Librería de Armas (y II)

Introducción

En la anterior entrega [1], vimos como crear un conjunto de armas, unas defensivas, otras ofensivas y otras, incluso, armas de fuego. Junto a las armas de fuego, se desarrollaron los mecanismos necesarios para poder dispararlas a cualquier cosa (objeto o personaje), aprovechando el mecanismo de la librería estándar atacar().

Todo eso está muy bien, pero necesitaremos poder atacar a alguien, y sobre todo, que éste nos responda y podamos pelear con él.

Y ése, precisamente, es el objetivo de esta entrega.

La siguiente parte de la librería

La siguiente parte de la librería empieza recubriendo con una pequeña capa las armas mixtas:

```
class armaBlanca
class arma_mixta
;
```

Es cierto, es cierto, no es gran cosa, pero es que para un arma blanca no necesitamos demasiado.

Bueno, ahora en serio, empezamos con la clase general de guerreros. Veremos que el combate se divide en dos partes: la defensa, que cualquier guerrero puede hacer, y el ataque, que el jugador realiza de una forma y el guerrero (PSI) de otra distinta. La primera clase se encarga de la defensa.

```
class guerrero
private
   armaBlandida nothing,
   nivel_vida 0,
   habilidad_def 0,
   habilidad_of 0,
   suerte 0,
   atacadoUltimaVez false,
   turnoAtacadoPorUltimaVez 0,

   elRestoNoAtacado [x;
      objectloop(x ofclass guerrero)
      {
         x.resetAtacado();
      }
   ],

   heSidoAtacado [;
      self.elRestoNoAtacado();
      self.atacadoUltimaVez        = true;
      self.turnoAtacadoPorUltimaVez = turnos;
   ]
```

En la parte privada, se declaran las características de los guerreros. El nivel de vida, cuando llega a cero, implica la muerte del guerrero. Además, hay una habilidad defensiva, una habilidad ofensiva, y un factor suerte. El atributo armaBlandida lleva el arma que emplea el combatiente. Los métodos elRestoNoAtacado() y heSidoAtacado() se emplean para poder llevar la

cuenta de quién ha hecho el último ataque, y, para cada combatiente, en qué turno lo hizo. No es información crucial, pero podría serlo en algún momento dado para algún uso de la librería.

```
with
   resetAtacado [;
      self.atacadoUltimaVez = false;
   ],
   devAtacadoUltimaVez [;
      return atacadoUltimaVez;
   ],
   devTurnoAtacadoPorUltimaVez [;
      return turnoAtacadoPorUltimavez;
   ],

   estaVivo [; return (self.nivel_vida > 0); ],
   devSuerte [; return self.suerte; ],
   devVida [; return self.nivel_vida; ],
   devHabDef [; return self.habilidad_def; ],
   devHabOf [; return self.habilidad_of; ],

   ponArmaBlandida [x;
      self.armaBlandida = x;
   ],
```

En estas funciones, se trata de obtener los valores de las características antes mencionadas. Nótese que éstas no pueden modificarse. Por ejemplo, la vida sólo la modifica la propia librería cuando el guerrero es atacado. No tiene sentido que sea modificado desde fuera, aunque sí podría ser modificado desde un método antesDeRecibirAtaque(), como se verá más adelante. Sí es posible modificar el arma blandida en cada momento.

```
   devArmaBlandida [x y;
      if (x == ATAQUE)
         y = buscaArmaDeAtaqueEn(self);
      else if (x == DEFENSA)
            y = buscaArmaDeDefensaEn(self);

      if (self.armaBlandida == nothing)
         self.ponArmaBlandida(y);

      return self.armaBlandida;
   ],
```

Esta función es un poco más compleja de lo que debería, porque en caso de que no haya ningún arma blandida (y hasta el momento, no se extenderá la acción atacar estándar), la busca dentro de los objetos del jugador. Puede buscar dos tipos de armas: de defensa y de ataque (de ahí la utilidad de dividir los objetos en clases).

```
   estadisticas [;
      print "Vida: ",self.devVida(),
           "^Suerte: ", self.devSuerte(),
           "^Habilidad Defensiva: ", self.devHabDef(),
           "^Habilidad Ofensiva: ", self.devHabOf(),
           "^^";
```

```
],

describeme [;
    return "Un guerrero con cara de pocos amigos ...";
],

descripcion [;
    print (string) self.describeme();
    print "^";
    self.estadisticas();
],

antes [;
    atacar: return (self.recibe_ataque());
],
```

