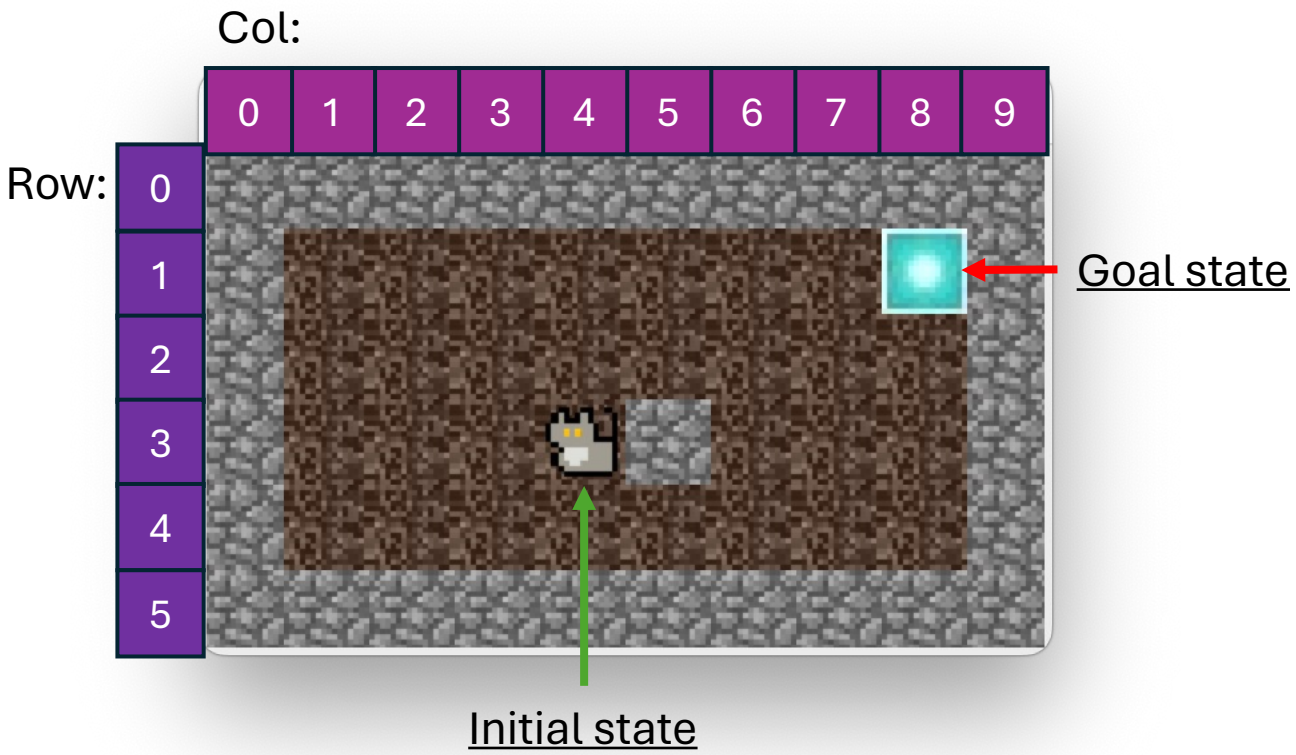
2-D Maze Example

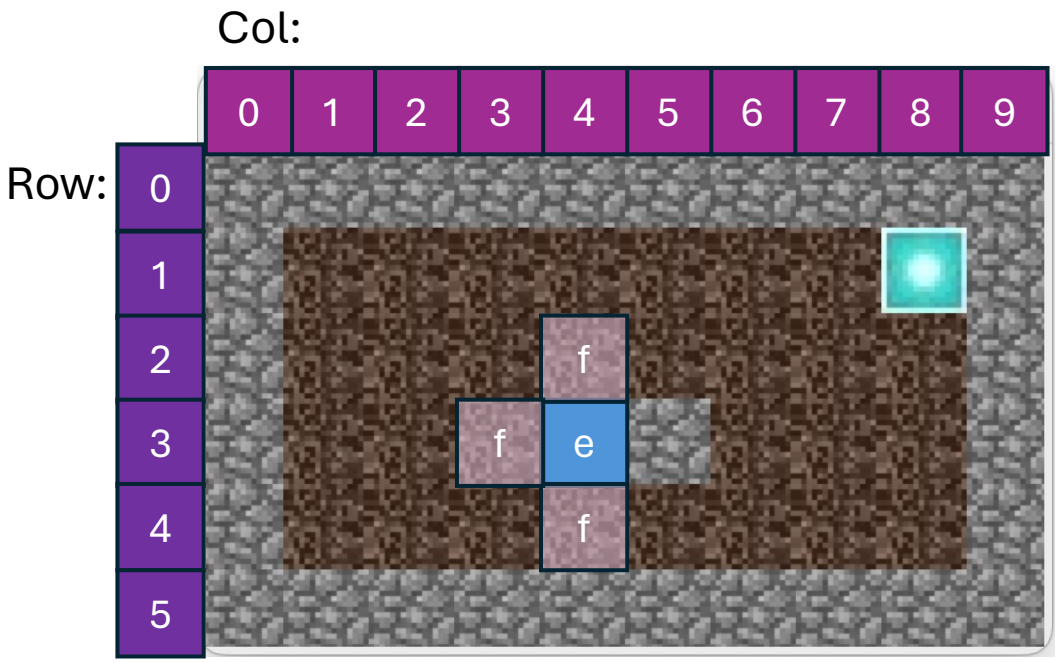


Key:

