Práctica 2. Programación dinámica

Ignacio Rodríguez Pérez
ignacio.rodriguezperez@alum.uca.es
Teléfono: +34 697 469 718

30 de noviembre de 2020

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

   Para asignar un valor a una defensa, se ha tenido en cuenta si es un mapa *muy poblado* de obstáculos, o no. De este modo, si estamos en un mapa en el que **más de un tercio** del mismo está cubierto por obstáculos (lo cual se considerará como **un mapa con mucha densidad** de éstos), le daremos un mayor valor a las defensas que tengan más rango (ya que **empezarán a atacar antes**). Con esta filosofía, el valor de cada defensa vendría dado por:

$$f(damage, attacksPerSecond, range, proportionObstacles) = \begin{cases} \dfrac{(damage + rango * 0.17)}{0.01 * cost}, & proportionObstacles < 0.30 \\ \dfrac{(damage + rango * 0.34)}{0.01 * cost}, & proportionObstacles >= 0.30 \end{cases}$$

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

   Tal y como se explicó en clase, el algoritmo ha de añadir la primera defensa y, por ende, nuestro presupuesto (ases) debe contemplarla. Teniendo esto en cuenta, se utiliza una matriz (de números decimales), cuyas dimensiones vienen dadas por el **número de defensas** de las que disponemos tras la extracción de lo que será la base, y tantos **ases** como nos queden después de comprar la defensa previamente seleccionada.

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 evalueDefs(std::list<Defense*> &defenses, std::vector<Data> &vDatas, float
    proportionObstacles)
{
    int index = 0;
    for(auto const &it : defenses)
        vDatas[index++] = Data((it)->cost, defAbility(it, proportionObstacles), it);
}

void generateTSR(std::vector<std::vector<float> > &matValues, std::vector<Data> &vDatas,
    const unsigned int &ases)
{
    for(int j = 0; j <= ases; ++j)
        matValues[0][j] = (j < vDatas[0].cost ? 0: vDatas[0].value);

    for(int i = 1; i < vDatas.size(); ++i)
        for(int j = 0; j <= ases; ++j)
            matValues[i][j] = ( j < vDatas[i].cost ? matValues[i-1][j]:
            std::max( matValues[i-1][j], matValues[i-1][j-vDatas[i].cost]+vDatas[i].value) );
}
```

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 bestCombinationDefs(std::vector<std::vector<float> > &matValues, std::vector<Data> &
    vDatas, std::list<int> &selectedIDs, unsigned int &ases)
{
    for(int i = vDatas.size()-1; i > 0; i--)
        if(matValues[i][ases] != matValues[i-1][ases])
            selectedIDs.push_back(vDatas[i].defense->id),
            ases -= vDatas[i].cost;

    if(matValues[0][ases] != 0)
        selectedIDs.push_back(vDatas[0].defense->id);
}
```

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.