COMPSCI 777 S2 C 2004
Computer Games Technology

—A* Path Planning—

Hans W. Guesgen
Computer Science Department

THE UNIVERSITY OF AUCKLAND
NEW ZEALAND

The Path-Planning Problem

- What kind of problem is path planning?
- A planning problem?
- A search problem?
- A special search problem?

Path Planning as Informed Search

- Given a start and a goal, find a path that is optimal with respect to given criterion.
- Informed search uses problem-specific knowledge to finding such a path.
- Usually, the knowledge is encoded in an evaluation function that returns a number indicating the desirability of expanding the node.
- When the nodes are ordered so that the one with the best evaluation is expanded first, the resulting strategy is called best-first search.

The A* Algorithm

- The algorithm uses an evaluation function $f(N) = g(N) + h(N)$.
- $g(N)$ denotes the costs so far, i.e., the costs to get from the start to the current position.
- $h(N)$ denotes the estimated cost to the goal, i.e., an approximation of the costs from the current position to the goal.
- $h(N)$ is an admissible heuristic, i.e., it never overestimates the actual cost of the best solution through N.
- A* search is complete and optimal.

Graceful Decay of Admissibility

- For some problems, the only way to always avoid overestimating the cost to reach the goal is to set h to zero, which leads to uniform cost search.
- On the other hand, if h rarely overestimates the actual cost to reach the goal by more than δ, then the algorithm will rarely find a solution whose cost is more than δ greater than the cost of the optimal solution.
- This might be good enough for most computer games applications.

Pseudo-code for the A* Algorithm

1. Let S be the starting point.
2. Assign f, g, and h values to S.
3. Add S to the Open list. At this point, S is the only node on the Open list.
4. Let B be the best node from the Open list (best node has the lowest f-value).
   (a) If B is the goal node, then quit (a path has been found).
   (b) If the Open list is empty, the quit (a path cannot be found).
5. Let C be a valid node connected to B.
   (a) Assign f, g, and h values to C.
   (b) Check whether C is on the Open or Closed list.
      i. If so, check whether the new path is more efficient (lower f-value).
         If so, update the path.
      ii. Otherwise, add C to the Open list.
      iii. Repeat Step 5 for all valid successors of B.
6. Put B on the Closed list and repeat from Step 4.

Some Heuristics

- Given current node $N = (N_x, N_y)$ and goal node $G = (G_x, G_y)$
- Euclidean distance:

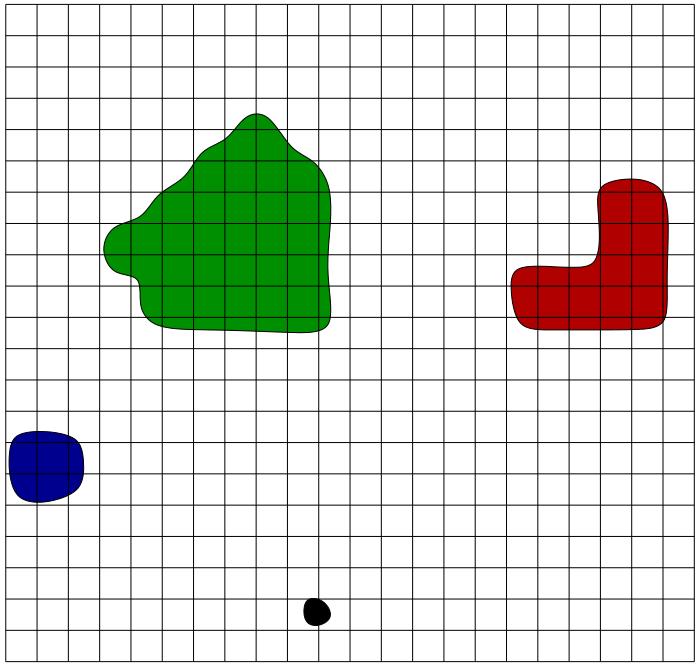
$$h(N) = \sqrt{(N_x - G_x)^2 + (N_y - G_y)^2}$$

