32

How to Catch a Ninja

NPC Awareness in a 2D Stealth Platformer

Brook Miles

32.1 Introduction

Mark of the Ninja is a 2D stealth platformer game by Klei Entertainment. The player, as the Ninja, sneaks through levels keeping to the shadows, crawling through vents, and ambushing unsuspecting guards. The engine we used, however, was based on Klei's previous game *Shank 2*, and if Shank knows one thing … it sure isn't how to be sneaky.

The AI enemies in *Shank* and *Shank 2* only cared about attackable targets (the player, or multiple players in a coop game). They would spawn, choose an appropriate player to attack when one came within range, and do so until they killed, or were killed by, that player.

This worked well for the *Shank* games, which focus on constant, head-on combat. But for *Mark of the Ninja* we needed more subtle behavior. Guards needed to have multiple levels of alertness; the player needed to be able to distract them with sounds or movement,

or break line of sight to escape detection before circling back to strike from behind. Guards needed to display some awareness of their surroundings, notice when something is amiss, investigate whatever catches their attention, and respond to fallen comrades.

To address the need for AI characters to be aware of events and objects in the world around them, one of the changes we implemented was a data-driven interest system that allows designers or scripters to define **interest sources** in the world, and have agents detect and respond to them appropriately.

In *Mark of the Ninja*, targets and interests are similar concepts; they each represent an object with a position in the game world that an agent is aware of and should react to. However, there are a significant number of differences, both in the data used to represent each concept and in the associated behavior of the agents, which resulted in the decision to implement them as separate entities within the game.

Perhaps most importantly, all targets are basically created equal: if it's an enemy, shoot it! Most of the processing going on while an agent has a target is dedicated to tracking its position and attacking it. An agent that has a target will always be on high alert, and will disregard most other stimuli until either it or its target is dead. Interests, on the other hand, represent things that are not targets; they are assumed to be stationary but are much more numerous and varied, as we will see later. When an agent has an interest, it will usually attempt to investigate the interest and search the surrounding area, all the while keeping an eye out for targets, or other potential interests of greater importance.

32.2 From Shank to Ninja—Noticing Things Other Than Your Target

As part of the *Ninja* branch from the *Shank 2* source tree, agents initially gained the ability to notice **points of interest**. A point of interest consisted simply of a 2D point in the level, something to approach but not necessarily shoot at.

These points of interest were created in the update loops of each agent's brain from sources such as sounds, dead bodies, and broken lights. The update loop would collect and iterate over each type of game object or, in the case of sounds, a simple list of 2D points. After the list of potential interests was collected, one would be chosen based on proximity, or other hard-coded criteria, to be the current interest which the agent would then respond to. This setup worked in some cases, but there were significant problems that we wanted to overcome.

First, if the designer wanted an agent to take interest in any new and previously undefined object or event, they would need to request a programmer to add a new set of checks into the brain, and possibly new data structures to track whatever was being sensed. It was already apparent at the time that this process could be tedious and time consuming, but it ultimately would have proven to be a serious limitation. We ended up with around 60 different types of interest sources in the game and development may have been seriously hampered by this programmer-dependent process.

Second, there was no accounting for multiple agents reacting to the same point of interest. If you made a loud noise, everyone nearby would come running, which makes some sense, but what if you just broke a light? Does it make sense for a group of four guards all to walk over and stare dumbly up at the light, each remarking separately to themselves that somebody really ought to do something about that broken light?

If a group of agents standing together detects an interest source, ideally one or two of that "group" (see Section 2.5 for more on how *Ninja* deals with groups) would be dispatched to check it out, while the rest are simply put on alert and hang back waiting for the result of the search. Not only does this feel more natural, it provides more interesting gameplay opportunities to the player, allowing them to separate closely clustered guards and deal with them individually off in a dark corner, instead of running into a brightly lit room and getting shot by five guys with automatic weapons.

32.3 Senses

We determined that fundamentally there were two broad categories of interest sources our agents needed to detect in the world, things they could see, and things they could hear, which gave us our two senses: **sight** and **sound**.

Detection by the sight test involves a series of checks including these questions: Is the interest source within one of the agent's **vision cones**? Is the game object associated with the interest source currently lit by a light source, or does the agent have the night vision flag, which removes this requirement? Is there any collision blocking line of sight between the agent's eye position and that of the interest source?

