Lecture 20

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**

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

A

B
Pathfinding: Breadth-First
Pathfinding: Breadth-First

[Diagram of a grid with a starting point A and a destination B, with paths marked by arrows and numbers indicating distances.]
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First

A

B

X

X

X

X

Pathfinding
Pathfinding: Breadth-First

A

B

The Pathfinding initiative at Cornell University

Pathfinding
Pathfinding: Breadth-First
Pathfinding: Breadth-First
Pathfinding: Breadth-First

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>24</td>
<td>20</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>14</td>
<td>10</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>14</td>
<td>10</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>24</td>
<td>20</td>
<td>24</td>
<td>34</td>
</tr>
<tr>
<td>38</td>
<td>34</td>
<td>30</td>
<td>34</td>
<td>38</td>
</tr>
</tbody>
</table>

A: Start
B: Goal

The pathfinding algorithm explores the grid in a breadth-first manner, expanding the search layer by layer from the start point A to the goal point B.
Pathfinding: Breadth-First
Pathfinding: Breadth-First
### Pathfinding: Breadth-First

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>24</td>
<td>20</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>14</td>
<td>10</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>14</td>
<td>10</td>
<td>14</td>
<td>54</td>
<td>58</td>
</tr>
<tr>
<td>28</td>
<td>24</td>
<td>20</td>
<td>24</td>
<td>34</td>
<td>44</td>
</tr>
<tr>
<td>38</td>
<td>34</td>
<td>30</td>
<td>34</td>
<td>38</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>34</td>
<td></td>
<td>38</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>34</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>54</td>
<td></td>
</tr>
</tbody>
</table>

Node A is the start point, and Node B is the target point. The green path represents the Breadth-First search algorithm.
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 $\in$ N(n)) {
    add m to L;  
    update f(m);
  }
  pick n $\in$ L with f least;
}  
return path to goal;
### Heuristic Search

#### Intuition

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

#### Algorithm

```java
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”

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

A* Algorithm

\[
\begin{align*}
\text{f: } & 74 & \text{g: } & 14 & \text{h: } & 60 \\
\text{f: } & 60 & \text{g: } & 10 & \text{h: } & 50 \\
\text{f: } & 54 & \text{g: } & 14 & \text{h: } & 40 \\
\end{align*}
\]
Pathfinding: A* Algorithm

A* Algorithm

\[ f = g + h \]

- **f**: Total cost
- **g**: Cost from start to current node
- **h**: Heuristic cost from current node to goal

Example:

- Node A: \( f = g + h \) = 74
- Node B: \( f = g + h \) = 74
- Node X: \( f = g + h \) = 54
- Node X: \( f = g + h \) = 60

Path from start to goal through intermediate nodes.

Graph showing nodes A and B with calculated \( f, g, \text{ and } h \) values.
Pathfinding: A* Algorithm

The A* algorithm is a popular pathfinding algorithm used in computer science and artificial intelligence to find the shortest path between two points in a graph. It combines the information from an informed heuristic and the cost of the path through the graph.

In the diagram, the green square represents the start point (A) and the red square represents the goal point (B). The blue area represents an obstacle. Each cell in the grid represents a possible location where the path can be directed. The values f, g, and h are calculated for each cell, where f = g + h.

- f: Total estimated cost from the start to the goal.
- g: Cost from the start to the current cell.
- h: Heuristic estimate from the current cell to the goal.

The algorithm expands the shortest path by selecting the cell with the smallest f value and updating the costs of its neighboring cells. The process continues until the goal cell is reached.
Pathfinding: A* Algorithm

A* Algorithm

The A* Algorithm is a popular pathfinding algorithm that uses a heuristic function to guide its search. It is widely used in game development and robotics.

The algorithm uses the following formula to calculate the total cost of a path:

\[ f(n) = g(n) + h(n) \]

Where:
- \( f(n) \) is the total cost of the path.
- \( g(n) \) is the cost of the path from the start node to node \( n \).
- \( h(n) \) is the heuristic estimate of the cost from node \( n \) to the goal.

The algorithm explores the graph from the start node, expanding the node with the lowest \( f(n) \) value. It uses a priority queue to keep track of the nodes to be expanded, sorted by \( f(n) \).

The A* Algorithm guarantees to find the optimal path if the heuristic function is admissible (never overestimates the true cost).
Pathfinding: A* Algorithm

A* Algorithm

\[ f(n) = g(n) + h(n) \]

Where:
- \( f(n) \) is the total cost of the path through node \( n \)
- \( g(n) \) is the cost to reach node \( n \) from the start
- \( h(n) \) is the estimated cost to reach the goal from node \( n \)

The algorithm works by expanding the node \( n \) with the smallest \( f(n) \) value. It continues this process until the goal node is reached.
Pathfinding: A* Algorithm

- Pathfinding: A* Algorithm

- A: f=74, g=14, h=60
- B: f=74, g=28, h=60

- Pathing Algorithm:
  - f = g + h
  - g = distance from start
  - h = heuristic estimate to goal

- Algorithm:
  1. Initialize: open set = initial state, closed set = empty
  2. If open set is empty, halt
  3. Select node with lowest f value
  4. If node is the goal, halt
  5. Add node to closed set
  6. For each neighbor of current node:
     - If neighbor is in closed set, skip
     - Calculate new f value
     - If neighbor is in open set, update f value if new f is lower
     - Add neighbor to open set

- Pathing Example:
  - Initial state:
  - Goal state:
  - Path found through heuristic estimation.

- A*的优点:
  - Guarantees an optimal path when heuristic is admissible.
  - Efficient, as it uses a priority queue.

- Applications:
  - Game development
  - Robotics
  - Maze solving

- Further Reading:
  - Artificial Intelligence: A Modern Approach
  - Pathfinding in Games: A* and Beyond
Pathfinding: A* Algorithm

A* Algorithm is a popular algorithm for pathfinding. It is an informed search algorithm that can use heuristics to make informed decisions about which path to follow. The algorithm uses a combination of the actual cost to reach a node (g), the estimated cost to reach the goal from a node (h), and the total cost so far (f = g + h) to determine the best path.

In the example shown, we have a grid with two goal states, A and B, and a path that connects them. The algorithm calculates the f, g, and h values for each node, and then moves towards the goal state with the lowest f value. The path found is the one with the lowest total cost.
Pathfinding: A* Algorithm

A* Algorithm

Pathfinding
Pathfinding: A* Algorithm
Pathfinding: A* Algorithm
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

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

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

```
Pathfinding
```
## LibGDX Support

### IndexedGraph
- Array of **IndexedNode** objs
  - Can implement as an array
  - Hard part is IndexedNode
- Each **IndexNode** must store
  - Index into the graph array
  - 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
  - The start and end nodes
  - **Heuristic** implementation
  - **GraphPath** for the answer
- Can give search a **timeout**
  - Abort if it takes too long
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
- To 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
IndexedGraph

- Array of IndexedNode objs
- Can implement as an array
- Hard part is IndexedNode
- Each IndexedNode must store
  - Index into the graph array
  - 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

Pathfinding
Path Smoothing
Path Smoothing
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.
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`
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
Easy Pathfinding on Meshes

Center of each Region

Corners of the Mesh
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
LibGDX Support

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 implementation
  - 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