BUILD NOTES

El proyecto esta dividido en dos clases principales que son la clase Entity y la clase LogicManager.

La clase Entity es una interfaz de la que heredan tres clases; Character, Bullet y Enemy.

La clase LogicManager es un Singleton y su función es la de implementar el render y la lógica de nuestro juego.

A parte de estas dos clases principales también decidí implementar dos ficheros que incluían funciones auxiliares que me ayudarían con el pintado del mapa y las entidades y con la implementación de los sonidos (SoundFuncts y PrintFuncts).

Cada clase que hereda de Entity tiene funciones prácticamente iguales, pero con pequeños cambios, todas tendrán una posición y todas deben tener una forma de moverse (incrementando o decreciendo su posición) y a parte tendrán algunas funciones propias cada una de ellas, pero como el juego es bastante sencillo la verdad es que no hay demasiada diferencia entre unas y otras.

Decidí crear un Singleton para el LogicManager porque solo quería que se pudiese crear una única instancia de esta clase, desde esta misma se llama a toda la funcionalidad del renderizado de las Entities y su lógica, aunque pienso que hubiese sido mejor idea crear otra clase que se encargase de la gestión del renderizado de las Entities y que LogicManager solo se encargase de ejecutar la lógica del juego.

Con las PrintFuncs me ayudo a establecer la posición de cada Entity y la posición de cada cosa que se va a pintar en la pantalla gracias a la función gotoCords(), que se le pasa por parámetro el eje x a modificar y de manera estática tiene establecido un punto fijo para el eje y ya que es un juego que se maneja en una sola dimensión, la idea de utilizar esta función la he conseguido gracias a una práctica que hicimos anteriormente de fundamentos de c++ y me ha sido realmente útil para esta práctica.

SISTEMAS BAJO/MEDIO/ALTO NIVEL

Como sistema de **bajo nivel** he utilizado sobre todo nomenclaturas, sobre todo para las variables por parámetro que si tienen _ delante (ejemplo _Position) quiere decir que es una variable solo de entrada, si tuviese una atrás seria de salida y si tiene una atrás y otra adelante seria de entrada y salida, en este caso solo he utilizado de entrada. También he utilizado nombre-verbo para nombrar las funciones y alguna clase, por ejemplo: (LogicManager, SpawnEnemies(), BulletMovement(), etc…).

También he intentado comentar lo más importante y lo que podría ser mas complejo de entender para alguien que nunca haya visto el código de antes. A parte de los comentarios también he definido varias regiones, para separar las funciones de lógica de las del render en el LogicManager.cpp por ejemplo.

Como idioma he pensado que lo mejor era poner todo en inglés, tanto el código como los comentarios.

Como sistema de **nivel medio** he decidido implementar un patrón Singleton para mi clase LogicManager, pienso que en este caso era lo óptimo ya que no se busca crear mas instancias a esta clase, con una única instancia era suficiente. Si hubiese implementado una clase que manejase el render a parte de esta como comentaba anteriormente, pienso que también la habría hecho Singleton por el mismo motivo.

Como sistema de **nivel alto** he analizado todo el problema antes de implementar nada. También cuando empecé a implementarlo empecé haciendo la funcionalidad por partes y hasta que esa parte no funcionase correctamente no pasaba a la siguiente, por ejemplo; primero implementaba el mapa en la posición que quería realmente, luego el personaje, después el movimiento del personaje, cuando el personaje se movía como quería le implementé que pudiese disparar una única bala hacia la izquierda y así constantemente hasta que acabé la práctica.

¿QUE ME GUSTARIA HABER IMPLEMENTADO?

Realmente me hubiese gustado más mejorar cosas que implementar alguna nueva. Si hubiese implementado algo más a parte de unos contadores de las balas que tenemos disponibles para poder ver las balas que nos quedan en pantalla y la funcionalidad de ganar cuando el SCORE llegue a 20 que ya implementé por mi cuenta (que esta funcionalidad la hice sobre todo para comprobar más posibles errores y darle algo más de vida al juego), hubiese implementado otro input que cuando pulsases una tecla lanzase un “rayo láser” que destruyese todos los enemigos del lado al que el láser hubiese sido disparado.

Lo que me hubiese gustado mejorar sobre todo es que al haber creado un vector enemigos y balas para cada lado (EnemyL-EnemyR y BulletL-BulletR), he repetido mucho código en el que solo cambian un par de variables o condiciones. Pienso que hubiese quedado mucho mejor y más limpio hacer un único vector de enemigos y un único vector de balas.

Lo que más me ha molestado no solucionar de esta practica es que el juego como tal de vez cuando empieza a parpadear y es bastante molesto, por más que le he dado vueltas al código no he conseguido dar con cual podía ser el problema.

La razón por la que no he implementado y mejorado todo eso ha sido por mi mala gestión del tiempo, ya que se me fueron acumulando varias prácticas y tuve que empezar esta practica con menos tiempo del que debería haberla empezado y si le sumo de que tuve varios problemas al comienzo de la práctica de que no me funcionaban bien bastantes cosas como las colisiones de las balas con los enemigos y que el mapa se me descolocaba al disparar por fallos que eran bastante fáciles de solucionar pero que en su momento no veía porque estaba bastante bloqueado, pues.. se me echó un poco el tiempo encima. Aún así estoy bastante contento con el resultado de la práctica, pienso que pese a todo ha quedado medianamente bien estructurado, organizado y limpio dentro de lo que cabe.