Práctica 4. Exploración de grafos

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

25 de diciembre de 2022

1. Comente el funcionamiento del algoritmo y describa las estructuras necesarias para llevar a cabo su implementación.

   Este es un algoritmo de búsqueda de ruta A* (A star), que se utiliza para encontrar una ruta óptima entre dos puntos en un mapa.

   La función recibe como parámetros dos nodos: el nodo de origen y el nodo de destino, además de información sobre el tamaño del mapa y una matriz de costos adicionales. También recibe una lista vacía, que se llenará con la ruta encontrada.

   La función utiliza dos listas: una lista "Abierta" y una lista "Cerrada". La lista "Abierta" contiene los nodos que todavía deben explorarse y la lista "Cerrada" contiene los nodos que ya han sido explorados.

   El algoritmo comienza agregando el nodo de origen a la lista "Abierta". A continuación, se itera hasta que se haya encontrado el nodo de destino o hasta que la lista "Abierta" esté vacía o se haya alcanzado un límite máximo de iteraciones para garantizar que los nodos estén conectados.

   En cada iteración, se selecciona el nodo de menor costo en la lista "Abierta" y se lo agrega a la lista "Cerrada". Si este nodo es el nodo de destino, se recupera el camino completo utilizando la información de parentesco de cada nodo y se termina el algoritmo. Si el nodo seleccionado no es el nodo de destino, se iteran a través de sus nodos adyacentes y se calcula el costo de llegar a ellos desde el nodo actual. Si el costo es menor que el costo previamente calculado para llegar a ese nodo, se actualiza el costo y se establece el nodo actual como el nodo padre. Si el nodo adyacente no está en la lista "Abierta" ni en la lista "Cerrada", se agrega a la lista "Abierta".

   La función utiliza la función "NodeCompare" para ordenar la lista "Abierta" en cada iteración y la función "recuperaCamino" para recuperar la ruta completa desde el nodo de origen hasta el nodo de destino una vez que se ha encontrado.

   Para llevar a cabo la implementación del algoritmo A*, se utiliza la estructura "AStarNode", que representa un nodo en el mapa. Esta estructura incluye información sobre la posición del nodo en el mapa, así como los valores G, H y F utilizados por el algoritmo A*. También incluye un puntero al nodo padre y una lista de punteros a los nodos adyacentes.

2. Incluya a continuación el código fuente relevante del algoritmo.

```
    void DEF_LIB_EXPORTED calculatePath(AStarNode *originNode, AStarNode *targetNode, int
        cellsWidth, int cellsHeight, float mapWidth, float mapHeight, float **additionalCost,
         std::list<Vector3> &path)
{
    std::list<AStarNode *> Abierta;
    int maxIter = 100;
    originNode->G = 0;
    originNode->H = _sdistance(originNode->position, targetNode->position);
    originNode->F = originNode->G + originNode->H;
    Abierta.push_back(originNode);
    std::list<AStarNode *> Cerrada;
    bool fin = false;
    while (!fin && !Abierta.empty() && maxIter > 0)
```

```
    {
        Abierta.sort(NodeCompare);
        AStarNode *Actual = Abierta.front();
        Abierta.pop_front();
        Cerrada.push_back(Actual);
        if (Actual == targetNode)
        {
            path = recuperaCamino(originNode, targetNode);
            fin = true;
        }
        else
        {
            for (auto it = Actual->adjacents.begin(); it != Actual->adjacents.end(); it++)
            {
                float coste = Actual->G + _sdistance(Actual->position, (*it)->position);
                auto posAbierta = find(Abierta, *it);
                auto posCerrada = find(Cerrada, *it);
                if (coste < (*it)->G)
                {
                    if (posAbierta != Abierta.end())
                    {
                        Abierta.erase(posAbierta);
                    }
                    if (posCerrada != Cerrada.end())
                    {
                        Cerrada.erase(posCerrada);
                    }
                }

                if (posAbierta == Abierta.end() && posCerrada == Cerrada.end())
                {
                    (*it)->G = coste;
                    (*it)->H = _sdistance((*it)->position, targetNode->position);
                    (*it)->F = (*it)->H + (*it)->G;
                    (*it)->parent = Actual;

                    Abierta.push_back(*it);
                }
            }
        }
        --maxIter;
    }
}



bool NodeCompare(AStarNode *A, AStarNode *B) { return A->F < B->F; }
std::list<AStarNode *>::const_iterator find(const std::list<AStarNode *> &Nodes, const
    AStarNode *value)
{
    std::list<AStarNode *>::const_iterator it = Nodes.begin();
    for (it = Nodes.begin(); it != Nodes.end(); it++)
    {
        if (value == *it)
        {
            return it;
        }
    }
    return it;
}
```

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 esta práctica confirmo que he leído la normativa de la asignatura, incluido el punto que respecta al uso de material no original.