Implementation of Path Finding Algorithms in a 3-Dimentional Environment

Firas Abdullah Thweny Al-Saedi[1], Fadi Khalid Ibrahim[2]

[1,2] Computer Engineering Department, Al-Nahrain University, Baghdad, Iraq

***Abstract* — This paper discusses the use of the path finding algorithms in a 3-Dimensional (3D) military training environment. It describes how to represent the nodes in a 3D environment. Two algorithms are used: the Waypoint Navigation and the A* path finding algorithm. A comparison between the two path finding algorithms is made to evaluate their performance. Also, a solution to the problem of finding the first node to go to by the object is solved.**

***Keywords* — 3D, Path finding, A*, Introduction, Waypoint navigation.**

I. INTRODUCTION

Before moving into the subject the reader must know what is Virtual Reality (VR), VR is a computer-simulated environment, whether that environment is a simulation of the real world or an imaginary world. Most current VR environments are primarily visual experiences, displayed either on a computer screen or through special or stereoscopic displays, but some simulations include additional sensory information, such as sound through speakers or headphones. Some advanced, haptic systems now include tactile information, generally known as force feedback, in medical and gaming applications. Users can interact with a virtual environment or a Virtual Artifact (VA) either through the use of standard input devices such as a keyboard and mouse, or through multimodal devices such as a wired glove, the Polhemus boom arm, and omni-directional treadmill. The simulated environment can be similar to the real world, for example, simulations for pilot or combat training, or it can differ significantly from reality, as in VR games. In practice, it is currently very difficult to create a high-fidelity virtual reality experience, due largely to technical limitations on processing power, image resolution and communication bandwidth. However, those limitations are expected to eventually be overcome as processor, imaging and data communication technologies become more powerful and cost-effective over time [1].

In this paper and to be cost-effective, Microsoft Visual C# 2008 [2] along with the new XNA 3.0 [3][4][5] graphics technology released by Microsoft were used, actually, the graphics technology used is games-quality, this technology was used to generate a VR environment that is used individually or through network of two computers (this can be expanded easily). Also, the input device used is either the standard keyboard and mouse or using the new Nintendo Wii Remote (Wiimote) [6][7].

Another paper on A* path finding algorithm in a bird's eye perspective can be found in [8]. Also, another paper that uses the A* path finding algorithm and discuss its uses in games can be found in [9].

The soldiers controlled by the computer AI need to know the path in the simulation environment, for this reason, the path finding algorithms are used. The path finding algorithms related to the soldiers in the simulation are discussed in the following sections.

II. THE PATH FINDING PROBLEM

An example of path-finding algorithms is the algorithms that are used in network routers. Each router represents a node in the system. When the data enters the router, its destination address is checked, then using path finding algorithms, its route is determined and it is sent. In some circumstances, the whole path from source to destination can be calculated in one router. In the case of the 3D simulation, it is different in some aspects, the 3D environment must be planted with nodes and connections between those nodes must be made in a way that makes it possible for an object to move from one node to the other either directly or by passing through any number of nodes. It is better to use a few number of nodes but they must be sufficient relative to the environment size. By sufficient, it is meant that there must be a node in every corner or point of the 3D environment. All this is made to make the object able to reach any point in the environment.

A node in a 3D environment is represented by a bounding sphere. A bounding sphere is a data structure that has two properties, a center, which is represented by a (X,Y,Z) coordinate and a radius which represent the size of the sphere. The reason beyond using the bounding spheres as nodes is that they are used as indicators for where the object is in the moment. When the bounding box of the object intersects with a bounding sphere, it is known now where the object is and the path finding routine can be activated again to find the path to another node. Fig. 1 is a simple illustration of how to use nodes in a 3D environment (top view).

