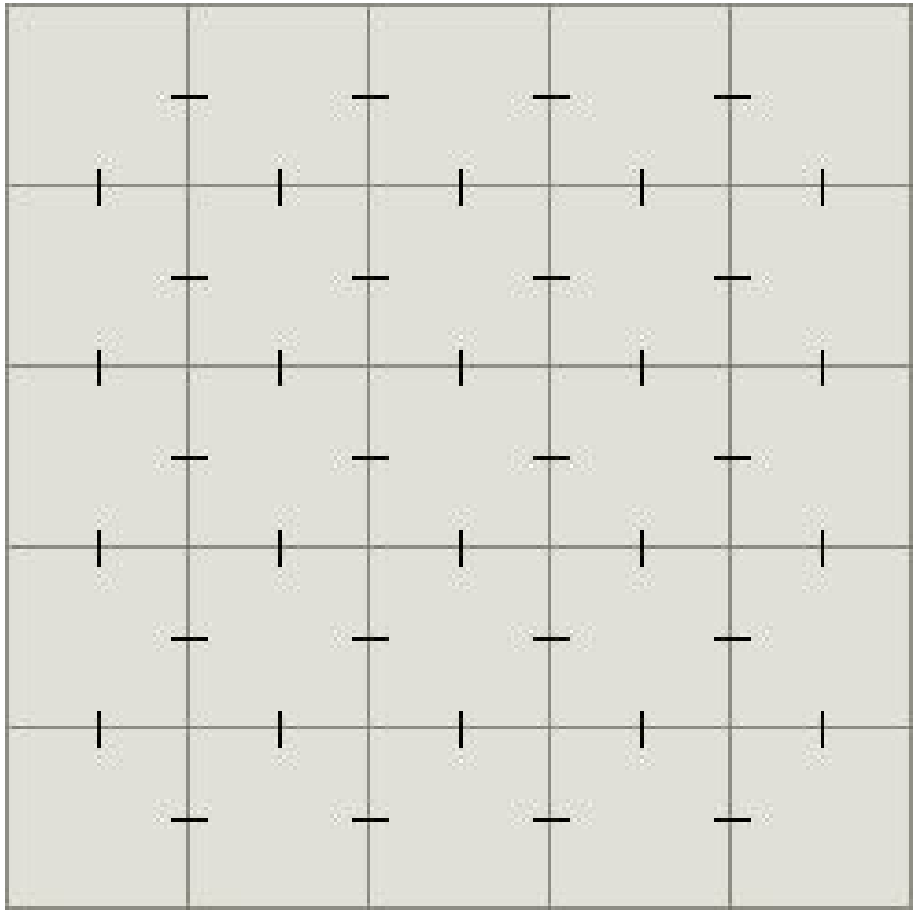

A ★

Allan Barbosa
Wagner Williams
Rodolfo Moreira