- Manhattan distance:

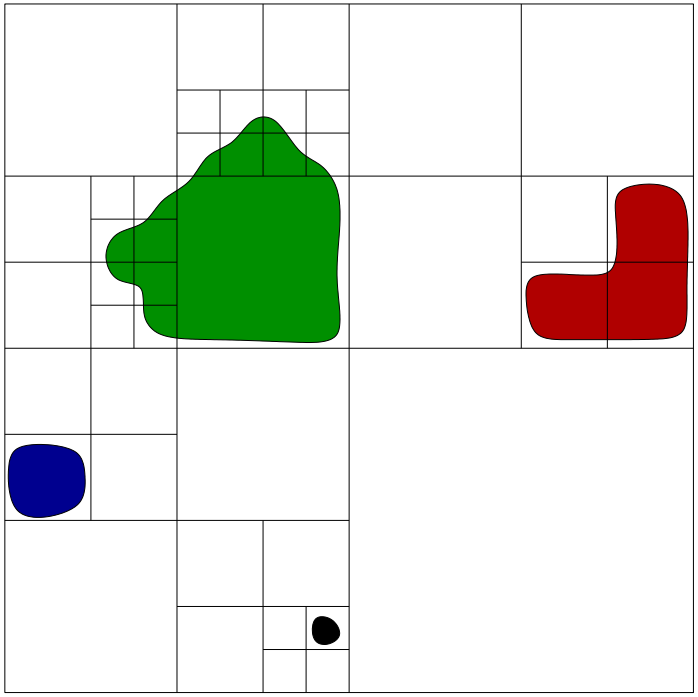
$$h(N) = |N_x - G_x| + |N_y - G_y|$$

- The Manhattan distance might not be admissible.

