2024 4th International Conference on Neural Networks, Information and Communication (NNICE) | 979-8-3503-9437-5/24/$31.00 ©2024 IEEE | DOI: 10.1109/NNICE61279.2024.10498463

Artificial intelligence pathfinding based on Unreal Engine 5 hexagonal grid map

Hongbo Xing[a], Mengyao Chai, Yaju Song*
College of Physics and Electronic Engineering, Hengyang Normal University
Hengyang, China
16673484042@163.com[a], yjsong@hynu.edu.cn*

***Abstract*—This paper proposes an A* artificial intelligence pathfinding algorithm based on the hexagonal grid map of Unreal Engine 5. This algorithm utilizes the rich tools and resources provided by Unreal Engine 5 to evaluate each node through a heuristic function, thus finding the shortest path. Test results show that this algorithm not only can quickly find the shortest path, but also can effectively avoid obstacle grids, with advantages such as high efficiency, flexibility, and scalability. This research result has high practical value for solving pathfinding problems on hexagonal grid maps and can provide strong support for game development and other fields of artificial intelligence applications.**

Keywords-Unreal Engine 5; Artificial intelligence pathfinding algorithm; Hexagonal grid map.

I. INTRODUCTION

The artificial intelligence pathfinding algorithm aims to automatically find the shortest path on complex maps, which has wide application value in urban planning, logistics transportation, autonomous driving, and game design[1]. Unreal Engine 5 is currently one of the most advanced game development engines, providing rich tools and resources. Therefore, optimizing artificial intelligence pathfinding algorithms based on Unreal Engine 5 has become a research hotspot. In addition, compared with square grid maps, hexagonal grid maps can contain more information and adapt to different scene requirements, with advantages such as high efficiency, flexibility and uniformity[2][3]. However, traditional pathfinding algorithms often have problems such as slow operation or unable to find the shortest path when dealing with hexagonal grid maps[4]. To solve this problem, this paper proposes an A* artificial intelligence pathfinding algorithm based on the hexagonal grid map of Unreal Engine 5, which evaluates each node through a heuristic function to find the shortest path. The test results show that the algorithm can quickly find the shortest path and effectively avoid obstacle grids. This result has high practical value and application prospects for solving pathfinding problems on hexagonal grid maps. It can provide strong support for game development and other fields of artificial intelligence applications.

II. IMPLEMENTATION OF HEXAGONAL GRID MAP BASED ON UNREAL ENGINE 5

In this section, we designed a set of blueprint classes based on Unreal Engine 5 to construct hexagonal grid maps using a structured approach. This method relies on the tools, features provided by the engine, as well as custom code and plugins. The specific steps are as follows:

Firstly, we design the BP_HItem blueprint class, which inherits from the Actor class. It is used to handle the display of hexagonal grid instances, coordinates, and pathfinding costs, while also presenting appearance color features. Secondly, the BP_HMeshController class also inherits from the Actor class. It can create instances of BP_HItem based on spatial and cubic coordinate information, and maintain the correspondence between coordinates and instances to ensure precise control and fast retrieval. Thirdly, the H_Info class inherits from the ActorComponent class, and is used to store key information about the grid, such as spatial coordinates, obstacle identification, path point status, A* algorithm cost value, and the cubic coordinates of the previous grid, becoming the core of data storage and management. Finally, the BP_HController class serves as the core, responsible for initializing grid data, creating displays, and implementing pathfinding algorithms.

At the same time, we implemented the initialization, data creation, and display functions of the hexagonal grid through a constructor function. This constructor function can not only be used to encapsulate implementation details, parameter adjustments, and real-time display, but also calculate the cube coordinates corresponding to H_Info by using the row and column numbers and the offset coordinates, thereby obtaining the dictionary key. The following is the formula for the relationship between coordinates and column widths:

$$\begin{cases} X = W - H/2 \\ Y = -(W - H/2 + H) \\ Z = H \end{cases} \quad (1)$$

where W represents the number of columns and width of the hexagonal grid, H represents the number of rows and height, X, Y, and Z are the coordinates of the cube, and the sum of these three coordinates is 0.

With the above prerequisites, using the built-in AddHinfo method of the Unreal Engine and setting the initialization parameters of H_Info, the setting of the dictionary values is completed. Creating the grid is divided into three main steps: first obtaining an instance of BP_HController, then binding and generating the delegate event of the obstacle based on the coordinate information provided by HexInfList, and finally calling the methods within BP_HController to display the coordinates, starting point style, and ending point style based on the corresponding coordinates. The above constitutes the main method and process for implementing a hexagonal grid map in Unreal Engine 5. The constructed effect is shown in Figure 1, where green represents the starting point, yellow

represents the target point, red represents the obstacle, and X, Y, Z are cube coordinates.

