Advanced Computer Architecture

Pathfinding with applications to robotics and videogames

University of Pavia

2020/2021

Lucia Federica Cappucci

Macri Sara

1. **Analysis of the serial algorithm**

The aim of the project is to serialize one of the various algorithms used in pathfinding, where the chosen one is the A\* Search algorithm.

Given a square grid whose valid cells have the same cost equal to 1, the scope of the algorithm is to find the shortest path from the source node to the destination one (verifying that they are valid nodes), avoiding the obstacles present in the grid.

In order to find the shortest path, at each step we pick a node and compute its cost, which is the sum of the two costs “g” and “h”; where “g” is the deterministic cost to move from the source node to a given one, by following a certain path and “h” is the heuristic cost to move from that given node to the destination.

The heuristic cost is an approximation given by a particular heuristic function. Typically the heuristic functions are three: Manhattan Distance, Diagonal Distance and Euclidean Distance.

* Manhattan Distance is obtained as the sum of the absolute values of the difference in the destination’s coordinates x and y (number of the corresponding row and column) and the current cell’s ones.
* Diagonal Distance is the maximum of absolute values of differences in the goal’s coordinates x and y and the current cell’s coordinates.
* Euclidean Distance is the distance between the current cell and the goal cell obtained using the following formula:

Immagine che contiene testo

Descrizione generata automaticamente

Among these three methods, the chosen one is the Euclidean distance since we wanted to be able to move in 8 directions, with Manhattan Distance it’s possible to move in 4 directions only, and between the Diagonal distance and the Euclidean distance, the last one has been deemed more efficient.

This algorithm, in order to work, requires two lists: the open list and the closed list, where the open list consists of the nodes that are been considered to find the optimal path, while in the closed list we place the nodes that don’t need to be taken into consideration since they have been already visited. In the open list we initially put the starting node and compute its neighbours and their “f” cost. Then we put them in the open list, by making sure they are not already in the closed list, and we reorder them in an increasing order of cost and then find the node with lowest “f” cost which is at the top of the list. The node is put off the open list and placed in the closed list. If one of bordering nodes is the destination, the shortest path has been found and the algorithm terminates.

Here is the pseudocode of the algorithm:

1. Initialize the open list

2. Initialize the closed list

3. Put the starting node on the open

list (leave its **f** at zero)

4. while the open list is not empty:

* pick the node with the lowest **f** on

the open list since it is ordered in an increasing order with

respect to f , call it "q"

* pop q off the open list and put in the

closed list

* generate q's 8 neighbors (8 like the number of directions

we can move and set their parents to q

5. for each neighbor:

* compute neighbour.**g** = q.**g** + distance between

neighbour and q

neighbour.**h** = distance from goal to

neighbour(done in this case by using

Euclidean Heuristics)

neighbour.**f** = neighbour.**g** + neighbour.**h**

if it is the destionation, we stop the search

* If it is in open list and the cost of the current node is less than the one in the open list, substitute the node
* If it is in closed list and the cost of the current node is less than the one in the closed list, then substitute the node in the closed, otherwise if it is not in the closed list, we put it in the open list

end (for loop)

end (while loop)

**References**

* https://www.geeksforgeeks.org/a-search-algorithm/