2012 International Conference on Education Technology and Computer (ICETC2012)
IPCSIT vol.43 (2012) © (2012) IACSIT Press, Singapore

Planar Map Pathfinding Based On The A* Algorithm

Li Mingcui[+]

School of information engineering, East China Jiaotong University, Nanchang 330013,China

Abstract. This article focuses on the pathfinding of game planar map based on A* algorithm. On the basis of the structure of the map and the way of calculating the distance between two points, a rational design of the heuristic function f*(n) is proposed, and the search procedure is discussed in details. This paper also analyses the improvement of search performance by using the balanced binary search tree to describe the open queue, and dynamically modifying the directions of the search directions. The design and implementation of the search method is feasible according to the search results got in actual maps.

Keywords: A* algorithm, pathfinding, heuristic function

1. Introduction

One of the use of the artificial intelligence in the electronic games is automatic pathfinding. There are various ways of pathfinding in the games. For the living beings who always walk around aimlessly in the scenes, the random pathfinding algorithm can be used. That is, choosing a random direction and speed to move, and then another direction and rate is randomly selected. Simple tracking and evasion algorithm can be achieved by adding and subtracting the monster's coordinates according to the target's coordinates, as discussed in [1]. And [2] discussed more complex doDge algorithm. To figure out the path from A point to B point for the game characters in the map rapidly and accurately, one good pathfinding algorithm must be used. A* algorithm can find the shortest and the most direct route, by using heuristic information and designing the path according to the minimum cost, and therefore is qualified for the game path search.

2. A* search algorithm

A* search is an iterative ordered search, the search can take advantage of heuristic information, adjust the search strategy intelligently. The comparison of A* and other algorithms can be find in [3]. Consider the following definitions:

g*(n) to estimate the shortest sequence of moves from the initial state to state n.

h*(n) to estimate the shortest sequence of moves from state n to the target state.

f*(n) to estimate the shortest sequence of moves from the initial state to the target state through state n.

Therefore, f*(n)=g*(n)+h*(n). The most critical part of f*(n) is the calculation of heuristic function h*(n). Because g*(n) can be calculated in the search process through the depth of node n. If h*(n) can not distinguish accurately the states which are valuable to continue the search from those which have no value to continue the search, then the A* search does not perform better than any of the blind search algorithms. If h*(n) can be estimated accurately, then using f*(n) will be able to get a minimal cost solution.

A* search maintains an open queue and a close set. Assuming we search the path from the initial state "start" to the target state "goal", the search steps of A* algorithm are as follows:

(1) Move the initial state start into the open queue.

(2) If the open queue is NULL, then go to step (6).

(3) Remove node n which has the lowest expense from the open queue, and insert n into the close set.

[+] Corresponding author. Tel.: 0791-87046245.
E-mail address: cacaony@gmail.com.

(4) If n is not equal to the goal, then go to step (5). If n is equal to goal, it means the target state has been reached. Return the walk sequence form the initial state to the goal state, end the search.

(5) For each state m adjacent to n, if m has not be detected, then insert m into the open queue; if m has been in the close set, and it's current cost is less than that of the state in the close set corresponding to m, then remove the state corresponding to m from the close set, and insert it into the open queue. Goto step (3).

(6) End the research. The walk sequence from the initial state to the target state goal can not be found.

3. A* search used in planar map

3.1. Calculating the distance of planar grid map

Map used for testing is the traditional grid map, which is generated by rows and columns. In this map, the distance between every two adjacent grid is 1, namely 1 step. From a grid in the map, there are 8 directions to go: upper left, up, top right, left, right, lower left, down, lower right. The coordinates of the grid in upper left corner is (0,0), other grid coordinates is (x, y). Among them, the x is the steps from the left of the map, y is the steps from the top of the map. Movement along the diagonal on the map is allowed. Supposing the cost of moving along straight line and along the diagonal are both D. Then the distance between two points m(x, y1) and n(x2,y2) will be the maximum number of abs(x2-x1) and abs(y2-y1).

3.2. Definition of data structures

Three main data structures are used in the test, which are defined as follows.

3.2.1. Defining an array testmap (width, height)

Define an array testmap (width, height) to determine the status of the points of the map. The value of 0 indicates that the point can pass, 1 indicates that the point is an obstacle, and -1 is representative of the wall.

