Práctica 1. Algoritmos devoradores

David Luna Jurado
david.lunajurado@alum.uca.es
Teléfono: 663590384
NIF: 32098835N

12 de noviembre de 2022

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.

   El valor de las celdas para la colocación del extractor de minerales se realiza calculando la distancia de dicha casilla respecto al centro. El valor de la casilla será el inverso de la distancia, de esta forma las casillas más cercanas al centro tendrán más valor.

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

   Para la función de factibilidad tenemos en cuenta lo siguiente, una casilla será factible siempre y cuando cumpla cada una de las siguientes condiciones:

   1. La casilla esté libre.
   2. La defensa no se sale del mapa al colocarla (Posicion de la casilla mas el radio de la defensa tiene que ser menor al ancho y alto del mapa)
   3. La defensa no choca con ningún obstáculo. (La distancia con respecto a los obstaculos es menor que la suma e los radios de la defensa y el obstáculo)
   4. La defensa no choca con ninuna defensa ya colocada (La distancia de las defensa es mayor que la suma de sus radios, para cada defensa colocada).

   Esas son todas las comparaciones que realiza la función de factibilidad.

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.

```
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;
```

```
//* Asigna un valor a una casilla segun el extractor
double cellValueExtractor(Casilla *C, bool **freeCells, int nCellsWidth, int nCellsHeight, float mapWidth, float mapHeight, List<Object *>
obstacles, List<Defense *> defenses)
{
    int centroX = nCellsWidth / 2;
    int centroY = nCellsHeight / 2;

    float cellWidth = mapWidth / nCellsWidth;
    float cellHeight = mapHeight / nCellsHeight;
    Casilla Centro(centroY, centroX);
    Vector3 posCentro = CasillaToCoordenada(Centro, cellWidth, cellHeight);
    Vector3 posCasilla = CasillaToCoordenada(*C, cellWidth, cellHeight);

    double distancia = _distance(posCentro, posCasilla);


    if (distancia == 0)
    {
        return 2;
    }
    return 1 / distancia;
}
```

Figura 1: Estrategia devoradora para la mina

```
    int nCasillas = nCellsWidth * nCellsHeight;
    std::list<Casilla> Casillas;
    int n = 0;
    Casilla seleccionada;
    int contador = 0;

    // Meter todas las casillas en el conjunto de Casillas
    for (size_t i = 0; i < nCellsHeight; i++)
    {
        for (size_t j = 0; j < nCellsWidth; j++)
        {
            Casillas.emplace_back(Casilla(i, j));
        }
    }

    bool Colocadas[Casillas.size()]; //Vector de Defensas Colocadas.

    std::fill_n(Colocadas, Casillas.size(), false);
    //*Asignar valor a las celdas para el extractor de minerales
    for (Casilla &C : Casillas)
    {
        C.value = cellValueExtractor(&C, freeCells, nCellsWidth, nCellsHeight, mapWidth,
            mapHeight, obstacles, defenses);
    }
    //* Ordenar las casillas por valor
    Casillas.sort([](Casilla a, Casilla b) -> bool
                  { return a.value > b.value; });

    //* Tomamos el centro de extraccion
    Defense *centroDeExtraccion = defenses.front();
    std::list<Casilla> CasillasExtractor(Casillas);

    //* Colocamos el extractor
    while (!CasillasExtractor.empty() && !Colocadas[contador])
    {
        seleccionada = CasillasExtractor.front();

        CasillasExtractor.pop_front();

        if (esFactible(seleccionada, freeCells, cellWidth, cellHeight, mapWidth, mapHeight,
            obstacles, defenses, *centroDeExtraccion, Colocadas))
        {
            centroDeExtraccion->position = CasillaToCoordenada(seleccionada, cellWidth,
                cellHeight);
            freeCells[seleccionada.row][seleccionada.col] = false;
            Colocadas[0] = true;
        }
    }
}
```

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

   El conjunto de Candidatos son las casillas.
   El conjunto de Candidatos seleccionados son las casillas en las que se coloca una defensa.
   La función solucion en este caso es !Colocadas[0], que nos indicará si se ha colocado el extractor o no.
   La función de selección: CasillasExtractor.front() Debido a que la lista de Casillas está ordenada.
   La función de factibilidad de corresponde a esFactible(seleccionada, freeCells, cellWidth, cellHeight, mapWidth, mapHeight, obstacles, defenses, *centroDeExtraccion, Colocadas)
   La función objetivo es cellValueExtractor(&C, freeCells, nCellsWidth, nCellsHeight, mapWidth, mapHeight, obstacles, defenses); Que asigna valor a las casillas. El objetivo es maximizar el tiempo que duran las defensas en batalla.

