Impact Factor 5.8701 (ICI)

International Journal of Engineering Research-Online
A Peer Reviewed International Journal
Articles available online http://www.ijoer.in; editorijoer@gmail.com

Vol.7., Issue.4, 2019
July-August

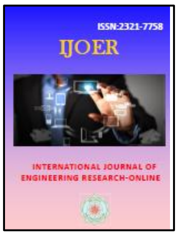
RESEARCH ARTICLE

ISSN: 2321-7758

ROUTE SEARCH AND EXPLORATION OF SPECIFIC OBJECTS IN COMPUTER GAMES

Dr. AMAMER KHALIL MASOUD AHMIDAT[1],
MUHAMAD ABDULLA MUHAMAD ABDUSSALAM[2]
[1]Higher Institute of Medical Technology in Bani Walid.
Email:mkmasoud@hotmail.com
[2]Higher Institute of Medical Technology in Bani Walid.
Email: mrabett34@gmail.com

DOI: 10.33329/ijoer.74.72



ABSTRACT

This paper introduces the routing methods commonly used in modern games as well as the search spaces in which they work. It introduces you to the principles of route guidance in basic matters and thus fits in, for example, as an introduction to the subject. The approach is pragmatic and aims to take into account the limitations of gaming equipment, the difficulty of implementing search methods, and the requirements of the rules of the game.

Key Words: Route Search, Computer Game, Dijkstra's route search method, Route search using the A * -method

1.0 INTRODUCTION

Many computer games have game characters that move around the playing area according to the rules of the game. Generally speaking, the game character must be able to move from any point in the game area where the game character is allowed to move to any point in the game area to which the game character is allowed to move. Other areas should be avoided. In general, it is also desirable that the path of the game character is so-called. Optimum is usually meant to be the shortest possible route, but in addition to the length, the optimal route may consist of other factors such as ease and safety. The rules of the game will ultimately influence what is included in the concept of route optimality. In addition to finding the optimal route, the route search must be fast enough so that it does not interfere with the smooth running of the game.

The game area must be made up of so-called. Search space before a route search can be performed. There are many types of exploration spaces, and different types of exploration spaces are suitable for different game areas. The search space traditionally used is the Waypoint graph, which in its simplest form consists of nodes and arcs connecting them with length. The waypoint network is inexpensive because many search methods work directly, but are also limited. Navigation network (navigation mesh, navmesh) is an alternative way to define the search space[1]. It allows the search space to be defined in more detail as an area rather than just transitions.

In games, the A * algorithm developed from the Djikstra method (pronounced A-star)[2] is very often used for route guidance. The A * algorithm has become a standard in practice because of its versatility, the simplicity of its further implementation. The A * algorithm works directly on the waypoint network. It introduces the A * Path Finding Algorithm and how it can be used to find the paths of game characters in the waypoint network. The following describes the navigation network model and its features, as well as a few methods for establishing a waypoint network. Finally, various methods for speeding up the route search are briefly introduced.

2.0 Game area and route search

Level is the area where the game characters are located and function[3]. The playing area is also commonly referred to as the field and level. The game area contains a variety of essential information about the features of the game area that affect the action of the game characters and visualization of the area and the game situation. Usually, the game area contains information about obstacles, difficult and dangerous locations, area entrances and exits, and visual influencing factors such as lighting and game area texture.

2.1 Features of the game area

All information in the game area is placed in a coordinate system whose axes are perpendicular to each other. Usually, the game area is the level at which two axes are needed to define it. If the playing area also takes height into account, a third axis is required. In this thesis, the axes are named so that the X and Y axes define the length and width of the mirror area, and the Z axis the height. The shape is determined by the game design. The rules of the game may require some form and, on the other hand, form may be defined by practical and aesthetic considerations. The same factors affect the size of the gaming area, but often the features of the gaming device, such as the amount of memory, determine the maximum size of the gaming area. Game characters are dynamically moving and acting agents in the game area[4]. Game characters operate in the game area according to its rules. Game characters usually have their own goals to achieve. Goals can be very simple or complex depending on the game and its rules. To achieve their goals, game characters usually move around the game area and interact with the game area and other game characters.

2.2 Finding a route in the game area

The gaming area as such is not well suited for route guidance because it contains a lot of information that is irrelevant to route guidance and the applicable information may not be in a form that can be effectively utilized for route guidance. For this reason, information is mainly added to the game area, which is mainly used for route guidance. This information is generated either manually or automatically and stored in a suitable data structure. Network nodes correspond to allowed locations and arcs to allowed transitions between locations. The nodes store the position of the at least the node in the coordinate system of the game area, and the transitions can be stored in the arcs, which can be more generally understood as the advantage of the transition. The nodes that arc connected are called neighbors. This kind of network is called waypoint graph and is traditionally very commonly used in games for search space[5]. Figure 1 illustrates a waypoint network.

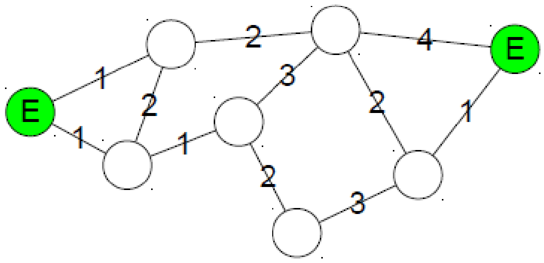


Figure 1: Example of waypoint network. Cells marked with an "E" are exits. The lengths of the arcs, which in this case do not correspond to the geometric length, are marked on the arches.

