Lecture 21

Pathfinding
Take Away for Today

- What are the primary goals for pathfinding?
- Identify advantages/disadvantages of A*
  - In what situations does A* fail (or look bad)?
  - What can we do to fix these problems?
- Why combine steering and A*?
  - Is this combination always appropriate?
- What do commercial games use?
Pathfinding

- You are given
  - Starting location $A$
  - Goal location $B$
- Want **valid** path $A$ to $B$
  - Avoid “impassible” terrain
  - Eschew hidden knowledge
- Want **natural** path $A$ to $B$
  - Reasonably short path
  - Avoid unnecessary turns
  - Avoid threats in the way
Abstraction: Grid & Graph

- Break world into grid
  - Roughly size of NPCs
  - Terrain is all-or-nothing
  - Majority terrain of square
  - Terrain covering “center”
- Gives us a weighted graph
  - Nodes are grid centers
  - Each node has 8 neighbors
  - Weight = distance/terrain
- Search for shortest path

- Real distance not required
  - 14:10 ratio for diagonals
  - Allows us to use integers
Breadth-First Search (Lab 2)

Intuition

- **Search maintains**
  - Current node, initially start
  - List of nodes to visit

- **Basic Steps**
  - Have we reached the goal?
  - Add neighbors to end of list
  - Work from first node in list
  - Process “first-in first-out”

Algorithm

\[
n = \text{start}; \ L = \{ \};\\
\text{while} \ (n \ \text{not goal}) \ \{\\
\quad \text{add } n \ \text{to visited};\\
\quad N(n) = \text{unvisited neighbors}\\
\quad \text{foreach } (m \in N(n)) \ \{\\
\quad\quad \text{add } m \ \text{to end of } L;\\
\quad\}\\\
\quad n = \text{removeFirst}(L);\\
\}\\\
\text{return} \ \text{path to goal};\]
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Breadth-First is Slow!

- Searches too many grids
  - Grids far away from goal
  - Works “radially outward”
- What is the problem?
  - Using graph algorithms
  - No spatial knowledge
- Idea: Spatial+Graph
  - Measure distance normally
  - Pick neighbor close to goal
Heuristic Search

**Intuition**

- Modified version of BFS
  - Have a list of candidates
  - Always pick *best* candidate
- Need $f$, **heuristic** function
  - Used to pick next step
  - Avoids stupid choices
- Regularly **update** $f$
  - Recompute on all neighbors
  - Reassign value if smaller

**Algorithm**

```
n = start; L = { }; 
while (n not goal) {
    add n to visited; 
    N(n) = unvisited neighbors
    foreach (m ∈ N(n)) {
        add m to L; 
        update f(m);
    }
    pick n ∈ L with f least;
}
return path to goal;
```
Heuristic Search

Intuition

- Modified version of BFS
  - Have a list of candidates
  - Always pick best
- Need $f$, heuristic
  - Used to pick next step
  - Avoids stupid choices
- Regularly update $f$
  - Recompute on all neighbors
  - Reassign value if smaller

Algorithm

```python
n = start; L = {};
while (n not goal) {
    add n to visited;
    N(n) = unvisited neighbors
    foreach (m ∈ N(n)) {
        add m to L;
        update f(m);
    }
    pick n ∈ L with f least;
}
return path to goal;
```

Examples:

- **Dijkstra’s Algorithm**
  $f = \text{dist. from source}$
- **Greedy Algorithm**
  $f = \text{estimated dist. to goal}$
A* Algorithm

- **Idea**: Dijkstra + Greedy
  - $g$: distance on **current path**
    - An “exact calculation”
    - Distance along graph
  - $h$: estimated dist. to **goal**
    - Spatial distance
    - Ignores all obstacles
  - Final heuristic $f = g + h$

- Many variations for $h$
  - Regular distance
  - “Manhattan Metric”