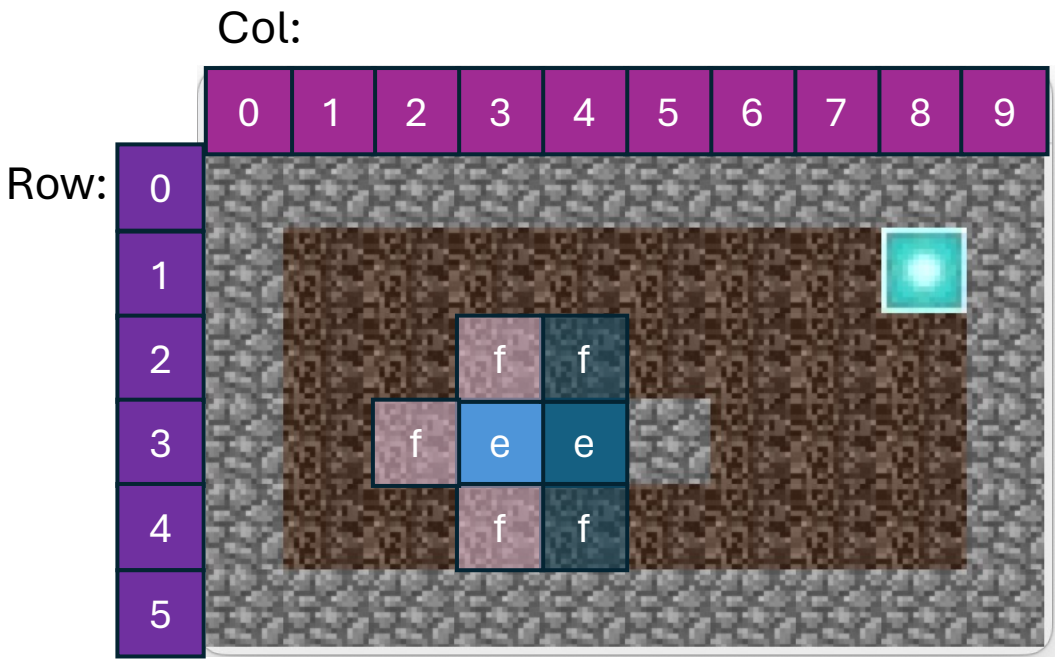
e — Expanded node

f — Node on the frontier

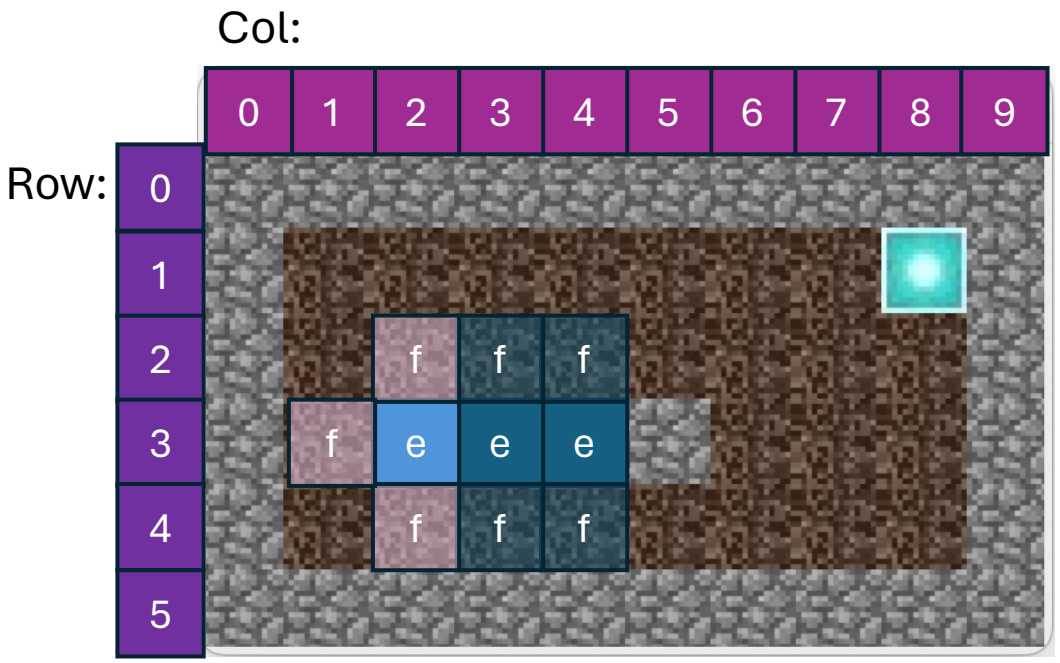
DFS



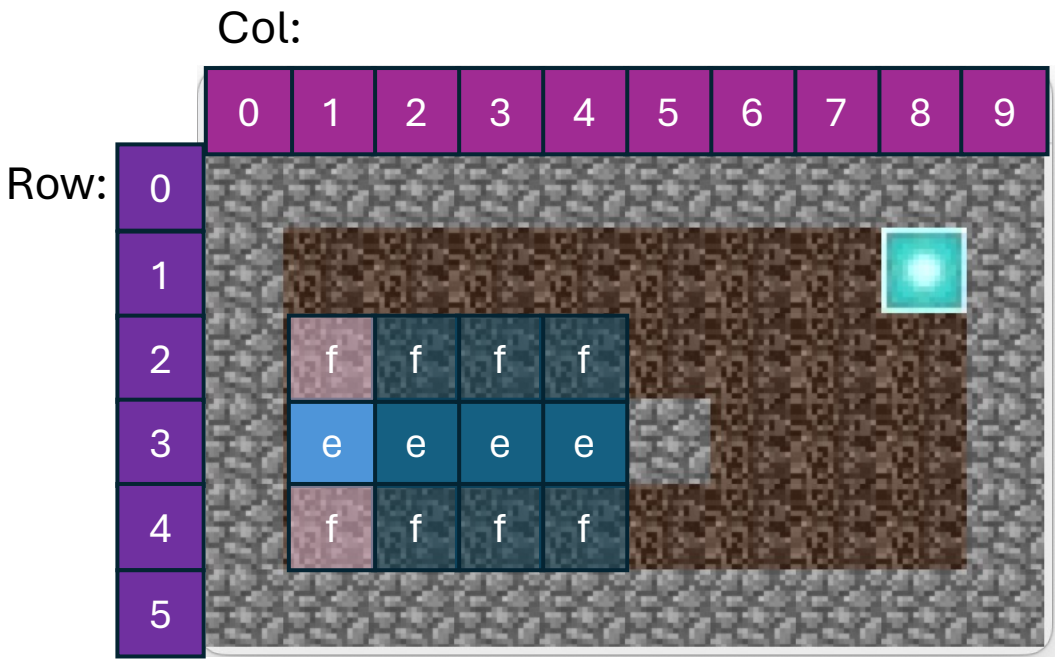
DFS



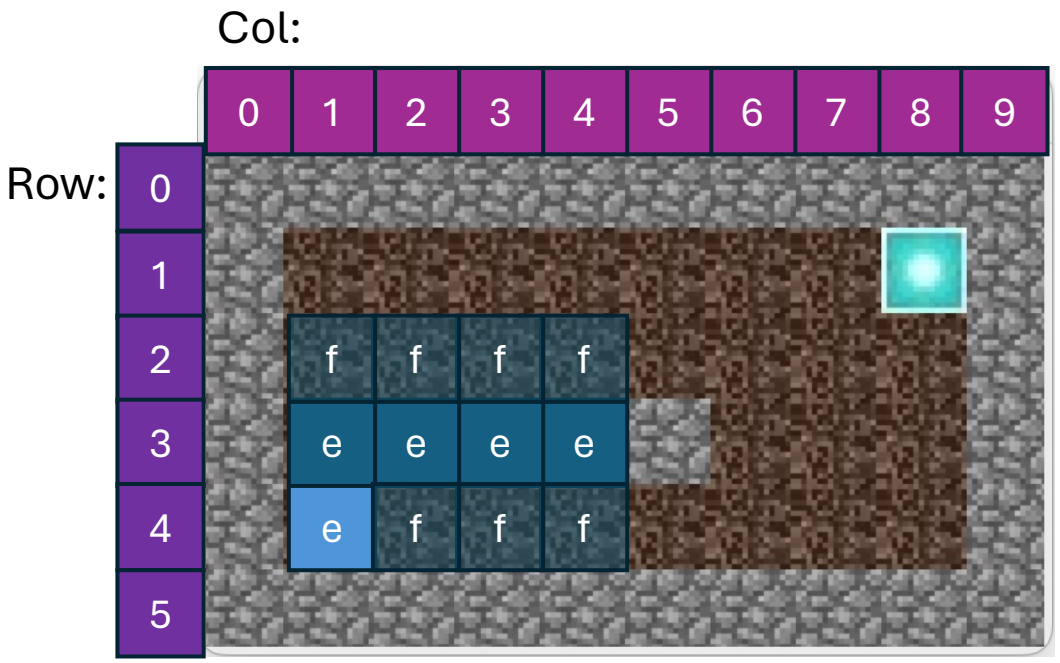
DFS



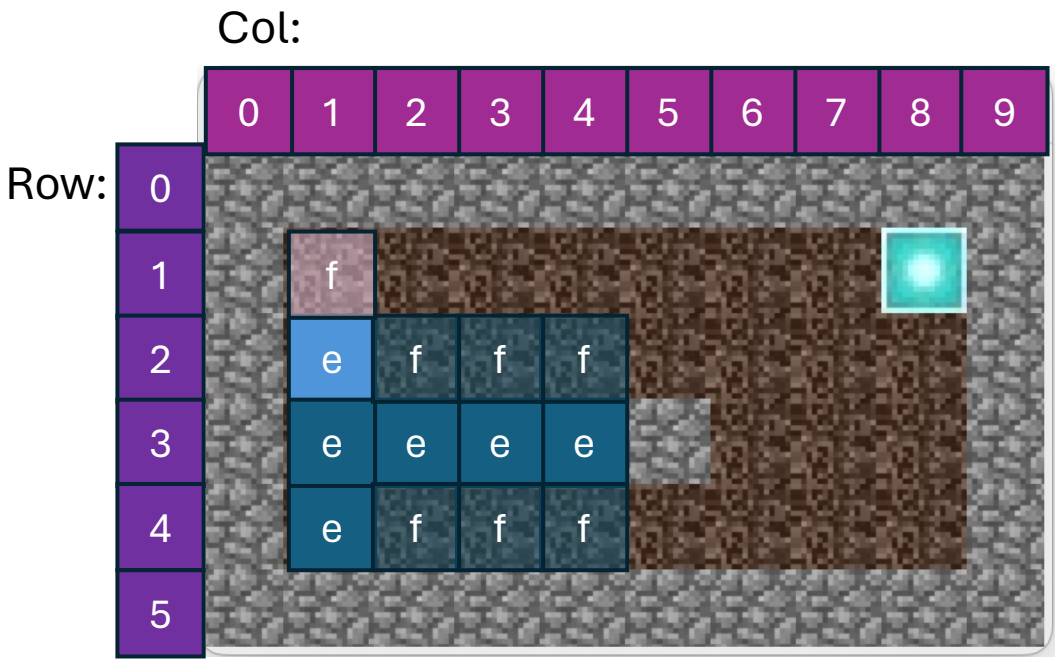
DFS



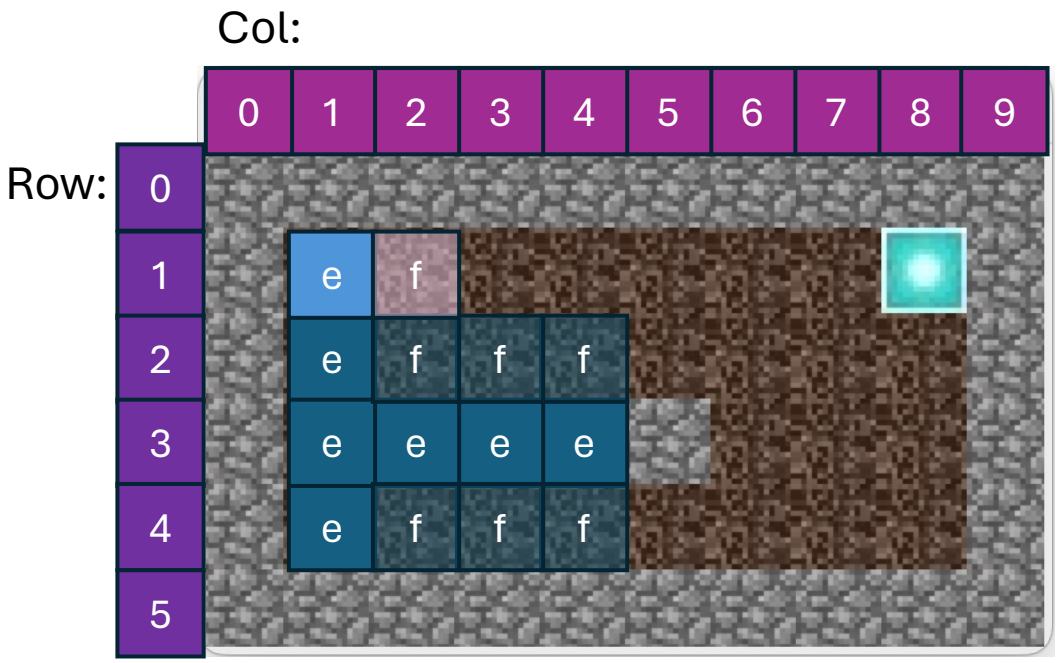
DFS



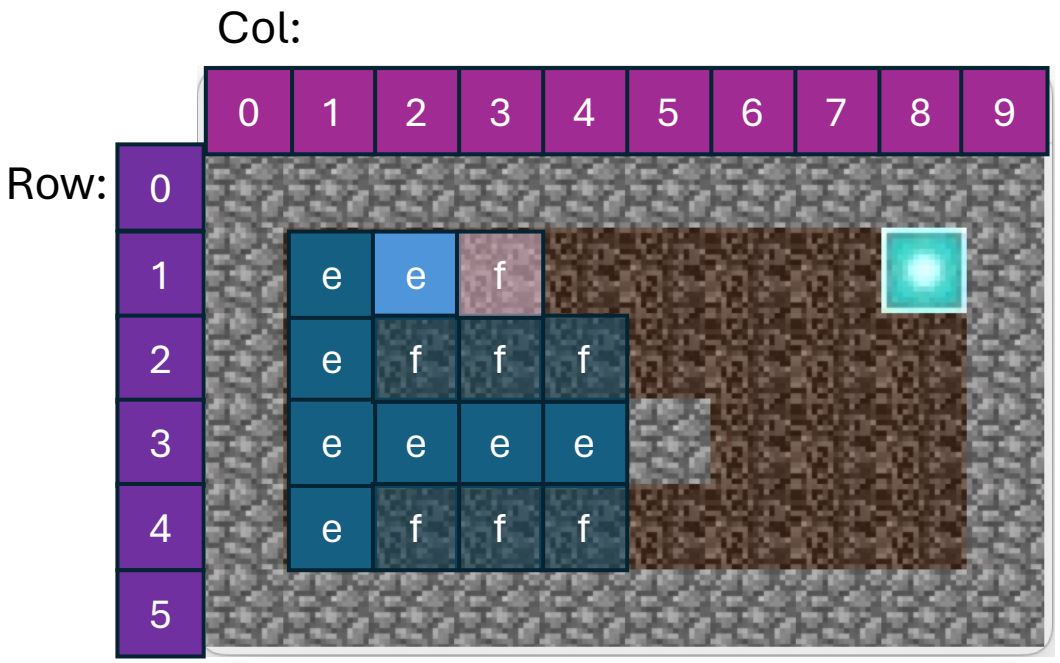
DFS



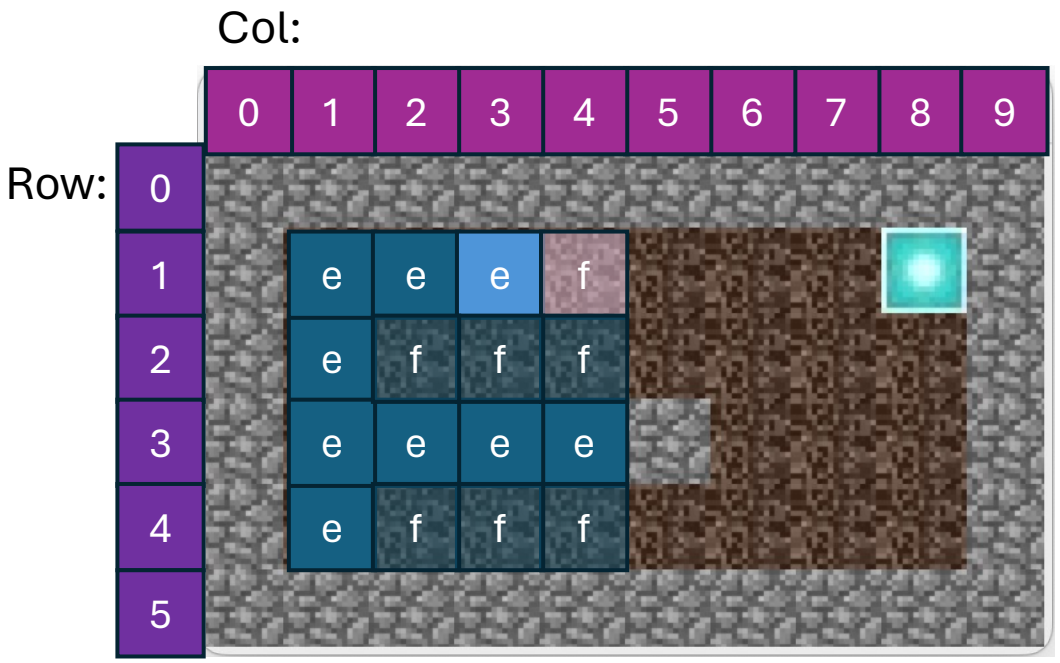
DFS



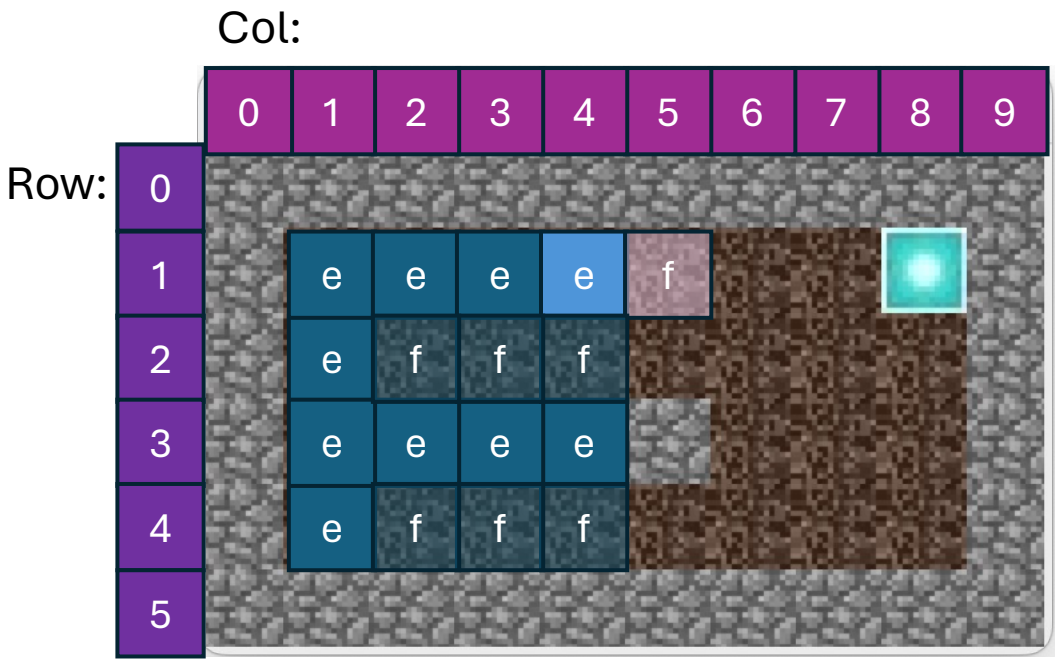
DFS



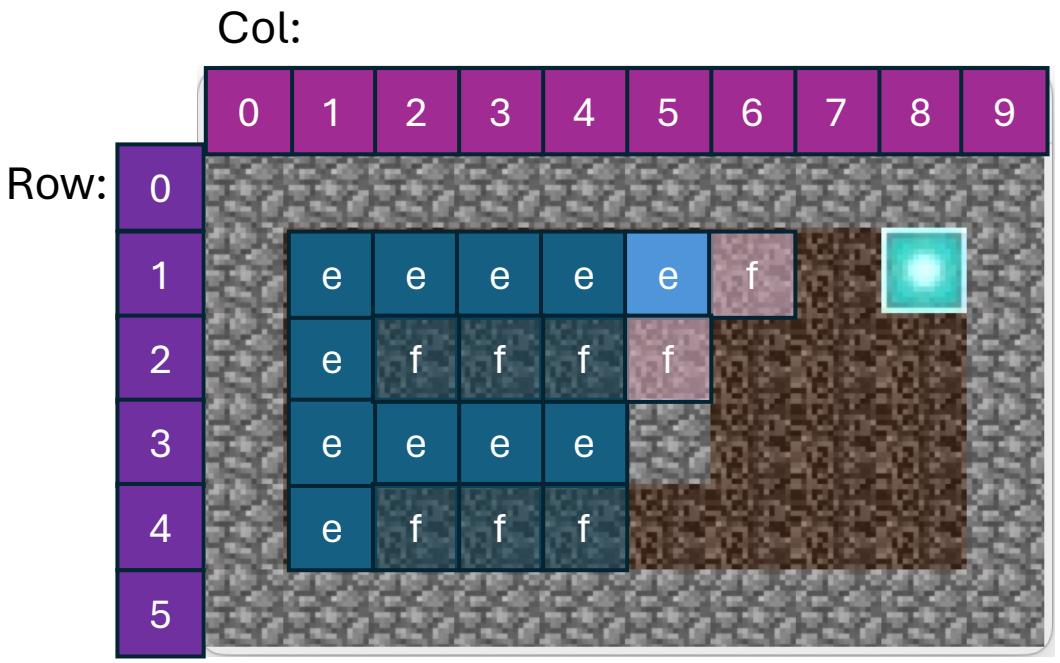
DFS



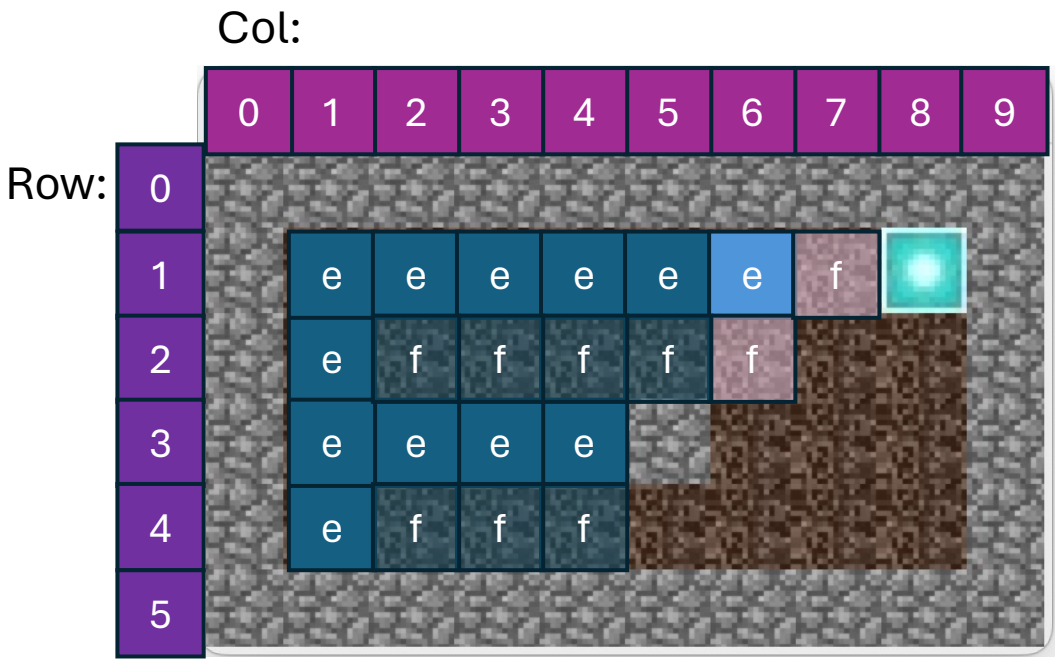
DFS



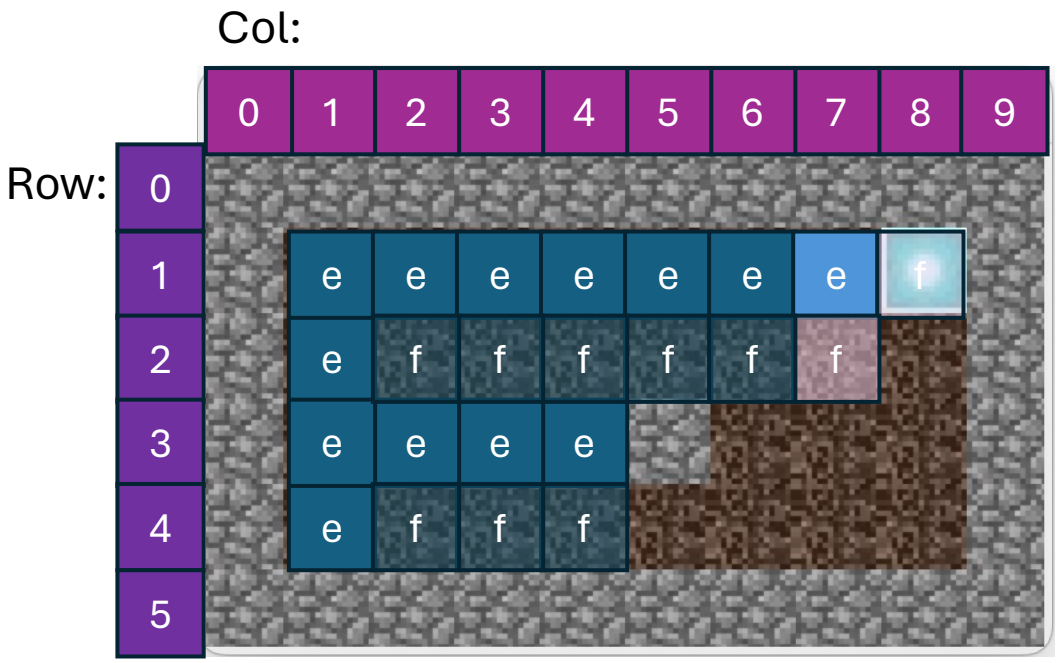
DFS



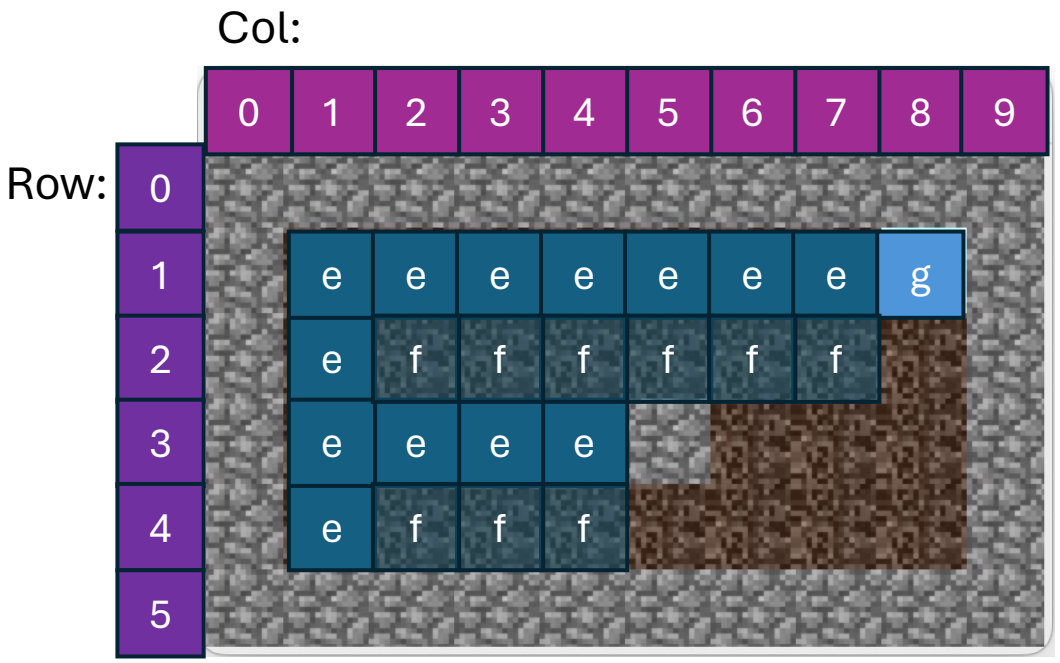
DFS



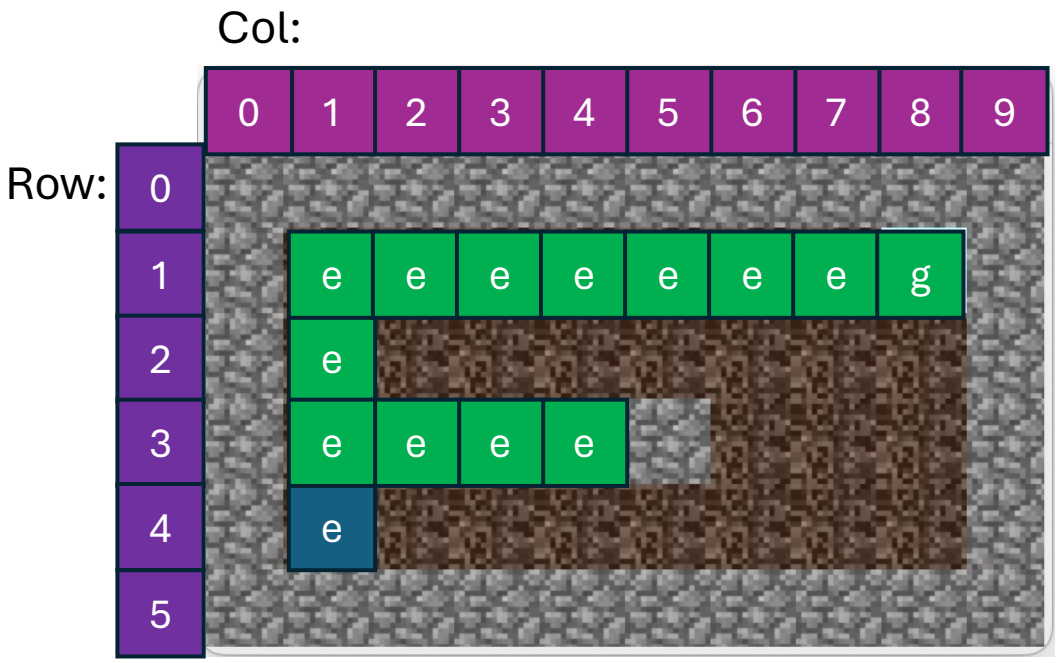
DFS



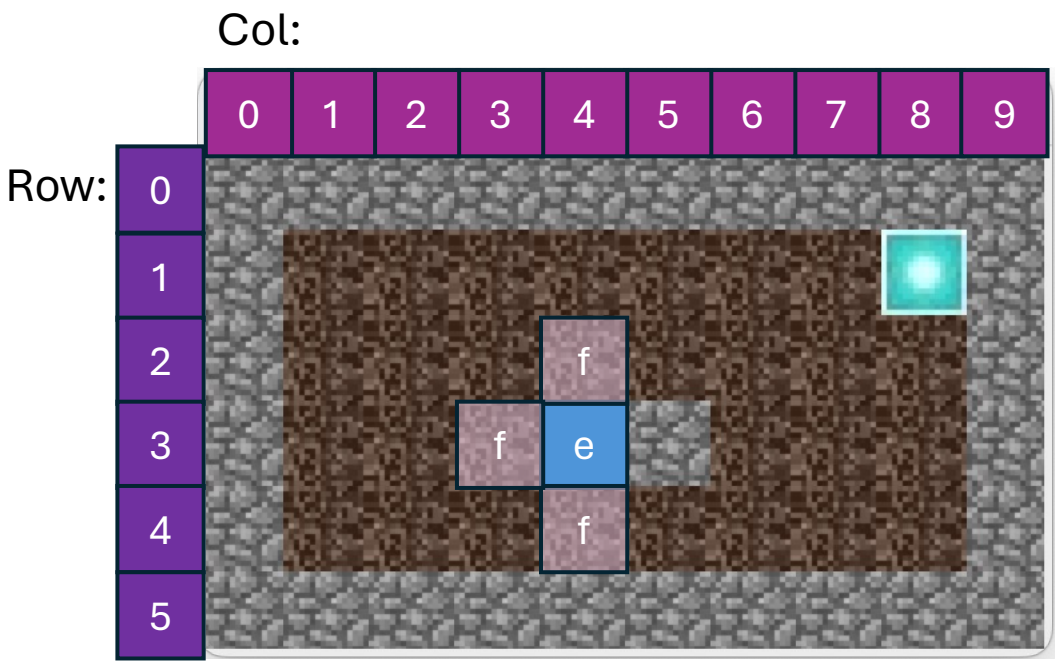
DFS



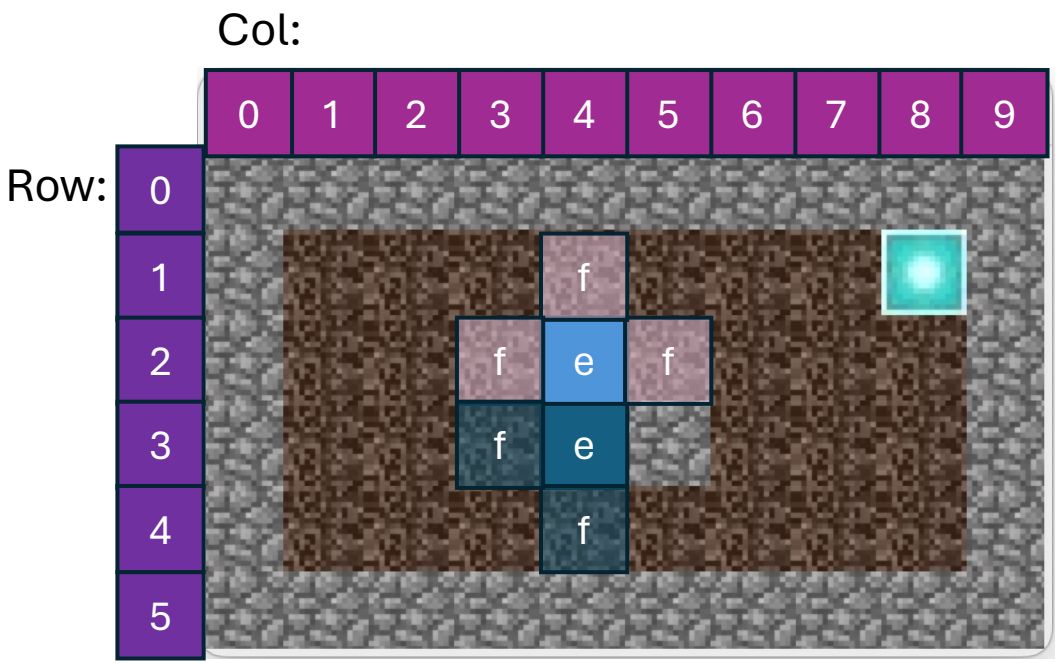
DFS



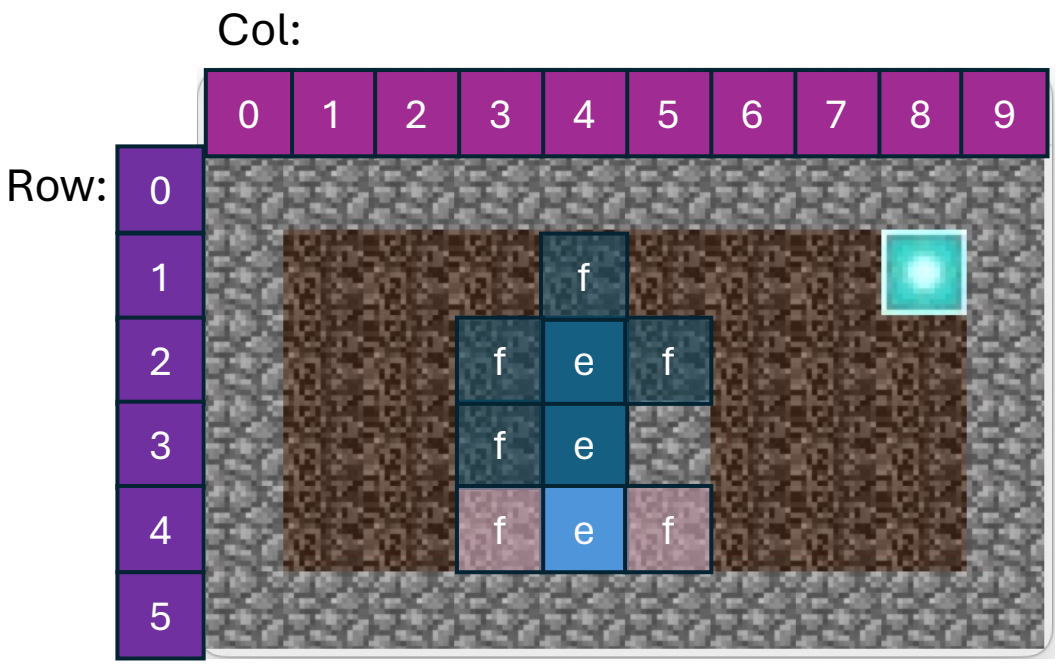