However, the search space alone is not enough to search for a route, but we also need a method to find the route. Route search methods that are applicable to a multi-point network are based on the Dijkstra route search method and are different optimizations. One of the most commonly used methods in games is A *, which optimizes the Dijkstra method by controlling heuristics to the direction in which the node is supposed to be located. Route search can also be optimized by utilizing the various hierarchies found in the waypoint network, and adding network information to the route point network to provide real-time route search as long as the number of nodes and arcs in the network remains appropriate and the game machine's computing power is high. Usually this is not a problem with modern game machines, but computing power may be needed for other game operations to the extent that route searching needs to be simplified. In such cases, the routes may, for example, be calculated in advance in a table.

2.2.1 Dijkstra's route search method

In 1959, Edsger Dijkstra introduced a method[6] to find the shortest path lengths from a network node to other network nodes. This addition method can be used to find the shortest route from any node in the network to another node in the network. The shortest route is the route whose sum of the lengths of the transitions is as small as possible. Thus, a route is not necessarily the shortest in geometry, but it can be complex and contains more transitions than the route with the least number of transitions. Thus, the features of the exploration space essentially influence the easiest route. The method requires that the lengths of the arcs of the net be positive. The Dijkstra route search method requires input V as network input as well as output node SL, where path lengths are searched. In addition, the method takes an optional input target node SM, whereby the method can terminate the search when a node SM is found and, with a simple additional method, form a route from node SL to node SM. The method maintains three data structures. In the table of shortest lengths, the length of the shortest known path from the source node to each node is stored. This table serves as the return value of the method. In the table of result nodes, the previous one is stored for each node from which the node can be accessed along the shortest known path.

Initially, each node Si of network V is examined. Set length [Si] = 0 if Si = SL, otherwise length [Si] = ∞. Next, set the table's previous [Si] to undefined. Finally, Si is added to the queue unprocessed. Next, the method begins to go through the queue unprocessed. Method in the search string for the node Smin that has the smallest value of length [Smin]. Smin is removed from the queue unprocessed. If the value [Smin] is infinite, then we know that there is no output node ath to the end of the queues of untreated nodes and the search is terminated. On the other hand, if SM is defined and Smin = SM, the search can be terminated and the path node SL to node SM can be formed. Otherwise, the node neighbors of the current node that are not yet queued are searched for and searched for. Let it be PSN arc length from node Smin to node Ni. Let PN be the length output node SL to node Ni. Then PN = length [Smin] + PSN. If PN <length [N], set length [N] to PN and previous [Ni] to Smin. Algorithm 2.1 introduces the Dijkstrareit lookup method in virtual programming language. If the node SM is defined and the length [SM] ≠ ∞, the path to this node can be formed as follows. Let the path be a list of nodes and S be the node to be processed. Set to S = SM. If the value above [S] is specified, add the S list path to the beginning. Then set S = previous [S] and repeat that the previous [S] has been defined. This is continued until the previous [S] is not defined, in which case the list path contains the route from nodeSL to node SM.

```
Table Search Path Lengths (Network Network, Node Source Node, Node Goal Node)

{
        Table length = new Table;
        Table previous = new Table;
        Queue untreated = new Queue;
        iterate (Node Node: Network.Nodes ()) {
                length [knot] = ∞;
                previous [node] = null;
                käsittelemättömät.lisää (node);
        }
        length [departure node] = 0;

        repeat (untreated.size ()> 0) {
                Node smallest =
                unprocessed.SearchMinimum
                (length);
                if (goal node! = null && smallest ==
                goal node) {
                        restore length;
                }
                käsittelemättömät.poista
                (minimum);
                if (length [smallest] == ∞) {
                restore length;
                }
                List neighbors = network.
                neighbors (smallest);
        iterate (Node neighbor: neighbors)
        {
            Length length = length [smallest] + mesh.arch length (smallest, neighbor);
            if (length <length [neighbor]) {
                length [neighbor] = length;
                previous [neighbor] = smallest;
                }
             }
          }
            restore length;
      }
```

Algorithm 1 The Djikstra method written in virtual programming language.

2.2.2 Route search using the A * method

A * is the old route search method. It was developed in 1968 and is still a very popular and widely used method. The A * method is used in addition to routing to solve many other types of problems. Crucially, the problem can be transformed into a spatial network similar to a waypoint network, which can be thought of as a waypoint network overlay. The A * method is often used in games[7]. The A * method is based on the Dijkstra route search method. Unlike the Dijkstan router retrieval method, the A * method looks only for the shortest route in the waypoint network between the starting ground point and not for route lengths to other nodes in the network. The A * method uses the so-called "best first" search method, in which the most promising nodes are examined first. The most promising nodes are inferred heuristically by estimating the total length of the route through each node and updating this information as needed during the search.

The A * method maintains two data structures to store and delete nodes as the search progresses. For the sake of clarity and simplicity, these data structures are referred to as open and closed lists, even if the actual data structures are not lists. During the search, the open list contains nodes that have not been found yet. The closed list contains nodes that have a guest. For each node found, the length of the path that led to the node, as well as an estimate of the length from that node to the finish, and the node from which this node came, are also stored. The length function of the route from the origin to the node is called in the literature as g (x), where x is the node to be considered. . If h (x) is not allowed, then the route found may not be optimal. The function f (x) is given by g (x) + h (x) and its value is used for search control.

