Práctica 2. Programación dinámica

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

16 de enero de 2019

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

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

   La función que se encarga de asignar valores a las defensas la hemos planteado de forma simple tal que divide lo considerado "bueno" (daño, rango y salud) entre lo considerado "malo" (radio). Esto es debido a que queremos defensas que sean lo más fuertes posible pero que sean lo más pequeñas en comparación a su fuerza, siendo la fuerza "$dano * rango * salud$".

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

    for (int i = 1; i < defenses.size(); ++i) {
        for (int 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.

```
// ###### Config options ################


// #####################################

#define BUILDING_DEF_STRATEGY_LIB 1

#include "../simulador/Asedio.h"
#include "../simulador/Defense.h"

using namespace Asedio;

float valor(std::list<Defense*>::iterator defensa){
    return ((*defensa)->range * (*defensa)->damage * (*defensa)->attacksPerSecond * (*defensa
        )->health) / ((*defensa)->dispersion * (*defensa)->cost);
}

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

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

    for (int i = 1; i < defenses.size(); ++i) {
        for (int 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]);
            }
        }
    }
}

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*>::iterator it = defenses.begin();
    while (it != defenses.end()) {
        costeTotal += (*it)->cost;
        ++it;
    }

    if (costeTotal <= ases) {
        std::list<Defense*>::iterator iter = defenses.begin();
        while (iter != defenses.end()) {
            selectedIDs.push_back((*iter)->id);
            ++iter;
        }
    }else{
        unsigned int coste[defenses.size()];
        float valores[defenses.size()];

        // Algoritmo de la mochila

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

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

```
        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.