Manhattan distance $= 30 + 20 = 50$
Pathfinding: A* Algorithm

A* Algorithm

Pathfinding
Pathfinding: A* Algorithm

A* Algorithm

f: 74  g:14  h:60
f: 60  g:10  h:50
f: 40  g:10  h:30
f: 54  g:14  h:40

X

X

X

X

A

B

Pathfinding
Pathfinding: A* Algorithm

A

f: 74

g: 14

h: 60

B

f: 60

g: 24

h: 50

X

f: 54

g: 20

h: 30

X

f: 60

g: 10

h: 50

X

f: 74

g: 14

h: 60

X

f: 74

g: 24

h: 50
Pathfinding: A* Algorithm
Pathfinding: A* Algorithm
Pathfinding: A* Algorithm
Pathfinding: A* Algorithm
Pathfinding: A* Algorithm

```
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th></th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>f: 88</td>
<td>g:28</td>
<td>h:60</td>
<td></td>
</tr>
<tr>
<td>f: 74</td>
<td>g:24</td>
<td>h:50</td>
<td></td>
</tr>
<tr>
<td>f: 74</td>
<td>g:14</td>
<td>h:60</td>
<td></td>
</tr>
<tr>
<td>f: 60</td>
<td>g:10</td>
<td>h:50</td>
<td></td>
</tr>
<tr>
<td>f: 60</td>
<td>g:10</td>
<td>h:50</td>
<td></td>
</tr>
<tr>
<td>f: 54</td>
<td>g:14</td>
<td>h:40</td>
<td></td>
</tr>
<tr>
<td>f: 40</td>
<td>g:10</td>
<td>h:30</td>
<td></td>
</tr>
<tr>
<td>f: 74</td>
<td>g:14</td>
<td>h:40</td>
<td></td>
</tr>
<tr>
<td>f: 94</td>
<td>g:24</td>
<td>h:70</td>
<td></td>
</tr>
<tr>
<td>f: 80</td>
<td>g:20</td>
<td>h:60</td>
<td></td>
</tr>
<tr>
<td>f: 74</td>
<td>g:24</td>
<td>h:50</td>
<td></td>
</tr>
</tbody>
</table>
```
Pathfinding: A* Algorithm
Pathfinding: A* Algorithm

\[
\begin{array}{|c|c|c|}
\hline
f: 94 & f: 80 & f: 74 \\
g: 24 & g: 20 & g: 24 \\
h: 70 & h: 60 & h: 70 \\
\hline
f: 74 & f: 60 & f: 54 \\
g: 24 & g: 10 & g: 10 \\
h: 70 & h: 60 & h: 50 \\
\hline
f: 80 & f: 60 & f: 74 \\
g: 20 & g: 10 & g: 14 \\
h: 60 & h: 50 & h: 50 \\
\hline
f: 94 & f: 80 & f: 74 \\
g: 24 & g: 10 & g: 24 \\
h: 70 & h: 60 & h: 50 \\
\hline
\end{array}
\]
Pathfinding: A* Algorithm

In case of tie, use most recently added
In case of tie, use most recently added
Pathfinding: A* Algorithm

In case of tie, use most recently added
Pathfinding: A* Algorithm

In case of tie, use most recently added
Pathfinding: A* Algorithm
<table>
<thead>
<tr>
<th>IndexedGraph</th>
<th>IndexedAStarPathFinder</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Array of IndexedNode objs</td>
<td></td>
</tr>
<tr>
<td>• Can implement as an array</td>
<td></td>
</tr>
<tr>
<td>• Hard part is IndexedNode</td>
<td></td>
</tr>
<tr>
<td>• Each IndexNode must store</td>
<td></td>
</tr>
<tr>
<td>• Index into the graph array</td>
<td></td>
</tr>
<tr>
<td>• Array of Connection objs</td>
<td></td>
</tr>
<tr>
<td>• Each Connection must have</td>
<td></td>
</tr>
<tr>
<td>• The start and end node</td>
<td></td>
</tr>
<tr>
<td>• The cost to traverse edge</td>
<td></td>
</tr>
<tr>
<td>• Construct with a graph</td>
<td></td>
</tr>
<tr>
<td>• Must use with IndexedGraph</td>
<td></td>
</tr>
<tr>
<td>• Graph reference immutable</td>
<td></td>
</tr>
<tr>
<td>• To search for path, give</td>
<td></td>
</tr>
<tr>
<td>• The start and end nodes</td>
<td></td>
</tr>
<tr>
<td>• Heuristic implementation</td>
<td></td>
</tr>
<tr>
<td>• GraphPath for the answer</td>
<td></td>
</tr>
<tr>
<td>• Can give search a timeout</td>
<td></td>
</tr>
<tr>
<td>• Abort if it takes too long</td>
<td></td>
</tr>
</tbody>
</table>
## LibGDX Support

