Práctica 2. Programación dinámica

Federico Carrillo Chaves
federico.carrilloch@alum.uca.es
Teléfono: +34615158732
NIF: 32095180Z

21 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.

   La estrategia de selección de defensas que en mi opinión mejor se adapta a la estrategia por defecto es la siguiente:

   $$f(mapWitdth, numObst, radio, health, aps) = \begin{cases} \frac{1}{radio} & si \quad numObst \geq 21 \vee mapWitdth \geq 450 \\ \frac{1}{radio} + 1.30 * health + aps & si \quad numObst < 21 \wedge mapWitdth < 450 \end{cases}$$

   Con esta estrategia calificaremos mejor las defensas con menor radio y mayor salud. Eso hará que protejan mejor a nuestra defensa principal.

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

   Nuestra tabla de subproblemas resueltos (en adelante TSR) se define de la siguiente manera:

   $$TSR_{m,n} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix} \forall m, n \in Z^+$$

   siendo m = número de defensas y n = ases.

   Dentro de ella guardaremos una serie de valores. Esos valores verán determinados por el valor o el peso de la defensa, que están guardados un std::vector de objetoMochila.
   Dicho objetoMochila es una clase donde guardaremos el valor, el coste y el identificador de cada defensa a partir de la $defensa_1$, ya que la $defensa_0$ ya está en la lista de seleccionados.

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 valueDefenses(const std::list<Defense *> &listaDefensas, std::vector<objetoMochila> &
    vectorObjetosMochila, const float &mapWidth, const unsigned int &numeroObstaculos) {...}
```

```
void selectDefenses(std::list<Defense *> defenses, unsigned int ases, std::list<int> &
    selectedIDs, float mapWidth, float mapHeight, std::list<Object *> obstacles) {
[...]
  std::vector<objetoMochila> vectorObjetosMochila;
  valueDefenses(defenses, vectorObjetosMochila, mapWidth, obstacles.size());

  float TSR[defenses.size()][ases];

  for (int i = 0; i < defenses.size(); i++) {
    for (int j = 0; j < ases; j++) {
        if (i != 0) {
            TSR[i][j] = j < vectorObjetosMochila[i].peso_ ? TSR[i - 1][j] : std::max(
```

```
                (float) TSR[i - 1][j],
                TSR[i - 1][j - vectorObjetosMochila[i].peso_] + vectorObjetosMochila[i].
                    valor_
                );
            } else {
                j < vectorObjetosMochila[0].peso_ ? (TSR[0][j] = 0) : (TSR[i][j] =
                    vectorObjetosMochila[0].valor_);
            }
        }
    }
[...]
}
```

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 selectDefenses(std::list<Defense *> defenses, unsigned int ases, std::list<int> &
    selectedIDs, float mapWidth, float mapHeight, std::list<Object *> obstacles) {
[...]
  for (int i = defenses.size() - 1; i > 0; i--) {
    for (int j = ases - 1; j >= 0; j--) {
      if (i != 0) {
          if (TSR[i][j] != TSR[i - 1][j]) {
              selectedIDs.push_back(vectorObjetosMochila[i].id_);
              j = j - vectorObjetosMochila[i].peso_ + 1;
              --i;
          }
      } else if (TSR[i][j] != 0) {
          selectedIDs.push_back(vectorObjetosMochila[i].id_);
          --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.