COMP 3200
Artificial Intelligence

Lecture 5

Avoiding Repeated States
Heuristic Search

MEMORIAL UNIVERSITY

Note: Terminology

- Algorithm terminology / name from

- Artificial Intelligence: A Modern Approach
  - Stuart Russel and Peter Norvig
  - Not required for course, but amazing book

- Names not important, concepts are

Avoiding Repeated States

- One of the most important search ideas
- Especially important with reversible actions
- Most infinite loops / wasted time can be avoided by not returning to identical states
- We can remember nodes expanded
  - Don’t re-expand them
- Can lead to exponential savings in the number of nodes generated

‘Closed List’

- Store states we have already expanded
- Closed List stores expanded states
    - Often implemented as a hash table / dictionary for efficient lookup
- Check if a node is closed before expanding
- Open List = fringe of unexpanded nodes
    - Often implemented as a priority queue

General Tree-Search (recall)

```
Function Graph-Search(problem, strategy)

    open = {Node(problem.initial_state)}
    while (true)
        if (open.empty) return fail
        node = strategy.select_node(open)
        if (node.state is goal) return solution


        open.add(Expand(node, problem))
```

General Graph-Search

```
Function Graph-Search(problem, strategy)
    closed = {} // add closed list
    open = {Node(problem.initial_state)}
    while (true)
        if (open.empty) return fail
        node = strategy.select_node(open)
        if (node.state is goal) return solution
        if (node.state in closed) continue // check if closed
        closed.add(node.state) // mark node closed
        open.add(Expand(node, problem))
```

General Graph-Search

```
Function Graph-Search(problem, strategy)
    closed = {}
    open = {Node(problem.initial_state)}
    while (true)
        if (open.empty) return fail
        node = strategy.select_node(open)
        if (node.state is goal) return solution
        if (node.state in closed) continue
        closed.add(node.state)
        open.add(Expand(node, problem))
```

Assignment 1: BFS

```
Function Graph-Search(problem, BFS)
    closed = {}
    open = Queue{Node(problem.initial_state)}
    while (true)
        if (open.empty) return fail
        node = open.pop() // get from front of queue
        if (node.state is goal) return solution
        if (node.state in closed) continue
        closed.add(node.state)
        open.add(Expand(node, problem))
```

Assignment 1: DFS

```
Function Graph-Search(problem, DFS)
    closed = {}
    open = Stack{Node(problem.initial_state)}
    while (true)
        if (open.empty) return fail
        node = open.pop() // get from top of stack
        if (node.state is goal) return solution
        if (node.state in closed) continue
        closed.add(node.state)
        open.add(Expand(node, problem))
```

Tree-Search vs Graph-Search

- Tree Search remembers nothing
  - Will re-expand states multiple times
- Graph-Search remembers states visited
  - Will never re-expand state a second time
- Tree/Graph Search = Just a name

Graph-Search Complexity

- Time Complexity O(states)
  - Each state expanded at most once
  - Worst case, expand every state in environment
  - In most cases, far lower than $O(b^d)$
  - Same upper bound no matter which search strategy
- Space Complexity O(states)
  - Must store entire search space in memory
  - Space = nodes generated = # states
  - Better, but often times still too much
  - Previously linear space methods may now use more memory

Graph-Search Optimality

- Graph-Search may no longer be Optimal
  - Tricky issue, since it depends on underlying strategy
  - We a repeated state is detected, the algorithm has found another path to the same state
  - Graph-Search (as shown) keeps the original path, and discards new paths found, which may be optimal
  - Only optimal if we can guarantee that the first solution found is the optimal one (BFS, UCS, with same costs)

Bonus Enhancement

- Open list nodes contain their g values
- When generating a node, check its g value
- If a node already exists on the open list with a ≤ g value, it has a shorter path
- The path we generated has > g, so is worse
- Do not add the generated node to open list
- Note: Only for action costs > 0

Enhanced Graph-Search

```
Function Graph-Search(problem, strategy)
    closed = {}
    open = {Node(problem.initial_state)}
    while (true)
        if (open.empty) return fail
        node = strategy.select_node(open)
        if (node.state is goal) return solution
        if (node.state in closed) continue
        closed.add(node.state)
        for c in Expand(node, problem)
            if (node n in open with c.state and n.g ≤ c.g) continue
            open.add(c)
```

