An Improved A* Algorithm Applying to Path Planning of Games

Cheng Zhang*, Lei Ao, Junsheng Yang and Wenchuan Xie
City Institute, Dalian University of Technology, Dalian 116600, China
Email: zhangc@dlut.edu.cn

Abstract. Pathfinding is the core issues in the artificial intelligence field of games, and how to establish an effective method of pathfinding is still focused on. A* algorithm is the most widespread type of game pathfinding at present but the problems still exist such as the curve path and the time-consuming. Based on the analysis and research of traditional A* algorithm, this paper proposes an A* algorithm based on region search. Firstly, the algorithm divides the game map into several connected regions where the role can move freely and without barriers. Then the algorithm uses region search instead of node search. It reduces the search path and the search time. According to the portal between regions, the optimal feasible path is extracted from the region paths. It can effectively solve the zigzag path problem. Experimental simulation results show that the algorithm has shorter search time and can generate smoother path and meet the real-time and real experience requirements of pathfinding in the game.

1. Introduction
Pathfinding in the game is to find an optimal barrier-free path from the starting point to the target point for characters in the game map [1]. A* algorithm is currently the most widely used in the game pathfinding algorithm, which can effectively solve the problems of basic pathfinding, but the A* algorithm also has many limitations, such as: the larger search space, the lower algorithm performance and the higher demand for CPU and memory resources, but not suitable for large map search [2]. In particular the paths created by A* algorithm are often zigzag paths, which are too mechanized, lacking reality [3]. And it is only suitable for static environment and has poor performance in a dynamic environment and so on [4]. An improved idea of A* algorithm is proposed in this paper, and it is more advantageous than the traditional A* algorithm as explained below: According to the characteristics of game map, the map is divided into several regions of the same shape where the roles can move freely and without barriers. According to graph theory techniques, the game map is divided into the weighted graphs regarded region as the node, and the best path is created by A* from the start region to end region. The best path is navigated from the start point to end point according to the algorithm. The algorithm can be much faster and especially suitable for pathfinding in the wide map. It can find the path smoother and can improve the features of autonomy and its playabilities. All are also demonstrated by Simulation results.

2. Environmental Representation in the Game
Grids method is generally used in the environment of games [5]. The space is divided into grids of the same size and grid array is used to represent the environment. Every grid is either in the free space or in the barrier one as shown in figure 1 [6]. Black grids marked “1” are obstacles and white grids marked “0” represent free space in the grid array [7]. The characteristics of grids method will be easy
to implement and have an ability to represent irregular obstacles.

Figure 1. Grids method for space representation.

3. A* Algorithm

3.1. Introduction of A* Algorithm

A* algorithm based on Dijkstra is a heuristic search algorithm and widely used in game maps and the real world to search the shortest path between two points [8]. A* algorithm mainly maintains a heuristic evaluation function as shown in equation (1):

\[ f(n) = g(n) + h(n) \]  

(1)

In the equation, \( f(n) \) represents the estimated distance function from the current node to the target node, \( g(n) \) is the real distance function from the starting node to “n” node and \( h(n) \) is expressed as the estimated distance from “n” node to the target node. In order to find the best path, the value of \( h(n) \) must be less than or equal to the real distance from “n” node to the target node, \( h(n) \) is also called heuristic function that uses the Euclidean distance from the current node to the target node as explained in equation (2):

\[ h(n) = \sqrt{(x_n - x_{goal})^2 + (y_n - y_{goal})^2} \]  

(2)

\( x_n \) is the horizontal coordinate of “n” point and \( y_n \) is the vertical coordinate of “n” point. \( x_{goal} \) is the horizontal coordinate of the target point and \( y_{goal} \) is the vertical coordinate of the target point.

3.2. Process of A* Algorithm

A* algorithm needs to use two tables, Open table and Closed table in the process [9]. And open table stores all nodes that have been generated but not reviewed, and closed table records the nodes that have been visited. The process of A* algorithm is shown below:

(1) Add the starting point to the Open table.