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.

El valor de las celdas para la colocación del extractor de minerales se realiza calculando la distancia de dicha casilla respecto al centro de extracción de minerales. El valor de la casilla será el inverso de la distancia, de esta forma las casillas más cercanas al centro de extracción tendrán más valor. Aunque se le dará más valor a las casillas que estén a una distancia de dos Casillas con respecto al centro de extracción, para asegurar que haya casillas con una distancia prudencial respecto al centro.

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)
{

        float cellWidth = mapWidth / nCellsWidth;
        float cellHeight = mapHeight / nCellsHeight;
        int nCasillas = nCellsWidth * nCellsHeight;
        std::list<Casilla> Casillas;
        int n = 0;
        Casilla seleccionada;
        int contador = 0;
        //* Meter todas las casillas en el conjunto de Casillas
        for (size_t i = 0; i < nCellsHeight; i++)
        {
            for (size_t j = 0; j < nCellsWidth; j++)
            {
                Casillas.emplace_back(Casilla(i, j));
            }
        }
        bool Colocadas[Casillas.size()];
        std::fill_n(Colocadas, Casillas.size(), false);

        //Primer algorimo voraz
        //*Asignar valor a las celdas para el extractor de minerales
        for (Casilla &C : Casillas)
        {
            C.value = cellValueExtractor(&C, freeCells, nCellsWidth, nCellsHeight, mapWidth,
                mapHeight, obstacles, defenses);
        }

        //* Ordenar las casillas por valor
        Casillas.sort([](Casilla a, Casilla b) -> bool
                      { return a.value > b.value; });

        //* Tomamos el centro de extraccion
        Defense *centroDeExtraccion = defenses.front();
        std::list<Casilla> CasillasExtractor(Casillas);

        //* Colocamos el extractor
        while (!CasillasExtractor.empty() && !Colocadas[0])
        {
            seleccionada = CasillasExtractor.front();

            CasillasExtractor.pop_front();

            if (esFactible(seleccionada, freeCells, cellWidth, cellHeight, mapWidth,
                mapHeight, obstacles, defenses, *centroDeExtraccion, Colocadas))
            {
                centroDeExtraccion->position = CasillaToCoordenada(seleccionada, cellWidth,
                    cellHeight);
                freeCells[seleccionada.row][seleccionada.col] = false;
                Colocadas[0] = true;
            }
        }

        // Segundo algorimo voraz

        //* Reasignamos el valor a las casillas para el resto de defensas
```

```
        for (Casilla &C : Casillas)
        {
            C.value = cellValue(&C, freeCells, nCellsWidth, nCellsHeight, mapWidth, mapHeight
                , obstacles, defenses);
        }

        //* Reordenamos las casillas
        Casillas.sort([](Casilla a, Casilla b) -> bool
                      { return a.value > b.value; });
        contador = 0;

        //* Colocamos todas las defensas
        for (auto currentDefense = defenses.begin(); currentDefense != defenses.end();
            currentDefense++)
        {

            while (!Casillas.empty() && !Colocadas[contador])
            {

                seleccionada = Casillas.front();
                Casillas.pop_front();
                if (esFactible(seleccionada, freeCells, cellWidth, cellHeight, mapWidth,
                    mapHeight, obstacles, defenses, *(*currentDefense), Colocadas))
                {
                    (*currentDefense)->position = CasillaToCoordenada(seleccionada, cellWidth
                        , cellHeight);
                    Colocadas[contador] = true;
                    freeCells[seleccionada.row][seleccionada.col] = false;
                }
            }

            contador++;
        }
}
```

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.