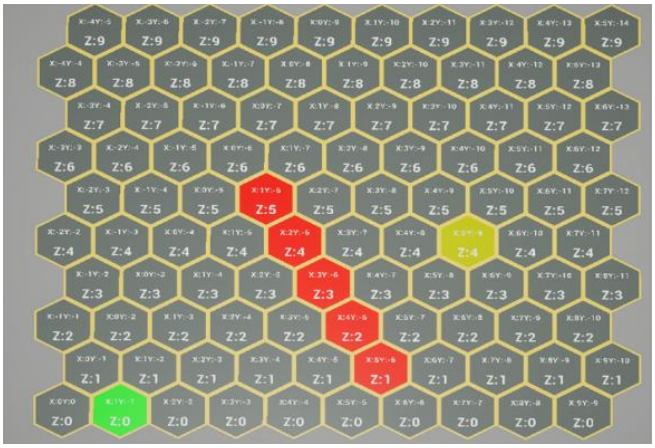

Figure 1. The map construction effect in Unreal Engine.

III. DESIGN OF HEXAGONAL GRID MAP PATHFINDING ALGORITHM BASED ON UNREAL ENGINE 5

A. *Breadth first search algorithm*

Breadth-first search (BFS) is primarily a comprehensive search strategy that is widely used in blind search. This algorithm does not target a specific direction or location for search, but instead covers all nodes in the map comprehensively to find the shortest path. Although BFS can find the shortest path, it requires a large amount of memory resources during its operation[5]. Figure 2 shows how the dijkstra algorithm finds the target.

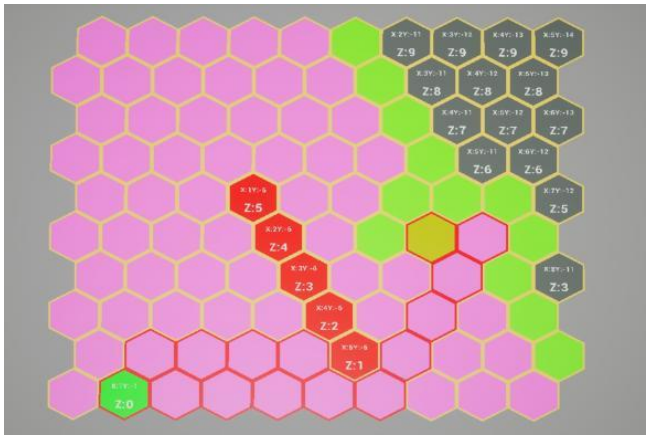

Figure 2. BFS pathfinding processing.

B. *Dijkstra algorithm*

Dijkstra's algorithm is an improved algorithm based on breadth-first search. It uses greedy algorithms to calculate by adding weights to the edges between each vertex, thereby finding the optimal path. This algorithm can avoid long routes and areas that are not suitable for walking, showing more intelligent pathfinding capabilities. Compared to traditional breadth-first search algorithms, Dijkstra's algorithm is more flexible and efficient in path planning[6]. However, the algorithm still has a tendency towards blind search, and the pathfinding process lacks clear directionality.

Figure 3 shows how the dijkstra algorithm finds the target. In this figure, the search process is completed by a radial search segmentation. Starting by detecting the path around the target, the target is captured until the detected target coordinates are the same as the end target coordinates, at which point the program ends and the search path is found by backtracking.

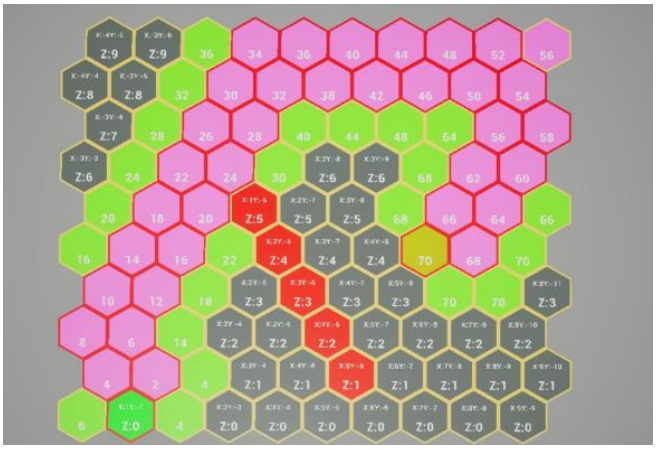

Figure 3. Dijkstra pathfinding processing.