Enhanced Graph-Search

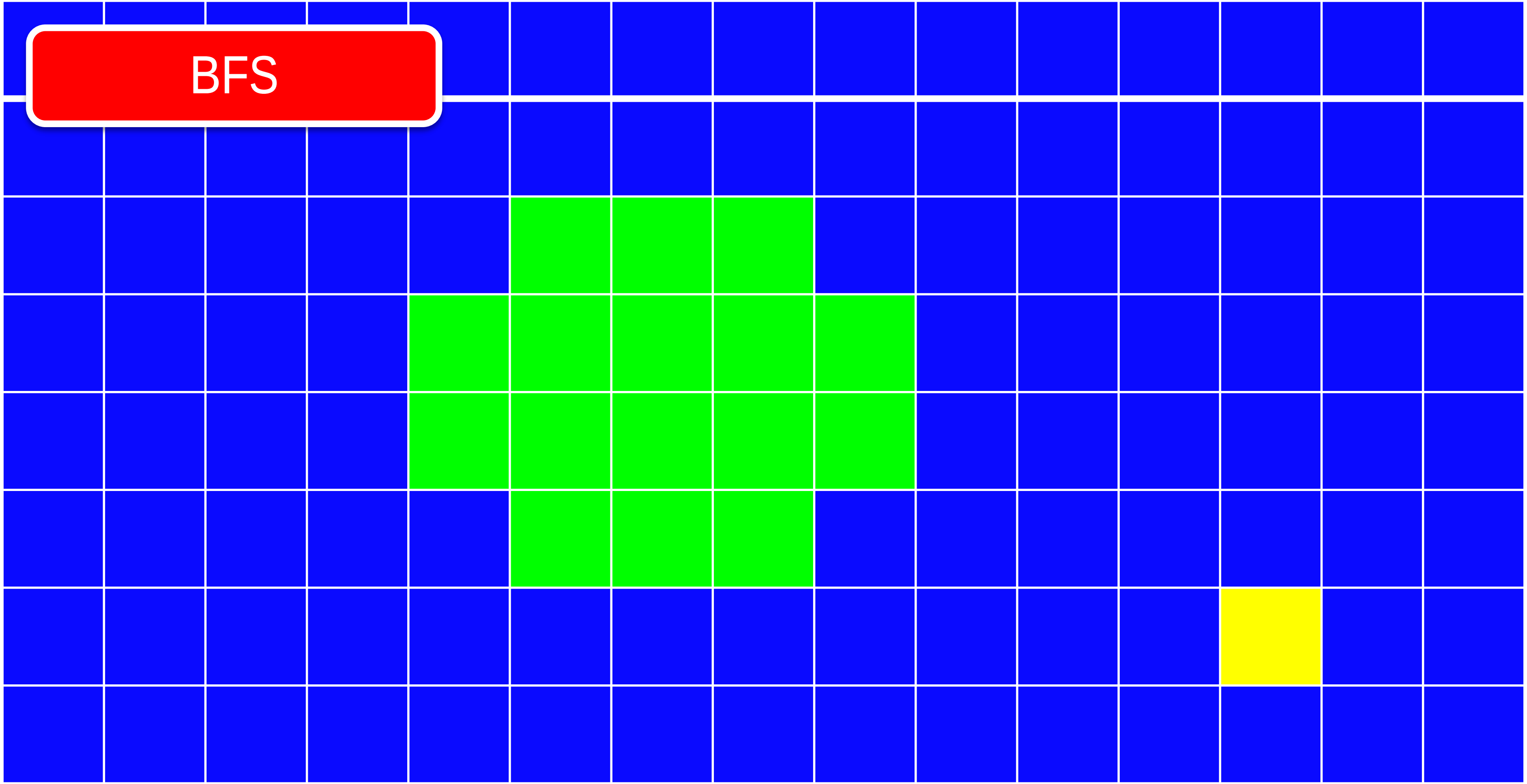
```
Function Graph-Search(problem, strategy)
    closed = {}
    open = {Node(problem.initial_state)}
    while (true)
        if (open.empty) return fail
        node = strategy.select_node(open)
        if (node.state is goal) return solution
        if (node.state in closed) continue
        closed.add(node.state)
        for c in Expand(node, problem)
            if (node n in open with c.state and n.g ≤ c.g) continue
            open.add(c)
```

