MEMORIA DE PROYECTO
C++

Programación Avanzada.

Samuel González Linde

Máster Universitario en Computación Gráfica, Realidad Virtual y Simulación

Diciembre 2021

Contents

1 Introducción

El proyecto consiste crear un juego tipo Battle Royal que permita al usuario seleccionar un número de jugadores y personalizarlos añadiendoles una herramienta para combatir. También es posible elegir una clase que modificarán sus atributos. Finalmente, se realizará una simulación en la que los jugadores podrán encontrarse con otros jugadores o animales y luchar o dar un paseo tranquilamente y no encontrar problemas.

Al final, un único ganador será anunciado tras la muerte del resto de participantes.

A continuación se enumeran los diferentes características que se han implementado para llevar a cabo la solución:

1. Jerarquía para los personajes.
2. Jerarquía para las herramientas.
3. Clase que simule las batallas
4. Menú de usuario con selecciones en consola.

2 Solución propuesta

Veamos como se ha implementado las características anteriormente descritas para realizar la implementación del Battle Royale en el lenguaje de C++.

2.1 Jerarquía de personajes

2.1.1 Actores

La clase **Actors** es la raiz de la jerarquía de los personajes que residen dentro del juego. Contiene una variable de tipo *string* para almacenar el **nombre** que va a tener ese actor y es posible consultarla mediante el metodo **GetName**.

2.1.2 Combatientes

Esta clase, llamada **Combatant**, hereda de los actores y son aquellas que van a poder combatir dentro de nuestro *Campo de Batalla*.

A la vez que el nombre (debido a su clase padre), van a tener un atributo para la vida (**health**), otro para el daño (**damage**) y otro para la velocidad (**speed**).

Así mismo, se pueden consultar cualquiera de estos tres parámetros mediante las funciones **GetHealth**, **GetDamage** y **GetSpeed**.

También implementan a la interfaz **IDamageable** la cual nos proporciona el método para recibir daño: **TakeDamage**, que recibe una cantidad de daño como parámetro. Para poder recibir el daño, se ha implementado también

una funcion de ataque (**Attack**) que aplica el daño de un combatiente a un combatiente objetivo.

Todos los combatientes tienen una probabilidad (15%) de esquivar el ataque que reciben. Esta probabilidad es calculada dentro de la funcion **TakeDamage**.

Finalmente, la funcion **IsDead()** que se usa para comprobar si la salud del combatiente ha sido reducida a 0.

2.1.3 Jugadores y Animales

Estas dos clases, **Player** y **Animal**, son las actuales hojas de nuestra jerarquía.

En ambos casos se tiene un atributo estático para el conteo de la cantidad de jugadores y animales que hay en la escena. En el caso de los jugadores se irán reduciendo según vayan siendo eliminados del juego. Por el contrario, para el total de animales, como siempre serán encuentros aleatorios, no será necesario reducir su cantidad y nos servirá para tener las dimensiones de los arrays que contienen todos los animales que pueden ir apareciendo.

Nuestra clase **Player** contiene también una referencia a **Weapon** que será el arma que van a utilizar para incrementar su daño a la hora de combatir.

2.2 Jerarquia de herramientas

2.2.1 Items

De manera similar a la clase para los Actores, la clase Item se encarga de almacenar el nombre del objeto mediante una string y solicitar acceso a su contenido mediante una funcion *Get*.

2.2.2 Armas

Las armas vienen representadas en la clase **Weapon** que hereda de **Item**. Extiende la funcionalidad de esa clase añadiendo un atributo para almacenar el daño que provoca dicha arma. Este daño se almacena en forma de *int* y puede ser consultado con una función *Get*.

2.3 Simulación de la batalla

Para la simulación de la batalla se han creado dos clases específicas para tener un control completo sobre ella.

Por un lado tenemos **DataBase** y por otro lado tenemos **BattleGround**.

2.3.1 Base de Datos

En esta clase tenemos toda la información sobre las armas, animales y jugadores que se van a crear durante la partida.

En esta clase se encuentran definidas todas las armas que van a poder ser utilizadas y todos los animales que van a poder aparecer aleatoriamente en los encuentros. En ambos casos se encuentran almacenados en arrays de los tipos Weapon y Animal, respectivamente.

También contiene definiciones de adjetivos y nombres que van a ser combinados para crear de manera aleatoria nombres para los jugadores que indique el usuario.

Tenemos dos métodos *Get* para obtener las armas y los animales, y dos métodos *GetRandom* para obtener un arma aleatoria (en caso de que así lo decida el usuario) y un nombre aleatorio (generado a partir de los adjetivos y nombres antes nombrados).

2.3.2 Campo de batalla

Dentro de la clase **BattleGround** encontramos un array con los jugadores que están participando y otro array con los animales que se van a poder generar.

Esta clase será la encargada de simular los eventos que ocurren dentro del juego, conocer qué jugadores quedan vivos y mencionar al jugador ganador.

A continuación un listado con los métodos más importantes y una breve explicación:

- **PrintPlayers**: itera por el array de jugadores e imprime los jugadores que estan vivos.
- **NextEvent**: genera el siguiente evento de forma aleatoria, usando unos porcentajes definidos al comienzo de la clase (50% combate con jugador, 25% combate con animal, 25% turno tranquilo).
- **StartBattle**: simula la batalla entre dos combatientes. Comienza siempre el que mayor velocidad tenga y en caso de tener la misma, el primer combatiente. Haciendo uso de un array con los dos combatientes, se van pasando el turno el uno al otro dañandose hasta que alguno de los dos cae, entonces es cuando se anuncia el ganador del encuentro y se elimina del array de jugadores vivos al perdedor.
- **RemovePlayer**: si el jugador que se ha indicado se encuentra en el array, se desplaza todo el array una posición a la izquierda y se reduce el número total de jugadores en 1 (variable que sirve como indicador del tamaño del array y nos permitiría ëliminarëlementos del array).
- **GetPlayerOponent**: según el indice generado anteriormente de forma aleatoria, se busca otro indice para el array que no coincida con el del primer jugador.

2.4 Menú de usuario

Dentro del menú de usuario podremos elegir cuantos jugadores queremos que combatan en la partida; y a la hora de crear cada jugador, el arma que van a llevar con ellos para luchar.

Para el resto de interacciones con el menú, se le pedirá al usuario que vaya pulsando la tecla *Enter* para ir avanzando en los eventos del juego.

3 Herramientas utilizadas

Para el desarrollo de este proyecto se ha utilizado el lenguaje de programación C++, como ya se ha comentado anteriormente, junto con el IDE de Visual Studio Community 2019.

Se han utilizado las siguientes librerías de C++:

- **iostream**: para el tratamiento de cadenas de texto.
- **thread**: que nos permite parar el hilo de ejecución durante un determinado tiempo.
- **chrono**: para crear estructuras de datos de tipo tiempo (segundos, milisegundos, etc.).

4 Manual de Usuario

Para ejecutar la solución y poder usar la aplicación, tenemos que seguir los siguientes pasos:

1. Abrimos la solución en Visual Studio 2019
2. Ejecutamos la solución y esperamos a que se abra la consola
3. Una vez estamos en la consola iremos haciendo lo que el programa nos indica:
    - Indicamos el número de jugadores que van a participar.
    - Escogemos las armas que van a utilizar.
    - Vamos avanzando por los eventos pulsando *Enter* cuando se nos indique.