Los métodos anteriores sirven para visualizar estadísticas y descripciones. Todos ellos pueden ser sobrescritos por clases derivadas (quizás no se desee que se impriman las estadísticas en algún caso). Por último, tenemos la parte de la librería estándar encargada de relacionar la acción atacar estándar con la capacidad de ataque del guerrero.

```
antesDeRecibirAtaque [atacante arma; return true; ],
despuesDeRecibirAtaque [atacante arma puntosperdidos; return true; ],

calculaPerdida [ atacante fuerza_arma aux;
    ! Tener en cuenta:
    !   fuerza_of del atacante
    !   fuerza_def de self
    !   fuerza del arma
    !   suerte de self

    aux = atacante.devHabOf() + fuerza_arma;
    aux = aux - self.devHabDef();
    aux = (aux - random(self.devSuerte())) + random(self.devSuerte());

    return aux;
],

alMorir [;
    " ... muerto ...";
],
```

Los métodos anteriores son o ganchos o métodos llamados desde la parte encargada de recibir el ataque. CalculaPerdida() es un método importante, ya que calcula los puntos de vida que va a perder el personaje con este ataque. AlMorir() se ejecuta cuando el personaje muere (lo que podría emplearse incluso para resucitarlo).

```
recibe_ataque [atacante elarma fuerza_arma puntos;
    ! oh, Dios mío !
    self.heSidoAtacado();

    ! quién me ataca ?
    if (atacante == nothing)
```

```
        atacante = jugador;

    ! obtener fuerza del arma
    if (atacante ofclass guerrero)
        elarma = atacante.devArmaBlandida(ATAQUE);
    else elarma = nothing;

    if (elarma != nothing
     && elarma ofclass arma)
        fuerza_arma = elarma.devOfensa();
    else fuerza_arma = 1;

    ! Llamar al gancho del ataque
    if (self.antesDeRecibirAtaque(atacante, elarma)) {
        ! Quitar la puntuacion de vida
        puntos = self.calculaPerdida(atacante, fuerza_arma);
        self.nivel_vida = self.nivel_vida - puntos;

        ! Está muerto ?
        if (self.nivel_vida <= 0) {
            self.nivel_vida = 0;
            self.alMorir();
        }
     }

    return self.despuesDeRecibirAtaque(atacante, elarma, puntos);
  ]
;
```

Lo anterior es el método principal, que se encarga de recibir el ataque de otro guerrero (o del jugador). Lo que se hace es, a partir del atacante, localizar el arma con la que realizará el ataque, y llamar al gancho antes...(). Si todo va bien (antes...() ha devuelto true), entonces se le quitan al guerrero sus puntos de vida. Si está muerto, se llama al gancho de alMorir(), y en todo caso, se llama a después...(), cuyo valor es devuelto como valor de retorno de todo el método.

```
class PSIGuerrero
class guerrero
private
  objetivo nothing,
  atacoEnTurno 0,              ! Guarda el turno en el que atacó por última vez
  ultimoTurnoAtaco false,      ! Fue el PSI del ultimo ataque ?

  anulaLosAtacantes [x;
    objectloop(x ofclass PSIGuerrero)
    {
      x.resetAtaco();
    }
  ],
  esteEsAtacante [;
    self.anulaLosAtacantes();
    self.ultimoTurnoAtaco = true;
    self.atacoEnTurno     = turnos;
  ]
```

```
with
   resetAtaco [;
      self.ultimoTurnoAtaco = false;
   ],
   devUltimoTurnoAtaco [;
      return ultimoTurnoAtaco;
   ],
   devTurnoUltimoAtaque [;
      return atacoEnTurno;
   ],
```

Lo anterior es ya la clase PSIGuerrero. Todos los guerreros tienen capacidad para defenderse, aunque no todos atacan de la misma forma. Un PSIGuerrero debe atacar a alguien "por su cuenta", mientras que el guerrero que representa al jugador ataca cuando le da la gana (sin que tengamos que hacer nada extra en la librería). Lo más importante es la propiedad "objetivo", que lleva a quien debe este PSI atacar. Además, como antes, guardamos alguna información extra para saber quien le ha pegado a quien la ultima vez.

```
   suObjetivo [; return self.objetivo; ],
   cambiaObjetivo[x; self.objetivo = x; ],

   antesAtacar [ objetivo;
      rtrue;
   ],

   despuesAtacar [ objetivo;
      rtrue;
   ],
```

