VNIVERSITAT DE VALÈNCIA

ESTRUCTURAS DE DATOS Y ALGORITMOS
***Grado en Ingeniería Multimedia (2º).* Curso 2022-23**

Práctica Nº 7: Tablas Hash en C++

Periodo de realización: Semana del **12/12/2022 a 16/12/2022**

Objetivos

- Utilizar tipos definidos en la biblioteca estándar de C++ para trabajar con montículos: `priority_queue`.
- Utilizar tipos definidos en la biblioteca estándar de C++ para trabajar con tablas hash: `unordered_map`.
- Utilizar clases definidas por otros programadores.

Introducción

En esta práctica se va a continuar con el trabajo realizado en la práctica 6, mejorando algunos aspectos del proceso. La simulación del juego y el comportamiento del NPC (*Non-Player Character*) deben modificarse respecto a la práctica anterior de acuerdo con los siguientes criterios:

- El NPC puede ganar nivel de vida en sus combates: En la práctica 6, el NPC siempre perdía nivel de vida en sus combates. Ahora, este personaje puede ganar o perder un combate. Una victoria implicará ganar nivel de vida y una derrota implicará reducir ese nivel.
- El NPC tiene memoria: El NPC "recuerda" el resultado de sus enfrentamientos con todos los personajes (humanos) con los que se haya encontrado en todas las sesiones del juego (no solo en la actual). La memoria recoge el número de victorias y número de derrotas sufridas con cada personaje.
- Cambia el criterio para determinar el personaje al que el NPC debe atacar: En la práctica 6 este criterio solo dependía del *aggro* de los atacantes en cada ciclo del juego, ahora también dependerá de los resultados en los combates previos contra cada atacante (usa la memoria).

Realizar estas modificaciones en el juego requiere cambiar algunos aspectos de la definición de la clase NPC y, por ello, se debe descargar la nueva versión del archivo "`NPC.h`". El cambio más importante es la introducción de la memoria del NPC, lo que implica disponer de una estructura que asocia el nombre de todos los personajes que en algún momento han jugado contra el NPC con los resultados globales de los combates del NPC con esos personajes. Para esta declaración se ha utilizado el tipo estándar `unordered_map` (tabla hash). En esta tabla se asocia el nombre del jugador con el número de victorias y el número de derrotas del NPC respecto a ese jugador. De esta manera, el acceso a la información de la memoria se puede realizar en tiempo `O(1)`.

Trabajo a realizar

Git: Debes trabajar en la carpeta "Pr7" del repositorio que ya tienes creado para la asignatura. Descarga los archivos disponibles en Aula Virtual y realiza un primer *commit* ("`Pr.7: Versión inicial`") y *push* con ese material inicial. En este caso, debes incorporar los archivos de la práctica anterior. A partir de aquí, ya sabes que debes registrar periódicamente los cambios realizados. La política de gestión de los repositorios es responsabilidad de los estudiantes y se valorará en el apartado "Organización" de los criterios de evaluación de las prácticas. Registra todos los *commits* como "`Pr.7: …`" para facilitar la identificación de cambios en el repositorio.

Se debe tomar como referencia la aplicación realizada en la práctica 6 e ir haciendo los cambios puntuales que se indican en cada tarea. Es fundamental descargar la nueva versión del interfaz de la clase NPC ("`NPC.h`").

Tarea 1: Incorporar memoria a un personaje NPC

En la revisión de la clase NPC se han definido los elementos necesarios para dotar al NPC de memoria. En particular, se ha definido en su parte privada la variable `memoria`, como una tabla hash cuya clave es el nombre de un personaje y cuyo valor asociado es un `Resultado`, que incluye el número de victorias y derrotas en los anteriores combates contra el personaje.

Obviamente, es necesario dotar de contenido a la memoria del NPC, por eso se requiere implementar el método `CargarMemoria()`, que debe cargar en la tabla `memoria` los datos contenidos en el archivo "`memoria.csv`". El archivo contiene los resultados de los encuentros previos del NPC con más de 1.800 personajes durante diferentes partidas del juego.

