Research and Application of Path-finding Algorithm Based on Unity 3D

Zhang He
School of Computer Science
Communication University of China
Beijing, China
13207144228@163.com

Minyong Shi, Chunfang Li
School of Computer Science
Communication University of China
Beijing, China

***Abstract*—Unity 3D is a cross-platform 3D game engine. That is an important problem of seeking the road in the process of game production. The main solution is path-finding algorithm in which the player finds a path to the target in game scene. In this paper, by general path-finding game scene as background, we focus on the A* algorithm and "Navmesh Grid" path-finding in Unity 3D, and put forward the improvement and application of algorithm in the special game scene.**

Keywords—Unity 3D; A* algorithm; "Navmesh Grid" path-finding

I. INTRODUCTION

Unity 3D is a comprehensive game development tool which is developed by Unity Technologies. Most games in the process of production will encounter the problem of seeking the road. How to try to find a path that from the starting point to the target point in game map? More complex games also needs to consider the file structure of the game map and the passable of target location etc. In this paper, we focus on the two common path-finding algorithm in Unity 3D, and put forward the improvement and application of algorithm in the special game scene.

II. A* ALGORITHM

A* algorithm is the most widely used in the game map, which uses the heuristic function to estimate the distance of any point to the target point, so as to reduce the search space and improve the search efficiency [1].

A. A Algorithm Theory*

A* algorithm is a heuristic search algorithm based on "Dijkstra" algorithm. Heuristic search is searching to evaluate each extension node in the state space, and select the best node's location, and then search from the start node until it finds the target node. In heuristic search, the location of the evaluation is very important, and the use of different evaluation may have different effects. The valuation function represents the estimated cost of moving from the current node to the target node, which is heuristic. In the path-finding and maze problem, we usually use the Manhattan estimation function to estimate the cost. The valuation function of the current node n is expressed as:

$$F(n)=H(n)+G(n) \qquad (1)$$

Formula (1), F (n) is an evaluated function, H (n) is the heuristic value of the shortest path of any node n to the target point, G (n) is the shortest path of the start point to any node n . It can be proved that if the evaluated function satisfies the compatibility condition that the evaluated function H (n) is less than the actual cost of the node n to the target node, and the original problem exists the optimal solution, then the A* algorithm can find the optimal path [2]. In the game, the A* algorithm is used to search in the map and create the corresponding node by through the game object's location to save the location information of moving objects. Before using the A* algorithm, the map should be divided to each piece as a node. Quadrilateral division is the simplest form because some terrains on the scene may be divided into a convex polygon map, four fork tree, and other shapes. As shown in figure 1:

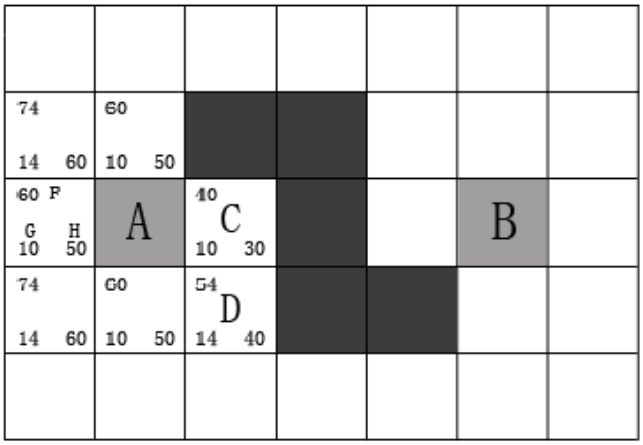

Fig. 1. Example of A* search

We need to find a shortest path from B to A and the black part is the obstacle in the middle of part. We use 10 and 14 for simplicity's sake. The distance between the diagonal angles is 14 and the distance between the right angles is 10. F is printed in the top left, G is printed in the bottom left, and H is printed in the bottom right. In the search for the first time to add A of the adjacent squares to the open list, find out the adjacent

978-1-5090-0806-3/16/$31.00 copyright 2016 IEEE
ICIS 2016, June 26-29, 2016, Okayama, Japan

squares C of the smallest F value as the next square. Take it out from the open list added to the closed list, traversing the adjacent squares, and ignoring the square obstruction. Because there are two same squares of the minimum F value near the C, freely choose one squares records for D, and G value is increased by 10 from C to D, G value only need 14 from A to D. It's more direct to get that square from the starting square by moving one square diagonally to get there, rather than moving horizontally one square, and then vertically one square and the square D of the parent node is updated to A. Repeat this process until the target node is added to the open list. The results are shown in figure 2:

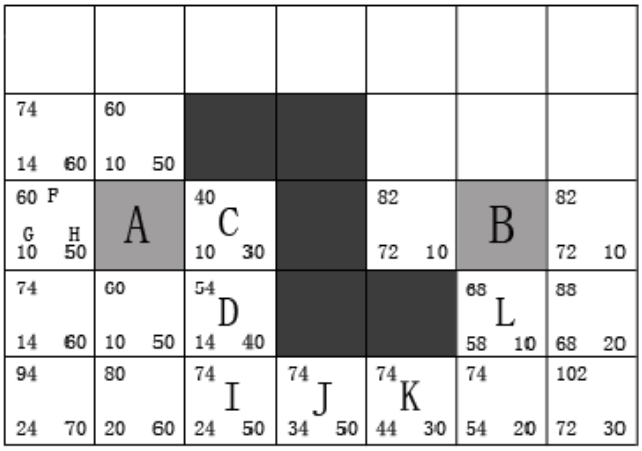

Fig. 2. A * search results

After searching, finding the shortest path turned by the parent node, as shown in figure 2, the target node B's parent is L, so you can get a path from A to B (A, D, I, J, K, L, B,).

B. Implementation steps

After the scene was built in Unity 3D, create a new class "Node" for information storage node, including the radius of the grid, world coordinates, parent node, and create a constructor for transfer value. Select the desired walking ground, and determine the number and size of the grid according to the size of the ground and the radius of the Node, Marking it where the barrier mesh layer as "Unwalkable". Determine the grid of the player by the world coordinates of the starting point of the player and declaring an open set and a closed set, and add the player's start point grid to the beginning of the collection. If the open list contains some data, circling the following steps:

1) Set the current node to an adjacent node of the minimum F value by traversing the open list.

2) Remove the current node from the open set and add it to the closed set.

3) If the current node is the target node, end query, received by the parent node back to find a shortest path.

4) Traverse each adjacent node of the current node:

a) If the adjacent nodes cannot be accessed or the adjacent nodes are closed in the collection, skip the adjacent node.

b) Calculate the G value of the current node and the distance between the current node and the adjacent node, and then gain the new path distance.

c) If the distance between the new path and the adjacent node is shorter, or the open set does not include the adjacent nodes:

- Reset the F value.
- Set the parent node to the current node.
- Add the adjacent node to the open set.

Drawing the created map form in the "OnDrawGizmos" function and making a statement for target node in path-finding script. The shortest path can be observed by moving dynamic the position of player.

C. Algorithm Improvement and Application

A* algorithm has the shortest time in theory, but also has its disadvantages, and its spatial growth is exponential, so that further optimization is needed. It can make the table data increase gradually while finding the minimum value in all nodes by searching constantly for collection, resulting in the search process more and more slowly. For the data reading problem, some scholars put forward the A* algorithm based on the restricted area to reduce the loading of data [3-5]. Because A * is a detrimental algorithm [3], the search may result in less than the shortest path. A simple approach is to sort the data table, simply select a node when searching. It's a good choice to use two binary heap for inserting node, because the time complexity of the insert and delete nodes is only O(log n) [6].

The path calculated by using the A* algorithm in game scene which often appears jagged sharp and is very unreal. Therefore, it is necessary to smooth the path to achieve the purpose by changing the valuation function. When a new node changes the direction of the original path, increase the value of node generation, and gain the smooth path. The player will become an ideal point When making a path-finding plan by using the A* algorithm, there is a risk for those path planning in some cases [7]. In a real scenario, the player can't move when meets any other obstacles because of its collision box. Change the barrier adjacent nodes' priority, when the path is generated regardless of adjacent nodes of obstacles, allowing players to complete path-finding.

III. "NAVMESH GRID" PATH-FINDING

"Navmesh grid" path-finding is a path-finding algorithm integrated in the Unity 3D, only need to set up the map and add "Navmesh Agent" component for players. Component is the basic unit of the material form in the Unity 3D. A basic component of an object includes a mobile component, a grid component, and rendering. The position of the object can be controlled by moving the component, the shape of an object can be created by the grid component, and the material of the object can be rendered by the rendering component. Build a player who can find the way through different component.