Podemos cambiar el objetivo de un PSIGuerrero en cualquier momento. Como veremos, él mismo lo modifica en el caso de su muerte o la muerte del objetivo. Pueden verse, además, los ganchos de antes del ataque y después del ataque.

```
   cada_turno [;
      if (self.suObjetivo()          ~= nothing)          ! Tiene un objetivo ?
      {
       if (enQueLugar(self.suObjetivo()) == enQueLugar(self)     ! Están en el mismo lugar ?
       && self.estaVivo()
       && self.antesAtacar(self.suObjetivo()))
       {
         ! Marcar a éste como el último en atacar
         self.esteEsAtacante();

         if (self.suObjetivo() ofclass Guerrero)
         {
            ! Atacar
            self.suObjetivo().recibe_ataque(self);

            ! Dejamos de blandir el arma de ataque
            self.suObjetivo().ponArmaBlandida(nothing);

            ! Si ha muerto, nos quedamos tranquilos
            if (~~self.suObjetivo().estaVivo())
```

```
                self.cambiaObjetivo(nothing);
            }
            else <<atacar objetivo>>;

            return self.despuesAtacar(self.suObjetivo());
        }
       }
       rtrue;
   ]
has animado;
```

Finalmente, en el caso de que haya objetivo, de que el objetivo esté en el mismo sitio que el guerrero, y que antes ...() devuelva true, se plantea el ataque al objetivo, llamando a su método recibe_ataque().
En caso de que no sea un guerrero, simplemente utilizamos el mecanismo estándar de ataque de informATE.

Algunas funciones generales

Algunas funciones generales se encuentran en la parte superior del archivo combate.h. Son útiles porque automatizan algunas acciones importantes de la librería y están disponibles también para le usuario.

```
[enQueLugar x;
   if (parent(x) == nothing)
       return x;
   else return enQueLugar(parent(x));
];

[numBalasDisparadas;
   return children(LugarBalasPerdidas);
];
```

La primera función devuelve el lugar donde está un objeto. Es una llamada recursiva a sí mismo, hasta que encuentra el objeto "sin padre". La segunda devuelve el número total de balas que se han disparado (de todas las armas).

```
[ultimoPSIGuerreroAtacante x toret;
   toret = nothing;
   objectloop (x ofclass PSIGuerrero)
   {
      if (x.devUltimoTurnoAtaco()) {
         toret = x;
         break;
      }
   }
   return toret;
];

[ultimoPSIGuerreroAtacado x toret;
   toret = nothing;
   objectloop (x ofclass PSIGuerrero)
   {
```

```
        if (x.devTurnoAtacadoPorUltimaVez()) {
            toret = x;
            break;
        }
    }
    return toret;
];

[buscaArmaDeDefensaEn obj x toret;
    toret = nothing;
    objectloop (x in obj)
    {
        if ((x ofclass arma_defensiva
          || x ofclass arma_mixta)
         && ~~(x ofclass armaDeFuego))      ! Hay que dispararlas, no valen
        {
            toret = x;
            break;
        }
    }

    return toret;
];

[buscaArmaDeAtaqueEn obj x toret;
    toret = nothing;
    objectloop (x in obj)
    {
        if ((x ofclass arma_ofensiva
          || x ofclass arma_mixta)
         && ~~(x ofclass armaDeFuego))      ! Hay que dispararlas, no valen
        {
            toret = x;
            break;
        }
    }

    return toret;
];

[buscaArmaEn obj toret;
    toret = buscaArmaDeAtaqueEn(obj);
    if (toret == nothing)
        toret = buscaArmaDeDefensaEn(obj);

    return toret;
];


[armaDisparada x toret;
            toret = nothing;

            objectloop(x ofclass armadefuego) {
```

```
            if (x.devDisparadoEnTurno()==turnos) {
                toret = x;
                break;
            }
        }

        return toret;
];
```

Las anteriores funciones son todas muy parecidas. Se trata de encontrar a los últimos guerreros que han atacado o se han defendido últimamente. Las ultimas permiten localizar armas de alguna clase dentro de algún objeto (presumiblemente, otro guerrero).