A vision cone, as shown in Figure 32.1, is typically defined by an offset and direction from the agent's eye position, an angle defining how wide it is, and a maximum distance. Other vision geometry is possible as well; we have some which are simply a single ray, an entire circle, or a square or trapezoid for specific purposes.

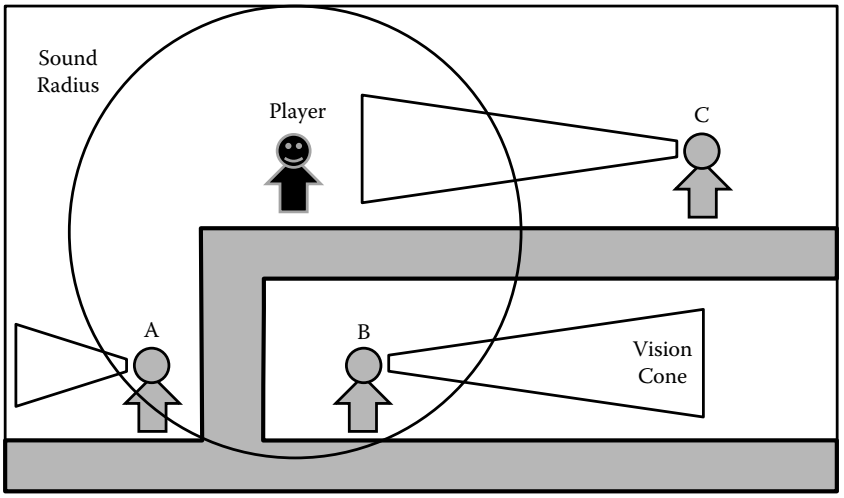

Figure 32.1

The player has made a noise by running, represented by the Sound Radius circle. Guard A will hear the sound and turn around to investigate. Guard B is within the Sound Radius but can't hear the sound because there is no path from the guard to the sound source. Guard C is outside the Sound Radius and can't hear the sound; however, if the player enters Guard C's vision cone, the player will be spotted.

Detection by sound is determined by pathfinding from the interest source's position to the position of the agent's head. When a designer exports a level from our level editor, one of the steps is to generate a triangle mesh from all of the empty space within the level (we also generate the inverse mesh, of all of the solid collision in the level, for the purposes of rendering the map). We use this "sound mesh" at runtime to perform A* pathfinding [Millington 06] from the "sound" interest source to any agent who might potentially hear it.

This same pathfinding operation is performed by the audio system between the player character and the source of sound events within the world to determine the amount of filtering to apply. Both the sound and map meshes were generated using the program **Triangle** by Jonathan Shewchuck [Shewchuk 05], which generates very clean meshes and has a variety of useful options to control mesh quality, density, and other interesting properties for those more mathematically inclined.

For both performance and gameplay reasons, sight and sound interest sources define a maximum radius, and only agents within that radius are tested to determine whether they can detect the interest source.

32.4 Definition of Interest Sources

From our two core senses, we can now allow the designers to create an **interest source** representing whatever object or event they want, so long as it can be detected via the sight or sound tests. Designers can specify a "gunshot" sound, or a "footstep" sound, a "corpse" sight, or a "suspect" sight. The agent's behavioral scripts can use the specified interest source type to determine any special behavior, but the backend only needs to know how to determine whether the agent can see or hear the interest source.

You can also bend the definition of "sight" and "sound" somewhat. One special case of agent in *Mark of the Ninja* is guard dogs, who we want to be able to "smell" the player in the dark but, for gameplay reasons, only over a very short distance. In this case, instead of needing to create an entirely new smell test, we can simply define a sight interest source which is attached to the player game object, and require that any agent noticing it have the "dog" tag as shown in Listing 32.1. Voila, we have a "smell" interest.