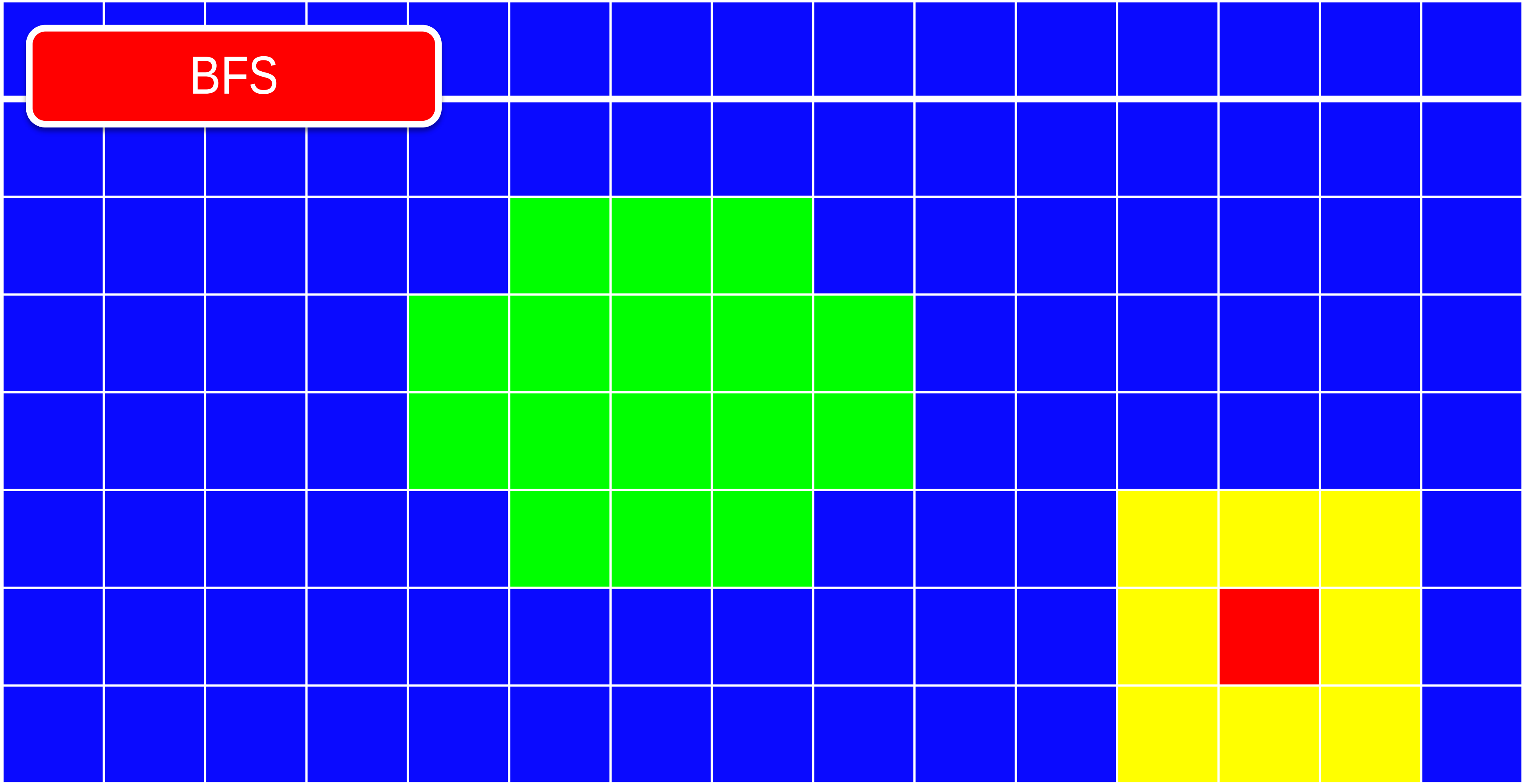
Heuristic Search

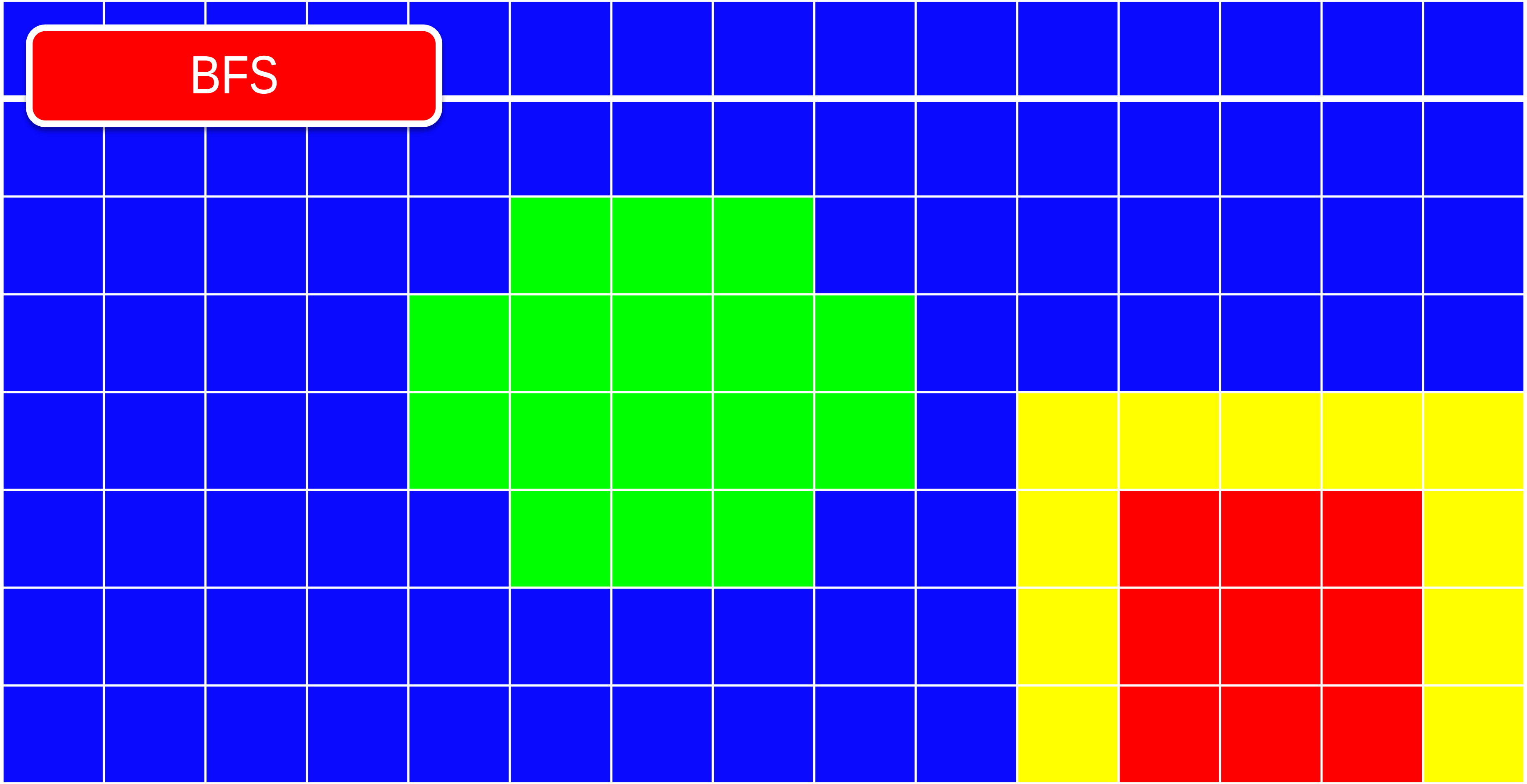
Informed (Heuristic) Search

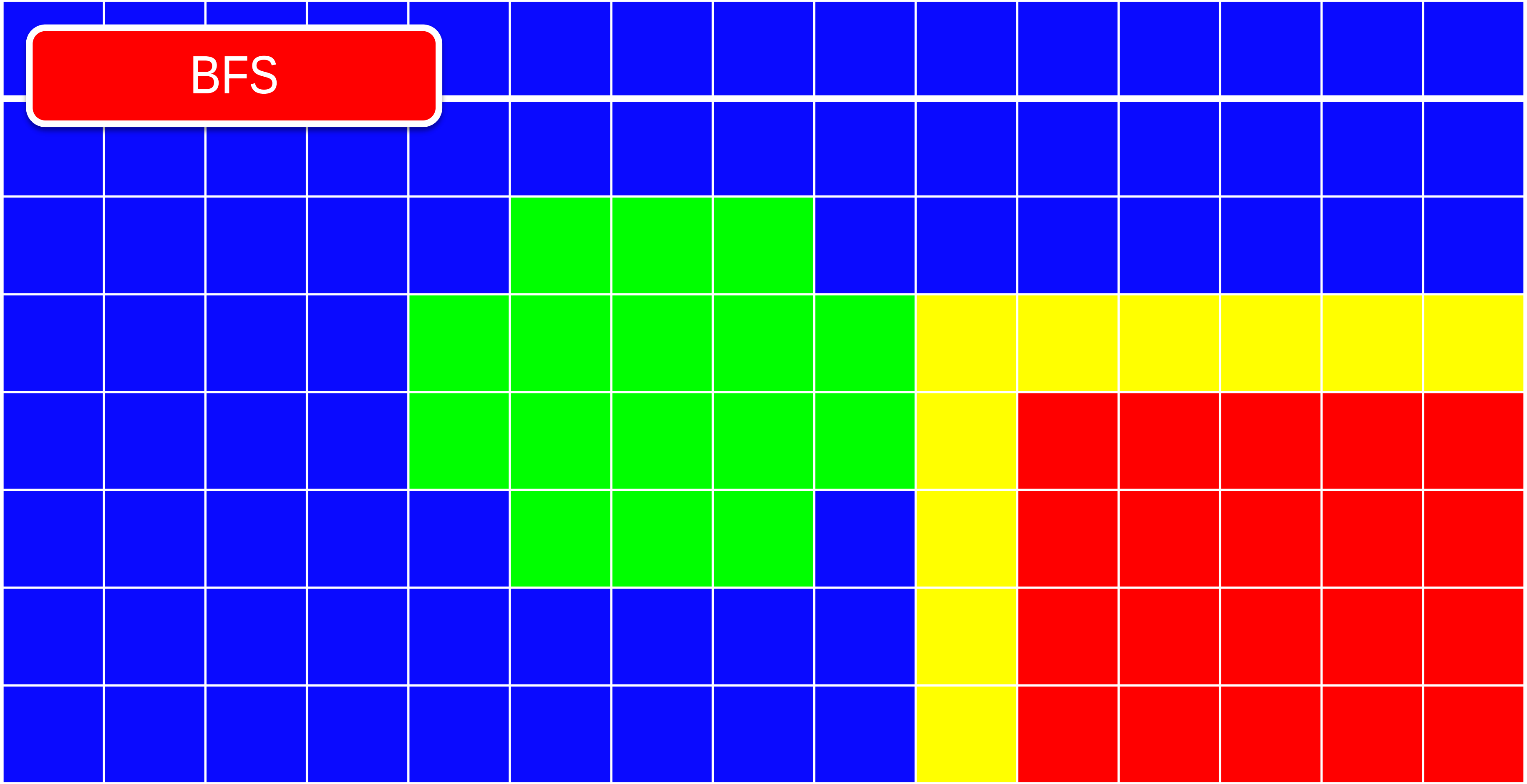
- An informed search strategy uses problem-specific knowledge beyond just the problem description itself
- Uses guesses (heuristics) to guide search toward the direction of the goal
- Reduce total nodes searched
- Speeds up search times to goal