At the beginning of the search, the closed list is empty and the open list only contains the parent node. The most promising node is the node with the smallest f (x). The most promising node is added to the list of exclusive ones. If the node is a destination, the search is terminated and the route is formed by proceeding along the chain of result nodes until the departure node is reached. If, on the other hand, the node is not a node, then each of the adjacent nodes of the node is calculated as g (x), h (x), and f (x). If a neighbor node is not in the open list, it will be added there. If the neighbor node is open or closed and g (x) is smaller than the previous value, the value of the neighbor node is recalculated, the result node is set to the current node, and the neighbor node is removed from the excluded list if it was there. If the list of open nodes is cleared before the destination node is reached, the search is terminated and a route is not found. Algorithm 2 introduces the A * method in general terms in virtual programming language.

```
Route FindRoute (Network Network, Node Output Node, Node GoalNode, Heuristics Heuristics) {
        List open = new List;
        List closed = new List;
        avoimet.lisää (output node);
        repeat (open.size ()> 0) {
            Node Most Promising = Open.SearchMinimum Length ();
            if (most promising == paintNode) {
                Path Path = Build Path (FinishNode);
                restore route;
        }
        closed more (most promising);
        List neighbors = network.nighbors (most promising);
        iterate (Node neighbor: neighbors) {
                Number LengthNextile = Most Promising.Length () + web.arch length (most promising, neighbor);
                    if (LengthFor neighbor <neighbor.length ()) {
                    if (open.includes (neighbor)) {
                    avoimet.poista (neighbor);
        }
            if (closed.includes (neighbor)) {
            closed.delete (neighbor);
            }
        }
            if (open.noInside (neighbor) and closed.noInside (neighbor)) {
            naapuri.asetaPituusTähän (length neighbor);
            avoimet.lisää (neighbor);
                naapuri.asetaPituus (the length of neighbor + heuristics.valueLength (neighbor, goalNode));
                naapuri.asetaTulosolmu (most promising);
        }
    }
                }
                return blank;
```

Algorithm 2. A * method written in virtual programming language.

The A * process has many advantageous properties. First, it finds the path departure node to the destination node if such a path exists in the search space. Third, the A * method uses the heuristics more efficiently than any other search method, i.e., the A * method least investigates to find the optimal route. All of these properties are proven in the[8] .A * algorithm is also customizable. The functions g (x) and h (x) can be modified and extended to take into account factors other than the mere lengths of the waypoint grids. For example, the route search can be made more tactile[9].

2.3 Route search using a transition table

In some cases, finding a route in real time while the game is running is undesirable. An example of such a situation could be a situation where the gaming machine's computing power is limited or where the computing power is required for other than route searching. The solution could be to calculate routes in advance to[10].

The method is based on a two-dimensional table, where node identifiers are stored. This kind of table is called a transition table. The transition table has a row column for each node in the search space. Let's define that each row corresponds to the output node and that each column corresponds to the node. For each array in the table, the cell defines the transition that is performed when we want from the release exit node to the finish. Figures 2 illustrate a waypoint network and Fig. 3 illustrate it as a transition table.

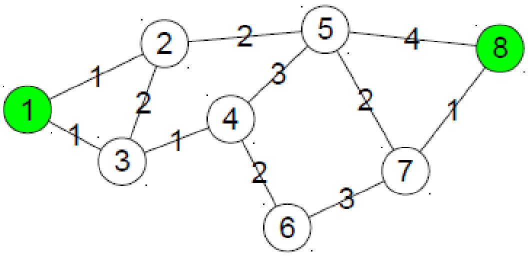
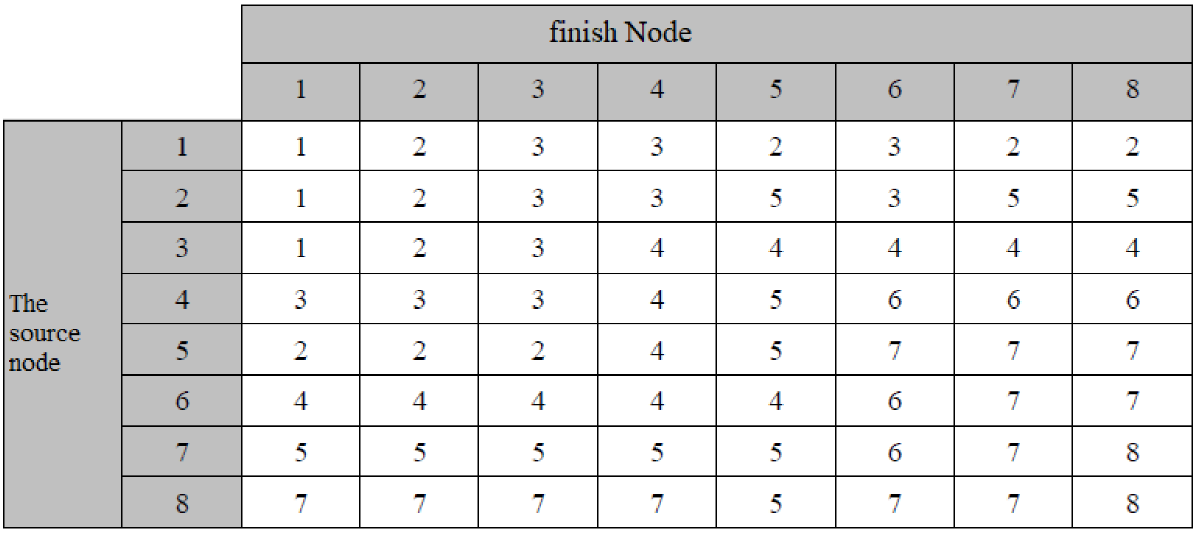