Listing 32.1. Just a couple of the 60 or so different interest source declarations used in the final game.

```
CreateInterestSource {sense = "sound", priority =
    INTEREST_PRIORITY_NOISE_LOUD, radius = RUN_ON_LOUD_RADIUS,
    ttl = 4*FRAMES, offset = {0, 1.5*TILES}, forgetlosttarget = true}

CreateInterestSource{sense = "sight", source = "suspect", priority =
    INTEREST_PRIORITY_SUSPECT, radius = INTEREST_RADIUS_SUSPECT, ttl =
    INTEREST_TTL_FOREVER, canberediscovered = true, noduplicateradius = 0,
    removeduplicates = true, followowner = true, condition =
    HasAttributeTag("dog")}
```

32.4.1 Interest Sources Live in the World, Interests Live in Your Brain

An **interest** is just the record in the agent's brain of what he's interested in right this moment. It may have a reference to the interest source that created it (if there was one), but even if it doesn't, it still has a copy of all of the necessary information, the sense for the interest, the source type, its position, priority, and so on. When an agent is determined to have detected an interest source, an interest record is added to its brain, and this is the information that the agent uses from that point on.

While sight and sound are the only available sense types for interest sources in *Mark of the Ninja*, interests of any arbitrarily defined sense can be added directly to an agent's brain by the designer through a script call, as no additional testing needs to be done against them; the designer or script writer has already determined that this agent should be interested in whatever it is.

For example a "touch" interest may be added to an agent's brain when he is struck with a dart projectile, or a "missing partner" interest can be added if the agent's partner goes off to investigate a sound and fails to return.

32.5 Driving Updates from Interest Sources and Lightweight Group Behavior

A question that arose early on was how groups of agents should respond when they all sense an interest simultaneously. At first it was every man for himself; each agent did its own test and upon sensing an interest would react, most likely by running to investigate it. This typically resulted in entire groups of agents running towards the slightest noise or converging on the player en masse. This wasn't the kind of gameplay we were looking for. We want the player to be able to manipulate the agents, distract them, split them apart, and dispatch them on the player's own terms.

We looked for ways in which we could have some level of apparent cooperation between the agents without going so far as to explicitly manage group behavior or coordinated movement. We really only needed guards reacting to the same thing, at the same time, to have a variety of responses. Some should go and investigate, one might play a line of audio dialog telling another nearby guard to go check it out, and others might just glance over and wait for the previously mentioned guards to deal with the situation. Instead of driving this scenario from the agent side and attempting to coordinate updates, or creating a separate concept of grouping, we chose to rely on the implicit group that already existed: the set of agents who detected an interest.

By driving the detection of interest sources from the interest source itself, instead of from each agent individually, we can easily collect all of the information we need in order to determine who should be reacting, and how they should react.

The sensory manager update loop tests each interest source against all possible "detection candidates." Given this list, it makes some decisions based mainly on group size, but possibly also by location or distance from the interest source. If only a single agent can detect the interest, our work is done, the agent is notified, and he goes to investigate. If more than one agent can detect the interest, we can assign **roles** to each agent, which are stored along with the interest record in the agent's brain. Roles only have meaning within

the context of a specific interest, and when that interest is forgotten or replaced, the role associated with it goes away too.

If multiple agents can detect the interest, one is chosen as the "sentry" or "group leader" and he plays audio dialog telling the other agents nearby to go check out the interest and then hangs back waiting. One or more agents are given the "investigate" role and will go and investigate, seemingly at the command of the "group leader." Any remaining agents will get the "bystander" role, and may indicate they've seen or heard the interest but otherwise hold position and decrease the priority of the interest in their mind so they are more likely to notice new interests for which they might be chosen as leader or investigator.

The key is that once the roles are assigned, and the sensory update is complete, there is no "group" to manage. Each agent is acting independently, but due to the roles that were assigned, they behave differently from each other in a way that implies group coordination.

32.6 Prioritizing Interests

If you are investigating a broken light and come across a dead body, should you stop and investigate the body, or continue to look at the light? What if you hear your partner being stabbed by a Ninja, and then discover a broken light on your way to help him? Should you stop to investigate? Clearly, certain types of interest must be prioritized over others.

Interest sources, and by extension interest entries in agents' brains, contain a simple integer value of **priority**, where higher priority interests can replace lower or equal priority interests, and the agent will change his focus accordingly. If the agent currently holds a high priority interest, lower priorities interests are discarded and never enter the agent's awareness.

