International Journal of Engineering & Technology, 7 (2.6) (2018) 54-57

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research Paper

Novel optimization using hierarchical Path finding A* (HPA*) algorithm for strategic gaming setup

[1]Aqsa Zafar, [2]Krishna Kant Agrawal

[1]M.Tech Scholar, Department of Computer Science,
[2]Assistant Professor, Department of ComputerScience,
Amity School of Engineering & Technology,
Amity University, Uttar Pradesh, Lucknow.
**Corresponding author E-mail: aqsazafar81@gmail.com*

Abstract

In game Industry, the most trending research area is shortest path finding. There are many video games are present who are facing the problem of path finding and there is various algorithms are present to solve this problem. In this paper brief introduction is given in the most using algorithm for path finding and A* algorithm has been proved the best algorithm for resolving the problem of shortest path finding in games. It provides the optimal solution for path finding as compare to other search algorithm. At the start of the paper, brief introduction about the path finding is given. Then the reviews of different search algorithm are presented on the basis of path finding. After that information of A* algorithm and optimization techniques are described. In the last, application and examples how the path finding techniques are used in the game is addressed and future work and conclusion are drawn.

Keywords: *Path Finding, Optimal Path, A* algorithm, Shortest Path, A* Optimization, Hierarchical Path finding A* (HPA*)*

1. Introduction

The first important thing come into the mind when we talk about Artificial Intelligence in games is computer controlled players or non-player characters (NPC) [4]. The role of AI in games is to make an intelligent behavior of NPC just like a human behavior which walk through in environment and avoid obstacles on the path. In games there are many situations where NPS has to send from one place to any other destination place and NPC has to avoid obstacles as well as to find the shortest path. So this arise a problem that NPC search the minimum distance node by avoiding obstacles [3]. The method used to solve this issue is known as Path Finding. Path finding usually find the shortest path. The concept of path finding has become very popular as the importance of game industry is gaining more popularity. There are various algorithms are present for path finding some of them are described in this paper, but A* algorithm gives the better solution for path finding in an efficient way [8].

2. Literature survey

2.1. Breadth first search

In BFS Queue is used for storing the record of each node. BFS start with level 0 and it searches level by level until the goal is find. If there exist a solution more than one so BFS choose the shortest solution in which less number of steps are required.

Drawback of Breadth First Search is that it requires a lot of memory because at each level record is saved for expanding the next node [2].

2.2. Depth first search

DFS expands firstly the children nodes in deepest level and finds the solution [2]. If solution is found so it doesn't expand other nodes, it stops searching, means if solution exist it explores one branch only.
Drawback of DFS is that there is no assurance to discover the solution, and there is also no assurance to discover minimal solution, if there exists more than one result in the problem. Another drawback of DFS is that there is a chance that path finding may go under the left most path permanently. The complexity depends on the number of paths exist in the graph.

2.3. Best first search

Best First Search calculate the evaluation function to know the distance of a goal node from the particular node. Best First Search uses heuristic function h(n) to expand the next node, the node who has less h(n) value must be selected as a next node. Formula for finding evaluation function is-

$$f(n) = h(n) \quad (1)$$

2.4. Dijkstra's algorithm

Dijkstra's Algorithm gets started from the beginning node and choose the unvisited minimum distance node, and from that node it again selects its minimum distance neighbor node. This algorithm works only with positive weight nodes.
Dijkstra's Algorithm discover minimum distance path O (E+V Log (V)) if it uses min priority queue with Fibonacci heap. Otherwise if any other implementation of priority queue is taken so it takes E log (E) +V [2].

2.5. A* algorithm

A* algorithm is an, which is used for finding the shortest path between two nodes. A* algorithm always gives the optimal solutions to the problem. This algorithm firstly calculates the cost function of each node and on the basis of cost function value is selects the node for further expansion. Node whose cost function is minimum is selected.
Cost function is calculated as-

$$f(x) = g(x) + h(x) \quad (2)$$

Where,
g (x) = movement cost from the start node to any node x,
h (x) = movement cost from any node x to the goal node.

h (x) is known as heuristic function. Cost function f(x) is the addition of g(x) and h (x). A* Algorithm select the next node whose f(x) value is minimum. A* Algorithm gives the assurance to discover a minimal path from initial node to the end node, if there exist a path. And it gives the optimal solution if h (x) is permissible heuristic.
In A* algorithm there are two list open list and closed list. In the open list nodes which are not yet expanded is put and in the closed list, nodes which are expanded are put. A* algorithm optimal solution because it always chooses minimum cost function node and from that node it againselects minimum cost function node, this process continues until it reaches to the goal node.

