PATH FINDING SOLUTIONS FOR GRID BASED GRAPH

Dr. R.Anbuselvi M.Sc., M.Phil. Ph.D[1]

[1]Assistant Professor in Computer Science, Bishop Heber College, Trichy-17

satbhc@yahoo.com

ABSTRACT

Any path finding will work as long as there are no obstacles on distractions along the way. A genetic A algorithm has been used for more advanced environments in graph. Implementation of the path finding algorithm for grid based graph with or without obstacles.*

KEYWORDS

Path, Finding, Solution

1. PROBLEM STATEMENT

In this paper, a path finding algorithm has been analyzed to find the shortest path between two point's even obstacles. A Path finding algorithm has been proposed for grid based graph, that finds the least cost path from a given initial node to goal node. The solutions for the path finding algorithm has been analyzed to find the shortest path between two points even with obstacles.

1.1 Proposed Algorithmic Solutions.

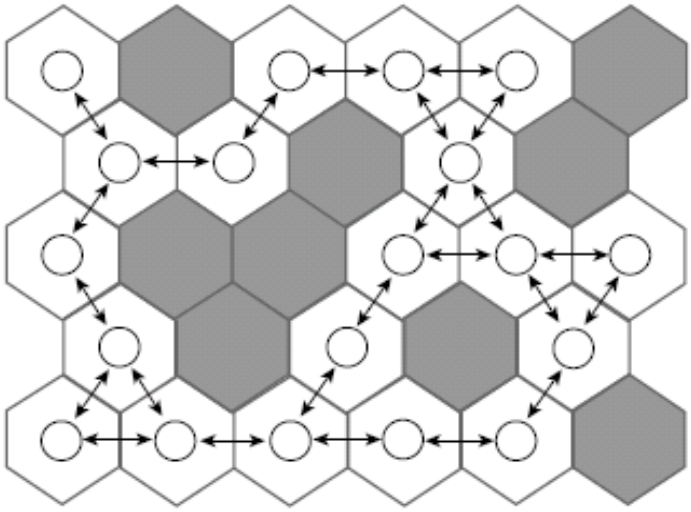
A breadth first search visits nodes in the order of their distance from the start node, where distance is measured as the number of traversed edges. Applying the A* algorithm in a game, it is essential to interpret the game environment as a graph. Using the A* algorithm with a BSP tree the edges between neighbor nodes have a clear path. Path finding algorithmic solution for grid-based graph

2. INTRODUCTION

Path finding is one of the required elements of computer networks. A breadth first search is a path finding algorithm which visits nodes in the order of their distance from the Start node where distance is measured as the number of traversed edges. Applying the A* algorithm in a game, it is essential to interpret the game environment as a graph.

In this work A* algorithm has been referenced which works by visiting vertices in the graph, starting with the objects starting point and created the algorithm for grid based graph. A* Algorithm is focused once again which is the most popular choice for path finding, because it is fairly flexible and can be used in a wide range of contexts.A* is like other graph-searching algorithms in that it can potentially search a huge area of the map as declared in path finding algorithm by Wvan Niftrik (2006). It is like Dijkstra's algorithm in that it can be used to find a shortest path. It is like BFS in that it can use a heuristic to guide itself. In the simple case, it is as fast as BFS

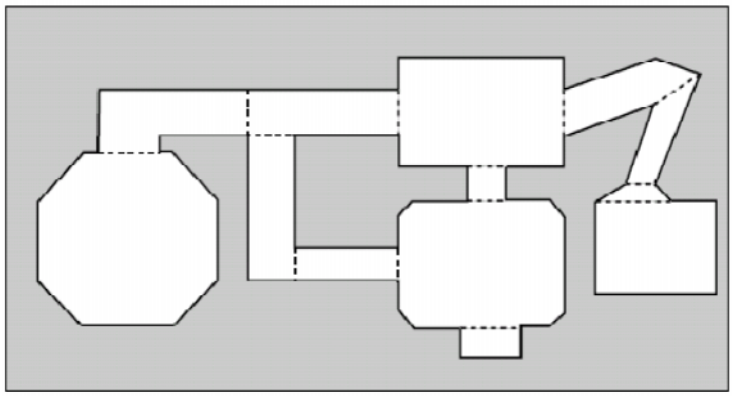
Fig:: 2.1 visiting vertices in a graph. [A tile based Graph]



3. PORTALS

The main advantage is that our rooms in the floor plan are convex.If two entryways are collinear (they exist on the same line in space) and a wall was between them (think doorways in a dormhall) then a bot could collide with the wall between them. This means a bot travelling from one entry way of the room to another will never hit any walls, as long as the entryways are not collinear.[5],[6]. Entryways between convex rooms are often called portals. Portals are generally an invisible polygon (or in a 2D world an invisible line) that separates two convex areas.
In this figure for an example, the dotted lines represent the portals.