Balancing the priority of various interests turned out to be a challenging and ongoing task all throughout development. Listing 32.2 shows what the priorities for various types of interest sources were near the end of the project, but they had changed many times during development in response to unanticipated or blatantly unrealistic behavior, resulting from the current set of interest priorities. Making them easy to change and try out new combinations was a big help.

Initially, we assumed that finding corpses would be one of the highest priority interests in the game, superseded only by seeing a target (the player), but it turned out to be not so

Listing 32.2. *Mark of the Ninja*'s priority definitions for interests.

```
INTEREST_PRIORITY_LOWEST = 0
INTEREST_PRIORITY_BROKEN = 1
INTEREST_PRIORITY_MISSING = 2
INTEREST_PRIORITY_SUSPECT = 4
INTEREST_PRIORITY_SMOKE = 4
INTEREST_PRIORITY_CORPSE = 4
INTEREST_PRIORITY_NOISE_QUIET = 4
INTEREST_PRIORITY_NOISE_LOUD = 4
INTEREST_PRIORITY_BOX = 5
INTEREST_PRIORITY_SPIKEMINE = 5
INTEREST_PRIORITY_DISTRACTIONFLARE = 10
INTEREST_PRIORITY_TERROR = 20
```

simple. If you hear a sound, and while investigating the sound you discover a body, clearly it makes sense to pay attention to this new discovery. But what if the situation were reversed? If you're investigating a body and you hear a sound, should you ignore it? This is potentially very unwise, especially if the noise is footsteps rapidly approaching you from behind.

It turns out that corpses, noises, and a variety of other specific interest types should all be treated on a newest-first basis. The last thing you notice is probably the most important thing. Our solution here was simply to make this set of interests all the same priority. However, new interests of equal priority (to your current interest) are treated as more important.

Other interests truly are more or less important than others, regardless of the order they are encountered. Seeing a broken pot is always less interesting than a shadowy figure or a gunshot. Seeing your partner impaled on a spike trap is always more important than the sound of glass breaking in the distance.

32.7 Investigation and Rediscoverability

Once an interest source has been noticed by an agent, the agent has an opportunity to "investigate" it, which might simply mean looking at a broken light and commenting that it should be fixed. Or it could involve seeing a fallen comrade, running over to check for a pulse, and then calling the alarm.

Once an agent determines that an interest source has been dealt with, particularly in the case of mundane things like broken lights, we really don't want every guard that walks past to stop and take notice. This would be repetitive and doesn't make for especially compelling gameplay. Even worse would be the same agent noticing the same interest source over and over. When an agent has completed whatever investigation is called for, the interest source associated with the agent's interest record is marked as investigated, which then removes the interest source (but not the game object it was associated with) from the world, never to be seen or heard of again.

This still doesn't completely solve our problem, though; in the case of corpses, for example, what happens if you draw the attention of a guard who is investigating a body before he has a chance to thoroughly investigate and sound the alarm? After losing his target, he may turn around and see the body again. It doesn't make sense for him to be as surprised to see his fallen comrade as he was a moment ago. It would be natural for him to return to complete his initial investigation, but this ultimately felt like going a little bit too far down the rabbit hole. We made the decision that each agent would be aware of only one interest at a time and there would be no stack or queue of interests. Once the highest priority interest was dealt with at any given time, we wanted the situation to naturally reset to a default state and not have the guard continuing on to revisit every past source of interest they may have come across during an encounter.

By default then, we only want each agent to detect or notice each interest source once. When that happens, we record that this discovery took place and that agent will be excluded from noticing the interest source again. Since we drive updates from the direction of interest sources looking for agents to discover them, the interest source keeps a list of discoverers and doesn't cause itself to be noticed by the same agent twice.

In rare cases, we do want an interest source to be noticeable multiple times by the same agent, the primary example being the "suspect" interest source attached to the player. This interest source is detectable from farther away than the agent's ability to see the player as

a target, and so draws them in to investigate without immediately seeing and shooting the player. We want this to happen every time the agent detects the player, and so this particular interest source is marked as being rediscoverable, and it doesn't bother keeping a list of who has seen it in the past.

32.8 Using Interests for Lightweight Agent Scripting

In addition to allowing agents to passively notice events or objects in the world, there are various situations in the course of designing the game levels where some level of scripted encounter is desired. In some cases that scripting is quite rigid; you want an agent to play specific animations at certain times and at specific places. For these cases, more specific level scripts are created.

