Lecture 20

Optimizing Behavior
Stealth tip: Use WALK to move slowly and very quietly. Use CREEP to move even more slowly and be completely silent.
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

Time per tick:
- 3 units
- 2 units
- 1 unit
Recall: Sensing Performance

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

- Example: morale
  - $n$ knights, $n$ 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>
<th>1 unit</th>
<th>2 units</th>
<th>3 units</th>
</tr>
</thead>
</table>

Optimizing Behavior
Example: Collision Detection

Naively $O(n^2)$

for each object $x$:

for each object $y$:

if $x$ not $y$ and $x$, $y$ collide:

resolve collision of $x$, $y$

Checks objects obviously far apart from each other
**Example: Collision Detection**

**Lab Optimization**

for each object x:

put x into cell slot

for each cell location:

for each object x:

for each object y:

if x \(!=\) y and x, y collide:

resolve collision
Similar Ideas Exist in AI

- **Area of Interest**
  - Limit the sensing range
  - Only “see” what in range
  - Used in targeting, stealth

- **Works in both directions**
  - **Nimbus**: “can see” radius
  - **Aura**: “can be seen” radius

- Can use cell optimization
Similar Ideas Exist in AI

- **Area of Interest**
  - Limit the sensing range
  - Only “see” what in range
  - Used in targeting, stealth

- Works in both directions
  - **Nimbus**: “can see” radius
  - **Aura**: “can be seen” radius

- Can use cell optimization
Area of Interest Management *Thief*

- Long Distance
- Focused View
- Short Distance
- Peripheral Vision
- Motion Detection
Problem with this Idea

Cell-Based AI

for each entity x:
    put x into cell slot

for each cell location:
    for each entity x:
        for each entity y:
            if x can see y:
                add y to sense of x

Optimizing Behavior
Solution: Event Driven AI

Finite State Machines

Decision Trees

Can support arbitrary (boolean) functions here

state 2

state 1

test

t

f

action

test

Optimizing Behavior
Solution: Event Driven AI

Finite State Machines

Decision Trees

But we only want simple tests!

Event: Precomputed result before AI thinking starts
Event: Encoded Sense Data

- **Sight Event**
  - Type of entity seen
  - *Location* of entity seen

- **Sound Event**
  - Type of sound heard
  - *Direction* of sound heard

- **Smell Event**
  - Type of smell perceived
  - *Proximity* of the smell
Sense-Think-Act Revisited

Sense

Think

Act

Event Processor → Behavior Controller → Gameplay Controller

Events → Choices → Outcomes

Optimizing Behavior
Example: Line-of-Sight

- Use **Box2D** for sensing
  - Method `rayCast` in `World`
  - Provide a `RayCastCallback`

- Think inside the **callback**
  - *Parameters* are sense events
  - Use this to choose an action

- Act in **main update** method
  - Do after all physics done
  - Ensures all thinking done
Communicating Sense Events
Communicating Sense Events

First Hand
LOS
Sight & Sound

Optimizing Behavior
Communicating Sense Events

First Hand
LOS
Sight & Sound

First Hand
LOS
Sight & Sound
Communicating Sense Events

First Hand
LOS
Sight & Sound

Second Hand
Sight & Sound

First Hand
LOS
Sight & Sound
Communicating Sense Events

First Hand
LOS
Sight & Sound

Second Hand
Sight & Sound

First Hand
LOS
Sight & Sound

Optimizing Behavior
Event Communication in LibGDX

**MessageDispatcher**

- Send with `dispatchMessage`
  - `delay` (0 if immediate)
  - `sender` (can be null)
  - `target` (null for subscribers)
  - `type` (user defined int code)
  - `data` (object, like Box2D)

- Subscribe with `addListener`
  - NPC to receive message
  - Type (int) to subscribe to

**Telegram**

- Stores the event message
  - Entries of `dispatchMessage`
  - Except for the `delay` value
  - Preaggregated sense in `data`

- Received by **Telegraph**
  - Interface for the receiver
  - Implemented by the NPC
  - One method: `handleMessage`
Event Communication in LibGDX

