Práctica 1: Heroes of Zambunga

Programación 2

Curso 2021-2022

En esta primera práctica de la asignatura desarrollarás el juego de espada y brujería *Heroes of Zambunga*. Los conceptos necesarios para desarrollar esta práctica se trabajan en el *Tema 1* de teoría.

Condiciones de entrega

- La fecha límite de entrega para esta práctica es el **viernes 25 de febrero**, hasta las **23:59**
- Debes entregar un único fichero llamado `prac1.cc` con el código de todas las funciones

Código de honor

Si se detecta copia (total o parcial) en tu práctica, tendrás un **0** en la entrega y se informará a la dirección de la Escuela Politécnica Superior para que adopte medidas disciplinarias

Está bien discutir con tus compañeros posibles soluciones a las prácticas
Está bien apuntarte a una academia si sirve para obligarte a estudiar y hacer las prácticas

Está mal copiar código de otros compañeros para resolver tus problemas
Está mal apuntarte a una academia para que te hagan las prácticas

Si necesitas ayuda acude a tu profesor/a
No copies

Normas generales

- Debes entregar la práctica exclusivamente a través del servidor de prácticas del Departamento de Lenguajes y Sistemas Informáticos (DLSI). Se puede acceder a él de dos maneras:
  - Página principal del DLSI (`https://www.dlsi.ua.es`), opción "ENTREGA DE PRÁCTICAS"
  - Directamente en la dirección `https://pracdlsi.dlsi.ua.es`
- Cuestiones que debes tener en cuenta al hacer la entrega:
  - El usuario y la contraseña para entregar prácticas son los mismos que utilizas en UACloud
  - Puedes entregar la práctica varias veces, pero sólo se corregirá la última entrega
  - No se admitirán entregas por otros medios, como el correo electrónico o UACloud
  - No se admitirán entregas fuera de plazo
- Tu práctica debe poder ser compilada sin errores con el compilador de C++ existente en la distribución de Linux de los laboratorios de prácticas

- Si tu práctica no se puede compilar su calificación será 0
- El 70 % de la nota de la práctica dependerá de la corrección automática, por lo que es imprescindible que respetes estrictamente los textos y los formatos de salida que se indican en este enunciado. El otro 30 % dependerá de la revisión manual del código que haga tu profesor de prácticas, por lo que debes también ajustarte a la guía de estilo de la asignatura. Además se tendrá en cuenta que hayas actualizado tu código en *GitHub* todas las semanas
- Al comienzo de todos los ficheros fuente entregados debes incluir un comentario con tu NIF (o equivalente) y tu nombre. Por ejemplo:

prac1.cc

```
// DNI 12345678X GARCIA GARCIA, JUAN MANUEL
...
```

- El cálculo de la nota de la práctica y su relevancia en la nota final de la asignatura se detallan en las transparencias de presentación de la asignatura (*Tema 0*)

1. Descripción de la práctica

Heroes of Zambunga es un juego de combate por turnos en el que el personaje principal se enfrentará a diferentes enemigos y acumulará puntos de experiencia hasta que caiga derrotado. Toda la acción del juego se llevará a cabo en el terminal del ordenador en modo texto.

2. Detalles de implementación

En el Moodle de la asignatura se publicarán varios ficheros que necesitarás para la correcta realización de la práctica:

- `prac1.cc`. Este fichero contiene un esqueleto de programa sobre el que realizar tu práctica. Descárgalo y añade tu código en él. El fichero proporcionado incluye la siguiente información:
  - Los registros (`struct`) y el tipo enumerado descritos en la Sección 4
  - Una función `showMenu()` que muestra por pantalla el menú principal descrito en la Sección 7
  - Una función `rollDice()` que simulará el lanzamiento de un dado de 20 caras, generando números aleatorios entre 1 y 20
  - Los prototipos de las funciones principales que se deben implementar obligatoriamente para gestionar las opciones del juego. **Puedes modificar los parámetros de entrada y valor devuelto de estas funciones**. Lo único que no puedes hacer es cambiarles el nombre. Además de éstas, deberás incluir las funciones que consideres oportunas para que tu código sea más sencillo y fácil de leer e interpretar por un humano (más concretamente, por el profesor que tiene que corregir tu práctica)
  - Una función `main()` donde se define la semilla inicial con `srand()` a partir de un valor entero pasado por línea de comando. Esta semilla permite que se genere siempre la misma secuencia de números aleatorios con `rand()`, de manera que puedas obtener los mismos resultados en diferentes ejecuciones del programa. Un ejemplo de llamada al programa sería:

