Práctica 1. Algoritmos devoradores

Francisco Jesus Diaz Zajara
francisco.diazza@alum.uca.es
Teléfono: 673540730
NIF: 44062267V

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.

   Para otorgar valores a las celdas, en primer lugar, vamos recorriendo el mapa por cuadrantes, y según su proximidad al centro del mapa vamos aumentando el valor de dichas celdas. De esta forma la defensa principal será situada en el punto más centrico del mapa.

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

   Mi función esFactible se encarga de comprobar que la posición de una celda es válida para colocar una defensa dada en ella. Para ello comprobamos en primer lugar que la celda no se encuentra en una posición fuera del mapa y si se da este caso devolvemos false. A continuación comprobamos que la defensa dada no colisiona con ninguna otra defensa que haya sido colocada antes, para cual aplicamos la fórmula de la distancia euclídea. Si dicha distancia entre las posiciones de ambas defensas es menor que la suma de sus radios, significa que hay colisión, por tanto devolvemos false. Por último, utilizamos el mismo método que antes para comprobar si hay colisión con un obstáculo 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.

```
List<Celda>::iterator celdaActual = listaCeldas.begin();
        List<Defense*>::iterator defensaActual = defenses.begin();
        bool colocada=false;
        int col, fil;

        while(celdaActual != listaCeldas.end() && colocada==false){

                seleccionCelda(matrizCeldas, nCellsWidth, (*celdaActual));

                if(esFactible(obstacles, defensasColocadas, mapWidth, mapHeight, (*
                    defensaActual), (*celdaActual))){

                        (*defensaActual)->position.x = (*celdaActual).pos.x;
                        (*defensaActual)->position.y = (*celdaActual).pos.y;
                        (*defensaActual)->position.z = (*celdaActual).pos.z;
                    defensasColocadas.push_back((*defensaActual));
                    colocada = true;

                col = (*celdaActual).id % nCellsWidth;
                fil = (*celdaActual).id / nCellsWidth;
        }
        celdaActual++;
    }
```

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

   Conjunto de candidatos: las celdas del mapa.

   Un conjunto de candidatos seleccionados: celdas seleccionadas.

   Función solución: comprobamos que la defensa no haya sido colocada, e iteramos las celdas candidatas hasta que dicha defensa sea colocada, poniendo a true nuestra variable booleana.

Función de selección: recorremos nuestra matriz de celdas buscando siempre la celda con mayor puntuación, así nos aseguramos de escoger siempre el candidatos más idóneo para nuesra solución.

Función de factibilidad: si la celda seleccionada cumple los requisitos explicados anteriormente en el ejercicio 2, entonces es factible.

Función objetivo: colocar las defensas.

Objetivo: defender la defensa principal el mayor tiempo posible antes de su destrucción por los ucos.

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.

   En este caso utilizo en primer lugar el mismo método para otorgar valores a las celdas que para el caso de la defensa principal, pero con la diferencia de que además comprobamos la distancia entre la defensa principal y las secundarias adyacentes y otorgamos un valor a la celda según esa distancia. Es decir, cuánto más cerca esté de la defensa principal, mayor valor tendrá esa celda para colocar una defensa secundaria. Así conseguimos que la base quede rodeada del resto de defensas.

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.

```
        celdaActual = listaCeldas.begin();
        defensaActual = ++defenses.begin();

        while(defensaActual != defenses.end() && celdaActual != listaCeldas.end()) {

                seleccionCelda(matrizCeldas, nCellsWidth, (*celdaActual));

                if(esFactible(obstacles, defensasColocadas, mapWidth, mapHeight, (*
                    defensaActual), (*celdaActual))){

                        (*defensaActual)->position.x = (*celdaActual).pos.x;
                        (*defensaActual)->position.y = (*celdaActual).pos.y;
                        (*defensaActual)->position.z = (*celdaActual).pos.z;
            defensasColocadas.push_back((*defensaActual));
                        defensaActual++;
                }
                celdaActual++;
        }
```

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.