3.2.2. Defining the node type

We need to find the path sequence, so each node has a point pointing to it's Father node and a point pointing to it's next node. The definition is as follows:

```
Type point_type
   Id As Integer
   X As Integer
   Y As Integer
   Father As Integer
Next As Integer
   Ds As Integer
Dg As Integer
End Type
```

Id is used to identify each state node uniquely. And (x,y) are the coordinates of the nodes. Father represents the Id of the node's parent, next is the Id of the next node. Ds is the distance from the initial node, Dg is the distance to the target node.

3.2.3. Defining the valpoint structure

The valpoint structure is used to determine whether the node has been detected.

```
Type valpoint
   pid As Integer
   tested As Integer
End Type
```

Among them, pid is the node's Id. The variable tested has two values: 0 and 1. 0 indicates the node has not been detected, and 1 indicates the node has been detected.

3.3. Designing g*(n) and h*(n)

For different problems, g*(n) and h*(n) are different. According to the definitions of the map and data structures in the previous section, the design ideas of g*(n) and h (n) are as follows: record the Ds value of each node in the search process, which is the distance from the initial node to node n. The initial node's Ds value is set to 0, other node's Ds value can be caculated according to it's Father's Ds value; And h*(n) can be caculated according to the coordinates of node n and the target node.

The Ds is calculated as follows:

```
start.Ds=0
n.Ds=n.Father.Ds+1
```

Therefore, g*(n) represents the reverse order from node n through it's Father to the start node. Here the key is to maintain the sequence of Father nodes accurately. It can be realized in this way: with n as the center, detect the adjacent nodes of n from eight directions. Suppose one of the adjacent nodes of n is node m. If m has not been detected, then insert m into the open queue. If m has been detected and m.Ds<n.Father.Ds, then modify n's Father node to m.

h*(n) represents the distance from node n to the target node. h*(n) can be figured out by the value of Dg. According to the distance of planar grid map defined in previous section, suppose the coordinates of node n are (xn,yn), the target node goal's coordinates are (xg,yg), then distance form n to goal is the larger of abs(xg-xn) and abs(yg-yn). That is, n.Dg=IIf(abs(xg-xn)>abs(yg-yn), abs(xg-xn), abs(yg-yn)).

3.4. Maintenance of the open queue and the close set

The open queue is used to store the nodes which have been detected, but their adjacency have not. Because the operations of insertion and deletion will happen frequently, the open queue takes a linked list implementation. Arrangement of the nodes in the queue affects the efficiency of the entire search. According to the design of h*(n), nodes in the open queue are sorted by nodes' Dg values in ascending order. When any new node is detected, calculate the Dg value of the new node, insert it into the open queue with the position before the first node whose Dg is greater than the new node's. In the search process, each time take the first node from the open queue to start a new round of detection. The nodes removed from the open queue will be inserted into the close set. So after each round of detection, the node with the shortest distance to the target node is taken from the open queue to start the next round, which helps to improve the search efficiency.

3.5. The Search Process

Assuming the initial node is start(xs,ys), and the destination node is goal(xg,yg), the pseudo-code of the path searching process from start to goal is as follows:

```
Sub pathsearch( )
Initialize the map
Load the data of the  new scene
Initialize testmap(rows, columns)
If the point(x,y) is a barrier then
testmap(x,y)=1
Else
 testmap(x,y)=0
End If
Dim testnode(rows,columns) as valpoint
start.Id=0
start.Father=-1
start.Ds=0
start.Next=-1
start.Dg=IIf(abs(xg-xs)>abs(yg-ys),abs(xg-xs),abs(yg-ys))
open=new priorityqueue
closed=new set
insert(open, start)
Do While open is not empty
        n=minimum(open)
insert(closed, n)
        For i=-1 to 1
           For j=-1 to 1
               get one node  m adjacent  to node n(x,y)
                If testmap(x+i,y+j)=0 then
If testnode(x+i, y+j).tested=0 then
                   m.Ds=n.Ds+1
                    If m.Dg=0 then
 has reached goal , return "Solution"
                          Else
insert(open, m)
End If
                  End If
                   If testnode(x+i, y+j).tested=1 then
                           If m.Ds<n.Father.Ds then n.Father=m
                    End If
End If
        Next j
Next  i
Loop
Return "No Solution"
End Sub
```

Fig.1 shows the auto-routing result from the starting point s(9,28) to the destination point g(24,11) based on A* algorithm. In this map, "s" is the start point, "g" is the destination point, and the trees are obstacles, the smile faces indicate the final route found by the programme.

