An Improved Pathfinding Algorithm in RTS Games

Xiang Xu and Kun Zou

Department of Computer Engineering
University of Electronic Science and Technology of China Zhongshan Institute
Zhongshan, P.R. China
xushawn@sina.com

Abstract. Pathfinding is core component in many games especially RTS(real-time strategy) games, the paper proposed an improvement algorithm to solve the pathfinding problem between multi-entrances and multi-exports buildings. Firstly, the paper studied the A* algorithm, aiming at the buildings with multiple entry and exit characteristic in games, improved the algorithm and make it can find a shortest path by only one traverse between multiple start nodes and multiple stop nodes, avoided frequent call for the algorithm. Secondly the algorithm had given the solution to the actual terrain cost problem, and calculated actual path cost according to the different terrain influence factor. Finally, introduced collision detection to the dynamic path change state space, solved effectively the influence of dynamic path changes.

Keywords: pathfinding; RTS game; A* algorithm; terrain cost; collision detection.

1 Introduction

Traditionally the algorithm using in pathfinding is A* algorithm[1][2], used in performing fast search for the optimal path connecting two points on the map of a game. A* is more suited to an environment where there are multiple routes around the environment. In other words, it is a very time consuming and more complexity method in game's implementation. In order to better use A* algorithm in games, still need to solve a series of problem, such as data storage structure optimization and search strategy improvement, heuristic function selection, collision detection, obstruction avoiding collision and so on. The paper firstly studied A* algorithm in theory, and put forward some actual problems when using A* algorithm in RTS games, finally a set improvement strategy with application value is given.

2 Background

A* is a graph search algorithm that finds the least-cost path from a given start node to one goal node (out of one or more possible goals). It uses a distance-plus-cost heuristic function (usually denoted f(n)) to determine the order in which the search visits nodes in the tree. The distance-plus-cost heuristic is a sum of two functions: the path-cost

G. Shen and X. Huang (Eds.): CSIE 2011, Part II, CCIS 153, pp. 1–7, 2011.

function (usually denoted g(n), which may or may not be a heuristic) and an admissible heuristic estimate of the distance to the goal. The path-cost function g(n) is the actual cost from start node to current node, and h(n) is the estimate cost from current node to the goal. Because g(n) is known, it can be calculated by reverse tracking from current node to start node in accordance with a pointer to its parent, then accumulate all costs in the path. So, heuristic function f(n)'s heuristic information relies mainly on h(n). According to a certain known conditions of state space, heuristic function will select one node with minimum cost to search, again from this node continue to search, until reach the goal or failure, but not expanded nodes need not search [3].

The quality of A* algorithm depends on the quality of the heuristic estimate h(n). If h(n) is very close to the true cost of the remaining path, its efficiency will be high; on the other hand, if it is too low, its efficiency gets very bad. In fact, breadth-first search is an A* search, with h(n) being trivially zero for all nodes -- this certainly underestimates the remaining path cost, and while it will find the optimum path, it will do so slowly. Here adopt the Manhattan heuristic function, namely obtain the minus of abscissa from current node to the goal, and also the minus of ordinate from current node to the goal, again both absolute value adding together. The Manhattan heuristic function is shown below:

$$h(n)= (abs(dest.x - current.x)+abs(dest.y - current.y)) \quad (1)$$

In order to enhance the efficiency of the algorithm, may preprocess those unreachable nodes in the game map before starting search. And may adopt the binary heaps structure for the Open list.

There are situations where A* may not perform very well, for a variety of reasons. The more or less real-time requirements of games, plus the limitations of the available memory and processor time in some of them, may make it hard even for A* to work well. A large map may require thousands of entries in the Open and Closed list, and there may not be room enough for that. Even if there is enough memory for them, the algorithm used for manipulating them may be inefficient. On the other hand, in many RTS games, the start location and the goal does not have to be a single location but can consist of multiple locations. The estimate for a node would then be the minimum of the estimate for all possible nodes. Next, because A* algorithm is according to the grid map, although the path is the shortest, but often zigzag and not realistic, this is caused by the fact that the standard A* algorithm searches the eight neighbour nodes surrounding current node, and then proceeds to next node. It is fine in primitive games where units simply hop from node to node, but is unacceptable for the smooth movement required in most RTS games today.

