Práctica 4. Exploración de grafos

Sergio Rodríguez Vázquez
sergio.rodriguezvazquez@alum.uca.es
Teléfono: +634471641
NIF: 77171745Y

15 de enero de 2021

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

   Implemento el algoritmo A*, para ello uso dos estructuras de tipo vector llamadas opened (vector de nodos abiertos) y closed (vector de nodos ya visitados). Para ello, hago uso de la estructura AStarNode, la cual incluye una lista de nodos adyacentes, el nodo padre del mismo y tres flotantes que son F, G y H, el primero es la suma de los dos siguientes valores, el segundo de ellos es la distancia del origen al nodo y el tercero es la distancia del nodo al destino.

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

```
void DEF_LIB_EXPORTED calculatePath(AStarNode* originNode, AStarNode* targetNode, int cellsWidth, int
{
    AStarNode* actual = originNode;
    actual->G = 0;
    actual->H = _distance(actual->position, targetNode->position);
    actual->F = actual->G + actual->H;
    float d = 0;

    std::vector<AStarNode*> closed, opened;
    opened.push_back(actual);

    std::make_heap(opened.begin(), opened.end());

    bool found = false;

    while(!found && opened.size() > 0)
    {
        actual = opened.front();
        pop_mont(opened);
        closed.push_back(actual);

        if(actual == targetNode)
        {
            found = true;
        }
        else
        {
            for(std::list<AStarNode*>::iterator it = actual->adjacents.begin(); it != actual->adjacent
            {
                std::vector<AStarNode*>::iterator aux;
                aux = std::find(closed.begin(), closed.end(), *it);
```

```
                if(aux == closed.end())
                {
                    aux = std::find(opened.begin(), opened.end(), *it);

                    if(aux == opened.end())
                    {
                        (*it)->parent = actual;
                        (*it)->G = actual->G + _distance(actual->position, (*it)->position);
                        (*it)->H = estimatedDistance((*it), targetNode);
                        (*it)->F = (*it)->G + (*it)->H;
                        push_mont((*it), opened);
                    }
                    else
                    {
                         d = _distance(actual->position, (*it)->position);
                        if((*it)->G > (actual->G + d))
                        {
                            (*it)->parent = actual;
                            (*it)->G = actual->G + d;
                            (*it)->F = (*it)->G + (*it)->H;
                            update(opened);
                        }
                    }

                }
            }
        }

    }
    path = recuperar(originNode, targetNode);
}


float estimatedDistance(AStarNode* a, AStarNode* b)
{
    return abs(a->position.x - b->position.x) + abs(a->position.y - b->position.y);
}


List<Vector3> recuperar (AStarNode *origen, AStarNode *destino)
{
    std::list<Vector3> path;
    AStarNode* aux = destino;
    path.push_back(aux->position);
    while(aux->parent != origen)
    {
        path.push_front(aux -> position);

        aux = aux -> parent;
    }

    return path;
}


bool belongs(std::vector<AStarNode*> &vector, AStarNode* node)
```

```
{
    std::vector<AStarNode*>::iterator aux;

    aux = std::find(vector.begin(), vector.end(), node);

    return (aux != vector.end());
}


bool criterion(AStarNode* A, AStarNode* B)
{
    return A->F < B->F;
}
void push_mont(AStarNode* node, std::vector<AStarNode*> &vector)
{
    vector.push_back(node);
    std::push_heap(vector.begin(), vector.end(), criterion);
}


void pop_mont(std::vector<AStarNode*> &vector)
{
    std::pop_heap(vector.begin(), vector.end(), criterion);
    vector.pop_back();
}


void update(std::vector<AStarNode*> &vector)
{
    std::sort_heap(vector.begin(), vector.end(), criterion);
}
```

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.