IJFANS INTERNATIONAL JOURNAL OF FOOD AND NUTRITIONAL SCIENCES

ISSN PRINT 2319-1775 Online 2320-7876

 UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, Oct 2021

A Path Based Finding Computer Games in Modern ERA

Dr. Koparthi Suresh[1] , Dr. RVVSV Prasad[2]

[1]Professor, Bhimavaram Institute of Engineering & Technology, Pennada, Bhimavaram, India[1]

[2]Professor, Swarnandhra College of Engineering & Technology, Narsapur, - 534275, India[2]

sureshkgrl@gmail.com[1], ramayanam.prasad@gmail.com[2]

Abstract:

The process of pathfinding in video games has been studied for quite some time. It's the most talked about and infuriating artificial intelligence (AI) issue in games right now. There were several attempts to discover an optimum solution to the shortest route issue prior to the development of the A* algorithm. These included Dijkstra's algorithm, bread first search method, and depth first search algorithm. Since its inception, thousands of researchers have been drawn in to contribute to it. Many other algorithms and methods based on A* were developed. Several well-known A*-based algorithms and methods are analyzed and compared in this study. Its goal is to investigate the connections between the different A* algorithms. The first part of this article provides a high-level introduction to pathfinding. Then, using that discussion of the A* algorithm's inner workings as a springboard, we present many optimization methods from various vantage points. Finally, a conclusion is made and a number of real-world examples of the pathfinding algorithms' implementation in actual games are provided..

Introduction

Finding the quickest route between two places is what we mean when we talk about pathfinding. Some examples of this kind of issue include those involving public transportation, telephone traffic routing, navigating mazes, and programming robot routes. The popularity and frustration of the pathfinding issue in games has grown with the significance of the gaming industry. Characters in games like role-playing games and real-time strategy games are commonly given tasks that need them to go from where they currently are to a different place, either of which may have been chosen by the player. The most typical pathfinding challenge in a video game is finding the least restrictive route across randomly generated landscapes.

The exponential growth of the game's complexity quickly outpaced early solutions to the pathfinding problem in computer games, such as depth first search, iterative deepening, breadth first search, Dijkstra's algorithm, best first search, the A* algorithm, and iterative deepening A*. Pathfinding difficulties in a more complicated context, when time and resources are limited, need more effective solutions.

Many academics are banking on a faster version of the A* algorithm to meet the ever-evolving demands of the field after seeing its tremendous success in route finding [1]. game. Over the last several decades, a lot of work has gone into improving this algorithm, and hundreds of new, improved versions have been successfully implemented. Heuristic method enhancements, map representation enhancements, the introduction of new data structures, and memory footprint reductions are all examples of such improvements. In the following paragraphs, you'll find an introduction to some of the A* approaches employed extensively in today's video game development.

A* algorithm

A* is a general-purpose search algorithm that has numerous applications beyond only

ISSN PRINT 2319-1775 Online 2320-7876

 UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, Oct 2021

pathfinding. The A* algorithm, used for pathfinding, iteratively investigates the most promising uncharted area it has encountered. If the destination is already known, the algorithm stops after exploring its immediate vicinity; otherwise, it remembers all of the possible destinations within a certain radius. When it comes to artificial intelligence (AI) in video games, A* is by far the most often used route finding algorithm. [2].

1. Add the starting node to the open list.
2. Repeat the following steps:
   a. Look for the node which has the lowest f on the open list. Refer to this node as the current node.
   b. Switch it to the closed list.
   c. For each reachable node from the current node
      i. If it is on the closed list, ignore it.
      ii. If it isn't on the open list, add it

Fig. 1 Pseudocode of A* [3].

g(n) is the precise cost from the origin to any destination in the standard A* language. point n, h(n) is the expected cost from point n to the destination, and f(n)=g(n)+h(n) is the function we're interested in. A detailed diagram of the method is shown in Fig. 1.