3. Difficulties with pathfinding in games

Real-time strategy game, Age of Empires is an excellent game. In this game grids are used to represent the map. A 256×256 grid produces 65,536 feasible position. The motion of military unit is clarified as if going an object by a maze. A* star algorithm is used in Age of Empires. Lot of Age of Empires players are irritated by the abominable path finding. The problem occurs in this game is when a group of objects moved simultaneously at a same time half of them get jammed in the tree, this problem occurs when the number of trees are increased. [5]. Representation of Age of Empires is shown in Fig 2.

Figure 1. A snapshot of Age of Empire [10]

Another real-time strategy game is Civilization V, in which representation of maps is shown by hexagonal tiles as shown in Figure 3. A path finding algorithm is applied here for supervising the group of military going to the wanted place by a collection of "movable" hexagonal tiles. Civilization V also faces the problem of poor path finding. This game was published in November 2010 [1].

Figure 2. A snapshot of Civilization V [11]

In strategy games, thousands of units move at the same time so half of them get stuck because each unit calculate their cost value for moving one node to another node, so when the number of units are more so it takes more time for moving that's why half of them gets jammed in the way. A* algorithm works well in first person shooter games because in FPS few units or player move from one place to another at the same time.

4. A* optimizations

As discussed, A* algorithm gives the optimal solution for path finding in a less time. A* algorithm is proved best algorithm for finding the shortest path and gives the best possible solution. But in some games like real time strategy games and role playing games faces some difficulties as discussed above, so there are various optimizations are present to overcome with these difficulties. These A* optimizations are discussed below-

4.1. Search space

In game, for any AI character, underlying data structure or search space representation is essential for planning the path from start node to the goal node. So for achieving better pathfinding and improving its performance it is very hard to find and implement the suitable data structure for representing the search space. If search space is easy and simple, the working of A* become faster and the work of algorithm which it needs to do gets reduced. Large size search space takes too much time for finding the shortest path. Reducing the size of search space, could cut down half of the time for calculating the cost value and goal node is achieved in short amount of time. Small size search space takes less time for calculating the cost value and it finds the shortest path.
There are various ways for representing the search space by using a rectangular grid in (Fig. 4a), quad tree in (Fig. 4c), convex polygon in (Fig. 4d), points of visibility in (Fig. 4e) and generalized cylinders in (Fig. 4f) [1].

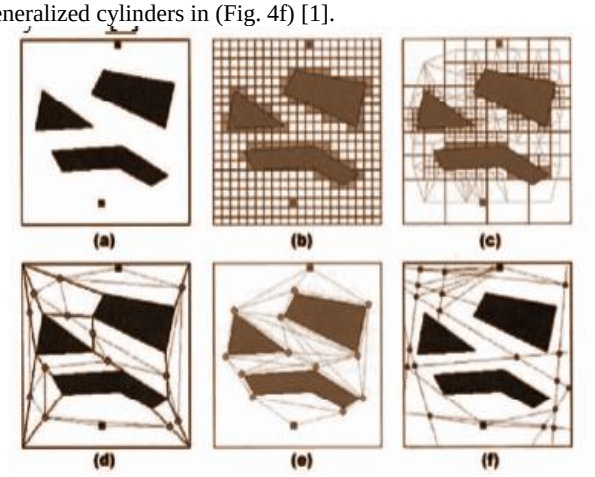

Figure 3. Five ways to represent search space [1]

These following subsections reviews two most popular A* - based algorithm, these algorithms optimize A* algorithm by reducing the search space.

4.1.1. Hierarchical pathfinding A*(HPA*)

In path finding, Hierarchical path finding is very robust technique for improving the speed of path finding. In this technique the game world is broken hierarchically so its reduces the problem complexity. This algorithm is known as ten times quicker than the normal A* algorithm.
Working strategy of Hierarchical path finding is based on divide and conquer strategy [4], like in divide and conquer large problem is divided into smaller subproblems and then these subproblems are solved and after solving, these subproblems are combined. Similar to this strategy Hierarchical path finding breaks the larger map into small sub parts and these sub parts are easier to solve. There are following steps are performed in hierarchical path finding [7].
At the beginning the whole level is converted into grids with each of similar size. Each and every cell is taken as node. Now we examine every cell's environment or background to look if the particular cell can be attached with its nearest next cell.
At the next step the whole large grid is divided into small sub grids. Then make the Abstract Problem Graph and do transition where we declare two nodes connected with each other via an edge [4]. The edge is connection between two cells. These cells may belong to same grid or belong to different grid.
Then add the start location s and the goal location g in the abstract graph. A* search the path from the start location to the destination location.
Thus, This Hierarchical path finding algorithm gives the faster approach for path finding and also reduce the requirement of memory.

