Librería de combate (3ª parte)

Introducción

Este artículo es el final sobre la serie dedicada a la librería de combate desarrollada a lo largo de cada entrega [1], y que está disponible en la web del autor [2].
Se trata de posibilitar, de forma automatizada, la aparición de guerreros y armas, y cómo éstas pueden emplearse para el combate entre ellos. Existen recursos para el caso en el que los guerreros ataquen "algo" que no sea un guerrero, y, a su vez, el jugador puede ser un guerrero o no.
Las clases básicas son PSIGuerrero, que permite representar un PNJ con posibilidades de combate, y ArmaBlanca, para armas mixtas (ofensivas y defensivas), ArmaDeFuego, que es la clase para objetos como pistolas y rifles, y, finalmente, cualquier arma en general, que pueda imaginarse, a partir de Arma_Ofensiva y Arma_Defensiva.

Los últimos ajustes de la librería

A continuación, se especifican los últimos ajustes necesarios para que la librería funcione ya correctamente.

- En primer lugar, es necesario tener en cuenta el arma del que defiende en calculaPerdida(), cuando hasta ahora mismo no se tenía en cuenta. Por ejemplo, podemos atacar a alguien con una pistola que tiene un chaleco antibalas.

```
calculaPerdida [ atacante fuerza_arma_of fuerza_arma_def aux;
        ! Tener en cuenta:
        !    fuerza_of del atacante y fuerza de su arma ofensiva
        !    fuerza_def de self y fuerza de su arma defensiva
        !    suerte de self

        aux = self.devHabDef() + fuerza_arma_def;
        aux = aux - (atacante.devHabOf() + fuerza_arma_of);

        if (aux < 0) aux = 3;    ! El que ataca tiene ventaja

        aux = (aux - random(self.devSuerte())) + random(self.devSuerte());

        if (aux < 0) aux = 1;

        return aux;
]
```

La función es casi autoexplicativa: sumamos la habilidad defensiva del que defiende (self), y su arma defensiva, y el arma del atacante y su arma ofensiva.

Si el que defiende "gana", aún así, se le da al que ataca unos puntos de descuento, ya que es el que inicia el ataque. En todo caso, se le asigna una proporción aleatoria para que no toda la pérdida de vida tenga que ver con las armas y las habilidades de los guerreros (un arma puede ser disparada sin puntería, por ejemplo).

- Es necesario modificar recibe_ataque(). Hasta ahora, sólo reacciona a "ataca al guardia", cuando podríamos hacer "ataca al guardia con la espada".

```
recibe_ataque [atacante elarmaOf elarmaDef
```

```
            fuerza_arma_of fuerza_arma_def puntos;

        ! quién me ataca ?
        if (atacante == nothing)
            atacante = jugador;

        ! oh, Dios mío !
        self.heSidoAtacado(atacante);

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

        if (elarmaOf != nothing
         && elarmaOf ofclass arma)
            fuerza_arma_of = elarmaOf.devOfensa();
        else fuerza_arma_of = 0;


        ! obtener fuerza del arma de self (defensa)
        elarmaDef = self.devArmaBlandida(DEFENSA);
        if (elarmaDef != nothing
         && elarmaDef ofclass arma)
            fuerza_arma_of = elarmaDef.devDefensa();
        else fuerza_arma_of = 0;


        ! Llamar al gancho del ataque
        if (self.antesDeRecibirAtaque(atacante, elarmaOf, elarmaDef)) {

            ! Quitar la puntuacion de vida
            puntos = self.calculaPerdida(atacante,
fuerza_arma_of, fuerza_arma_def
  );

            self.nivel_vida = self.nivel_vida - puntos;

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

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

Así, será necesario pasarle "nothing" (ó 0), como parámetro "otro" a recibe_ataque(), para señalar

cuando el ataque se efectúa sin especificar armas.
Con pasarle "otro", será suficiente, ya que valdrá nothing si el jugador ha especificado un arma o no. Nótese que la rutina "recibe_ataque()" sólo será llamada desde aquí cuando es el jugador el que la invoca (es decir, cuando es el jugador el que ataca). Los otros guerreros llaman a recibe_ataque() directamente. "nothing" como primer parámetro indica al jugador como atacante en todo caso.

```
antes [;
        atacar: return (self.recibe_ataque(nothing, otro));
],
```

Habíamos creado la gramática "AtacarCon" para especificar un ataque a alguien con un arma, cuando es posible, simplemente, extender "atacar".
La gramática quedará como: atacar <x> con <arma> ó atacar <x>. Así, sólo es necesario añadir "al final" de la gramática para "atacar", las siguientes líneas (y la gramática para "AtacarCon" ya no es necesaria):

```
Extend 'ataca' last
        * 'a//' noun 'con' held -> atacar
        * 'a//' noun 'utilizando' held -> atacar
        * 'a//' noun 'empleando' held -> atacar
        * 'a//' noun 'usando' held -> atacar
        * 'al' noun 'con' held -> atacar
        * 'al' noun 'utilizando' held -> atacar
        * 'al' noun 'empleando' held -> atacar
        * 'al' noun 'usando' held -> atacar
;
```

- Hacer que la búsqueda de armas devuelva la más poderosa es una medida más que interesante. Sería frustrante que la librería escogiera una espada para parar un ataque teniendo un escudo. O que utilizase un cuchillo para atacar a alguien cuando tenemos una pistola de rayos láser.