Terminal

```
$ prac1 4545
```

donde `4545` es la semilla. Siempre que ejecutes la práctica con esa semilla de entrada, verás que los tipos de enemigos y las puntuaciones de ataque/defensa que van saliendo al lanzar el dado son las mismas, porque `rand()` genera siempre la misma secuencia de números aleatorios. Usa una semilla diferente (ej. `777`) y verás cómo cambian esos valores

- `autocorrector-prac1.tgz`. Contiene los ficheros del autocorrector para probar la práctica con algunas pruebas de entrada. La corrección automática de la práctica se realizará con un programa similar, con estas pruebas y otras más definidas por el profesorado de la asignatura
- `prac1`. Fichero ejecutable de la práctica (compilado para máquinas Linux de 64 bits) desarrollado por el profesorado de la asignatura, para que puedas probarlo con las entradas que quieras y ver la salida correcta esperada

- En la corrección de la práctica se introducirán siempre datos del tipo correcto, aunque con valores que pueden ser incorrectos. Por ejemplo, si se piden los valores de ataque y defensa del personaje principal, se probará siempre con un valor entero, que podría ser `-1237` o `0`, pero nunca se introducirá un valor de otro tipo (carácter, `string`, número real, ...)
- El resto de decisiones en la implementación quedan a tu criterio, pero ten en cuenta que el código fuente será revisado por tu profesor de prácticas siguiendo la guía de estilo publicada en el Moodle de la asignatura. Parte de la nota de la práctica depende de dicha revisión

3. Funcionamiento del programa

El juego contará con un personaje principal (el héroe o heroína), que será el que maneje el jugador, y cuyas características deberán definirse al inicio del programa. La mecánica del juego es sencilla: en cada ronda aparecerá un enemigo (cuyas raza se seleccionará de manera aleatoria y que será controlado por el ordenador) al que deberá enfrentarse el jugador en un combate por turnos.

Cada vez que haya un combate, el héroe atacará en primer lugar y el enemigo se defenderá. A continuación, será el enemigo quien ataque y el héroe quien se defienda. Los puntos de ataque y defensa que tengan el héroe y el enemigo, junto con el lanzamiento de un dado, determinarán el resultado del combate que acabará cuando uno de los dos personajes muera. Si muere el enemigo, se generará uno nuevo y se procederá a iniciar una nueva ronda. Si muere el héroe, la partida termina y el programa finaliza.

Cuando el héroe derrota a un enemigo acumula puntos de experiencia (en función del tipo de enemigo). El objetivo del juego es acumular el máximo de puntos posible antes de caer derrotado.

4. Componentes

En el programa deben definirse tres tipos de registros: uno que almacene las características básicas del héroe y los enemigos (nivel de ataque, nivel de defensa y puntos de vida), otro que almacene el resto de datos del héroe y un tercero que almacene el resto de datos de los enemigos. Los siguientes apartados describen cada uno de estos registros en detalle. A estos registros no les puedes añadir nuevos campos ni modificar los existentes, pero sí puedes definir nuevos registros si crees que los necesitas.