4.1.2. Navigation mesh (Nav mesh)

Navigation Mesh is next most powerful technique for path finding. This is basically suitable for 3D world games. Nav mesh consist a group of polygons which shows the "walkable" surface of 3D game world [1]. Nav mesh is easy and the NPC use this for moving and finding the path in game environment. Suppose if a character wants to travel from its location to another location which is situated in another polygon so it

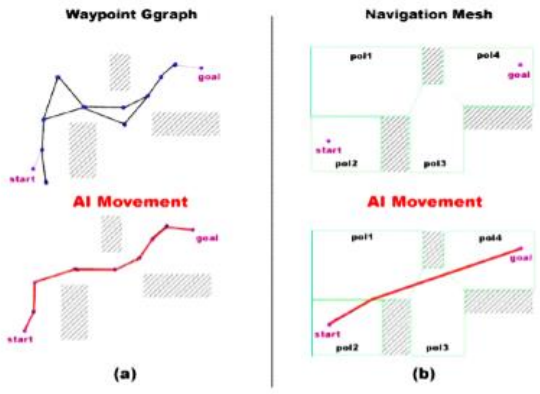

Figure 4. Different representations of waypoint graph and Nav Mesh [1]

firstly, looks up to the next polygon to reach for reaching to its goal location. This process is continued until the character and the destination are in the same polygon. After that character can easily travel to the destination in a single straight line [4]. Compare to the waypoint graph, Nav mesh takes a surety to find the optimal path and explore less data.

4.2. Memory

A* algorithm proved as optimal path finding algorithm but the management of memory requirement is essential because A* takes a lot of memory. In A* algorithm huge amount of memory is required for taking the information of each node by calculating the cost value of each nodes and memory requirement is depends upon the size of the game world. If the environment size of game is larger so it takes too much calculation and also require large memory use. That's why memory management is very important aspect for optimization of A* and in reducing the memory has been a too much work in this area [1].
The very famous method to ignore the memory loss is to pre-allocate a minimum amount of memory [4]. Before starting the A* execution provide some fixed amount of memory for its execution. If all provided memory gets used while execution, so provide a new buffer to continue the search process. The size of this buffer is dynamic to ignore the memory loss. One more thing to notice is that to allocate a minimum size memory and it is totally dependent on the game environment.

4.3. Heuristic function

A* algorithm is too much popular because it uses a heuristic function. In Dijkstra's algorithm there is no heuristic function is used and A* extent the Dijkstra's algorithm by introducing a heuristic function. As in Dijkstra's algorithm it requires a lot of memory usage because it expands all the node for finding the goal node but with the use of heuristic function in A* algorithm it improves the functionality of Dijkstra's algorithm. Heuristic function only examines the most promising node and leaves all unnecessary nodes, so it also reduces the memory usage. A* by using heuristic function improves the computational complexity by expanding only suitable nodes and ignores all other node. Heuristic function works in a manner it calculates the cost value from any particular node to the destination node,

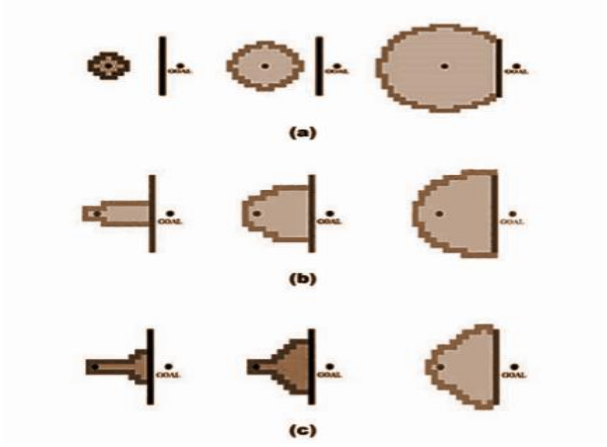

Figure 5. Comparison between different heuristics [4]