Cambiando el juego

Para aprovechar las nuevas características de la librería, será necesario incluir algunas partes más. Para empezar podemos incluir una espada. Por otra parte, vamos a incluir un guardia que no nos permitirá salir hasta que le venzamos.

```
armaBlanca espadaToledana "espada" mostrador
private
      poder_ofensivo 15,
      poder_defensivo 15
with
      nombre 'espada' 'sable' 'toledo' 'toledana' 'espanola',
      with
            descripcion "Es una espada de fabricación Toledana."
has femenino;
```

La espada (un arma mixta) está ahora en el mostrador, disponible para el jugador.

```
PSIGuerrero guardia "guardia" pasillo
private
   suerte 5,
   habilidad_of 10,
   habilidad_def 6,
   nivel_vida 100
with
   nombre 'guardia' 'jaime' 'guardian' 'centinela',
   describeme [;
      if(self.estaVivo())
      {
        return "Es Jaime, el guardia de la sala de tiro.
            Probablemente esté enfadado por la pasta que no le has devuelto
            últimamente ...";
      }
      else return "Jaime la ha diñao ... gracias a tí ...";
   ],
   antesDeRecibirAtaque [ atacante elarma;
      if (elarma ~= nothing)
          print "^^", (name) atacante, " ataca a Jaime con ", (el) elarma, ".^";
      else print "^^", (name) atacante, " ataca a Jaime con sus manos desnudas.^";
```

```
        return true;
    ],
    despuesDeRecibirAtaque [ atacante elarma puntosperdidos;
        print "Jaime ha perdido ", puntosperdidos, " de vida.^";
        return true;
    ],
    antesAtacar [objetivo;
        "El guardia te mira con fiereza.";
    ]
;
```

Este es el gran protagonista de la aventura. Nuestro Jaime defenderá intrépidamente la entrada de la sala de tiro. Y cada vez que te vea, intentará zurrarte, no lo dudes. Sería interesante una combinación de esta librería y la PNJMóvil, ¿no creéis ?.

A Jaime podremos atacarle con la espada, con el cetme, la pistola o las manos desnudas. Para atacarle se utiliza el método habitual (la propia librería buscará el arma adecuada) o el de las armas de fuego, que como vimos en la entrega anterior, era diferente (cargar, apuntar y disparar).

```
Guerrero jugator
private
    suerte 5,
    habilidad_of 10,
    habilidad_def 10,
    nivel_vida 100
with
    nombre 'Luís',
    descripcion "... vaya, tienes el cuerpo de goma ...",
    antesDeRecibirAtaque [atacante elarma;
        print "El ataque ", (del) atacante, " rebota en tu cuerpo de goma.^";
        rfalse;
    ],
;

[Inicializar;
     cambiarjugador(jugator);
     guardia.cambiaObjetivo(jugador);
                localizacion = salaDeTiro;

                "Un largo y frustrante día de trabajo ...
                 esperas poder relajarte en la sala de tiro, afinando tu puntería ...^";
];
```

Como vemos, en el caso del jugador, ningún ataque puede detenerle, ya que lo impedimos con el gancho antes....(). Por último, sólo es necesario crear al jugador en la función de inicialización, además de instruir al guardia para que intente cepillarse al jugador.

... y ... ¡eso es todo, amigos!

Conclusiones

En esta entrega, hemos introducido ya a los combatientes en nuestras refriegas por la sala de tiro. Ahora el juego es un poco más interesante, de forma que podemos cargarnos al pobre Jaime con varias armas. De hecho, no se puede cruzar la puerta sino es cargándose al sufrido guardián.

Hemos vuelto a comprobar las posibilidades de los ganchos, que se ejecutan antes y

después de las acciones cruciales y permiten modificar el comportamiento de la librería, sin tocar una línea de código. Así, al protagonista en esta entrega le rebotan todos los ataques.
En la siguiente parte de la entrega, puliremos estas nuevas características y añadiremos algunas otras que conformarán un conjunto más trabajado.

Referencias

[1] Disponible en SPAC 24 (http://usuarios.lycos.es/SPAC/) y en la página (http://usuarios.lycos.es/elarquero/) del autor (baltasarq@yahoo.es).