A. Theory of "Navmesh Grid"

It's same with A* algorithm, "Navmesh" also needs to divide the map into a polygon. But the A* algorithm will work well if the polygon is regular, it isn't necessary for the latter. There are many other reasons in real game scene, such as Item properties, collision, Coordinate dimension, data storage, file size etc. The data design of item need to consider the theme of entertainment, Games nature, audience groups [8], operating platform, and other factors. We can detect the collision between the two objects in Unity 3D, but also can detect the collision between the specific collision, and even the use of ray projection detection collision [9]. Light irradiation is the most commonly algorithm used in "Navmesh" path-finding. As shown in Figure 3, the navigation mesh is composed of arbitrary convex polygons. Gray grid represent inaccessible area, white grid represent enterable area. All polygon vertexes in the grid store in clockwise order.

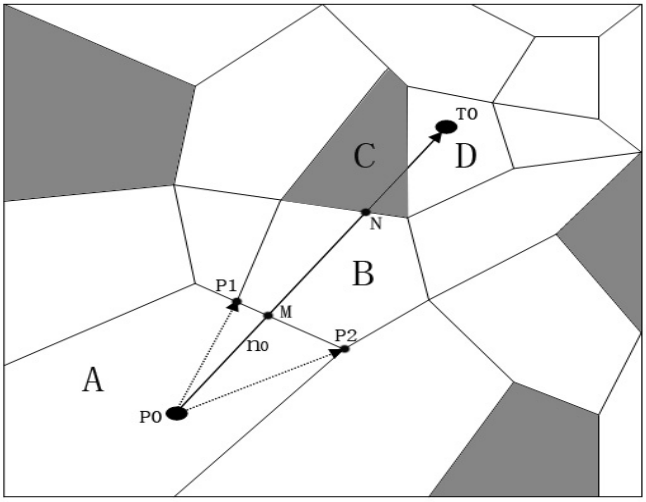

Fig. 3. Light irradiation

As shown in figure 3, it needs to calculate the convex polygon where P0 passed if P0 reach T0 in D region. First, we need to calculate the relationship between P0 and polygons, intersection or parallel, and the side of ray piercing if it's intersect. Taking polygon A and ray r=(p0,n0) as an example, determining the relationship between A and ray r. Traversing A of each side e of the polygon, then get the begin point and end point of the each side. As shown in figure 3, a vector start value as shown in formula (2):

$$start = \xrightarrow[(P0,P1)]{} \times n0 \tag{2}$$

Set the vector end value as shown in formula (3):

$$end = \xrightarrow[(p0,p2)]{} \times n0 \tag{3}$$

Compare direction of two vectors in the Z component by using the right-hand rule of fork. If it's same, the start vector and end vector are located on the same side of the n0; if not, the edge e doesn't intersect with ray r. If $start.z > 0$ and $end.z < 0$ is true, e is an outlet side in the polygon A intersected with ray r. If $start.z < 0$ and $end.z > 0$ is true, then e is intersection in the polygon A intersected with the opposite direction of ray r. Find out the outlet point and edge in the polygon intersected ray r, and determine whether the other side edge of the polygon is enterable. If it's the polygon of obstacle, skip detection, and gain the other side of the side. Repeat sequentially until you find the target point.

B. Implementation of path-finding

After creating the scene in Unity 3D, open "Windows Navigation", select the walk ground, make sure to add "Mesh Renderer" component, check the Navigation Static, set the Navigation Area to walk, adjust the parameters under the Bake menu, set the walking road's radius, height etc. Click on the Bake button to bake, after the completion of baking the obstacles in the above steps, set "Area Navigation" to "Not Walk", set the radius of baking. Add "Navmesh Agent" component in player and add the following functions in the code:

1) Get the target position.

2) Get path-finding component.

3) Control players' movement to the target point by the "SetDestination" function.

It will automatically calculate the required travel path in the path-finding process.

The above operation procedure is the most basic in the path-finding. There are still some necessary steps when we develop practical projects. Game maps are often not simple 2D graphics but the combination of 2D and 3D, so we need to select the appropriate map according to the game. In some large games, the player is often more than one, so we should plan the player's independence before making the game. The game AI is also a very important problem in the path-finding game, player will get a better experience if the virtual enemy has a more higher intelligence. The focus of the interactive product lied on the process of product display, but also in real-time interaction on system with the user [10]. It is an important problem how to display the path in the game in a graphical way and it can be dynamically changed. Game world is a virtual place, and there are a variety of unexpected scenarios, so we need to dynamically plan based on the scene of the production process.