(2) Traverse the Open table, find the node with the minimum “f” value, regard it as the current processing node, and add it to the Closed table.

(3) Judge the eight adjacent grids of the node. If the grid is unreachable or in Closed table, it is ignored; otherwise, do the following:

(a) Add the adjacent grid to the Open table and set parent to that node, and calculate “f”, “g”, and “h” values if the adjacent grid is not in the Open table.

(b) If the adjacent grid is already in the Open table and the new “g” value is less than the old, the parent adjacent to the grid is set to the node and “f” value is recalculated.

(4) Repeat step 2 and 3 until the target point is added to the Open table, indicating the path is found; If the Open table is empty, there is no path.

(5) If the Open table is empty, it indicates no path.

(6) Finally, start to trace back along the parent node of each node from the target point until
returning to the starting point, and the optimal path between the two points is found.

3.3. Problems in A* Algorithm
Firstly, there is a contradiction between search accuracy and search efficiency \([10]\). Map is usually made up of interconnected nodes in A* algorithm. The smaller grid is divided in the map and the more nodes there are. The higher the search accuracy of A* algorithm is, and the better the search effect is. However, the search range is multiplied and the search efficiency is suddenly reduced.

Secondly, there is the problem of zigzag path. The path generated by A* algorithm is composed of series of nodes, but the path often contains some unnecessary nodes, which will produce zigzag path with bad reality \([11]\). There are no obstacles between the starting point and the target point, which can be reached along a straight line, but the path generated by A* is unnecessarily curved as shown in figure 2.

![Figure 2. Path created by A* algorithm.](image)

From the above analysis, we must give consideration to search efficiency and search accuracy in order to produce good practical results, and solve the problem of zigzag path.

4. The Improved A* Algorithm

4.1. Create the Region Map
The map can be divided into several connected blocks where the role can move freely and without barriers by merging adjacent and passable nodes in the map. Each region is polygons of the same shape, in which can be traversed in a straight line without any obstacles. The adjacent parts of the two regions are called portal and all the regions in the map are connected together by them. The map shown in figure 3 is made up of 20*30 squares and the squares correspond to the pathfinding nodes. Region map created by merging nodes is shown in figure 4. Region represents the pathfinding node. As you can see from in figure 3, under the same conditions, the number of nodes in figure 4 is much less than that of in figure 3. Since the search efficiency is inversely proportional to the number of nodes to be searched in A* algorithm, pathfinding for region has shorter search time and higher search efficiency.

4.2. Region Search
The improved A* algorithm takes region as the search node. And the distance between the center point of the adjacent region is the distance between the two regions. The algorithm is shown below:

1. Set the starting point as S and the target point as G, initialize the Open and Closed tables, and obtain “Rs” region where S is located and “Rg” region where G is located respectively.
2. If “Rs” and “Rg” are non-connected regions, the algorithm ends and there is no path between the two regions.
3. If “Rs” is “Rg”, the algorithm ends. The starting point and the ending point are in the same region, and the path is a line segment between the two points.
(4) Add “Rs” and all adjacent regions to the Open table, and set “Rs” as the parent node of its adjacent regions. Remove “Rs” from the Open table and place it into the Closed table.

(5) Take the minimum region “Rn” of estimated value “f” from the Open table. Check all adjacent regions of “Rn”, and operate any adjacent region “Rm” such as the following: If “Rm” is in a Closed table, it is not operated; If “Rm” is not in the Open table, “Rn” is regarded as the parent node of “Rm” and the values of g (Rm), h (Rm) and f (Rm) are calculated, and “Rm” is added to the Open table; If “Rm” is in the Open table and the new value of “g” is smaller than the original value of “g”, then “Rn” is taken as the parent node of “Rm” and the values of g(Rm) and f (Rm) are recalculated.

(6) If “Rn” is the target region “Rg”, the algorithm ends. According to the pointer of the parent node in the region, the optimal path between the two regions is obtained by tracing back from the target region “Rg” to the starting region “Rs”. Otherwise, remove “Rn” from the Open table and place it in the Closed table.

