# Práctica 1. Algoritmos devoradores

Luis Fernando Torres Moya
luis.torresmo@alum.uca.es
Teléfono: 673346473
NIF: 32093709S

13 de noviembre de 2021

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.

   Es muy importante el matíz de que la celda esta destinada a ser para el centro de recursos, esta será nuestra primera forma de valorar una celda. El centro de recursos es la primera "defensa" que se instalara en el mapa. Esto viene en la documentacion de la Práctica 0 y sera nuestra valoracion principal a la hora de añadir puntuación a las celdas.

   Ahora bien, ¿como se implementaría?. Nos surge una duda, el mapa. El mapa es una matriz, y no sabemos la posición en esta matriz de la celda a tratar. Vamos a usar una función de la que se habla en el apartado de "preguntas frecuentes" en el campus. Con esta función conseguiremos obtener una vision mas clara de la celda con la que estamos trabajando.

   ¿Que factores vamos a tener en cuenta a la hora de decidir si una celda es mas valida para la colocacion del centro de recoleccion de recursos que otra? Pues muy claramente, los obstaculos que tendra al rededor. Estos seran la principal determinacion de la funcion a la hora de asignar un valor a la celda. El valor sera la distancia que hay desde la celda a la posicion del obstaculo. Otra consideracion a tener en cuenta (aunque no tan importante) a la hora de valorar la celda, sera la posicion general en el mapa. Por ejemplo, si una celda esta mas centrada en el mapa, sera mas valida. Esto lo implementamos de forma similar, ya que tenemos una variable en la clase la cual determina la distancia de los bordes.

   Con estos dos factores valorados, se colocaria el centro de obtencion de recursos en la zona mas optima, es decir, la que mas puntos tenga a la hora de elegirla.

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

   La funcion de factibilidad es sencilla, simplemente tenemos que comprobar dos cosas para saber si una celda es factible para ser usada por el usuario:

   -Que no este fuera de los limites establecidos del mapa. -Que no sea una celda ya ocupada por un obstaculo o por una defensa ya puesta.

   Pues bien, veamos que hacer para que se cumplan estas dos condiciones. Primero usaremos la misma funcion que en el apartado anterior para crear una variable Vector3 la cual nos indique la celda en la cual nos encontramos, y con eso nos aseguraremos de que esta dentro del mapa comprobando si la posicion es mayor que el "mapWidth" y el "mapHeigth". Si alguna de estas se cumple, ponemos false al booleano "esValido"(el cual usaremos para devolver false si no es factible y true si lo es).

   Segunda parte, que no este ocupado. Esto es un poco mas intrinseco. En ambos (Defensas y Obstaculos) crearemos un iterador para que recorra todos los elementos de la lista y asi comprobar mediante bucles "if" si la posicion de algunas defensas u obstaculos coincide con la posicion de la celda en la cual nos encontramos actualmente. Una vez comprobado todo devolveremos el booleano que se ha creado al principio ya sea con true si es factible o con false si no lo es, dependiendo si ha encontrado algun fallo.

Figura 1: Estrategia devoradora para la mina

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.

```
bool factibilidad_centroExtraccion(int row, int col, Defense* defensa, List<Object*>
    obstacles, List<Defense*> defenses,
                bool **freeCells,float mapHeight, float mapWidth,int nCellsWidth,
                int nCellsHeight) {

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

    bool is_factc = true;

    Vector3 cellInPosition = cellCenterToPosition(row, col, cellWidth, cellHeight);

    List<Object*>::iterator currentObstacle = obstacles.begin();
    while (currentObstacle != obstacles.end()){
        if((*currentObstacle)->position.subtract(cellInPosition).length()<0){
            is_factc = false;
        }
    }
    return is_factc;
}
```

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

   La principal caracteristica que lodefinecomo un algoritmo devorador es el conjunto de candidatos que va a ir "devorando", es decir eliminando para seleccionar el que mejor se adecue a la estrategia devoradora que hayamos elegido.

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 row, int col, bool** freeCells, int nCellsWidth, int nCellsHeight
        , float mapWidth, float mapHeight, List<Object*> obstacles, List<Defense*> defenses,
            List<Map*> map) {
    float value = 0;

    float cellWidth = mapWidth / nCellsWidth;
    float cellHeight = mapHeight / nCellsHeight;
    //Vamos a revisar las celdas de alrededor para ver que hay y si es una buena celda
    //Los criterios a seguir seran:
    //  -Cuanto mas cerca este de una defensa mejor
    //  -Cuanto mas lejos este de un obstac lo mejor
    //Antes que nada debemos determinar la celda, eso lo haremos con la funcion que se nos
        proporciona en preguntas frecuentes.

    Vector3 cellInPosition = cellCenterToPosition(row, col, cellWidth, cellHeight);

    //El primer factor de valor, que est  lejos de un obstaculo
    List<Object*>::iterator currentObstacle = obstacles.begin();
    while (currentObstacle != obstacles.end()){
        //Como es mejor cuanto mas lejos este de un obstaculo mejor, esta distancia se sumara
             al value
        value += (*currentObstacle)->position.subtract(cellInPosition).length();
        ++currentObstacle;
    }

    //El segundo factor de valor
    //Usamos el cellCenterTo position para buscar la celda mas cercana al centro,
    //  con esto ya podemos comparar la posicion de la celda con la del centro
    //  Cuanto mas cerca este, mejor. Entonces invertimos la distancia y ese es el puntuaje.
    int mapCenterAuxW = mapWidth / 2;
    int mapCenterAuxH = mapHeight / 2;
    Vector3 mapCenter = cellCenterToPosition (mapCenterAuxW, mapCenterAuxH, mapWidth,
        mapHeight);
```

```
    value += 1/(mapCenter.subtract(cellInPosition).length());
    //Hasta aqui seria la implemetacion del primer ejercicio, pero como no hay que
        implementarla
    //  usare esta para a adirle las variables que hace que esta funcion sea para darle
        valores
    //  a las celdas de defensa.
    //El primer criterio para la colocacion de una defensa sera que haya mas defensas cerca
    List<Defense*> :: iterator currentDefense = defenses.begin();
    while (currentDefense != defenses.end()){
        value += 1/((*currentDefense)->position.subtract(cellInPosition).length());
        ++currentDefense;
    }
        return value; // implemente aqui la funcin que asigna valores a 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.

```
// sustituya este codigo por su respuesta
void placeDefenses(...) {

    List<Defense*>::iterator currentDefense = defenses.begin();
    while(currentDefense != defenses.end() && maxAttemps > 0) {

        (*currentDefense)->position.x = ((int)(_RAND2(nCellsWidth))) * cellWidth + cellWidth
            * 0.5f;
        ...
        ++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.