Teresa Garvía Gallego y Teresa Franco Corzo
3º A IMAT

Proyecto Final Paradigmas y Técnicas de la programación: SpellBound Forest

Contenido

1. Sobre el desarrollo del videojuego

Debido a la gran capacidad de extensión del videojuego hemos tenido que ir recortando implementaciones que teníamos en mente para poder implementar una versión funcional con los requisitos básicos del videojuego. Comenzamos generando el menú principal y el menú del juego, con varias funcionalidades, entre ellas un menú de Ajustes que está abierto para añadir futuras extensiones. Esta lógica básica del menú se adaptó para ser controlada desde el MainMenuManager y el GameMenuManager, que se corresponden cada uno con la gestión de una escena.

Para el movimiento del hechicero, primero se planteó utilizar las flechas del teclado y que se pudiera mover en ángulos de 0, 45, 90, 135 o 180 grados dejando la cámara estática centrada en el hechicero. Sin embargo, posteriormente se planteó que un movimiento más fluido sería mejor para que el usuario disfrutase más del juego. Por lo tanto, se cambiaron las teclas para la comodidad del usuario, permitiendo únicamente que el hechicero se mueva hacia delante o hacia atrás (W o S respectivamente) y su rotación con A o D. También se decidió implementar un control manual de la cámara independiente del movimiento del hechicero mediante el movimiento del cursor.

El bucle de juego cerrado que hemos conseguido implementar empieza desde el juego y llega al estado de victoria o el estado, reiniciando correctamente el juego, y contiene varios lienzos que indican de manera visual el estado en que se encuentra la partida.

El sistema de puntuaciones consiste en la vida del jugador, que empieza en el máximo posible (100/100) pero a medida que se encuentra con los enemigos que le quitan la vida, va disminuyendo. El objetivo del juego es, como en nuestra propuesta, alcanzar el tesoro, buscándolo en el mundo creado. El tesoro se genera de manera aleatoria entre 5 posiciones seleccionadas, y el mago entre 10 posiciones seleccionadas en el terreno.

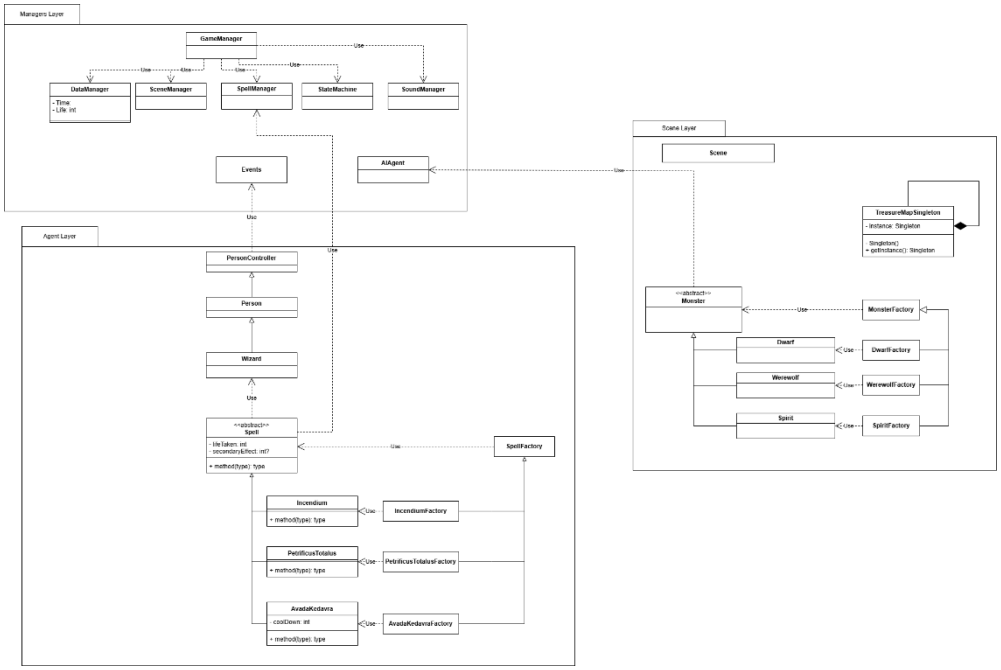
En un principio, habíamos decidido hacer tres tipos de monstruos (Dwarf, Werewolf, y Spirit) del que solo hemos logrado generar el enano, debido a la complejidad que nos llevaba ajustar el *prefab* y su IA correspondiente. Para lograr esto, hemos realizado una IA con el paquete *Navigation AI* disponible en Unity, que nos permitió primero, renderizar nuestro terreno y permitir y seleccionar porqué lugares queremos que camine el enano. Por otra parte, ajustar al *prefab* para otorgarle la calidad de Agente, por lo que puede caminar por este terreno generado. Su funcionamiento se reduce a una máquina de estados entre sus animaciones, lo que permite utilizar el patrón estado. Para lograr una IA de funcionamiento, se le asignaron cuatro esferas, a las que se dirige caminando de vez en cuando, variando entre su animación “idle” y “walk” , lo que permite un comportamiento muy similar a un NPC de los videojuegos que conocemos. Estas bolas (waypoints) se mueven con él, y cuando llega a alguna de ellas se dirige a otra de manera aleatoria. Para lograr el “ataque” de este enano, hemos diseñado una detección a través de distancias, y cuando detecta que el mago se encuentra a una distancia fija, le persigue, y cuando se encuentra a una distancia menor, le ataca. Para poder atacarle

y herirle tuvimos que tratar las físicas necesarias y los colisionadores para que se detectase únicamente la colisión entre el hacha y el cuerpo del jugador, que es en nuestro caso, el mago. Por su parte el mago también posee un complejo sistema de animación y disparo de hechizos, aunque en un principio visionamos poder usar 3 distintos y uno de ellos con *cooldown*, ahora mismo solo tenemos implementado un único hechizo que consta de un sistema de partículas el código en una factoría, abierta para la extensión en un futuro para añadir los demás hechizos. También se ha implementado un patrón factoría para añadir los enanos en la escena, seleccionando manualmente las posibles posiciones a lo largo y ancho del mapa. Estos poseen una barra de vida, como el mago, y fallecen si el mago los ataca las suficientes veces como para quitarles la vida, lanzándoles el hechizo. Esta gestión de las barras de vida se ha realizado conectando los módulos de los agentes de la escena con los managers, a través de eventos.

En nuestro caso, se destruyen tanto los enanos como el tesoro tras cada partida, pero el jugador (el mago) en sí no se destruye, porque es necesario para la cámara. Lo que se hace es reubicarle en el escenario.

Por último, no hemos decidido implementar un multijugador local ya que, para el movimiento final del jugador, que utiliza ratón y teclas del teclado, no consideramos que fuese buena idea añadir otro ratón. Tal vez se podría implementar para jugar en la nube, pero eso ya queda fuera del ámbito de este juego.

2. Comparación del UML



Como hemos comentado antes, no hemos implementado todas las funcionalidades, ni hemos instanciado todos los objetos a los que nos referíamos en nuestro UML inicial. También hemos tenido que prescindir del mapa del tesoro y del sistema de gestión del tiempo de juego. En un principio, teníamos pensado hacer que el jugador muriese si no encontraba en un tiempo determinado el tesoro, peor nos ha faltado tiempo para implementar esto. Hemos usado Singleton para hacer el MainMenuManager y el SpawManager entre otros managers que también son Singleton, Factorías para los hechizos y los enanos, y máquinas de estado para los estados del bucle de juego y animaciones.

Teresa Garvía Gallego y Teresa Franco Corzo
3º A IMAT