if this cost is similar to the actual cost then the node is selected for further expansion and no other node is expanded. This functionality of heuristic function improves the speed of path finding. Thus, a good heuristic function makes the algorithm faster and quicker. By using heuristic, when the true cost is overestimated a bit, the algorithm becomes faster and find the most promising and optimal path [6].
In, Fig. 6 there are three cases are described for heuristic function. In the first case, if the heuristic function is 0, so A* algorithm converted into Dijkstra's algorithm and expands all the possible nodes [4]. In the second case, if heuristic do the accurate cost estimation it selects the node which look a better choice. And in the third case, if heuristic over estimates it chooses the closest possible node to the destination. This leads to improve the speed of the algorithm as well as reduce the memory usage. There is a problem with over estimation of heuristic is how much the cost is overestimated and there is no feasible answer is available yet.

4.4. Data Structure

When the node has been initialized, these nodes have to put somewhere for accessing the information related to each node. So placing these nodes, hash table is a best place for putting these all nodes. It provides a fast access of each node. Hash table helps to trace, which node lies in the open list and which lies in the closed list. Open list is maintained properly by the priority queue. This can be done by binary heap [1]. There is more possibility of research in this area to find out a better data structure for maintaining and fast accessing of open and closed list.

5. Conclusion

In this paper, there are different path finding algorithm are reviewed, working of these algorithm is described and which algorithm is best for path finding and gives optimal solution is also reviewed as A* algorithm. A* algorithm is proved as best algorithm for path finding, it gives optimal path. But this algorithm also requires optimization with respect to memory usage, data structure, search space. These optimizations are also reviewed in this paper. A great amount of work has been done for optimizing the A* algorithm in different perspectives. Thesearesearch space, data structure, memory requirement and improving the heuristic function.

6. Future scope

In the field of path finding, A* algorithm works in a best way and provides an optimal solution but A* algorithm also needs some optimization. A lot of efforts has been done for optimizing the A* algorithm with respect to search space, memory requirement, data structure and heuristic function. Future research is to continue optimizing A* algorithm for the above discussed optimization techniques and to merge these all perspective into one to make A* algorithm more optimal.

Acknowledgement

Authors would like to acknowledge the contribution of Wg. Cd. (Dr.) Anil Kumar, Director, Amity School of Engineering & Technology, Amity University, Lucknow Campus for his continuous guidance and support and for providing a healthy research environment to carry out this research work.
I would also like to thank my course guide and my all faculty members for great guidance and good advices.

References

[1] Xiao Cui &Hao Shi, "A*- based Path finding in Modern Computer Games," IJCSNS, School of Engineering and Science, Victoria University, Melbourne, Australia, International Journal of Computer Science and Network Security, p125, VOL11 No1, Jan 2011.
[2] Amit S. Wale, Taranpreet Singh Saini, Ahmed Mohammad Ali, Vandana S. Jagtap, "Survey Heuristic Search for Shortest Path finding in Games," ISSN-Print 2393-8374, Online-ISSN 2394-0697, VOL2, I 12,2015
[3] Xiang Xu and Kun Zou," An Improved Path finding Algorithm in RTS Games", Communications in Computer and Information Science- Springer 153, pp. 1-7, 2011.
[4] Mehta Parth, Shah Hetasha, ShuklaSoumya, VermaSaurav," A Review on Algorithm for Path finding in Computer Games", available at www.researchgate.net/publication/303369993.
[5] Rubén Tous," Real Time Planning for Path finding in Computer Games".
[6] GeethuElizebeth Mathew," Direction Based Heuristic for Path finding in Video Games", Procedia of Computer Science 47,262-271, 2015.
[7] Guni Sharon," Thesis Summary Optimal Multi-Agent Path finding Algorithm", Published at Twenty Ninth AAAI-Conference on Artificial Intelligence Proceedings.
[8] Rubén Tous," Real Time Planning for Path finding in Computer Games", UniversitatPompeuFabra(UPF), Department de Tecnologia, Spain.
[9] Age of Empire Game by Microsoft www.microsoft.com/games/empires, read on January 10, 2017.
[10] Firaxis Games, "Sid Meier's Civilization V", www.civilization5.com, read on January 10, 2017.
[11] N. H. Barnouti, Al- Dabbagh, S S M &Naser, M A S "Path finding in Strategy Games & Maze Solving Using A* Search Algorithm". @016 Journal of Computer & Communication, 4, 15-25. http://dx.doi.org/10.4336/jcc.2016.411002.