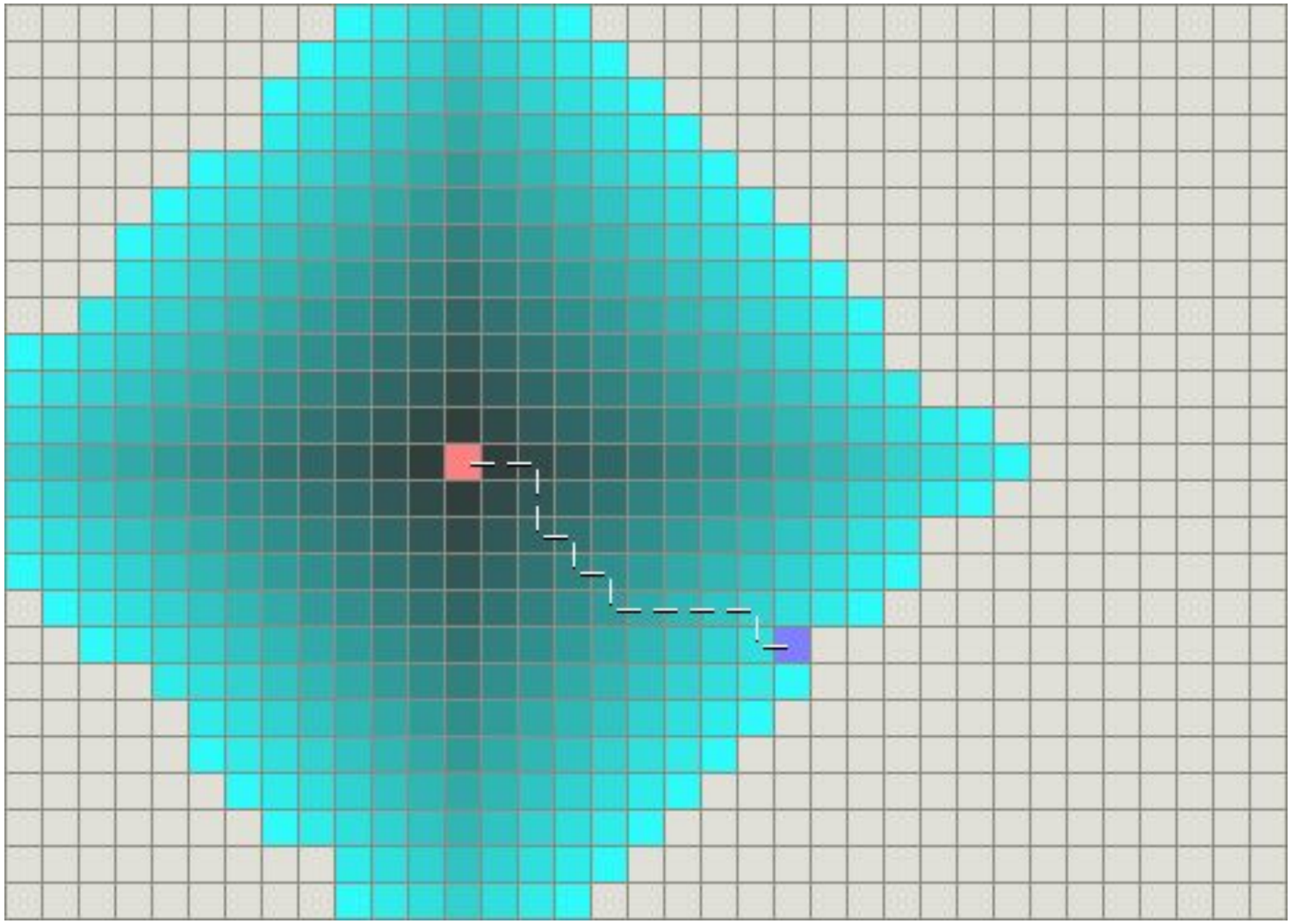
A*

- **Como encontrar o menor caminho entre dois pontos?**



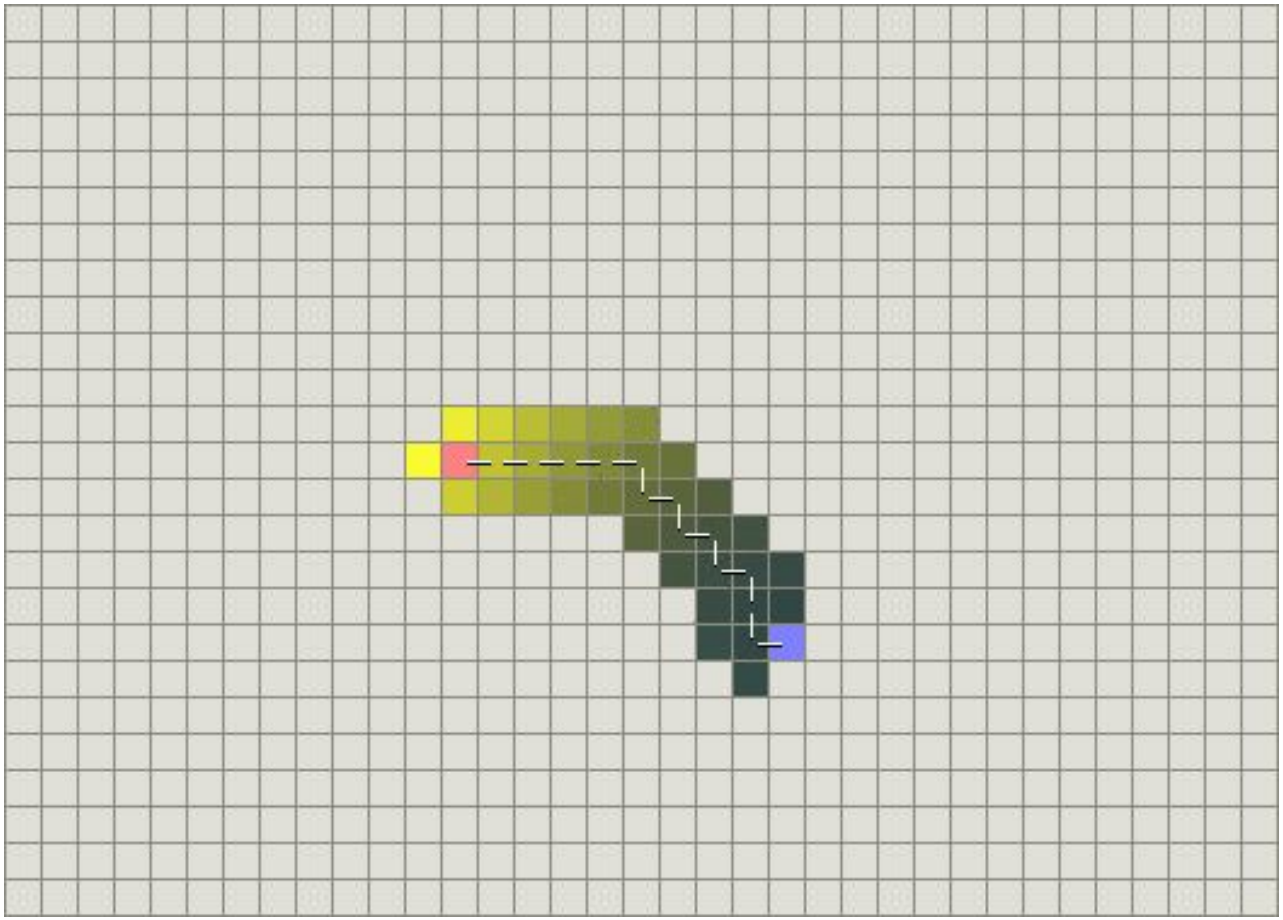
