§5 Path Finding

- common problem in computer games
  - routing characters, troops etc.
- computationally intensive problem
  - complex game worlds
  - high number of entities
  - dynamically changing environments
  - real-time response

Problem statement

- given a start point *s* and a goal point *r*, find a path from *s* to *r* minimizing a given criterion
- search problem formulation
  - find a path that minimizes the cost
- optimization problem formulation
  - minimize cost subject to the constraint of the path

The three phases of path finding

1. discretize the game world
   - select the waypoints and connections
2. solve the path finding problem in a graph
   - let waypoints = vertices, connections = edges, costs = weights
   - find a minimum path in the graph
3. realize the movement in the game world
   - aesthetic concerns
   - user-interface concerns

Discretization

- waypoints (vertices)
  - doorways, corners, obstacles, tunnels, passages, …
- connections (edges)
  - based on the game world geometry, are two waypoints connected
- costs (weights)
  - distance, environment type, difference in altitude, …
- manual or automatic process?
  - grids, navigation meshes

Grid

- regular tiling of polygons
  - square grid
  - triangular grid
  - hexagonal grid
- tile = waypoint
- tile's neighbourhood = connections

Navigation mesh

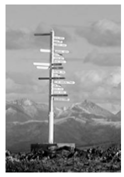
- convex partitioning of the game world geometry
  - convex polygons covering the game world
  - adjacent polygons share only two points and one edge
  - no overlapping
- polygon = waypoint
  - middlepoints, centre of edges
- adjacent polygons = connections

Solving the convex partitioning problem

- minimize the number of polygons
  - points: n
  - points with concave interior angle (notches): $r \leq n - 3$
- optimal solution
  - dynamic programming: $O(r^2 n \log n)$
- Hertel–Mehlhorn heuristic
  - number of polygons ≤ 4 × optimum
  - running time: $O(n + r \log r)$
  - requires triangulation
    - running time: $O(n)$ (at least in theory)
    - Seidel's algorithm: $O(n \lg^* n)$ (also in practice)

Path finding in a graph

- after discretization form a graph $G = (V, E)$
  - waypoints = vertices (V)
  - connections = edges (E)
  - costs = weights of edges ($weight : E \rightarrow \mathbf{R}_+$)
- next, find a path in the graph



Graph algorithms

- breadth-first search
  - running time: $O(|V| + |E|)$
- depth-first search
  - running time: $\Theta(|V| + |E|)$
- Dijkstra's algorithm
  - running time: $O(|V|^2)$
  - can be improved to $O(|V| \log |V| + |E|)$

Heuristical improvements

- best-first search
  - order the vertices in the neighbourhood according to a heuristic estimate of their closeness to the goal
  - returns optimal solution
- beam search
  - order the vertices but expand only the most promising candidates
  - can return suboptimal solution

Evaluation function

- expand vertex minimizing

  $$f(v) = g(s \rightsquigarrow v) + h(v \rightsquigarrow r)$$
- $g(s \rightsquigarrow v)$ estimates the minimum cost from the start vertex to v
- $h(v \rightsquigarrow r)$ estimates (heuristically) the cost from v to the goal vertex
- if we had exact evaluation function f^*, we could solve the problem without expanding any unnecessary vertices