Partitioning the Space

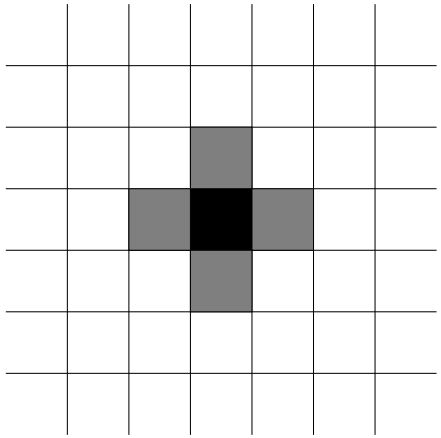


Rectangular grid

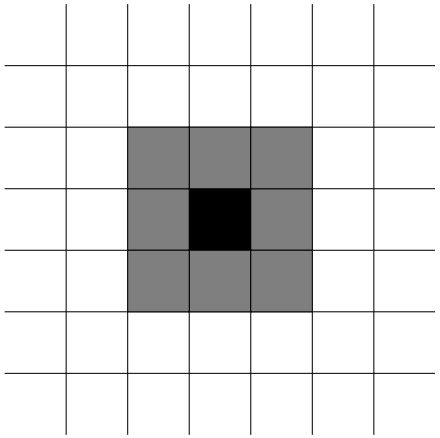


Quadtree

Successors in Rectangular Grids

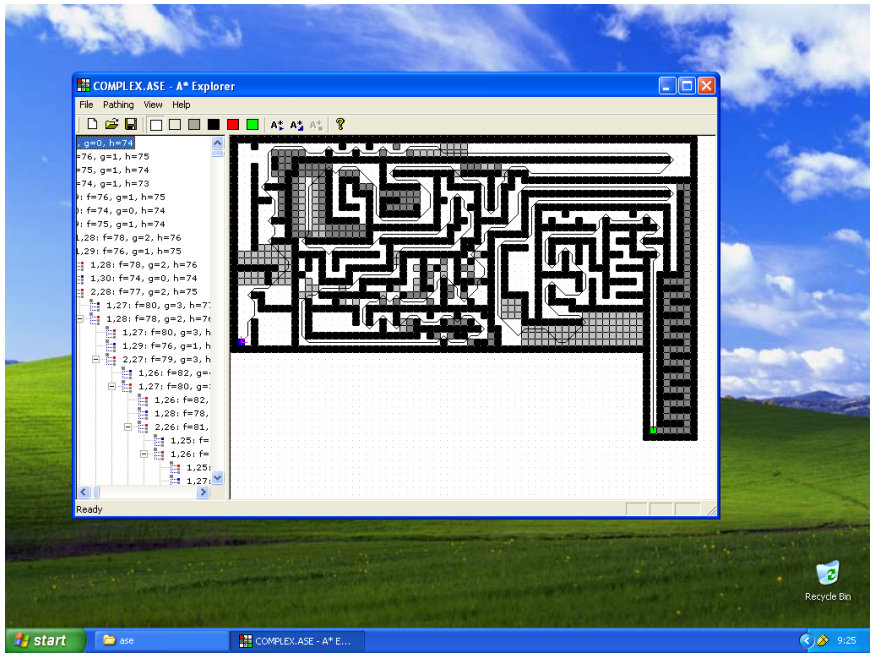


Edge neighbors



Vertex neighbors

The A* Explorer



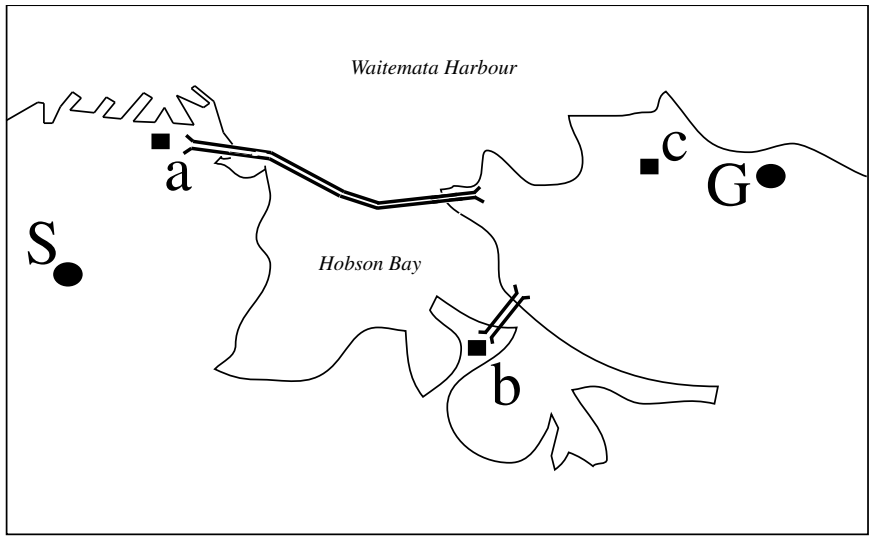
- Written by James Matthews of Generation5.
- Good for exploring various aspects of the A* algorithm.
- Available from the course website.

Designing a Path-Finding Engine

- Players will complain if the computed paths are not appropriate, which means that good path planning is essential.
- Good path planning is expensive but has to fit into the allocated CPU time (to avoid "game freeze").
- A path-finding engine might have to perform time-sliced path planning to compute good path in reasonable time.

