Práctica 2. Programación dinámica

Miguel Angel Chaves Perez
miguelangel.chavesperez@alum.uca.es
Teléfono: 675287145
NIF: 49071750p

December 2, 2016

1. Formalice a continuación y describa la función que asigna un determinado valor a cada uno de los tipos de defensas.

$$asignarValor(dano, ataque, rango, vida, dispersion)$$

$$(20 * daño) + (20 * ataque) + (20 * rango) + (30 * vida) + (10 * dispersion)$$

   Cada atributo de las defensas tendrán diferente valores, dichos valores se diferencian en la importancia de cada uno para que una defensa sobreviva antes los ataques de los ucos,he considerado 100 a repartir entre los atributos. El atributo de la vida lo he considerado más importante por lo tanto se lleva 30puntos, daño ataque y rango para mi punto de vista son de la misma importancia le he considerado 20 cada uno y como menos importancia la dispersion que se lleva 10 puntos

   Una vez calculado los puntos totales de cada atributo de cada defensas, se le suma todos ellos y la defensa sabrá su valor de importancia

2. Describa la estructura o estructuras necesarias para representar la tabla de subproblemas resueltos.

   Se crea un vector para guardar los distintos valores de las defensas y una matriz donde se almacenaran los valores máximos que se puede obtener dado un numero de defensas y ases.

3. En base a los dos ejercicios anteriores, diseñe un algoritmo que determine el máximo beneficio posible a obtener dada una combinación de defensas y *ases* disponibles. Muestre a continuación el código relevante.

```
green!40!blackvoid blackasignarValor(blackstd::blacklist<blackDefense*> blackdefenses,
    green!40!blackfloat* blackvalor)
{
        green!40!blackfloat blackdano=0, blackataque=0,blackrango=0, blackvida=0, blackdispersion
            =0;
        green!40!blackint blacknDefensa=0;

        green!40!blackfor(blackstd::blacklist<blackDefense*>::blackiterator blackdefensa  =
            blackdefenses.blackbegin(); blackdefensa != blackdefenses.blackend(); blackdefensa++)
        {
                blackdano = (20*(*blackdefensa)->blackdamage);
                blackataque = (20*(*blackdefensa)->blackattacksPerSecond);
                blackrango = (20*(*blackdefensa)->blackrange);
                blackvida = (30*(*blackdefensa)->blackhealth);
                blackdispersion = (10*(*blackdefensa)->blackdispersion);

                blackvalor[blacknDefensa] = blackdano + blackataque + blackrango + blackvida +
                    blackdispersion;
                blacknDefensa++;
        }
        blackvalor[0] = 9999999999;
}


        green!40!blackfloat blackmatriz[blackdefenses.blacksize()][blackases];
        green!40!blackfloat blackvalor[blackdefenses.blacksize()];
```

```
blackasignarValor(blackdefenses,blackvalor);

blackstd::blacklist<blackDefense*>::blackiterator blackdefensa = blackdefenses.blackbegin()
    ;

green!40!blackfor(green!40!blackint blacki = 0; blacki <= blackases; blacki++)
{
        green!40!blackif(blacki < (*blackdefensa)->blackcost)
                blackmatriz[0][blacki] = 0;
        green!40!blackelse
                blackmatriz[0][blacki] = blackvalor[0];
}
++blackdefensa;
green!40!blackint blacki = 1;

green!40!blackwhile(blackdefensa != blackdefenses.blackend())
{
        green!40!blackfor(green!40!blackint blackj = 0; blackj <= blackases; ++blackj)
        {
                green!40!blackif(blackj < (*blackdefensa)->blackcost)
                        blackmatriz[blacki][blackj] = blackmatriz[blacki-1][blackj];
                green!40!blackelse
                        blackmatriz[blacki][blackj] = blackstd::blackmax(blackmatriz[blacki
                            -1][blackj], blackmatriz[blacki-1][blackj-(*blackdefensa)->
                            blackcost] + blackvalor[blacki]);
        }
        ++blacki;
        ++blackdefensa;
}
```

4. Diseñe un algoritmo que recupere la combinación óptima de defensas a partir del contenido de la tabla de subproblemas resueltos. Muestre a continuación el código relevante.

```
blacki = blackdefenses.blacksize()-1;
green!40!blackint blackj = blackases-1;
blackstd::blacklist<blackDefense*>::blackiterator blackasignar = blackdefenses.blackend();
--blackasignar;

green!40!blackwhile(blacki > 0)
{
    green!40!blackif(blackmatriz[blacki][blackj] != blackmatriz[blacki-1][blackj])
    {
            blackselectedIDs.blackpush_back((*blackasignar)->blackid);
            blackj = blackj - (*blackasignar)->blackcost;
    }
    --blacki;
        --blackasignar;
}

green!40!blackif(blackmatriz[0][blackj] != 0)
    blackselectedIDs.blackpush_back((*blackasignar)->blackid);
```

Todo el material incluido en esta memoria y en los ficheros asociados es de mi autoría o ha sido facilitado por los profesores de la asignatura. Haciendo entrega de este documento confirmo que he leído la normativa de la asignatura, incluido el punto que respecta al uso de material no original.