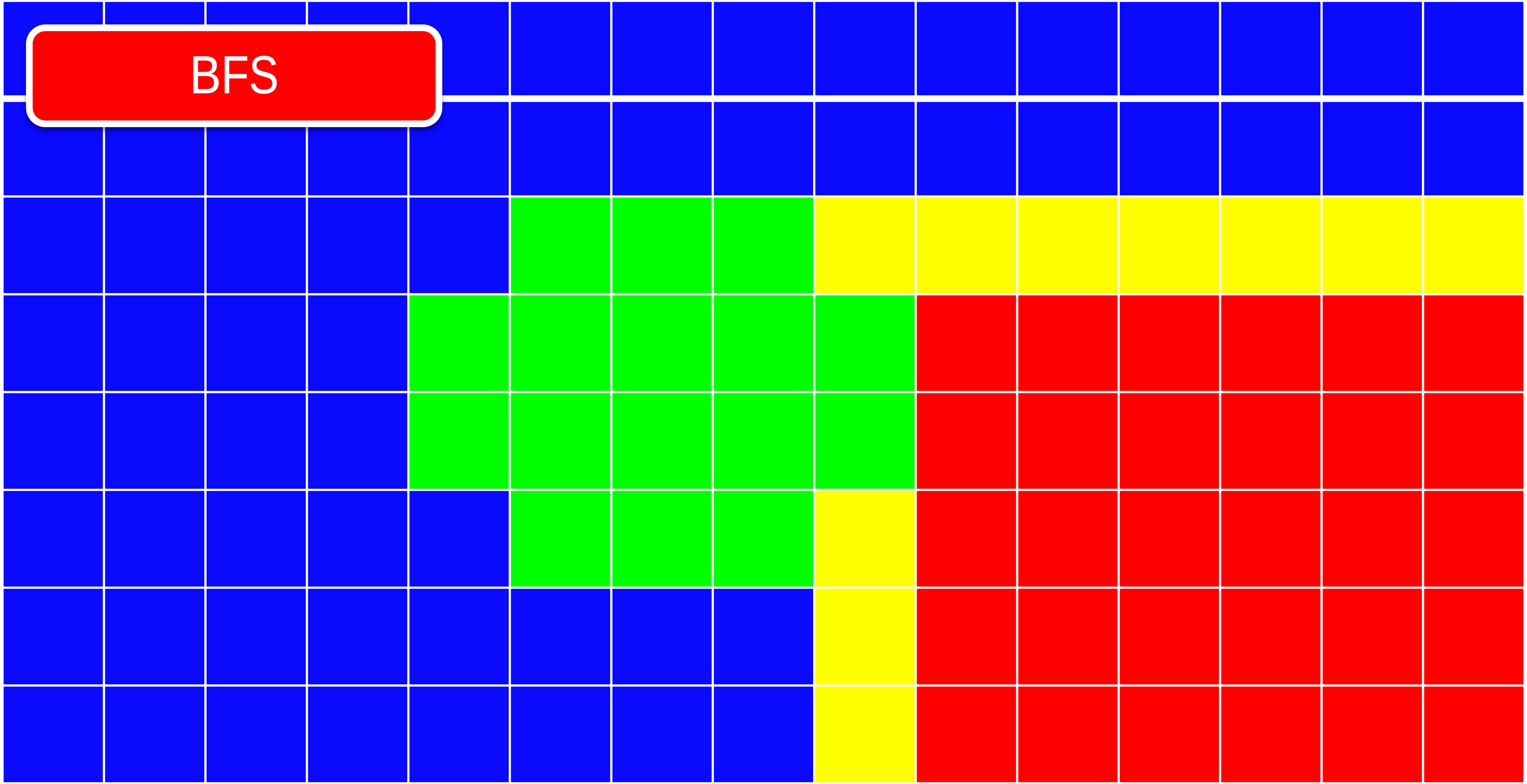
BFS

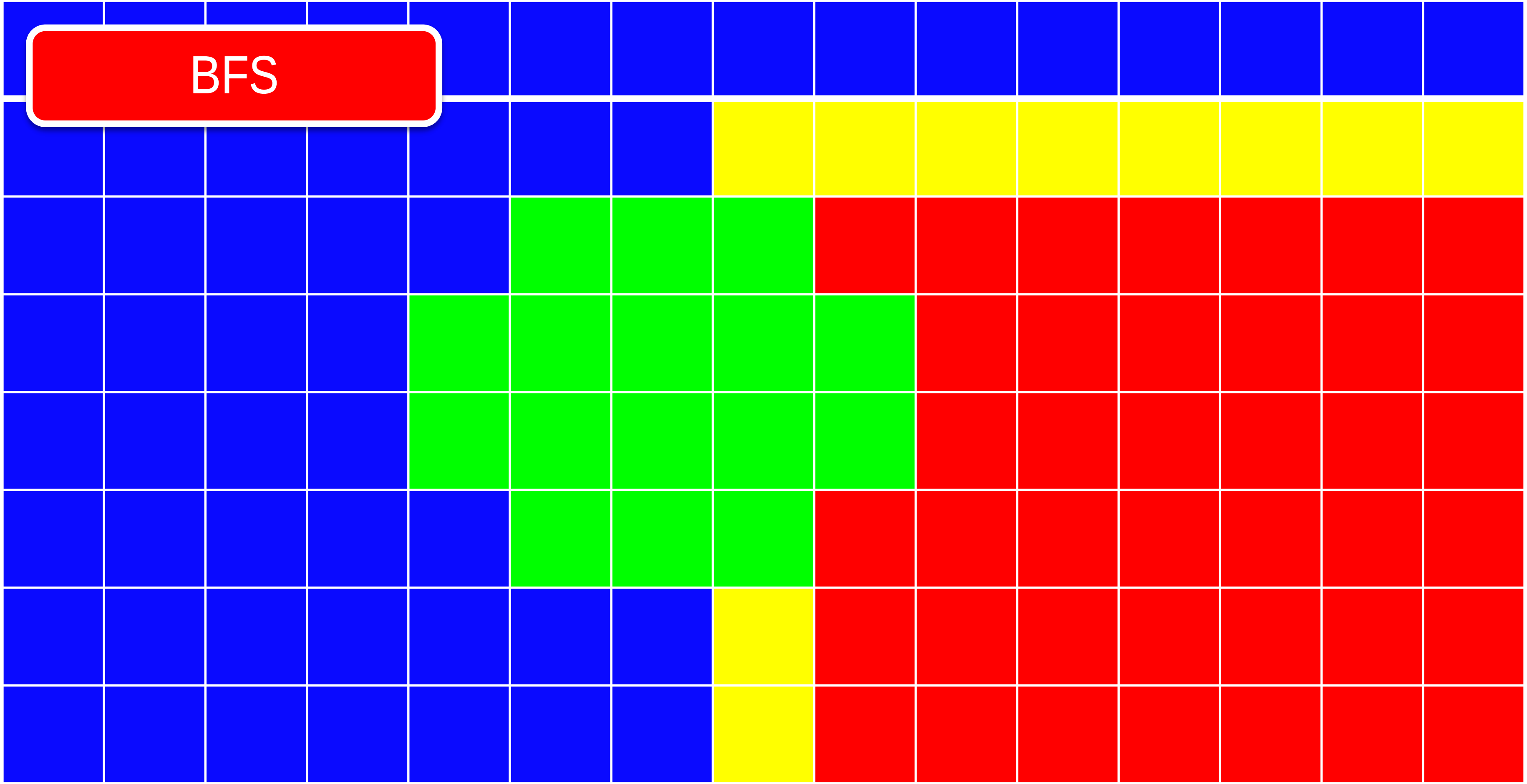



BFS


BFS


BFS


BFS


BFS

Best-First Search (BeFS)

