

Using Nodes to Develop Strategies For Combat with Multiple Enemies

Lars Lidén

Valve Software
520 Kirkland Way #201
Kirkland, WA 98033
lars@valvesoftware.com

Abstract

Nodes (or waypoints) are commonly used in computer games for the navigation of computer controlled characters (also known as non-player characters or NPCs). The following paper demonstrates how these nodes can also be used to develop combat strategies for NPCs that engage in combat with multiple enemies. By storing data about node relationships in a bit string class, tactical information about locations in the environment can efficiently be calculated and exploited by NPCs.

Introduction

As behavior of computer controlled characters (or NPCs) in computer games becomes more sophisticated, the need for efficient algorithms for determining behavior becomes critical. Determining intelligent combat strategies for an NPC with multiple enemies requires that the NPC be aware of its environment and the strategic value of various locations within the environment for any given configuration of its enemies.

In many 3D computer games, it is common practice to place nodes (or waypoints) in the world that NPCs use to navigate (Lidén 2000). Nodes are usually placed in key locations in an environment by a level designer who is familiar with the behavior of the NPCs and their ability to navigate. Connections between nodes are calculated automatically in a pre-processing step or are specified by a level designer. NPCs then use algorithms such as A* to find paths through the node graph.

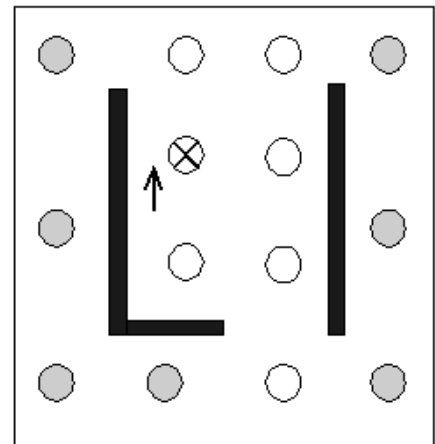
The following paper demonstrates that the same set of nodes that is used for navigation can also be used to efficiently calculate strategic combat information. In addition to calculating the connectivity between nodes one can pre-calculate the visibility between nodes. During run-time visibility information can provide a rough approximation of the danger of particular areas in a given map and locations from which an NPC can mount an intelligent attack.

Using Visibility For Evaluating Spatial Locations

For an NPC with single enemy, danger nodes are calculated by determining the enemy's nearest node. All nodes that are visible from the enemy's nearest node are considered to be dangerous. (See Figure 1).

Figure 1: Danger Nodes

An enemy and its facing direction are represented with an arrow. The enemy's nearest node contains an 'x'. Safe nodes are grey. Danger nodes are white.



As the enemy will not necessarily be on top of the danger node, only nearby, the labeling of dangerous areas will not be perfect. It serves as an initial gross approximation for the NPC who can fine tune its behavior as it approaches a node.

The key value in evaluating the dangerousness of different regions in this manner is that it is efficient. As visibility between nodes is pre-calculated the operation is cheap. Although one could use run-time line-of-sight checks with world geometry to determine the same information, when multiple NPCs with multiple enemies are involved geometry tests become too expensive.

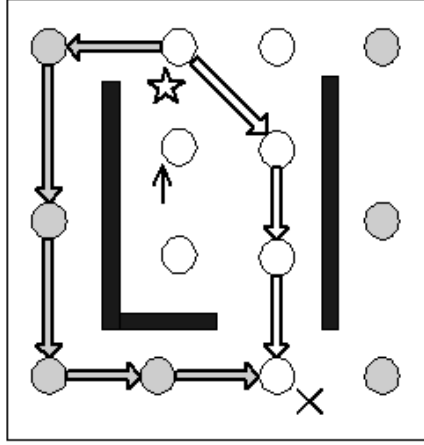
Safe Pathing

Details about the safety of locations in the environment can also be used to find safe paths for NPCs to traverse the environment. Most path finding algorithms use a cost function associated with the distance between nodes along the path. By adding a penalty to transitions whose

destination is an unsafe node, one can bias the path finding algorithm to find safe paths for an NPC. (See Figure 2).

**Figure 2:
Safe Paths**

An NPC at the position marked by a star will choose the longer safe path (shaded arrows) over the shorter dangerous path (clear arrows) to the target position (x) if a cost for danger is added to the path finding algorithm.