A*

- **Acertaram! Dijkstra!**



A*

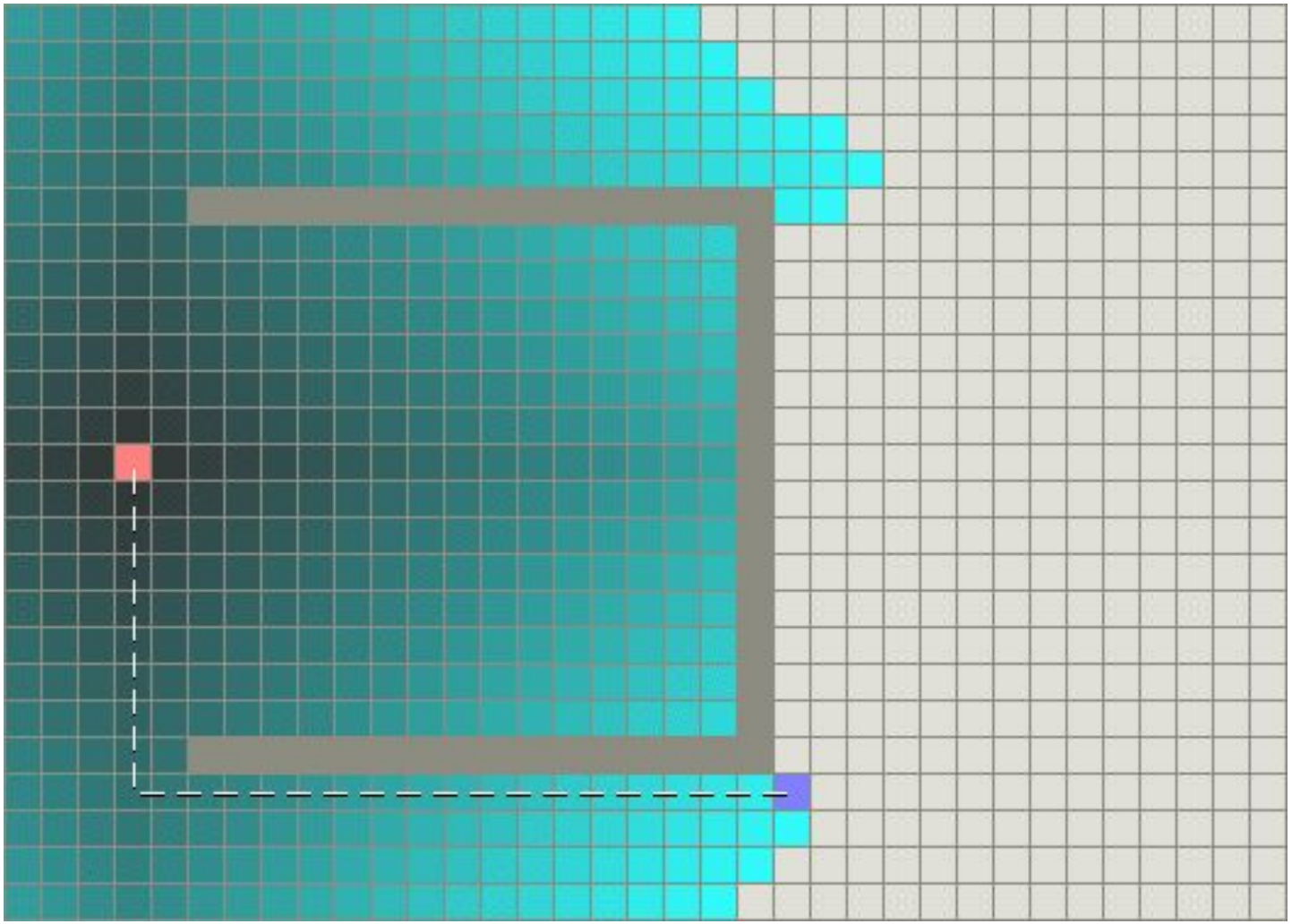
- Também poderíamos aplicar o algoritmo “melhor primeiro” (best-first search), que utiliza-se da heurística para achar o menor caminho:



