Informed Search

- Informed search uses a heuristic to choose the best node to search next
  - Also called Heuristic Search.
  - Incorporates knowledge (heuristic) of the problem domain to minimize search time.
- Algorithms define a strategy for node selection (which node to search next) rather than selecting the next node at random.

Heuristic

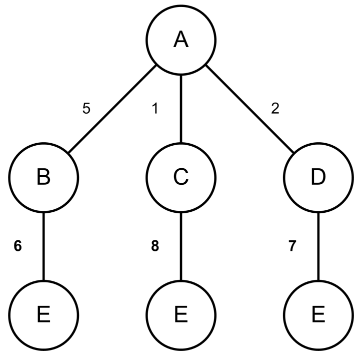
- A heuristic is a rule of thumb that may help solve a problem.
- Heuristics take problem knowledge into account to help guide the search in a problem domain.

Best-First Search (Best-FS)

- Best-FS is related to uniform cost (UCS) search in that it takes the cummulative cost of the solution into account.
- The big difference is that Best-FS uses a heuristic to estimate the *remaining cost* of the solution

Uniform-Cost Search (UCS)

- Find the least-cost solution in this state tree.
- Not all edges the same cost.
- Goal to find the path from start to finish with *least* cost (A->E).
- Note: this is important in navigation, least cost path to move from one location to another.
- Can be efficiently be implemented with a priority queue.



Step	Investigating Node	Priority Queue		
1		A(0)		
2	A	C(1)	D(2)	B(5)
		A(0)	A(0)	A(0)
3	C	D(2)	B(5)	**E(8)**
		A(0)	A(0)	C(1)
				A(0)
4	D	B(5)	**E(7)**	**E(8)**
		A(0)	D(2)	C(1)
			A(0)	A(0)
5	B	**E(6)**	**E(7)**	**E(8)**
		B(5)	D(2)	C(1)
		A(0)	A(0)	A(0)

Sort the priority queue according to cumulative cost g(n)

UCS Algorithm

```
procedure UCS(Graph, root, goal)
   n := root
   cost := 0
   Frontier := priority queue containing n only
   while Frontier is not empty
      n := Frontier.pop()
      if goal(n)
         return n
      for all neighbors w of n
         if w is not in Frontier
            Frontier.add(w)
         if w is in Frontier with higher cost
            replace existing node with w
```

Best-FS Algorithm

- Still uses a priority queue (the OPEN list)
- Improves the UCS in two ways:
  - Heuristic lookahead
  - Avoid loops in the search by maintaining a list of nodes already visited (the CLOSED list)

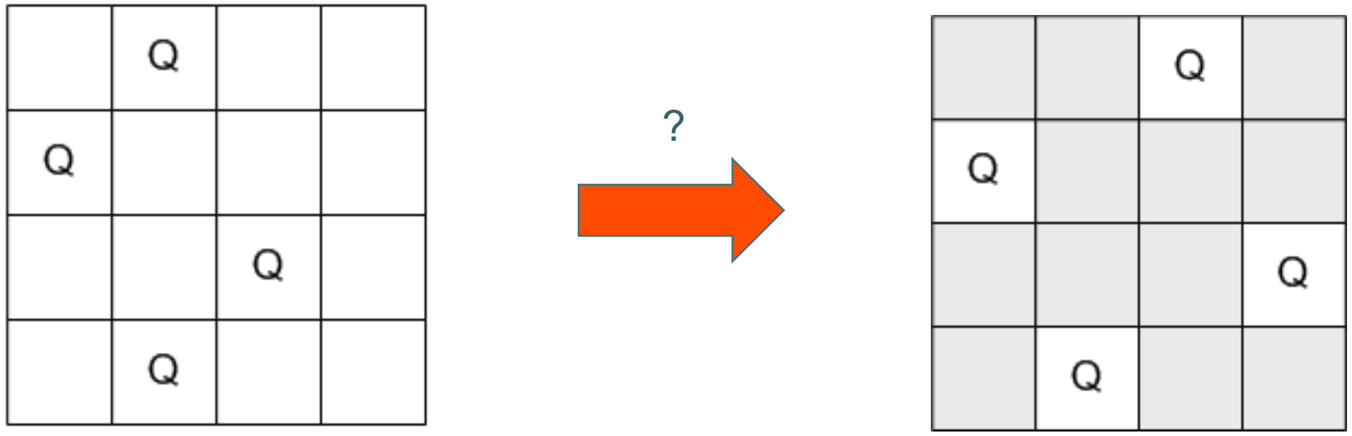
Best-FS Algorithm

```
OPEN := [all nodes]
CLOSED := [ ]
while OPEN is not empty
    Sort OPEN according to f(n).
    Remove the best node from OPEN, call it n, add it to CLOSED.
    If goal(n)
       return n
    Create n's successors.
    For each successor do:
       If it is not in CLOSED, add it to OPEN.
```

cost estimate of node n: $f(n) = \underbrace{h(n)}_{\text{Cummulative cost}} + \underbrace{g(n)}_{\text{Heuristic cost estimate}}$