4.1. Core

La estructura de tipo `Core` almacena la información relativa a las habilidades principales de los personajes, tanto del héroe como de los enemigos: los puntos de ataque (`attack`), los puntos de defensa (`defense`) y los puntos de vida (`hp`, abreviatura de *health points*). Todos estos campos serán de tipo entero y se almacenarán en un registro con el siguiente formato:

```
struct Core{
  int attack;
  int defense;
  int hp;
};
```

Los puntos de ataque determinan la fuerza con la que el personaje golpea (cuantos más mejor) y los puntos de defensa determinan la capacidad para resistir ataques (cuantos más mejor). Estos dos valores no varían a lo largo de la vida de los personajes. Los puntos de vida determinan la salud del personaje y se obtienen de manera automática multiplicando por dos los puntos de defensa, tal y como se describe en la Sección 6. Cuando este valor llega a `0` el personaje muere.

Por ejemplo, podríamos tener una estructura `Core` que almacenara un campo `attack` con valor `80`, un campo `defense` con valor `120` y un campo `hp` con valor `240`.

4.2. Enemy

Los enemigos son todos aquellos personajes controlados por el ordenador (*non playable character* o NPC) que se enfrentarán a nuestro héroe durante la partida. Cada enemigo pertenecerá a una raza específica (`name`) y tendrá una serie de características básicas (`features`) representadas por una estructura de tipo `Core` como la descrita en la sección anterior. La información de cada enemigo se almacenará en un registro con el siguiente formato:

```
struct Enemy{
  Breed name;
  Core features;
};
```

El tipo de dato `Breed` es un enumerado que identifica a la raza del enemigo y que se definirá de la siguiente manera:

```
enum Breed{
  AXOLOTL,
  TROLL,
  ORC,
  HELLHOUND,
  DRAGON
};
```

Es decir, en el juego existirán cinco tipos posibles de enemigos (de más débil a más fuerte): ajolote (`AXOLOTL`), trol (`TROLL`), orco (`ORC`), perro del infierno (`HELLHOUND`) y dragón (`DRAGON`). Cada una de estas razas tendrá unos valores diferentes para las características básicas de ataque y defensa que deberás tener en cuenta a la hora de crearlos durante la partida. La Tabla 1 resume estos valores.

name	attack	defense
AXOLOTL	40	40
TROLL	60	80
ORC	80	120
HELLHOUND	120	100
DRAGON	160	140

Tabla 1: Valores de ataque y defensa para cada raza de enemigo.

La primera fila de la tabla indica que el ajolote (`AXOLOTL`) tendrá un ataque de 40 y una defensa de 40, la segunda fila indica que el trol (`TROLL`) tendrá un ataque de 60 y una defensa de 80, etc.

Por ejemplo, podríamos tener un enemigo que fuera un dragón (`name = DRAGON`), con las siguientes características básicas (`features`): `160` puntos de ataque (`attack`), `140` puntos de defensa (`defense`) y `280` puntos de vida (`hp`).

- Los valores de los tipos enumerados se convierten internamente a un valor `int` que se corresponde con su posición en el `enum` (mira la transparencia 38 del *Tema 1*). Es decir, el enumerado `AXOLOTL` es igual a `0`, `TROLL` es igual a `1`, etc. Puedes aprovechar esta circunstancia para simplificar tu código y usar estos valores como índices de un array. Por ejemplo, `kills[AXOLOTL]` accederá a la posición `0` del vector `kills`, mientras que `kills[TROLL]` accederá a la posición `1`

4.3. Hero

El héroe o heroína es el personaje principal, que será manejado por el jugador. Tendrá un nombre (`name`), unas características básicas (`features`), un ataque especial (`special`), un número de veces que puede huir de la batalla (`runaways`), unos puntos de experiencia (`exp`) y una lista de enemigos derrotados (`kills`). Toda esta información se almacenará en un registro con el siguiente formato:

```
struct Hero{
  char name[KNAME];
  Core features;
  bool special;
  int runaways;
  int exp;
  int kills[KENEMIES];
};
```

La longitud máxima del nombre del héroe vendrá dada por la constante `KNAME`, que en nuestro programa deberá definirse de la siguiente manera:

```
const int KNAME=32;  // Máximo 31 caracteres y el '\0'
```