- **Mas e se tivesse uma pedra no meio do caminho? Ou um muro bem largo ?**

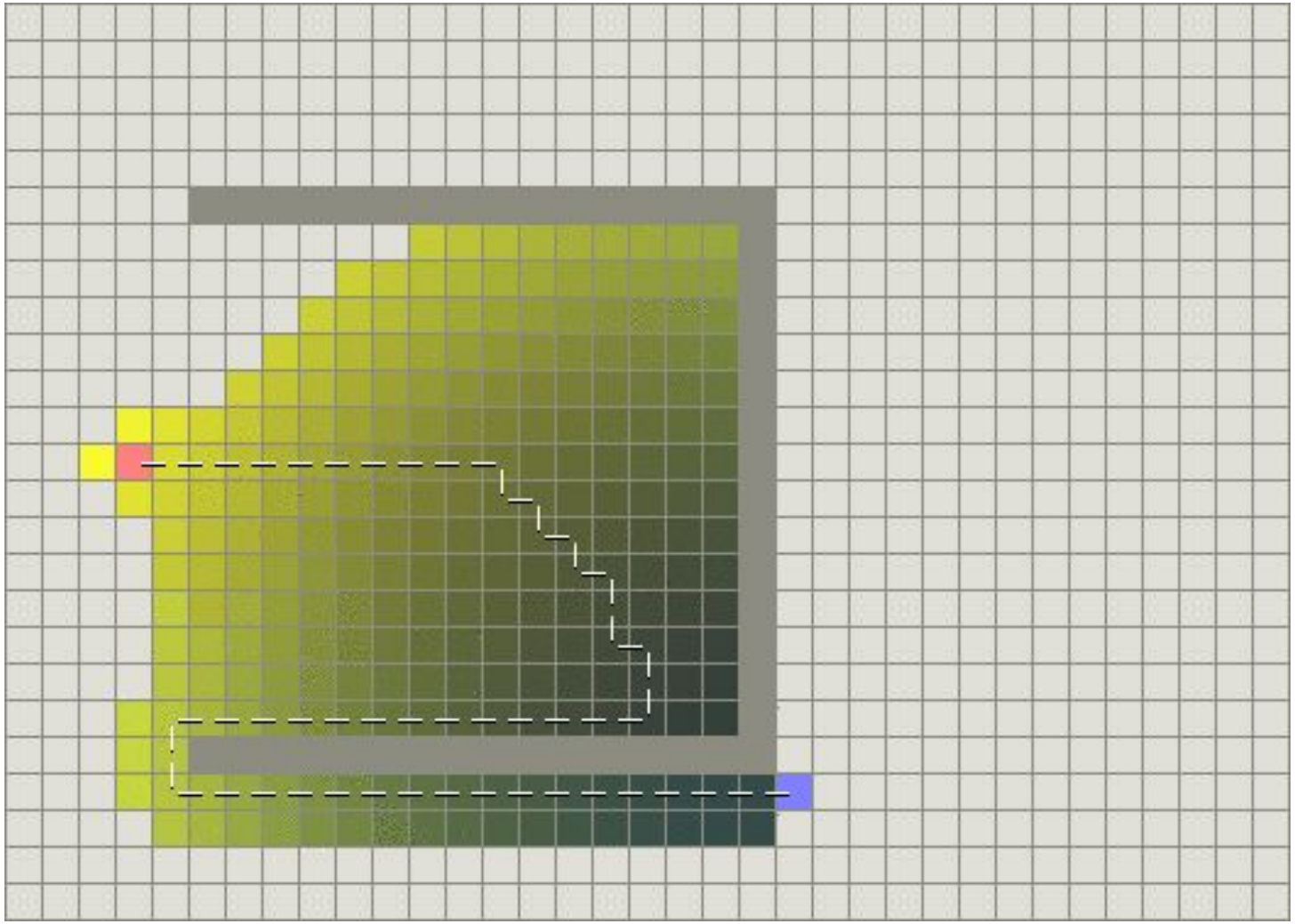
A★

- Dijkstra..