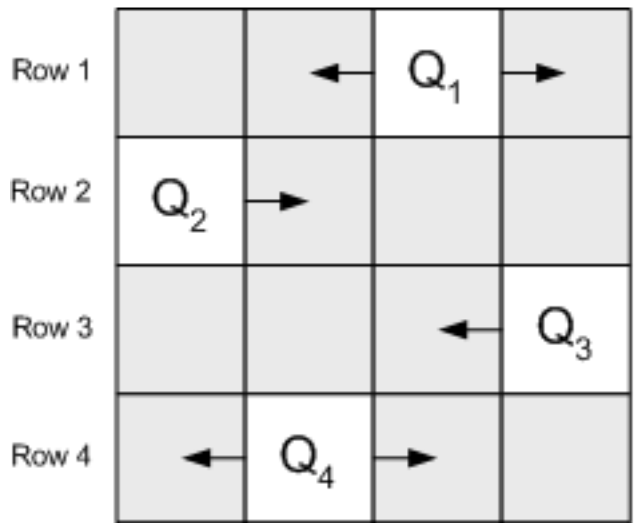
The N-Queens Problem

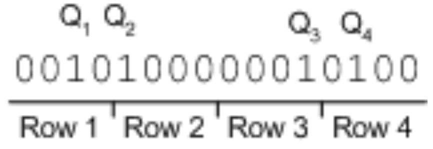
- Classic AI problem
  - Given an NxN board with N queens on it
  - Can you move the queens in such a way that they will not threaten each other?



The N-Queens Problem

- Queens are only allowed to move horizontally in "their" rows.
- Convenient representation of 4-Queens problem: unsigned 16-bit integer

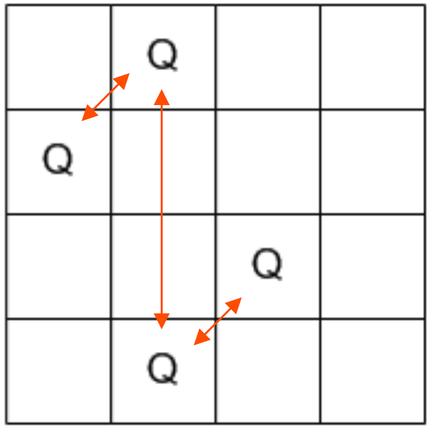




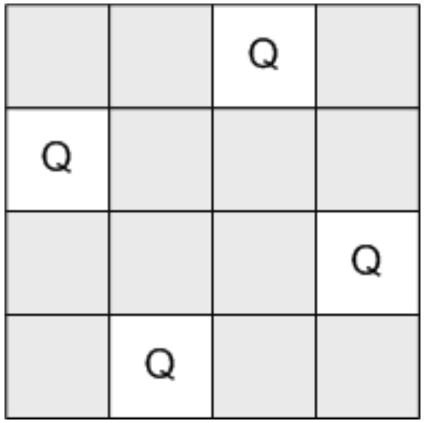
The N-Queens Problem

- In order to use Best-FS we need to design our evaluation function f(n) = g(n) + h(n):
  - h(n) is number of moves made so far
  - g(n) is number of queens threatening each other
    - idea: the more queens are threatening each other the more expensive it is to get to a non-threatening board configuration.