(7) Repeat steps 5 and 6 until “Rn” cannot be found and the algorithm ends.

4.3. Obtain the Path
According to the region path obtained by the above algorithm, the portal between each adjacent region is used to obtain the final feasible path from the starting point to the target point. The specific methods are as follows: Starting from the target region, move back to the starting region along the parent node of each region, record the position of center point of the portal between any two adjacent regions in the path, and take it as the path node of the final feasible path from the starting point to the target point.

5. Simulation Experiment
In order to accurately analyze the search performance of the algorithm, this paper compares it with the traditional A* algorithm. The hardware environment simulated in the experiment: CPU Intel (R) Core (TM) i5-9300H@2.4GHz, with 8GB of memory. Software environment: Operating system for Microsoft Windows 10, simulation software development platform for Visual Studio 2019.

5.1. Search Effect Analysis
Using the above algorithm, the simulation experiment is carried out in the map generated by simulation software. The experimental environment is a 20x30 map, in which the white area is the passable area for the roles and the black area is the impassable area occupied by hills, lakes or other obstacles. The search result of the traditional A* algorithm is shown in figure 5, and the search result of the improved A* algorithm is shown in figure 6. The shaded areas in figure 6 represent the region path from the starting region to the target region. From figure 5, the path planned by the traditional A* algorithm is rather zigzag, too mechanized, and lacks sense of reality, which is caused by its limitation of traditional A* algorithm. From figure 6, the path planned by the improved A* algorithm is relatively smooth and because the algorithm uses region search instead of node search. The path in any single region is always a line segment passing through the intermediate point of the portal and so there is no limitation of traditional A* algorithm. This guarantees effectively search accuracy.
5.2. Search Time Analysis

The search time in figures 5 and 6 are recorded in table 1. In order to reduce the error, average time of pathfinding for 10 times is recorded in figures 5 and 6. From the comparison of time for pathfinding in table 1, under the same software and hardware conditions, the search time of the improved A* algorithm is far less than that of the traditional A* algorithm.

| Map               | Traditional A* algorithm search time | Improved A* algorithm search time |
|-------------------|-------------------------------------|-----------------------------------|
| Figures 5 and 6   | 6.1                                 | 0.74                              |

From the above experimental simulation results, the improved A* algorithm balances the contradiction between search accuracy and search efficiency, ensures the search accuracy and efficiency, and makes the generated path smooth and real. This can enhance effectively the intelligence of game roles.

6. Conclusions

This paper analyzes the problems of A* algorithm applying on pathfinding in game map, proposes the improved A* algorithm. The algorithm divides the map into several passable regions of the same shape, uses region search instead of node search, balances the contradiction between search accuracy and search efficiency and enhances search efficiency and solves the problem of not enough smooth path. Experimental simulation results show that the algorithm has shorter search time and can generate smoother path and meet the real-time and real experience requirements of pathfinding in the game.

References

[1] Zhang C, Zhang K, Zhang Y Q and He D 2014 Applied Mechanics and Materials 556 3420.
[2] Zhao W, Xie X F and Sun Y L 2011 Computer Applications and Software 28 159.
[3] Zhu Y, Yang S Y and Ma Y 2011 Microcomputer Information 5 200.
[4] Cheng G, Fu S F and Zhou L H 2007 Science Technology and Engineering 15 3731.
[5] Qiu L 2011 Journal of Shaanxi University of Science & Technology 6 89.
[6] Thi T M, Cosmin C, Duc T T and Robin D K 2017 Applied Soft Computing 26 74.
[7] Wang C and Zhu W D 2018 Computer Systems & Applications 1 189.
[8] Zhao Q and Zhao A Q 2013 Journal of Electronics & Information Technology 4 952.
[9] Liu H H and Guo J J 2013 Journal of Civil Aviation University of China 6 61.
[10] Wei W M, Lu Z J and Ye Y T 2018 Journal of Shanghai University of Electric Power 2 180.
[11] Zhu Y H and Yu Y Q 2018 Computer Technology and Development 4 55.