# Práctica 1. Algoritmos devoradores

Alejandro Serrano Fernandez
ale.serranofer@alum.uca.es
Teléfono: 640217690
NIF: 20501318S

8 de noviembre de 2020

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 el caso del centro de extracción de materiales he seguido la siguiente estrategia. Aquellas celdas que se encuentren cerca del obstáculo con mayor radio, tendrán un mayor valor. En cambio, si estos obstáculos se encuentran cerca del borde del mapa, se escogerá aquel obstáculo que más al centro del mapa se encuentre. Para determinar si un obstáculo está cerca del borde del mapa, supondré un margen de un 40 % desde los extremos del mapa. Por ejemplo, si el mapa tiene una anchura y un alto de 340, todas los obstáculos fuera del margen de un ancho y un alto de 204 se descartarán, y se valorarán aquellos obstáculos que más al centro se encuentren.

   En cuanto al diseño de la función, vamos tomando todos los obstáculos y almacenaremos aquellos que tienen mayor radio y aquellos que más al centro del mapa se encuentren. Para determinar el valor del obstáculo que más al centro se encuentra, partimos de una variable valor inicializada a 255 (el valor máximo definido por mí), y conforme más alejado esté el obstáculo del centro de mapa, se le irá restando valor.

   Una vez determinado los obstáculos candidatos, comprobaremos si el obstáculo con mayor radio se encuentra dentro del margen, tal y como expliqué anteriormente. En caso contrario escogeremos el más cercano al centro del mapa.

   Finalmente repetimos la misma estrategia anterior para determinar los valores de las celdas más cercanas al obstáculo seleccionado. Inciamos de nuevo la variable valor a 255 y vamos restándole valor conforme más alejadas estén del obstáculo seleccionado.

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

   La función de factibidad realizada recibirá una celda y será evaluda de tal manera que cumpla los siguientes requisitos:

   1) Que no se interponga con ningún obstáculo. Para ello comprobaremos que la suma del radio de la defensa a colocar y de la defensa a comparar, no sea mayor que la distancia entre ambos centros.

   2) Que no se interponga con ninguna otra defensa. Para ello comprobaremos de tal manera que el apartado anterior.

   3) Que no salga del mapa, comprobando que la suma del radio y el centro de la defensa a colocar no sobrepase los límites del mapa.

   En cuanto a su implementación, ésta función recorrerá todos los obstáculos y defensas, y determinará si la celda escogida se interpone con alguna de ellas. Para ello sumaremos el radio de la defensa a colocar y el radio de la defensa u obstáculo a comparar. Si la suma de los radios es mayor que la distancia entre ambos centros, daremos por concluido que celda no es válida. Para determinar la distancia entre ambos centros, hemos de definir previamente una función distancia() que realizará dicho cálculo a través del teorema de Pitágoras.

   Finalmente, la función determinará si la defensa no sale del mapa, para ello sumamos el radio de la defensa y el centro de la celda a comprobar. Si la suma supera los límites del mapa, entonces descartaremos dicha celda.

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.

```
    float cellWidth = mapWidth / nCellsWidth;
    float cellHeight = mapHeight / nCellsHeight;
    bool colocado = false;

    //Tomamos la primera defensa
    List<Defense*>::iterator currentDefense = defenses.begin();
    std::vector<Celda> celdasCandidatas;

    //Introducimos todas las celdas candidatas, con sus respectivos valores
    for(int i = 0; i < nCellsWidth; i++)
    {
        for(int j = 0; j < nCellsHeight; j++)
        {
                celdasCandidatas.push_back(Celda(i,j,cellValue(i, j, freeCells, nCellsWidth,
                    nCellsHeight, mapWidth, mapHeight, obstacles, defenses)));
        }
    }

    //Ordenamos el vector, para asi obtener primero las celdas con mayor valor
    sort(celdasCandidatas.begin(),celdasCandidatas.end());

    //Colocamos el centro de extraccion en la celda factible con mayor valor
    while(!celdasCandidatas.empty() && !colocado)
    {
        if(factible(celdasCandidatas.back(), freeCells, nCellsWidth, nCellsHeight, mapWidth,
            mapHeight, obstacles, defenses, defenses.begin()))
        {
                (*currentDefense)->position.x = (celdasCandidatas.back().row() * cellWidth) +
                      (0.5f * cellWidth);
                (*currentDefense)->position.y = (celdasCandidatas.back().col() * cellHeight)
                    + (cellHeight * 0.5f);
                colocado = true;
        }
        celdasCandidatas.pop_back();
    }
```

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

   La principal característica que lo define como un algoritmo voraz estriba en el uso de una estrategia que consista en elegir siempre la mejor opción con la esperanza de llegar a la solución más óptima, en nuestro caso, elegimos la celda con más valor que nos permita aguantar el máximo tiempo posible nuestra base. Para ello vamos seleccionando celdas de mayor valor y comprobaremos si es factible. En dicho caso colocaremos la defensa en dicha celda. Una vez elegida la celda, ésta nunca más vuelve a ser considerada.

   Para nuestro algoritmo, distinguimos los siguientes elementos:

   ·Un conjunto de candidatos: Celdas

   ·Una función solución: ¿Se han colocado ya todas las defensas?

   ·Una función de selección: Elige la celda con más valor de entre todas las disponibles

   ·Una función de factibilidad: Comprueba si la celda seleccionada no se interpone con alguna defensa u obstáculo, ni salga de los límites del mapa

   ·Una función objetivo: colocar las defensas en las celdas con mayor valor

   ·Objetivo: maximizar

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 estrategia a seguir para colocar las defensas es la siguiente. Una vez colocado el centro de extracción, las celdas más cercanas a su radio tienen un valor mayor, de tal manera que las defensas se colocarán alrededor de él.

Para ello he declarado una variable valor inicializada a 255, que se decrementará en función de lo alejado que se encuentre la celda del centor de extracción, es decir, dependiendo del número de celdas a las que se aleje.

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.

```
    //Coemnzamos con la siguiente defensa a colocar
    currentDefense++;
    //Eliminamos el vector con todas las celdas anteriores, para colocar las nuevas con otro
        cellValue
    celdasCandidatas.clear();

    //Introducimos las celdas con sus correspodientes valores
    for(int i = 0; i < nCellsWidth; i++)
    {
        for(int j = 0; j < nCellsHeight; j++)
        {
                celdasCandidatas.push_back(Celda(i,j,cellValue2(i, j, freeCells, nCellsWidth,
                     nCellsHeight, mapWidth, mapHeight, obstacles, defenses)));
        }
    }

    //Ordenamos el vector
    sort(celdasCandidatas.begin(),celdasCandidatas.end());

    //Vamos colocando las defensas restantes en aquellas celdas factibles con mayor valor
    while(currentDefense != defenses.end())
    {
        colocado = false;

        while(!celdasCandidatas.empty() && !colocado)
        {
                if(factible(celdasCandidatas.back(), freeCells, nCellsWidth, nCellsHeight,
                    mapWidth, mapHeight, obstacles, defenses, currentDefense))
                {
                        (*currentDefense)->position.x = (celdasCandidatas.back().row() *
                            cellWidth) +  (0.5f * cellWidth);
                        (*currentDefense)->position.y = (celdasCandidatas.back().col() *
                            cellHeight) + (cellHeight * 0.5f);
                        colocado = true;
                }
                celdasCandidatas.pop_back();
        }
        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.