The N-Queens Problem

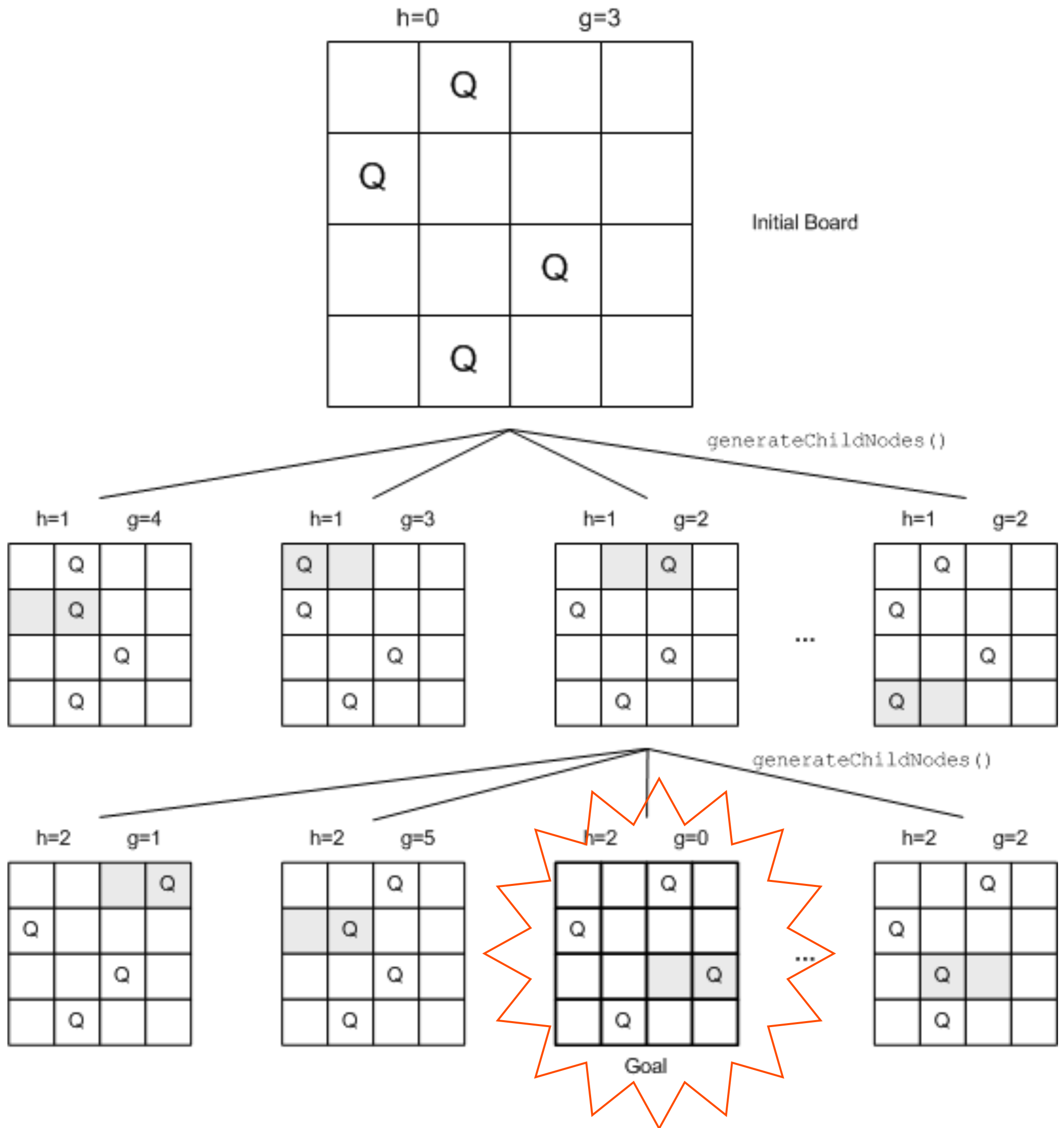


h=3



h=0

N-Queens Problem



A* Pathfinding

- The A* pathfinding algorithm is an <u>exhaustive search</u> heuristic which is guaranteed to find the <u>shortest path</u> between two points.
- Like Best-FS, but evaluates with:
  - g(n) is cost of the path initial state to current state
  - h(n) is the estimated cost of the path current state to goal state
- For Quake2 the basic assumption is that the search area is tiled (e.g. square tile) and that the agent moves from the center of one tile to the center of the next tile.
- Added complication: we need to keep track of the path so that we can plan agent actions once the shortest path has been found

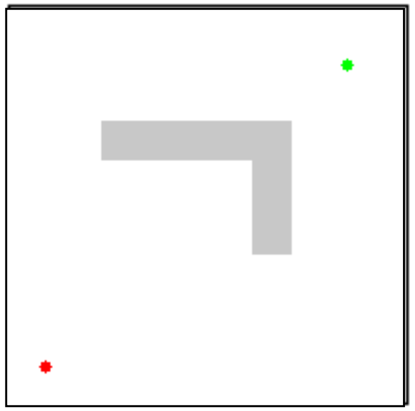
The A* Algorithm

F(n) = G(n) + H(n)

- Add the starting square to the open list.
- Repeat the following:
  - Sort the open list. Look for the lowest F cost square on the open list. We refer to this as the current square.
  - Switch it to the closed list.
  - For each square n adjacent to this current square
    - If n is not walkable or if it is on the closed list, ignore it.
    - If n is not on the open list, add it to the open list. Make the current square the parent of n. Record the F, G, and H costs of n.
    - If n is on the open list already, check to see if this path to that square is better, using G cost as the measure. If so, change the parent of n to the current square, and recalculate the G and F scores of n.
  - Stop when you:
    - Add the target square to the closed list, in which case the path has been found or
    - Fail to find the target square, and the open list is empty. In this case, there is no path.
- Save the path. Working backwards from the target square, go from each square to its parent square until you reach the starting square.