### IndexedGraph

- Array of **IndexedNode** objs
  - Can implement as an array
  - Hard part is IndexedNode
- Each **IndexedNode** must have
  - Index into the graph
  - Array of Connection objs
- Each **Connection** must have
  - The start and end node
  - The cost to traverse edge

### IndexedAStarPathFinder

- Construct with a graph
  - Must use with **IndexedGraph**
  - Graph reference immutable
- Search for path, give
  - The start and end nodes
  - Heuristic implementation
  - **GraphPath** for the answer
- Can give search a **timeout**
  - Abort if it takes too long

Everything in blue is an interface.
LibGDX Support

IndexedGraph

- Array of IndexedNode objs
- Can implement as an array
- Hard part is...
- Each IndexedNode
  - Index into the...
  - Array of Connection objs
- Each Connection must have
  - The start and end node
  - The cost to traverse edge

IndexedAStarPathFinder

- Construct with a graph
- Must use with IndexedGraph
- Graph reference immutable
- To search for path, give
  - start and end nodes
  - Heuristic implementation
  - GraphPath for the answer
- Can give search a timeout
- Abort if it takes too long

Only these have implementations
Issues with A*: Stair Stepping
Stair Stepping

- What is the problem?
  - Move one square at a time
  - All turns are at 45°

- **Idea**: Path smoothing
  - Path is a series of waypoints
  - Straight line between points
  - Remove unnecessary points

- Can combine with A*
  - Get *degenerative* solution
  - Remove to get waypoints

- Choose first $q$ after $p$ where
  - Line $pq$ is valid
  - Point $q$ has successor $s$
  - Line $ps$ is not valid
Path Smoothing
Path Smoothing

A

B
Path Smoothing
Path Smoothing

A

B
Path Smoothing
Path Smoothing

Limited LibGDX support via SmoothableGraphPath interface
Turning

- **Realistic** turns
  - Smooth paths into line segments
  - Round corners for realistic movement

- **Restricted** turns
  - Limit turns to angles drawn by artist
  - 16 angles standard for 2D top-down

- See online reading for today
  - Pinter, “Toward More Realistic Pathfinding”
  - Requires free registration to Gamasutra
Multiple NPC Sizes

- Grid to largest NPC?
  - Bad for small units
  - Unnecessary blocking
- Grid to smallest NPC!
  - Multiple squares for larger
  - Center fits on grid square
- Pathfinding larger NPCs
  - A* for center-to-center
  - Size to check blocking
  - May alter the path
Multiple NPC Sizes

- Grid to largest NPC?
  - Bad for small units
  - Unnecessary blocking
- Grid to smallest NPC!
  - Multiple squares for larger
  - Center fits on grid square
- Pathfinding larger NPCs
  - A* for center-to-center
  - Size to check blocking
  - May alter the path
Multiple NPC Sizes

- Grid to largest NPC?
  - Bad for small units
  - Unnecessary blocking

- Grid to smallest NPC!
  - Multiple squares for larger
  - Center fits on grid square

- Pathfinding larger NPCs
  - A* for center-to-center
  - Size to check blocking
  - May alter the path
Multiple NPC Sizes