3 Pathfinding between "Container Game Object"

3.1 Problem Statement

In many RTS games, some game objects (e.g. barracks), often stationed a number of units. By Specifying a destination, these units should go out from the game object and move to the destination. Usually called this kind of game object as a "container game object"[4]. In actual games, "container game object" usually demonstrated as buildings

or all kinds of transporters, this kind of building usually has multiple entry or multiple exit. The game pathfinding problem will become searching a shortest path between multiple entry nodes and multiple exit nodes. The similar phenomenon might be found in other games. For example: a player need to dispatch some units to move to a target area, possibly has many contiguous barracks can be selected, how to choose one barracks which is recent to the target area, and guarantee units may fastest arrive, becoming game's inevitable problems.

Regarding multiple start nodes of situation, pathfinding's main work is to select the optimal node from all exit nodes surrounding the game object, and let these units start leaving from this node. The simplest and most rapid method is predefined a node order, then choose a walkable node, but this method does not consider position of the destination, before leaving, those units will usually forced to walk round and round in the game object. To avoid this problem, should choose the optimal exit node, and ensure the chosen exit node to the destination has a minimum path cost, meanwhile this path has already considered obstacles distributed situation between the start node and the destination. One effective method is calculating each path cost from each exit node to the destination, and chooses the lowest cost path. Such need call A* algorithm many times, and carries on the comparison many times, performance overhead is too high.

The same problem also appears in multiple entry nodes. In game development, the supposition dispatches some units to garrison a game object (e.g. barracks), namely, gives a combat task is to choose an optimal entry from all entry nodes surrounding the game object, let the units enter. If only simple choice an entry node has minimum straight distance, will face not considered the obstacles. General solution is to calculate each path cost between each entry node and start node, need continuously call A* algorithm, then found an optimal entry node through the comparison.

Above problems all boil down to multiple start nodes and multiple stop nodes pathfinding problem, among which, the first game object provides multiple start nodes in the exit location, and the second game object provides multiple stop nodes in the entry location.

3.2 Algorithm Improvement

Through examining the A* algorithm, can find that algorithm most of time is carrying on node traversal. Regarding need to call A* algorithm with sole start node and sole stop node many times, the traversal number of times will increase greatly, even many nodes are repeated traverse. Through revising standard A* algorithm, let it accepts multiple start nodes as function parameter, and before start searching, set these start nodes' G value(path cost value) equals 0, and H value(estimate cost value) calculated by the Manhattan heuristic function. After calculated each start node's F value, put them all into the Open list, and guarantee sorting. The rest of the treatment process is the same with the sole start node process. Because the Open list provides all start nodes sorting, it can help us choose an optimal start node, and treat it as a part of the final path. The same holds for multiple stop nodes, considering the goal node's main purpose is as traverse stop point, can use multiple stop nodes to replace one goal node. These stop nodes set on the way to the goal. In pathfinding process, once discovered a stop node, can think path has been found. The improved algorithm flow chart shown below:

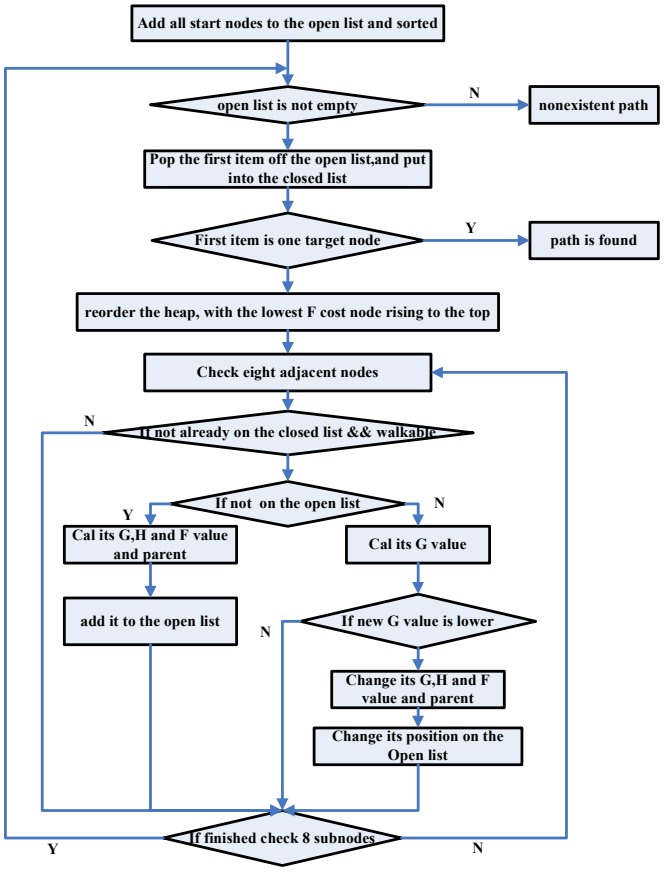

Fig. 1. Improved A* algorithm flow chart. Support multiple start nodes as function parameter, and once searched a stop node, the algorithm will think path has been found.

In Figure 2, the barracks is seen as a game object, it peripheral has 8 exit nodes. There have some obstacles in the map. The optimal exit node can guarantee a minimum cost path to the goal. The barracks' right bottom corner has an exit node, it has the shortest straight distance to the goal, but it is not an optimal exit node.

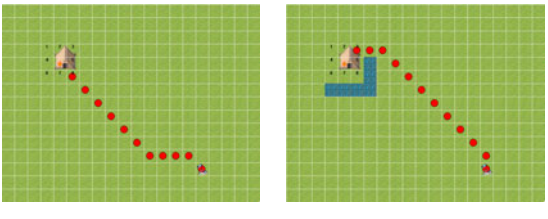

Fig. 2. Left Graph does not have any obstacles, the barracks' right bottom exit node is the optimal node; Right Graph has some obstacles nearby the barracks, caused the optimal node to turn the top right corner exit node

Figure 3 is given more than one entry node, to find the optimal entry node. The barracks is surrounded by 8 entry nodes. The optimal entry node can guarantee a minimum cost path from the start node. The barracks' left top entry node has the shortest straight distance from the start node, but it is not an optimal entry node.

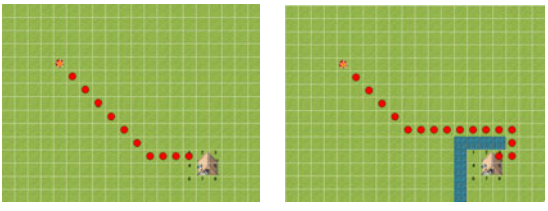

Fig. 3. Left Graph does not have any obstacles, the barracks' left top entry node is the optimal node; Right Graph has some obstacles nearby the barracks, caused the optimal node to turn the top right corner entry node

Figure 4 shows an example of two buildings -- barracks A and barracks B. Barracks A provides more than one exit node, barracks B provides more than one entry node. Assuming that the goal location is on the barracks' left bottom corner (marks with the flag image), this position is away from each entry node the distance is the same.

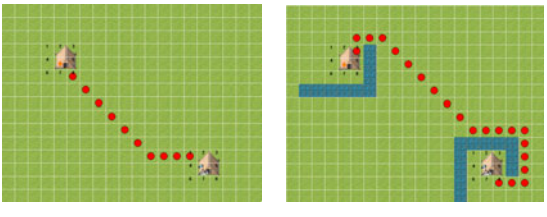

Fig. 4. Left Graph does not have any obstacles; the barracks A's right bottom exit node is the optimal node, and the barracks B's left top entry node is the optimal node. Right Graph has some obstacles nearby two barracks, caused the optimal node changed.

4 Introduction Terrain Influence Factor

The cost of going from one position to another can represent many things: the simple distance between the positions; the cost in time or movement points or fuel between them; penalties for traveling through undesirable places (such as points within range of enemy artillery); bonuses for traveling through desirable places (such as exploring new terrain or imposing control over uncontrolled locations); and aesthetic considerations-for example, if diagonal moves are just as cheap as orthogonal moves, you may still want to make them cost more, so that the routes chosen look more direct and natural.

Due to the path-cost function g(n) has taken terrain cost into account, just before assumes all terrain costs are same, so set each node cost is equal. But in the actual game, when calculates g(n), may for the different terrain node designation different cost, thus realizes the lowest path cost, rather than the shortest path.

