Artificial Intelligence Searching in Unknown Environments

Matthew Russell

Abstract

I implemented a probabilistic Artificial Intelligence and used it to create an opponent for a simplistic but randomized Hide and Go Seek 2D grid based game in JavaScript as an example of probabilistic reasoning in an unknown environment. The goal was to demonstrate probabilistic reasoning in stealth-based games where the omniscience of normal AI is a detriment to exciting gameplay. This should be useful for realistic stealth games and games like Hide and go Seek, where the programmer may not want to program in where to check manually, or where the level changes over time or wants to have a unique map each time. Hopefully the AI will also be a more realistically human opponent, and be harder to trick or exploit during gameplay.

Introduction

Most modern adversarial games use Artificial Intelligence to simulate human antagonists, allowing for rich gameplay against challenging opponents. These AI are usually given an unfair advantage; knowing everything about the map and player position. While this can lead to fun and challenging opponents, the issues with this approach become prevalent in stealth based games, where the AI's near omniscience can quickly become frustrating and make the game unplayable.

The solution can be found in limiting the AI's information about the world, restricting their view and conceivable actions to something more human-like. The problem then becomes how can an AI reason about a

world it cannot fully see? Many of today's enemy searching algorithms choose to utilize a fixed map and give the AI a list of places to search, which can be a fine implementation, but this approach cannot account for a changing environment or unexpected player behavior; if the player can find a hiding spot the AI will never even think of checking, that is a major problem. Another approach is to have the AI search the player's last known position, but this method does not help the AI act in a human-like manner if it does not even try to persist in searching other areas. This old method is rather inflexible and has trouble accounting for enemies moving while hidden from view.

This paper seeks to propose using a probability distribution map of the player's likely position as the basis for AI in any stealth-like game. This will ideally allow for greater flexibility in hiding spaces for players and a more human-like approach to searching for the player because the AI's omniscience is removed, and is dependent only on its field of vision. The key here is that this probability map of the world is built up over time as the AI receives information about the world, allowing it to build something it can then reason on.

Background

There are not terribly many in production games that would focus on this aspect of gameplay; stealth based games are usually cater to a patient niche of players looking more for the euphoria of winning without alerting a single guard. While most stealth games use AI characters as antagonists, I do not know of any utilizing probability to track player position while the player is unobserved. There are games that simulate it; the *Thief* franchise uses visual and auditory 'senses' to give information to their AI, specifically view cones that let the AI 'see' the world around them, providing information to act on. The AI in *Tom Clancy's Splinter Cell, Conviction*, utilizes the player's last known position but does not try to figure out where the player has gone beyond that based on probability.

Problem

Because the AI needs to be segregated from the world for this approach, there are four main problems to overcome: AI field of vision, exploring the map, player prediction propagation, and AI path determination. Combining the solutions for all three should make for an AI that can explore an unknown space while looking for a moving enemy. Interestingly, the player prediction map does not need to be continuously updated, as it is useless when the player hasn't been seen before, or is currently observed.

The AI should act differently based on what it has observed, leading to 3 general states: Exploration, Pursuit, and Predictive Tracking. Exploration is the default state, consisting of the AI determining the closest unknown location on its probability map and trying to path there using A*. Pursuit is the state wherein the AI can actively see the player in its field of vision. In this state, A* is used to path from the AI's position to the player's position. Predictive Tracking is when the player has been seen, but is currently not in the AI's field of vision. This state uses the map built up in the other states to reason about where the player might have gone. Because the AI doesn't have a complete map of the world, it has to assume all unknown grid positions are freely movable spaces with regards for estimating where the player can travel until it can update it's understanding of that space.

Implementation

1. Field of Vision

The first step is determining what the AI can see of the world. The easiest way to do so is with a cone of vision, consisting of a multitude of rays extending out from the AI with a starting and ending angle that terminate once some maximum length or a wall is reached. These rays are generated using the Bresenham Line Algorithm, modified to return a list of tuples containing a map position and whether that position is a

wall or a space. These lists are combined in a unique union, meaning the combined field or rays will only contain one of each position, even if multiple rays trace it.

2. Map Exploration

Map Exploration is simply done by determining where the closest void position is and setting the AI to path to that position. This is done by making a list of all positions adjacent to free space nodes that are not yet known by the AI, then iterating over all of those to find the one closest to the AI. Because this is a cheap calculation, it can be done repeatedly very easily. When combined with actively orienting the field of vision to that spot, the AI often never has to actually reach that spot before moving onto the next one.

