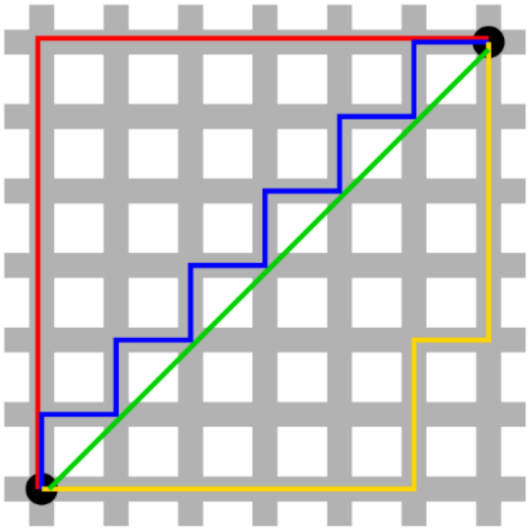
Greedy First Search and A*:

Heuristic (h) :

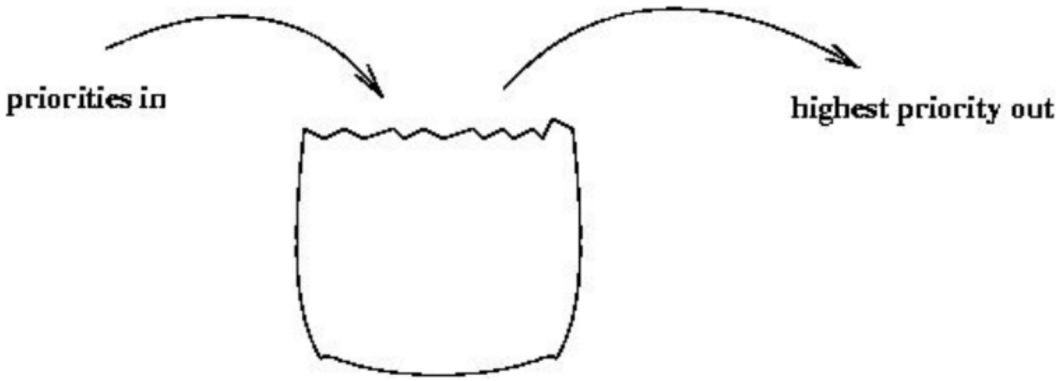
Using heuristic as manhattan distance :

- manhattan(x,y)=abs(x-x_{goal})+abs(y-y_{goal}) : for movement in (left,right,up, down)
- eculdian(x,y)= sqrt((x-x_{goal})^2 + (y-y_{goal})^2) : used for any direction

The best heuristic for our pac man is manhattan



- Total Distance (g): total distance from start node to current goal
- Evaluation function (f):
  - **f(n)=g(n)Xalgrotim_type+h(n)**
  - **Algorithm_type=0 for greedy**
  - **Algorithm_type=1 for A***
- The node with lowest f is chosen to visit by using priority queue :Think of a priority queue as a kind of bag that holds priorities. You can put one in, and you can take out the current **highest** priority. (Priorities can be any **Comparable** values; in our examples, we'll just use numbers.)A priority queue is different from a "normal" queue, because instead of being a "first-in-first-out" it is “higesht priority out ”



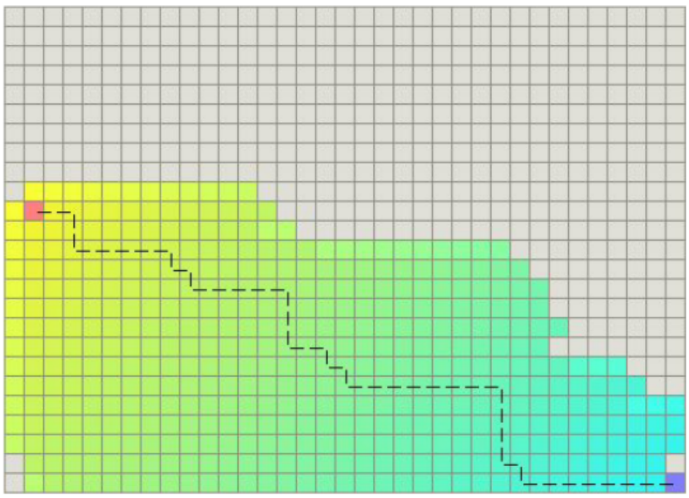
- Steps:

Start point inserted to piority queue
Set visited flag for all points to false
expanded=0
Every points path is intialized to start
Every point distance is defaulted to 0
While queue has elements:
Current node =pop(piorty queue)
If current node is visited
Search for the next available node
If current node is goal :
Return path[goal],distance[goal],expanded
Else :
Expanded++
visited[child]=true

Else
Mark this node as visited (set visited flag to true)
Search for children of current node
For each child :
Store distance $_{child}$ g(n) = distance $_{parent}$ + path cost (1)
f(n)=g(n)+h(n)
Insert (priority queue,child)
Path of child=path of parent .append(child location)

Breaking ties in herustics

In some grid maps there are many paths with the same length. For example, in flat areas without variation in terrain, using a grid will lead to many equal-length paths. A* might explore all the paths with the same f value, instead of only one.**[1]**

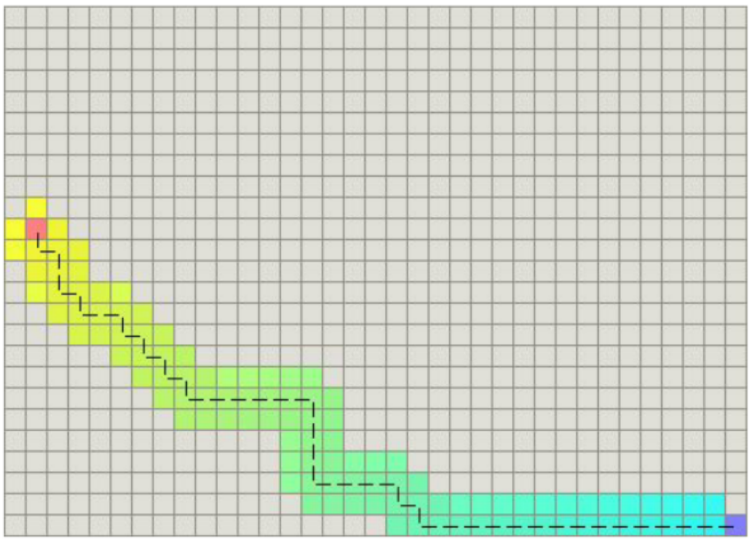


The quick way to work around this problem is to either adjust the g or h values. The tie breaker needs to be deterministic with respect to the vertex (i.e., it shouldn't be a random number), and it needs to make the f values differ. Since A* sorts by f value,

making them different means only one of the “equivalent” f values will be explored.One way to break ties is to nudge the scale of h slightly. If we scale it downwards, then f will increase as we move towards the goal. Unfortunately, this means that A* will prefer to expand vertices close to the starting point instead of vertices close to the goal. We can instead scale h upwards slightly (even by 0.1%). A* will prefer to expand vertices close to the goal.

heuristic *= (1.0 + p)

p <(minimum cost of taking one step)/(expected maximum path length)



A different way to break ties is to prefer paths that are along the straight line from the starting point to the goal:

p=cross-product-of(vector-of-goal-to-start,vector-of-goal-to-current-point)*1/10²

Heurstic for many goals:

With many golas : $goal_1$, $goal_2$, $goal_3$,$goal_4$, …. . there will be n! possible end states and (n)! start possibe states and choosing witch will be NP hard problem to find shortest road .with heuristic information we can rearrange goals order according to h.