Práctica 1. Algoritmos devoradores

Claudia Soriano Roldan
claudia.sorianoroldan@alum.uca.es
Teléfono: 609939281
NIF: 45339131J

11 de noviembre de 2019

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.

   Existen dos funciones para esta tarea. -cellValueRC: Funcion que le da un valor a una celda en funcion de la distancia que haya desde esta al centro del mapa. -candidatesRC: Funcion que genera todos los candidatos para colocar el Centro de Recursos. Llama a cellValueRC para darle un valor a cada celda del mapa y despues las guarda en un vector y despues ordena ese vector de mayor puntuación a menor puntuación.

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

   La función funfact, comprueba que no vaya a chocar contra ningún objeto del mapa (el centro de recursos, otras torretas o un obstáculo) y comprueba que no se salga del propio mapa

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)
{
        //PARTE ALGORITMO CENTRO RECURSOS
    int maxAttemps = 1000;
    List<Defense*>::iterator currentDefense = defenses.begin();

    Candidates candidatesRCC = candidatesRC(nCellsWidth,nCellsHeight,mapWidth / nCellsWidth,
        mapHeight / nCellsHeight);
    Cell possible;
    bool solution = false;

    while(!solution && !candidatesRCC.empty())
                {

      possible = *candidatesRCC.begin();

      candidatesRCC.erase(candidatesRCC.begin());

      if(funfact((*currentDefense)->radio,possible.x,possible.y,mapWidth / nCellsWidth,
          mapHeight / nCellsHeight,mapWidth,mapHeight,obstacles,defenses))
                      {
        solution = true;

        (*currentDefense)->position = CellToPos(possible.x,possible.y,mapWidth / nCellsWidth,
            mapHeight / nCellsHeight);
      }
    }
```

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

   Conjunto de Candidatos: candidatosCER, un vector de Cell que se ordenan de mayor a menor por su valor, asi todas las extracciones del vector seran de coste elemental.

Conjunto de Candidatos Seleccionados: Cell possible, que contiene al candidato seleccionado en cada iteración del algoritmo.

Funcion de Solucion: variable solution, es un bool que indica si se ha conseguido colocar el centro de recursos.

Funcion de Factibilidad: funfact(), comprueba que no se salga del mapa y no choque con obstaculos en el mapa.

Objetivo: elegir la posicion que tenga maxima puntuacion para colocar el centro de recursos.

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.

   candidatesD:Genera los candidatos para colocar las defensas. Llama a cellValueRC para calcular el valor de cada celda junto con sus coordenadas en un vector. Ordena ese vector de mayor a menor.

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.

```
//PARTE ALGORITMO DEFENSAS
    ++currentDefense;
    Candidates candidatesDE = candidatesD((*defenses.begin())->position, nCellsWidth,
        nCellsHeight, mapWidth / nCellsWidth, mapHeight / nCellsHeight);

    while(currentDefense != defenses.end() && maxAttemps > 0 && !candidatesDE.empty()){

      possible = *candidatesDE.begin();

      candidatesDE.erase(candidatesDE.begin());

      if(funfact((*currentDefense)->radio,possible.x,possible.y,mapWidth / nCellsWidth,
          mapWidth / nCellsWidth,mapWidth,mapHeight,obstacles,defenses))
                        {
                                (*currentDefense)->position = CellToPos(possible.x,possible.y
                                    ,mapWidth / nCellsWidth,mapHeight / nCellsHeight);
                                ++currentDefense;
        --maxAttemps;
      }
      else
        --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.