Fig:3.1 Portals between convex areas (Dotted lines).



Portals can also be used as a rendering technique with portal rendering, one start by drawing all the polygons in the room that the camera is in.If any portals are in the view, the room on the other side of the portal is drawn, and then so until the view is filled.

If two entryways are collinear (they exist on the same line in space) and a wall was between then, a bot could collide with the wall between them.[7],[8],[9],[10].Using portals as nodes means a graph can be created easily for traveling room to room as in **fig**.
The portals are the nodes in a graph representing the map.If the Tiles were square travelling diagonally would have a greater cost than travelling straight worth, south, east or west. Because the diagonal distance between two squares Tiles is greater. [2],[3],[4].

4. DIJKSTRA'S ALGORITHM

Dijkstra's algorithm has been referenced which works by visiting vertices in the graph starting with the object's starting point and created the algorithm for grid based graph. It works harder but is guaranteed to find a shortest path. It only considers the cost to get to the goal and ignores the cost of the path so far, it keeps going even if the path it is on has become really long.

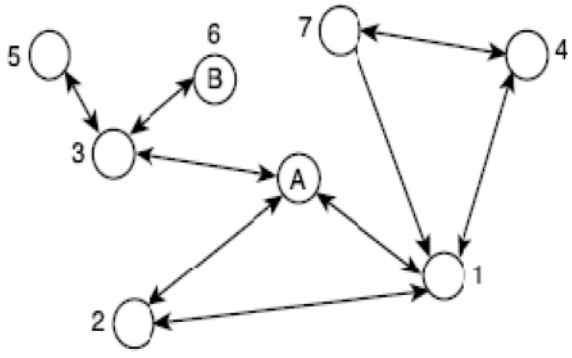
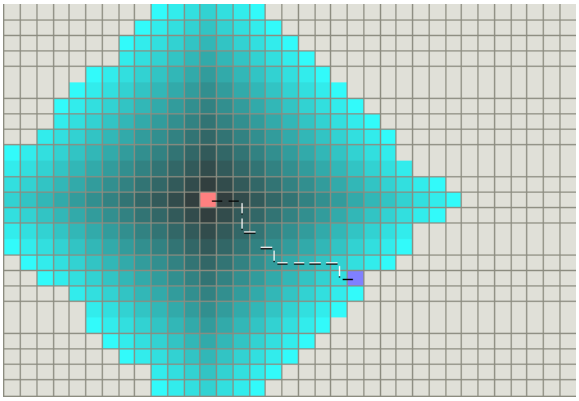


Figure:4.1 Simple Graph

- A graph is given as input.
- If the starting point and the ending point is given the path will be generated.

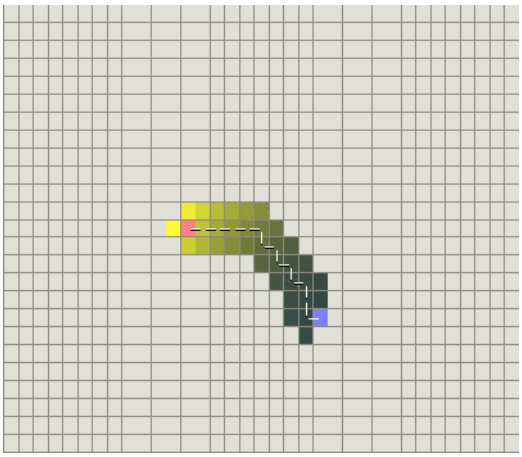
Figure :4.2 Visiting vertices in the graph



Best First Search

If the goal is to the south of the starting position, BFS will tend to focus on paths that lead southwards.

Figure: 4.3 BFS find paths very quickly compared to Dijkstra's Algorithm



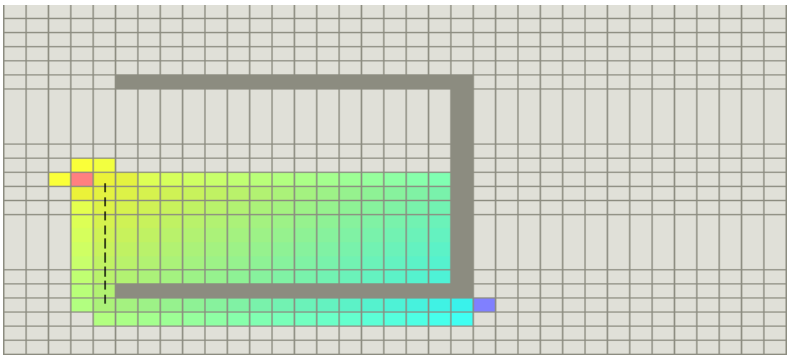
- Best first search is one of the path finding algorithms which is used in this work. The above figure states when the map has no obstacles the shortest path is straight line. The starting point and the end point is choosen so that the path will be generated.