C. *Pathfinding design based on A* algorithm*

The traditional A* algorithm uses Manhattan distance as a heuristic function, but in a hexagonal grid map, the traditional binary formula as a heuristic function is not suitable. Therefore, this paper proposes a heuristic function based on a hexagonal grid map, in which the Manhattan distance formula is extended from binary to ternary, to better guide the pathfinding algorithm. Besides, the method of detecting the target point in the end adopts the method of directly comparing the absolute coordinates, which is more efficient than the traditional method of traversing the open list. The A* algorithm is a classic pathfinding algorithm that finds the shortest path from the starting point to the target point by assigning a G value and an H value to each node. In the A* algorithm, the heuristic information is represented by the evaluation function F(x) as:

$$\mathrm{F}(x) = G(x) + H(x) \tag{2}$$

The x represents the current evaluation point, G(x) represents the actual cost from the initial node to node x. H(x) is a heuristic function of the formula, and H(x) represents the estimated cost of the optimal path from the current node x to the target node.

In grid maps, the heuristic function H(x) usually uses the Manhattan distance to estimate the distance. The Manhattan distance formula isAssuming that the processing power of the satellite S_i is expressed as the number of CPU cycles required to process one bit of input data, then the processing time of the data volume L_i is:

$$\mathrm{H}(x) = |A_x - B_x| + \left|A_y - B_y\right| + |A_z - B_z\,| \tag{3}$$

Algorithm 1 Search for A* within a hexagonal grid map

Input: G, H, F, OpenList = [start], Start.G=0, Start.F=Start.G+H(start,End)
Output: OpenList

```
Function TracePath(Start,End)
If The coordinates of Start are exactly equal to
Return the OpenList
Else
    Get a first node Start in OpenList and remove it from OpenList
    for Traverses the child node B of each Start do
        If B is not on the open or closed list or is not a barrier point then
            Find out the G value of B and calculate the F and H values of B
            If node B is not in the open list then
                Join the B node to the open list
                Update OpenList F and G values and parent node information
        Else
            If The F value of B is less than the F value of the OpenList then
                Update OpenList F and G values and parent node information
    Sort by F value in OpenList
    TracePath( OpenList[0],End)
    Output OpenList
```

D. Implementing a hexagonal grid map pathfinding algorithm within Unreal Engine 5

Now we introduce in detail the data structure and construction method of hexagonal grids. By using Actors, we created a hexagonal grid blueprint with features such as displaying coordinates, displaying generation values, changing colors, and generating obstacles when the mouse is pressed. Then we used Actor components to store grid information[7]. Behind each hexagonal grid, we used an Actor component to store the grid's F value, G value, whether it is an obstacle, spatial coordinates, parent node, and whether it has been selected as a path. This information is crucial for implementing pathfinding algorithms. Through the definition of the data structure and method described above, we can create a hexagonal grid system suitable for specific applications, providing basic support for the implementation of subsequent pathfinding algorithms[8].

Next, we learn the principles and structure of the algorithm through A* pseudocode and program the algorithm

using Blueprints. Firstly, create a module to set local variables to facilitate variable calls. Secondly, determine whether the target point has been found. If it has been found, backtrack the path points. If the target point has not been found, add this grid to the closed list, process the surrounding hexagonal grid, find the data with the smallest F value in the open list, and use this minimum as the starting point for another search. The blueprint module flow of the A* algorithm is as follows:

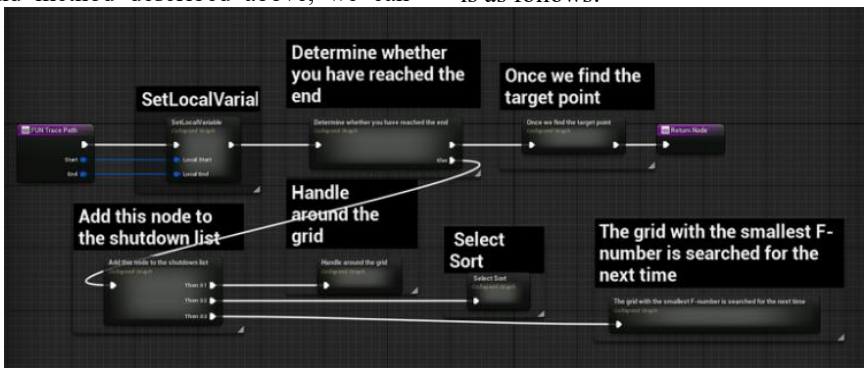

Figure 4. The blueprint module flow of A* algorithm

After making the above preparations, run the editor. The green grid represents the addition of the open list, and the purple grid is the selected path (which has been placed in the closed list)[9]. If there are two grids with the same minimum F value in the open list, the grid with the larger G value is selected as the next grid to be added to the closed list. When the target grid is in the open list, it is moved to the closed list, completing the main loop. Next, we need to determine the final path. The grid in the closed list constitutes the entire path, but not all grids need to be used[10]. Backtracking is required, returning from the target grid to the starting point. At each step, the parent grid of this grid is checked to find the shortest path.