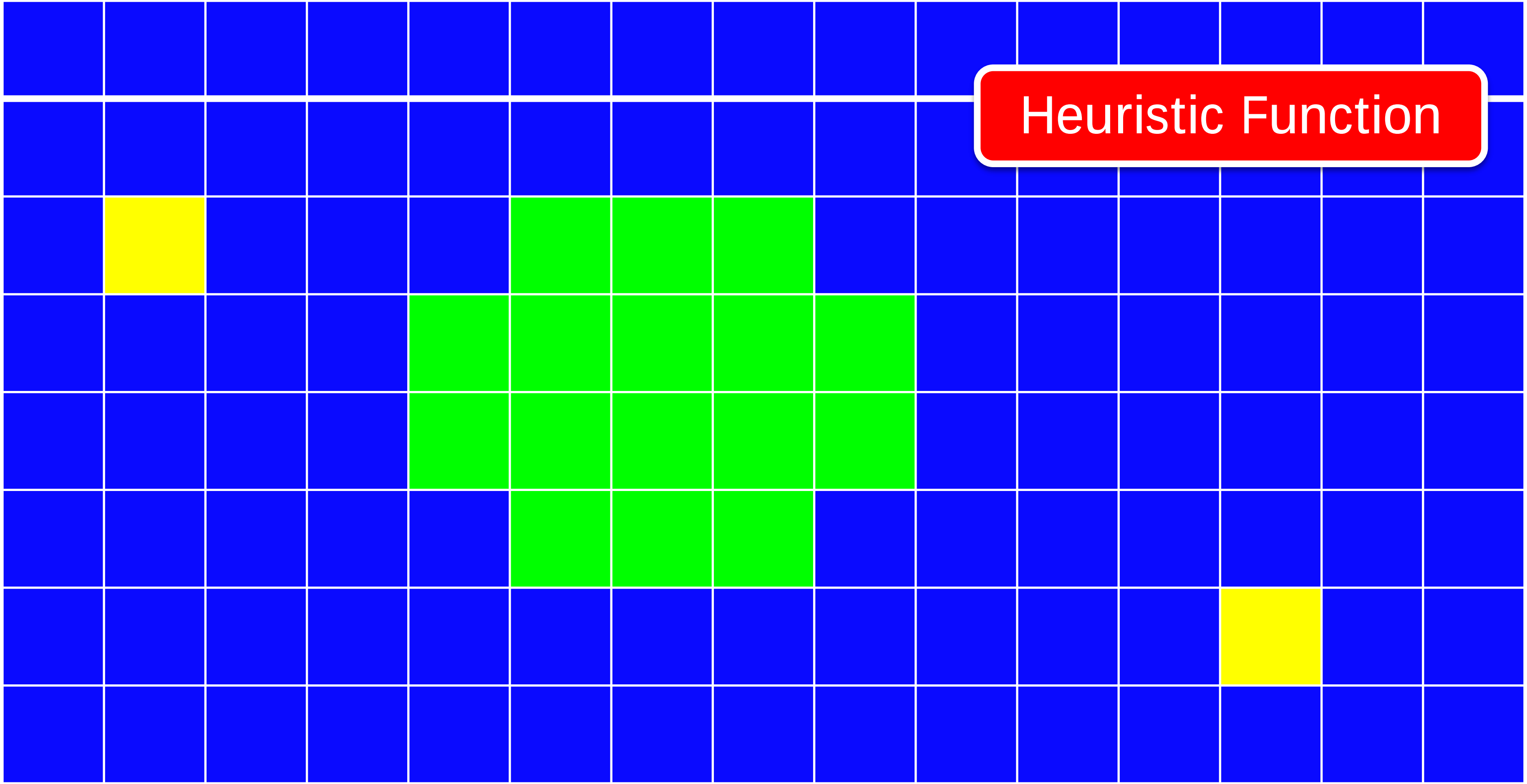
- Instance of general tree search
- Select a node for expansion based on an evaluation function = f(n)
- Select the node with minimum value of f(n), measures distance to goal
- "Best First" is slightly misleading
  - Knowing the true best node = go straight to goal
  - More like "Best Guess First"

