Práctica 1. Algoritmos devoradores

Manuel Diaz Gil
manuel.diazgil@alum.uca.es
Teléfono: 667361517
NIF: 45382945N

19 de noviembre de 2017

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.

   La funcion diseñada valora la celda con la suma de un valor en funcion de su cercania al centro,y la distancia de las piedras que le rodean,restando la distancia que hay hacia la roca y esta se le resta a la distancia del mapa (media del acho y el alto del mapa)

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

   La funcion de factibilidad comprueba que la defensa que se va a colocar, pasandole la posicion la fila y la columna dodne se va a colocar y devuelve si es posible colocarla o no, comprobando si colisiona con otra defensa, un obstaculo, o se sale de los limites del 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.

```
//struct celda para almacenar en la cola de prioridad
     struct celda
    {
        int row,col;
        float valor;
        celda():row(0),col(0),valor(0){}
        celda(int x,int y,float v):row(x),col(y),valor(v){}


    };
//Codigo del algoritmo voraz
    float cellWidth = mapWidth / nCellsWidth;
    float cellHeight = mapHeight / nCellsHeight;
    bool colocado=false;

    celda cactual;
    std::priority_queue<celda> mceldas;
    List<Defense*>::iterator currentDefense = defenses.begin(); //Primera defensa (Generador)
    for (int i = 0; i < nCellsWidth; ++i)
    {

        for (int j = 0; j < nCellsHeight; ++j)
        {
                mceldas.push(celda(i,j,cellValue(i,j,freeCells,nCellsWidth,nCellsHeight,
                    mapWidth,mapHeight,obstacles,defenses)));

        }
    }

    while(!mceldas.empty() && !colocado){
        cactual=mceldas.top();
        mceldas.pop();
        if(factible(cactual.row,cactual.col,nCellsWidth,nCellsHeight,mapWidth,mapHeight,
            obstacles,defenses,currentDefense)){
                (*currentDefense)->position.x = (cactual.row * cellWidth) + cellWidth * 0.5f;
                (*currentDefense)->position.y = (cactual.col * cellHeight) + cellHeight * 0.5
                    f;
```

```
                (*currentDefense)->position.z = 0;
            colocado=true;
        }

    }
```

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

   Contiene un conjunto de candidatos (cola de prioridad, que contiene las celdas con el valor), se escoje al mejor candidato y si es factible, y por tanto solucion (comprobamos si es factible y colocamos la defensa).

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.

   La funcion tiene en cuenta el centro y las distancia de los obstaculos, al igual que la del centro de extraccion, pero con la diferencia de que a esto se le suma la distancia con el centro de extraccion de minerales multiplicado por 2 para que se le de mas importancia.

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.

```
std::priority_queue<celda> mceldas2;
    for (int i = 0; i < nCellsWidth; ++i)
    {
        for (int j = 0; j < nCellsHeight; ++j)
        {
                mceldas2.push(celda(i,j,cellValue2(i,j,freeCells,nCellsWidth,nCellsHeight,
                    mapWidth,mapHeight,obstacles,defenses)));
        }
    }


    while(currentDefense != defenses.end()) {
        colocado=false;
        while(!mceldas2.empty() && !colocado){
                cactual=mceldas2.top();
            mceldas2.pop();
                if(factible(cactual.row,cactual.col,nCellsWidth,nCellsHeight,mapWidth,
                    mapHeight,obstacles,defenses,currentDefense)){
                        (*currentDefense)->position.x = (cactual.row * cellWidth) + cellWidth
                             * 0.5f;
                        (*currentDefense)->position.y = (cactual.col * cellHeight) +
                            cellHeight * 0.5f;
                        (*currentDefense)->position.z = 0;
                colocado=true;
                }
        }
        ++currentDefense;

    }

#ifdef PRINT_DEFENSE_STRATEGY

    float** cellValues = new float* [nCellsHeight];
    for(int i = 0; i < nCellsHeight; ++i) {
       cellValues[i] = new float[nCellsWidth];
       for(int j = 0; j < nCellsWidth; ++j) {
           cellValues[i][j] = ((int)(cellValue(i, j))) % 256;
       }
    }
    dPrintMap("strategy.ppm", nCellsHeight, nCellsWidth, cellHeight, cellWidth, freeCells
                         , cellValues, std::list<Defense*>(), true);
```

```
    for(int i = 0; i < nCellsHeight ; ++i)
        delete [] cellValues[i];
        delete [] cellValues;
        cellValues = NULL;
```

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.