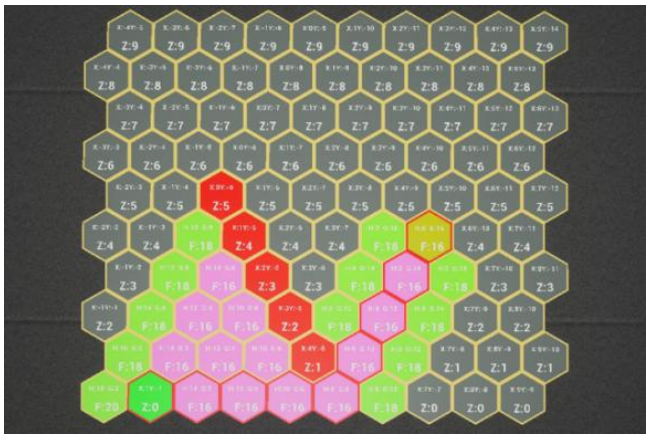

Figure 5. Displaying the running result in Unreal Engine 5

IV. EXPERIMENTS

The A* algorithm has become one of the best solutions to solve the shortest path problem required by today's industry. There are differences in the concept of path area between BFS, dijkstra and A* shortest path problem. Dijkstra detects the target through radial detection, while A* uses a radial layer around the target and then points the best direction to the nearest target point. It continues to repeat until it finds the target. Compared to the experiments in Figure 2 and Figure 4, the A* algorithm has better pathfinding performance than the Djikstra algorithm. The comparison data is shown in Table 1.

Table 1 Comparison of pathfinding algorithms

Experiment	Name of Algoritm	Steps
Figure 2	BFS	2745
Figure 3	Dijkstra	233
Figure 5	A*	121

V. CONCLUSION AND PROSPECT

This paper studies the artificial intelligence pathfinding algorithm based on the hexagonal grid map of Unreal Engine 5. The effectiveness and superiority of the A* pathfinding algorithm have been verified through experimental testing. The algorithm can quickly find the shortest path and effectively avoid obstacles. Although this study has achieved certain results, it still faces challenges such as optimizing algorithm performance, considering dynamic environments, combining with other intelligent algorithms, and promoting practical applications. Future research will focus on these aspects to further promote the development of artificial intelligence pathfinding technology and provide more effective solutions to solve practical problems.

REFERENCES

[1] Algfoor Z A, Sunar M S, Kolivand H. A comprehensive study on pathfinding techniques for robotics and video games[J]. International Journal of Computer Games Technology, 2015, 2015: 7-7.

[2] Xie Y, Chen Y, Wang Z, et al. AI intelligent wayfinding based on Unreal Engine 4 static map[C]//Journal of Physics: Conference Series. IOP Publishing, 2022, 2253(1): 012016.

[3] Zhang X, Zhang X. Based on Navmesh to implement AI intelligent pathfinding in three-dimensional maps in UE4[C]//Proceedings of the 2022 5th International Conference on Algorithms, Computing and Artificial Intelligence. 2022: 1-5.

[4] Zikky M. Review of A*(A star) navigation mesh pathfinding as the alternative of artificial intelligent for ghosts agent on the Pacman game[J]. EMITTER International journal of engineering technology, 2016, 4(1): 141-149.

[5] Wayama K, Yokogawa T, Amasaki S, et al. Verifying Game Logic in Unreal Engine 5 Blueprint Visual Scripting System Using Model Checking[C]//Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering. 2022: 1-8.

[6] Rüdel M O, Ganser J, Weller R, et al. Unrealhaptics: A plugin-system for high fidelity haptic rendering in the unreal engine[C]//Virtual Reality and Augmented Reality: 15th EuroVR International Conference, EuroVR 2018, London, UK, October 22–23, 2018, Proceedings 15. Springer International Publishing, 2018: 128-147.

[7] Mahmud S, Ghosh S S. Unreal Engine Blueprint: Developing Hack and Slash Using Spawn Projectile Method[C]//2021 3rd International Conference on Electrical & Electronic Engineering (ICEEE). IEEE, 2021: 81-84.

[8] Hasugian A H, Pristiwanto P, Rikki A. Perancangan Memory Game dengan Menggunakan Unreal Engine[J]. MATIAS, 2019, 1(1): 1-7.

[9] Zhang J,Juan Z. Implementation and Optimization of Particle Effects based on Unreal Engine 4[J]. Journal of Physics: Conference Series,2020,1575(1).

[10] Manzoor M I, Kashif M, Saeed M Y, et al. Applied Artificial Intelligence in 3D-game (HYSTERIA) using UNREAL ENGINE 4[J]. Applied Artificial Intelligence, 2018, 9(9).