Figure 2: A waypoint network with node identifiers as numbers.



		finish Node							
		1	2	3	4	5	6	7	8
The source node	1	1	2	3	3	2	3	2	2
	2	1	2	3	3	5	3	5	5
	3	1	2	3	4	4	4	4	4
	4	3	3	3	4	5	6	6	6
	5	2	2	2	4	5	7	7	7
	6	4	4	4	4	4	6	7	7
	7	5	5	5	5	5	6	7	8
	8	7	7	7	7	5	7	7	8

Figure 3: Transition table formed from the waypoint network

The search method works as follows. Let's say we want to find a route from Node B to Node B. First, we set Node A as the departure node and add it to the list where the whole route from Node A to Node B is stored. Next, we check whether the output node is the same node as B. If so, we have found the path, so we return it. The cell that explicitly intersects the found row and column contains the identifier of the node that we need to move next to get to the target node B.Set the found node to a parent node and start by checking whether the new parent node is the same as B, etc. Algorithm 3 clarifies the search method.

```
Route FindRoute (Node departure node, Node goal node)
{
        Node Source Node = Source Node;
        Route Route = new Route;
        Route.more (source node);
        repeat (source node! = goal node) {
                source node = TRANSITION TABLE
                [source node] [paint node];
                route.more (source node);
        }
        restore the route
   }
```

Algorithm 3 A method for finding a precalculated path in a table written in virtual programming language.

For example, the A * method can be used to search for a path to the corresponding node in the output node corresponding to the table cell and to store the first transition of the found path in the cell.

3.0 GAME SPACE EXPLORATION SPACES

This chapter introduces frequently used game area types and related search spaces. Note that all examples use the waypoint grid search space, even though the principles used to create the game area are very good. The technical implementation of the waypoint network may vary.

3.1 Grid based game area

A grid based game area is a rectangular area consisting of squares that are all squares of the same size and are perpendicular to each other so that each square is attached to its sides in up to four other squares and up to eight in the other corners. In addition, each square divides at least two pages with other tiles and an angle with at least three tiles. Chessboard is a classic grid based game area. The screen is the smallest part used to create such a game area, and all information in the game area is placed to the exact resolution of each screen as needed. It is conceivable that the squares are the smallest and only available resolution for the game area coordinate system to store data. The coordinate system has X and Y axes that are perpendicular to each other. The X coordinate is horizontal and the values increase as you move to the right and the Y coordinate is vertical and its values increase as you go down. Origon can be located anywhere, but for practical reasons, it is usually located in the upper left corner of the area or in the middle of the area. Figure 4 illustrates a screen-based game area.

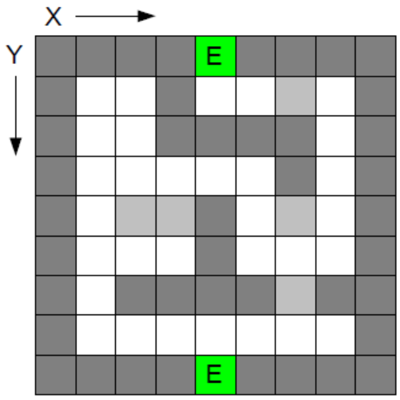


Figure 4: The picture shows an example of a screen-based game area. Dark gray panes are impassable panes, light gray panes are difficult, light panes are light. E-screens allow you to move to other game areas.

The grid based game area is traditionally a much used type of game area, especially when the game is simple in rules or when the game device is particularly limited in memory or performance. The game area can easily be stored in a two-dimensional table where each cell corresponds to one square of the game area. Let's position the screen in a table so that their neighbors and the corresponding cells are the same. The table is now easy to navigate, and the neighbors of the screen can be easily found by editing the table's pointers. For example, the northmost neighbor of the box that is in the coordinate (x, y) is found in the coordinate (x, y - 1), the neighbor in the right (x + 1, y), etc.

In the grid-based game area, the search space is naturally formed by placing nodes of the waypoint grid in boxes. The arcs are formed between nodes whose corresponding boxes are adjacent to each other and are allowed to move between them. Nodes length can be defined as a geometric length weighted by a value that makes it difficult to move to the opposite screen. Figure 5 illustrates a search space formed from the game area of Figure 4.

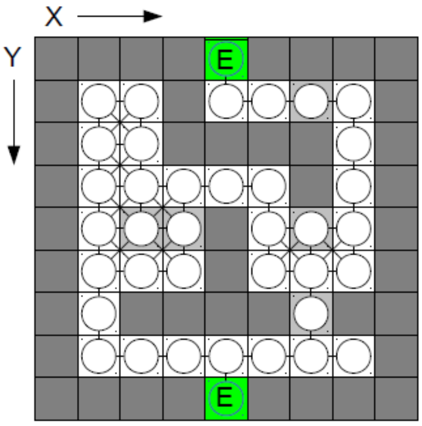


Figure 5: An illustration of an exploration space formed from a grid based game area. Due to lack of space, the lengths of the arcs are not marked. Arches that attach to nodes in gray areas are longer.

Instead of squares, the grid-based game area can also consist of other types of regular shapes[11]. It is enough that the tiles are uniform in shape and size and that they can be stacked side by side so that the game area is covered. One often used alternative to a square is the hexagon. A hexagonal game area deviating from a hexagonal game area, especially so that the nodes have at most six neighbors. Then each cell also shares a page with its neighbor.

