Práctica 4. Exploración de grafos

Federico Carrillo Chaves
federico.carrilloch@alum.uca.es
Teléfono: +34615158732
NIF: 32095180Z

5 de diciembre de 2020

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

   El algoritmo A* funciona de la siguiente manera:

   - Como es el primer nodo, lo agregamos a la lista de nodos abiertos y trabajamos con él.
   - Buscamos los hasta 8 posibles nodos adyacentes a ese nodo. Si el nodo no es accesible se ignora.
   - Calculamos la F de los nodos adyacentes si no están en la lista de abiertos tomando como G la distancia euclídea al nodo origen y H la distancia de Manhattan al nodo destino.
   - Si hay un nodo adyacente que está en la lista de abiertos se mira si la nueva G es menor que la que ya tenía y si es así actualiza.
   - Se pasa el nodo actual a la lista de cerrados (también se puede hacer justo después de meterlo en la lista de abiertos).
   - Se saca el nodo con menor F calculada y volvemos a ejecutar todo el proceso hasta que el nodo actual sea el nodo objetivo.
   - Una vez que hemos llegado al nodo objetivo se llama a la función Recupera para ir saltando desde el nodo objetivo hasta su padre, luego hasta el padre de su padre y así sucesivamente hasta llegar al nodo origeen. Con eso iremos sacando cuál es el camino más óptimo para llegar hasta nuestro objetivo porque se ha actualizado en caso de que hayamos encontrado un F menor.

   En este caso, la lista de adyacencia de un nodo viene ya como miembro de la calse AStarNode. Como el nodo con mejor F puede cambiar muchas veces, quizás no es conveniente ordenar un vector cada vez que tengamos un nuevo nodo con menor F sino que podemos ir agregando los nodos a nuestro montículo y tener siempre a mano el nodo con mejor F para procesarlo. El montículo empleado es de la STL.

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

```
    // Algoritmo basado en el pseudocodigo de las transparencias de teoria
void DEF_LIB_EXPORTED
calculatePath(AStarNode *originNode, AStarNode *targetNode, int cellsWidth, int cellsHeight,
    float mapWidth, float mapHeight, float **additionalCost, std::list<Vector3> &path) {
  float cellWidth = mapWidth / float(cellsWidth);
  float cellHeight = mapHeight / float(cellsHeight);
  std::vector<AStarNode *> opened;
  std::vector<AStarNode *> closed;

  AStarNode *cur = originNode;
  opened.push_back(cur);
  std::make_heap(opened.begin(), opened.end());

  bool target = false;
  while (!target && !opened.empty()) {
    cur = opened.front();
    std::pop_heap(opened.begin(), opened.end(), ComparaValor());
    opened.pop_back();
```

```
    closed.push_back(cur);

    if (cur == targetNode) {
      target = true;
      path.push_front(cur->position);
    } else {
      double d;
      for (auto j = cur->adjacents.begin(); j != cur->adjacents.end(); ++j) {
        if (closed.end() == std::find(closed.begin(), closed.end(), (*j))) {
          if (!(opened.end() == std::find(opened.begin(), opened.end(), (*j)))) {
            d = _distance(cur->position, (*j)->position);
              if (cur->G + d < (*j)->G) {
                (*j)->parent = cur;
                (*j)->G = cur->G + float(d);
                (*j)->F = (*j)->G + (*j)->H;
              }
           } else {
            int pos_x = int((*j)->position.x / cellWidth);
            int pos_y = int((*j)->position.y / cellHeight);
            (*j)->parent = cur;
            (*j)->G = cur->G +
            _distance(cur->position, (*j)->position) + additionalCost[pos_x][pos_y];
            (*j)->H = (float) dManhattan((*j)->position, targetNode->position);
            (*j)->F = (*j)->G + (*j)->H;
            opened.push_back(*j);
            std::push_heap(opened.begin(), opened.end(), ComparaValor());
          }
        }
      }
      std::sort_heap(opened.begin(), opened.end(), ComparaValor());
    }
  }
  recupera(originNode, path, cur);
}
```

```
void recupera(const AStarNode *originNode, std::list<Vector3> &path, AStarNode *cur) {
  while (cur->parent != originNode) {
    cur = cur->parent;
    path.push_front(cur->position);
  }
}
```

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.