A video of the AI exploring the space can be found here. Notice how in the first run, there is a point at which the A* pathing takes a long backwards 'C' shaped route to an unknown location. This bug, explained later stems from the A* library's inability to path to a previously visited location.

3. Predictive Mapping

Once sight of the player has been lost, the AI needs to be able to reason about where the player has gone. Since the AI cannot know how the player moved to evade the field of vision, the player's last known position on the probability density map is set to 1, and the rest of the map is set to 0. We do this because we know exactly where the player was, he cannot be anywhere else. Immediately after, the probabilities across the map are propagated once to simulate the player's move to this frame. The AI then uses A* to determine the best path to the highest probability on the probability map.

Probability propagation is done in an interesting manner; instead of polling the adjacent nodes to update a node, the node updates its adjacent nodes and itself based on the current probability of the player

being in either position, and the probability of the player moving to that node using a discrete sum:

$$P(X) = \Sigma_a(P(X_a) * P(X_a \rightarrow X))$$

Where X is the probability that the player is at a specific map position, X_a is all adjacent positions including X, and $P(X_a \rightarrow X)$ is the probability of the player moving from X_a to X based on the number of available movement options. Belief propagation is done this way to preserve the net probability of the map and because the probabilities are propagated from the nodes rather than to them, allows the AI to set probabilities to nodes it has not yet seen. This allows the AI to reason on unknown nodes, which can be updated later with their actual values.

Consider the following example where the AI has spotted the player (fig 1), but by the next frame, the player has moved out of the AI's line of vision (fig 2).

Figure 1

The AI is the red square; the player is the green square. The AI's line of vision is white and currently includes the player's square. There is a wall to the left of the player.

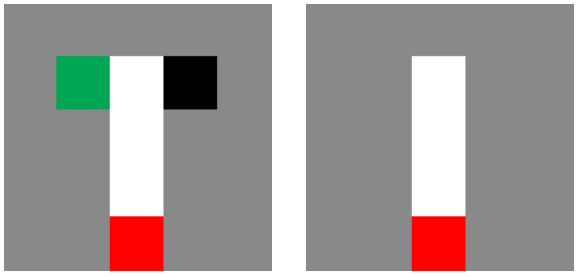

Figure 2

The player has moved out of the AI's cone of vision. Notice how the AI can see neither the player nor the wall, because of its limited cone of vision.

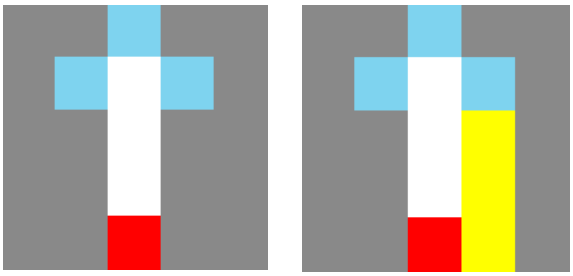

Figure 3

Because the AI has no idea which direction the player went, nor any idea about the space currently outside its cone of vision, it must reason the player has moved to one of the three positions adjacent to the player's last known position (fig 3). In this case it decides to check the rightmost location using the yellow path.

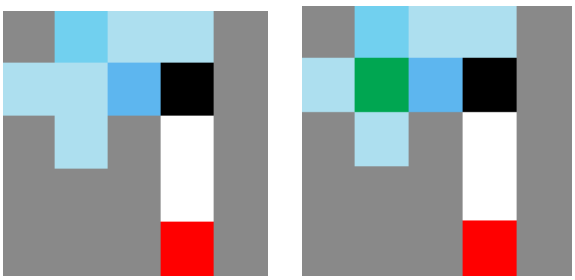

Figure 4

One frame later, the player probability map has been propagated, and the AI has updated it with the fact that there is a wall where it originally thought there was a player. Notice how the player is not in one of the two spots of high probability; this method is not and shouldn't be an infallible approach. The ideal is giving a general sense of the player's location with which the AI can check.

4. AI Movement

While pathing through the known states was a relatively simple implementation of A*, The AI also had to guess how it could find a path through a partially unknown map, due to differences in how the field of view ray casting and the A* pathing algorithm worked. An example is shown below in Figure 5.

Figure 5

Here the AI's vision cone (in white) can see the player diagonally through the missing corner, but there exists no path through the known states (also in white) to the player's position. The solution to this problem is to have the AI guess that all unknown nodes are empty space until it can observe otherwise and can update its assumptions.

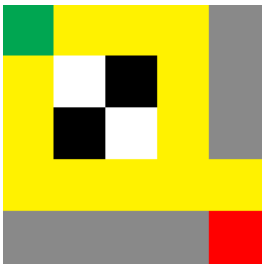

Figure 6

Figure 6 shows two possible paths to the player's position using a permissive known states map. If the AI chooses the lower path, it will eventually see the wall blocking that route and choose the other path.

