Práctica 2. Programación dinámica

Moisés Guerrero López
moises.guerrerolopez@alum.uca.es
Teléfono: xxxxxxxx

December 7, 2018

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

$$f(salud, rango, daño, dispersion, ataquesporsegundo, coste) = salud + \frac{daño * dispersion * ataquesporsegundo}{coste} * rango$$

   Como queremos que la mejor defensa tenga el valor mas alto multiplicamos el daño, la dispersión y el ataquesporsegundo entre si,como son valores cercanos a cero de esta forma obtenemos un valor mayor cuanto mayor sean estos parámetros, después queremos lo opuesto para el coste, que sea lo menor posible, luego dividimos nuestro resultado obtenido de las multiplicaciones por el coste, cuanto menor sea el coste mayor sera el resultado de la división. Valoramos el rango de las defensas también, queremos que tenga una importancia similar al daño, por lo tanto al resultado de la división lo multiplicamos por el daño y para finalizar, el valor al que damos más importancia es la salud, al resultado anterior le sumamos la salud

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

   Para representar la tabla de subproblemas resueltos usamos una matriz de filas igual al numero de defensas menos uno, no contamos el centro de extracción de recursos, y de columnas el numero de ases después de restar el coste del centro de extracción de recursos. Ademas de dos listas, una con los valores que hemos asignado a cada defensa, ordenado de menor a mayor y el coste de cada defensa ordenado segun la lista de valores, es decir, el primer coste de la lista corresponde con el coste de la defensa con el menor valor de la lista valores.

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 asignarValores(std::list<Defense*>& defenses,std::list<float>& valores)

{

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

        float valor;

        ++it;


        while(it!=defenses.end())

        {

                valor=(*it)->health+(((*it)->damage*(*it)->dispersion*(*it)->attacksPerSecond
                    )/((float)(*it)->cost))*(*it)->range;

                valores.push_back(valor);

                ++it;
```

```
        }


}

std::list<float> asignar_Orden(std::list<Defense*>& defenses,std::list<float>& valores,std::
    list<int>& orden)

{


        bool visitados[valores.size()];

        float valor;

        int select;

        std::list<float> valores_ordenados;

        std::list<float>::iterator orden_v;

        std::list<Defense*>::iterator checkDefense;

        Defense* actual;


        //std::cout<<"Tama o de valores: "<<valores.size()<<std::endl;


        for(int i=0;i<valores.size();++i)

        {

                valor=0;

                orden_v=valores.begin();

                checkDefense=defenses.begin() ;

                ++checkDefense;


                for(int j=0;j<valores.size();++j)

                {


                        if(valor<=*orden_v && !visitados[j])

                        {

                                select=j;

                                valor=*orden_v;

                                actual=*checkDefense;

                        }

                        ++checkDefense;

                        ++orden_v;

        }

        visitados[select]=true;


        orden.push_back(actual->id);

        valores_ordenados.push_back(valor);
```

```
        }

        return valores_ordenados;
}
Defense* sacar_defensa(int id,std::list<Defense*>& defenses)
{
        List<Defense*>::iterator currentDefense = defenses.begin();
        while(currentDefense!=defenses.end() && (*currentDefense)->id!=id)
        {
                ++currentDefense;
        }
        return (*currentDefense);
}

std::list<int> sacar_costes(std::list<int>& orden,std::list<Defense*> defenses)
{
        std::list<int>::iterator it=orden.begin();//La peor valorada esta al final
        std::list<int> costes;
        while(it!=orden.end())
        {
                Defense* actual=sacar_defensa(*it,defenses);//Aqui
                costes.push_back(actual->cost);
                ++it;
        }
        return costes;
}
void DEF_LIB_EXPORTED selectDefenses(std::list<Defense*> defenses, unsigned int ases, std::
    list<int> &selectedIDs
, float mapWidth, float mapHeight, std::list<Object*> obstacles) {

        //Introducimos la primera si o si

        std::list<Defense*>::iterator checkDefense=defenses.begin();
        selectedIDs.push_back((*checkDefense)->id);
        ases-=(*checkDefense)->cost;

        std::list<float> valores;
        asignarValores(defenses,valores);
```

```
std::list<int> orden;
valores=asignar_Orden(defenses,valores,orden);
std::vector<std::vector<float>> tabla;
tabla.resize(valores.size());
for(int i=0;i<valores.size();++i)
{
        tabla[i].resize(ases+1);
}

//-------------------------------------mochila
valores.reverse();//Estan introducidos de mayor a menor y los quiero al contrario
orden.reverse();

std::list<int> costes=sacar_costes(orden,defenses);
std::list<float>::iterator itv=valores.begin();
std::list<int>::iterator it=costes.begin();

for(int j=0;j<ases+1;++j)
{
        if((float)j<(*it))
        {
                tabla[0][j]=0;
        }
        else
        {
                tabla[0][j]=*itv;
        }
}

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

        ++it;
        ++itv;
        for(int j=0;j<ases+1;++j)
        {
                if(j==0)tabla[i][j]=0;
```

```
                        else

                        {

                                if(j<*it)tabla[i][j]=tabla[i-1][j];

                                else tabla[i][j]=std::max(tabla[i-1][j],tabla[i-1][j-*it]+*
                                    itv);

                        }

                }

        }


//---------------------------------
```

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.

```
//--------------------------------------recuperar_mochila
orden.reverse();

costes.reverse();

std::list<int>::iterator ito=orden.begin();//Ahora la mejor valorada esta al comienzo

std::list<int>::iterator itc=costes.begin();



for(int i=orden.size()-1;i>0;--i)

{

        Defense* actual=sacar_defensa(*ito,defenses);

        if(tabla[i][ases]!=tabla[i-1][ases])

        {

                selectedIDs.push_back(*ito);//se pasa la mejor defensa

                ases=ases-*itc;

        }

        ++ito;

        ++itc;


}

if(tabla[0][ases]>0)selectedIDs.push_back(*ito);

//----------------------------------------
}
```

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.