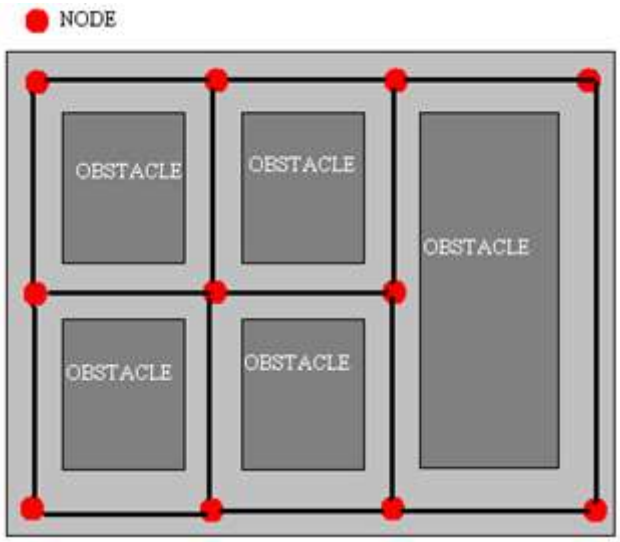

Fig. 1 Illustration of node distribution

As seen, the filled circles are the nodes (bounding spheres) with an appropriate radius. The radius is important because when the object moves in the environment, it may move in 'n' steps per frame. The more 'n' the more is the speed that the object appears to be moving. If the object is moving at a relatively high speed and passes over a node, the object's bounding box may not intersect with the node and in this case the path will not be calculated at that node and the object will continue moving in its direction which may lead it to an obstacle. For this reason, the bounding sphere that represents a node must have an appropriate radius, that is, it must be greater than the object's step. For example, if the object moves two units in a frame, the bounding sphere radius must be greater than two.

In the simulation, each soldier has the capability to find the way to any point in the environment through the use of the nodes that are distributed in the environment. There are steps that the soldier in the simulation follows to go to a node. These steps are demonstrated in Fig. 2.

The origin node and the destination node are found by finding the nearest node to the source and destination respectively. The reason beyond using nearest node is that it is impossible to cover the whole environment with nodes. This means, to get to any point, the path finding routine must find the nearest node to that point and then, from that point, the soldier can go directly to the desired point. In the military simulator, the desired point is always a soldier and its running most of the time. This will make it impossible to find the entire path from one node because the destination node is always changing due to the change in the destination point (destination solder's position). For this reason, the destination node is found and the path is calculated when the soldier reaches any node only to find the next node.

For the soldier to go to a point, it must turn by an angle that makes it face the point directly. Fig. 3 is a demonstration of this problem.