3.2 Free-form play area

A free-form game area is a game area where the content of the game area is placed in the free-play area without any special restrictions. Unlike the grid-based game area, where content was always placed in screen resolution, the free-form game area allows content to be freely positioned anywhere in the game area. The objects may be of very different shapes and sizes. This type of game board is very common nowadays, especially in games that aim to produce high quality, varied and rich content and often seek to create an image of a realistic game world. Many modern gaming machines can also handle a wide range of gaming areas. The increase in the computing power of the gaming devices and the increase in memory have also allowed for more versatile gaming areas. Figure 6 illustrates a simplified free-form game area.

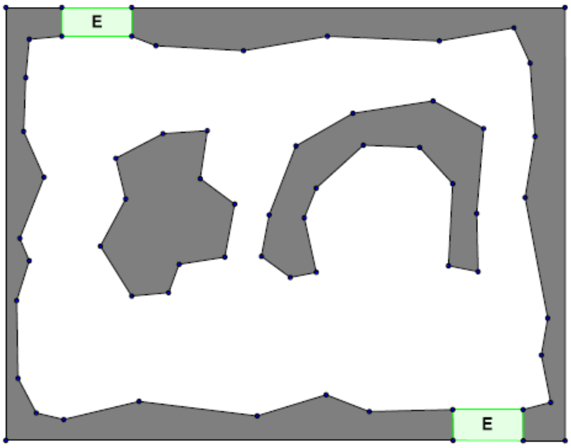


Figure 6: In the free play area, obstacles, partial obstacles, objects, and escape paths can take you anywhere in the play area. In addition, they can be very informal.

Figure 7 illustrates a waypoint network that is manually formed in the game area of Figure 6

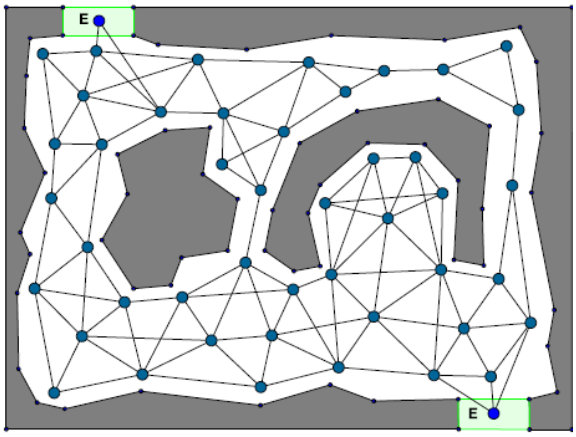


Figure 7: An example of a waypoint network formed in the game area of Figure 6

Problems also arise with the simultaneous movement of game characters and the avoidance of objects in the waypoint network. The characters may move at the same time along the same or adjacent curves so that the game characters collide with each other. Some games allow characters to move through each other so collisions are not a problem. In some games such pass-through is undesirable and must be prevented by available means, such as dynamically moving the game characters so that they do not rotate extremely inconvenient because it does not provide enough information about the area that is suitable for movement. Since only nodes and arcs are known, deviation from them can lead to the movement of the game character out of the area suitable for movement. This can cause all sorts of problems, which at the very least are visual but at the very least break the game rules ruining the gaming experience. There is not a very good solution to problems like this if you have a waypoint network.

4.Navigation Network Model

The navigation network model is a search space that seeks to fill in gaps in the waypoint network. The navigation network model expands the waypoint network by changing the way the network is displayed, adding useful information about route discovery space. In addition to the allowed transitions, the navigation network model defines the areas that are allowed to move.

The navigation network model can be created manually using any tool that can be used to create and edit a 3D model. Alternative navigation network model can be generated automatically [Ham08]. Route search in navigation system model is also usually done using, for example, A * method, but also alternative methods are[12]. Since the A * and Djikstran methods require a waypoint network, the navigation network model must be one like this. There are different methods for setting up a waypoint network and finding the final route.

4.1 The structure of the navigation network model

The navigation network model consists of triangles. Each triangle is defined by three points located in either two-dimensional or three-dimensional space, depending on the game area. Each triangle is known to define a surface. The surface of the navigation network model port is interpreted as the area in which the game character is allowed to move. Figure 8 shows a batch navigation network model with the triangles dividing the portal connected to their priorities. The resulting broken line indicates the permissible transitions between the triangles.

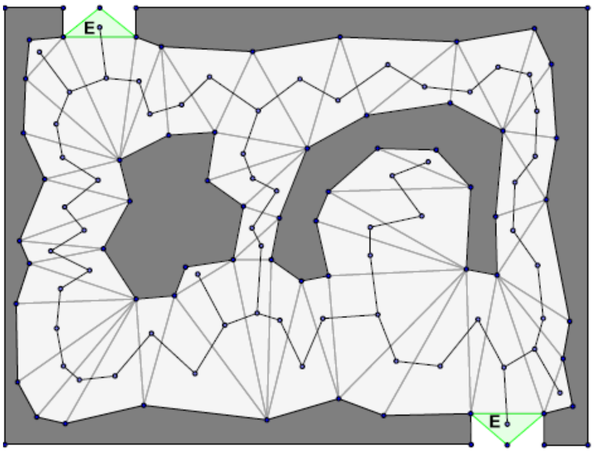


Figure 8: One navigation network model. The curves of the broken lines are connected to the priorities of the triangles.