BFS/UCS with 1 cost per action



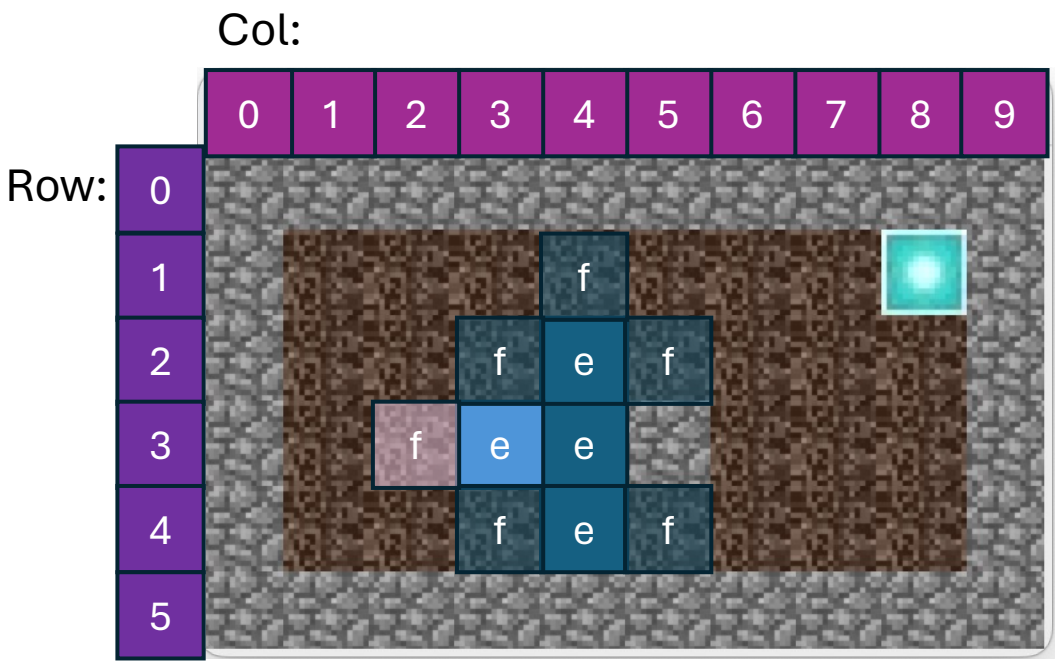
BFS/UCS with 1 cost per action



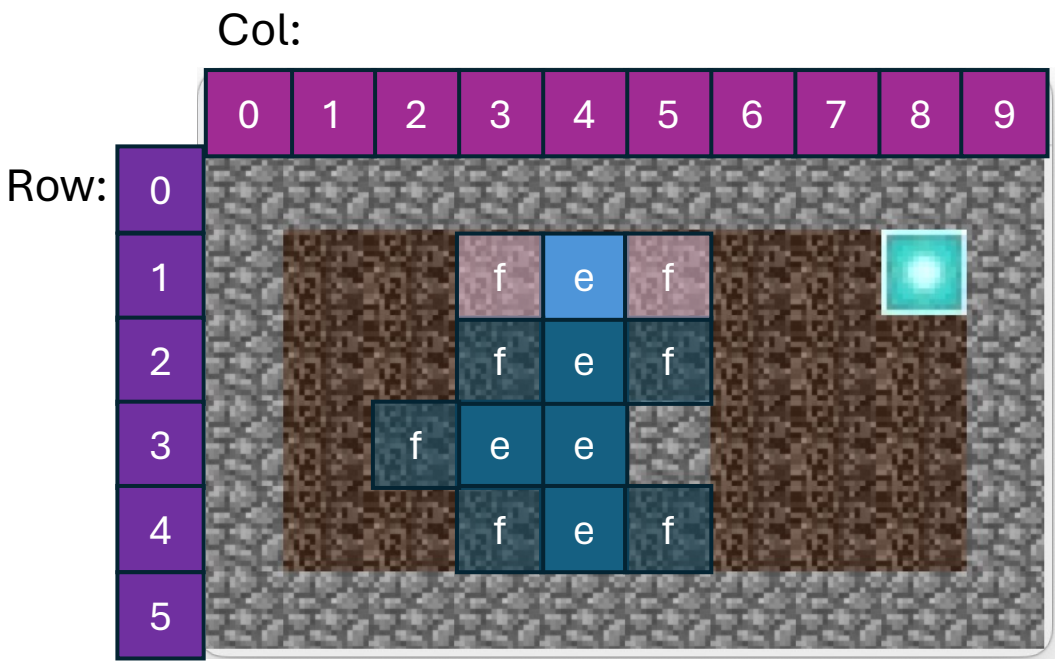
BFS/UCS with 1 cost per action



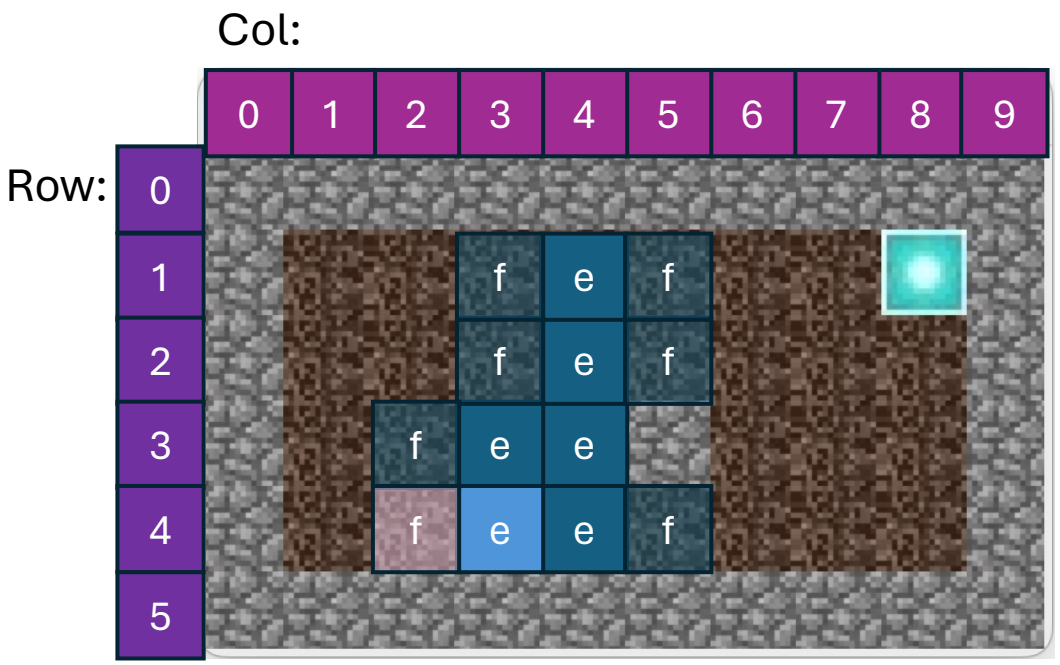
BFS/UCS with 1 cost per action



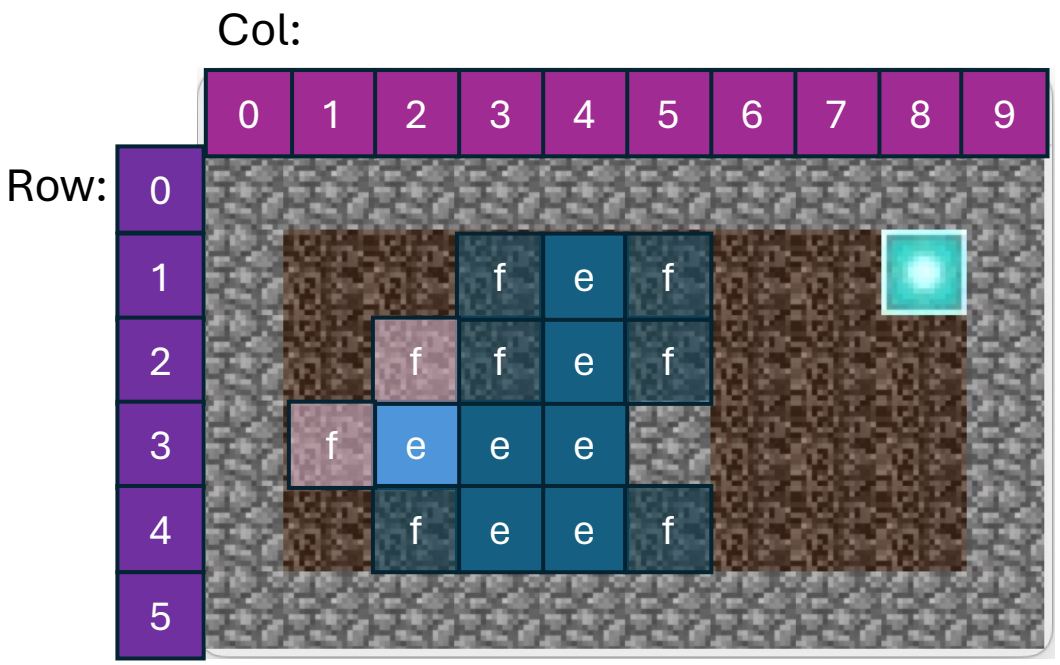
BFS/UCS with 1 cost per action



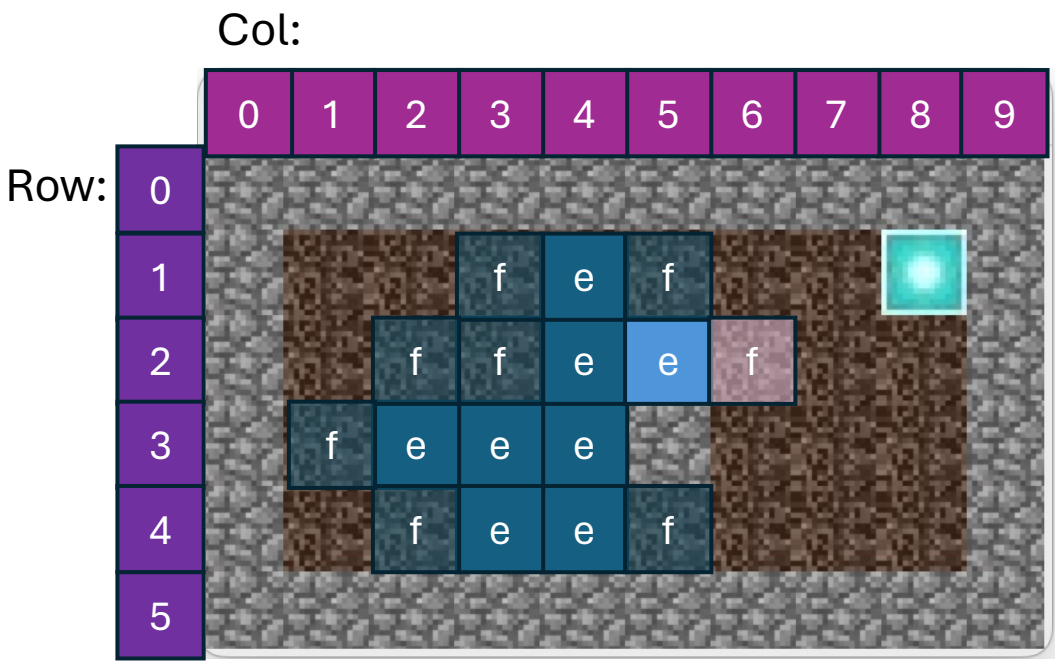
BFS/UCS with 1 cost per action



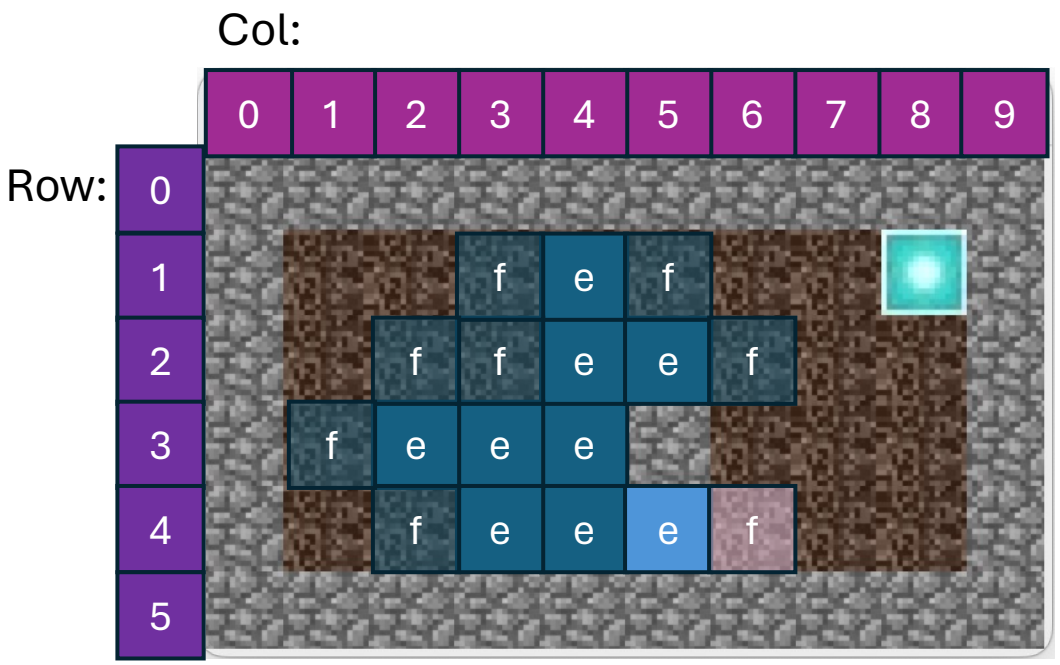
BFS/UCS with 1 cost per action



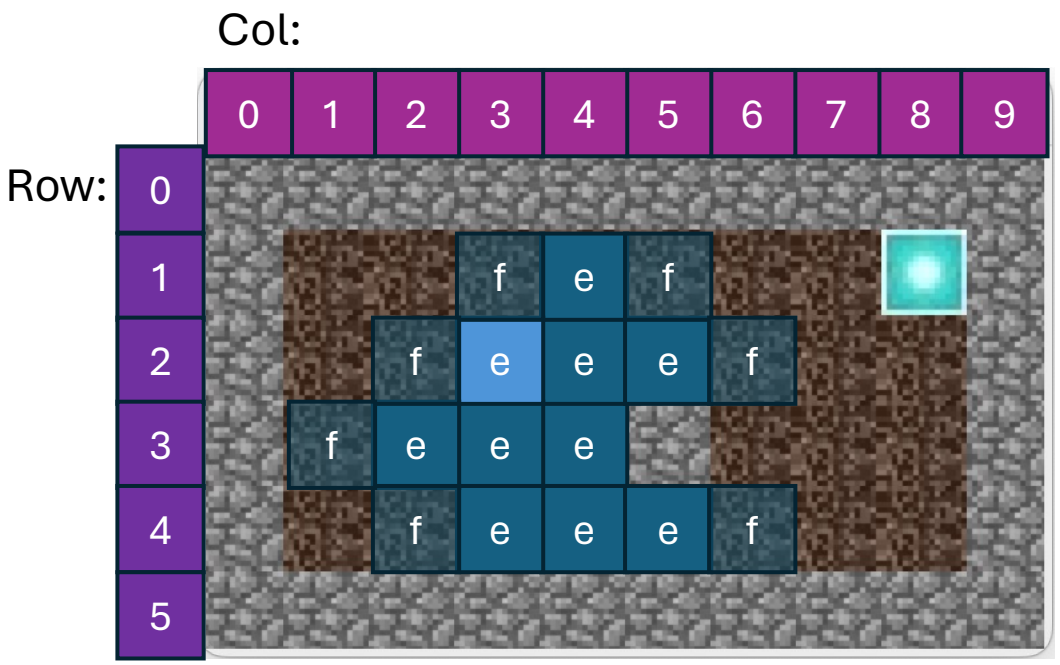
BFS/UCS with 1 cost per action



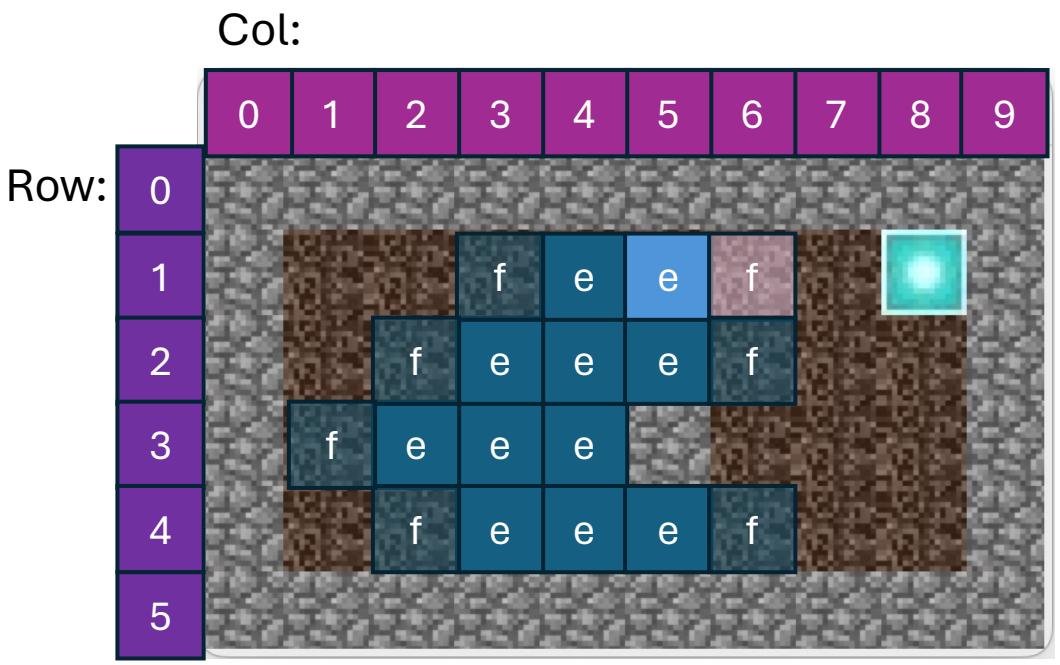
BFS/UCS with 1 cost per action



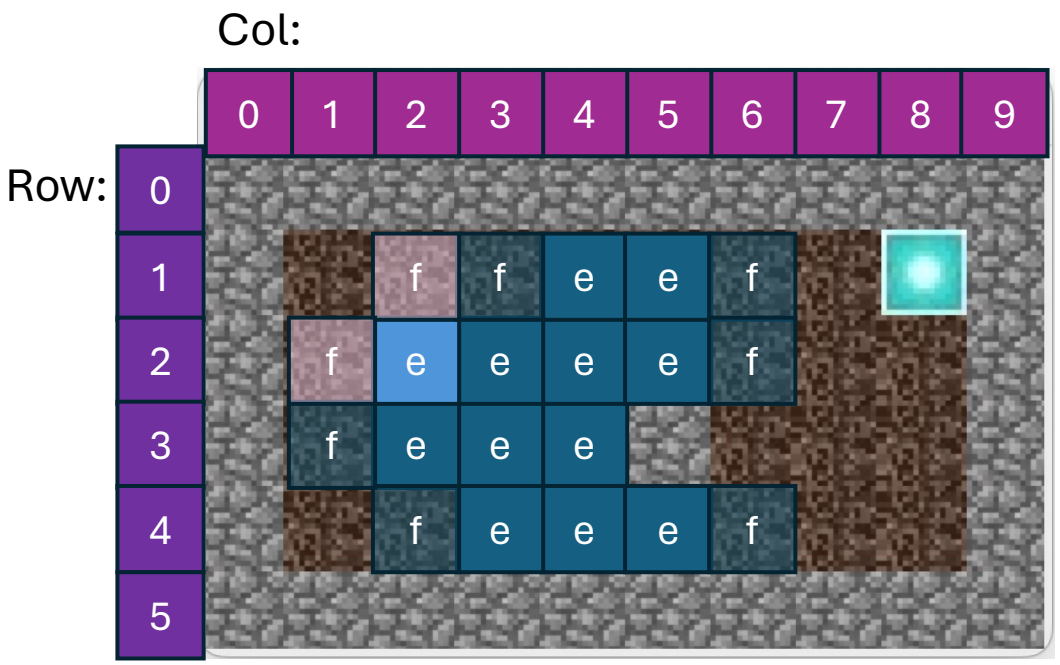
BFS/UCS with 1 cost per action



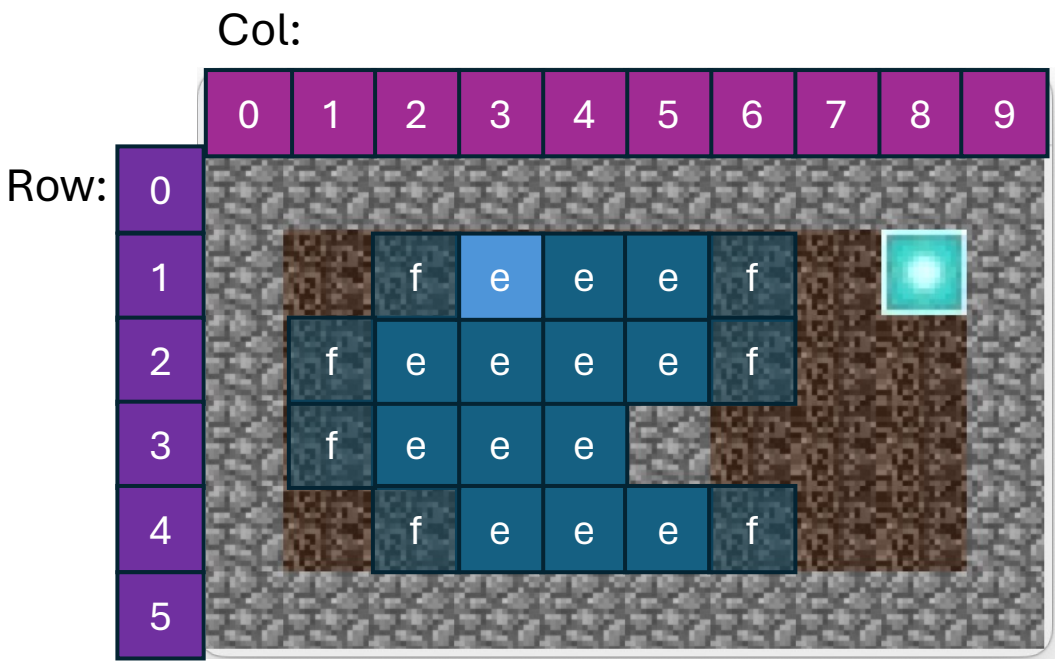
BFS/UCS with 1 cost per action



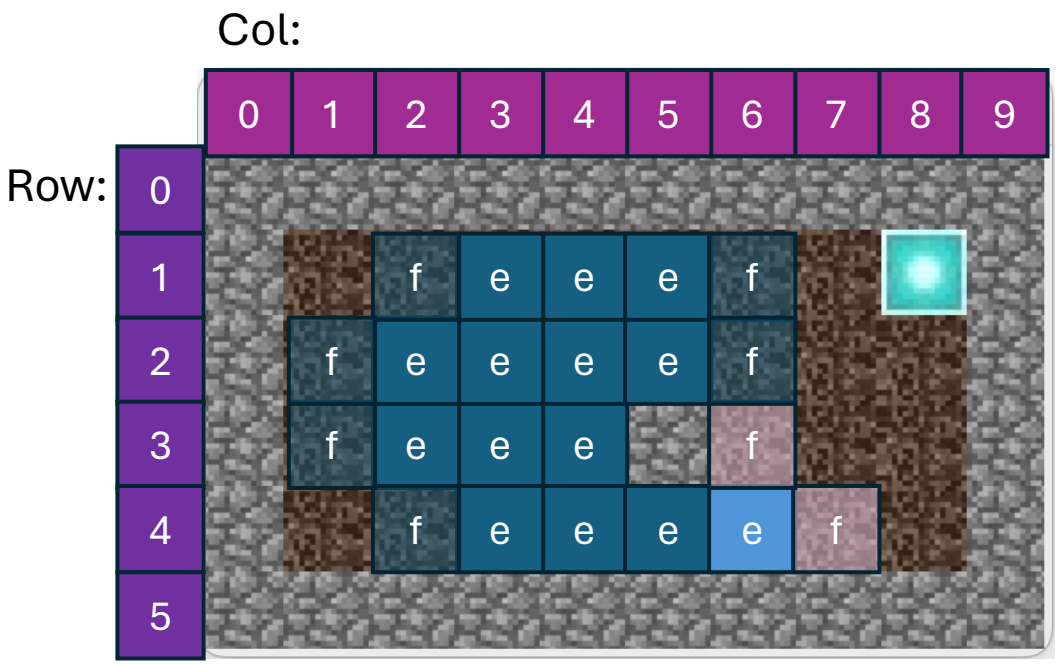
BFS/UCS with 1 cost per action



