Práctica 1. Algoritmos devoradores

Jose Luis Cumbrera Sánchez
joseluis.cumbrerasanchez@alum.uca.es
Teléfono: 633531341
NIF: 77175565P

November 17, 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.

```
        int cellValueBase(int i,int j,int nCellsWidth,int nCellsHeight,std::list<Object*>
            obstacles){
        int distance=abs(nCellsWidth/2-i)+abs(nCellsHeight/2-j);
        std::list<Object*>::iterator it;

        for(it=obstacles.begin();it!=obstacles.end();it++){
                distance=distance+abs(i-(*it)->position.x)+abs(j-(*it)->position.y);
        }
        return distance;
}
```

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

```
bool factibilidad(Defense* currentDefense,std::list<Defense*> defenses,std::list<Object*>
    obstacles,float mapWidth,float mapHeight,bool** freeCells,int i,int j){
        //Si la celda esta ya ocupada:
        if(freeCells[i][j]==false){
                return false;
        }

        //Hay varias opciones de que el resultado sea negativo, que choque con un obstaculo o
             que se salga de rango.
        //Sale de rango (0 o max posible):
        if(currentDefense->position.x-currentDefense->radio<=0 || currentDefense->position.x+
            currentDefense->radio>=mapHeight || currentDefense->position.y-currentDefense->
            radio<=0 || currentDefense->position.y+currentDefense->radio>=mapWidth){
                return false;
        }

        //Choca contra otra estructura del juego ya colocada (obstaculo u otra defensa):
        std::list<Defense*>::iterator verify;
        for(verify=defenses.begin();verify!=defenses.end();verify++){
                float euclid=_distance((*verify)->position,currentDefense->position);

                float radio=(*verify)->radio+currentDefense->radio;

                if(radio>=euclid){
                        return false;
                }
        }

        std::list<Object*>::iterator v;
        for(v=obstacles.begin();v!=obstacles.end();v++){
                float euclid=_distance((*v)->position,currentDefense->position);

                float radio=(*v)->radio+currentDefense->radio;
```

```
                if(radio>=euclid){
                        return false;
                }
        }

        //Si se llega hasta aqui enhorabuena, es factible.
        return true;
}
```

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
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;

        Celda Values[nCellsWidth*nCellsHeight];
        std::list<Celda> Ordenadas;

        int aux=0;
        int p,q;

        for(int i=0;i<nCellsWidth;i++){
                for(int j=0;j<nCellsHeight;j++){
                        Vector3 x=cellCenterToPosition(i,j,cellWidth,cellHeight);
                        positionToCell(x,p,q,cellWidth,cellHeight);
                        Values[aux].x_=x.x;
                        Values[aux].y_=x.y;
                        Values[aux].valor_=cellValueBase(i,j,nCellsWidth,nCellsHeight,
                            obstacles);
                        aux++;
                }
        }

        Ordenadas=ordenar(Values,nCellsWidth,nCellsHeight);

        std::list<Defense*> colocadas;
    List<Defense*>::iterator currentDefense=defenses.begin();
        bool find=false;
        std::list<Celda>::iterator it;
        Defense* Base;


        for(it=Ordenadas.begin();it!=Ordenadas.end() && find==false;it++){
                        (*currentDefense)->position.x=(*it).x_;
                        (*currentDefense)->position.y=(*it).y_;
                        positionToCell((*currentDefense)->position,p,q,cellWidth,cellHeight);
                        if(factibilidad((*currentDefense),colocadas,obstacles,mapWidth,
                            mapHeight,freeCells,p,q)){
                                find=true;
                                Ordenadas.erase(it);
                                colocadas.push_back((*currentDefense));
                                Base=(*currentDefense);
                                currentDefense++;
                                freeCells[p][q]=false;
                        }
        }

}
```

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

   - Tiene un conjunto de candidatos (casillas).
   - Función factibilidad, para comprobar si un conjunto de candidatos es factible.
   - Disponemos de una función para determinar un valor para cada candidato.

- Disponemos de una función para asignar la mejor casilla a una 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.

```
float cellValue(int i,int j,Defense* Base,std::list<Object*> obstacles){
        float distance=abs(Base->position.x-i)+abs(Base->position.y-j);
        std::list<Object*>::iterator it;

        for(it=obstacles.begin();it!=obstacles.end();it++){
                distance=distance+abs(i-(*it)->position.x)+abs(j-(*it)->position.y);
        }
        return distance;
}
```

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){
        aux=0;

        for(int i=0;i<nCellsWidth;i++){
            for(int j=0;j<nCellsHeight;j++){
                Vector3 x=cellCenterToPosition(i,j,cellWidth,cellHeight);
                positionToCell(x,p,q,cellWidth,cellHeight);
                Values[aux].x_=x.x;
                Values[aux].y_=x.y;
                Values[aux].valor_=cellValue(i,j,Base,obstacles);
                aux++;
            }
        }

        Ordenadas=ordenar(Values,nCellsWidth,nCellsHeight);

        while(currentDefense!=defenses.end()){
            find=false;
            for(it=Ordenadas.begin();it!=Ordenadas.end() && find==false;it++){
                (*currentDefense)->position.x=(*it).x_;
                (*currentDefense)->position.y=(*it).y_;
                positionToCell((*currentDefense)->position,p,q,cellWidth,cellHeight);
                if(factibilidad((*currentDefense),colocadas,obstacles,mapWidth,mapHeight,
                    freeCells,p,q)){
                    find=true;
                    Ordenadas.erase(it);
                    colocadas.push_back((*currentDefense));
                    currentDefense++;
                    freeCells[p][q]=false;
                }
            }
        }
}
```

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.