5. Route search in a navigation network model using a channel

Because game characters are allowed to be located only in the area designated by the navigation network model, every valid path of the game character must also be located there in its entirety. So if we take the two points of the navigation network model, and assume that there is a route that connects these points, then this route must travel only along the triangle surfaces of the navigation network model. The triangles through which the route passes form a subset of triangles in the navigation network model. This triangle subset is called a channel. When looking for a path for a game character, we can first search for a channel and use it to form the final path for the game character.

5.1 Channel

The channel contains all the triangles in the navigation network model that the game character must pass in order to reach the departure node goal node. The channel does not include triangles in the other navigation network model. When defining the path of a game character, the channel is treated as a separate entity, a kind of sub-navigation network model. Transitions between non-channel triangles are not allowed. Figure 9 illustrates a channel

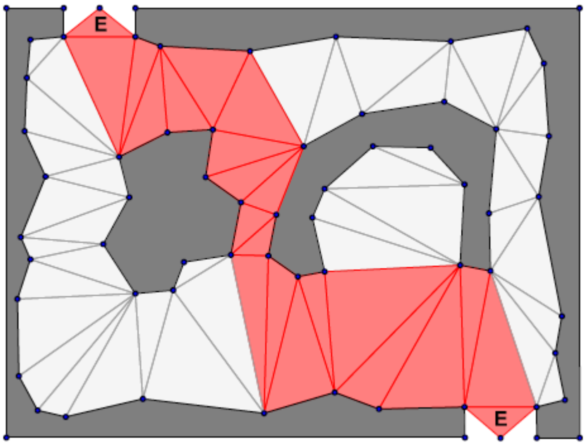


Figure 9: Channel example.

Figure 10 clarifies the data structures used by the funnel algorithm. In the picture, point PL and vertex form the path, vertex , V_1 and V_2 form the left wall and vertex, O_1 and O_2 form the left wall. Between points V_2 and O_2 is a portal that has just been processed.

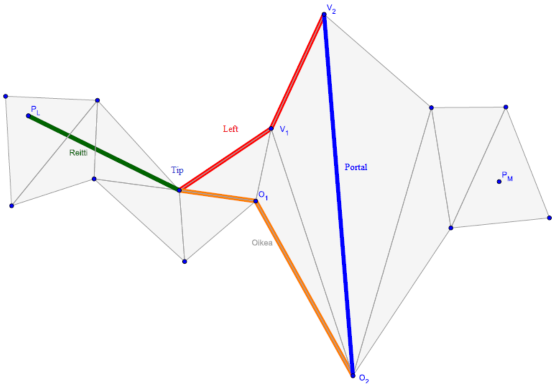


Figure 10: Funnel algorithm status in a channel.

6.0 Creating a Channel

The channel can be easily formed as follows. A waypoint network corresponding to the vessel navigation network model is formed so that a node for each triangle of the navigation network model is added to the waypoint network. For example, the node can be positioned in the center of the triangle. An arc is then inserted between the nodes in the waypoint network if and only if there is a portal between the triangles in the navigation network model corresponding to the nodes. Next, we look for the triangles in which the start and end points are located, defining the starting end node corresponding to these triangles, and using the A * method, we divide the shortest route between these nodes. Finally, we go through the nodes of the route point network found and extract the triangles of the navigation network model corresponding to these nodes. The extracted triangles form a channel. Figure 11 illustrates this situation. There are three in the picture dashed lines, a, b, c, and d. The lengths of the lines are marked next to the break lines in the image. The break line c is the actual shortest route between the start and finish points. The broken line represents the route that would have formed the channel that the actual shortest route would follow.

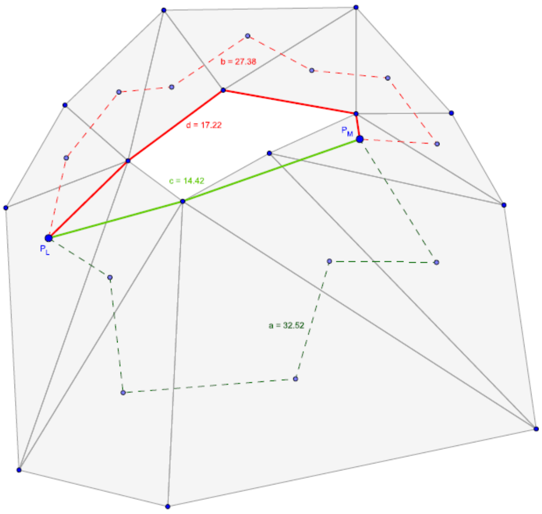


Figure 11 : Problematic navigation network model

6.2 Route search in a navigation network model using a visibility map

Route search in the navigation network model can be performed using the visibility map[13]. A visibility map is generally a grid connected to a polygon, showing which angles of the composition see each other.

6.3 Visibility map in the navigation network model

This can be accomplished by using a visibility map as the basis for a waypoint grid. The nodes of the network are connected by an arc key if the triangle points of the navigation network model corresponding to the nodes can see each other. In a navigation network model, two points can only see each other if the points can be joined by a segment so that the segment does not intersect the edges of the navigation network model.

Naturally, the angles of each triangle see each other, so the arcs corresponding to each side of the triangle belong to the waypoint grid. Adding arcs after all with a naive algorithm is trivial but slow. The naive algorithm examines the segment formed by the intersection of each point pair on the edge of the navigation network model. If not, we add the line corresponding to the line to the waypoint grid. The figure illustrates the existing waypoint network. The waypoint network formed is a visibilitygraph (vgraph). Visibility maps are used, for example, in robot navigation, and neon is a commonly used method for finding game paths. There are also more effective methods for creating a visibility map[14].

