Lecture 22
Character AI: Sensing & Perception
Take Away for Today

- Sensing as the primary bottleneck
  - Why is sensing so problematic?
  - What types of things can we do to improve it?

- Optimized sense computation
  - Can we improve sense computation performance?
  - Can we share sensing between NPCs?

- Sense event matching
  - What are events and how are they represented?
  - What is the advantage of an event system?
Review: Sense-Think-Act

- **Sense:**
  - Perceive the world
  - Reading the game state
  - **Example:** enemy near?

- **Think:**
  - Choose an action
  - Often merged with sense
  - **Example:** fight or flee

- **Act:**
  - Update the state
  - Simple and fast
  - **Example:** reduce health
Recall: Sensing Performance

- Sensing may be slow!
  - Consider *all* objects

- Example: morale
  - $n$ knights, $n$ skeletons
  - Knights fear skeletons
  - Proportional to # seen

- Count skeletons in view
  - $O(n)$ to count skeletons
  - $O(n^2)$ for all units
Recall: Sensing Performance

- Sensing may be slow!
  - Consider *all* objects

- Example: morale
  - $n$ knights, $n$ skeletons
  - Knights fear skeletons
  - Proportional to # seen
  - Count skeletons in view
    - $O(n)$ to count skeletons
    - $O(n^2)$ for all units

How Do We Make it Faster?

<table>
<thead>
<tr>
<th>Time per tick</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 unit</td>
</tr>
<tr>
<td>2 units</td>
</tr>
<tr>
<td>3 units</td>
</tr>
</tbody>
</table>
Aggregation

- Idea taken from databases
  - Unordered set of information
  - Combine into single value
  - Used in statistical analysis
  - **Examples**: sum, avg, mode

- **Decomposable Aggregates**
  - Split the set up into subsets
  - Aggregate on each subset
  - Combine values from subsets
  - Only for some aggregates

\[
\begin{align*}
6 & \quad \text{avg} = 8 \quad (16,2) \\
10 & \quad \text{avg} = 6 \quad (12,2) \\
5 & \quad \text{avg} = 7 \quad (28,4) \\
7 & \quad \text{avg} = 7 \quad (28,4) \\
9 & \quad \text{avg} = 7 \quad (56,8) \\
9 & \quad \text{avg} = 7 \quad (56,8) \\
7 & \quad \text{avg} = 6 \quad (12,2) \\
3 & \quad \text{avg} = 6 \quad (12,2)
\end{align*}
\]
**Aggregation**

- Idea taken from databases
- Unordered set of information
- Combine into single value
- Used in statistical analysis
- **Examples**: sum, avg, mode

**Decomposable Aggregates**

- Split the set into subsets
- Allows for fast parallel computation
- Compute values from subsets
- Only for some aggregates

![Diagram showing examples of aggregation calculations](image)
AI and Aggregation Trees

- Number of Allies
- Strength of Allies
- Number of Enemies
- Strength of Enemies

- Allied Strength
- Enemy Strength

- Threat Ratio
- Proximity to Base
- Urgency

- My Health
- Proximity to Leader

- My Morale
- Retreat %

Slide courtesy of Dave Mark
AI and Aggregation Trees

Number of Allies | Strength of Allies | Number of Enemies | Strength of Enemies

Allied Strength | Enemy Strength

Threat Ratio | Proximity to Base

Urgency | Retreat%

Do all “sides” at once using key-value pairs

Map-Reduce Pipeline

My Health | Proximity to Leader

My Morale

Sensing & Perception
Influence Maps: Pathfinding and AI

Slide courtesy of Dave Mark
Implementing Influence Maps

- Use the pathfinding grid
  - Track movement in square
  - Track if friend or foe

- Keep count as a queue
  - Count is sum of queue
  - Allows us to “time out”
  - Otherwise, marked forever

- Use queue as a predictor
  - Look at rate of change
  - Also valuable for AI

Sensing is at grid, not NPC
Advantages of Influence Maps

Influence data reflects changes

Slide courtesy of Dave Mark
Advantages of Influence Maps

Slide courtesy of Dave Mark

Add *a priori* assumptions
Sensing: Perception Groups

- **Vision**: limited field of view
  - Gives exact object location, information
  - Limited by obstacles and range
  - Little information (motion) at periphery

- **Sound**: omni-directional
  - Gives direction & distances
  - Requires you track the “sounds” actions make

- **Smell**: omni-directional
  - No direction or distance; *proximity* only
  - Requires you track the “smells” actions make
Sensing: Line-of-Sight
Sensing: Line-of-Sight
3D Line-of-Sight: Ray Casting
3D Line-of-Sight: Ray Casting
3D Line-of-Sight: Ray Casting