El ataque especial (`special`) permite que el jugador multiplique su potencia de ataque, pero esta ventaja solo puede utilizarse una vez durante la partida, tal y como se describe en la Sección 8.3. El jugador puede huir hasta 3 veces y evitar así al enemigo. El campo `runaways` se inicializará a 3 al crear el personaje y se irá decrementando cada vez que escape de una batalla (ver Sección 8.2). El vector `kills` almacenará cuántos enemigos se han derrotado de cada raza. El tamaño del array vendrá determinado por la constante `KENEMIES`, que indica cuántas razas diferentes de enemigo hay. En nuestro programa se definirá de la siguiente manera:

```
const int KENEMIES=5;  // Ya que hay 5 razas diferentes
```

Por ejemplo, podríamos tener un héroe llamado `Lord Papanatas` (`name`), cuyas habilidades básicas (`features`) sean: `90` puntos de ataque (`attack`), `110` puntos de defensa (`defense`) y que le queden `57` puntos de vida (`hp`). Este personaje podría no haber gastado todavía el ataque especial (`special == true`), tener `1000` puntos de experiencia (`exp`) y haber matado 3 ajolotes, 1 perro del infierno y 1 dragón, que se almacenarían de la siguiente manera en el vector `kills`:

kills	3	0	0	1	1

En la posición `0` del vector se almacenará el número de ajolotes muertos, en la `1` los trols, en la `2` los orcos, en la `3` los perros del infierno y en la `4` los dragones.

5. Lanzamiento del dado

Una característica importante para que el juego sea más entretenido es que exista un componente de aleatoriedad que haga cada partida diferente. Este efecto se conseguirá mediante el lanzamiento de un dado de 20 caras, que determinará tanto la raza de nuestros enemigos como la potencia del ataque y la defensa en cada turno del combate.

El lanzamiento del dado se simulará mediante la función `int rollDice()`, que generará de manera aleatoria un número entero comprendido entre 1 y 20. Esta función viene ya definida en el fichero `prac1.cc` proporcionado por el profesorado de la asignatura, por lo que simplemente tendrás que llamarla cada vez que quieras simular el lanzamiento del dado.

6. Comienza la partida

Al iniciar el programa se pedirá al jugador que introduzca una serie de datos para configurar al personaje. Esta tarea se deberá llevar a cabo en la función `createHero()` proporcionada en el fichero `prac1.cc`. En primer lugar, se pedirá el nombre del héroe con el siguiente mensaje:

Terminal

```
Enter hero name:
```

El valor introducido se almacenará en el campo `name` del registro `Hero`. El nombre podrá contener caracteres alfanuméricos (letras y números) y espacios en blanco, pero no podrá incluir ningún tipo de signo de puntuación (interrogantes, puntos, comas, paréntesis...) y deberá comenzar siempre por una letra. Si el nombre introducido es incorrecto, o el usuario deja el nombre en blanco, se mostrará el siguiente mensaje de error y se le volverá a solicitar el nombre hasta que se introduzca uno válido:

Terminal

```
ERROR: wrong name
```

A continuación se le pedirá al usuario que introduzca la distribución de puntos de ataque y defensa que quiere hacer para su personaje. Esta información se almacenará, respectivamente, en los campos `attack` y `defense` del campo `features` del registro `Hero`. Los puntos de habilidad iniciales que se pueden distribuir entre ataque y defensa estarán guardados en la constante `KPOINTS`, que se definirá en el código de la siguiente manera:

```
const int KPOINTS=200;  // Tenemos 200 puntos para distribuir
```

Para solicitar al jugador la distribución de estos puntos se mostrará el siguiente mensaje:

Terminal

```
Enter attack/defense:
```

El usuario deberá introducir como respuesta dos valores numéricos separados por una barra inclinada (`/`), representando el porcentaje de puntos de habilidad que se asigna a cada concepto. Por ejemplo, `40/60` indicaría que queremos asignar el 40 % de puntos al ataque y el 60 % a la defensa. La única limitación a la hora de distribuir estos valores es que ambos deben ser siempre mayores que `0` y su suma debe ser `100`. En caso contrario, deberá mostrarse el siguiente mensaje y volver a pedir la distribución al usuario:

Terminal

```
ERROR: wrong distribution
```

En el ejemplo anterior, con una distribución `40/60`, de los `200` puntos iniciales se asignarían `80` puntos al ataque y `120` a la defensa.