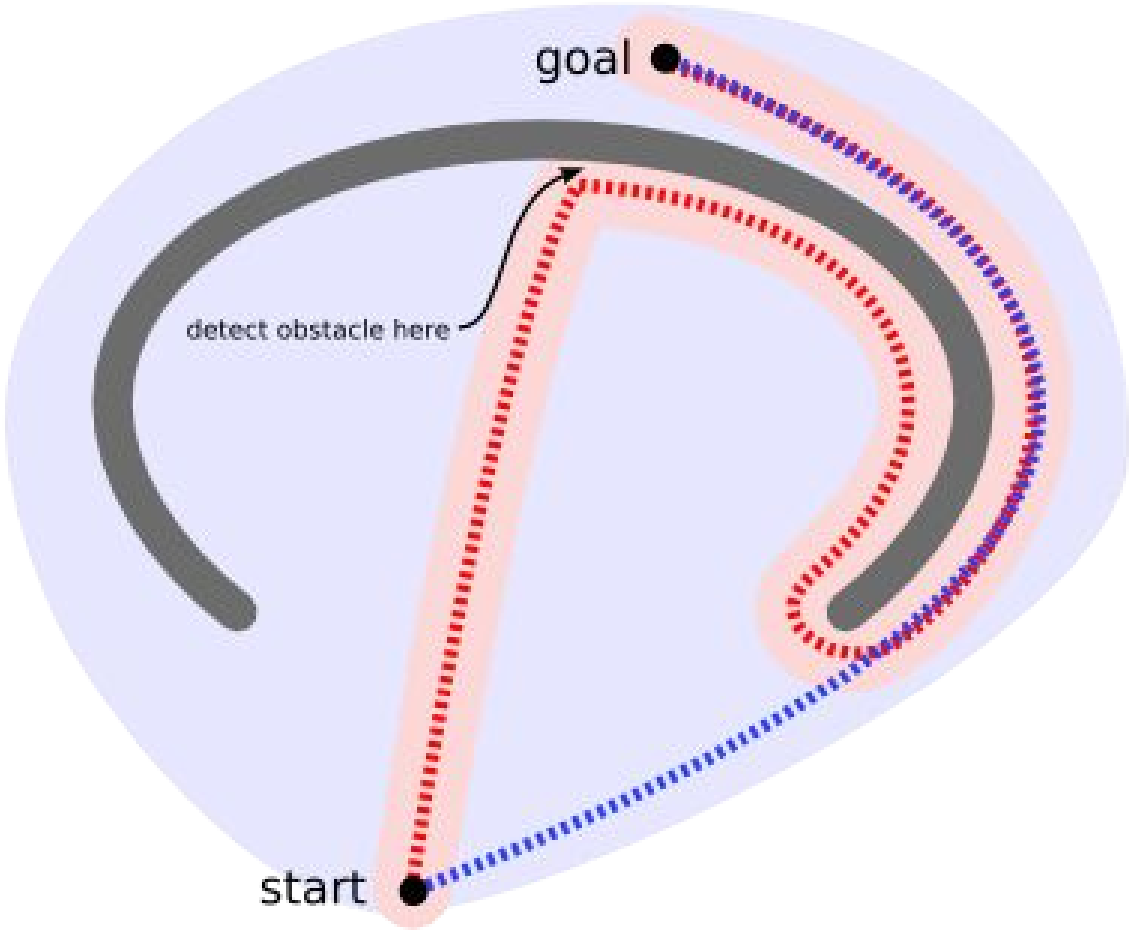


A*

- **Best-First Search ..**



A*

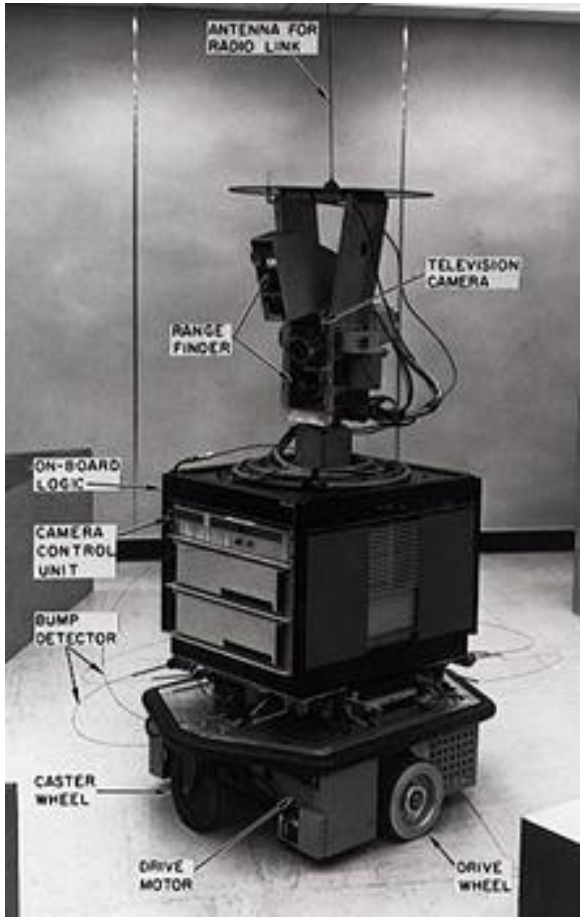



- **Será que existe um modo inteligente