Práctica 2. Programación dinámica

Jesús Rodríguez Heras
jesus.rodriguezheras@alum.uca.es
Teléfono: 628576107
NIF: 32088516C

27 de noviembre de 2018

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

$$f(radio, rango, salud, dano, coste, dispersion, ataquesPorSegundo) = \frac{rango * dano * ataquesPorSegundo * salud}{dispersion * coste}$$

   La función que se encarga de asignar valores a las defensas la hemos planteado de forma simple tal que divide lo considerado "bueno" (rango, daño, ataques por segundo y salud) entre lo considerado "malo" (dispersión y coste).

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

   La estructura utilizada para representar la tabla de subproblemas resueltos es una matriz de tipo `float` de tantas filas como defensas hay y tantas columnas como ases tenemos a nuestra disposición.

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(float* valores, unsigned int* coste, unsigned int ases, float** matriz, std::
    list<Defense*> defenses){

    for (size_t j = 0; j <= ases; ++j) {
        if (j < coste[0]) {
            matriz[0][j] = 0;
        }else{
            matriz[0][j] = valores[0];
        }
    }

    for (size_t i = 1; i < defenses.size(); ++i) {
        for (size_t j = 0; j < ases; ++j) {
            if (j < coste[i]) {
                matriz[i][j] = matriz[i-1][j];
            }else{
                matriz[i][j] = std::max(matriz[i-1][j], matriz[i-1][j-coste[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.

```
void DEF_LIB_EXPORTED selectDefenses(std::list<Defense*> defenses, unsigned int ases, std::
    list<int> &selectedIDs, float mapWidth, float mapHeight, std::list<Object*> obstacles) {

    unsigned int costeTotal = 0;

    std::list<Defense*>::const_iterator it = defenses.begin();
    while (it != defenses.end()) {
        costeTotal += (*it)->cost;
        ++it;
    }

    if (costeTotal <= ases) {
        std::list<Defense*>::const_iterator iter = defenses.begin();
        while (iter != defenses.end()) {
            selectedIDs.push_back((*iter)->id);
            ++iter;
        }
    }else{

        // Algoritmo de la mochila

        i = defenses.size() - 1;
        int j = ases;

        std::list<Defense*>::const_iterator iterDef = defenses.end();
        --iterDef;

        // Ahora, recuperamos la combinacion de defensas

        while (i>=0 && j>0) {
            if(i>0 && matriz[i][j] == matriz[i-1][j]){
                --i;
                --iterDef;
            }else{
                selectedIDs.push_back((*iterDef)->id);
                j -= (*iterDef)->cost;
                --i;
                --iterDef;
            }
        }
    }
}
```

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.