Práctica 2. Programación dinámica

Sergio Rodríguez Vázquez
sergio.rodriguezvazquez@alum.uca.es
Teléfono: +634471641
NIF: 77171745Y

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.

   En mi caso he tenido en cuenta cuatro atributos de la defensa en concreto: el daño, los ataques por segundo, la vida y el rango. Primero calculo el daño por segundo que es capaz de realizar la defensa y luego le sumo la vida y el rango.

$$f(damage, attacksPerSecond, health, range) = (damage * attacksPerSecond) + health + range$$

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

   Utilizo primero una estructura creada por mí que me relaciona una defensa con el coste asignado previamente por mi estrategia. Esta estructura se llama DefenseCost, al comenzar el problema, recorro la lista de defensas y guardo en un vector de la STL todos los objetos de tipo DefenseCost para tener así todos los correspondientes costes.

```
//Estructura DefenseCost
struct DefenseCost
{
    DefenseCost();
    DefenseCost(Defense *defense, float cost): defense_(defense), cost_(cost){}
    Defense *defense_;
    float cost_;
};

    //Bucle para rellenar los "candidatos" del problema
    //Quitamos la primera defensa porque no la tenemos que tener en cuenta

    selectedIDs.push_back((*it)->id);
    ases = ases - (*it)->cost;
    it++;


    //Recorremos la lista de defensas que nos dan y vamos creando la lista de candidatos para
         aplicar el algoritmo de la mochila
    while(it != defenses.end())
    {
        DefenseCost defenseiterated((*it), calculatecost((*it)));
        defensecosts.push_back(defenseiterated);
        ++it;
    }
```

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.

   Sólo nos haría falta aplicar el algoritmo de la mochila de la siguiente forma:

```
// sustituya este codigo por su respuesta
void DEF_LIB_EXPORTED selectDefenses(std::list<Defense*> defenses, unsigned int ases, std::
    list<int> &selectedIDs, float mapWidth, float mapHeight, std::list<Object*> obstacles)
{
    std::vector<DefenseCost> defensecosts;

    std::list<Defense*>::iterator it = defenses.begin();

    //Quitamos la primera defensa porque no la tenemos que tener en cuenta

    selectedIDs.push_back((*it)->id);
    ases = ases - (*it)->cost;
    it++;


    //Recorremos la lista de defensas que nos dan y vamos creando la lista de candidatos para
         aplicar el algoritmo de la mochila
    while(it != defenses.end())
    {
        DefenseCost defenseiterated((*it), calculatecost((*it)));
        defensecosts.push_back(defenseiterated);
        ++it;
    }

    //algoritmo de la mochila
    float F[defensecosts.size()][ases + 1];


    for(int j = 0; j <= ases; j++)
    {
        if(j < defensecosts[0].defense_->cost)
            F[0][j] = 0;
        else
            F[0][j] = defensecosts[0].cost_;
    }
    int i, j;
    for(i = 1; i < defensecosts.size(); i++)
    {
        for(j = 0; j <= ases; j++)
        {
            if (j < defensecosts[i].defense_->cost)
                F[i][j] = F[i - 1][j];
            else
                F[i][j] = std::max(F[i - 1][j],F[i - 1][j - defensecosts[i].defense_->cost] +
                     defensecosts[i].defense_->cost);
        }
    }

    i = defensecosts.size() - 1;
    j = ases;

        while(i > 0)
        {
            if(F[i][j] != F[i - 1][j])
            {
                selectedIDs.push_back(defensecosts[i].defense_->id);
                j -= defensecosts[i].defense_->cost;
            }
            --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.

   Citado anteriormente en el ejercicio 3, esta sería la parte correspondiente del código que recupera la solución del problema una vez el algoritmo de la mochila ya ha creado la matriz de subproblemas resueltos:

```
    i = defensecosts.size() - 1;
```

```
    j = ases;
        while(i > 0)
        {
            if(F[i][j] != F[i - 1][j])
            {
                selectedIDs.push_back(defensecosts[i].defense_->id);
                j -= defensecosts[i].defense_->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.