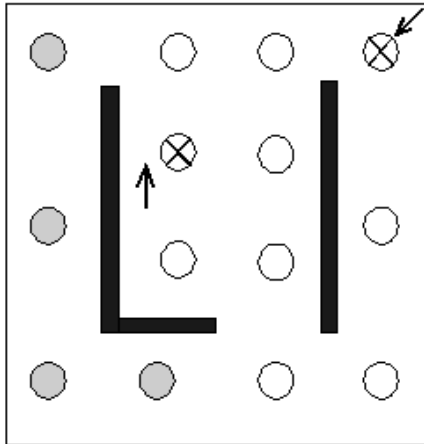
BitStrings

When an NPC has to contend with multiple enemies it becomes critical that an efficient data structure is used to store node visibility data. An effective and economical method for calculating safe locations for multiple enemies is to create a *bitstring* class that consists of a string of bits of arbitrary length along with operators for Boolean operations such as *<and>*, *<or>* and *<not>*.

For a given network of nodes, visibility is represented by a bitstring whose length is equal to the total number of nodes in the network. Each node has one bitstring whose values represent the visibility of all other nodes in the network. Calculating safe locations for a NPC with multiple enemies then consists of *<and>*-ing the visibility bitstrings for the each of enemy's nearest nodes. The resulting bitstring is a map of the safe locations for the NPC from all its enemies (See Figure 3).

**Figure 3:
Multiple
Enemies**

Enemies and their facing directions are represented by arrows. Their nearest nodes contain 'x's. Safe nodes are grey. Danger nodes are white.



Node Based Combat Strategies

An NPC lives on a network N_n consisting of n nodes. Network visibility is represented by a set of pre-calculated bitstrings, A_n . Network connectivity is represented by a set of pre-calculated bitstrings, ψ_n .

The NPC also has a set of k enemies E_k . If $\beta(E_i, t)$ is a function that returns the nearest node for enemy i at time t than the visibility of an enemy, i at time t is given by the bitstring:

$$1. \quad V_{it} = \Lambda_{\beta(E_i, t)}$$

For notational simplification, the time component will not be included in equations as it is understood that formulas apply to a specific time.

Fleeing and Fighting

NPCs can use information about which nodes are dangerous to find locations to flee (safe nodes) or locations from which they have a line of sight to shoot at an enemy (danger nodes).

The set of danger nodes for an NPC is the bitstring:

$$2. \quad \theta = \bigvee_{j=0}^{j=k} V_j$$

The set of all safe nodes its inverse, $\bar{\theta}$

Flanking

One interesting combat behavior is that of flanking. Flanking consists of finding a location behind a particular enemy, E_a from which it can be attacked. Additionally the flanking position should not be visible to any of the other NPC's enemies. Otherwise while the NPC is attacking E_a from its flanking position it becomes a sitting duck to its other enemies.

The initial candidates for flank nodes is the set of visible nodes from the selected enemy's nearest node, or:

$$3. \quad F_a = \Lambda_{\beta(E_a)}$$

A new bitstring, F'_a is created by eliminating those nodes from F_a that E_a is facing. This is done using a simple dot product operation between E_a 's facing direction and the vector formed between the each node and E_a .

Next, nodes that are potentially visible to all other enemies, D , need to be eliminated from the set of possible flank positions. D is given by:

$$4. \quad D = \bigvee_{j=0}^{j=k} V_j, j \neq a$$

Its inverse, \overline{D} , is the set of all nodes that are safe from all other enemies.

The set of legal flanking nodes, F''_a is given by the intersection of the set of potential flanking nodes and the set of nodes that are safe from all other enemies:

$$5. \quad F''_a = F'_a \cap \overline{D}$$

If there is more than one legal flanking node in F''_a , the best flank node still needs to be determined. This can be accomplished in several ways. The simplest method is to pick the closest flanking node. Alternatively, one can choose the flanking node that can be reached using the safest path. (See Figure 4.)