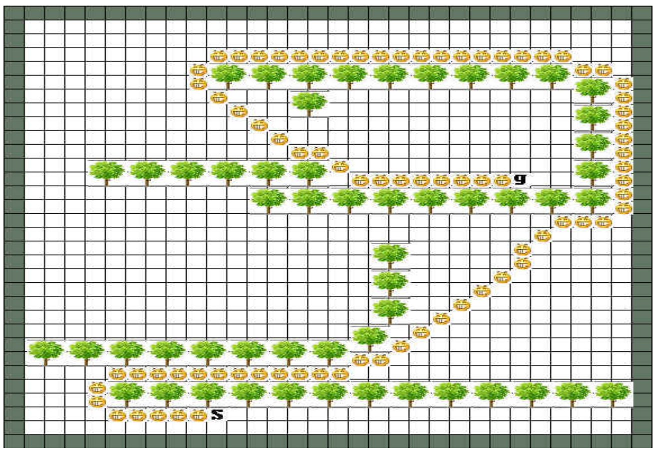

Fig. 1. Auto-routing example of the programme.

4. A* search used in Planar map

4.1. Optimizing the Open queue

Open queue has the following characteristics: need to insert and delete frequently, and each time to insert a new node into the queue is according to the Dg value of the node. To insert a new node m into the right place, first of all we must find the first node t whose Dg value is greater than the new node's, then insert the new node m before node t. If there are a large number of nodes, and each time find the node in the list by sequential search, then the seeking process is very inefficient. In this case, we can use a new data structure to reduce the maintenance of the open queue.

Search tree can be used to hold the open queue. On frequent operations of insertion and deletion, binary search tree's performance is very good. Each node n of binary search tree points to two sub-binary search trees, Tl and Tr, and all elements' key values in Tl are less than or equal to the key value of node n, all elements' key values in Tr are greater than the key value of n. On insertions and deletions, it may occur in many cases some branches of the tree are much longer or shorter than the other branches. The worst case is the degradation of the structure of the binary search tree into a linked list, therefore we need to balance the tree to avoid bias. The implementation of balanced binary search tree is much more complex than the linked list, but can significantly improve search performance (the average performance of sequential search in linked list is O(n), while the average performance of balanced binary search tree is O(log n)).

4.2. Improving the method of detecting the adjacent nodes of node n

After node n has been removed from the open queue, we will detect the adjacent nodes of n by a dynamic order rather than a fixed order (e.g. upper left, left, lower left, just above, just below, top, right, right, lower right). The detection order is determined by the direction vector from node n to the destination node goal. For example, when the destination node goal is just to the right of node n, the adjacent node of n which is just to the right of n will be detected in the first place, and the other adjacent nodes of n will be detected in the next place. In implementations, we divide the goal's place relative to node n into the following categories: angle between the vector form node n to the destination node goal and the positive x-axis is 0°; 9 °; 180°; 270°; node goal is in the first quadrant in the coordinate system with n as the origin; in the second quadrant; in the third quadrant; in the fourth quadrant, which are totally 8 directions to be processed separately. Table 1 lists the search time costed before optimization and after optimization of several paths (due to the difficulty of calculating accurately the time spent by codes in millisecond timescale, we insert the same code to enlarge the time).

Table 1. Comparison of the path detection time

Start node	Destination node	Search time before optimization(ms)	Search time after optimization(ms)
(1, 30)	(12, 31)	48.875	31.25
(2, 17)	(33, 28)	109.375	93.699
(36, 9)	(24, 39)	140.625	93.75
(37, 17)	(1, 22)	187.5	125
(1, 12)	(24, 21)	77.999	62.5
(19, 19)	(14, 9)	32	15.625
(25, 25)	(8, 8)	46.999	31.25
(11, 29)	(26, 24)	47.001	31.25
(27, 8)	(21, 4)	78.901	62.5
(1, 30)	(12, 31)	48.875	31.25

Data in the table indicate the search speed is improved effectively by modifying the search direction dynamically. And by changing the priority of search direction dynamically, the final direction of the path chosen by the programme is much closer to the road taken by users themselves.