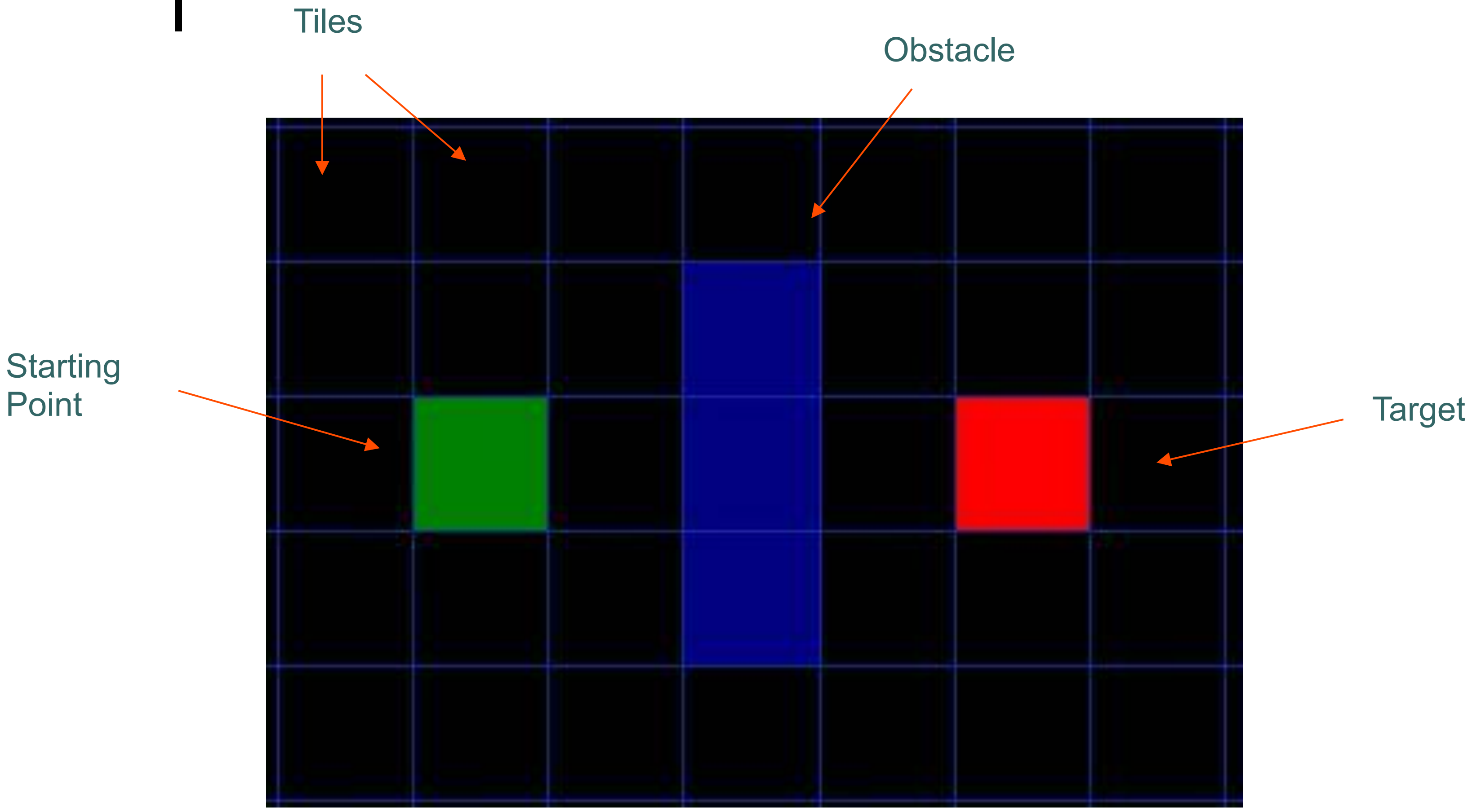
Source: *A* Pathfinding for Beginners*, Patrick Lester, http://www.policyalmanac.org/games/aStarTutorial.htm