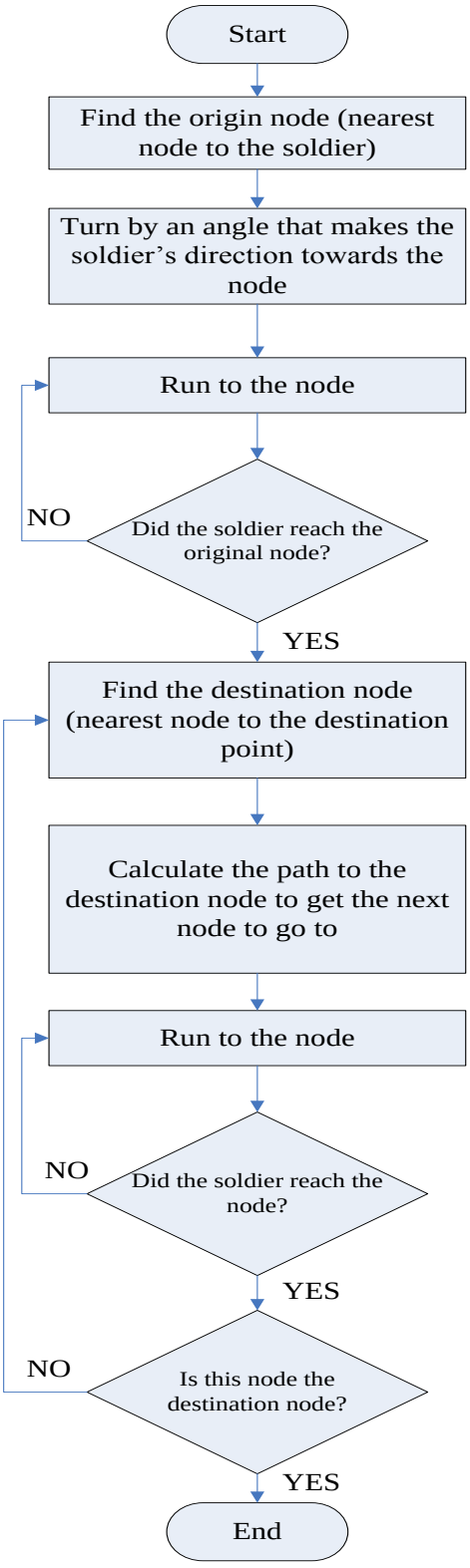

Fig. 2 Steps followed by the soldier to get to any point

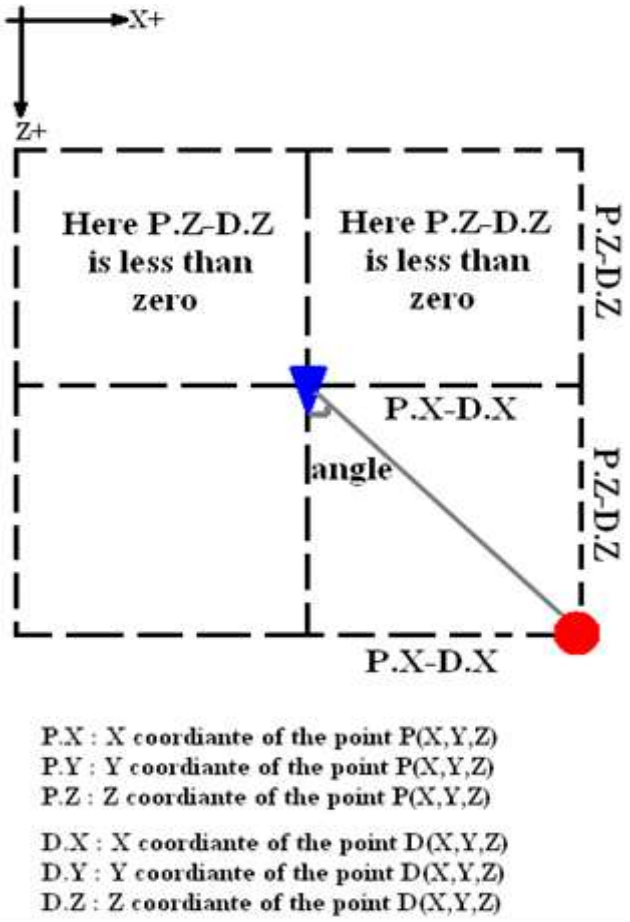

Fig. 3 Demonstration of soldier looking at a point

Using the soldier's position P(X,Y,Z) and the point to look at position D(X,Y,Z) which is the center of the bounding sphere, the P.Z-D.Z and P.X-D.X can be found and used to find the angle shown above by using the algorithm shown in Fig. 4.

To check if the soldier had reached the destination node, its collision with the bounding sphere that represents the node is checked. If there is a collision then the soldier is at the node and then the path finding routine should be activated again to find the next node to go to. In the next sections, the two path finding algorithms used are discussed also the difference between them is stated.

III. Path Finding Algorithms

There are a lot of algorithms used for path finding [10][11]. Each algorithm has its own uses. The two algorithms used are discussed, the first one is the waypoint navigation [10] and the second one is the A* (A-Star) search algorithm [10][11] to find the shortest path.