Island Search (Dillenburg)

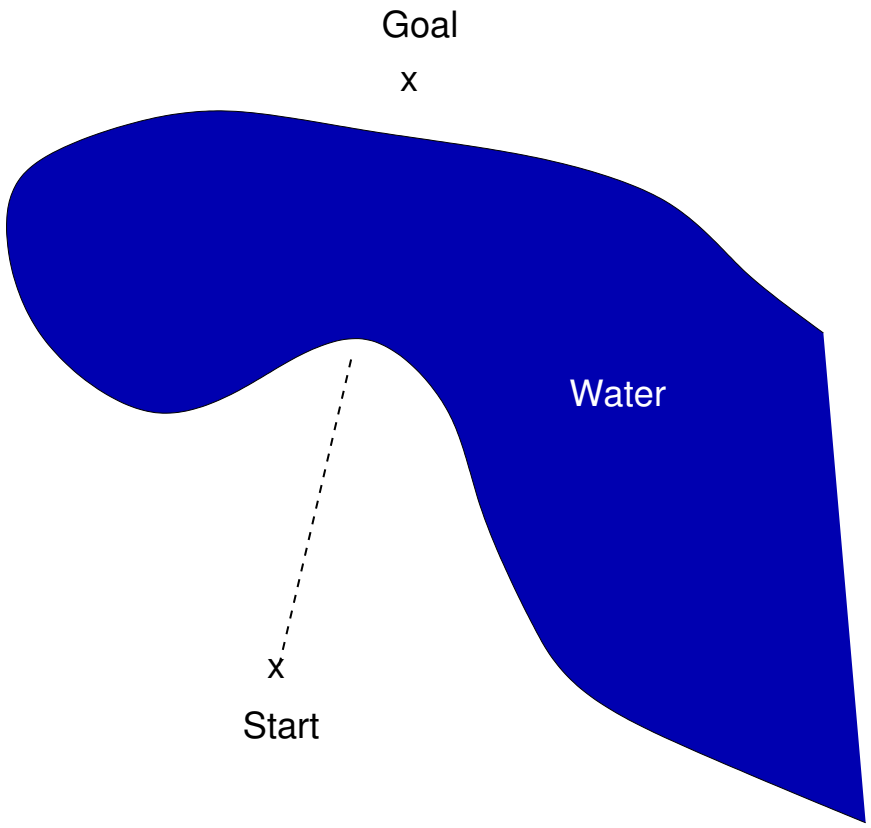
- Search with subgoals (islands) can reduce node expansions.
- It has potential for path finding as knowledge of commonly traversed intersections is often known prior to search.



Three-Step Path Finding (Higgins)

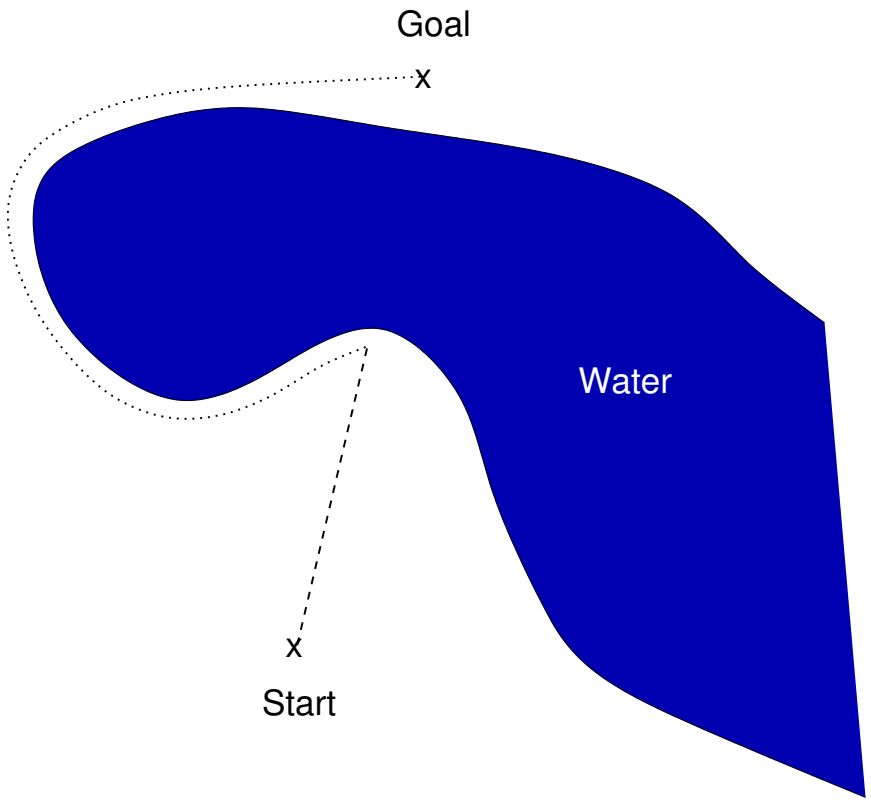
- Assume the player gives a order to move a unit in the game.
- The player expects the unit to respond quickly, rather than not to move until a perfect path is computed.
- To achieve this goal, path finding can be divided into three phases:
  - Computing a quick path
  - Computing a full path
  - Computing a splice path

