objetos:**

En el juego podremos encontrar dos tipos diferentes de objetos: los héroes y el mapa.

Los **héroes** son los objetos interactuables del juego. Los jugadores podrán controlar su desplazamiento con las flechas de dirección del teclado, moviéndose todos los héroes presentes en el tablero una casilla en la dirección seleccionada, siempre y cuando pueda realizar dicho movimiento.

￼￼

*En las dos imágenes de arriba, podemos ver un ejemplo del desplazamiento de los personajes por el mapa. En la primera imagen, podemos apreciar la posición de los héroes antes de que uno de los jugadores realice una pulsación de teclado, en este caso, la flecha de dirección derecha. Tras ejecutar la lógica del movimiento, todos los héroes han sido desplazados una casilla hacia la derecha, menos el héroe de color amarillo, que no puede realizar susodicha maniobra debido al obstáculo que hay en el camino y, por tanto, permanece en su posición actual.*

Cada héroe poseerá un identificador propio, con el cual determinaremos su aspecto (es decir, el color que lo representa: azul, rojo, verde o amarillo) y cual será su casilla de salida (esta casilla tendrá el mismo color del héroe, aunque formará parte del mapa). Además del identificador ya mencionado, los héroes pertenecerán a un grupo que permitirá identicarlos a la hora de realizar diferentes tipos de maniobras, ya sea indicando que han conseguido alcanzar la salida o bloqueando su movimiento, como podemos apreciar en el ejemplo anterior.

El otro objeto que podemos encontrar en el juego es el **mapa**, compuesto por casillas de tres tipos: muros, salidas y vacías. Los héroes podrán estar situados en las casillas vacías (que son casillas representadas con el color gris) y en las casillas de salida (en cada sala habrá el mismo número de salidas y héroes, habiendo una salida para cada héroe), pero no en las casillas de muros.

El mapa cambia en cada sala, habiendo implementado un algoritmo que genera de forma semialeatoria el mismo. Este algoritmo está encargado de posicionar a los personajes en el mapa, generando una serie de caminos de longitud aleatoria entre 20 y 40 pasos. El final de cada camino se utilizará para establecer el punto de inicio de cada héroe. Una vez establecidos todos los caminos, el algoritmo crea muros en las casillas restantes del mapa, terminando de formar el diseño del nivel.

Un punto que debemos resaltar respecto al mapa es la existencia de un nivel diseñado a mano, utilizada como primera sala en cada partida. Es un nivel sencillo, con cuatro personajes, que funciona como un pequeño tutorial para introducir las mecánicas del juego.

**1.4 Replicación:**

El juego utilizará un modelo **servidor-cliente UDP**, debido a que buscamos comprobar con cada envío de mensajes, quien es el jugador que ha realizado la pulsación de teclado, para comprobar si ha pulsado las teclas correspondientes o, en el caso de salir del juego, cual es el protocolo que debemos seguir.

Tras inicializar el juego, el servidor enviará un mensaje de *nuevo mapa* al cliente, que contiene toda la información que corresponde a la distribución de las diferentes casillas del mapa y la posición inicial de los personajes.

Durante el transcurso del juego, el cliente mandará mensajes de *movimiento* al servidor. El servidor calculará los resultados y enviará un nuevo mensaje de *posiciones* al cliente, que contendrá solo las nuevas posiciones de los personajes, tanto las que han cambiado como las que no. De este modo, el cliente no necesita realizar los cálculos detrás de sus movimientos, encargándose de ello el servidor, que sincronizará ambos clientes.

En el caso de llegar a su correspondiente casilla de salida, el personaje será eliminado y desaparecerá del mapa. Si todos los héroes han sido eliminados, se generará un nuevo mapa de juego y se realizará el envío del mensaje de *nuevo mapa*.

Aparte de los mensajes mencionados, tendremos mensajes de *desconectado* y *conectado,* que solo serán enviados por el cliente cuando este se conecta o desconecta al servidor.

**1.5 Protocolo de aplicación:**

Poner aquí datos generales sobre el motor para el protocolo

**1.6 Serialización:**

El envío de los mensajes se realiza de forma binaria, en bytes. Los mensajes transmitidos luego son deserializados en otro mensaje.

A pesar de que el cliente y el servidor no pueden enviar todos los mismos tipos de mensajes, es importante que ambos los reconozcan, ya que deben deserializar aquellos que no puedan enviar y viceversa.

//Creo que podemos inflarlo mas

**2. Diseño del Servidor**

El servidor del juego está conformado por los scripts de Manager, Component, Game, HeroSystem y MapSystem, encargados de realizar la lógica del juego. Estos scripts están encargados de recibir la información de los mensajes y, según que tipo de mensaje reciban, realizar unas acciones u otras (Algunos ejemplos pueden ser calcular las próximas posiciones de los héroes y comprobar que son válidas, generar un nuevo mapa de juego, determinar si la pulsación de teclado es correcta, etcétera)

En cada bucle de juego, el servidor actualiza el estado del mapa y de los héroes, realizando además un método refresh para eliminar las entidades que no pertezcan al juego.

*En esta sección describir las características del servidor, la lógica que ejecuta. Hilos de ejecución para replicación, simulación y los aspectos relevantes de su implementación. Discutir la tasa de actualizaciones.*

*Añadir las sub-secciones que se consideren necesarias*

**3. Diseño del Cliente**

El cliente del juego utilizará un script llamado RenderSystem, para renderizar en pantalla el contenido de los mensajes que reciba, y varios scripts utilizados para el intercambio de mensajes entre servidor y cliente.

(...)

En este diseño planteado, el cliente reduce el coste de recursos empleados a la hora de ejecutar el juego, debido a que no necesita realizar la lógica del mismo. Al ser un juego por turnos, no es necesario que haya una tasa de actualización demasiado elevada, pudiendo realizarse los cambios de renderizado con cada actualización de la pantalla.

*En esta sección describir las características del cliente, la lógica que ejecuta. Hilos de ejecución para la comunicación por red, entrada y render….*

*Añadir las sub-secciones que se consideren necesarias*

***x. Otras...***

*Opcionalmente incluir otras secciones que se consideren necesarias, por ejemplo análisis del impacto de la latencia en el modelo, optimizaciones realizadas….*

**4. Conclusiones**

En lo referente a los apartados positivos, hemos conseguido implementar un juego que modifica su distribución con cada partida jugada, logrando transmitir tanto la información del mapa como de los personajes entre el cliente y el servidor.

Uno de los puntos más importantes respecto a la realización del proyecto son las complicaciones que han surgido durante la realización de la conexión de redes, teniendo que desarrollar una comunicación más sencilla de lo que planeábamos en un inicio. Por este motivo, el código empleado para la realización de la mensajería podría mejorarse,

*Describir en esta sección los aspectos más importantes del proyecto y posibles mejoras.*

**Referencias**

*En esta sección listar las referencias consultadas que hayan servido para el desarrollo del proyecto. Las referencias deben estar numeradas y debe referenciarse en el lugar del texto que sea relevante.*