Los puntos de vida iniciales (campo `hp` de `features`) se calcularán de manera automática multiplicando por dos los puntos de defensa. En el ejemplo anterior, con `120` puntos de defensa, los puntos de vida iniciales serían `240`.

El campo `special` de la estructura `Hero` se pondrá inicialmente a `true`, indicando de esta manera que el ataque especial está disponible (ver Sección 8.3 para más información).

Los puntos de experiencia del personaje (`exp`) se inicializarán a `0` al comenzar la partida y se irán incrementando siguiendo el procedimiento que se describe en la siguiente sección.

Finalmente, el número de enemigos derrotados de cada tipo (los componentes del vector `kills`) deberá inicializarse a `0`.

- El nombre introducido para el héroe nunca tendrá más de `KNAME-1` caracteres (recuerda que hay que dejar un espacio para el `'\0'`), tal y como se definió en la Sección 4.3. No hace falta comprobarlo
- El nombre del héroe nunca contendrá letras acentuadas (solo letras del alfabeto inglés). No hace falta comprobarlo

7. ¡Al ataque!

Una vez introducidos los datos del personaje se generará el primer enemigo usando la función `createEnemy()`. Dentro de esta función, en primer lugar se deberá determinar la raza del enemigo lanzando un dado de 20 caras (recuerda que hay que usar la función `rollDice()` para simular el lanzamiento del dado):

Valor del dado	Raza
1 a 6	AXOLOTL
7 a 11	TROLL
12 a 15	ORC
16 a 18	HELLHOUND
19 a 20	DRAGON

Por ejemplo, si `rollDice()` devuelve `14`, el enemigo creado será un orco.

A continuación, se guardarán las características básicas (ataque, defensa y puntos de salud) del enemigo. El ataque y la defensa dependerán de la raza, tal y como se describió en la Tabla 1. Los puntos de salud se calcularán de igual manera que para el héroe, multiplicando por dos el valor del atributo `defense`.

Una vez generado el enemigo, se imprimirán por pantalla todas las características de éste. A continuación se muestra un ejemplo del formato de salida que se debe seguir:

Terminal

```
[Enemy]
Breed: Dragon
Attack: 160
Defense: 140
Health points: 280
```

Al mostrar por pantalla la raza de los enemigos se deberá hacer de la siguiente manera (respetando mayúsculas y minúsculas): `Axolotl`, `Troll`, `Orc`, `Hellhound` y `Dragon`.

Después de mostrar la información del enemigo, se ofrecerá un menú de opciones para que el jugador decida la acción a realizar, usando para ello la función `showMenu()` proporcionada en `prac1.cc`:

Terminal

```
[Options]
1- Fight
2- Run away
3- Special
4- Report
q- Quit
Option:
```

Las opciones válidas que puede introducir el usuario son los números del `1` al `4` y la letra `q`. Si la opción elegida no es ninguna de éstas (por ejemplo un `9`, una `x` o un `;`) se mostrará el siguiente mensaje de error y después se volverá a enseñar el menú de opciones:

Terminal

```
ERROR: wrong option
```

Cuando el usuario elige una opción correcta, se debe ejecutar el código asociado a dicha opción. Al finalizarla, si el héroe sigue vivo, se volverá a mostrar el menú principal y a pedir otra opción hasta que el usuario decida salir del programa utilizando la opción `q` o el héroe muera.

8. Opciones

En los siguientes apartados se describe el funcionamiento que deberá tener cada una de las cuatro opciones que se muestran en el menú principal del programa.

8.1. Fight

Esta acción se activará cuando el jugador elija la opción `1` del menú principal, permitiendo al jugador atacar al enemigo haciendo uso de la función `fight()` definida en el fichero `prac1.cc` proporcionado.

