Build Notes – Andrés Pérez

El juego funciona de forma sencilla:

- Al iniciar el juego se reproduce un loop de música y el player (O) aparece en la mitad del campo de juego. El score inicial es de 0-0.
- A partir de ahí, el jugador podrá moverse hacia izquierda o derecha con teclas "h" o "l" o disparar hacia izquierda o derecha con las teclas "j" o "k". El jugador solo podrá hacer una de estas acciones en cada frame.
- Al final de cada frame de lógica y de forma aleatoria, es posible que aparezca un enemigo en un extremo del mapa, los cuales irán avanzado hacia el player a velocidad de casilla por frame.
- En caso de que una bala se encuentre en la misma casilla que un enemigo, ambas entidades serán destruidas y el score aumentará en un (p.ej. 0-0 → 1-0).
- En caso de que un enemigo alcance la posición del jugador, se activará un sonido y se reiniciará la partida manteniendo el score, el cual aumentará en uno para los enemigos (p.ej. 1-0 →1-1)
- Para terminar de jugar se podrá pulsar la tecla "escape" en cualquier momento.

Estructura de clases

Se pueden diferenciar dos tipos de clases: los Singletons y aquellas que heredan de Entity.

Singletons

- GameLogic: Tiene referencias a las entidades que hay en juego, se encarga de gestionar la partida y de la eliminación de memoria de las entidades.
- Renderer: Se encarga de renderizar los caracteres asociados a cada entidad de juego y el mundo de juego.

Entity

- Entity: Esta clase sirve como padre de todas las otras e introduce el comportamiento mínimo que van a tener todas ellas: Posición, Carácter, Activo, Init, Update, Exit y Movimiento.
    - Bullet: Esta clase se encarga de actualizar la posición de las balas y chequear colisiones con enemigos.
    - Character: Esta clase sirve como inicio de una nueva rama de la herencia. Las entidades que hereden de Character tendrán que implementar la función de Muerte.
        - Player: Esta es la clase del jugador. Incluye el control del personaje, la habilidad de disparar y el input.
        - Enemy: Esta es la clase de los enemigos. Se encarga de moverlos y chequear colisiones con el player.

Se eligió esta estructura polimórfica porque daba mucha flexibilidad a la hora de hacer el método de actualización del GameManager y porque encapsulaba de forma correcta todas las funcionalidades a introducir de cada entidad de juego.

Estructura de Archivos

Dentro del zip, se encuentran las carpetas adicionales de Resources y Data, además de las generadas por Visual Estudio.

Resources

En esta carpeta se encuentran los archivos de código .h y .cpp, hay dos subcarpetas (Entities y Singletons) en las que se encuentran los archivos para las distintas clases y dos archivos: main.h y main.cpp que es donde se encuentra el bucle principal de juego.

Nomenclatura

Personalmente, he seguido la siguiente nomenclatura:

- "m_variable" para variables miembro.
- "_variable" para parámetros de entrada de una función.
- "variable_" para parámetros de salida de una función.
- "variable" para variables dentro de un scope específico.

Cabe destacar que antes del nombre de las variables miembro se indica el tipo:

- "m_iVar" para ints.
- "m_fVar" para los enteros.
- "m_uVar" para los unsigned ints.
- "m_cVar" para los chars
- "m_tVar" para los vectores y arrays.
- "m_pVar" para los punteros.

Elementos que me hubiera gustado incluir

Sistema de input

Me hubiera gustado implementar un sistema de input más responsivo y centralizado en un InputManager, pero no he sido capaz de hacerlo con los conocimientos actuales que poseo.

Diferentes velocidades de movimiento y Delta Time

Debido a la naturaleza del juego (1D y por casillas), el hecho de tener diferentes velocidades complicaba mucho las colisiones y por lo tanto las tuve que eliminar.

Mejor sistema de audios

La librería utilizada para reproducción de audios es muy limitada (solo tiene un canal de reproducción) y por lo tanto no queda un audio resultón. En caso de que hubiera sido posible me habría gustado hacer un mánager de audio que centralice la reproducción, cargar y descarga, pero debido a lo limitado que es, esto no ha resultado necesario.

IA

No se ha introducido una IA porque debido a la naturaleza del juego no se consideraba necesario ninguna implementación.

Pools de entidades

Para evitar hacer news durante el juego se pensó en utilizar pools de entidades inicializadas al comenzar la ejecución. Esto no se llevó acabo porque es algo que pensé el día de antes de la entrega gracias a un compañero y no daba tiempo a introducirlo.

Principios tenidos en cuenta

Superficiales:

- 0 Warnings
- Espaciado y tabulación constantes
- Nomenclatura constante
- Inclusión de comentarios
- Organización de ficheros

Intermedio

- Constructores
- Método Init, Update y Exit para todas las clases
- Singletons

Alto nivel

- Seudocódigo antes de comenzar a programar
- División en clases (sobre todo con Entity)
- No programación de features innecesarias
- Código modular y polimórfico.
- No implementación de nuevas tecnologías, todo visto antes en otras asignaturas y reconvertido.
- Separación de código en clases y funciones.
-

Conclusión

Considero que el código realizado para este proyecto es bastante expandible y flexible para diferentes tipos de juego o expansiones de este. Esto se consigue gracias al polimorfismo de la clase Entity y la centralización de información en Singletons. De esta forma se pueden cambiar los comportamientos de ambos campos y no se tendrán que actualizar muchos elementos en el otro para hacer que funcione.