

Práctica 1. Algoritmos devoradores

Jesús Rodríguez Heras
jesus.rodriguezheras@alum.uca.es
Teléfono: 628576107
NIF: 32088516C

16 de noviembre de 2018

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.

La función que evalúa las celdas para el caso del centro de extracción de minerales se llama `cellValueCentro` y recibe los siguientes parámetros:

- `float** mapa`: Matriz que representa el mapa de juego.
- `int nCellsWidth`: Número de celdas de ancho que tiene el mapa.
- `int nCellsHeight`: Número de celdas de alto que tiene el mapa.
- `float cellWidth`: Ancho de las celdas.
- `float cellHeight`: Alto de las celdas.
- `float mapWidth`: Tamaño de la anchura del mapa.
- `float mapHeight`: Tamaño del alto del mapa.
- `std::list<Defense*> defensas`: Lista de defensas del juego.

Para dar valores a las celdas de la matriz del terreno de batalla la función calcula la posición del centro de extracción respecto de la diferencia con los bordes del mapa en una variable de tipo `Vector3` llamado `posicionDefensa`.

Luego creamos una variable de tipo `Vector3`, llamado `posMax`, que estará formada por el tamaño total del mapa en sus coordenadas “X” e “Y”.

A continuación, se asigna el valor de cada celda como la resta de los módulos de ambos vectores, `posMax` y `posicionDefensa`.

El valor de las celdas es devuelto por la misma variable `mapa` por referencia.

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

La función de factibilidad diseñada recibe los siguientes parámetros de entrada:

- `float x`: Posición en el eje “X” donde se va a comprobar la factibilidad de colocar una defensa.
- `float y`: Posición en el eje “Y” donde se va a comprobar la factibilidad de colocar una defensa.
- `Defense* defensa`: Defensa que se va a comprobar.
- `std::list<Object*> obstaculos`: Lista de obstáculos del terreno de juego.
- `float mapWidth`: Tamaño de la anchura del mapa.
- `float mapHeight`: Tamaño del alto del mapa.
- `std::list<Defense*> defensas`: Lista de defensas del juego.
- `float cellWidth`: Ancho de las celdas.
- `float cellHeight`: Alto de las celdas.

Primeramente se declara una variable de tipo lógica llamada **entra** inicializada a **true**, que será lo que retornará esta función.

Luego se comprueban los siguientes factores:

- a) La defensa entra en el mapa. Es decir, no se sale por los bordes del mismo. Lo cual es calculado teniendo en cuenta el radio de la defensa y las dimensiones (ancho y alto) del mapa. En caso contrario, la variable **entra** tomaría el valor **false**.
- b) Colisión con obstáculo: Se comprueba que la defensa no colisiona con ninguno de los obstáculos del terreno de juego. En caso contrario, la variable **entra** tomaría el valor **false**.
- c) Colisión con otra defensa anteriormente colocada: Se recorre la lista de defensas comprobando que, de las ya colocadas, es decir, las $n - 1$ defensas anteriores, no colisionen con la defensa que queremos colocar. En caso contrario, la variable **entra** tomaría el valor **false**.

Finalmente se devuelve el valor de la variable **entra**.

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 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 maxAttempts = 1000;

    float** mapa = new float*[nCellsHeight];
    for (size_t i = 0; i < nCellsHeight; ++i) {
        mapa[i] = new float[nCellsWidth];
    }

    for (size_t i = 0; i < nCellsHeight; ++i) {
        for (size_t j = 0; j < nCellsWidth; ++j) {
            mapa[i][j] = cellValue(i, j, freeCells, nCellsWidth, nCellsHeight,
                mapWidth, mapHeight, cellWidth, cellHeight, obstacles, defenses);
        }
    }

    int fila = 0, columna = 0;
    float x = 0, y = 0;

    std::list<Defense*>::iterator currentDefense = defenses.begin();
    while(currentDefense == defenses.begin() && maxAttempts > 0){
        seleccion(mapa, nCellsWidth, nCellsHeight, &fila, &columna);
        x = fila*cellWidth + cellWidth*0.5f;
        y = columna*cellHeight + cellHeight*0.5f;
        if(factibilidad(x, y, (*currentDefense), obstacles, mapWidth, mapHeight,
            defenses, cellWidth, cellHeight)){
            (*currentDefense)->position.x = x;
            (*currentDefense)->position.y = y;
            (*currentDefense)->position.z = 0;

            cellValueCentro(mapa, nCellsWidth, nCellsHeight, cellWidth,
                cellHeight, mapWidth, mapHeight, defenses);

            ++currentDefense;
        }
        --maxAttempts;
    }

    while (currentDefense != defenses.end() && maxAttempts > 0) {
        seleccion(mapa, nCellsWidth, nCellsHeight, &fila, &columna);
        x = fila*cellWidth + cellWidth*0.5f;
        y = columna*cellHeight + cellHeight*0.5f;
        if(factibilidad(x, y, (*currentDefense), obstacles, mapWidth, mapHeight,
            defenses, cellWidth, cellHeight)){
            (*currentDefense)->position.x = x;
            (*currentDefense)->position.y = y;
            (*currentDefense)->position.z = 0;
        }
        ++currentDefense;
    }
}
```

```

        ++currentDefense;
    }
    --maxAttempts;
}

```

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

Escriba aquí su respuesta al ejercicio 4.

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.

Escriba aquí su respuesta al ejercicio 5.

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 código por su respuesta
void placeDefenses(...) {

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