The A* Algorithm



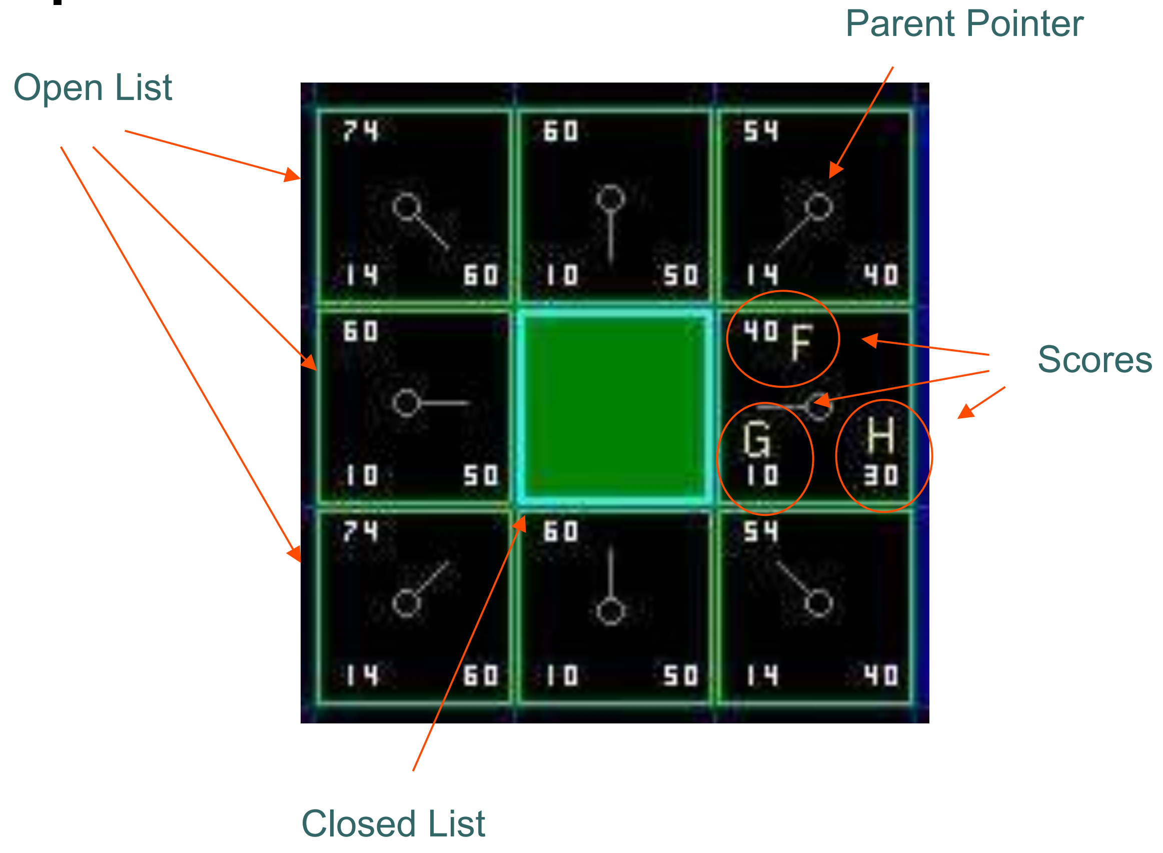
Source: http://en.wikipedia.org/wiki/A*_search_algorithm

The Search Area



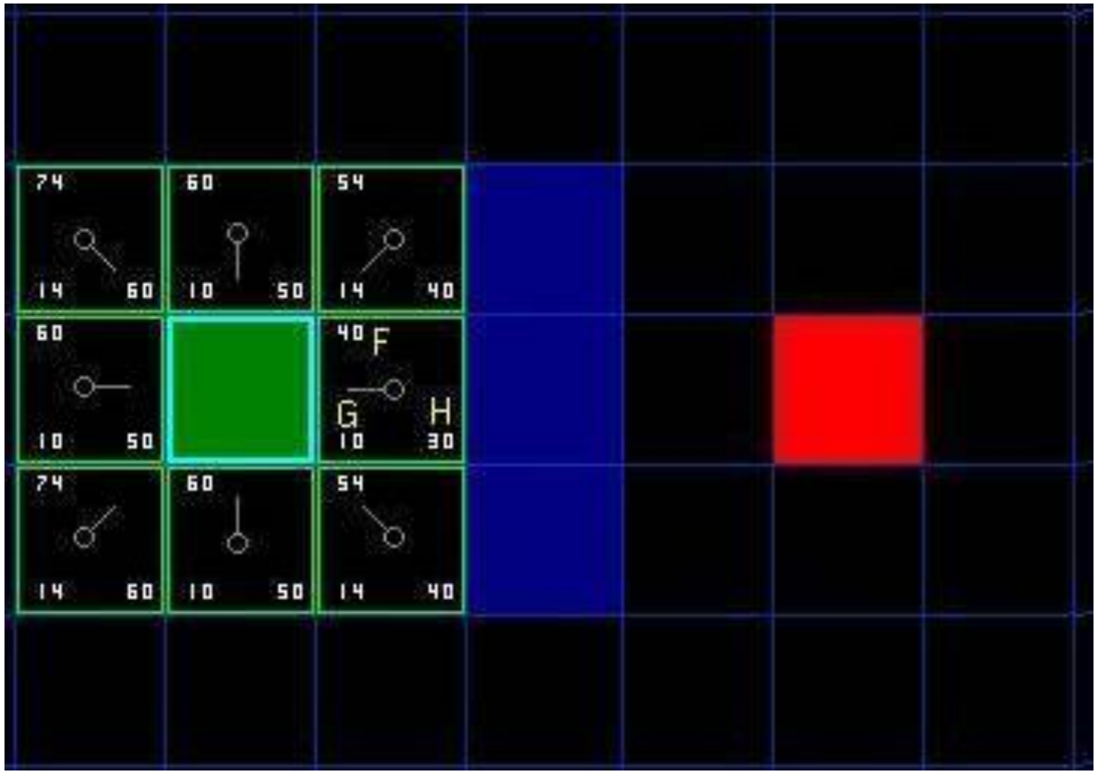
What is the shortest path from the starting point to the target?