If we assume that the navigation network model does not change during the game, we can calculate the visibility map in advance and connect it to the navigation network model before starting the game.

In this case, the algorithm used to form the visibility map does not necessarily have to be efficient. However, before the actual route search, nodes corresponding to the start and finish points must be placed in a network of waypoints formed from the visibility map. As a result, the nodes formed are added to the waypoint network and then added.

If we assume that the navigation network model does not change during the game, we can calculate the visibility map in advance and connect it to the navigation network model before starting the game.

In this case, the algorithm used to form the visibility map does not necessarily have to be efficient. However, before the actual route search, the nodes corresponding to the start and finish points must be placed in a network of waypoints formed from the visibility map. As a result, the nodes formed are added to the waypoint network and then we add the arcs from these nodes to the other nodes in the network only if the nodes see each other does not match the new paint point. Figure 12 is a simple navigation network model with PL as the starting point and PM as the destination point, a visibility map formed from a mixed navigation network model. The figure shows a thick broken line of the shortest route between the start and finish points when the visibility map is used as a waypoint grid.

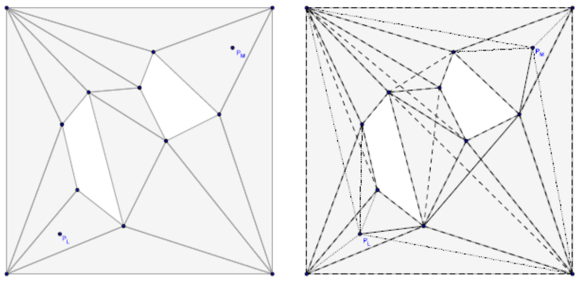


Figure 12: A simple navigation network model on the right and a visibility map formed here on the left. The arcs leaving the PL and PM nodes are temporary. The thick break line marks the shortest route between these nodes.

7. Route search enhancement methods

The Dijkstra method and its variant A * are not very effective if the route point network is rich in nodes and arcs. If the route search is too heavy and the waypoint network cannot be simplified without disrupting the game, the opportunity remains to improve the search method. It is good to understand that enhancement methods complicate search methods, possibly making them obscure and difficult to maintain. Many methods also require heavy work search space preprocessing, which can increase tool chain runtime or load time. Implementing enhancement methods and preprocessing can be very demanding and time A*-consuming, so the benefits of implementing them should be carefully considered. The methods are based on adding some sort of path search accelerator information to the search space and modifying the search method to take advantage of this added knowledge. Such information includes, for example, the creation of hierarchies in the route point network, and the insertions and additions to the network. Many methods can be further enhanced, for example, by performing bidirectional search.

The A * routing method can also be performed bidirectionally. Since the A * method directs the search by estimating the remaining distance by the function h (x), the bidirectional transformation requires two different functions hM (x) and hL (x), of which functionahM (x) is used to advance from the departure node to hL (x). If the function paths give the same estimate at the same node, there is no certainty that the path optimality will encounter when searching. The functions hM (x) and hL (x) can be defined so that they take into account both directions of the search and ensure the optimal route of the encounter[15].

7.1 Exploiting hierarchies

Different hierarchies can be found in waypoint networks, which can be utilized to enhance route search. For example, it is possible to define upper-level master point networks in a waypoint network, which hide the nodes in the lower level waypoint network and act as some sort of shortcut. The search can then be performed using these shortcuts, and the search then moves to lower-level nodes only as needed. A hierarchy is formed by dividing a network of waypoints into areas that are bounded by nodes. At the upper level of the network, these edge nodes are interconnected by arcs to form an upper level network. The network-level nodes are connected to the adjacent-level nodes by special arcs, thus allowing the search to proceed from one level to another.

7.2 Two-way route search

There is a variant of the Dijkstra route search method, where the search is performed simultaneously from the source node to the goal node and from the wave node to the departure node. Bending until both searches find the same node, so the search can be stopped and a route can be created using the partial paths found by both searches. The idea is that when searching in the direction side by side. In addition, it can be combined with many other route search enhancement methods and is also an integral part of many enhancements.

Advanced search methods

The A * method is a directed search method because it directs the search towards the target, thus avoiding the handling of irrelevant nodes. There are other methods by which the search can be directed and thus enhanced. One way to direct your search is to add some kind of signposts to the waypoint network. These signposts provide information on which nodes can be accessed through a node or arc, or through which node can be accessed. Road signs are formed as a pre-processing of a waypoint network and this can be time consuming. On the other hand, signposts need storage space, so they may not be suitable for all kinds of situations, especially when memory is limited.

The nodes processed by the Dijkstra search method can be pruned based on signposts. Each curve of the waypoint network stores a road sign indicating the nodes that can be accessed along the shortest path through the arc. In other words, each arc k stores a set of nodes L (k) whose shortest path starts at arc k. You can reduce memory consumption by dividing nodes into groups on a geometric basis and storing at the reference group level instead of saving each node separately. The number of nodes processed with the help of the method can be reduced to one tenth.

The use of signposts can be further enhanced by combining them with the idea of highway networks, for example. A method called SHARC shows that a combination of these two can produce an efficient search method without increasing the memory requirements too high.

