Práctica 2. Programación dinámica

Juan Manuel Grondona Nuño
juanmanuel.grondonanu@alum.uca.es
Teléfono: 656485032
NIF: 49193526E

26 de noviembre de 2022

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

$$f(damage, ataquesporsegundo, rango, dispersion, salud, coste) =$$
$$3 \cdot \frac{damage \cdot ataquesporsegundo}{coste} + 6 \cdot \frac{dispersion}{coste} + 6 \cdot \frac{rango}{coste} + \frac{salud}{coste}$$

   Esta fórmula suma las diferentes propiedades relativas a su coste, es decir, calcula una característica como pondría ser la salud, y la divide por su coste, y de este modo podemos ver quien nos da más salud relativo al coste, y así con todas las características y después se suman. Estas características están ponderadas, es decir, que tiene un factor multiplicativo en cada una, estas están ahí para dar más importancia a una característica que a otra, de este modo tendremos la mejor torreta relación calidad precio. Esto lo calculamos de esta manera a causa de que el dinero es un factor limitante en el número posterior de torretas, por lo que es mejor usar las mejores y más baratas para usar más torretas, mejor que pocas con un poco de más daño.

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

   Para este caso, es necesitado de una matriz llamada tabla, y un vector llamado valor. Con esto podría ser suficiente, pero en este caso para facilitar algunas operaciones sin cambiar la estructura de la lista de defensas, he creado dos vectores auxiliares, que son coste e id. Estos vectores se encargan de guardar el coste de una defensa y de su id. De esta manera ya tenemos de todos los datos necesarios para la los siguientes algoritmos sin depender de la lista de defensa que nos paran, y todos los datos se pueden recorrer fácilmente con un simple for. Además, decir que todos los vectores anteriormente mencionados tiene tantas posiciones como defensas la lista. Por otro lado, la matriz tiene tantas filas como defensas y tantas columnas como ases menos el coste de la primera torreta, ya que esta siempre tiene que colocarse primero y no entra dentro de las valoraciones.

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.

```
//meter primer torreta

selectedIDs.push_back((*defenses.begin())->id);
cost -= (*defenses.begin())->cost;

//dar valor

valorar(defenses, coste, id, valor);

//matriz dinamica
float** tabla = new float* [defenses.size()];
for(int i = 1; i < defenses.size(); ++i) {
    tabla[i] = new float[ases+1];
}

//algoritmo mochila
for (int j = 1; j <= cost; j++){
```

```
    if (j < coste[1]){
        tabla[1][j] = 0;
    }
    else{
        tabla[1][j] = valor[1];
    }
}
for (int i = 2; i < defenses.size(); i++){
    for (int j = 1; j <= cost; j++){
        if (j < coste[i]){
            tabla[i][j] = tabla[i-1][j];
        }
        else{
            tabla[i][j] = std::max(tabla[i-1][j], tabla[i-1][j-coste[i]] + valor[i]);
        }
    }
}

//funcion calorar llamada anteriormente
void valorar (std::list<Defense*> Defensas, int* p, int* id, float* v){

    int Vdamage       = 3;
    int Vrango      = 6;
    int Vdispersion = 6;
    int Vsalud      = 1;

    float* aux = new float[Defensas.size()];
    int i = 1;
    for (auto it1 = ++Defensas.begin(); it1 != Defensas.end(); it1++){
        double damageporCoste = (*it1)->attacksPerSecond * (*it1)->damage / (*it1)->cost;
        double rangoporCoste = (*it1)->range / (*it1)->cost;
        double dispersionporCoste = (*it1)->dispersion / (*it1)->cost;
        double saludporCoste = (*it1)->health / (*it1)->cost;

        aux[i++] = Vdamage * damageporCoste + Vrango * rangoporCoste + Vdispersion *
            dispersionporCoste + Vsalud * saludporCoste;
    }

    //ordenar lista

    int k = i, max = 0, imax = 0;
    auto it = Defensas.begin();
    for(int k = 1; k <= i; k++){
        it = ++Defensas.begin();
        max = 0;
        imax = 0;
        for (int j = 1; j < i; j++){

            if (aux[j] > max){
                max = aux[j];
                imax = j;
            }
        }
        v[k] = aux[imax];
        aux[imax] = 0;
        while(imax > 1){
            imax--; it++;
        }
        p[k] = (*it)->cost;
        id[k] = (*it)->id;
    }

}
```

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.

```
int i = defenses.size()-1;
while(i > 1){
    if ((tabla[i][cost] != tabla[i-1][cost])){
```

```
        cost -= coste[i];
        selectedIDs.push_back(id[i]);
    }
    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.