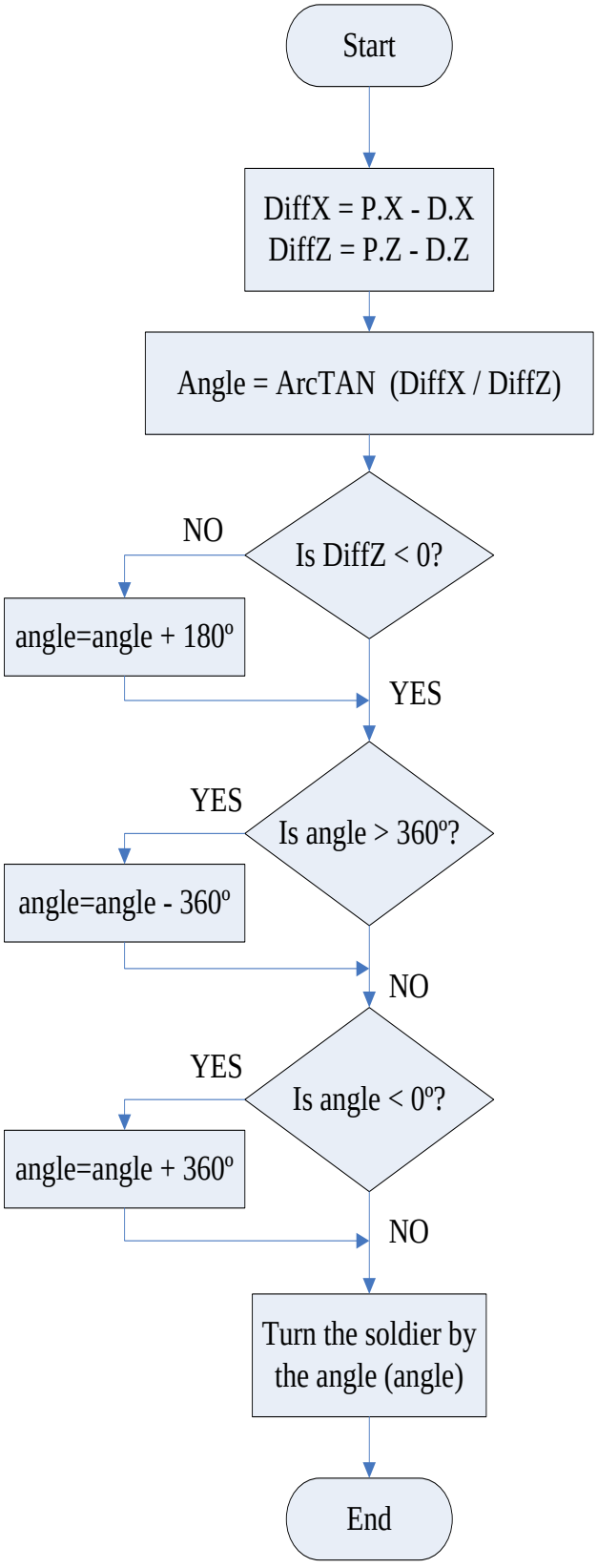

Fig. 4 Finding the angle that the soldier's angle that faces the point D(X,Y,Z)

1) Waypoint Navigation

Path-finding can be a very time-consuming. One way to reduce this problem is to pre-calculate paths whenever possible. Waypoint navigation reduces this problem by carefully placing nodes in the simulation environment and then using pre-calculated paths or inexpensive path-finding methods to move between each node. Fig. 5 demonstrates nodes distributed in a floor in a building.

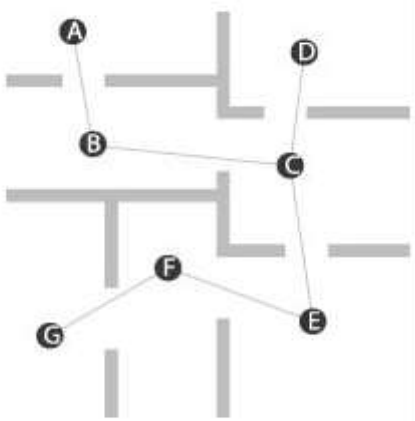

Fig. 5 Nodes distribution in a floor in a building

It should be noticed that there must be a line of sight between a node and its neighbor. In waypoint navigation, the path between a node to another must be calculated by the designer and store the path in a table. During the program execution, the path between any node and the other can be found easily by finding the next node to go to only. For example, Fig. 6 demonstrates the pre-calculated paths between nodes.

Start \ End	A	B	C	D	E	F	G
A	—	B	B	B	B	B	B
B	A	—	C	C	C	C	C
C	B	B	—	D	E	E	E
D	C	C	C	—	C	C	C
E	C	C	C	C	—	F	F
F	E	E	E	E	E	—	G
G	F	F	F	F	F	F	—

Fig. 6 Node table for the nodes in Fig.5

So, if the soldier is at node A and it needs to go to node E, the first node to go to in this case is node B. When the soldier reaches at node B, it will find the destination node and let it be node E, also it will find the next node to go to which is node C. and so on until it reaches to the destination node. In the next sub-section, the A* algorithm is demonstrated which is used to calculate the shortest path between any two nodes in a node-based system, so there is no need to pre-calculate the paths manually.

2) A Path-Finding*

Fortunately, the A* algorithm provides an effective solution to the problem of path-finding. What makes the A* algorithm so appealing is that it is guaranteed to find the best path between any starting point and any ending point, assuming, of course, that a path exists. Also, it's a relatively efficient algorithm, which adds to its appeal. The A* algorithm is efficient, but it still can consume quite a few CPU cycles, especially if it is needed to do simultaneous path-finding for a large number of objects.