A* Algorithm

- The nodes are numbered in the order of the search. The A* search algorithm is used, which has several similarities to a breadth-first search.
- A* will be implemented with the breadth-first search algorithm.

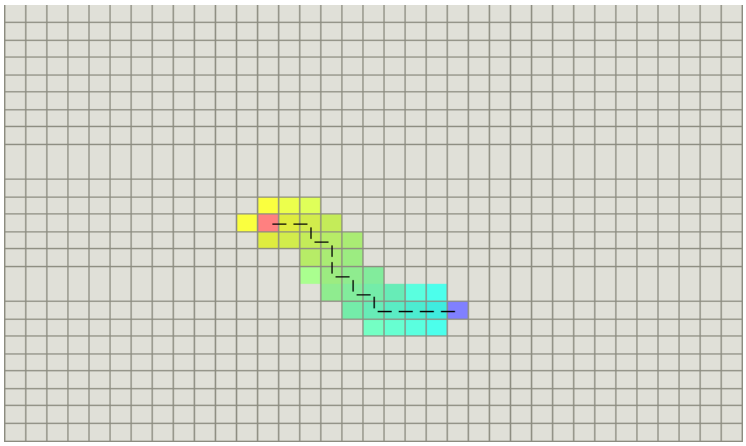
- The neighbors list is simply a list of all the nodes' neighbors. The path Parent node is used for searching only.
- The path from the start node to the goal node as a tree, with each node having only one child. The path Parent node is the node's parent in the path tree.
- When the goal is found, the path can be found by traversing up the path tree from the goal node to the start node as defined by Botea. A,Muller.M(2004) in hierarchical path finding.
- For example, if A has neighbors B and C, and B has neighbor BA, It is not needed to visit A again or it will end up in an infinite loop.

Figure 4.4:With a concave obstacle, A* finds a good path



The proposed finds path with obstacles or without obstacles

Figure 4.5:With a concave obstacle, A* finds a complete path



5. PROPOSED PATH FINDING ALGORITHM FOR GRID BASED GRAPH

- The proposed algorithm finds path with obstacles/ without obstacles.
- If the starting and end points are given the path will be generated.

Breadth-First Search

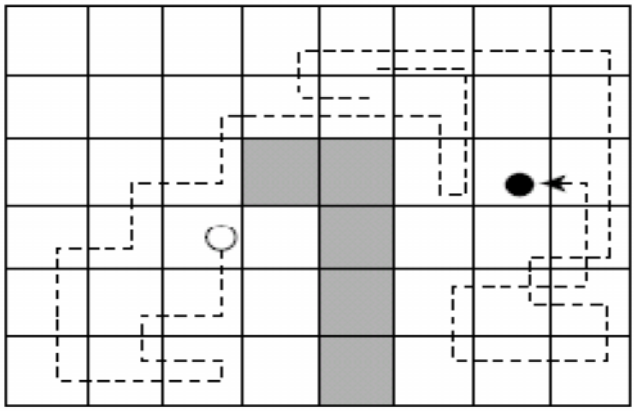
A breadth-first search visits nodes in the order of their distance from the start node, where distance is measured as the number of traversed edges.With a breadth-first search, first all the nodes one edge away from the goal are visited, then those two edges away are visited, and so on until all nodes are visited. Visit the neighbor nodes, then the neighbor's neighbor nodes, and so on until the goal node is found as stated in simple parallel algorithm for the single source shorted path by JLTraff (2000).
An example of a breadth-first search is in Figure in which the nodes are numbered in the order they are visited.

- Applying the A* path finding algorithm for grid based graph.
- To used the A* algorithm in a game, it is essential to interpret the game environment as a graph.
- In this case, the distance between neighbor nodes is the same, but the cost between neighbor nodes does not have to be.
- If the tiles were square traveling diagonally would have a greater cost than traveling straight north, south, east on west because the diagonal distance between two square tiles is greater.
- One benefit of a top-down tile based world is that it is easy to dynamically modify the graph. Using the A* algorithm with a BSP tree
- It is essential to make sure, the edges between neighbor nodes, have a clear path(No obstacles in the way) and that there enough nodes to cover the entire map.