The A * method can be enhanced by a method called ALT (Landmarksand the Triangle Inequality). The method is based on landmarks that are in a special position in the search space nodes. As route exploration progresses, observing the triangular inequalities formed between landmarks can be eliminated by observing the disadvantageous branches of the waypoint network. The ALT can be further combined with example highway hierarchies.

8. Conclusion

Many games have game characters with different goals. Many times, in order to reach these price targets, the game character must move from their current location to another location in the game area. The rules of the game determine the conditions of movement, and game characters must abide by these conditions. In addition, in many games, it is desirable that the game characters move naturally and reasonably in terms of the game character. If the rules of the game require the character to avoid obstacles while moving, the character must find a route by which the character can move to their destination by obstacles.

The area suitable for the movement of the game characters defines the search space, and the router search methods search for routes from the search space. The rules of the game and the gaming area together determine which search space is most appropriate for the game. The Waypoint Network is a common search space because it is applicable to a wide variety of gaming areas and because multi-point search methods eventually require the Waypoint Network to be its search space. For a game area consisting of squares, where the game characters are always located in one screen, the waypoint grid is an excellent choice for search space. Such games include many board games and strategy games. However, many modern games require very informal gameplay areas, where every allowed location of the game characters cannot be predefined without impairing game play. In such games, it is often better to focus on defining the areas within which the game characters can move freely. The navigation network model is such a search space. Route search can be done further in a route point network, which can be formed in many different ways prior to the search of a navigation network model. For example, a channel or a visibility map can be used in the search. In channel search, the search is performed in two steps, the first of which is to find a channel consisting of triangle centers and a side point network formed by triangles in the navigation network.

In a visibility map search, the waypoint grid is formed by taking the angles of the triangles in the navigation network model, as well as the start and end points of the search, and combining all the points that can be seen with the arcs.

The navigation network model also provides essential additional information from the game area to the exploration space, which can be used to make route searching more flexible without breaking the rules of the game.

Taking advantage of this additional information, the game characters can, for example, dodge each other's search for air paths. The game area can also be made into narrow sections that only suitably small game characters can pass through.

Route search can take too long if the waypoint network is rich in nodes and if the game device is not efficient enough to perform the search. In some games, slowing down routing may be so undesirable that you need to intensify routing. The trivial solution is to reduce the nodes in the waypoint network, but this is not always the case possible. In this case, you will need to find a variety of router enhancement methods that can significantly speed up your search. On the other hand, multipath enhancement methods store additional information in a waypoint network, increasing its total memory and rendering search methods more complex. Game devices and games are diverse. Gaming devices determine the resources available, within which the game can be played, and on the other hand, the game rules set the requirements that are required for the game to run smoothly. Some gaming devices, such as new game consoles and PCs, offer a large amount of memory and performance, while others, such as phones and pocket consoles, may offer more limited memory and performance. If a game requires route exploration, programmers should consider the different route exploration spaces and route exploration methods, taking into account the requirements of the game and the gaming device resources, and select the appropriate solutions for that game.

References

1 Snook, G., Simplified 3D Movement and Pathfinding Using Navigation Meshes. Teoksessa Game Programming Gems, Charles River Media, 2000, Page 288-297

2 Dijkstra, E. W., A Note on Two Problems in Connexion with Graphs. Numerische Mathematik, 1, 1 (1959), Page 269-271.

3 McGuire, M., Jenkins, O., Creating Games: Mechanics, Content, and Technology, A K Peters, 2008, Page 104-105.

4 Franklin, S., Graesser, A., Is it an agent, or just a program?: A taxonomy for autonomous agents, Intelligent Agents III. Agent Theories, Architectures, and Languages, Lecture Notes in Computer Science Volume 1193, 1997, Page 21-35

5 Tozour, P., Search space representations. Teoksessa AI Game Programming Wisdom 2, Charles River Media, 2003, Page 85-102

6 Dijkstra, E. W., A Note on Two Problems in Connexion with Graphs. Numerische Mathematik, 1, 1 (1959), Page 269-271

7 Stout, B., The Basics of A* for Path Planning. Teoksessa Game Programming Gems. Charles River Media, 2000, Page 254-262.

8 Russell, S. J., Norvig, P. Teoksessa Artificial Intelligence: A Modern Approach. Upper Saddle River, N.J.: Prentice Hall, 2003, Page 97–104.

9 van der Sterren, W., Tactical Path-Finding with A*. Teoksessa Game Programming Gems 3, Charles River Media, 2002, Page 294-306.

10 Dickheiser, M., Inexpensive Precomputed Pathfinding Using a Navigation Set Hierarchy. Teoksessa AI Game Programming Wisdom 2, Charles River Media, 2003, Page 103-113.

11 Yap, P., Grid-based Pathfinding. Lecture Notes in Artificial Intelligence, vol 2338 (2002), Page 44-55.

12 White, S., Christensen, C., A Fast Approach to Navigation Meshes. Teoksessa Game Programming Gems 3, Charles River Media, 2002, page 307-320.

13 Nguyet, T. T. N., Hoai, T. V., Thi, N. A., Some Advanced Techniques in Reducing Time for Path Planning Based on Visibility Graph. Third International Conference on Knowledge and Systems Engineering (KSE), 2011, page 190-194

14 Ghosh, S. K., Mount, D. M., An output sensitive algorithm for computing visibility graphs. SIAM Journal on Computing, 28th Annual Symposium on Foundations of Computer Science, 1987, page 11-19

15 Ikeda, T., et al. A fast algorithm for finding better routes by AI search techniques. Proc. Vehicle Navigation and Information Systems Conference, 1994, page 291 – 296