e rápido para encontrar o menor caminho ?**

# A*

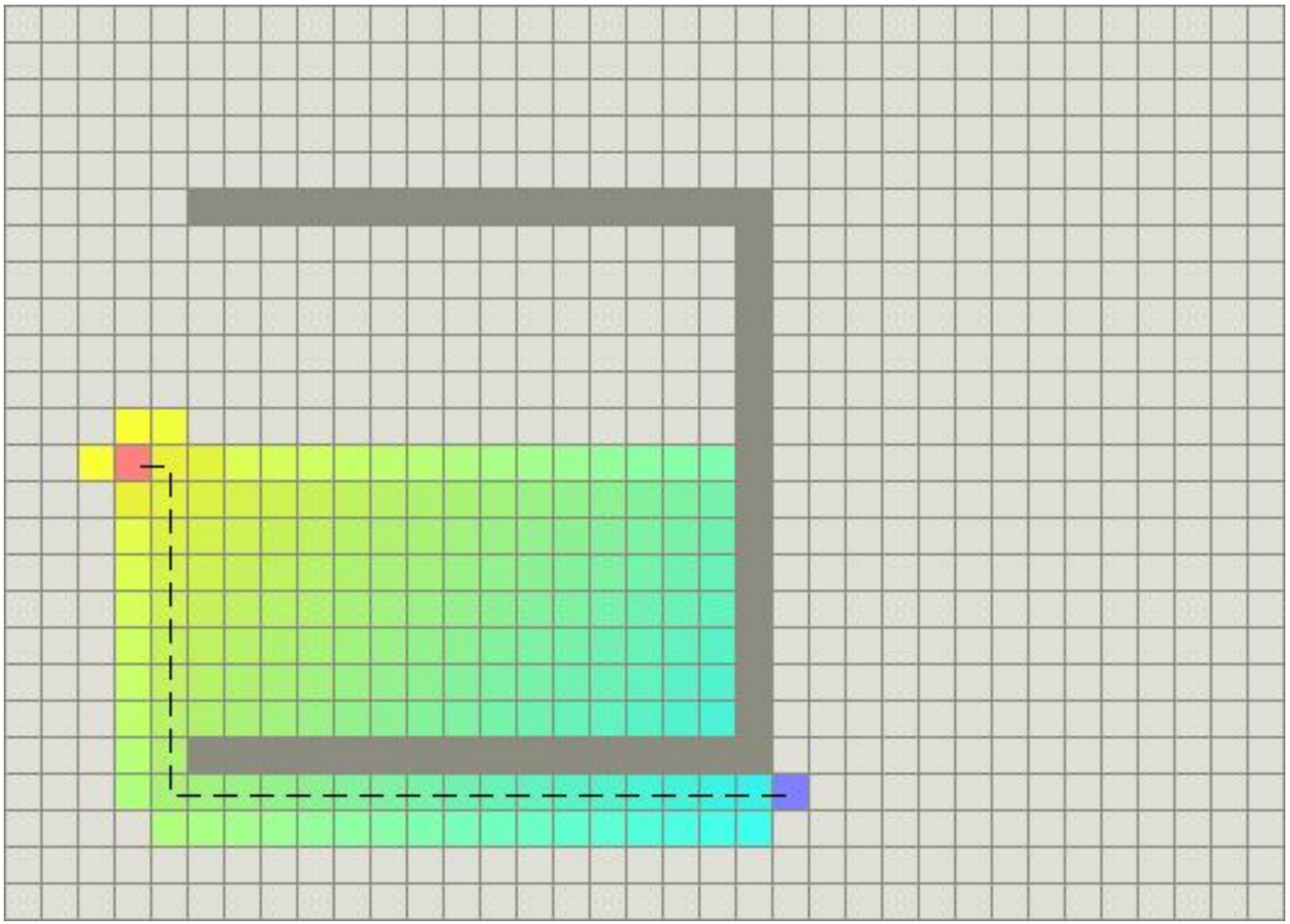
- A* ! Desenvolvido por Nils Nilsson , Bertram Raphael e Peter E. Hart.