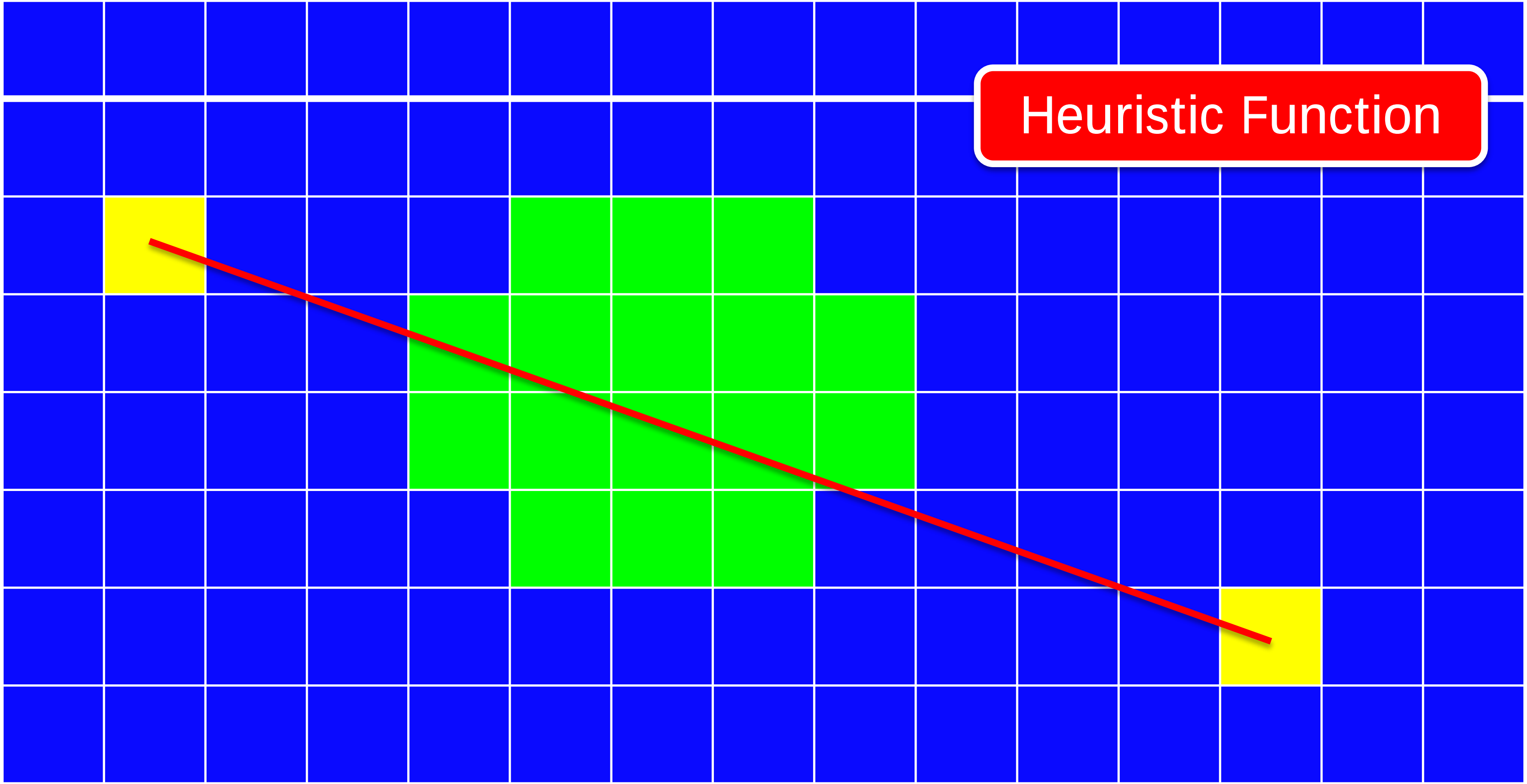
Best-First Search (BeFS)

- BeFS can be implemented with general Tree-Search by using a priority queue for the open list, sorted on f(n)
- Whole family of BeFS algorithms
  - Have different evaluation functions
  - Most have a heuristic function h(n)

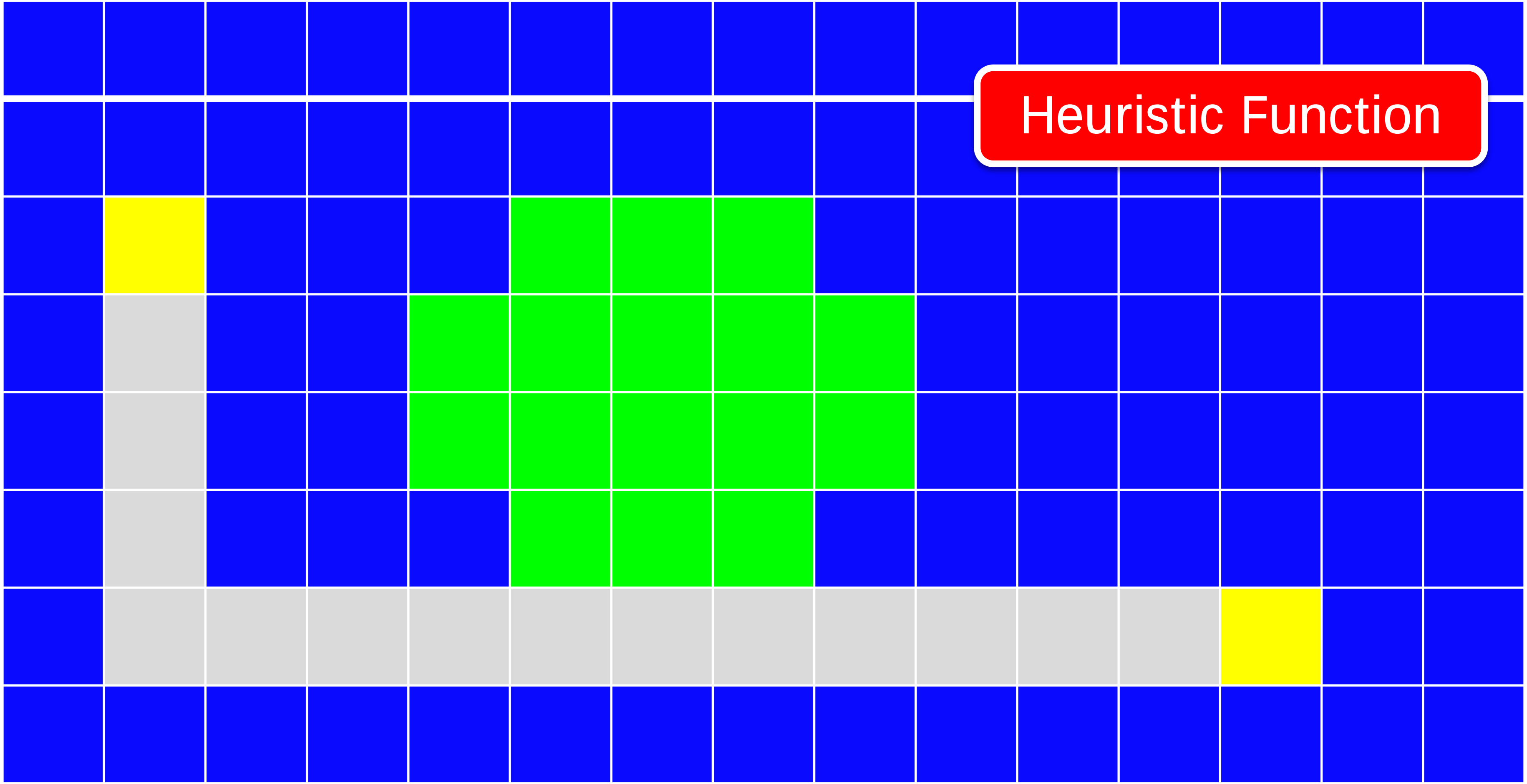
Heuristic Function h(n)