Simulation of path finding algorithm-a bird's eye perspective

- But when obstacles occur between point A and point B, things get a little hairy. For example, if the bot just moves straight toward the player, it will end up just hitting a wall and getting stuck there.

Figure 5.1 Not the best solution:



If the environment is flat with no obstacles, path finding is no problem.
The bot can just move in a straight line to its destination.
But when obstacles occur between point A and point B things get a little hairy.

For ex, if the bot just moves straight toward the player, it will end up just hitting a wall and getting struck there.
The Bot(while circle) just runs straight toward the player(black circle) even if there are obstacles in the way.
Just keeping the right hand on the wall as moving through the maze, eventually the destination will be found.
A graph is similar to a tree, 3D scene management. Indefinite number of children or movement for a single object seems easy.

6. RESULT AND ANALYSIS

The algorithms presented in this work have been experimented with the customized software. The software has been developed in C# as .net Component.

The proposed algorithm is a best first graph search algorithm for grid based graph, that finds the least cost path from a given initial node to one goal node.

$f(x) = g(x) + h(x)$.
$g(x)$ - path-cost function
$h(x)$- admissible heuristic estimate of the distance to the goal.

This algorithm has been generalized into a bidirectional search algorithm. It runs two simultaneous searches.
In the table complexity of A*algorithm O(log f) and proposed algorithm is given as O(f) is the number of elements in the heap.
Path finding algorithm will work with obstacle or without obstacle.
The algorithm is meant for high density network.
The path finding is established with lowest f-score value.
It is important to keep in mind, that one will not merely looking for asymptotic (“Big O”) behavior. In order to look for a low constant lowest f-score value.
It may be noted that an implementation of the A* takes 10000*log(f) seconds while an implementation takes 2*f seconds.
The shift-down function is fast.
In each step if needs only two comparison and one swap.
The index value where is working doubles in each iteration so that atmost leg 2 e step is required.
In order to get the best performance a hybrid data structure is required.

6.1 Comparison of speed of the path finding algorithm for grid based graph.

	A* Algorithm	**Proposed**
Complexity	O (log f) Sec	O(f) Sec
Speed	10,000*log (f) Sec	2*f Sec

Memory	80,000 Kb	512 Kb
Time	2,00,000 Sec	80,000 Sec

The O(f) adjustment on the binary heap is used. Then indexed array could be used to store the location in the heap of each node, this would give up O(log f) for adjustment.
Memory is given in Kilo-Bytes.

7. PATH FINDING AROUND A NEARBY WALL

Path finding around a nearby wall by a swordsman.

Figure: 9.1 Path finding around a nearby wall

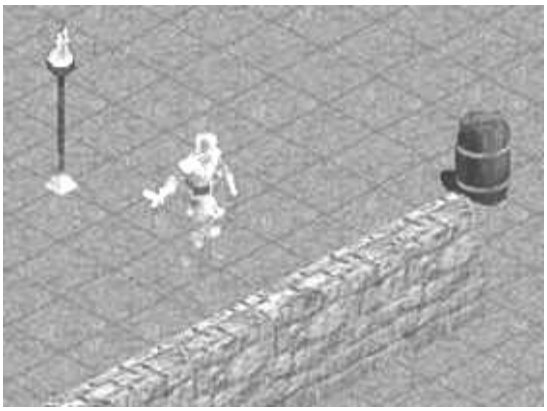
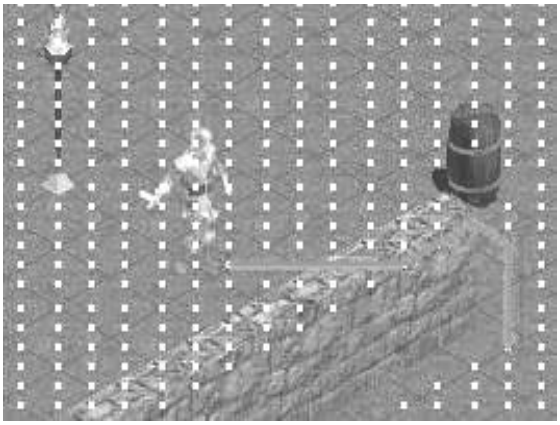


Figure 9.2 high-density node networks



Analysis of fastness of binary heap

In order to add an item to the heap, It must be placed at the very end of the array. It is compared to its parent, which is at location (items in heap)/2, rounding all fractions down.

If the new item's F score is lower, it is needed to s swap these two items. It is essential to compare the new item with its new parent, which is at (current position in heap)/2, rounding fractions down.

If its F value is lower, hence swapped again. This process is repeated until the item is not lower than its parent, or until the item has bubbled all the way to the top, which is in position #1 in the array.