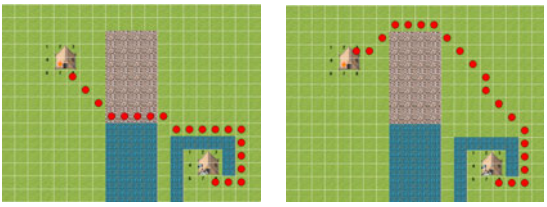

Fig. 5. Left Graph has not considered the terrain cost. Right Graph has considered the terrain cost, and set the lawn terrain influence factor equals 1, the mountain road terrain influence factor equals 2.

After introduced the terrain influence factor, searched path can judge different types of terrain cost, and choose a lowest cost, rather than the shortest path. Of course, the actual game will have more complex terrain, but can still by setting different terrain cost to realize. This method will make the game role walk the path of more and more intelligent and realistic.

5 Adapt to Dynamic Path Change State Space

In games, often encounter such a situation that may originally through the path now became cannot pass. For instance a node that has a bridge built on it suddenly blew up, either has built up a house suddenly, either other unit happen to moved this position [5]. This will block the path of moving, and now need pathfinding algorithm adapt to the dynamic changes of the state space.

In order to recognize this state changes in moving process, need to introduce collision detection subsystem. Because in the game, every character or obstacles is an independent entity, if can pass through mutually, will give the human one kind of false feeling, reduced game's authenticity and reality. A* algorithm in decision making realize character movement before must first get collision detection subsystem decision-making. Once discovered the next node become a blocked node, needed to re-run pathfinding. Usually can search a new path from blocking preceding node to the goal (the following chart shows), but this approach sometimes with a frequent search algorithm calls, search efficiency is quite low. Can only search a new path from the last walkable node before breaking to the first walkable node after breaking, then links this new path to the original path, replaces the breaking part. This can improve efficiency,

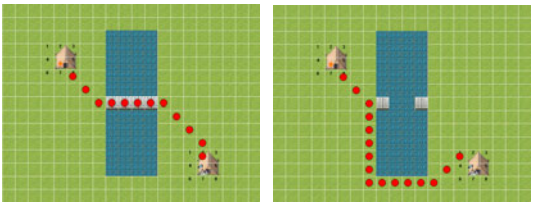

Fig. 6. Left Graph shows the path before the bridge blows up. Right Graph shows found path after the bridge blows up.

because many has obtained the paths can reuse, and search two not far away nodes is very fast, expand node less often. Also can be in certain steps before check the node status, not only check the next node, it increases the authenticity, but also will increase time consumption [6].

6 Conclusion

The paper studied the application of A* pathfinding algorithm in RTS games. According to the problem of pathfinding between "container game object", the algorithm is improved and optimized. Unified multiple start nodes together into the Open list for pathfinding, effectively avoid the frequent algorithm called. Meanwhile, in order to truly reflect different types of terrain influence, introduced terrain influence factor into the algorithm, and g(n) used for the calculation of different terrain cost, enables the path can find the good recognition terrain. After a series of improvement, generated by the search path can be reflected in the game actual path effect, and embodies the certain intelligence and humanization. Considering the RTS games of moving Units usually not one, but has a group, the next step will do further research on "Coordinated Unit Movement". Hope that the future games have more intelligent, more humanized characters, and hope there will be more better algorithm to solve problems in game pathfinding.

References

1. Dechter, R., Pearl, J.: Generalized best-first search strategies and the optimality of A*. Journal of the ACM 32, 505–536 (1985)
2. Bourg, D.M., Seemann, G.: AI for Game Developers, 1st edn., pp. 51–75. O'Reilly Media, Sebastopol (2004)
3. Khantanapoka, K., Chinnasarn, K.: Pathfinding of 2D & 3D Game Real-Time Strategy with Depth Direction A*Algorithm for Multi-Layer. In: 2009 Eighth International Symposium on Natural Language Processing (SNLP 2009), Bangkok, Thailand, pp. 185–186 (2009)
4. Higgins, D.: Generic Pathfinding. AI Game Programming Wisdom (2002)
5. Roth, U., Walker, M., Hilmann, A., et al.: Dynamic path planning with spiking neural networks, pp. 1355–1363 (1997)
6. Stentz, A.: Optimal and efficient path planning for partially-known environments. In: Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, pp. 3310–3317 (1994)