Por otro lado, en la nueva definición de la clase NPC también se ha modificado la declaración de la estructura `atacantes`, que almacena la información de los personajes que en esta partida concreta están interaccionando con el NPC. Estos son los personajes actualmente "visibles" por el NPC y, en general, serán muchos menos que los almacenados en su memoria. En la práctica 6 esta estructura era un `vector` y ahora es otra tabla hash, que asocia el nombre con un objeto de la clase *`Personaje`*. Como ha cambiado esta declaración, es evidente que se deben adaptar todas las operaciones que la usaban, en particular, se deben adaptar las implementaciones de los métodos `CargarAtacantes()` y `ActualizarCiclo()`[1].

Al cargar los atacantes del NPC hay que tener en cuenta que es posible que haya algún personaje nuevo que todavía no esté en la memoria del NPC. Es importante, detectar esta situación durante la carga y añadir a la memoria a cualquier personaje nuevo, con un número de victoria y derrotas igual a `0` (cero)[2].

Realiza las implementaciones indicadas y verifica su correcto funcionamiento.

[1] Estas son las modificaciones más relevantes, pero puede ser necesario modificar otros aspectos de tu implementación de la clase dependiendo de cómo la hayas realizado. Revisa todos los accesos a *`atacantes`*.
[2] Aviso: hay un atacante que no aparece en la memoria (`Eomer`).

Tarea 2: Modificar `Atacar()`

Como se ha dicho previamente, en esta práctica el NPC puede ganar o perder un combate contra un personaje humano. Una victoria implicará ganar nivel de vida y una derrota implicará reducir ese nivel. Por lo tanto, es necesario modificar el método `Atacar()` para que tenga en cuenta esta circunstancia, de acuerdo con el siguiente esquema algorítmico:

- Una vez seleccionado el personaje al que atacar en un ciclo del juego, se debe generar un número aleatorio entre `1` y `100`. Si el valor generado es menor o igual que la prioridad del personaje en el montículo entonces el NPC habrá sido derrotado[3] y se deberá:
  - Reducir la vida del NPC tanto como indique la prioridad del personaje
  - Actualizar la memoria para registrar una derrota más con este personaje.
- Si no se cumple la condición anterior, entonces el NPC habrá ganado el combate y se deberá:
  - Incrementar la vida del NPC tanto como indique la prioridad del personaje.
  - Actualizar la memoria para registrar una victoria más con este personaje.
- Las prioridades de los personajes podrían llegar a ser mayores que `100`. En ese caso, antes de determinar el resultado del combate, la prioridad se debe reducir al rango [`1`,`100`] dividiendo repetidamente por `10` hasta que sea menor que `100`.
- Todos los atacantes deberían ya estar incluidos en la memoria del NPC.

Tarea 3: Cambiar el criterio de selección del personaje

Se debe modificar el criterio para asignar prioridades en el montículo para los personajes atacantes. En la práctica 6, la prioridad era el *aggro* del personaje, pero ahora se van a tener también en cuenta el número de victorias y derrotas previas contra el personaje. La prioridad se debe definir como[4]:

*prioridad = aggro + victorias *(victorias – derrotas) / (victorias + derrotas)*

Esto implicará modificar todas aquellas operaciones en las que se utilice esta prioridad para insertar o extraer personajes del montículo (`CargarAtacantes()`, `ActualizarCiclo()`, etc.).

Tarea 4: Verificar la simulación del juego

El programa creado para la práctica 6 debería servir también para esta práctica, sin necesidad de realizar ningún cambio, ya que todas las modificaciones se han localizado en la lógica interna de funcionamiento de la clase NPC. Para cargar la memoria del NPC puedes invocar a la operación de carga en los constructores de la misma clase NPC, de manera que al crear un NPC se recupere automáticamente su memoria.

Para poder tener un banco de pruebas que permita contrastar los resultados con una referencia concreta es necesario poder controlar las victorias y derrotas que va a sufrir el NPC durante la simulación. Como este proceso se realiza de manera aleatoria, la forma que tenemos para controlar los resultados es fijar un valor concreto para la semilla de generación de números aleatorios. De esta manera, todas las ejecuciones utilizarán la misma secuencia de valores aleatorios y proporcionarán los

[3] De esta manera, la probabilidad de derrota depende de la prioridad del personaje en el montículo.
[4] Aviso: cuidado con las divisiones por `0`.

mismos resultados[5]. La semilla de generación de valores aleatorios se debe establecer también en los constructores de la clase NPC.

