State Estimation for Game AI

How to hunt the player with more variety and let them trick us

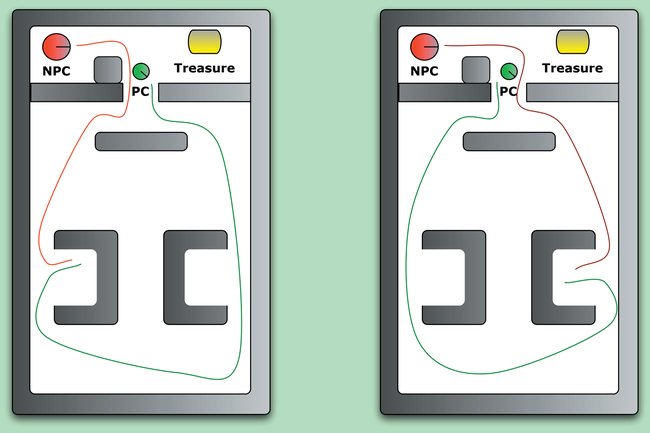
Curt Bererton (curt.bererton@gmail.com)
Carnegie Mellon University, now at Ikuni inc.

The Problem:

Non-Player Character (NPC) AI cheats too much. We use knowledge of the player's location to track them down even when we should not. This leads to predictable and unfair behavior of our NPCs.

Exhibit A:

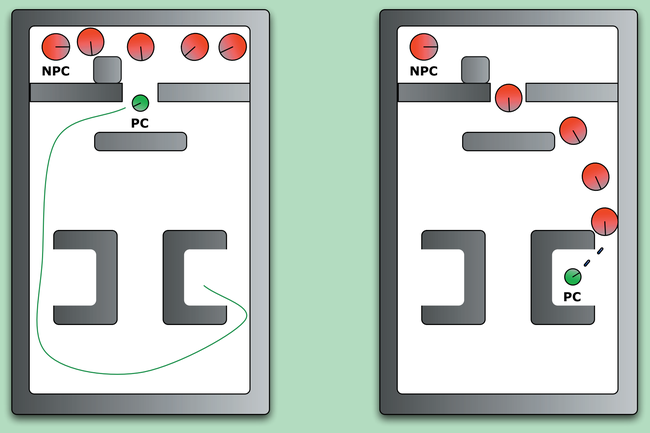
I can run but I can't hide!



The player can't hide because the NPC is using the PC's position as a goal for the path planner. The NPC will always find the player using unrealistically direct paths. No matter how the player runs all over the map, the NPC will find them.

Exhibit B:

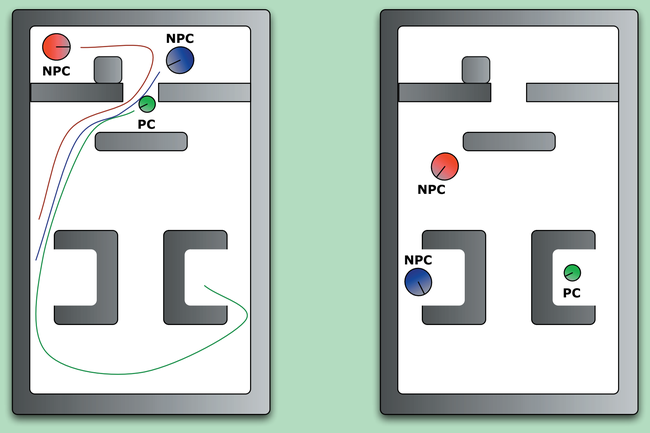
NPC cannon fodder



Identical NPC senses, state machines, and similar behaviors lead to predictable and identical AI. The player uses the same technique to win every time. NPCs often line up to die one after the other because they are all exactly the same.

Exhibit C:

Poor searching behaviors



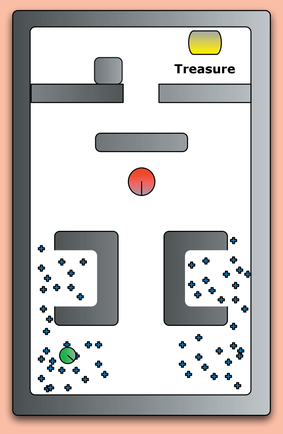
How can you pretend to search for the player if you don't maintain a representation about where the player could be since you last saw them? Often NPC AI just gives up rather than performing a search of the area as real people would do.

The Approach:

Particle Filters

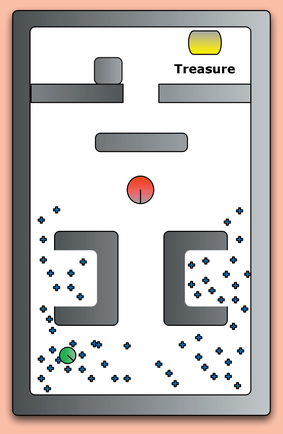
Use a technique from robotics called particle filters. This technique has been used on real robots to find targets in 2D and 3D environments using real sensors. Particle filters use the following four steps:

1. Starting Distribution



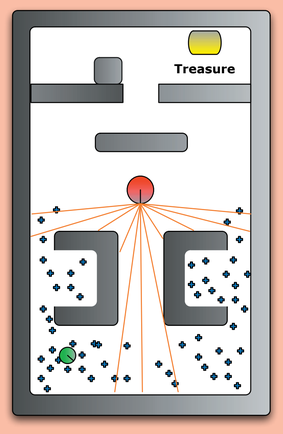
Each particle (represented by the small plus signs) represents a location where the NPC thinks the player might be at the current time and also how probable it is that the player is there.