Dentro de `fight()` se simulará el lanzamiento de un dado de 20 caras, cuya puntuación **se multiplicará por cinco** y se sumará al valor almacenado en el ataque del usuario. A continuación se simulará el lanzamiento de otro dado de 20 caras, cuya puntuación **se multiplicará por cinco** y se sumará al valor almacenado en la defensa del enemigo. La defensa total del enemigo (es decir, el valor `defense` sumado al valor sacado con el dado del enemigo multiplicado por cinco) se restará del ataque total del héroe (es decir, el valor `attack` sumado al valor sacado con el dado del héroe multiplicado por cinco). El resultado de esta operación serán los puntos de daño (`hit points`) causados al enemigo, que se restarán de sus puntos de vida.

Por ejemplo, un héroe con ataque `100` saca un `18` con el dado, que al multiplicarse por cinco se transforma en `90`, lo que hace un total de `190` puntos de ataque. El enemigo, con `160` puntos de vida, tiene defensa `80` y saca un `8` con el dado, que al multiplicarse por cinco da como resultado `40`, por lo que obtiene `120` puntos de defensa. Al restar estas dos cantidades se obtienen `70` puntos de daño, que se restarán de los puntos de vida del enemigo, quedando en `90`. Esta secuencia de acciones se mostrará por pantalla con el siguiente formato:

Terminal

```
[Hero -> Enemy]
Attack: 100 + 90
Defense: 80 + 40
Hit points: 70
Enemy health points: 90
```

Si los puntos de vida resultantes del enemigo son menores o iguales a `0` habrá muerto y se mostrará a continuación el mensaje:

Terminal

```
Enemy killed
```

Al matar a un enemigo deberán sumarse los puntos de experiencia conseguidos al campo `exp` del héroe. La siguiente tabla muestra cuántos puntos de experiencia proporciona cada enemigo que derrotamos en función de su raza:

Raza	Puntos
AXOLOTL	100
TROLL	150
ORC	200
HELLHOUND	300
DRAGON	400

Al morir un enemigo, se generará uno nuevo repitiendo el proceso descrito en la Sección 7 (mostrando también la información del mismo) y se volverá a presentar el menú principal para que el jugador elija la opción a ejecutar.

Si el enemigo no ha muerto, será su turno de atacar. Se repetirá el procedimiento pero a la inversa: se lanzará primero el dado del enemigo, sumando el valor multiplicado por cinco a sus puntos de ataque, luego el dado del jugador, sumando el valor multiplicado por cinco a sus puntos de defensa, y la diferencia entre ambos será el daño causado que se restará de los puntos de vida del héroe.

La información mostrada por pantalla será similar a la anterior, pero variando los roles del atacante y defensor. Por ejemplo, si el enemigo tiene un ataque de 60 y saca un 10 con el dado, y el héroe tiene 180 puntos de vida, una defensa de 90 y saca 3 con el dado, la información a mostrar sería:

```
Terminal
[Enemy -> Hero]
Attack: 60 + 50
Defense: 90 + 15
Hit points: 5
Hero health points: 175
```

Si los puntos de salud del héroe son mayores que 0, significa que ha sobrevivido al ataque y continúa la partida, volviéndose a mostrar el menú principal para que el jugador decida la siguiente acción a tomar (volver a atacar, huir, etc.). Si los puntos de vida del héroe son menores o iguales a 0 habrá muerto y se mostrará el mensaje:

```
Terminal
You are dead
```

Esto implica el **final de la partida**, mostrándose por pantalla antes de salir del programa la misma información que proporciona la opción `Report` del menú, tal y como se describe en la Sección 8.4.

