Building a Stealth Game Around Occupancy Maps

THIRD EYE CRIME

Damián Isla, Moonshot Games
THIRD EYE CRIME

GAME NOIR BY MOONSHOT

MOONSHOT GAMES
The [Modest] Goals

1) Create a game in which AI is the core mechanic / source of fun.
   - Just like physics is for physics games.
   - Just like A* is for tower defense

2) Prove that Occupancy Maps can be that source of fun
Agenda

- Intro
- Occupancy Maps
- Third Eye Crime
  - Top 4 Technical Hurdles
- Conclusions
OCCUPANCY MAPS
Mea Culpa

“Occupancy Maps” means something very different to the mobile robotics community ...

- method for sensor fusion + self-localization / mapping
- aka “occupancy grids”

They coined the term first.

It’s theirs.

And sooner or later, they’re going to want it back.
Represent target object location as a discrete probability distribution over space
2 Questions

1. Why do we want to do this?
   - What’s wrong with \((x, y, z)\) location representation?

2. Why not use a neater distribution representation?
   - e.g. Gaussian
Positive information (Expectation validation)

Negative information (Expectation violation)
OMap Algorithm

If target observed:
Find closest cell \( n^* \)
\[ p(n^*) \leftarrow 1 \]
\( \forall n \neq n^*, p(n) \leftarrow 0 \)

Otherwise:
Divide map cells into visible (V) and non-visible (N) sets
\[ p_{culled} = \sum_{n \in V} p(n) \]
\( \forall n \in V, p(n) \leftarrow 0 \)
\( \forall n \in N, p(n) \leftarrow \frac{1}{1 - p_{culled}} p(n) \)

Either way: Diffuse Probability
The Big Win

Emergent Search

```c
void s_agent::behavior_update()
{
    if (omap.confusion < k_pause_threshold)
    {
        s_pos2d pos= omap.get_target_position();
        move_to(pos);
    }
}
```
3 Big Ideas

- Make a *casual* stealth game with good AI Perception/Search
  - deemphasize light/dark gameplay, intermediate alertness states
  - emphasize the *chase* rather than the *sneaking*

- Render the OMap
  - AI debugging as a gameplay mechanic
  - omap = telepathy

- The Cut the Rope of AI
  - short levels
  - challenge stars
  - micro-systemic vs. macro-systemic
Top 4 Technical Hurdles

1. Visibility Testing
2. Diffusion dynamics
3. Group perception + behavior
4. OMap Rendering
1) Visibility Testing

Determine each time-step which cells are visible and which are not.

- Potentially a huge perf cost

Five approaches

- raycasting
- cached LOS
- POV rendering
- In 2d: rasterization
- In 2d: GPU-based stencil madness

THIRD EYE CRIME
Visibility Testing

For simulation:

**Rasterization approach**
- Gives us binary visible/occluded map at OMap’s resolution.
- Computed on CPU

Complicated. But broadly:
- Involves tracking visible spans of pixels over a single quadrant
- Spans of visible pixels shrink, and split, but never grow.

For rendering:

**Stencil Madness**
- Produces device-resolution vision cones
- Based generally on stencil shadows
- Computed on GPU
Visibility Testing

For simulation:

Rasterization approach

For rendering:

Stencil Madness
2) Diffusion Dynamics

\[ \alpha \frac{p}{\sqrt{2}} \]
Diffusion Dynamics

Problem: AI repeatedly check the same places
Diffusion Dynamics

Problem: AI repeatedly check the same places

Solution:
- Vary incoming diffusion rate as $F(\text{time-since-observed})$
3) Group Perception + Behavior

Never simulate/display more than 1 Occupancy Map at a time.

- too expensive to compute
- too confusing to the player
Group Perception + Behavior

Multiple AIs tracking a single target
- Technically, each should have their own OMap
- Instead they SHARE
- Each contributes their own set of observed cells.

AI tracking multiple targets
- Multiple player-controlled characters
- Targets share OMaps
- Leads to “gameable” behavior, but that’s life.
4) Rendering
Rendering
Rendering is Tricky

Try to maximize player information
  - Want intense glows in likeliest locations
  - Want enough glow to know which places are possible
  - Don’t want to oversaturate
Rendering

\[ P(\text{cell}) \]

\[ \text{log}(P(\text{cell})) \]
$$\sqrt{\min\left(\frac{2 \times p_{cell}}{p^*}, 1\right)}$$

Inflection at ~$10^{-30}$

Cutoff at $10^{-20}$
$[E[\log(p)], \log(p^*)] \rightarrow [0, 1]$ 

$f(p) = \frac{\log(p) - E[\log(P)]}{\log(p^*) - E[\log(P)]}$
1 More Rendering Trick

Shimmer on $N^*$
- (ideally, would shimmer on $N_0^*$, $N_1^*$, $N_2^*$ ...)

[Diagram with a yellow arrow pointing to a specific area]
CONCLUSIONS
The Cut the Rope of AI

3eC is just like a physics game

- no level-specific scripting
- simulated elements “do their thing”
- Analog puzzles
- challenge is embedded in the structure of the environment
One Thing I Didn’t Talk About

Level Design!

(Level design turned out to take WAY longer than expected.)
Open Questions

- What is the best way to render?
- How much of this could be made to run directly on the GPU?
  - How long will this be a relevant question?

The REAL question
  - Will it make any money?
Thanks!

DAMIÁN ISLA

NAIMAD@MOONSHOTGAMES.COM

@DAMIAN_ISLA

HTTP://NAIMADGAMES.COM