In 1968 [4], Hart, Nilsson, and Raphael demonstrated that A* has a number of desirable characteristics. To begin, if there is a way to go from the beginning to the end, A* will discover it. An acceptable heuristic h(n) is optimum if and only if it consistently under- or overestimates the true lowest route cost from n to the objective. Thirdly, A* is the heuristic that makes the best use of it. In other words, A* analyzes fewer nodes than any other search strategy that use the same heuristic function to determine the best route.

Although A* is the most prevalent option for pathfinding in computer games, how it is implemented relies on the game's design and how the environment is represented inside the game. For instance, there are 1,000,000 potential squares to look for in a rectangular grid with 10001000 cells. A lot of effort is required to locate a feasible route on such a map. This suggests that restricting the search space might greatly improve A*'s performance. In Chapter 3, we cover a variety of enhancements.

2. A* Optimizations

Several possible A* optimizations from four distinct angles are discussed here, as well as a discussion of certain well- known A*-based algorithms..

Search Space

To find their way about in a game, computer-controlled characters must rely on a fundamental data structure called a search space representation. The success of your game's movement and pathfinding relies on your ability to accurately describe the game world's search space using an appropriate data structure. A simpler search space will imply less work for A*,

ISSN PRINT 2319-1775 Online 2320-7876

 UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, Oct 2021

and less effort means the algorithm can run quicker, as shown by the preceding example. Rectangular grids (Fig. 2a), quadtrees (Fig. 2c), convex polygons (Fig. 2d), visibility points (Fig. 2e), and generalized cylinders (Fig. 2f) are all examples of such representations.

In the parts that follow, we'll take a look at two well-known variations on the A* algorithm, both of which aim to improve upon it by narrowing the search space.

Hierarchical Pathfinding A* (HPA*)

When it comes to speeding up the pathfinding process, hierarchical pathfinding is unrivaled. Organizing the world in a hierarchy may help simplify the difficulty. Think about the challenge of getting from L.A. to Toronto. Given such a specific

Given a road map of North America with all roads labeled with driving lengths, an A* implementation can determine the best route to take, albeit this might be a time-consuming process due to the map's vastness. However, at this granularity, a hierarchical route finding algorithm would be useless. Abstraction allows one to rapidly locate an exit. arranging a large-scale route at the city level first, and then arranging the inter routes at each city passing through, may be a more efficient solution to the issue outlined above.

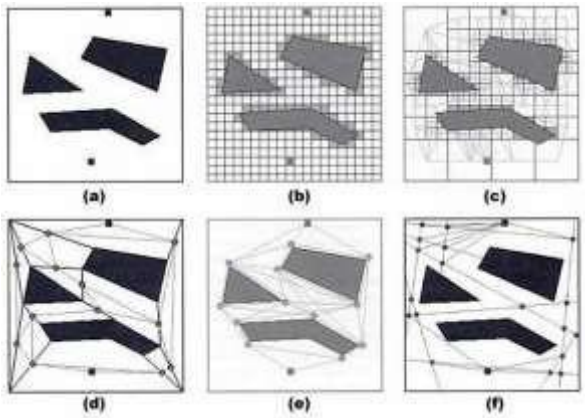

Fig. 2 Five ways to represent search space [5].

HPA* (Hyper-Parallel A*), introduced in [6], is an A*-based search method that returns nearly optimal results significantly more quickly. This method does not rely on any particular domain. Scalability for enormous problem spaces is improved by allowing the hierarchy to include more than two levels. The procedure consists of three stages. Step one is to make one's way to the neighborhood's edge, where the journey may officially begin. The second stage entails looking for a route connecting the first neighborhood's edge with the final neighborhood's edge. This process takes place on a more abstract level, where searching is less time-consuming. After reaching the neighborhood's edge, go to the target location to finish the route. In [6], it is shown that HPA* is 10 times quicker than a basic A*. The added expense of an abstraction layer might be a major

ISSN PRINT 2319-1775 Online 2320-7876

 UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, Oct 2021

drawback of this method.

