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

Optimizing Behavior
Recall: Sensing Performance

- Sensing may be slow!
  - Consider all objects

- Example: morale
  - $n$ knights, $n$ skeletons
  - Knights fear skeletons
  - Proportional to the number of skeletons seen

- Count skeletons in view
  - $O(n)$ to count skeletons
  - $O(n^2)$ for all units
Example: Collision Detection

Naively $O(n^2)$

for each object $x$:

    for each object $y$:

        if $x \text{ not } y$ and $x, y$ collide:

            resolve collision of $x, y$

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

**Lab Optimization**

```plaintext
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
```

Optimizing Behavior
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*

- **Motion Detection**
- **Peripheral Vision**
- **Short Distance**
- **Long Distance**
- **Focused View**

Optimizing Behavior
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
Solution: Event Driven AI

Finite State Machines

- Event: Precomputed result before AI thinking starts

But we only want simple tests!

Decision Trees

Optimizing Behavior
**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
- Event Processor
  - Events

Think
- Behavior Controller
  - Choices

Act
- Gameplay Controller
  - 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
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

Optimizing Behavior
Communicating Sense Events

First Hand
LOS
Sight & Sound

Second Hand
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 `Telegram`
  - Interface for the receiver
  - Implemented by the NPC
  - One method: `handleMessage`
Separation Allows Many Optimizations

Sense

Event Processor

Think

Behavior Controller

Act

Gameplay Controller

Events

Choices

Outcomes

Sense Think Act

Optimizing Behavior
## Compression: Aggregation Trees

<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>
<tbody>
<tr>
<td>Proximity to Leader</td>
<td>Proximity to Base</td>
<td>My Health</td>
<td></td>
</tr>
</tbody>
</table>

Slide courtesy of Dave Mark
Compression: Aggregation Trees

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

Allied Strength
Enemy Strength

My Health
Proximity to Leader

Proximity to Base

Slide courtesy of Dave Mark
Compression: Aggregation Trees

Number of Allies ➔ Strength of Allies ➔ Allied Strength

Number of Enemies ➔ Strength of Enemies ➔ Enemy Strength

Allied Strength ➔ Threat Ratio

Enemy Strength ➔ Proximity to Base

My Health

Proximity to Leader

Slide courtesy of Dave Mark
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

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

- Allied Strength
- Enemy Strength

- Threat Ratio
- Proximity to Leader

- Urgency
- Proximity to Base

- My Health
- My Morale

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 | Proximity to Base

Urgency | My Health | Proximity to Leader

My Morale | Retreat %

Slide courtesy of Dave Mark
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

Optimizing Behavior
Protecting Special Units

Slide courtesy of Dave Mark

Optimizing Behavior
Protecting Special Units

Slide courtesy of Dave Mark

Optimizing Behavior
Protecting Special Units

Flanking!!!
Protecting Special Units

Flanking!!!
Protecting Special Units

Flanking!!!

Slide courtesy of Dave Mark
Protecting Special Units

Flanking!!!

Slide courtesy of Dave Mark
Inversion: Influence Maps

Slide courtesy of Dave Mark
Inversion: Influence Maps

Send events to grid to make heat map

Slide courtesy of Dave Mark
Inversion: Influence Maps

NPC reacts to heat map

Slide courtesy of Dave Mark
Inversion: Influence Maps

Slide courtesy of Dave Mark
Inversion: Influence Maps
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>
<tbody>
<tr>
<td>Threat Ratio</td>
<td>Allied Strength</td>
<td>Enemy Strength</td>
<td></td>
</tr>
</tbody>
</table>
These are all DB Optimizations

Data Normalization

Query Rewriting

NPC

NPC

NPC

NPC

Tactical Manager
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