MessageDispatcher

- Send with `dispatchMessage`
  - `delay` (0 if immediate)
  - `sender` (can be null)
  - `target` (null for subscribers)
  - `type` (user defined int code)
  - `data` (object, like Box2D)

- Subscribe with `addListener`
  - NPC to receive message
  - Type (int) to subscribe to

Telegram

- Stores the event message
- Entries of `dispatchMessage`
- Except for the `delay` value
- Preaggregated sense in `data`

- Received by `Telegraph`
- Interface for the receiver
- Implemented by the NPC
- One method: `handleMessage`
Separation Allows Many Optimizations

Sense  Think  Act

Event Processor  Behavior Controller  Gameplay Controller

Events  Choices  Outcomes

Optimizing Behavior
Compression: Aggregation Trees

- Number of Allies
- Strength of Allies
- Number of Enemies
- Strength of Enemies
- Proximity to Base
- Proximity to Leader
- My Health

Slide courtesy of Dave Mark
Compression: Aggregation Trees

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

Allied Strength | Enemy Strength

My Health | Proximity to Leader

Proximity to Base

Slide courtesy of Dave Mark

Optimizing Behavior
Compression: Aggregation Trees

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

Allied Strength → Enemy Strength → Threat Ratio

- My Health
- Proximity to Leader
- Proximity to Base
Compression: 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

Optimizing Behavior
Compression: Aggregation Trees

Slide courtesy of Dave Mark

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

Optimizing Behavior
Compression: Aggregation Trees

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

Allied Strength
Enemy Strength

Threat Ratio
Proximity to Base

Urgency

My Health
My Morale
Proximity to Leader

Retreat %

Optimizing Behavior
Compression: Aggregation Trees

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

Allied Strength | Enemy Strength

Threat Ratio

Computable independent of the NPC

My Health | Proximity to Leader

Urgency | Proximity to Base

My Morale

Retreat %

Slide courtesy of Dave Mark

Optimizing Behavior
Delegation: Tactical Managers

- “Invisible NPC”
  - Assigned to NPC Group
  - Both *senses* and *thinks*
  - Sends *commands* as events

- Applications
  - Protecting special units
  - Flanking
  - Covering fire
  - Leapfrogging advance
Protecting Special Units

Slide courtesy of Dave Mark

Optimizing Behavior
Protecting Special Units

Slide courtesy of Dave Mark
Protecting Special Units

Slide courtesy of Dave Mark
Protecting Special Units

Slide courtesy of Dave Mark

Optimizing Behavior
Protecting Special Units

Slide courtesy of Dave Mark

Optimizing Behavior
Protecting Special Units

Flanking!!!

Slide courtesy of Dave Mark
Protecting Special Units

Flanking!!!

Slide courtesy of Dave Mark
Protecting Special Units

Flanking!!!
Inversion: Influence Maps

Send events to grid to make heat map
Inversion: Influence Maps

NPC reacts to heat map

Slide courtesy of Dave Mark
Inversion: Influence Maps

Slide courtesy of Dave Mark
Inversion: Influence Maps

Slide courtesy of Dave Mark
Resource for Sense Optimization

Optimizing Behavior
A Final Observation

for each entity x:
    for each entity y:
        if x senses y:
            output event

Sensing is a database table join
These are all DB Optimizations

Selection Pushing

Aggregation Pushing

<table>
<thead>
<tr>
<th>Number of Allies</th>
<th>Strength of Allies</th>
<th>Number of Enemies</th>
<th>Strength of Enemies</th>
</tr>
</thead>
</table>

Allied Strength

Enemy Strength

Threat Ratio
These are all DB Optimizations

Data Normalization

NPC
NPC
NPC
NPC

Tactical Manager

Query Rewriting

Optimizing Behavior
And This is Where it All Began

- **Scaling Games to Epic Proportions (SIGMOD 2007)**
  - Allow designers to write code naively as $O(n^2)$ loop
  - Use DB technology to optimize processing

- Requires that **behaviors $\ll$ NPCs**
  - NPCs have different state, but use similar scripts
  - Each NPC is a tuple in database query

- **Challenge**: Making the language user-friendly
  - Requires major restrictions to language
  - Similar issue with Microsoft LINQ