Navigation Mesh (NavMesh)

Another well-liked method for artificial intelligence navigation in 3D spaces is NavMesh. In order to depict the "walkable" surface of a 3D world, a NavMesh is created out of convex polygons. It's an easy-to-understand layout that can help computer-controlled characters go about the game's environment.

A NavMesh illustration is shown in Fig. 3b. From pol2, the figure travels to pol4, where he or she arrives at their goal. In this instance, we don't begin with when the target point is located inside a polygon. Therefore, it is up to the character to decide which polygon it will visit next. Keep doing this until the polygon containing the character and the polygon containing the objective are the same size. Then, the protagonist may go directly to the objective.

In contrast to the waypoint graph seen in Fig. 3a, the NavMesh method is guaranteed to locate a nearly optimum route while exploring far less data. When compared to a waypoint graph, a NavMesh's pathfinding behavior is far more advantageous.

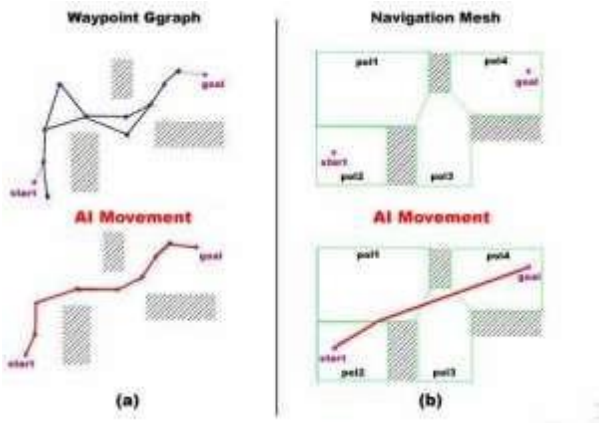

Fig. 3 Different representations of waypoint graph and NavMesh [7]

Creating navigation meshes that are highly simplified and easy for pathfinding is critical to achieving a good pathfinding. Tozour [9] describes how to construct a good navigation mesh and proposes a number of optimization techniques for navigation mesh.

Heuristic Function

The secret to the success of A* is that it extends Dijkstra's algorithm by introducing heuristic approach. Dijkstra's algorit hm is guaranteed to find a shortest path in a connected weighted graph as long as none of the edges has a negative value but it is not efficient enough because all the possible states must be examined. However, A* algorithm improves the computational efficiency significantly by introducing a heuristic approach. Using a heuristic approach means, rather than an exhaustive expansion, only the states that look like better options are examined. The heuristic function used in A* algorithm is to estimate the cost from any nodes on the graph to the desired destination. If the estimated cost is exactly equal to the real cost, then only the nodes on the best path are selected and nothing else is expanded. Thus, a good

ISSN PRINT 2319-1775 Online 2320-7876

 UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, Oct 2021

heuristic function which can accurately estimate the cost may make the algorithm much quicker.

On the other hand, using a heuristic that overestimates the true cost a little usually results in a faster search with a reasonable path [10]. Fig. 4 shows the growth of the search using various heuristic costs while trying to overcome a large obstacle. When the heuristic equals to zero (shown in Fig. 4a), A* algorithm turns to Dijkstra's algorithm. All the neighboring nodes are expanded. When the heuristic uses the Euclidean distance to the goal (shown in Fig. 4b), only the nodes that look like better options are examined. When the heuristic is overestimated a little (shown in Fig. 4c), the search pushes hard on the closest nodes to the goal. Thus, overestimating the heuristic cost a little may result in exploring much fewer nodes than non-overestimation heuristic approaches. However, how much should the cost be overestimated is a tricky problem. No general solution exists at present.

Memory

Although A* is currently the best search algorithm available, it has to be utilized carefully to avoid unnecessary resource waste. When searching in big and complicated settings, the A* algorithm's memory requirements to keep track of the state of each search may quickly become prohibitive. It's a challenging challenge in game AI to reduce the amount of memory needed for pathfinding. Numerous efforts have been made in this direction.