Other times, however, it's acceptable or desired to set up a more natural situation and let it play out, possibly with interference from the player. Instead of scripting an agent to follow a specific path to a specific point, play a specific line of dialog, and so on, an interest can be added directly to the agent's brain. The agent will then pathfind normally to their destination interest point, opening doors and so on as he goes. On reaching his destination, he will perform his normal search pattern. When he gets there, he may notice the object placed there for him to notice, in which case he can comment on it and continue investigating as usual or perform other contextual actions as a result.

The benefit of this simple approach is that no special handling needs to occur if the player alters the parameters of the situation. Perhaps the player distracts the agent before he reaches his goal. Or perhaps the player turns off the lights in the room the agent is headed towards, preventing him from noticing what he would have in the first case, causing him to give up and return to his patrol. No special cases need to be scripted for these situations, as the agent's behavior wasn't a script to begin with.

Other objects that the agent encounters, as this situation plays out, can be handled locally by behavior scripts independently of (or in conjunction with) the agent's current interests. Agents know how to open doors and turn on lights as they encounter them, and it's not necessary for these actions to interfere with interest handling.

Creating an interest for every possible little thing in the world that the agent can interact with is unnecessary and unwieldy, partly because of the rule of having only a single interest at a time. When an agent comes across a door, he doesn't need to be interested in it; his navigation and behavior scripts simply determine he needs to walk through a door and acts accordingly, after which he continues on his way investigating whatever his current interest is.

32.9 Limitations and Improvements

We found it difficult to achieve sensible behavior from agents who are receiving quickly repeating or alternating interests. It's not always immediately obvious what an agent should do when his current interest changes or moves from moment to moment. While this is primarily a problem outside the scope of interest detection, there was some special handling done in order to help address the issue. Specifically, when acquiring a new interest, the agent will compare the incoming interest to the current interest if any. If it's the same sense, source type, priority, and is relatively close to the previous interest, then

it's not handled as a new interest, but the timers and position of the current interest are updated to reflect the new information. A primary example of this is the player creating a series of footstep interests while running. It's not desirable to treat each footstep as a new interest.

Similarly, there is no direct support for tracking the movement of a single source of interest, as the vast majority of interest sources in the game have a fixed position. This imposes some limitations on their use; for example, one of the player's inventory items in *Mark of the Ninja* is a box made of cardboard that you can hide in. One implementation involved attaching an interest source to the box when the player (hiding inside) moved, as a way to draw the attention of nearby agents. However, this didn't result in the desired behavior, as most of the interest source handling code and scripts relied on an interest's position not changing after detection.

Section 32.4 touches on the idea of providing a "condition" that is evaluated against agents who may potentially detect an interest source. This functionality was added towards the end of the project and as a result didn't end up being used extensively. When used with caution, it can provide an extra dose of flexibility for all of those special cases that will invariably crop up during a project, but attention must be paid to avoid embedding too much complicated logic in the condition that would be better placed elsewhere.

32.10 Conclusion

What was originally intended to deal with the issue of noticing dead bodies and broken lights soon expanded to an ever increasing variety of other purposes, as described in this chapter. While by no means a silver bullet, this system granted the designers more control over the behavior of game characters, while reducing the need for special case scripting or programmer intervention. Overall, the sense detection architecture which assigns agent rates during the update of an interest source, combined with the designers' ability to data-drive interest source definitions and priorities, resulted in a simple but flexible system that created believable guard awareness and even provided light-weight group behavior with minimal additional complexity.

Acknowledgments

We would like to thank the entire team at Klei for making *Mark of the Ninja* a joy to work on, in particular Kevin Forbes whose work on *Shank 2* provided the foundation for the AI and who provided much helpful direction in expanding on those systems; and fellow AI/gameplay engineer Tatham Johnson who also contributed to the work described in this article.

References

[Millington 06] I. Millington. *Artificial Intelligence for Games*. San Francisco, CA: Morgan Kaufmann, 2006, pp. 233–246.

[Shewchuk 05] J. R. Shewchuk. "Triangle: A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator." http://www.cs.cmu.edu/~quake/triangle.html, 2005.