- Grid to largest NPC?
  - Bad for small units
  - Unnecessary blocking

- Grid to smallest NPC!
  - Multiple squares for larger
  - Center fits on grid square

- Pathfinding larger NPCs
  - A* for center-to-center
  - Size to check blocking
  - May alter the path
Multiple NPC Sizes

- Grid to largest NPC?
  - Bad for small units
  - Unnecessary blocking

- Grid to smallest NPC!
  - Multiple squares for larger
  - Center fits on grid square

- Pathfinding larger NPCs
  - A* for center-to-center
  - Size to check blocking
  - May alter the path
Fitting NPCs on a Grid

- Assume NPC is square
  - Represents “reach”
  - Simplifies turning
- Requires “odd” sizes
  - Center must be a grid
  - Radius in full grid squares
  - What about even sizes?
- “Tabletop” solution
  - Round down when moving
  - Round up when in place
Waypoints

Express paths as a sequence of segments

A

waypoint

B
Steering

- Alternative to pathfinding
  - Uses forces to move NPCs
  - Great for **small** paths

- **Examples**
  - Artificial potential fields
  - Vortex fields
  - Custom steering behaviors

- See Craig Reynold’s page
  - See “Physics & Motion”
  - [com.badlogic.gdx.ai.steer](com.badlogic.gdx.ai.steer)
Steering and Pathfinding

- Use waypoint as “goal”
  - Attract NPC to waypoint
  - When close, next waypoint
- Great for multiple NPCs
  - Pathfind for largest NPC
  - Steering to move along path
  - Repulsion keeps NPCs apart
- **Drawbacks:**
  - Military formations are hard
  - Get stuck at bottlenecks
Dynamic Obstructions

- Others can get in way
  - Enemies guarding locale
  - Friends waiting in queue

- Correct response?
  - Compute a new path?
  - Wait to be unblocked?

- What would you do?
  - See what is blocking
  - Making an educated guess
  - Character AI solution
Pathfinding in Practice

• **Navigation Meshes**
  • Indicates walkable areas
  • 2D geometric representation
  • Connected convex shapes
  • A* graph: center-to-center

• **Making Nav Meshes**
  • Often done by level editor
  • Can be modified by hand
  • Annotate special movement
  • **Example**: jump points
Center of each Region

Corners of the Mesh

Easy Pathfinding on Meshes
Alternative: Hierarchical Pathfinding
Alternative: Hierarchical Pathfinding
Alternative: Hierarchical Pathfinding
Alternative: Hierarchical Pathfinding

Cost depends on how entered
Alternative: Hierarchical Pathfinding

Design hierarchy to minimize cost artifacts

Cost depends on how entered
## LibGDX Support

### HierarchicalGraph
- Graph with multiple levels
- Has a current active level
- Graph API matches level
- Can switch this level on fly
- Also can convert levels
  - node + level => node
  - Rules to group nodes
  - Rules to split nodes

### HierarchicalPathFinder
- Specify a pathfinder to use
  - Could be A* or otherwise
  - Will use it on each level
- The implementation
  - Finds path at highest level
  - Expands nodes to next level
  - Refines path to expansion
  - Repeats until level 0
HierarchicalGraph

- Graph with multiple levels
- Has a current active level
- Graph API matches level
- Can switch level on fly
- Also can convert levels
  - node + level => node
- Rules to group nodes
- Rules to split nodes

HierarchicalPathFinder

- Specify a pathfinder to use
- Could be A* or otherwise
- Will use it on each level
- The class
  - Finds path at highest level
  - Expands nodes to next level
  - Refines path to expansion
  - Repeats until level 0
Summary

• **A* algorithm** is primary pathfinding tool
  • Make world into a grid/navigation mesh
  • Search for a path on associated graph
  • Adjust heuristics for terrain, threats

• But there are a lot of “special tricks”
  • Tricks to make movement realistic
  • Tricks to handle coordinated movement
  • Talk to Instructor (or TAs) if need more tricks