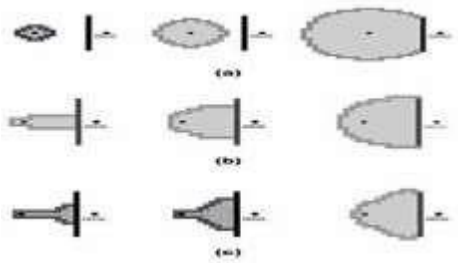

Fig. 4 Comparison between different heuristics [10].

The most common method of preventing memory from being wasted is to allocate at least some of it in advance [10]. The basic concept is to set aside some RAM (the "Node Bank") before A* is run. If all available memory is used up during execution, a new buffer must be constructed before the search may continue. In order to save unnecessary memory use, the size of this buffer is dynamic. Because of its size,

The intricacy of one's surroundings is the primary determinant of how little memory one needs. As a result, the approach must be tuned before it can be used in practice.

ISSN PRINT 2319-1775 Online 2320-7876

 UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, Oct 2021

Computing the whole route in little chunks is another option for minimizing A*'s storage needs. IDA* (Iterative Deepening A*) is a widely used variation of the A* algorithm, and its central idea is this. When the total cost of a route, f(n), is more than a certain limit, the path is eliminated from IDA*. Since g(start_node)=0, the initial threshold for IDA* is set to h(start_node) in this example. The next step is to keep growing the nodes around the current one until a solution is discovered that scores below the threshold or until all possible solutions have been exhausted. A new search is initiated once the threshold is raised by one. The key benefit of IDA* over A* is its much lower memory requirements.

Data Structure

An initialized node from the Node Bank (discussed in Section 3.3) must be stored conveniently for later use. Since a hash table can store and retrieve information quickly, it's possible it's the best option. Using this hash table, we can quickly determine if a node is on the CLOSED or OPEN list.

The most efficient method of managing an OPEN list is via a priority queue. A binary heap may be used to implement it. Not much has been done to improve the efficiency of OPEN and CLOSED list maintenance via the introduction of novel data structures. It is likely that using a different data structure to hold the information would greatly accelerate A*.

Relevant Applications in ComputerGames

The A* algorithm is often used in video games because it is an efficient pathfinding technique. The technique is straightforward in theory, but it's not so simple to put into practice in a practical computer game. In this part, we'll look at how alternative map representations might affect pathfinding performance, and we'll use an example from a well-known online game to do so.

Pathfinding Challenge in Game Industry

The real-time strategy genre has a true classic in Age of Empires. Locations on the map are represented by grids. With a 256 by 256 grid, there are 65,536 potential sites. Moving a military unit may be thought of as no more complicated than navigating a labyrinth with an item. Using the A* method, Age of Empires is analyzed. Despite its flawless appearance in theory, the poor pathfinding in Age of Empires has irritated many gamers. Such issues arise, for instance, when a Half of the units in the group get entangled in the trees as they try to traverse the forest to reach their destination (see Fig. 5). This kind of thing is inevitable, particularly as forest density rises.

ISSN PRINT 2319-1775 Online 2320-7876

 UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, Oct 2021

Fig. 5 A screenshot of Age of Empires II. [11]

As may be seen in Fig. 6, the hexagonal tiles used to depict real-world places in the strategy game Civilization V. A pathfinding method is used to direct the military unit as it traverses a set of "walkable" hexagonal tiles to reach its destination. Although it is the most recent installment in the Civilization series, launched in November 2010, Civilization V still has poor pathfinding, much like Age of Empires.

Fig. 6 A screenshot of Civilization V. [12]

In first-person shooters like Counter-Strike, where there are just a handful of troops in motion at once, A* performs significantly better than in strategic games with hundreds or thousands of units. One possible reason is that the gaming environment has become considerably more dynamic as the number of moving units has increased exponentially, making it difficult to offer optimum pathways for hundreds or thousands of units in real time utilizing limited CPU and memory resources. Massively

Multiplayer online role-playing games (MMORPGS) like World of Warcraft also heavily rely on real-time pathfinding.