4.3. Dealing with special destination nodes

If there is no way to reach the destination node from the start node, A* algorithm will detect all nodes reachable from the start node, and try a variety of paths. In this case, it is bound to spend a lot of time but can not find a way reaching the destination node. An effective way to improve the performance is to set the maximum number of nodes in the search according to the distance from the start node to the destination node (here, distance refers to the distance calculated based on coordinates). If it has reached the maximum number of nodes, but has not found the path to reach the destination node, then stop searching. At this point we can find out the node nearest (whose Dg value is the minimum) to the destination node from the nodes detected. And Set the nearest node as the amendment destination node, from the Father pointer of the node back to the start, return the route with the revised destination node.

Fig.2 shows two different routes form s(27, 8) to g(2, 14) chosen by the programme before optimization and after optimization. And the results indicate that the optimized route is more reasonable.

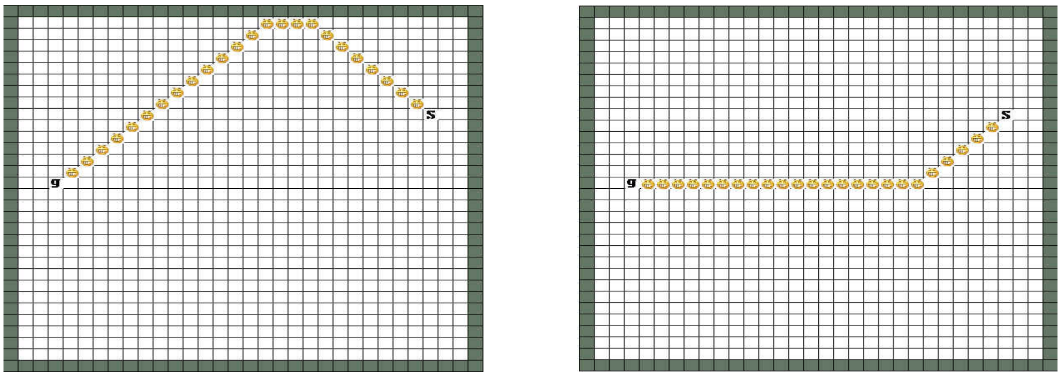

Fig. 2. (a) Path chosen before optimization; (b) Path chosen after optimization

5. Summary

A* algorithm searches for the shortest path using heuristic information, and different problems with different heuristic function f*(n)=h*(n)+g*(n). This article discusses the routing of the planar map with A* algorithm. First, we determine the map structure and the directions of the movement. And then determine the calculation mode of the distance. The heuristic functions g*(n) and h*(n) are designed based on the way of the distance calculated. The distance calculation in this article is based on the characteristics that the distance of one step along the diagonal is equivalent to that along horizontal lines or vertical lines. In different applications, the calculation mode of distance varies according to different functions, such as Manhattan distance, Euclidean distance, etc., which are discussed in [4], [5]. The article also focuses on proposing ways to improve the search performance by using the balanced binary search tree to store the open queue and changing the search directions dynamically. Experimental results show the A* search realized in this paper can find the shortest path from the start node to the destination node with good performance. If the search space is too large, we can cut the search space into several discrete search spaces. With the modified A* search algorithm optimal walking routes can be searched out within the time limit with limitations in space as much as possible.

6. References

[1] Fu chaohui, Ding meng, and Yuxin. Algorithm of Seeking the Path in Game Programming. Journal of Hunan University of Technology, Vol.21, No.4, July 2007:84-87.

[2] Cai fangfang, Yang shiying, Zhang xiaofeng, Liu dongping. Research on Two-tiered A-star Algorithm in Game Path-finding. Microcomputer Applications, Vol. 26, No. 1, 2010:26-28.

[3] George T. Heineman, Gary Pollice, and Stanley Selkow, Algorithms in a Nutshell. Beijing: Tsinghua University Press;2006:147-148.

[4] Cui zhenxing, and Gu zhihua. Shortest Paths Searching in Game Map Based on A* Algorithm. Software Guide, Sep. 2007:145-147.

[5] Gao qingji, Yuyongsheng, Hudandan. Advanced A* Algorithm for Path-Finding and Optimization. Journal of Civil Aviation University of China, Vol.23, No.4, August 2005:42-45.