Práctica 2. Programación dinámica

Raúl Arcos Herrera
raul.arcosherrera@alum.uca.es
Teléfono: xxxxxxxx
NIF: 77175935X

28 de noviembre de 2021

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

$$f(danno, rango, salud, ataquesporsegundo) = salud * \frac{(rango * danno)}{ataquesporsegundo}$$

   Utilizaré una función relativamente simple, donde se tiene en cuenta el ratio de daño por segundo por el ratio y la salud.

   Con esto se busca que las mejores defensas sean aquellas con mayor daño por segundo, alcance y vida.

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

   En mi caso he utilizado las siguientes estructuras:

   - **table**, matriz dinámica de float.
   - **cost**, vector de tamaño igual al numero de defensas - 1 (Debido a que la central siempre la compramos).

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.

```
//Compramos la primera defensa (Central)
   std::list<Defense*>::iterator currentDefense = defenses.begin();
   selectedIDs.push_back((*currentDefense)->id);
   ases-=(*currentDefense)->cost;
   currentDefense++;

   //Declaramos una matriz dinamica que usaremos como tabla de defensas-ases.
   float** table =  new float*[defenses.size()-1];
   //Creamos un vector para el coste (peso) de cada defensas
   int cost[defenses.size()-1];

   //Inicializamos tanto el vector de coste como la tabla TSR
   for(int i = 0; i < (defenses.size() - 1); i++)
   {
       table[i] = new float[ases];
       cost[i]=(*currentDefense)->cost;

       for(int j = 0; j < ases; j++)
           if(j < cost[i])
               //Si no es la primera defensa (Sin contar la central)
               if((*currentDefense)->id != 1)
                   table[i][j] = table[i-1][j];
               else
                   table[i][j] = 0;
           else
               //Si no es la primera defensa (Sin contar la central)
               if((*currentDefense)->id != 1)
```

```
                    table[i][j]=std::max(table[i-1][j], (table[i-1][j-cost[i]]+value((*
                        currentDefense))));
                else
                    table[i][j] = value((*currentDefense));
        currentDefense++;
    }
```

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.

```
int j = ases-2;
    int i;

    //Busqueda de la solucion
    for(i = defenses.size() - 2; i > 0 ; i--)
    {
        if(table[i][j] != table[i-1][j])
        {
            selectedIDs.push_back(i+1);
            j -= cost[i];
        }
    }
    if(i == 0 && cost[i] <= j)
    {
        selectedIDs.push_back(i+1);
        j -= cost[i];
    }
```

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.