Práctica 1. Algoritmos devoradores

Alejandro Guitarte Fernández
alejandro.guifer@alum.uca.es
Teléfono: 601048359
NIF: 32092444S

9 de noviembre de 2022

1. Describa a continuación la función diseñada para otorgar un determinado valor a cada una de las celdas del terreno de batalla para el caso del centro de extracción de minerales.

   En mi caso, la función asigna valores en función de la distancia al obstáculo más cercano, añadiendo (sumándole) una "penalización" equivalente a la distancia a la esquina del mapa más cercano. De esta manera, cuanto menor sea la distancia al obstáculo más cercano y menor sea la distancia a la esquina más cercana, menor será valor. La función de selección eligirá la celda con mayor valor, de manera que situemos el centro de extracción en un sitio alejado de las esquinas y de los obstáculos, donde podamos construir un "anillo" de defensas que rodee al centro de extracción.

   Los parámetros que recibe son:
   - Fila y columna de la celda a valorar.
   - Tamaño de las celdas, para valorar las distancias.
   - Número de celdas a lo ancho y a lo alto, para las esquinas.
   - Lista de obstáculos.

2. Diseñe una función de factibilidad explicita y descríbala a continuación.

   La función de factibilidad recibe la defensa a colocar más la fila y columna de la casilla a comprobar; así como las dimensiones del mapa, y listas de obstáculos y defensas. También recibe una matriz de tipo bool que representa si una casilla está disponible o no (ligera optimización).

   Así comprueba en primer lugar que la casilla esté dentro del mapa, sin salirse por los bordes, y luego que al colocar la defensa en esa casilla, no se salga del mapa por ser demasiado grande. Por último, recorre las listas de obstáculos y defensas y comprueba que no se produzcan colisiones en caso de insertar la defensa en la cela indicada.

3. A partir de las funciones definidas en los ejercicios anteriores diseñe un algoritmo voraz que resuelva el problema para el caso del centro de extracción de minerales. Incluya a continuación el código fuente relevante.

```
void DEF_LIB_EXPORTED placeDefenses(bool** freeCells, int nCellsWidth, int nCellsHeight,
    float mapWidth, float mapHeight, std::list<Object*> obstacles, std::list<Defense*>
    defenses) {

    float cellWidth = mapWidth / nCellsWidth;
    float cellHeight = mapHeight / nCellsHeight;

    int maxAttemps = 1000;
    List<Defense*>::iterator currentDefense = defenses.begin();
    while(currentDefense != defenses.end() && maxAttemps > 0) {
        if(currentDefense == defenses.begin()){ //Colocar centro de extraccion --- Algoritmo
            devorador
            List<tipoCelda> C = getList0(nCellsWidth, nCellsHeight, mapWidth, mapHeight,
                obstacles);   //Conjunto de candidatos
            bool solucionado = false;
            while(!solucionado && !C.empty()){
```

```
                tipoCelda p = C.back();    //Como C esta ordenado de menor a mayor valor, la
                    funcion de seleccion saca el elemento en la ultima posicion
                C.pop_back();              //Y la sacamos de la lista de candidatos
                if(factible(p.row, p.col, freeCells, nCellsWidth, nCellsHeight, mapWidth,
                    mapHeight, obstacles, currentDefense, defenses)){
                    (*currentDefense)->position = p.position;   //Lo ponemos en la celda
                    freeCells[p.row][p.col] = false;
                    solucionado = true; //Problema solucionado
                }
            }
        }
        else{   //Colocar el resto de defensas --- Algoritmo devorador
            //...
        }

        ++currentDefense;
        maxAttemps--;
    }
}
```

4. Comente las características que lo identifican como perteneciente al esquema de los algoritmos voraces.

   El algoritmo devorador empieza después del primer if (detecta que la defensa 0 debe seguir el algoritmo del centro de extracción).
   Vemos que cumple todas las características de un algoritmo voraz. Por un lado, tenemos un conjunto de candidatos (lista de todas las celdas puntuadas con la funcion cellValue) que vamos recorriendo en un bucle hasta que lleguemos a la solución (hemos colocado el centro, caso en el cual cambiamos la flag *solucionado* a verdadero), o nos quedemos sin candidatos (celdas).
   Como vemos, contamos con una función de selección (sacar el último elemento de la lista, ya que esta está ordenada) que obtiene el mejor candidato posible, y a continuación se saca del conjunto de candidatos C.
   Por último, existe una función de factibilidad que devuelve si se puede colocar la mina en la celda devuelta por la función de selección, y si es así, se coloca.

5. Describa a continuación la función diseñada para otorgar un determinado valor a cada una de las celdas del terreno de batalla para el caso del resto de defensas. Suponga que el valor otorgado a una celda no puede verse afectado por la colocación de una de estas defensas en el campo de batalla. Dicho de otra forma, no es posible modificar el valor otorgado a una celda una vez que se haya colocado una de estas defensas. Evidentemente, el valor de una celda sí que puede verse afectado por la ubicación del centro de extracción de minerales.

   En el caso del resto de defensas, la valoración es muy simple. Simplemente coincide con la distancia al centro de extracción de minerales, para que las defensas queden lo más cerca posible del centro de extracción, formando un "anillo" compacto al rededor de este. Cabe destacar que se devuelve la distancia, y como la lista de casillas se ordenará también ascendientemente según su distancia, la función de selección eligirá el **primer** elemento de esta, es decir, aquel con la distancia **menor** al centro.

   Los parámetros de la función serán:
   - Fila y columna de la casilla a valorar.
   - Centro de extracción de minerales.
   - Tamaño de las celdas.

6. A partir de las funciones definidas en los ejercicios anteriores diseñe un algoritmo voraz que resuelva el problema global. Este algoritmo puede estar formado por uno o dos algoritmos voraces independientes, ejecutados uno a continuación del otro. Incluya a continuación el código fuente relevante que no haya incluido ya como respuesta al ejercicio 3.

```
void DEF_LIB_EXPORTED placeDefenses(bool** freeCells, int nCellsWidth, int nCellsHeight,
    float mapWidth, float mapHeight
              , std::list<Object*> obstacles, std::list<Defense*> defenses) {

    float cellWidth = mapWidth / nCellsWidth;
    float cellHeight = mapHeight / nCellsHeight;

    int maxAttemps = 1000;
    List<Defense*>::iterator currentDefense = defenses.begin();
```

```
    while(currentDefense != defenses.end() && maxAttemps > 0) {
        if(currentDefense == defenses.begin()){ //Colocar centro de extraccion --- Algoritmo
            devorador
            //Ejercicio 3
        }
        else{   //Colocar el resto de defensas --- Algoritmo devorador
            List<tipoCelda> C = getListRest(nCellsWidth, nCellsHeight, mapWidth, mapHeight,
                *(defenses.begin()));   //Conjunto de candidatos
            bool solucionado = false;
            while(!solucionado && !C.empty()){
                tipoCelda p = C.front();    //Como C esto ordenado de menor a mayor distancia
                    , la funcion de seleccion saca el elemnto en la primera posicion
                C.pop_front();              //Y la sacamos de la lista de candidatos
                if(factible(p.row, p.col, freeCells, nCellsWidth, nCellsHeight, mapWidth,
                    mapHeight, obstacles, currentDefense, defenses)){
                    (*currentDefense)->position = p.position;   //Lo ponemos en la celda
                    freeCells[p.row][p.col] = false;
                    solucionado = true; //Problema solucionado
                }
            }
        }

        ++currentDefense;
        maxAttemps--;
    }
}
```

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.