

Introduction

My project uses the First Person Shooter video game, Unreal Tournament 2004, and an accompanying bot-programming framework, Pogamut. Pogamut allows for scripting the bots in the game, running as many as we want on a single server / game map, including human players as well. The platform also contains extensive debugging and logging services.

A* is used for implementing path-finding around the map for the bot, assuming his medium and long term intelligence set a goal that is not immediately reachable. Still, under the current system, the bot “naively” runs between the given way-points, following the shortest path.

My proposal is to modify this system, so that A* does not only take under consideration physical space (i.e. shortest path), but such variables as cover, fire zones, bonuses, etc. It does this by updating the weighted graph of way-points to reflect not only space, but tactical viability in general. The project itself will only effect the low level implementation of A*. In other words, **any** bot programmed can choose to use this modification, without it affecting the other parts of its medium or high level intelligence. In addition to the modularity of the code, it is designed to be lightweight, with the heavier calculations (such as map analysis) being carried out offline, while the real-time processing is designed to be no heavier than the A* search already being carried out. In sum, the modification is invisible both in terms of code complexity (it is on a lower level of abstraction from the mid and high level coding) and run-time complexity – resulting in considerably smarter behavior at virtually no cost.

Theory

In A*, the next vertex chosen for expansion is the one that minimizes the evaluation function:

$$f(v) = g(s \rightarrow v) + h(v \rightarrow r)$$

Where s is the starting vertex and r is our goal.

g is a simple function which sums the known weights between the vertices. When used for “regular” A* path-planning, the weights equate to the known distance between the nodes. It is known because each node stores its neighbors and their weights. h is the heuristic, our function’s “best guess” to the rest of the distance from the vertex to the goal. In “regular” A* path-planning, a simple Cartesian distance function is used¹. It is important to note that under such a scheme, if v_1, v_2 are neighboring vertices, then the following is true:

$$g(v_1 \rightarrow v_2) = h(v_1 \rightarrow v_2)$$

It is important to note that when the vertices are not neighbors, g and h are not identical. However, this shows they share a common ground - both are based on a measurement function m , which in regular A* path-planning is simply the Cartesian distance².

My proposal modifies this m , to consider not only physical distance, but real-time tactical considerations such as cover & visibility, proximity to enemies, desirable items, and so on.

¹ In addition to being a fairly good heuristic, especially in open spaces, Cartesian or “as the crow flies” measurement fulfills the most important criteria for A* heuristics: underestimation / monotony. The fact that A* is both admissible and optimal relies on this fact. The criteria: Given vertex v and goal r , let $h^*(v \rightarrow r)$ denote the exact cost of the shortest path from v to r . Thus our chosen heuristic $h(v \rightarrow r)$ must fulfill the following: $\forall v \in V : h(v \rightarrow r) \leq h^*(v \rightarrow r)$

² $m = \|\text{position}(v_2) - \text{position}(v_1)\|$

A fairly major difference between this system and the current one is that the former cannot be calculated offline³. However, some parts of m can be calculated offline, such as cover and field of view. These parts can be summed together in a linear fashion, with a set of weights w , like so:

$$m(v_1 \rightarrow v_2) = \sum_{i=0}^n [w_i * m_i(v_1 \rightarrow v_2)]$$

where m_i are the collections of the various tactical functions. It should be noted that the weights themselves, w_i , might actually be a short function which returns a weight according to the tactical situation. For example, the m_i function for finding more health packets, should receive a heavier weight if the bot's health is low. That is, arguments passed to w_i , should relate to the bot's state. To recapitulate, m_i is influenced by the map and environment and w_i by the bot's state.

In addition to the major change in the A* module itself, other changes regarding 2 minor systems are necessary:

- 1) **Map and real-time analysis in order to grade waypoints** - Both within the bot programming framework and without, one can analyze the geometry of the level for the purpose of ascertaining good cover points and so on (Mainly raycasting from the point to others).
- 2) **Changes to the waypoint data structure** - Change the waypoint data structure to store the additional information, such as the offline calculations of cover and visibility.

Other areas of interest, open questions and issues to be addressed:

Path recalculation - seeing as how the path is determined by real-time considerations, if the path is long, the latter part will be less relevant, resulting in poor decision making. Recalculating the path, while more expensive computationally, would result in more accurate behavior. Thus there is a substantial challenge here - maintaining flexibility of the behavior while keeping our promise of computing costs no higher than static A*.

Forgoing optimality / admissibility - If we forgo the underestimation criteria for our heuristic, we find suboptimal paths, but the search executes much quicker⁴. This property can be very important, especially in conjunction with the earlier point regarding performance. Also, in certain situations, a "quick-and-dirty" solution might be preferable to a perfect one - especially if we recalculate soon after.

Testing - creating matches between medium-intelligence bots, half possessing the A* enhancement and the other half without it, thus examining how great an impact the enhancement results in. Thus could be taken even further to employ a form of meta-heuristic learning, by playing with the weights in response to the game data.

³ Regular or "static" path-planning uses the physical distance between nodes, which in the case of most video games, depends on the level or map and thus does not change in real time. This has led to some approaches where all distances and paths are calculated beforehand, and a lookup table is used in real-time.

⁴ As our heuristic returns increasingly higher values, the search becomes closer to best-first or beam search.

Affecting other bot behavior - if some waypoints are suitable for certain behaviors, for example, crouching, then the system could signal this to the lower layer of bot control.

References:

Smed, J. and Hakonen, H.

Algorithms and Networking for Computer Games

(2006) John Wiley & Sons, Ltd, Chichester, UK, pg. 96-113.

Amit's A* Pages

<http://theory.stanford.edu/~amitp/GameProgramming/>

Don't follow the shortest path!

May 26, 2008 at 4:49 pm · Christer Ericson

<http://realtimecollisiondetection.net/blog/?p=56>

Pogamut Framework for UT Bots

<https://artemis.ms.mff.cuni.cz/pogamut>