```
[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
        {
            if (toret ~= nothing
             && toret.devDefensa() < x.devDefensa())
                toret = x;

            if (toret == nothing)
                toret = x;
        }
    }

    return toret;
];
```

De una forma similar, buscaArmaDeAtaqueEn() es modificada para encontrar aquella con mayor capacidad.

- Por otra parte, otro detalle es que por ahora, podríamos cepillarnos a algún guerrero y el tío se quedaría tan ancho. Cuando alguien le ataque, automáticamente debe ser su objetivo, si no tiene otro (así, es posible colaborar para cargarse a un tercer guerrero, mientras de vez en cuando se debilita al otro). Es muy posible implementar un algoritmo más completo que este para los objetivos de los guerreros. Este es el más sencillo que hay, y la funcionalidad que ofrece hace que tenga un gran rendimiento.
Sin embargo, existe un pequeño problema: el jugador será de la clase guerrero, mientras que el PSI será de la clase PSIGuerrero, clase hija de la anterior. LA cuestión es que el objetivo del guerrero sólo existe en esta última clase, ya que el guerrero que representa al jugador no necesita tomar decisiones sobre cuando o a quien atacar. La solución consiste en redefinir el método "heSidoAtacado()" (ver "recibe_ataque()", más arriba), para que cuando el guerrero sea atacado, éste cambie su objetivo.

```
heSidoAtacado [atacante;

        self.elRestoNoAtacado();
        self.atacadoUltimaVez         = true;
        self.turnoAtacadoPorUltimaVez = turnos;

        if (self.suObjetivo() == nothing)
           self.cambiaObjetivo(atacante);
]
```

- Un detalle molesto, es que los guerreros sólo se pegarán entre ellos, si existen dos PSI's guerreros que se odian lo suficiente, si el jugador está en su localidad. En lugar de utilizar "cada_turno()", emplearemos el "daemon()", que se ejecuta en cada turno, independientemente del jugador. Así, conseguiremos que un guerrero pueda pelearse sin que estés presente. Lo único malo es que el daemon() hay que arrancarlo ... pero por otra parte, hay que darle un objetivo de todas formas, así que una inicialización siempre sería necesaria:

```
inicializarGuerrero [objetivo;
        self.cambiaObjetivo(objetivo);
        ArrancarDaemon(self);
],

pararGuerrero [;
        PararDaemon(self);
],

alMorir [;
        print (el) self, " ... ha muerto ...";
        self.PararGuerrero();
]
```

Con esto se cubre esta nueva necesidad. Así, para trabajar con un guerrero hay que llamar a "inicializarGuerrero()". Para que deje de funcionar, "pararGuerrero()".

El ejemplo de los tiros

Como en cualquiera de estas librerías, la mejor prueba es poner a dos PSI enfrentados. Si la librería

funciona, entonces podremos estar razonablemente seguros de que está bien diseñada.
La función inicializar() ha cambiado un poco:

```
[Inicializar;
      cambiarjugador(jugator);
      guardia.inicializarGuerrero(gendarme);
      gendarme.inicializarGuerrero(guardia);
                  localizacion = salaDeTiro;

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

Ahora, tenemos dos guerreros que se enfrentarán el uno al otro. Se encuentran en el pasillo, donde los colocamos al crearlos. El guardia sigue siendo el bueno del Jaime, el otro es un "clon" creado única y exclusivamente para atacarle.
Por otra parte, el jugador ha dejado de tener el cuerpo de goma y ya puede sufrir ataques como los demás.

```
Guerrero jugator
private
   suerte 5,
   habilidad_of 10,
   habilidad_def 10,
   nivel_vida 100
with
   nombre 'Luís',
   describeme [; return "Eres Luís, inspector de la poli.^"; ],
   antesDeRecibirAtaque [atacante elarmaOf elarmaDef;
      print "El ataque ", (del) atacante, " parece que va a ser duro.^";
      if (elarmaOf ~= nothing)
         print "Utiliza ", (el) elarmaOf, ".^";
      if (elarmaDef ~= nothing)
         print "Te defiendes con ", (el) elarmaDef, ".^";

      rtrue;
   ],
   despuesDeRecibirAtaque [atacante elarmaOf elarmaDef puntos;
      "Has perdido ", puntos, " de vida.^^";
   ],
   alMorir [;
      banderafin = 1;
      "Has muerto.";
   ]
;
```

Conclusiones

Este artículo es el último de una serie en el que, mediante una serie de clases, que permiten clasificar los elementos a representar (armas y combatientes, en este caso), de una forma jerárquica

que permite también aplicar o no algunas operaciones.
En un futuro artículo veremos alguna aplicación de esta librería.

Ejercicios

Los mensajes de cada personaje en combate salen por pantalla. Sin embargo, en la línea 250 de tiros.inf se muestra como llamar al método “decir“, que comprueba si el jugador está allí para ver la acción.

Referencias

[1] SPAC http://usuarios.lycos.es/SPAC/
[2] Baltasar, el arquero. baltasarq@yahoo.es http://usuarios.lycos.es/elarquero/