Pathfinding and mazes
Pathfinding – problem statement

• Finding the shortest route between two points

• How to represent the environment

• *Lots of algorithms*
Environment representation

• 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
Grids
Grids as Graphs
Distance metrics

Manhattan: $|P_x - Q_x| + |P_y - Q_y|$

Euclidean: $\sqrt{(P_x - Q_x)^2 + (P_y - Q_y)^2}$
Waypoint graph
Pathfinding algorithms

- Random walk
- Random trace
- Breadth first search
- Best first search
- Dijkstra’s algorithm
- A* algorithm
Evaluating graph search 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 walk

• Agent takes a random step
  – If goal reached, then done

• Repeat procedure until goal reached

• Add intelligence
  – Only step in cases where distance to goal is smaller
  – Stuck? With a certain probability, allow a step where distance to goal is greater
Random trace

• Agent moves towards goal
  – If goal reached, then done
  – If obstacle
    • Turn around the obstacle clockwise or counter-clockwise (pick randomly) until free path towards goal

• Repeat procedure until goal reached
Random walk and trace properties

- Not complete algorithms
- Found paths are unlikely to be optimal
- Consume very little memory
Random walk and trace performance
Search algorithms

- Breadth-First
- Best-First
- Dijkstra’s
- A* (combination of Best-First and Dijkstra)

- Nodes represent candidate paths
- Keep track of numerous paths simultaneously
Breath First Search

Closed List (Those visited)
A (Path A);

Open List (To be visited)
B (Path A->B)
C (Path A->C)
Breath First Search

Closed List (Those visited)
A (Path A);
B (Path A->B);

Open List (To be visited)
C (Path A->C);
E (Path A->B->E);
D (Path A->B->D);
Breath First Search

Closed List (Those visited)
A (Path A);
B (Path A->B);
C (Path A->C);

Open List (To be visited)
E (Path A->B->E);
D (Path A->B->D);
Breath First Search

Closed List (Those visited)
A (Path A);
B (Path A->B);
C (Path A->C);
E (Path A->B->E);

Open List (To be visited)
D (Path A->B->D);
Breath First Search

Closed List (Those visited)
A (Path A);
B (Path A->B);
C (Path A->C);
E (Path A->B->E);
D (Path A->B->D);

Open List (To be visited)
G (Path A->B->D->G)
F (Path A->B->D->F)
Breath First Search

Closed List (Those visited)
A (Path A);
B (Path A->B);
C (Path A->C);
E (Path A->B->E);
D (Path A->B->D);
G (Path A->B->D->G)

Open List (To be visited)
F (Path A->B->D->F)
H (Path A->B->D->G->H)
**Breath First Search**

Closed List (Those visited)
- A (Path A);
- B (Path A->B);
- C (Path A->C);
- E (Path A->B->E);
- D (Path A->B->D);
- G (Path A->B->D->G);
- F (Path A->B->D->F)

Open List (To be visited)
- H (Path A->B->D->G->H)
Breath First Search

Closed List (Those visited)
A (Path A);
B (Path A->B);
C (Path A->C);
E (Path A->B->E);
D (Path A->B->D);
G (Path A->B->D->G);
F (Path A->B->D->F);
H (Path A->B->D->G->H) ... Found!

Open List (To be visited)
Graph search algorithms

• Two lists: **open** and **closed**

• Open list keeps track of unexplored nodes

• **Examine a node on the open list:**
  – Check to see if it is the goal
  – If not, check its edges to add more nodes to the open list
    • Each added node contains a reference to the node that explored it
  – Place current node on closed list (mark a field as visited)

• Closed list are those that have been explored/processed and are not the goal
Node selection

• Which open node becomes closed first?
  – **Breadth-First** processes the node that has been waiting the longest
  – **Best-First** processes the one that is closest to the goal
  – **Dijkstra’s** processes the one that is the cheapest according to some weight
  – **A*** chooses a node that is cheap and close to the goal
Breadth first search
Best first search
Suboptimal best first search
Dijkstra’s algorithm

• Disregards distance to goal
  – Keeps track of the cost of every path
  – No guessing / no search heuristic
• 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’s search
**A* algorithm**

- Admissible heuristic function that never overestimates the true cost
- To order the open list, use
  - Heuristic cost – estimated cost to reach the goal
  - Given cost – cost to reach node from the start

Final Cost = Given Cost + (Heuristic Cost * Heuristic Weight)
Heuristic weight

• Final Cost = Given Cost + (Heuristic Cost * Heuristic Weight)

• Heuristic weight controls the emphasis on the heuristic cost versus the given cost

• Controls the behavior of A*
  – If hw=0, final cost will be the given cost -> Dijkstra
  – As hw increases, it behaves more like Best-First
A* search
More reading on pathfinding

• Amit’s A* Pages
  http://theory.stanford.edu/~amitp/GameProgramming/

• Beginner’s guide to pathfinding
  http://ai-depot.com/Tutorial/PathFinding.html

• Waypoint graphs vs. navigation meshes
  http://www.ai-blog.net/archives/000152.html
Mazes

Maze taken from Image-guided maze construction, Xu and Kaplan 2007, Siggraph
Simple mazes – binary grid

- Matrix of booleans
- ex: 21 x 21

- Arbitrary mapping:
  - True = black pixel = wall
  - False = white pixel = open space
Game mazes– enumerated grid

• Matrix of enumerated values: ints

• Arbitrary mapping
  – 0 = white pixel = open space
  – 1 = black pixel = wall
  – 2 = health = red plus
  – Etc...
Procedural maze initial values

All open space – add walls

All walls – add open space (usually preferred)

Advanced algorithms can also start with artist input or an image
Maze generation – start with all walls

• Randomized depth first search
  – Mark current node as visited
  – Make list of neighbors, pick one randomly
  – If new node is not marked as visited, remove the wall between it and its parent
  – Recursive
Maze generation – *start with all walls*

• Randomized Kruskal's algorithm
  – Randomly select a wall that joins two unconnected open areas, and merge them
  – For an n x n grid there are initially $n^2$ open areas, one per grid cell

• Randomized Prim's algorithm
  – Randomly pick a starting point
  – Keep track of walls on the boundary of the maze
  – Randomly select a wall that “grows” the maze
Examples

Animation examples from
http://en.wikipedia.org/wiki/Maze_generation_algorithm
Image-based Mazes
Image-based Mazes