Comparison between Map Representations

Waypoint graph is a popular technique to represent map locations. Waypoints are a set of points that refer to the coordinates in the physic space. It is designed for navigation purpose and has been applied to a wide rangeof areas. Most game engines support pathfinding on a

ISSN PRINT 2319-1775 Online 2320-7876

 UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, Oct 2021

waypoint graph. Although it works well for most 2D games, it requires tons of waypoints to achieve an adequate movement in a 3D game because of the complexity of the environment. Thus, a new technique called NavMesh is created. As mentioned in Section 3.1.2, NavMesh only requires a couple of polygons to represent the map. It results in a much more quickly pathfinding because less data is examined.

Five reasons why NavMesh works better than waypoint approaches in 3D games using World of WarCraft as an example are addressed by Tozour in 2008 [8]. It shows the difference between waypoints approaches and NavMesh when representing a complex 3D environment.It uses the town of Halaa in World of WarCraft as an example as shown in Fig. 7. In Fig. 7a, it uses 28 waypoints to represent the possible locations while on the other hand, as shown in Fig. 7b, only 14 convex polygons are used. The movement in Fig. 7b also acts much more like an actual human than the movement in Fig. 7a.

Conclusion

In this work, we take an A*-optimized look at certain well-known algorithms and methods. There is a clear connection shown between the A* algorithm and its many iterations. There is much more to artificial intelligence in games than just the pathfinding algorithm at its foundation. The hardest part is figuring out how to put the algorithm to use in practice. The A* algorithm is widely considered the best pathfinding algorithm available. Since A* is provably optimum, it is almost impossible to discover a more effective algorithm. A lot of work has gone into making it faster by optimizing it in many ways. Optimizing the underlying search space, decreasing memory use, enhancing heuristic algorithms, and introducing novel data structures are all approaches to boost A* search's speed.

Further study may concentrate on improving the A* algorithm from various angles, or it could investigate combining several optimization methods into a single, more effective one. Not all computer games employ artificial intelligence, thus applying the methods mentioned above to such games would be a useful addition to the game AI community.

The methods discussed in this article are frequently employed in today's video game development. This article reviews them since they are now the most discussed subjects in the academic field of pathfinding, and many academics are working hard to implement them in actual games. The gaming industry could benefit from this study by learning where pathfinding research is headed in the near future.

Navigating from A to B using waypoint graph.

(a) Navigating from A to B on NavMesh.

Fig. 7 Comparison between wapoing graph and NavMesh [8].

References

[1] "Smart Moves: Intelligent Path-Finding," by B. Stout (pp.28-35) in 1996's Game Developer Magazine.
[2] For example, see [2] "Amit's A* page" by the Stanford Theory Group at http://theory.stanford.edu/amitp/GameProgramming/ASta rComparison.html (last visited Oct. 12, 2010).
[3] In 1998, Morgan Kaufmann Publishers in San Francisco published a book written by Nils Nilsson called Artificial Intelligence: A New Synthesis.
[4] "A formal basis for the heuristic determination of minimum cost paths," [4] P. Hart, N. Nilsson, and B. Raphael. 1968 IEEE Transactions on Systems, Computers, and Networks (Syst.
[5] According to [5] B. Stout, "The basics of A* for path planning," in Game Programming GEMS, Charles River Meida, USA, 2000, pp.254-262.

ISSN PRINT 2319-1775 Online 2320-7876

 UGC CARE Listed (Group -I) Journal Volume 11, Iss 10, Oct 2021

[6] According to [6] "Near optimal hierarchical path-finding," by A.Botea, M.Mueller, and J.Schaeffer, published in J. GD, volume 1, issue 1, pages 7–28, 2004.
[7] As of 2011-01-13, you may find the "Navigation mesh reference" on the Unreal Developer Network at http://udn.epicgames.com/Three/NavigationMeshReferenc e.html.