Implementation Dijkstra’s Path finding

In normal computer game a non player character can find a shortest path from initial point to final point by path-finding algorithm. In this report a Dijkstra’s path-finding will be implement and explained.

# Problem

E1

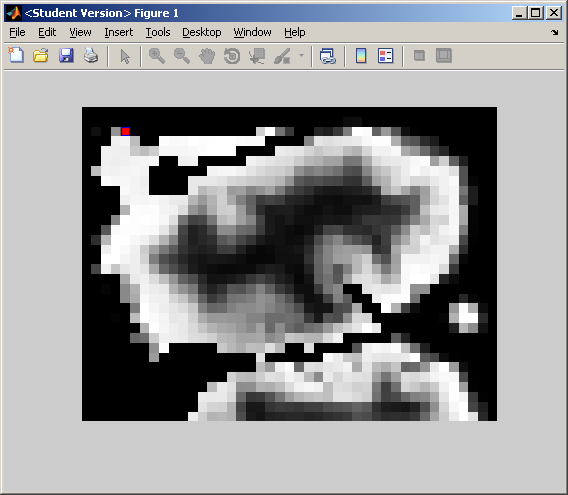
E2

E3

E4

E5

Paths are represented by graph model as below. Each edge can have different distance or mobility to other edge. We have to map each node V to physical location in x-y coordinate and distance can be determined from Euclidian distance. Mobility from node Vi to Vf determine ability to move between the nodes. The distance and mobility between nodes give effective distance as w=d/m, where w is weight or effective distance, d is Euclidian distance and m is mobility is in range from 0 to 1. We can create mobility map from bitmap image as similar as the figure.



# Data structure

A good algorithm is constructed from good data structure. The data structure has single graph g with many nodes and each node has many neighbour edges. So the data structure can implement as a tree.

G

V1

E1

E2

V2

E1

E3

E4

V3

E2

E3

E5

V4

E4

E5

A node Vi is associated to physically space so the node should contain position. In path-finding algorithm we have to generate a queue list Q, which is a list of connected non-visited nodes for considering nodes. I also add two more variable Wmin and Vmin. The Wmin is total shortest distance from final node to considering node and Vmin is nearest node in shortest path connected to considering node.

# Generating a graph

A mobility map is loaded and checked for valid space that a character possibly occupies. For each node the neighbour edges are listed and save in data structure. The weight w is computed and saved in edge. Once a graph is generated we can computed shortest path by Dijksta’s path-finding algorithm.

# Dijksta’s path-finding algorithm

In this implementation I will start searching form final node because when we visited all nodes the Vmin will be computed. Then we can find direction to final node with any initial node by simply reading Vmin at the particular node.

Function Main(Final\_Node)

* Set all node un-visited, Vmin=0 and Wmin=Inf;
* G=Visit(G, Final\_Node)

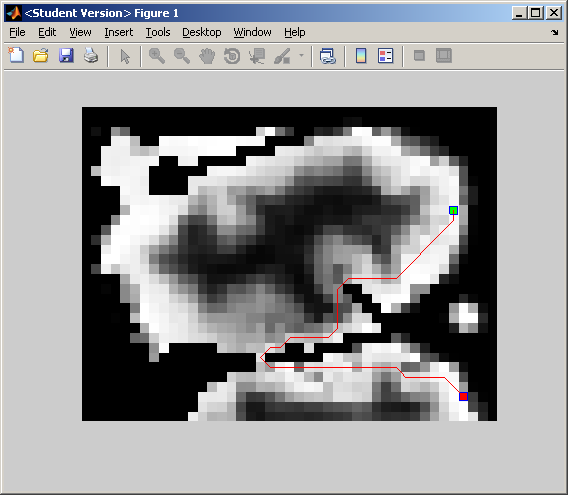
Function G=Visit(G,Q)

* Next\_Q=[];
* For each parent node Vp in Q of the Graph G
  + If node Vp has not visited
    - For each edge E connoting parent node Vp to child node Vc
      * If Vp.Wmin+E.W < Vc.Wmin
        + Vc.Wmin= Vp.Wmin+E.W
        + Vc.Vmin=Vp
      * End
      * Add Vc to Next\_Q
    - End
  + End
* End
* Set Vp as visited
* If Next\_Q is not empty
  + Visit(Next\_Q)
* End

You can find

Example result

The figure below shows a show test path from the red node to the green node. The mobility is show in white the completely black area is invalid space (a character cannot occupy the area).



Wroted by Wasit Limprasert