Flanking Multiple Enemies

A similar set of computations can be used to find a flanking position for multiple enemies. In this case the initial set of potential flanking nodes is given by:

$$6. \quad F_\xi = \bigcap_{j=0}^{j=k} \Lambda_{B(E_j)}, E_j \in \xi$$

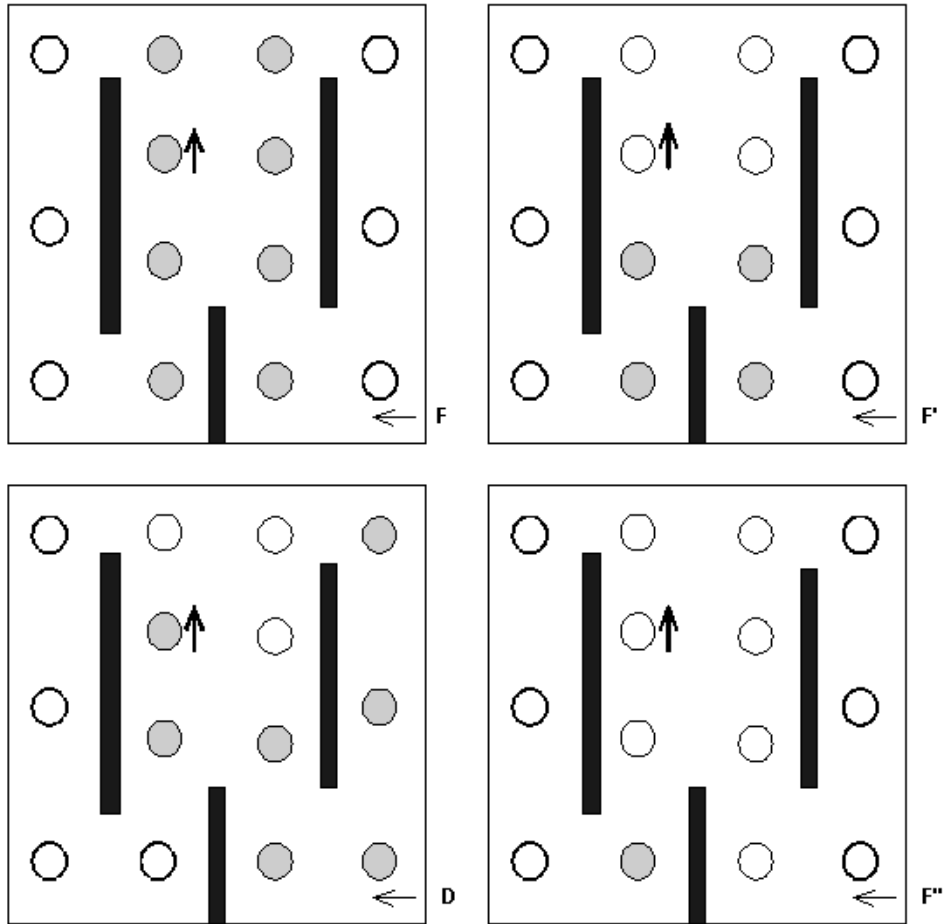
Where ξ is the set of enemies being flanked. F_ξ is created by eliminating those nodes from F_ξ , that any $E_j \in \xi$ is facing. The set of danger nodes is then given by:

**Figure 4:
Flanking**

An NPC finding a flanking position for the enemy marked with the bold arrow. The non-bold arrow represents another enemy.

F Nodes visible to enemy
F' Nodes flanking enemy
D Danger nodes
F'' Final set of flank nodes

The final flanking node has a line of site to the chosen enemy that is behind the enemy and not in the line of sight of the other enemy.



$$7. \quad D_{\xi} = \prod_{j=0}^{j=k} V_j, E_j \notin \xi$$

As before, the set of legal flanking nodes, F''_a for ξ is given by the intersection of the set of potential flanking nodes and the set of nodes that are safe from all other enemies:

$$8. \quad F''_{\xi} = F'_{\xi} \cap \bar{D}_{\xi}$$

Sniping

As previously discussed, establishing a line-of-sight for a NPC to shoot an enemy can be accomplished by finding the set of nodes that are visible to that enemy. However, a more sophisticated NPC would find a position from which to shoot that is both not in the line of sight of other enemies and near a safe place to run to when it needs to reload.

Such sniping, is another behavior that can be quickly and efficiently generated from node visibility information.

As before, the potential set of legal sniping nodes starts with the set of nodes that are visible to the selected enemy, E_a :

$$9. \quad S_a = \Lambda_{B(E_a)}$$

Nodes that are visible to other enemies (equation 4), are eliminated from the set, resulting in a subset of potential sniping nodes, S''_a , that are safe from other enemies.

$$10. \quad S''_a = V'_a \cap \bar{D}$$

To find a sniping location with a safe retreat, potential sniping nodes that do not have safe neighbors need to be eliminated. The set of all dangerous nodes is given by:

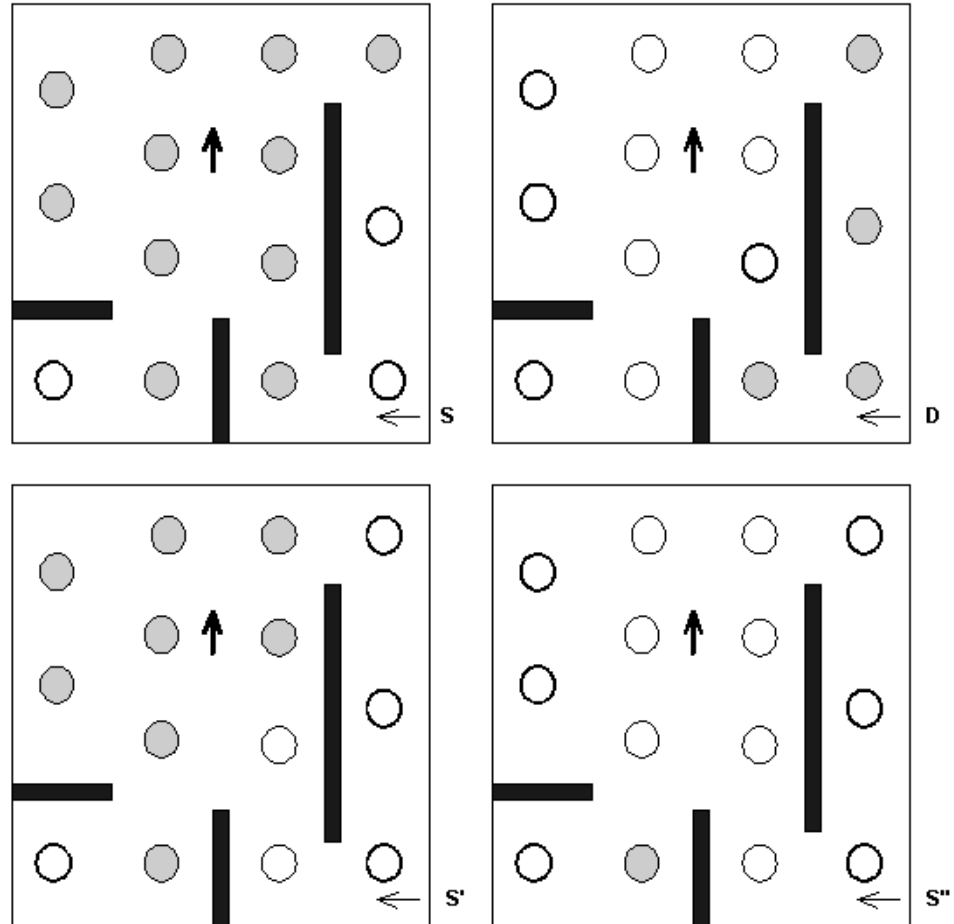
$$11. \quad H = \prod_{j=0}^{j=k} V_j$$

**Figure 5:
Sniping**

An NPC finding a sniping position for the enemy marked with the bold arrow. The non-bolded arrow represents another enemy.

S Nodes visible to enemy
D Danger nodes
S' Safe sniping node
S'' Final set of snipe nodes

The final sniping node has line of sight to the chosen enemy that is safe from the other enemies and has a safe retreat location nearby.



It's inverse, \bar{H} is the set of all safe nodes.

Network connectivity is represented by a set of pre-calculated bitstrings, ψ_n . Therefore, nodes in S''_a should be eliminated if:

$$12. \quad \psi_j \mathbf{I} \bar{H} = 0$$

The remaining set of nodes represents those that have a line of sight to the enemy, are not in the line of sight of any other enemies and have a nearby safe place where to retreat for reloading or if under attack. (See Figure 5.)

Conclusion

In many 3D computer games nodes are used to help NPCs navigate the environment. This paper discussed a few uses of information about node visibility and how it can be applied to combat strategies of NPCs. Particularly it was shown how nodes can be used to find places to flee, places to attack and how to derive more sophisticated behaviors such as flanking and sniping from node visibility.

There are several reasons that one might want to use node information in this manner. First calculating strategies using node visibility information is efficient even when large numbers of NPCs are involved and the representation of node visibility is compact. Secondly as nodes are already being used for navigation there is no need to place additional information into the world for strategic planning.

References

Lidén, L. 2000. The Integration of Autonomous and Scripted Behavior through Task Management. In *Artificial Intelligence and Interactive Entertainment: Papers from the 2000 AAAI Spring Symposium*, Technical Report SS-00-02, 51-55.