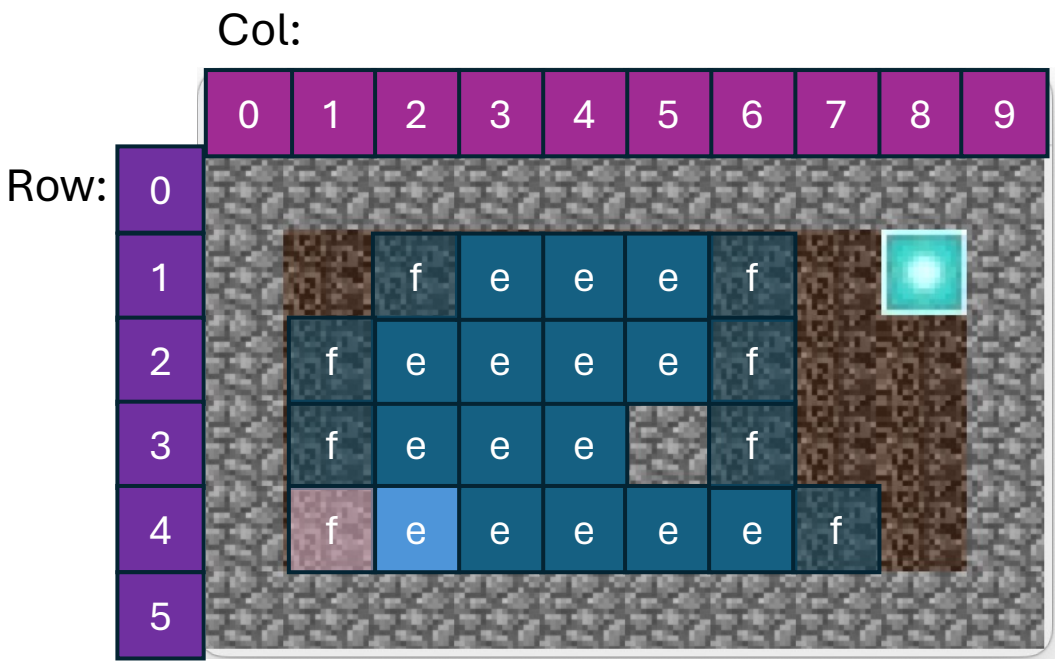
BFS/UCS with 1 cost per action



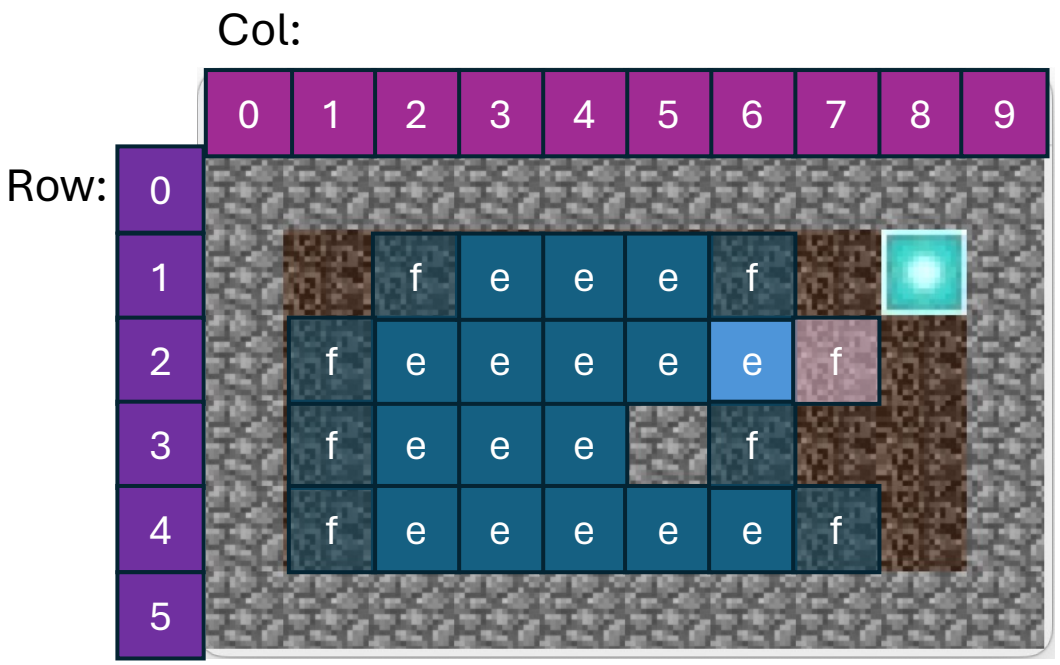
BFS/UCS with 1 cost per action



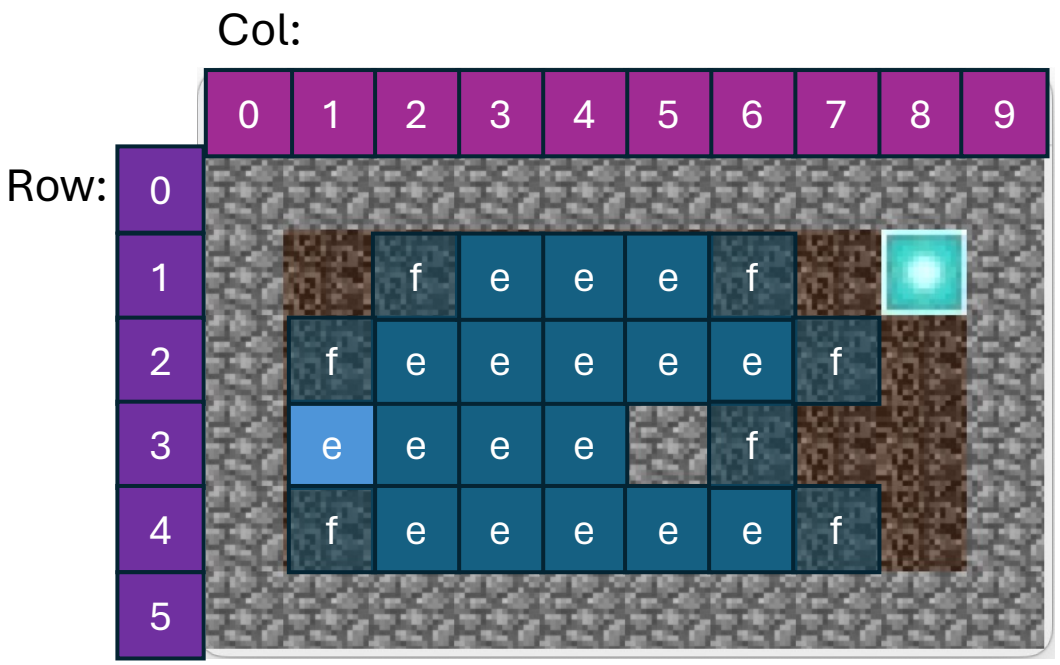
BFS/UCS with 1 cost per action



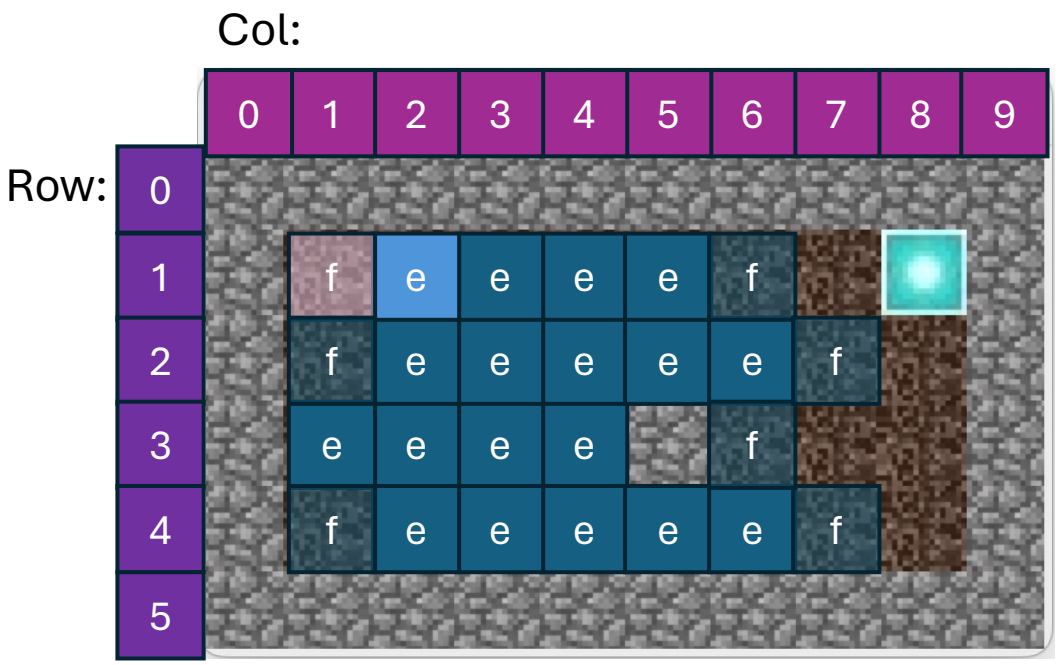
BFS/UCS with 1 cost per action



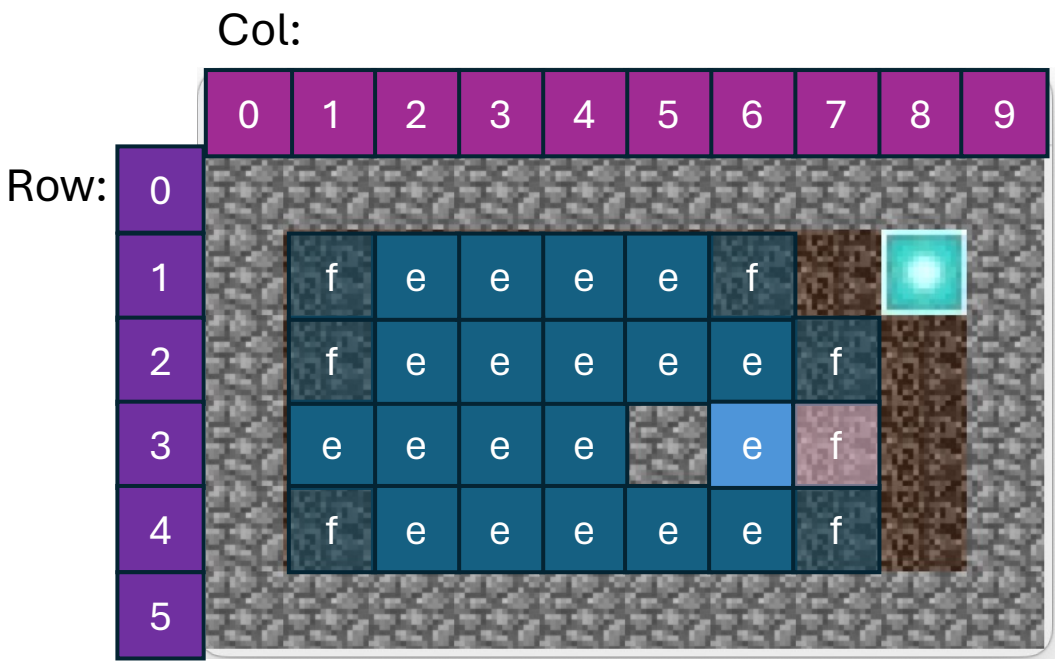
BFS/UCS with 1 cost per action



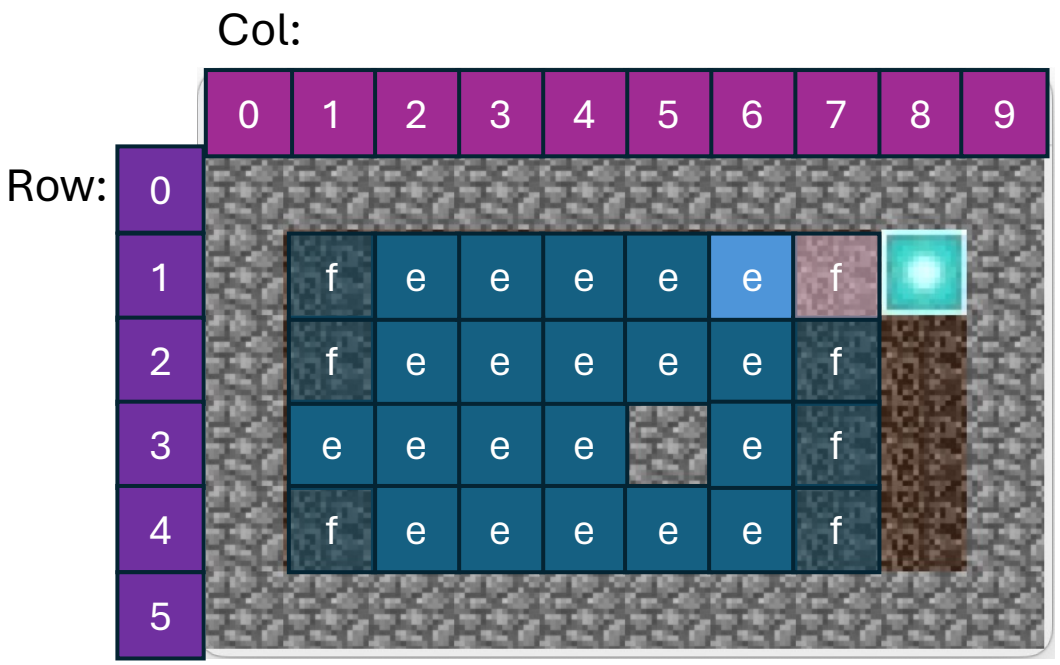
BFS/UCS with 1 cost per action



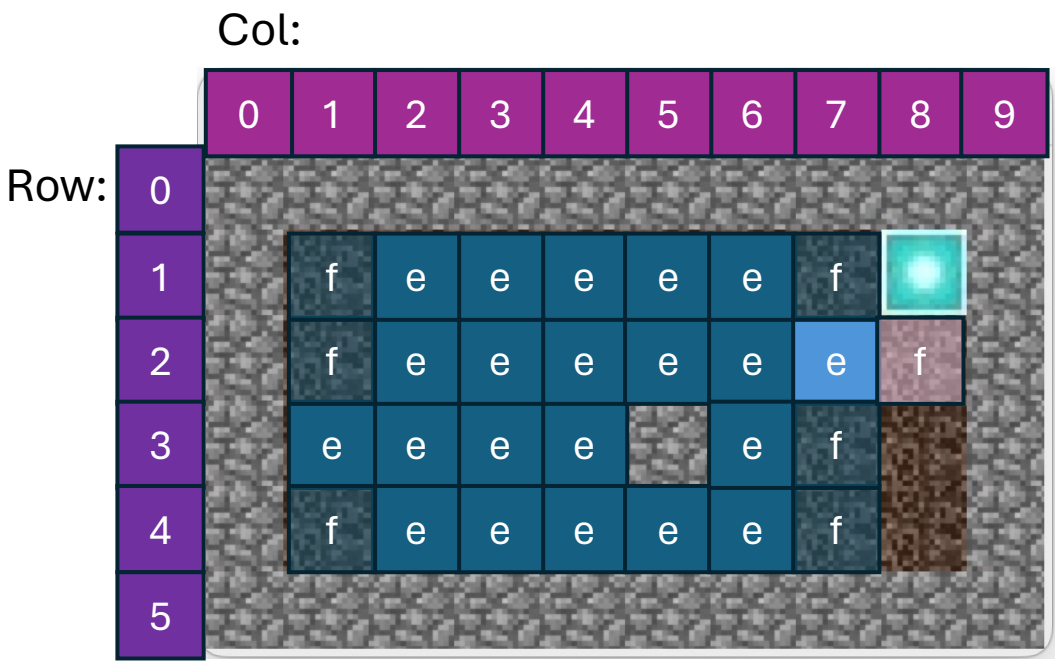
BFS/UCS with 1 cost per action



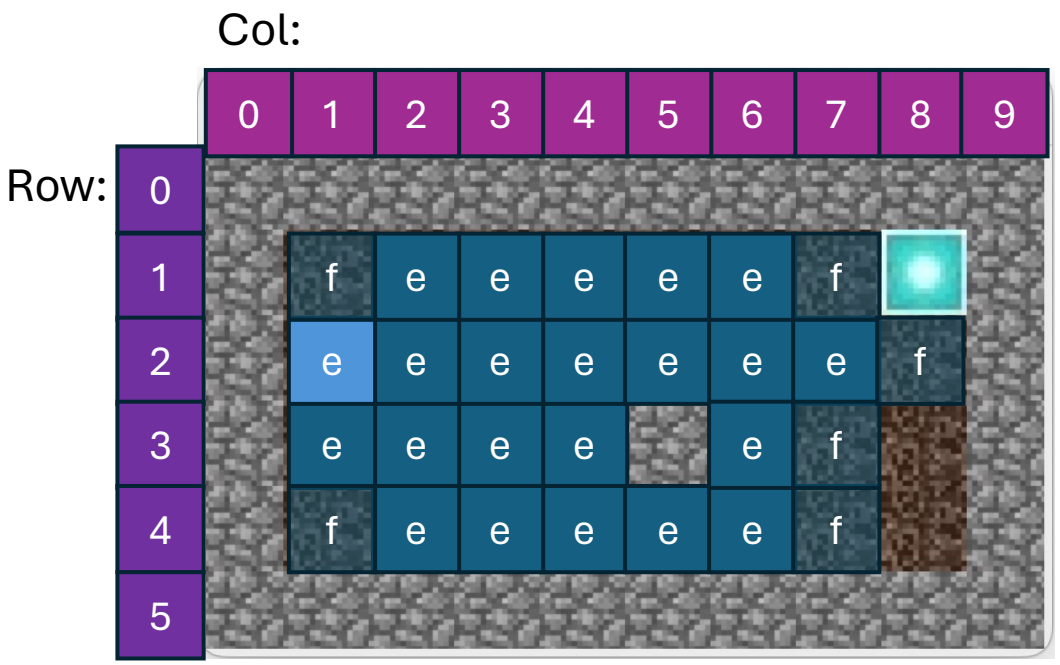
BFS/UCS with 1 cost per action



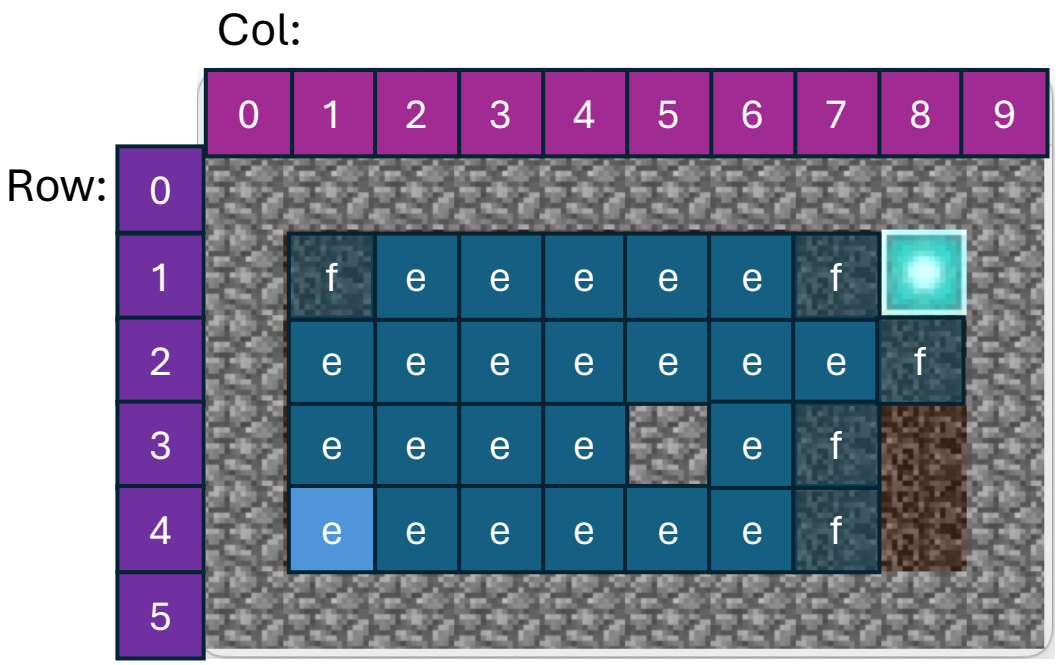
BFS/UCS with 1 cost per action



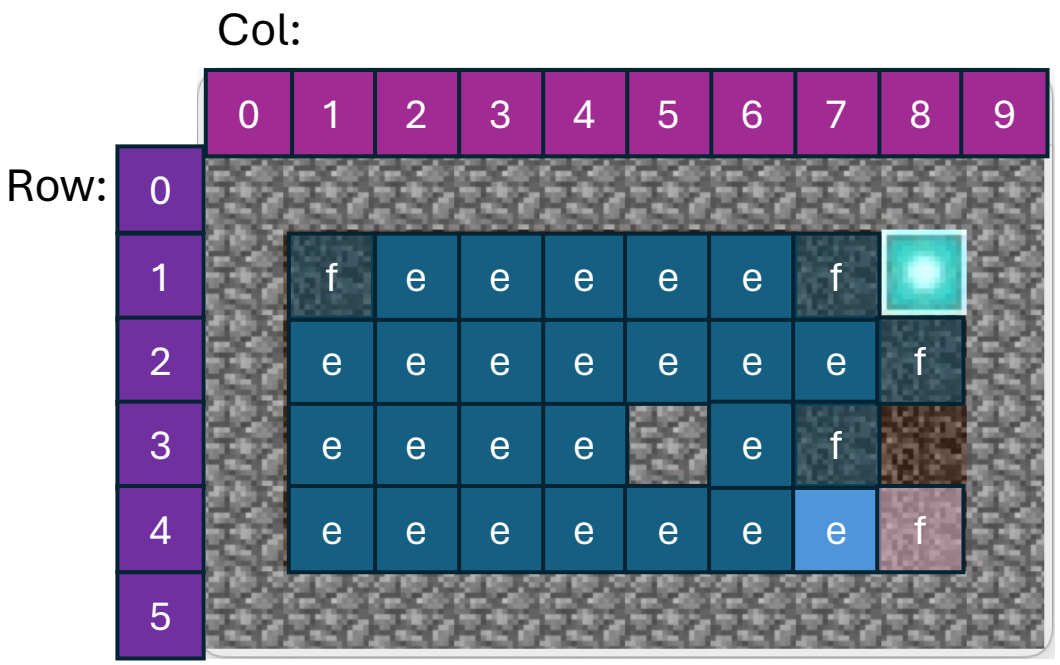
BFS/UCS with 1 cost per action



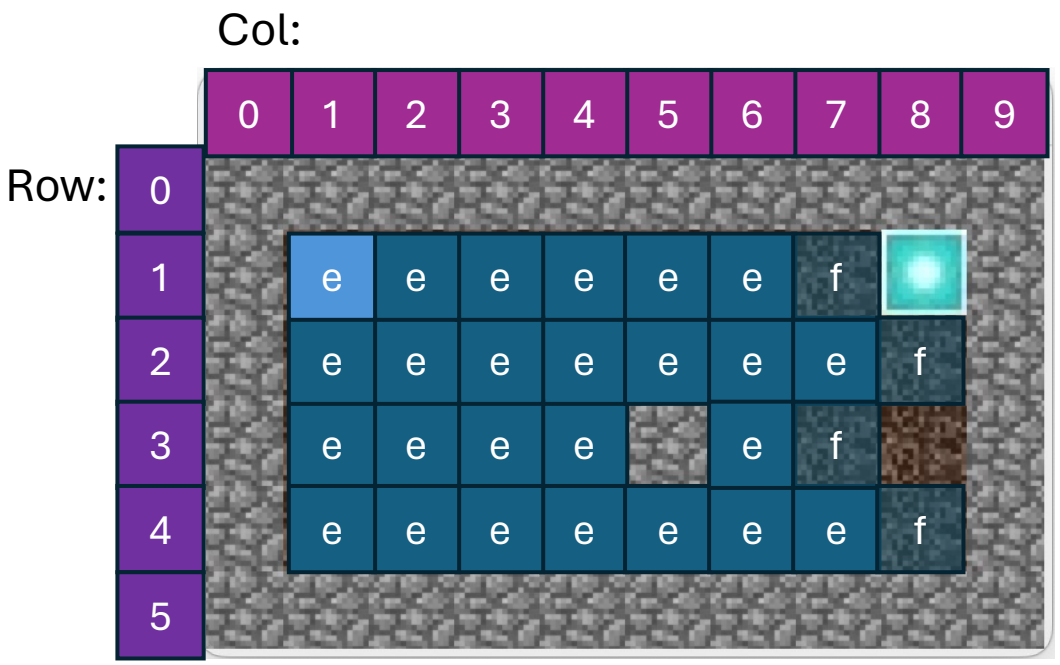
BFS/UCS with 1 cost per action



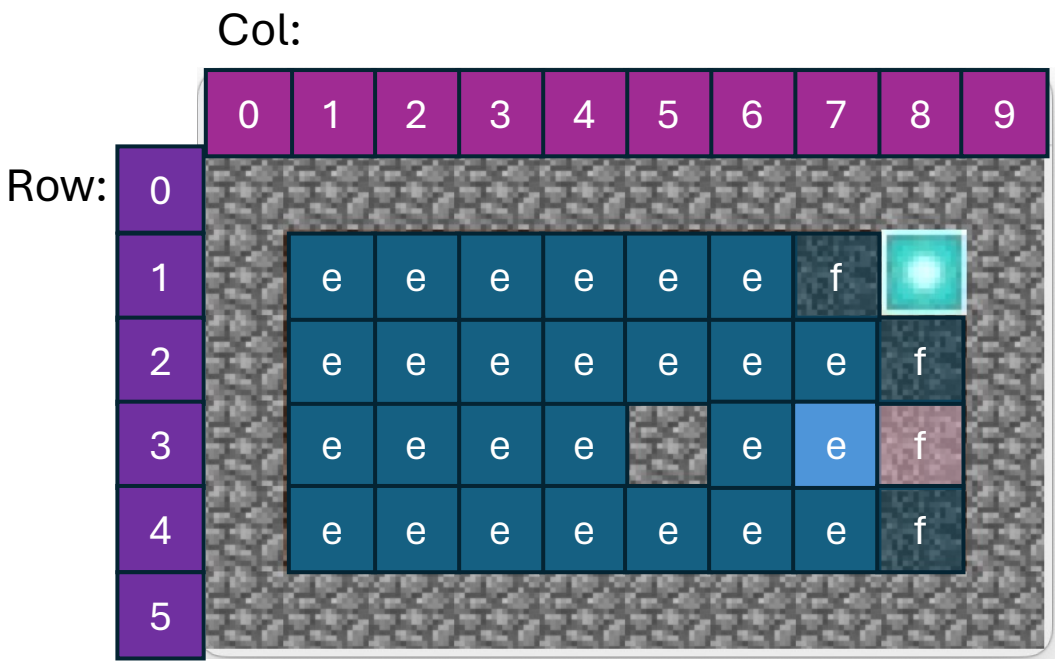