C. The Actual Game Development

In most of the scenes, there are some objects equipped with collision box. The player can't pass due to the collision and stuck in the collision point. It should be coupled for static objects with another judge, and it moves a certain distance to the other direction, continue to find the way. It's necessary to change dynamically the routing path, because the location information is always changing for dynamic objects. Add "Obstacle Navmesh" component, check the Carve option, the role will update the current navigation grid. In the process of moving, the path of the path can be changed dynamically.

It is more convenient to use the corner method when the shape of the grid is too long. Store the vertices of the common edge of each two adjacent polygons between the start point and the target point. Connect the start point with its nearest vertex from the start point, and then connect the next edge of the

point. Determine Whether start point is connected to the second side through the obstacle area, if it's not, update the path, if it is, take two line as a path to the starting point is set to the second point and get a corner of the path.

"Navmesh" grid path-finding is a path-finding algorithm integrated in the Unity 3D. We can write scripts in the bottom layer to achieve the appropriate effect according to the different scene. There are some algorithms which is general algorithm does not suitable apply to each game scene, we can make an algorithm to achieve our goal according to the demands of the game through the preparation of the underlying script.

IV. CONCLUSION

A* algorithm is the most effective method to solve the shortest path in the static grid. There is a good apply in the strategy game of the search, as well as the grid search path. Navmesh navigation grid algorithm is more used in monster path-finding of game scene and dynamic obstacle avoidance scene. In recent years, game AI development soon, a variety of AI technology is introduced to the game, such as genetic algorithm, artificial neural network computing, terrain analysis technology, the group team routing algorithm and A* algorithm. The technique solve well the game path-finding problems. With the development of graphics game, game scenes are more and more complex, the single algorithm cannot be applied to all path-finding scenario. So for different scenes, it still needs put forward the algorithm improvement and application on the path-finding.

ACKNOWLEDGMENT

This paper is partly supported by "the Excellent Young Teachers Training Project (the second level, Project number: YXJS201508)", "Key Cultivation Engineering Project of Communication University of China (Project number: 3132016XNG1606 and 3132016XNG1608)", "Cultural technological innovation project of Ministry of Culture of P.R.China (Project number: 2014-12)", and partly supported by "The comprehensive reform project of computer science and technology, department of science and Engineering". The research work was also supported by "Chaoyang District Science and Technology Project (CYXC1504)".

REFERENCES

[1] S. M. LaValle, Planning algorithms, Cambridge University Press, 2006.

[2] T. K. Whangbo, "Efficient Modified Bidirectional A * Algorithm for Optimal Route-Finding," New Trends in Applied Artificial Intelligence, Japan, vol. 4570, pp. 344–353, June 2007.

[3] M. Fu and B. Xue, "A Path Planning Algorithm Based on Dynamic Networks and Restricted Searching Area," IEEE International Conference on, Jinan, pp. 1193–1197, August 2007.

[4] X. Chen, Q. Fei and L. Wei, "A New Shortest Path Algorithm based on Heuristic Strategy," World Congress on Intelligent Control & Automation, Dalian, China, pp. 2531–2536, June 2006.

[5] Z.X. Li, M. Anson and G.M. Li, "A procedure for quantitatively evaluating site layout alternatives," Construction Management & Economics, UK, vol. 19, pp. 459–467, May 2001.

[6] P. Lester, "Using Binary Heaps in A* Pathfinding," http://www.poli-cyalmanac.org/games /binaryHeaps.htm, unpublished.

[7] D. LIN, "A return docking algorithm for indoor cleaning robot," Journal of Chongqing University of Science and Technology, China, vol. 12, pp. 110–112, May 2009.

[8] E. Adam, Fundamentals of Game Design, Berkeley: New Riders, February 2010.

[9] Y. Lang, "Research on collision detection method in unity," Software Guide, China, vol. 13, pp. 24–25, July 2014.

[10] H.X. Guo, "A research about interaction mechanism on Unity and HTML," Coal Technology, China, vol.30, pp. 228–229, September 2011.