Minimal Memory Abstractions
(As implemented for BioWare Corp®)

Nathan Sturtevant

University of Alberta
GAMES Group
February 22, 2007

Talk Overview

• Part I: Building Abstractions
• Minimizing memory requirements
• Performances measures
• Part II: BioWare Corp®
• Implementation
• Experience
**Background**

- State-space abstractions have commonly been used to speed search
- Pattern Databases for heuristics
- Graph abstractions for pathfinding
  - PRA*, HPA*, etc

**Motivation**

- Games have tight memory budgets
- ~4MB total memory
- 1024x1024 or larger maps
  - 1MB per byte per grid cell
- Can we use build an abstraction which minimizes memory usage?
**Assumptions**

- Grid world
  - No true 3-d movement
- Cells can be blocked/free/weighted
- May be height difference between cells
- Units can move across real-valued space

**Sectors / Regions**

- Divide world into large sectors
  - Fixed size
  - Index implicitly
- Divide sectors into regions
  - Regions entirely connected
  - Regions have a center point
Sectors / Regions

• Divide world into large sectors
• Fixed size
• Index implicitly
• Divide sectors into regions
• Regions entirely connected
• Regions have a center point
Edges

• Look at borders of regions to determine edges
Edges

- Look at borders of regions to determine edges
Edges

- Look at borders of regions to determine edges
**Abstract Graph**

- Original Map:
  - 32x32 = 1024 cells
- Abstract Graph:
  - 9 nodes
  - 10 edges

**Memory Usage**

- 32 bits per sector
- Can use less
- 16 bits per region
- 8 bits per edge
- 3 bits - direction
- 5 bits - region
- Skip some regions
- Edges duplicated

<table>
<thead>
<tr>
<th>Sector Data</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td># Regions</td>
<td>2</td>
</tr>
<tr>
<td>Memory Address</td>
<td>0</td>
</tr>
<tr>
<td>unused</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Region Data</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>center</td>
<td>196</td>
</tr>
<tr>
<td># edges</td>
<td>3</td>
</tr>
<tr>
<td>center</td>
<td>142</td>
</tr>
<tr>
<td># edges</td>
<td>4</td>
</tr>
<tr>
<td>variable-sized edge storage</td>
<td>left:3 upleft:1 up:1 up:2 up:1</td>
</tr>
</tbody>
</table>
Find Sector/Region

• Begin with $x/y$ location in real world
• Must find sector/region
• If sector only has 1 region, done
• Otherwise do BFS to find region center
  • Can do reverse A* search from region centers
• Avoids pointers!

Usage (1)

• Find sector/region for starts and goals
• Use A* to find a complete abstract path
• Now we must use the abstract path to guide the search for an actual path
**Usage (2)**

- Many different methods for using abstract path
- Simplest method:
  - Find path from start to first region
  - Compute path to successive regions
  - Find path from last region to goal

---

**Usage Example**

- Find abstract parents
- Find abstract path
- Find real path
Usage Example

• Find abstract parents
• Find abstract path
• Find real path
Usage Example

• Find abstract parents
• Find abstract path
• Find real path
Usage Example

- Find abstract parents
- Find abstract path
- Find real path

Total Pathfinding Cost

- Abstract planning cost + Refinement
- Refinement cost depends on obstacles and total path length
- Abstract planning cost depends on sector size
- For fixed path length, the total work should depend only on sector size
Optimizing Region Centers

• How to determine the region centers?
• Some locations are much better than others
Optimizing Region Centers

- Consider each region independently
- Measure the A* cost to path between region and all neighbors
- Choose the region center which minimizes the maximum cost
Pathfinding Optimization

- Refinement at start/goal can be inefficient
- Trimming helps
- Skip to next node at start/goal
Pathfinding Optimization

- Refinement at start/goal can be inefficient
- Trimming helps
- Skip to next node at start/goal
Pathfinding Optimization

- Refinement at start/goal can be inefficient
- Trimming helps
- Skip to next node at start/goal

Experimental Results

- 93,000 paths over 120 maps
- Maps scaled to 512x512
- Paths in 128 buckets length 1…512
- Measure:
  - Total cost
  - Incremental cost
Memory Usage

- How does the memory usage scale with sector size?
- How much memory can be saved with simple compression?
- Don’t store “default” regions
- 1 region, 8 neighbors

Maps Size 512x512

![Graph showing memory usage over sector size](image-url)
Dynamic Region Centers

- Is there a gain to dynamically optimizing region centers?
- Measure 95% work done in one-step path refinement

Dynamic Centers

Dynamic v. Static Centers (1-Step Planning)

**Nodes Expanding**

- Static (95th percentile)
- Dynamic (95th percentile)
• Paths will not be optimal
• Special cases for start/goal help a lot
• Smoothing will be applied as a post-processing step (not measured)
**Total Work**

- Sum of work needed:
  - Find parents
  - Find abstract path
  - Refine low-level path
  - Compare to A*

---

![Graph showing Total Work](image)
IMPLEMENTATION

• 2 weeks:
  • Implement abstraction
  • Implement pathfinding
  • Initial testing
  • Met pathfinding requirements
Observations

• Cannot be an expert in one thing
• Get it “good enough”
• Both more and less rigorous testing than expected
• Great people

Future

• Continuing work:
  • Smoothing
  • Placeables
More Info

- http://dragonage.bioware.com/
- http://www.1up.com/do/gameOverview?cId=2019479
- http://www.1up.com/do/previewPage?cId=3155733

Thanks