To find the shortest path between two nodes, there should be a way to put a weight for each connection between any two nodes. In this case, and because of the geographical nature of the simulation, the distance between each node and each of its neighbors is used as a weight. So in the code, the mesh of the nodes is defined as a data structure, and in the initialization, the weight (distance) between each node and its neighbors is calculated. Then in run-time the A* path-finding routine can be executed which can be represented in Fig. 7.

The "lowest cost" term mentioned above means the cost of the current node to the starting node which is the sum of the weights of the connections between the nodes from the starting node to the current node. Each node's parent node must be tracked in the A* path-finding routine because when the destination node is found, the routine will know the path from the starting node to the destination node. Although the whole path is calculated but the second node found in the search is used as the destination node and that is because the destination object is always moving and it is not expected that it would be in its original place when the path was calculated.

3) Performance Evaluation

Both path finding algorithms were used. The waypoint navigation as seen is good for small environments where there are a few number of nodes, also, it is perfect in systems that needs each bit of CPU cycles because the calculation needed in this algorithm is just accessing memory and getting the next node index which is a very fast and simple process and does not take a lot of CPU cycles. Also it is perfect when there are a lot of objects that need to know their paths in the same time. While the A* path finding algorithms seems more robust because it calculates the shortest path between any two nodes, it takes more CPU cycles than the waypoint navigation and this becomes worse when a lot of objects use the routine to calculate their own paths. In the simulation environment, the soldier needs to calculate the path only when it is at a node, so there is a small chance that all the soldiers in the simulation are at some node. For this reason, even when using a lot of soldiers, the performance will still be good because there is a very small chance that all the soldiers are using the path finding routine.

