Práctica 4. Exploración de grafos

Miguel Angel Chaves Perez
miguelangel.chavesperez@alum.uca.es
Teléfono: 675287145
NIF: 49071750p

January 28, 2017

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

   Se implementas dos estructura de la lista List¡AStarNode*¿ para obtener lista de los nodos abiertos y cerrados:

   . La lista cerrado se guardará nodos visitados y la lista abierto los nodos que aún no ha sido visitados . AstarNode* curren para saber que nodo tenemos actualmente . current sera expandido y mirara sus hijos para luego introducirlo en la lista abierta . al expandir current nos encontramos con un hijo que se encuentra en la lista de abiertos, se comprobara si es o no mejor seguir con ese nodo o repercutir hacía el padre . Si es mejor se cambiara el nodo actual current por el padre . Se suma un valor addicional a cada celda que representa un peso mayor o menor a la defensa principal,todo esto con la ayuda de la matriz additionalCost.

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

   language = c++

   bool logico(AStarNode* i, AStarNode* j);

   Vector3 cellCenterToPosition(int i, int j, float cellWidth, float cellHeight) return Vector3((j * cellWidth) + cellWidth * 0.5f, (i * cellHeight) + cellHeight * 0.5f, 0);

   void DEF$_L IB_E XPORTED calculateAdditionalCost(float**additionalCost, intcellsWidth, intcellsHeight, floatmapWi$
   $Object* > obstacles, List < Defense* > defenses)$

float cellWidth = mapWidth / cellsWidth; float cellHeight = mapHeight / cellsHeight; float coste,coorY,coorX;

for(int i = 0; i ¡ cellsWidth; i++) for(int j = 0; j ¡ cellsHeight; j++) coste = 0; coorY = j * cellHeight + cellHeight/2;//Punto medio coorX = i * cellWidth + cellWidth/2;

List¡Defense*¿::iterator def = defenses.begin(); coste = coste + sqrt(pow((*def)-¿position.x-coorX,2) + pow((*def)-¿position.y-coorY,2));

if(sqrt(pow((*def)-¿position.x-coorX,2) + pow((*def)-¿position.y-coorY,2)) ¡ 400) def++; while(def != defenses.end()) coste = coste + sqrt(pow((*def)-¿position.x-coorX,2) + pow((*def)-¿position.y-coorY,2)); def++; additionalCost[i][j] = coste;

void DEF$_L IB_E XPORTED calculatePath(AStarNode*originNode, AStarNode*targetNode, intcellsWidth, intcellsHeight, fl$
$*additionalCost, std :: list < Vector3 > path)$

float cellWidth = mapWidth / cellsWidth; float cellHeight = mapHeight / cellsHeight;

std::vector¡AStarNode*¿ abierto; std::vector¡AStarNode*¿ cerrado;

AStarNode* current = originNode;

current-¿H = $_s distance(current- > position, targetNode- > position); current- > F = current- > G + current- > H; abierto.push_b ack(current);$

bool encontrado = false; std::make$_h eap(abierto.begin(), abierto.end(), logico); intcoorX, coorY; floatdistancia;$

while(encontrado == false abierto.size() ¿ 0) current = abierto.front(); std::pop$_h eap(abierto.begin(), abierto.end(), logico); abier$

if(current == targetNode) encontrado = true;

else for(List¡AStarNode*¿::iterator it=current-¿adjacents.begin(); it != current-¿adjacents.end(); it++) if(cerrado.end()==std::fi
if(abierto.end()==std::find(abierto.begin(),abierto.end(),(*it))) coorX = (*it)-¿position.x/cellWidth; coorY = (*it)-
¿position.y/cellHeight; (*it)-¿parent = current; (*it)-¿G = current-¿G + $_d istance(current- > position, (*it)- >$
$position)+additionalCost[coorX][coorY]; (*it)- > H =_s distance((*it)- > position, targetNode- > position); (*it)- >$
$F = (*it)- > G+(*it)- > H; //Pushmonticuloabierto.push_b ack(*it); std :: make_h eap(abierto.begin(), abierto.end(), logico); el$
current = targetNode; path.push$_f ront(current- > position); while(current- > parent! = originNode)current = current- > p$
bool logico(AStarNode* abierto, AStarNode* cerrado) return (abierto-¿F ¿ cerrado-¿F);

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.