Los siguientes resultados han sido obtenidos estableciendo la semilla **1**. Se ha incluido información adicional sobre victorias y derrotas en los combates (y valor aleatorio generado) para facilitar la depuración. Verifica que los resultados de tu programa coinciden con estos:

```
Dragon de fuego
1000
------------------
Ciclo = -1
Personaje atacado: Ugluk (11)
aleat = 42
Victoria :-)
Vida NPC = 1011
------------------
Ciclo = 0
Personaje atacado: Tom Bombadil (21)
aleat = 68
Victoria :-)
Vida NPC = 1032
------------------
Ciclo = 1
Personaje atacado: Tom Bombadil (26)
aleat = 35
Victoria :-)
Vida NPC = 1058
------------------
Ciclo = 2
Personaje atacado: Frodo Gardner (30)
aleat = 1
Derrota :-(
Vida NPC = 1028
------------------
Ciclo = 3
Personaje atacado: Saruman (31)
aleat = 70
Victoria :-)
Vida NPC = 1059
------------------
Ciclo = 4
Personaje atacado: Grimbold (36)
aleat = 25
Derrota :-(
Vida NPC = 1023
------------------
Ciclo = 5
Personaje atacado: Grimbold (45)
aleat = 79
Victoria :-)
Vida NPC = 1068
------------------
Ciclo = 6
Personaje atacado: Eomer (44)
aleat = 59
Victoria :-)
Vida NPC = 1112
------------------
Ciclo = 7
Personaje atacado: Eomer (48)
```

[5] Para aleatorizar realmente la secuencia se debe usar una semilla diferente en cada ejecución. Suele ser habitual utilizar la hora del sistema cuando se ejecuta el programa para iniciar esta semilla. De esa forma, se tienen semillas diferentes sin necesidad de cambiar el código de la aplicación.

```
aleat = 63
Victoria :-)
Vida NPC = 1160
------------------
Ciclo = 8
Personaje atacado: Bill Helechal (46)
aleat = 65
Victoria :-)
Vida NPC = 1206
------------------
Ciclo = 9
Personaje atacado: Gollum (48)
aleat = 6
Derrota :-(
Vida NPC = 1158
------------------
Ciclo = 10
Personaje atacado: Bill Helechal (52)
aleat = 46
Derrota :-(
Vida NPC = 1106
------------------
Ciclo = 11
Personaje atacado: Bill Helechal (54)
aleat = 82
Victoria :-)
Vida NPC = 1160
------------------
Ciclo = 12
Personaje atacado: Grimbold (62)
aleat = 28
Derrota :-(
Vida NPC = 1098
------------------
Ciclo = 13
Personaje atacado: Grimbold (66)
aleat = 62
Derrota :-(
Vida NPC = 1032
------------------
Ciclo = 14
Personaje atacado: Grimbold (76)
aleat = 92
Victoria :-)
Vida NPC = 1108
------------------
Ciclo = 15
Personaje atacado: Grimbold (84)
aleat = 96
Victoria :-)
Vida NPC = 1192
------------------
Ciclo = 16
Personaje atacado: Grimbold (81)
aleat = 43
Derrota :-(
Vida NPC = 1111
------------------
Ciclo = 17
Personaje atacado: Bill Helechal (80)
aleat = 28
Derrota :-(
Vida NPC = 1031
------------------
Ciclo = 18
Personaje atacado: Bill Helechal (75)
```

```
aleat = 37
Derrota :-(
Vida NPC = 956
------------------
Ciclo = 19
Personaje atacado: Grimbold (75)
aleat = 92
Victoria :-)
Vida NPC = 1031

Ejecucion run finalizada
```

Tarea 3: Generación de documentación

El código del programa deberá estar correctamente documentado con Doxygen, de acuerdo con la guía de estilo. Una vez finalizado el programa, se deberá ejecutar Doxygen para verificar la correcta generación de la documentación del programa.