Chapter 5.4
Artificial Intelligence: Pathfinding
Introduction

- Almost every game requires pathfinding
- Agents must be able to find their way around the game world
- Pathfinding is not a trivial problem
- The fastest and most efficient pathfinding techniques tend to consume a great deal of resources
Representing the Search Space

- Agents need to know where they can move
- Search space should represent either
  - Clear routes that can be traversed
  - Or the entire walkable surface
- Search space typically doesn’t represent:
  - Small obstacles or moving objects
- Most common search space representations:
  - Grids
  - Waypoint graphs
  - Navigation meshes
Grids

- 2D grids – intuitive world representation
  - Works well for many games including some 3D games such as *Warcraft III*

- Each cell is flagged
  - Passable or impassable

- Each object in the world can occupy one or more cells
Characteristics of Grids

- Fast look-up
- Easy access to neighboring cells
- **Complete** representation of the level
A waypoint graph specifies lines/routes that are “safe” for traversing.

Each line (or link) connects exactly two waypoints.
Characteristics of Waypoint Graphs

- Waypoint node can be connected to any number of other waypoint nodes
- Waypoint graph can easily represent arbitrary 3D levels
- Can incorporate auxiliary information
  - Such as ladders and jump pads
- Incomplete representation of the level
Navigation Meshes

- Combination of grids and waypoint graphs
- Every node of a navigation mesh represents a convex polygon (or area)
  - As opposed to a single position in a waypoint node
- Advantage of convex polygon
  - Any two points inside can be connected without crossing an edge of the polygon
- Navigation mesh can be thought of as a walkable surface
Characteristics of Navigation Meshes

- **Complete** representation of the level
- Ties pathfinding and collision detection together
- Can easily be used for 2D and 3D games
Searching for a Path

- A path is a list of cells, points, or nodes that an agent must traverse.
- A pathfinding algorithm finds a path from a start position to a goal position.
- The following pathfinding algorithms can be used on:
  - Grids
  - Waypoint graphs
  - Navigation meshes
Criteria for Evaluating Pathfinding Algorithms

- Quality of final path
- Resource consumption during search
  - CPU and memory
- Whether it is a *complete* algorithm
  - A *complete* algorithm guarantees to find a path if one exists
Random Trace

- Simple algorithm
  - Agent moves towards goal
  - If goal reached, then done
  - If obstacle
    - Trace around the obstacle clockwise or counter-clockwise (pick randomly) until free path towards goal
  - Repeat procedure until goal reached
Random Trace (continued)

How will Random Trace do on the following maps?
Random Trace Characteristics

- Not a *complete* algorithm
- Found paths are unlikely to be optimal
- Consumes very little memory
Understanding A*

- To understand A*
  - First understand Breadth-First, Best-First, and Dijkstra algorithms
- These algorithms use nodes to represent candidate paths
Understanding A*

class PlannerNode
{
    public:
        PlannerNode *m_pParent;
        int m_cellX, m_cellY;
    ...
};

- The m_pParent member is used to chain nodes sequentially together to represent a path
Understanding A*

- All of the following algorithms use two lists
  - The *open* list
  - The *closed* list
- Open list keeps track of promising nodes
- When a node is examined from open list
  - Taken off open list and checked to see whether it has reached the goal
- If it has not reached the goal
  - Used to create additional nodes
  - Then placed on the closed list
Overall Structure of the Algorithms

1. Create start point node – push onto open list
2. While open list is not empty
   A. Pop node from open list (call it currentNode)
   B. If currentNode corresponds to goal, break from step 2
   C. Create new nodes (successors nodes) for cells around currentNode and push them onto open list
   D. Put currentNode onto closed list
Breadth-First

- Finds a path from the start to the goal by examining the search space ply-by-ply
Breadth-First Characteristics

- Exhaustive search
  - Systematic, but not clever
- Consumes substantial amount of CPU and memory
- Guarantees to find paths that have fewest number of nodes in them
  - Not necessarily the shortest distance!
- *Complete* algorithm
Best-First

- Uses problem specific knowledge to speed up the search process
- Head straight for the goal
- Computes the distance of every node to the goal
  - Uses the distance (or heuristic cost) as a priority value to determine the next node that should be brought out of the open list
Best-First (continued)
Best-First (continued)

- Situation where Best-First finds a suboptimal path
Best-First Characteristics

- Heuristic search
- Uses fewer resources than Breadth-First
- Tends to find good paths
  - No guarantee to find most optimal path
- Complete algorithm
Dijkstra

- Disregards distance to goal
  - Keeps track of the cost of every path
  - No guessing

- Computes accumulated cost paid to reach a node from the start
  - Uses the cost (called the given cost) as a priority value to determine the next node that should be brought out of the open list
Dijkstra Characteristics

- Exhaustive search
- At least as resource intensive as Breadth-First
- Always finds the most optimal path
- *Complete* algorithm
A*

- Uses both heuristic cost and given cost to order the open list
  
  Final Cost = Given Cost + (Heuristic Cost * Heuristic Weight)
A* (continued)

- Avoids Best-First trap!
A* Characteristics

- Heuristic search
- On average, uses fewer resources than Dijkstra and Breadth-First
- *Admissible* heuristic guarantees it will find the most optimal path
- *Complete* algorithm
Summary

- Two key aspects of pathfinding:
  - Representing the search space
  - Searching for a path
PathPlannerApp Demo
Waypoint Graph Demo

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
<th>[6]</th>
<th>[7]</th>
<th>[8]</th>
<th>[9]</th>
<th>[10]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>X</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>[1]</td>
<td>0</td>
<td>X</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>[2]</td>
<td>0</td>
<td>1</td>
<td>X</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>[3]</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>X</td>
<td>4</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>[4]</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>X</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>[5]</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>X</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>[6]</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>X</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>[7]</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>X</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>[8]</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>[9]</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>[10]</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

--- Result of Computing ALL paths using A*