Práctica 1. Algoritmos devoradores

Juan Carlos Lucena Monje
juancarlos.lucenamonje@alum.uca.es
Teléfono: 637929532
NIF: 45899713q

15 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.

   Calculo el valor en funcion de la distancia a la que este de los obstaculos. Utilizo un incrementador para clasificar las distancias, consiguiendo que a mayor distancia haya menos valor tendra.

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

```
bool factibilidad(Defense* defense, std::list<Object*> obstacles, std::list<Defense*>
    defensesPlaced, float mapWidth, float mapHeight){
    bool factible = true;
    List<Object*>::iterator currentObstacle = obstacles.begin();
    List<Defense*>::iterator currentDefense = defensesPlaced.begin();

    if((defense->position.x + defense->radio) > mapWidth || (defense->position.x - defense->
        radio) < 0 || (defense->position.y + defense->radio) > mapHeight || (defense->
        position.y - defense->radio) < 0){
        factible = false;
    }

    while(currentObstacle != obstacles.end() && factible){
        if((_distance(defense->position, (*currentObstacle)->position) - (defense->radio + (*
            currentObstacle)->radio)) < 0){
            factible = false;
        }
        currentObstacle++;
    }

    while(currentDefense != defensesPlaced.end() && factible){
        if((_distance(defense->position, (*currentDefense)->position) - (defense->radio + (*
            currentDefense)->radio)) < 0){
            factible = false;
        }
        currentDefense++;
    }
    return factible;
}Escriba aqu  su respuesta al ejercicio 2.
```

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.

```
// sustituya este codigo por su respuesta

struct Cells{
    int row=0, col=0;
    float value=0;
};

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

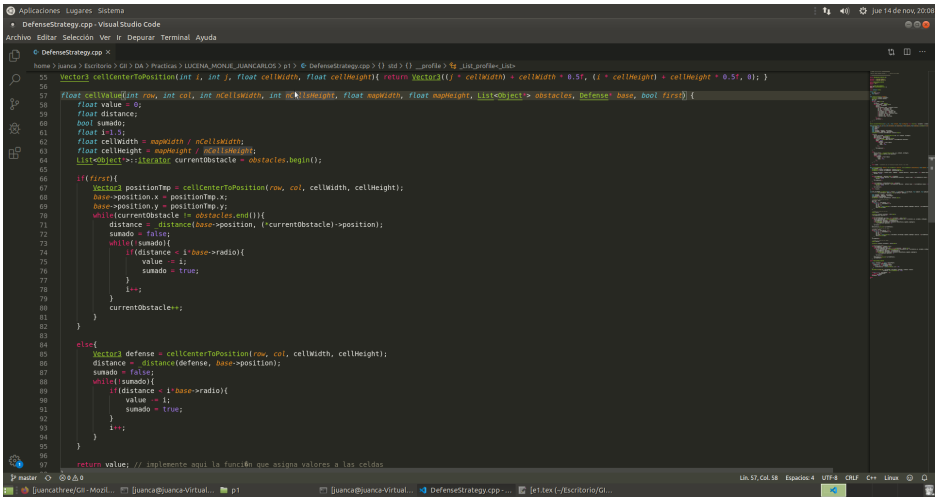

Figura 1: Estrategia devoradora para la mina

```
List<Cells> cells;
Cells aux;
for(int i=0; i < nCellsHeight; i++){
    for(int j=0; j < nCellsWidth; j++){
        aux.row = i;
        aux.col = j;
        aux.value = (int)cellValue(i,j, nCellsWidth, nCellsHeight, mapWidth, mapHeight,
            obstacles, (*currentDefense), true);
        cells.push_back(aux);
    }
}

// Ordenar la lista para la base
ordenar(cells);

List<Cells>::iterator currentCell = cells.begin();
(*currentDefense)->position.x=0;

// algoritmo voraz base
while((*currentDefense)->position.x ==0 && currentCell != cells.end()){
    Vector3 currentPosition = cellCenterToPosition((*currentCell).row, (*currentCell).col
        , cellWidth, cellHeight);
    (*currentDefense)->position.x = currentPosition.x;
    (*currentDefense)->position.y = currentPosition.y;
    if(!factibilidad((*currentDefense), obstacles, defensesPlaced, mapWidth, mapHeight)){
        (*currentDefense)->position.x = 0;
    }
    currentCell++;
}
defensesPlaced.push_back((*currentDefense));
```

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

   Una vez haya seleccionado una celda del conjunto si es valida se asigna y ya no puede excluirse y en el caso de que no sea valida no se vuevlve a tener nunca en cuenta.

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.

   Evaluo cada celda por su proximidad a la base, multiplicando el radio del centro de extraccion de minerales por un factor que va en aumento, y cuando mayor es este, menor es el valor de la celda.

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.

```
// sustituya este codigo por su respuesta
void DEF_LIB_EXPORTED placeDefenses(bool** freeCells, int nCellsWidth, int nCellsHeight,
    float mapWidth, float mapHeight
              , std::list<Object*> obstacles, std::list<Defense*> defenses) {

    // Asignacion valores para las demas
    List<Cells> cells2;
    for(int i=0; i < nCellsHeight; i++){
        for(int j=0; j < nCellsWidth; j++){
            aux.row = i;
            aux.col = j;
            aux.value = (int)cellValue(i,j, nCellsWidth, nCellsHeight, mapWidth, mapHeight,
                obstacles, (*currentDefense), false);
            cells2.push_back(aux);
        }
    }
    currentDefense++;

    // Ordenar la lista para las demas
    ordenar(cells2);

    List<Cells>::iterator currentCell2 = cells2.begin();

    // algoritmo voraz demas
    while(currentDefense != defenses.end()){
        (*currentDefense)->position.x = 0;
        while((*currentDefense)->position.x == 0 && currentCell2 != cells2.end()){
            Vector3 currentPosition = cellCenterToPosition((*currentCell2).row, (*
                currentCell2).col, cellWidth, cellHeight);
            (*currentDefense)->position.x = currentPosition.x;
            (*currentDefense)->position.y = currentPosition.y;
            if(!factibilidad((*currentDefense), obstacles, defensesPlaced, mapWidth,
                mapHeight)){
                (*currentDefense)->position.x = 0;
            }
            currentCell2++;
        }
        defensesPlaced.push_back((*currentDefense));
        currentDefense++;
    }
}
```

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.