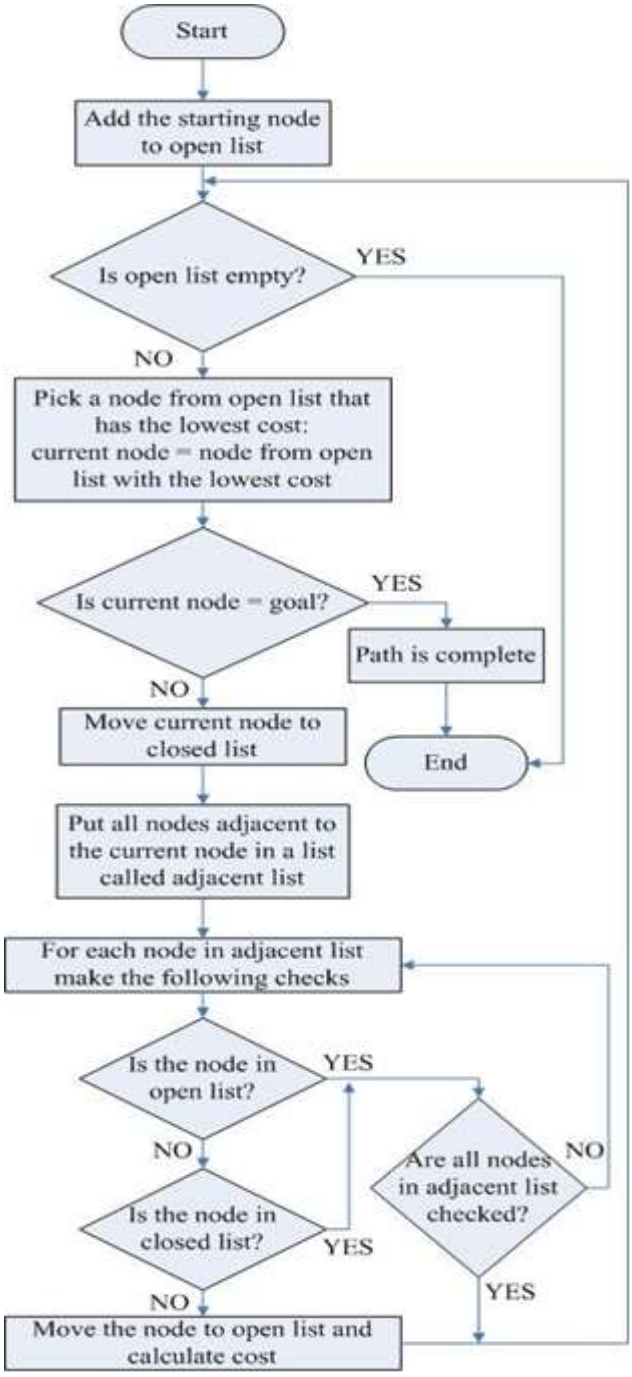

Fig. 7 A* path finding algorithm

IV. Proposed Solution to the Problem

As seen in Section II, the soldier turns to the nearest node to itself and then runs toward it to be able to locate itself and this is to calculate the path to the destination node. In some situations, the soldier happened to be outdoor, that is, it is not inside a building, and the nearest node is inside the building. In the traditional case, the soldier will turn to the nearest node and run toward it. This sometimes will result in a non realistic effect when there is an obstacle "the building's wall" between the soldier and the indoor node, the soldier will run through the wall, which is not realistic. This can be seen in Fig. 8.

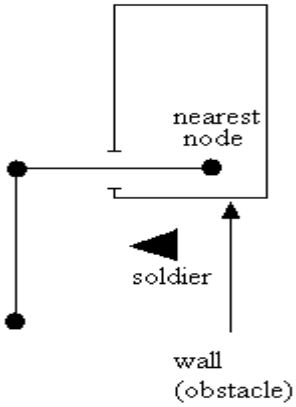

Fig. 8 The problem of nearest indoor nodes

For this reason, a solution was proposed, that is, the indoor nodes are marked with a flag by the designer when coding and in run-time each soldier is tracked to know that it is indoor or outdoor. If the soldier is outdoor, then when finding the nearest node to it, the indoor nodes are neglected and the nearest outdoor node is returned by the routine. This simply solves the problem.

V. Conclusions

The subjects discussed in this paper are parts of a project that simulates a military environment. The path finding system was discussed here and the two algorithms used where discussed. Also a comparison between them was made. For this project the A* path finding algorithms is better due to the size of the environment used which is a max of (86) nodes. Also, a solution was proposed to solve the problem of indoor and outdoor nodes. Fig. 9 shows the simulation environment in top view, the nodes are drawn to demonstrate the node distribution.

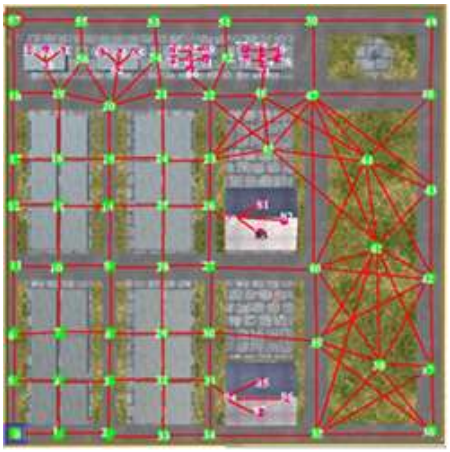

Fig. 9 The simulation environment (top view) with the distributed nodes

REFERENCES

[1] en.wikipedia.org/wiki/Virtual_reality.
[2] Rob Miles, "C# Developement", Department of Computer Sciences, University of HULL, October 2008.
[3] Aaron Reed, "Learning XNA 3.0", O'Reilly Media, 2009.
[4] Chad Carter. "Microsoft XNA Unleashed:Graphics and Game programming for XBOX360 and Windows", SAMS Publishing, 2008.
[5] Reimer Grootjans, "XNA 3.0 Game Programming Recipes: A Problem-Solution Approach", Apress, March 9, 2009.
[6] en.wikipedia.org/wiki/Wii.
[7] http://blogs.msdn.com/coding4fun/archive/ 2007/03/14/1879033.aspx.
[8] R.Anbuselvi and R.S.Bhuvaneswaran, "Simulation of Path Finding Algorithm – a Bird's Eye Perspective", 2009.
[9] Hui, Y.C. Prakash, E.C. Chaudhari, N.S, "Game AI: artificial intelligence for 3D path finding", 2005.
[10] David M Bourg, Glenn Seemann, "AI for Game Developers", O'Reilly Media, July 2004.
[11] Ian Millington, "Artificial Intelligence for Games", Elsevier Inc., 2006.