Práctica 2. Programación dinámica

Jose Manuel Barba Gonzalez
josemanuel.barbagonzalez@alum.uca.es
Teléfono: xxxxxxxx
NIF: 48899329H

28 de noviembre de 2021

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

$$f(coste, salud, daño, ataquesPorSegundo) = (salud/coste) + (ataquesPorSegundo + daño) + (rango/10)$$

   En la función para determinar la capacidad defensiva, que dada una defensa, se ha usado los parámetros descritos en "f(c,s,d,a)" determinando que la mejor proporción de una defensa se define como la relación entre la salud dividido por su coste más la suma de ataques más daño y la suma del rango entre 10.

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

   Estructura de la tabla, en la matriz abajo descrita se guarda la tabla de subproblemas resueltos del algoritmo de la mochila.

```
        std::vector< std::vector<float> > defensesAses(tlpWidth, std::vector<float>(tlpHeight));
```

   Lista de las defensas, se usa la lista de las defensas para realizar el calculo de la valoración dada por la función "defensiveCapacity".

```
        std::list<Defense*> defenses
```

   Valor del peso de cada defensa.

```
        (*itDefenses)->cost
```

   Valor de la capacidad defensiva de cada defensa, calculada individualmente por la función "defensiveCapacity".

```
        float value;
        value = defensiveCapacity(itDefenses);
```

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.

```
/**
* Función que dada una defensa devuelva su capacidad o valor defensivo. La capacidad o valor
    defensiva
* es una valoracion de los atributos health y cost mas attacksPerSecond y damage mas range.
* @param currentDefense Iterador a la defensa actual para calcular su valor.
* @return Devuelve un valor tipo float simulando la capacidad defensiva de la defensa.
*/
float defensiveCapacity(std::list<Defense*>::iterator currentDefense)
{
        return ((*currentDefense)->health / (float)(*currentDefense)->cost)
        + ((*currentDefense)->attacksPerSecond + (*currentDefense)->damage)
        + ((*currentDefense)->range / 10);
}
```

```
/**
* Funcion  que rellena la tabla de subproblemas resueltos con el algoritmo de la mochila
* @param defenses Lista de defensas disponibles para adquirir.
* @param ases Cantidad de ases/monedas para adquirir las defensas.
* @return Devuelve la tabla de subproblemas resueltos en forma de matriz, con forma de vector de
    vectores.
*/
std::vector< std::vector<float> > tableResults(std::list<Defense*> defenses, unsigned int ases)
{
        size_t tlpWidth = defenses.size();//objetos
        size_t tlpHeight = ases;//capacidad
        std::vector< std::vector<float> > defensesAses(tlpWidth, std::vector<float>(tlpHeight));
        float value;

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

        /**
        * valor = ((*currentDefense)->health / (float)(*currentDefense)->cost) + ((*currentDefense)
            ->attacksPerSecond + (*currentDefense)->damage)
        * peso = (*itDefenses)->cost
        * salttamos de la lista de defensas la primera que ya ha sido comprada.
        */
        ++itDefenses;
        for (int j = 0; j < tlpHeight; ++j)
        {
                if ( j < (*itDefenses)->cost )
                defensesAses[0][j] = 0;
                else
                defensesAses[0][j] = defensiveCapacity(itDefenses);
        }

        for (int i = 1; i < tlpWidth; ++i, ++itDefenses)
        {
                value = defensiveCapacity(itDefenses);
                for (int j = 0; j < tlpHeight; ++j)
                {
                        if (j < (*itDefenses)->cost)
                        defensesAses[i][j] = defensesAses[i-1][j];
                        else
                        defensesAses[i][j] = std::max(defensesAses[i-1][j], defensesAses[i-1][j -
                            (*itDefenses)->cost] + value);
                }
        }

        return defensesAses;
}
```

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.

```
/**
* Función que recupera la solucion de la tabla rellena, almacenando los "id" "selectedIDs"
* @param tlpResults Matriz o tabla de subproblemas resueltos previamente calculada.
* @param defenses Lista de defensas disponibles para adquirir.
* @param selectedIDs Paso por referencia de la lista de IDs seleccionado
* @return Devuelve la lista de IDs seleccionados en selectedIDs
*/
std::vector<float> pathResult(std::vector< std::vector<float> > tlpResults, std::list<Defense*>
    defenses, std::list<int> &selectedIDs)
{
        int tlpWidth = (int)tlpResults.size();
        int tlpHeight = (int)tlpResults.data()->size();
        std::vector<float> pathSolution(tlpWidth);

        auto itDefenses = --defenses.end();

        for (int i = tlpWidth-1; i > 0; --i, --itDefenses)
        {
                for (int j = tlpHeight-1; j > 0; j = j - (*itDefenses)->cost)
```

```
                {
                        if (tlpResults[i][j] != tlpResults[i - 1][j])
                        {
                                pathSolution[i] = tlpResults[i][j];
                                selectedIDs.push_back((*itDefenses)->id);
                        }
                        else
                        if (i == 1 and (tlpResults[i][j] == tlpResults[i-1][j]))
                        {
                                pathSolution[i] = tlpResults[1][j];
                                selectedIDs.push_back((*itDefenses)->id);
                        }
                }
        }

        return pathSolution;
}
```

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.