Práctica 2. Programación dinámica

Francisco Jesus Diaz Zajara
francisco.diazza@alum.uca.es
Teléfono: 673540730
NIF: 44062267V

2 de diciembre de 2017

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

   $$f(damage, range, attacksPerSecond, health, radio) = 1.7*(damage+range+attacksPerSecond)+1.3*(\frac{health}{radio})$$

   Para calcular el valor de cada defensa he supuesto que son más importantes las defensas con mayor daño de ataque, rango y velocidad de ataque para una mejor estrategia defensiva, por eso damos una ponderación mayor a la suma de estos 3 parámetros (1,7) y el valor restante (1,3) se lo he asignado a la relación vida/radio de la defensa, ya que considero más valiosas aquellas defensas cuya vida es mayor en relación al radio que tienen, puesto que al ser más pequeñas y tener más vida pueden ser menos vulnerables a los ataques de los ucos.

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

   He utilizado una matriz como estructura principal para la tabla de subproblemas resueltos, en la cual las filas son los vectores con los valores de cada defensa y las columnas son los vectores con los costes de cada defensa.

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.

```
void mochila(std::list<Defense*> defenses, float** &TSP, unsigned int ases, float* valores,
    unsigned int* costes){

        for(int j=0; j<ases; j++){
                if(j < costes[0])
                        TSP[0][j] = 0;
                else
                        TSP[0][j] = valores[0];
        }

        for(int i=1; i<defenses.size(); i++){
                for(int j=0; j<ases; j++){
                        if(j < costes[i])
                                TSP[i][j] = TSP[i-1][j];
                        else
                                TSP[i][j] = std::max(TSP[i-1][j], TSP[i-1][j-costes[i]]+
                                    valores[i]);
                }
        }
}
```

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.

```
std::list<Defense*>::iterator it = --defenses.end();
    int i = defenses.size()-1;
        unsigned int j = ases-1;
```

```
    while(j>0 && i>0) {
        if(TSP[i][j] != TSP[i-1][j]) {  //Si el valor de la TSP de la defensa actual es
            distinto al de la anterior,
                selectedIDs.push_back((*it)->id);       //guardamos el id de la defenesa para
                     luego colocarla
                j -= (*it)->cost;       //y restamos su coste a nuestro numero de ases
                    totales
                it--;
                i--;
        }
        else{           //Si no, pasamos a la siguiente defensa.
                        it--;
                        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.