Data Structures

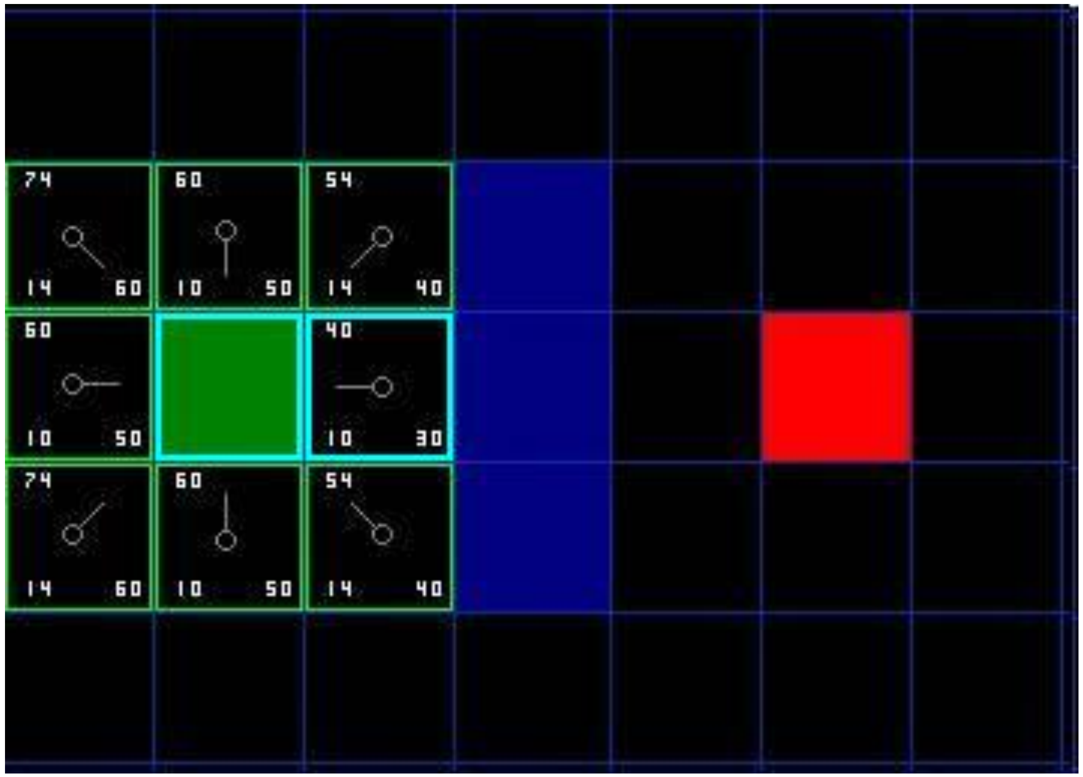


Scores:
F = G + H
G = the movement cost to move from the starting point to the given square following the generated path - 10 points for each vert./horiz. move and 14 points for moves across corners.
H = the estimated movement cost to move from the given square to the target - “manhattan distance”