BFS/UCS with 1 cost per action



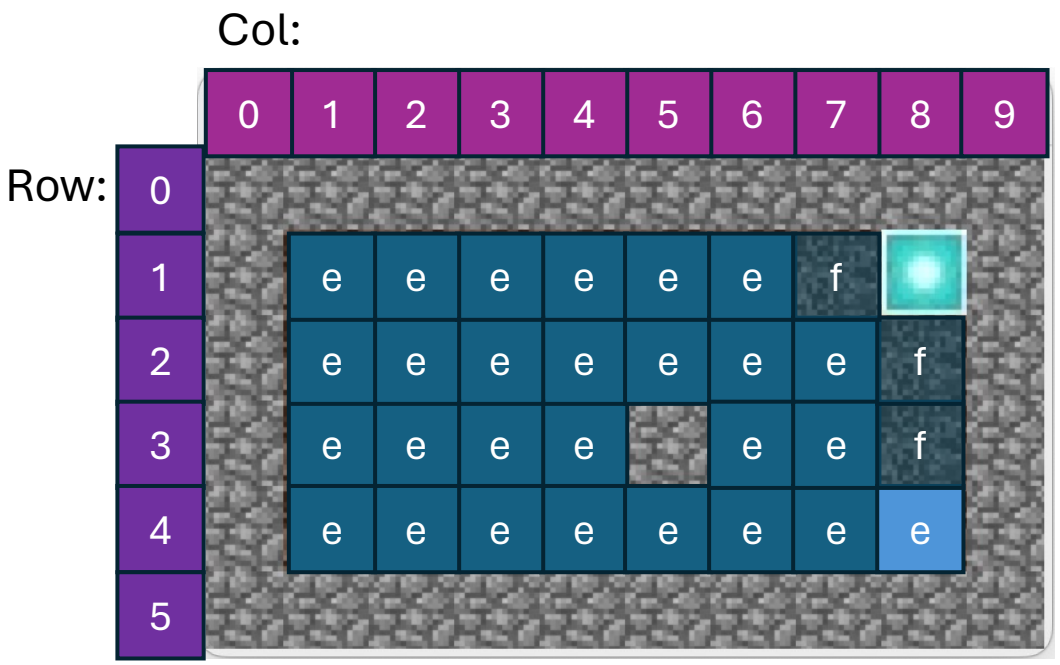
BFS/UCS with 1 cost per action



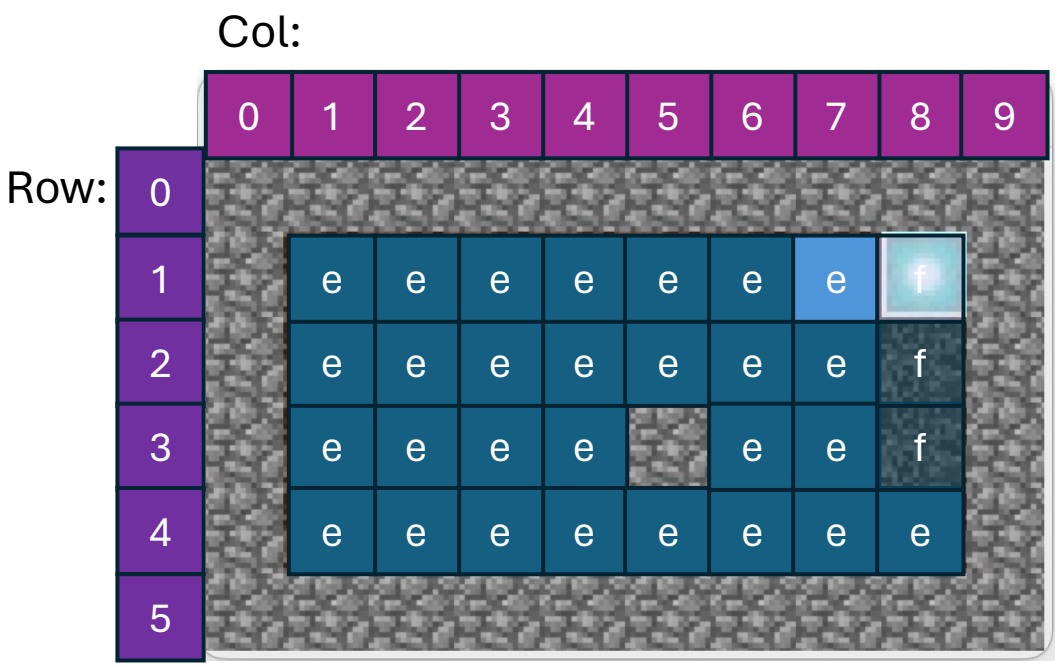
BFS/UCS with 1 cost per action



BFS/UCS with 1 cost per action



BFS/UCS with 1 cost per action



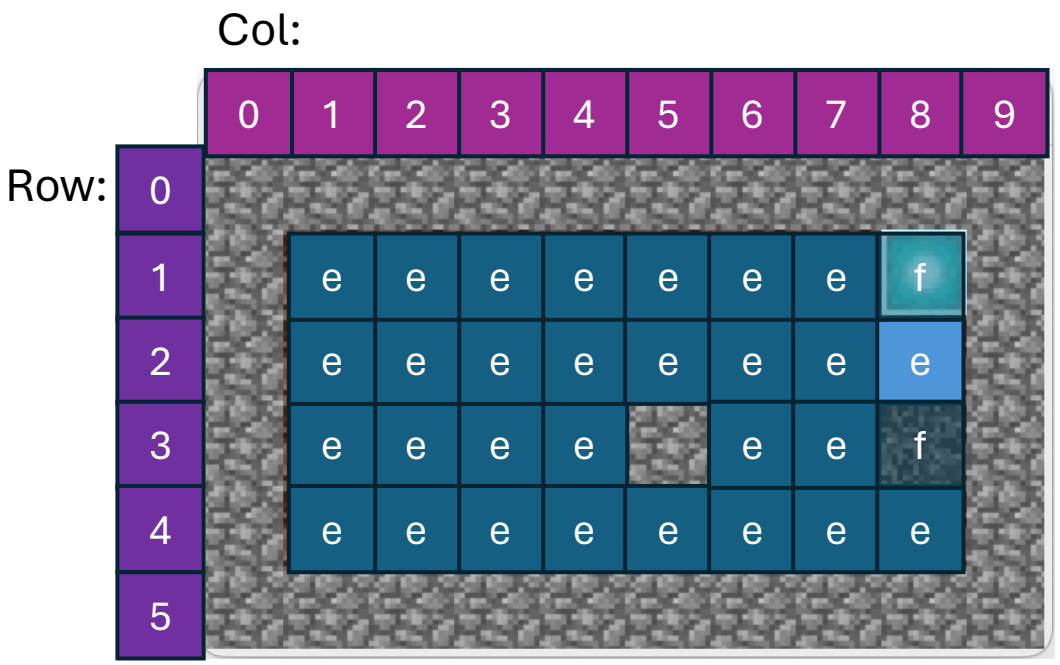
In **BFS**, the goal check is performed on node **generation**, so BFS will find the goal at this step. **UCS** performs the goal check on node **expansion** (i.e. when the node is selected from the frontier), so continues searching.

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure

function UNIFORM-COST-SEARCH(problem) returns a solution node, or failure
  return BEST-FIRST-SEARCH(problem, PATH-COST)
```

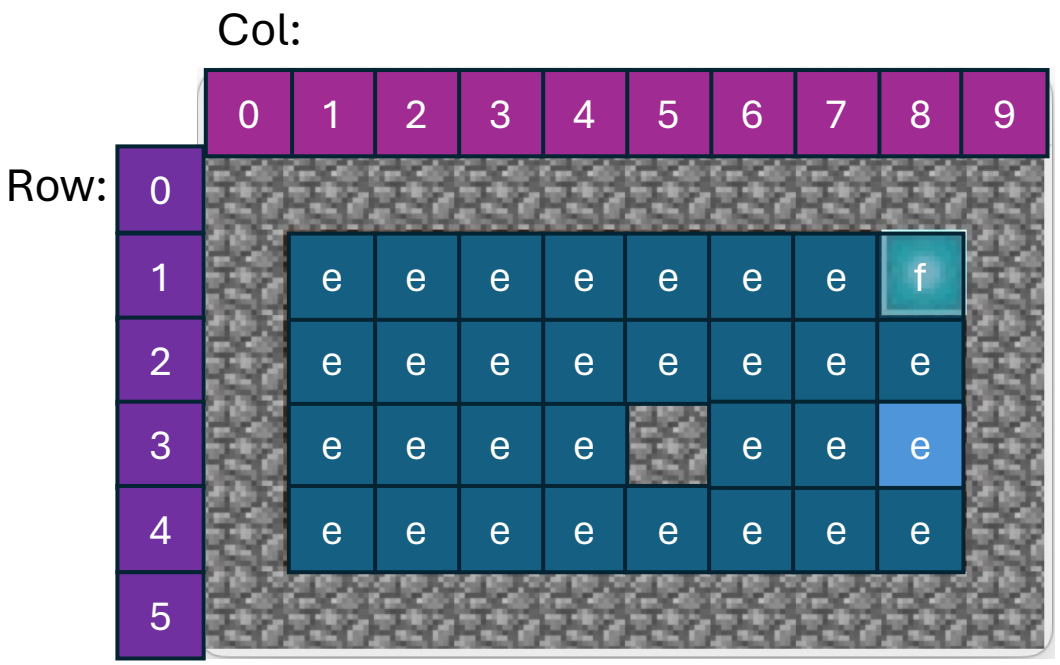