There are currently 7 items in the heap. So the new item would be placed in position no 8.

The new item is underlined.

10 30 20 34 38 30 24 <u>17</u>

Since 17 is lower than 34, it is swapped with 34. The heap will appear like this.

10 30 20 <u>17</u> 38 30 24 34

Then it is compared with its new parent. Now the user in position 4 it is essential to compare it to the item in position on 4/2=2. Since 17 is lower than 30, both are swapped and now the heap 100ks like this

10 <u>17</u> 20 30 38 30 24 34

Next step is to compare it to its new parent. Now the position is in #2. It is compared with the item in position number 2/2=1.Whcih is the top of the heap.

First, the item in slot#1 is removed, now which is empty. Then the last item is taken in the heap and moved to slot #1

The previously last item in the heap is under lined.

<u>34</u> 17 20 30 38 30 24

If not, to swap it with the lower of the two children. So in this case, the two children of the item in slot #1 are in position 1*2=2 and 1*2+1=3.

17 <u>34</u> 20 30 38 30 24

Comparing the item with its new children, which are in position 2*2=4 & 2*2+1=5. It turns out that it is not lower than both of its children. So it is essential to swap it with the lower of the two children (Which is 30 in slot #4)

Now the heap appears like this.

17 30 20 <u>34</u> 38 30 24

Finally the comparison is made with the item with its new children.

Binary heap will speed up our path finding by 2-3 times on average and more on larger maps (A map just 100*100 nodes has 10000 nodes on it)

8. RELATED WORKS

- A solution to solve this problem is pointed out by Rishwal(2010)in the optimal path finding algorithm
- Most of the path finding algorithms as in Craig Reynolds (1999), Dave pottingers (1999), Botea A. Muller M and S. Scheaeffer J. (2004) in the literature are designed with arbitrary graphs which is not realistic.
- Heuristic searches will find any path but will do faster than blind search as stated in path finding algorithm by Marco Pinter(2007).
- Whenever the map is a grid, it is easy to make the function dense described in speed optimization in game programming by Rabin.S (2000.A*).
- Path finding is more general tool that can be used to solve a wider variety of problems as stated in accelerated A* path finding by sisiak(2009).

- A* works at the level of simple passable / un passable grid spaces as defined in Dijkstra path finding algorithm.(2007).

9. CONCLUSION

- Currently A* path bots are god like meaning that they always know where the player is
- In addition it will make the bots strike back in order to fight against robots.
- In this work, some basic path finding routines has been proposed on a tile based map and the “popular right hand on the wall trick”.
- A genetic A* algorithm has been implemented that can be used for more advanced environments in graph. A* search has been merged with the BSP tree and a genetic path is created that follows any type of path. Whether it was created by A* search or not.
- While compared with the A* algorithm, the proposed algorithm have satisfactory performance in terms of complexity ,memory and computer speed and etc..,

References

[1] Botea A.Muller, and Schaeffer, J 2004, near optimal hierarchical path-finding. Journal of Game Development7.28.
[2] Hart P.Nilsson, N.J and Raphael, B.1968. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on System Science and Cybernetics 100.107.
[3] Holte, R.; Perez, M.; Zimmer, R.; and MacDonald, A. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. In proceedings AAA!-96,530.535.
[4] Rabin, S. 2000.A*: Speed optimizations in Game programming Gems.272.287.
[5] Craig Reynolds’s, (1999) “Sterring Behavior for Autonomous Characters”.
[6] Dave Pottinger, “Coordinated Unit Movement:” 22nd Jan 1999. Gamasutra Vol.3: Issue 3
[7] DavePottenger’s “Implementing Coordinated Movement:” 29th Jan 1999. Game Developer PP 48-58. Second in two-part series.
[8] Macro Pinter, “More Realistic Path Finding Article:” 14th March 2001. Gamasutra.
[9] Eric Marchesin, “ A Simple C# Genetic algorithm Article:” 22nd June 2003, .NET 1.0, 4.72
[10] http://www.gamasutra.com
[11] Path finding is more general tool that can be used to solve a wider variety of problems as stated in accelerated A* path finding by sisiak(2009).
[12] A* works at the level of simple passable / un passable grid spaces as defined in Dijkstra path finding algorithm.(2007).

Authors

Dr. R. Anbuselvi M.Sc., M.Phil., Ph.D., has been awarded Ph.D in computer science through Mother Teresa Women's University, Kodaikanal, in April-2012. She is working as Assistant Professor in computer science in Bishop Heber College, Trichy. Her Research interest includes Artificial Intelligence, Neural Network, and Data mining.