- Algoritmo “filho” de Dijkstra e do BFS.



A*

- A* encontra o menor caminho sem perder muito tempo no processo!



# A*

- A cada iteração do loop o algoritmo precisa saber para qual direção ele deve expandir, então um cálculo com base :

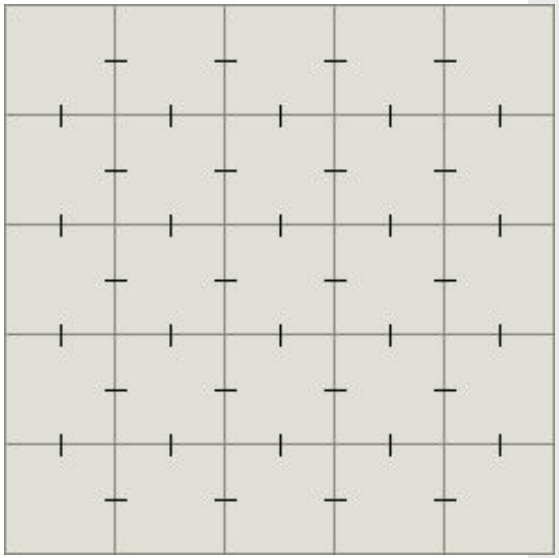
$$F(n) = G(n) + H(n)$$

No qual ‘n’ é o último nó do caminho, G(n) é o custo da posição inicial até o nó corrente e H(n) é a heurística que estima o menor caminho do nó corrente para o nó final.

O vértice que contiver o menor F(n) será o próximo ‘n’.

# A*

- **Pseudocode:**



```
function A*(start, goal)
  open_list = set containing start
  closed_list = empty set
  start.g = 0
  start.f = start.g + heuristic(start, goal)
  while open_list is not empty
    current = open_list element with lowest f cost
    if current = goal
      return construct_path(goal) // path found
    remove current from open_list
    add current to closed_list
    for each neighbor in neighbors(current)
      if neighbor not in closed_list
        neighbor.f = neighbor.g + heuristic(neighbor, goal)
        if neighbor is not in open_list
          add neighbor to open_list
        else
          openneighbor = neighbor in open_list
          if neighbor.g < openneighbor.g
            openneighbor.g = neighbor.g
            openneighbor.parent = neighbor.parent
  return false // no path exists

function neighbors(node)
  neighbors = set of valid neighbors to node // check for obstacles here
  for each neighbor in neighbors
    if neighbor is diagonal
      neighbor.g = node.g + diagonal_cost // eg. 1.414 (pythagoras)
    else
      neighbor.g = node.g + normal_cost // eg. 1
    neighbor.parent = node
  return neighbors

function construct_path(node)
  path = set containing node
  while node.parent exists
    node = node.parent
    add node to path
  return path
```

A*

- **Aplicações:**