Quick Path



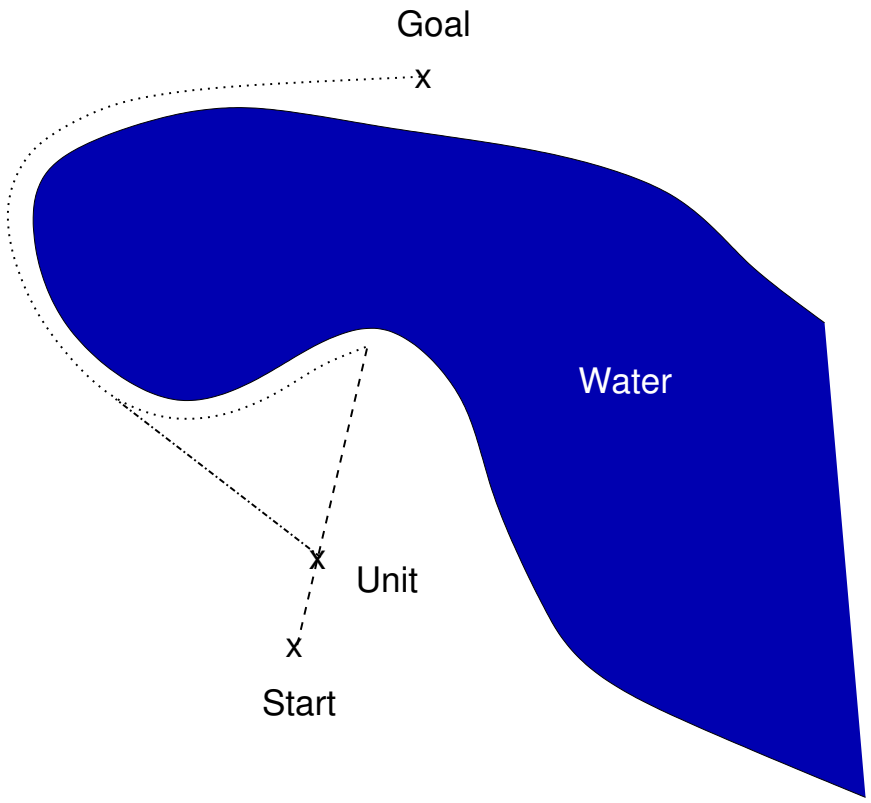
- Get a unit moving by using a high-speed short-distance path finder, which searches between 3 to 15 tiles towards the goal.

Full Path



- Perform a thorough search from the end of the quick path to the goal, which is done while the unit moves.

Splice Path



- Use the high-speed path finder again to compute a better path from the current position to a position on the full path.

Iterative Deepening A* Search (IDA*)

- IDA* is based on the same idea as regular iterative deepening:
  - Search is performed only to a limited depth.
  - If search fails, the depth limit is relaxed.
  - Instead of a depth limit, IDA* uses an f-cost limit.
- IDA* usually uses less memory than A*.
- IDA* speeds up the detection of almost-straight paths.
- IDA* improves performance in environments where other agents can temporarily block paths.