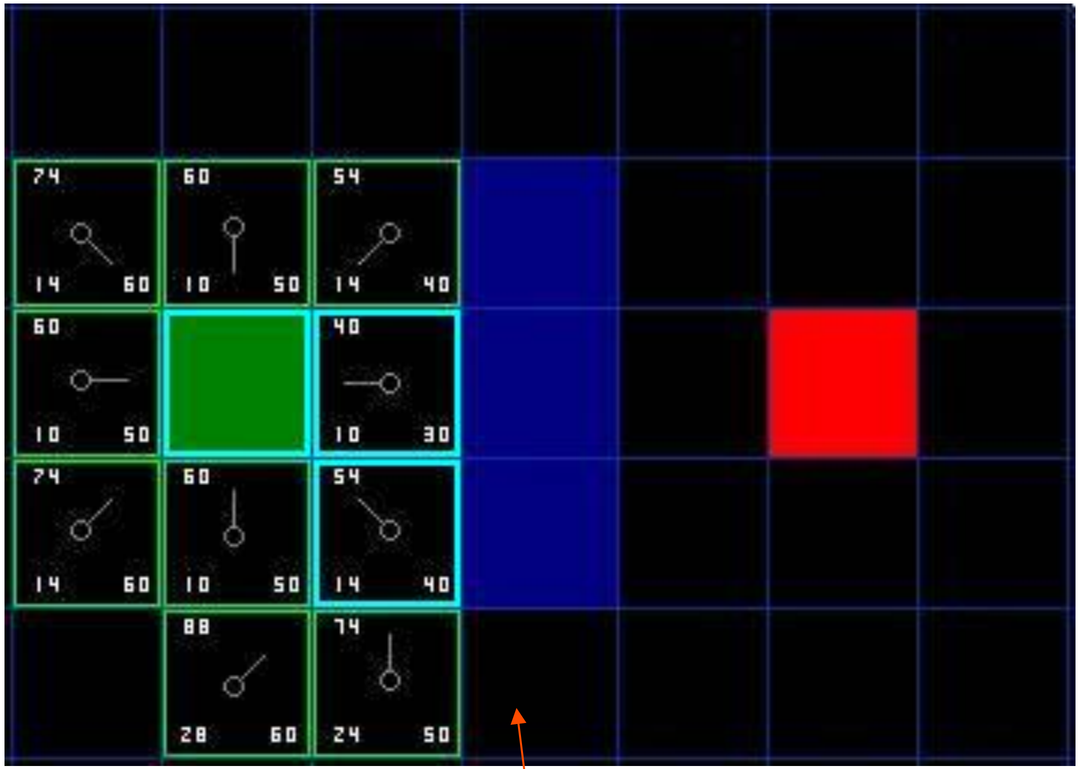
First Iterations



First Iterations

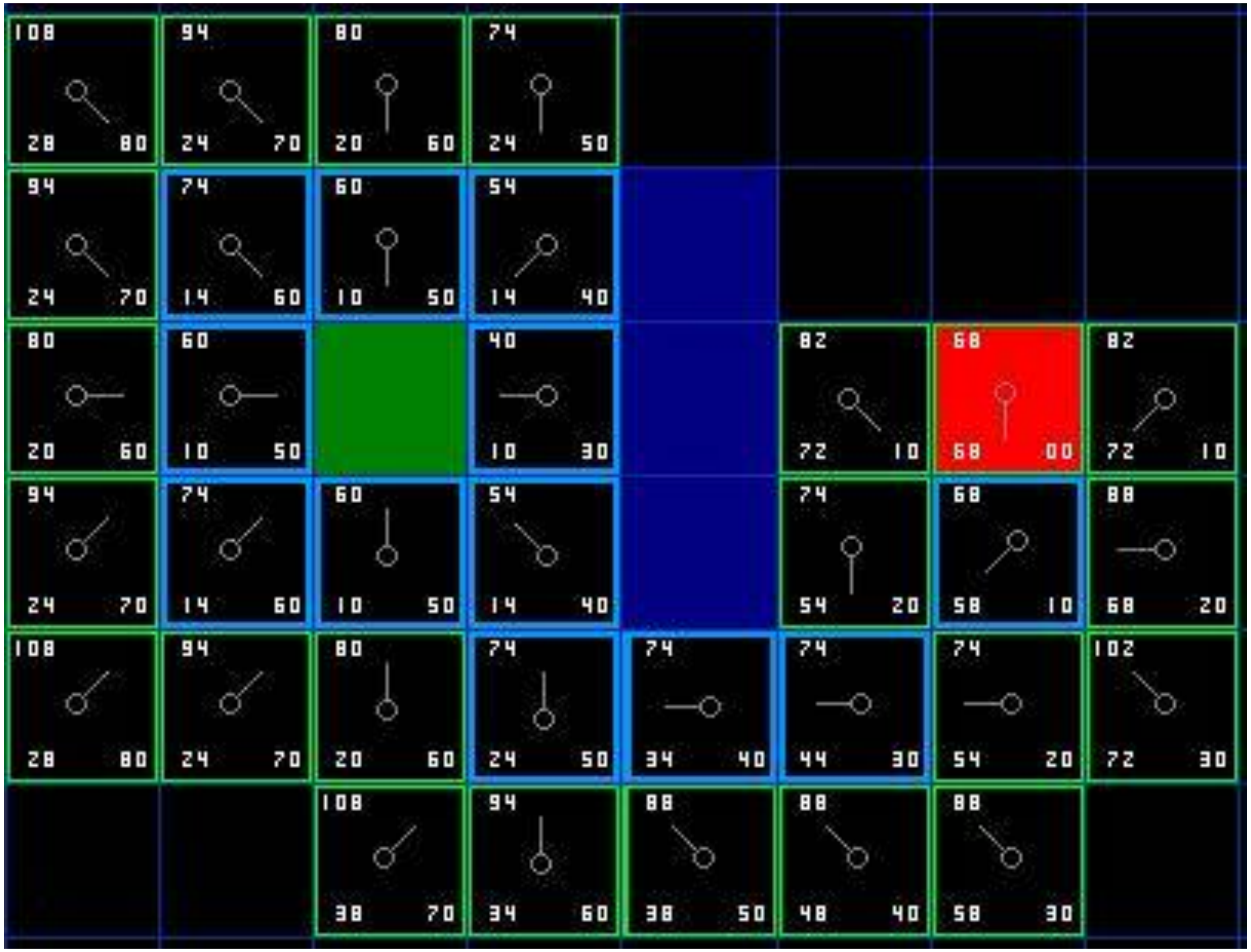


First Iterations

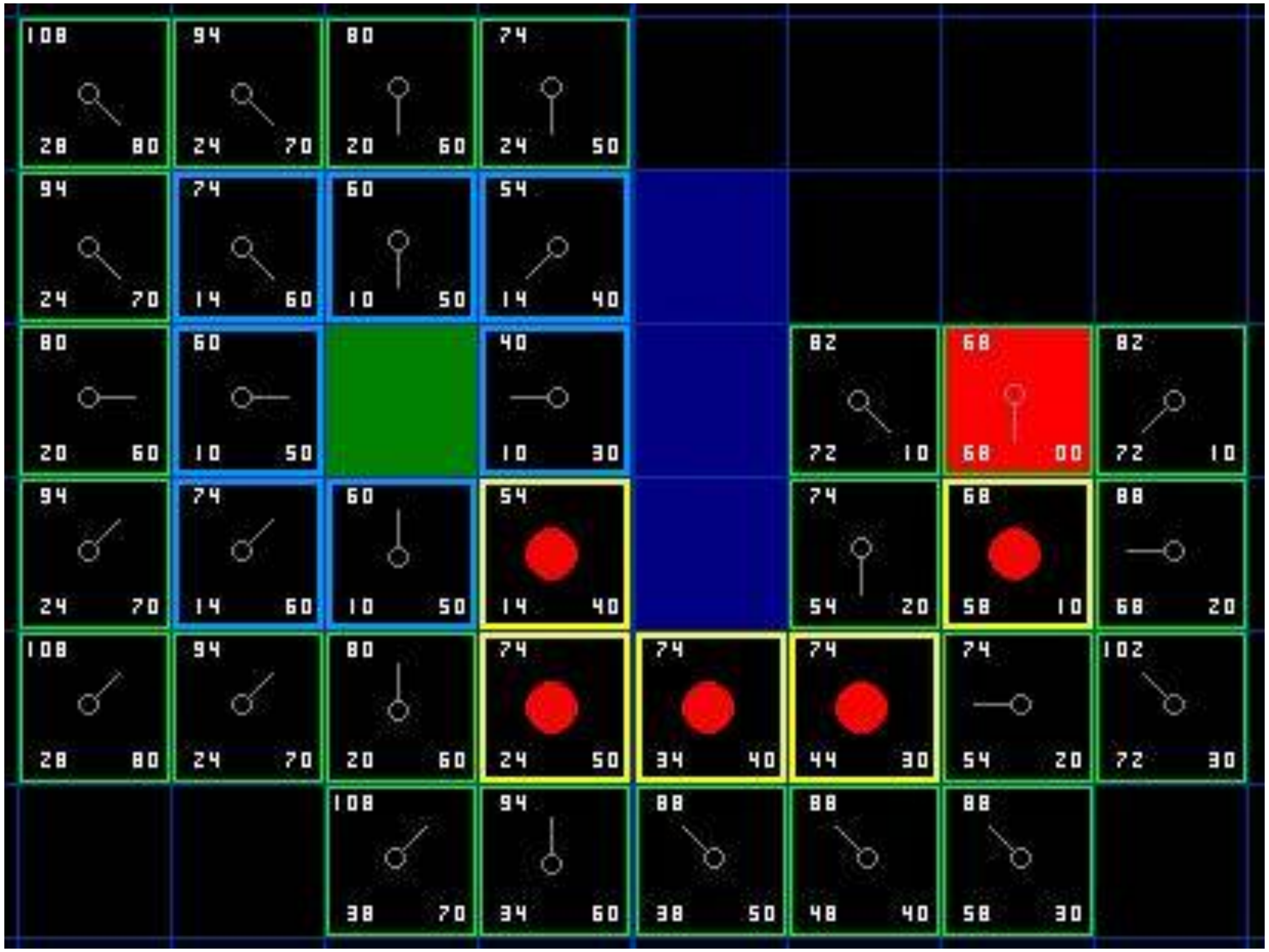


not added – would cut across the corner of the wall

Final Iterations



Final Iterations



Assignments

- Read Chapter 3
- Read the A* Tutorial on the course website:

  http://www.policyalmanac.org/games/aStarTutorial.htm