2. Proposal Distribution



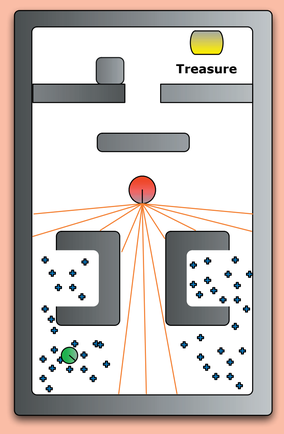
The NPC thinks about where each particle could move during the current time step (one frame). It then moves the particles around to reflect where the player could have moved.

3. Observations



The NPC then observes the environment. If it cannot see the player, then it knows that the particles that it has in its line of sight must be incorrect guesses as to the position of the player.

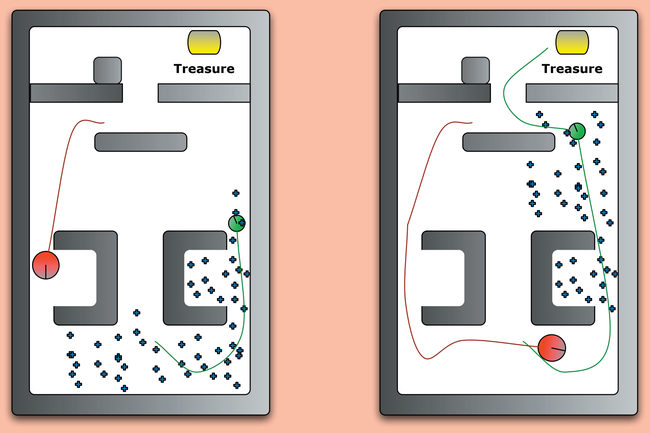
4. Resample



It then removes these low probability guesses during a resampling step. It then repeats the whole procedure on the next frame.

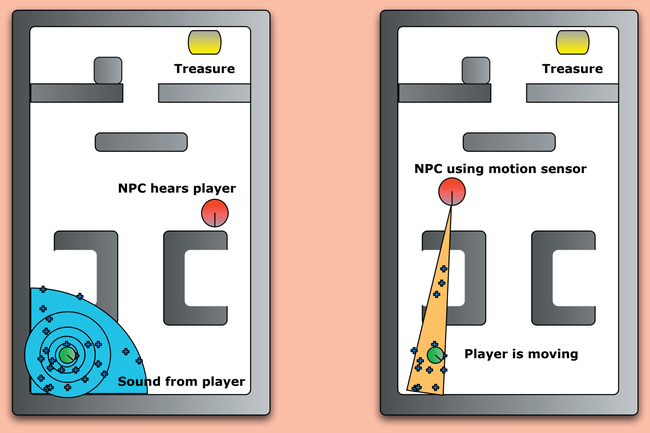
Solutions to exhibits A,B, and C:

A:



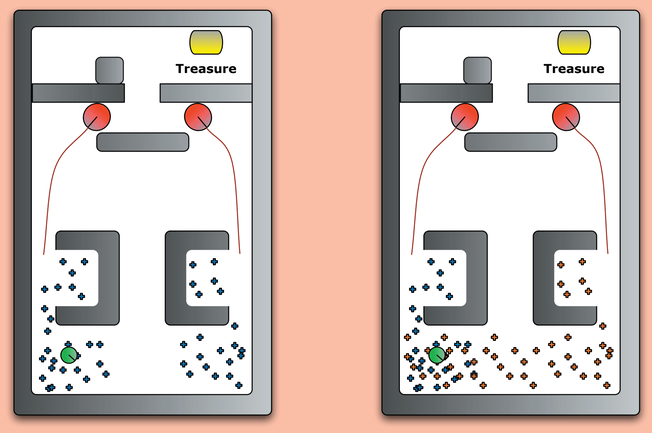
Use the particle filter to reason about which areas to search. While we are searching the bottom, the player can evade us and get the treasure at the top. We can also tune how much each NPC remembers about where they've looked by using more or less particles.

B:



We can also use different sensors with more or less accuracy for different NPCs We can use entirely new sensor types like smell or heat. All integrate into the same achitecture. Behaviors using these different kinds of input allow for a wider variety of NPC intelligence.

C:



Particle filters allow us to reason about where we have and have not searched using a solid mathematical framework. In addition, we can choose to have NPCs work together (left) or each look for the player separately (right).

Conclusions:

Particle filters are already successful in robotics for finding and tracking targets using multiple agents. They can be successfully applied to many game genres yielding new and interesting NPC behaviors. For more information come to the presentation to hear a more detailed description and see a brief video. Also see www.cs.cmu.edu/~curt/research.html