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Figure 3.7 The best-first search algorithm, and the function for expanding a node. The data structures used here are described in Section 3.3.2. See Appendix B for **yield**.

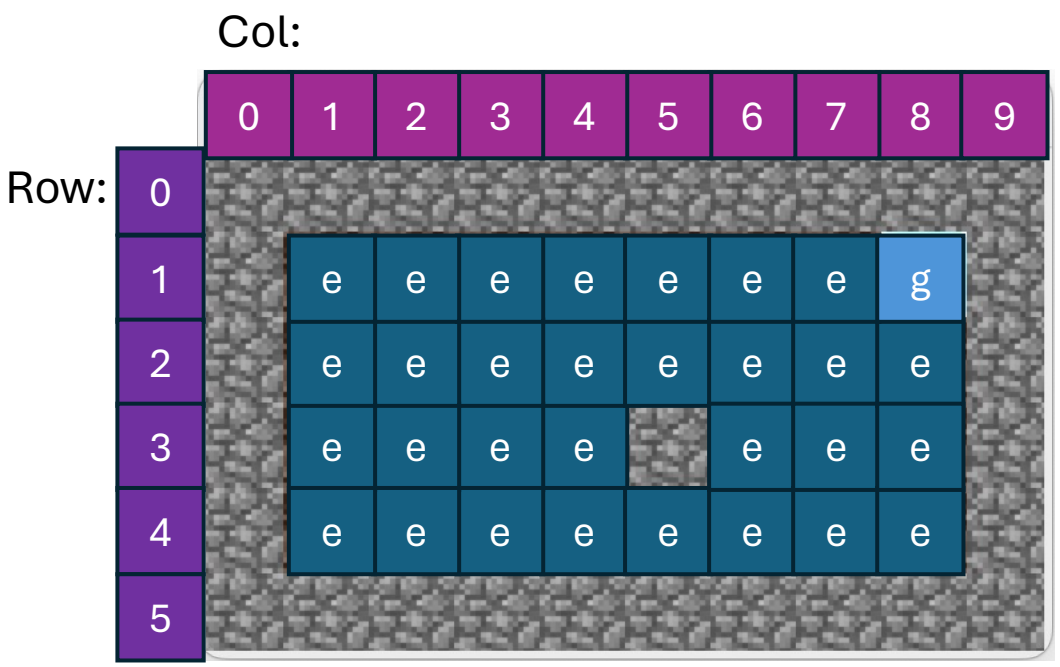
BFS/UCS with 1 cost per action



BFS/UCS with 1 cost per action



BFS/UCS with 1 cost per action



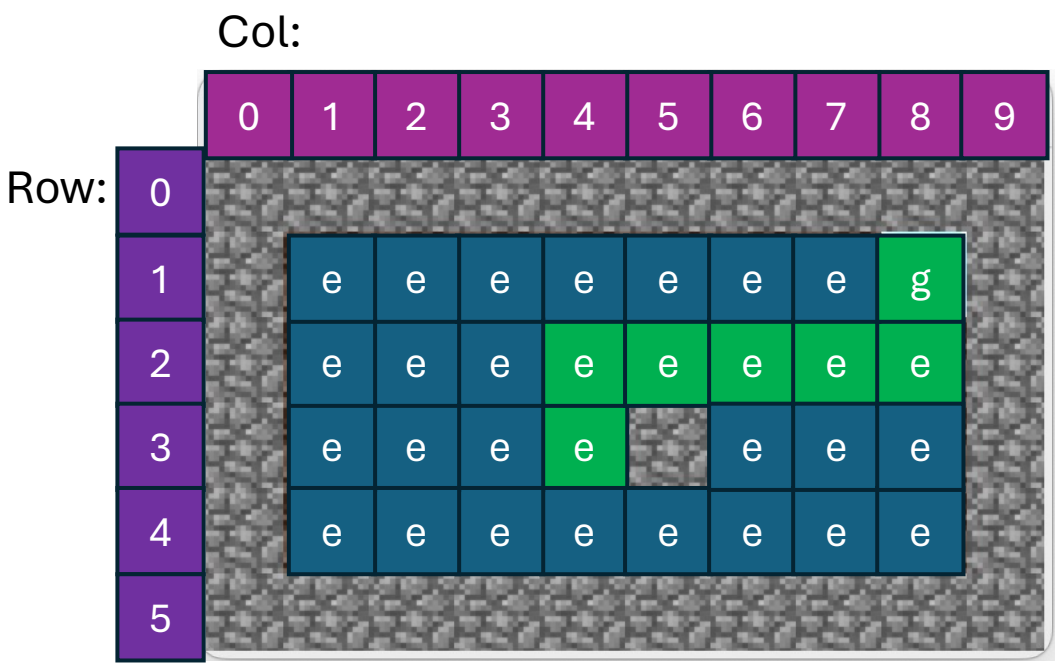
Since **UCS** performs the goal check on node **expansion** (i.e. when the node is selected from the frontier), it only detects the goal now.

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure

function UNIFORM-COST-SEARCH(problem) returns a solution node, or failure
  return BEST-FIRST-SEARCH(problem, PATH-COST)
```

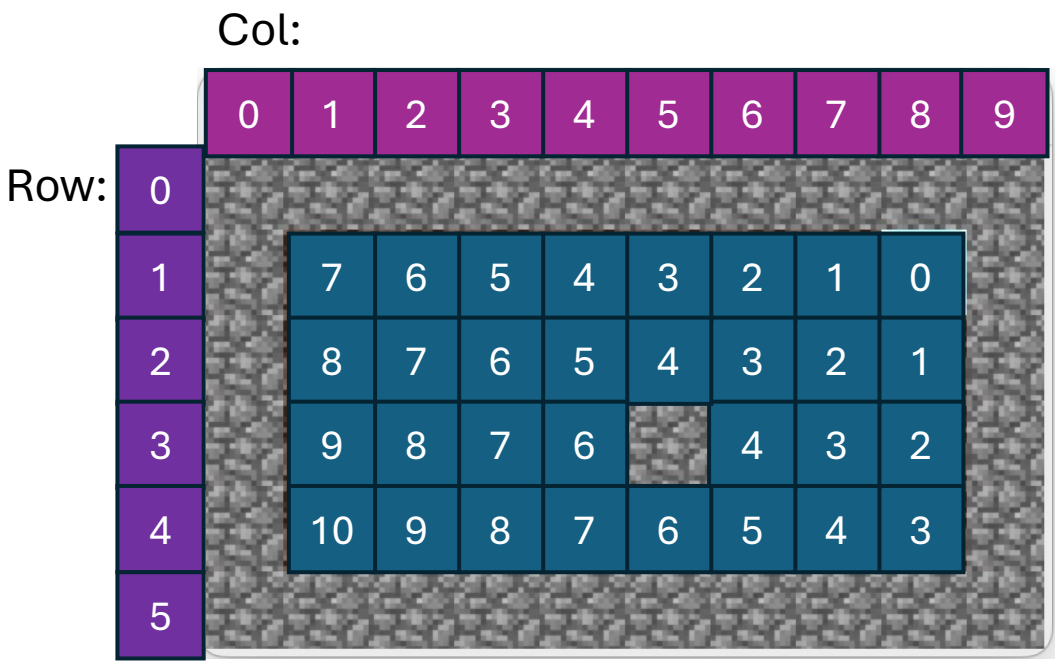
```
function BEST-FIRST-SEARCH(problem, f) returns a solution node or failure
  node ← NODE(STATE=problem.INITIAL)
  frontier ← a priority queue ordered by f, with node as an element
  reached ← a lookup table, with one entry with key problem.INITIAL and value node
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    if problem.IS-GOAL(node.STATE) then return node
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if s is not in reached or child.PATH-COST < reached[s].PATH-COST then
        reached[s] ← child
        add child to frontier
  return failure

function EXPAND(problem, node) yields nodes
  s ← node.STATE
  for each action in problem.ACTIONS(s) do
    s' ← problem.RESULT(s, action)
    cost ← node.PATH-COST + problem.ACTION-COST(s, action, s')
    yield NODE(STATE=s', PARENT=node, ACTION=action, PATH-COST=cost)
```

Figure 3.7 The best-first search algorithm, and the function for expanding a node. The data structures used here are described in Section 3.3.2. See Appendix B for **yield**.