- Si al restar los puntos de defensa de los puntos de ataque se obtuviera una cantidad menor o igual a 0, significa que el ataque no ha tenido efecto y no se restará ningún punto de vida al personaje, ya sea el héroe o el enemigo. Por pantalla, en la línea `Hit points` se mostrará un 0, nunca un número negativo
- Si el número de puntos de vida tras el ataque es negativo (tanto en el caso del héroe como en el caso del enemigo) en la línea `Enemy/Hero health points` se mostrará un 0, nunca un número negativo

8.2. Run away

Si pensamos que el enemigo es muy fuerte para nosotros, tenemos la posibilidad de huir del combate, tanto si hemos realizado ya algún ataque contra él como si no. Al elegir la opción `Run away`, que se activa cuando el usuario introduce el valor 2 en el menú principal, se mostrará el siguiente mensaje por pantalla:

```
Terminal
You run away
```

El enemigo actual se descartará y se generará uno nuevo, comenzando nuevamente el proceso descrito en la Sección 7. Esta opción de huida podrá usarse un máximo de 3 veces a lo largo de una partida, tal y como se indicó en la Sección 4.3, y nunca se podrá huir dos veces seguidas sin que haya un ataque de por medio. Cada vez que el jugador huya, deberá decrementarse el valor del campo `runaways` del registro `Hero`. En caso de que el jugador haya usado las tres opciones de huida (es decir, que `runaways` valga 0), o trate de huir de dos enemigos de manera consecutiva, en lugar del mensaje anterior se mostrará este:

```
Terminal
ERROR: cannot run away
```

A continuación se mostrará de nuevo el menú principal para que el usuario pueda elegir una nueva opción.

8.3. Special

El héroe tiene un ataque especial que pueden usar una única vez durante la partida. Esta acción se activa cuando el usuario elige la opción `3` en el menú principal. El efecto de realizar el ataque especial consiste en triplicar el valor obtenido al lanzar el dado de 20 caras durante la fase de ataque (para la fase de ataque debes usar la función `fight()`, tal y como hacías en la Sección 8.1). De esta manera, si el jugador tiene `90` puntos de ataque y saca un `10` con el dado, la potencia de ataque final será de `90 + 10 * 5 * 3 = 240` puntos.

Al usar el ataque especial, el campo `special` de la estructura `Hero` pasará a valer `false`, indicando que el ataque se ha gastado. Si el jugador tratara de usar de nuevo el ataque especial después de haberlo gastado, se mostrará el mensaje:

Terminal

```
ERROR: special not available
```

A continuación se mostrará de nuevo el menú principal para que el usuario pueda elegir otra opción.

8.4. Report

Esta opción mostrará un resumen del estado actual de la partida, donde se vean los datos del héroe y un resumen de los enemigos derrotados. Para mostrar esta información se usará la función `report()` proporcionada en el fichero `prac1.cc`. Se activa cuando el usuario elige la opción `4` del menú principal. El informe mostrado contendrá la siguiente información:

- `Name`: nombre del héroe
- `Attack`, `Defense` y `Health points`: puntos de ataque, defensa y vida del héroe, respectivamente
- `Special`: disponibilidad del ataque especial. Los valores posibles son `yes` si está disponible y `no` si ya se ha gastado
- `Runaways`: número de huidas que aún le quedan al héroe
- `Exp`: puntos de experiencia acumulados hasta el momento
- `Enemies killed`: número de enemigos derrotados para cada raza (`Axolotl`, `Troll`, `Orc`, `Hellhound` y `Dragon`) y el total de muertos (`Total`)

A continuación se muestra un ejemplo de salida por pantalla generado por esta función:

Terminal

```
[Report]
Name: Lord Papanatas
Attack: 90
Defense: 110
Health points: 57
Special: yes
Runaways: 2
Exp: 1000
Enemies killed:
- Axolotl: 3
- Troll: 0
- Orc: 0
- Hellhound: 1
- Dragon: 1
- Total: 5
```

En este ejemplo, el jugador ha matado a `5` enemigos en total: 3 ajolotes, 1 perro del infierno y 1 dragón. Los datos se deben mostrar siguiendo de manera exacta el orden en el que aparecen en el ejemplo. Es decir, primero se muestran los ajolotes que se han derrotado, luego los trols, etc.