Results

The game can be played here. The AI tries to fully explore the map and does a decently good job of following the player once seen. It can be fooled, and there are some problems in its predictive mapping. The game runs very quickly and does not appear to use much processing power. The FPS was limited to 5 to make it easier for players to look at the map and determine what to do. Because to the AI's decision making and per-frame movement speed, the player's best bet is to find a blind spots to hide in or hope the AI gets stuck somewhere.

Probability propagation could use some work, however. It tends to stay very low for the most part after a few propagations, but some very odd high spots occur after a long time of propagation. It does do a decently good job predicting player movements within the first few propagations however.

Issues

The AI can occasionally become stuck once it believes it has explored the entire map and not found the player anywhere, as there is no way for it to realize it needs to recheck the map, yet. The library used for the A* search sometimes cannot determine the optimal path back to a previous state, leading to amusing situations where the AI is aware of the player but cannot determine a path to the player's location. Finally, there are also sometimes occurrences of the AI getting stuck on trying to path through walls after it has seen the player. I believe this is due to it not correctly accounting for some walls once it sees them.

Due to time limitations, I was unable to implement my own version of A*, and while the library I found was useful, it has several limitations, namely being constrained to a 2D grid meaning the AI is currently confined to a grid type of map, and the aforementioned issue with return pathing.

The probability propagation can sometimes lead to weird results if the AI gets stuck and left to propagate for a long time. The fix for this might be a limit on how long the probability map is allowed to propagate after the AI loses sight of the player. It might also be because the AI currently might not be correctly updating the grid used for pathing through the space. Constructing a probability density function might work, but would require compensating somehow for walls.

The cone of vision has some holes in it depending on the resolution it is set to, and is non-optimal due to the fact that often several rays will trace the same grid square.

Future Work

Designing the probability map to be a hidden layer under a less restrained visualization of the game would be ideal, allowing for more movement options and game display styles than the 2D grid world allows. In addition, I would like to implement my own A* search algorithm that does not involve a base grid, allowing for a more freeform map than currently used.

Combining multiple AI to search together might be an interesting problem, seeing whether separate or shared probability distribution maps contribute to better decision-making or emergent behavior.

The probability distribution map does not need to be as exact as I first thought; some function that averages the player's probability between two adjacent nodes might make for a better looking probability distribution and might alleviate the problem with weird high-spots.

Conclusions

A probability distribution map does a good job of predicting player position in a known environment. However it tends to fail to produce

anything meaningful after a large amount of propagations, as the chances of finding the player after a certain point is simply too low.

Interacting with the cone of vision leads to interesting gameplay, and the ability of the AI to reason on the player's probable location makes trying to fool it exciting and a moderately hard challenge. It can easily follow me into a blind corner, for example.

There is some interesting emergent behavior on the initialization of the game; the AI starts out by first looking in all 4 directions, as those are the closest unknown nodes as it starts exploring.

The base AI should scale relatively well for larger maps, most calculations are done in O(n), where n is the number of known map locations. This should prove useful to anyone wishing to use this as a logic layer for a more graphically complex game.

Works Cited

Champandard, Alex J. "Occupancy Grids and Emergent Search Behavior for NPCs." *Occupancy Grids and Emergent Search Behavior for NPCs*. N.p., 28 Apr. 2011. Web. 17 May 2015. <http://aigamedev.com/open/tutorials/occupancy-grid-prototype/>.

Champandard, Alex J. "Sneaking Behind Thief's AI: 14 Tricks to Steal for Your Game." *Sneaking Behind Thief's AI: 14 Tricks to Steal for Your Game*. N.p., 17 Sept. 2007. Web. 17 May 2015. <http://aigamedev.com/open/review/thief-ai/>.

Klingensmith, Matt. "Moving from Point A to Point B: Agents and Goals." *Gamasutra Article*. N.p., 7 Sept. 2013. Web. 17 May 2015. <http://www.gamasutra.com/blogs/MatthewKlingensmith/20130907/199787/Overview_of_Motion_Planning.php>.

Russell, Stuart J., and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Upper Saddle River: Prentice-Hall, 2010. Print.

Bourgault, Frederic, Tomonari Furukawa, and Hugh Durrant-Whyte. "Optimal Search for a Lost Target in a Bayesian World." SpringerLink. Springer Tracts in Advanced Robotics, 7 July 2006. Web. 17 May 2015. <http://link.springer.com/chapter/10.1007/10991459_21>.