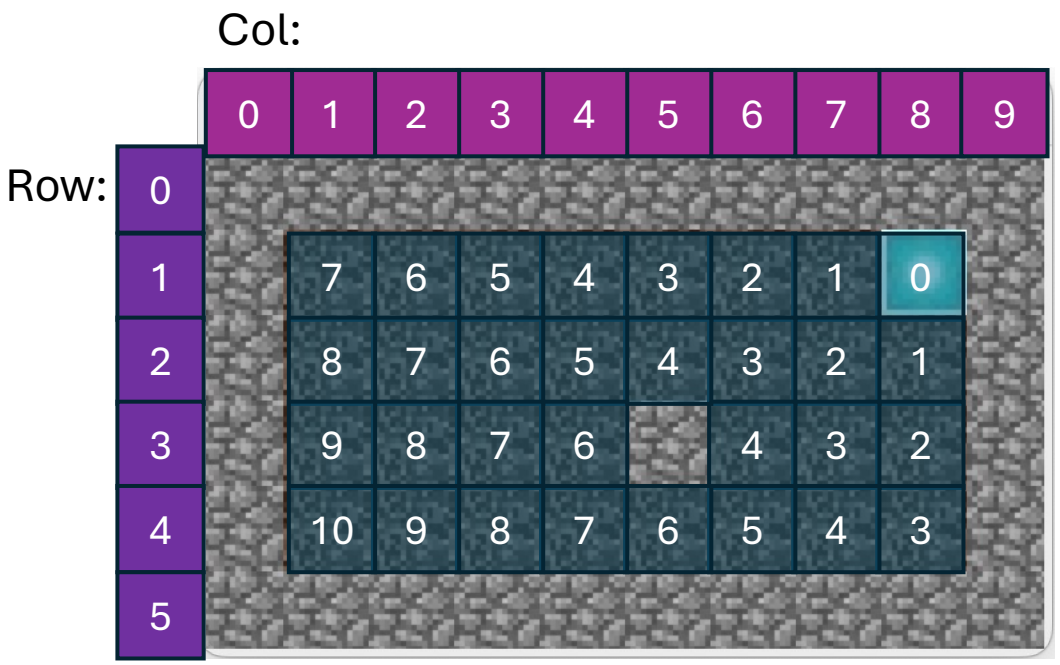
BFS/UCS with 1 cost per action



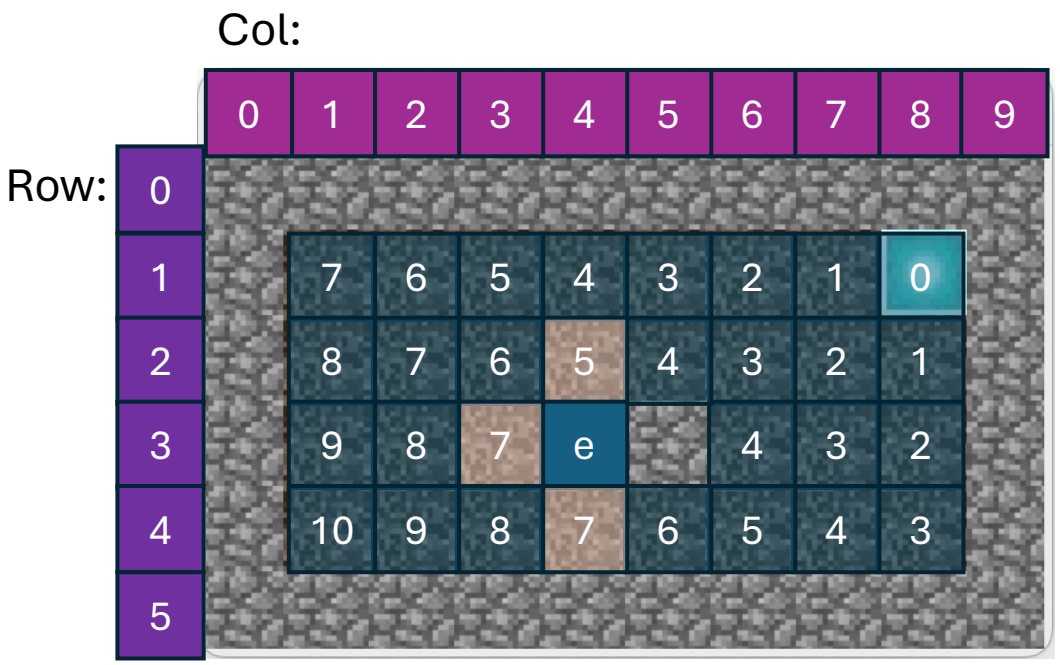
Heuristic → Manhattan distance to the goal



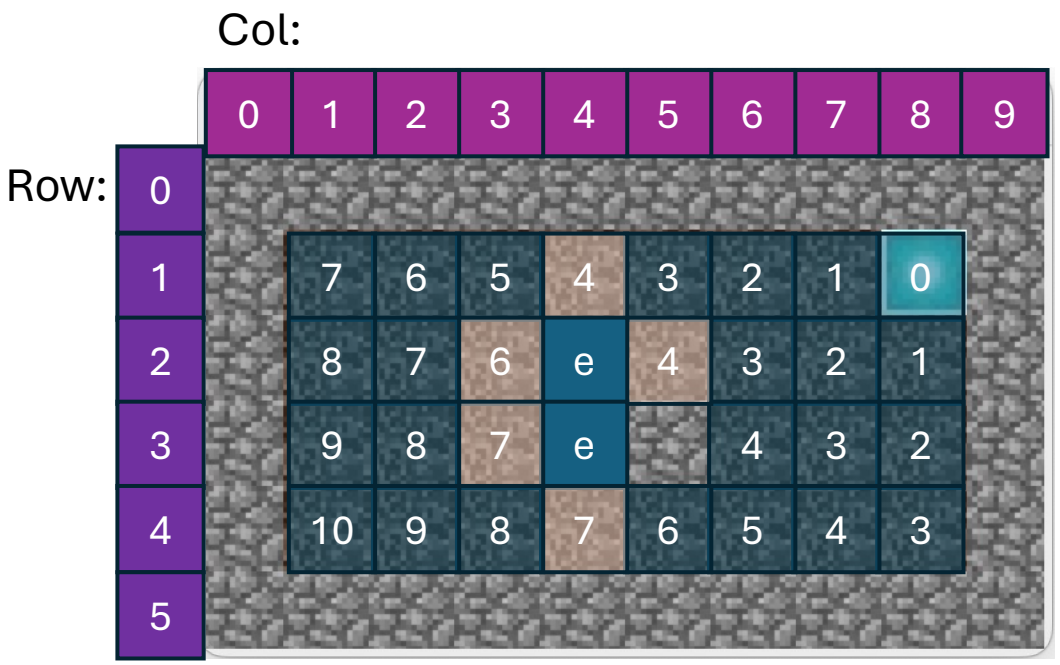
GBFS with heuristic, h(n) = Manhattan distance to the goal



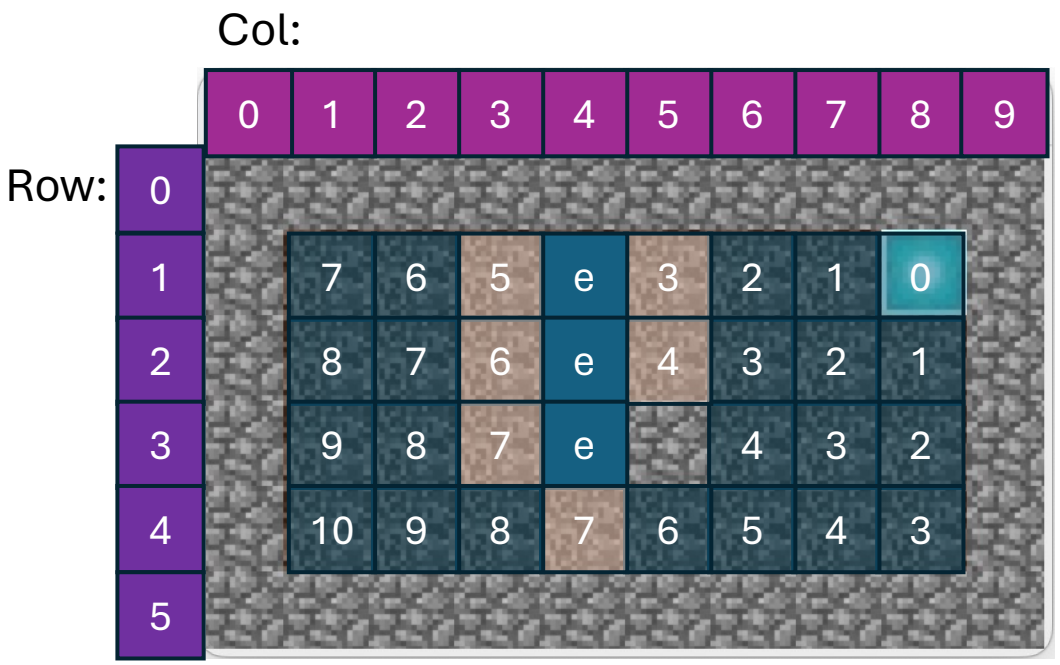
GBFS with heuristic, h(n) = Manhattan distance to the goal



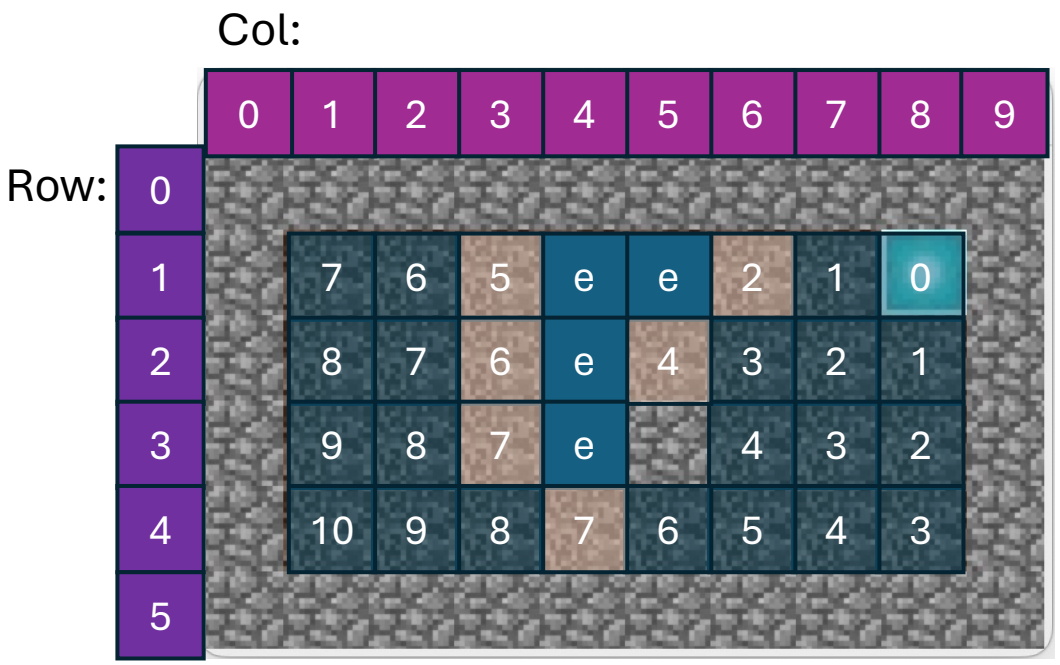
GBFS with heuristic, h(n) = Manhattan distance to the goal



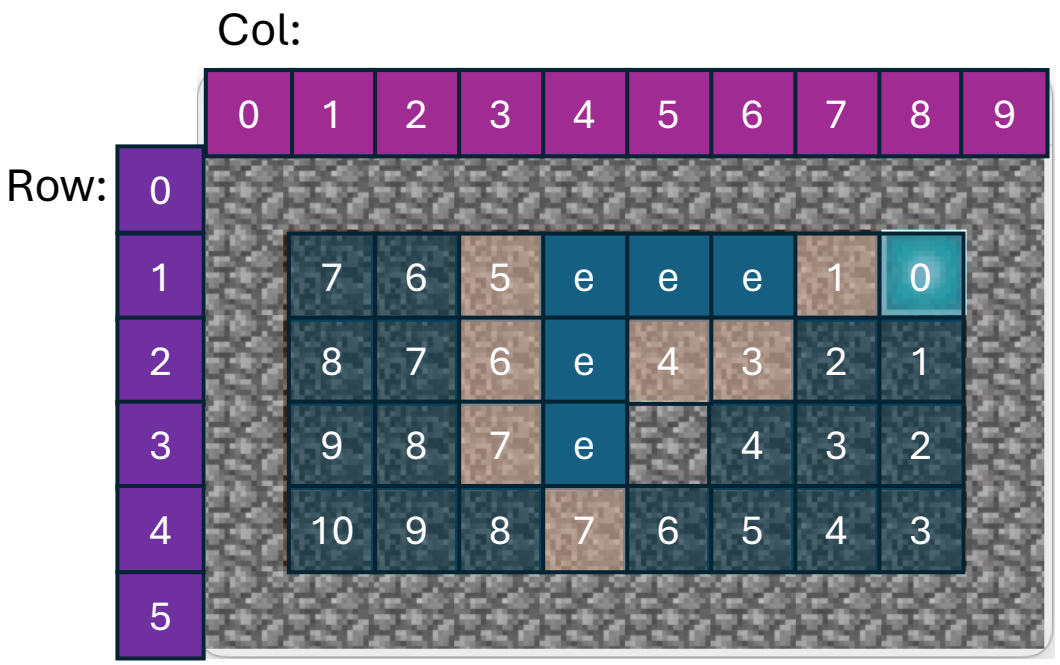
GBFS with heuristic, h(n) = Manhattan distance to the goal



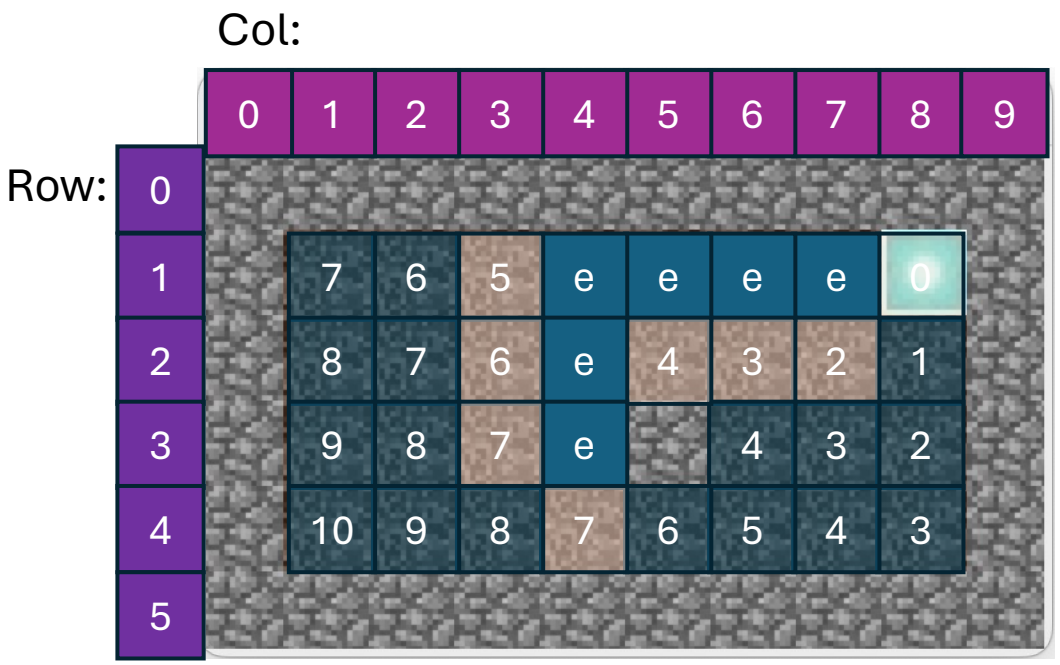
GBFS with heuristic, h(n) = Manhattan distance to the goal



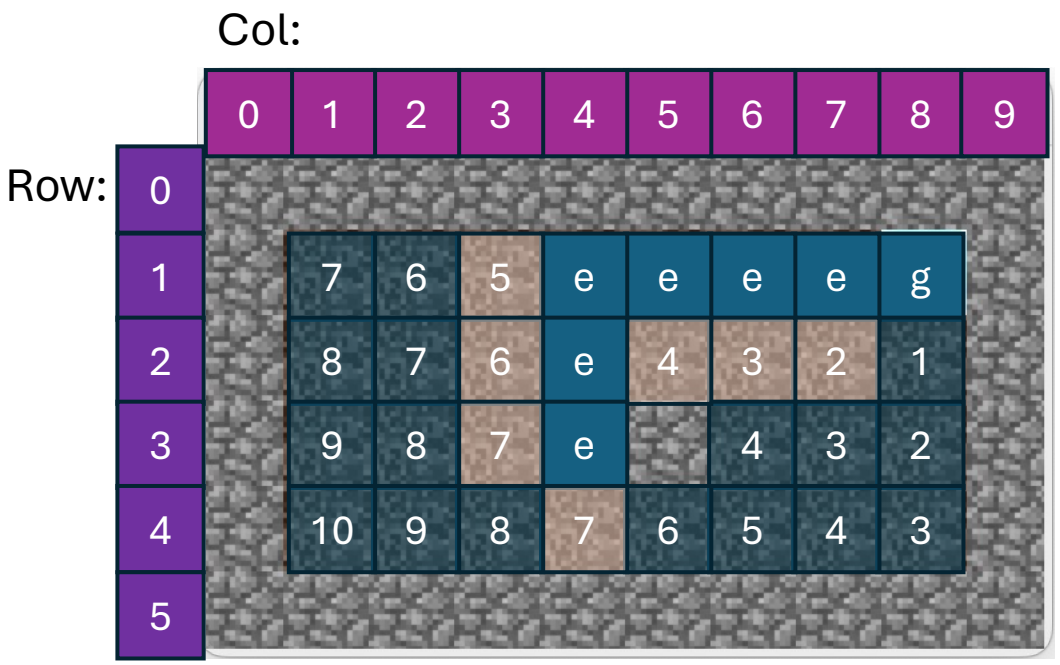
GBFS with heuristic, h(n) = Manhattan distance to the goal



GBFS with heuristic, h(n) = Manhattan distance to the goal



GBFS with heuristic, h(n) = Manhattan distance to the goal



GBFS with heuristic, h(n) = Manhattan distance to the goal