Similar to collisions!
Sense Events

- **Event**: encoded sense data
  - Tagged with sense type
  - Information self-contained
    - Object with methods
    - Class is sense type
- Sensing is **event matching**
  - Each event has a type
  - NPCs “register” for a type
  - Send NPC registered events
  - Check if event is relevant

Event: CharacterExposed

```java
boolean isSeen(NPC guard);
float distanceTo(NPC guard);
Vector moveDirection();
```
Sense Event Matching

- sound
- sight
- sound
- sound
- smell

Register events of interest

Event Handler

Game Loop
Sense Event Matching

Notify of any matching events

Event Handler

Check for any matching events

Game Loop
Sense Event Matching

**Advantage**: Sensing is Global, Not Local

Check for any matching events

Game Loop
Representing Events

Lightweight

Memory

Player

Reference to player

CharacterExposed

Heavyweight

Memory

Player

Copy of player

CharacterExposed
## Representing Events

### Lightweight

```java
class Event {
    Player ref;
    Event(Player p) {
        ref = p;
    }
}
```

- **Advantages**
  - Fast to create event
  - No additional memory

### Heavyweight

```java
class Event {
    Player ref;
    Event(Player p) {
        ref = p.copy();
    }
}
```

- **Advantages**
  - Stores past events
  - Can be *communicated*
Simple Pub/Sub Architecture

- NPC Hashtable
  - Event types are keys
  - Values are lists of NPCs
  - Say NPC subscribes to \( e \)

- Each update cycle…
  - Generate all of the events
  - Get NPCs for each event
  - Send those events to NPC
  - Process NPCs normally
Simple Pub/Sub Architecture

NPC Behavior

Pub/Sub System

tag1

tag2

tag1, tag3

tag4

Game Loop

Sensing & Perception
Spatial Optimizations

- Restrict to nearby NPCs
  - Have detection range
  - Limits events sensed
  - Easy to combine with event matching system
- Works in both directions
  - **Nimbus**: “can see” radius
  - **Aura**: “can be seen” radius
  - **Area of interest** management
Case Study: *Thief* Series
Stealth tip: Use WALK to move slowly and very quietly. Use CREEP to move even more slowly and be completely silent.
Line-of-Sight in *Thief*

Motion Detection

Peripheral Vision

Short Distance

Focused View

Long Distance
Sounds in *Thief*

- “Easier” than vision
  - Primarily distance-based
  - Decays probabilistically
  - Tag with level of interest

- Sounds can be blocked
  - Not same as line-of-sight
  - Use alternate level map
  - Or tag your visible map

- Not physically realistic
  - Echoes? Reflections
Sounds in *Thief*

- Sounds are general purpose
  - Resuable framework
  - Code is lightweight
  - Encodes other senses

- **Example**: Smell
  - Treated as “pseudo-sound”
  - Generate like any sound

- Again, ignores other factors
  - Wind direction
  - Masking smells
Representing Events

### Lightweight

```java
class Event {
    Player ref;
    Event(Player p) {
        ref = p;
    }
}
```

- **Advantages**
  - Fast to create event
  - No additional memory

### Heavyweight

```java
class Event {
    Player ref;
    Event(Player p) {
        ref = p.copy();
    }
}
```

- **Advantages**
  - Stores past events
  - Can be *communicated*
Communicating Senses

First Hand
LOS
Sight & Sound

Second Hand
Sight & Sound

?
Alertness: Active Senses

High Alert

First Hand
LOS
Sight & Sound

Second Hand
Sight & Sound

Medium Alert

First Hand
LOS
Sight & Sound

High Alert

First Hand
Sound

Sensing & Perception
Thief: Sense Events and Aggregation

- Position
- Lighting
- Movement
- Exposure
- Viewcone Selector
- Visibility
- Viewcone
- Look
- Listen
- Sense Pulse Receiver
- Ramp Up
- Delay
- Cool-down Capacitor
- Sense Links
- Game Mechanics and Configuration
- Sound System
- Sound Queue
- Non-specific Spatial Events
- Inter-Agent Communication
- Inter-Agent Observation
Summary

- Sensing is the most expensive part of AI
  - Each character “looks” at every object in game
  - Often leads to $O(n^2)$ behavior (bad!)

- Can **optimize** sense gathering
  - Aggregation is amenable to parallelization
  - Can piggyback some data onto pathfinding

- Event matching **inverts** the sensing problem
  - Creation of sense makes a data event
  - Forward event to “relevant” NPCs