- Estimated cost of the optimal path from node n to a goal node
- Heuristic functions are the most common way that additional knowledge of a problem is given to the search algorithm
- If n is a goal node, $h(n) = 0$
- Perfect heuristic = $h^*(n)$

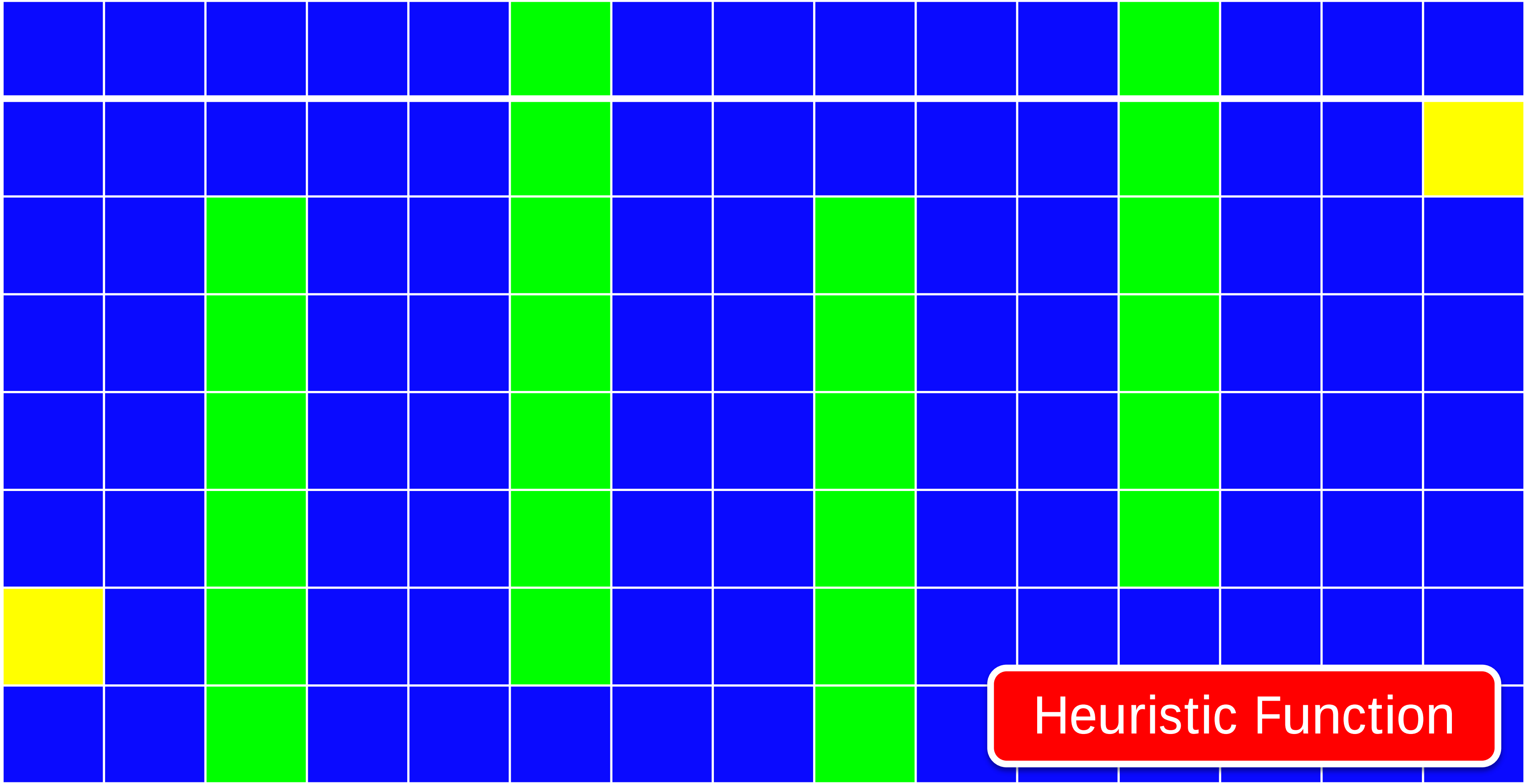

Heuristic Function

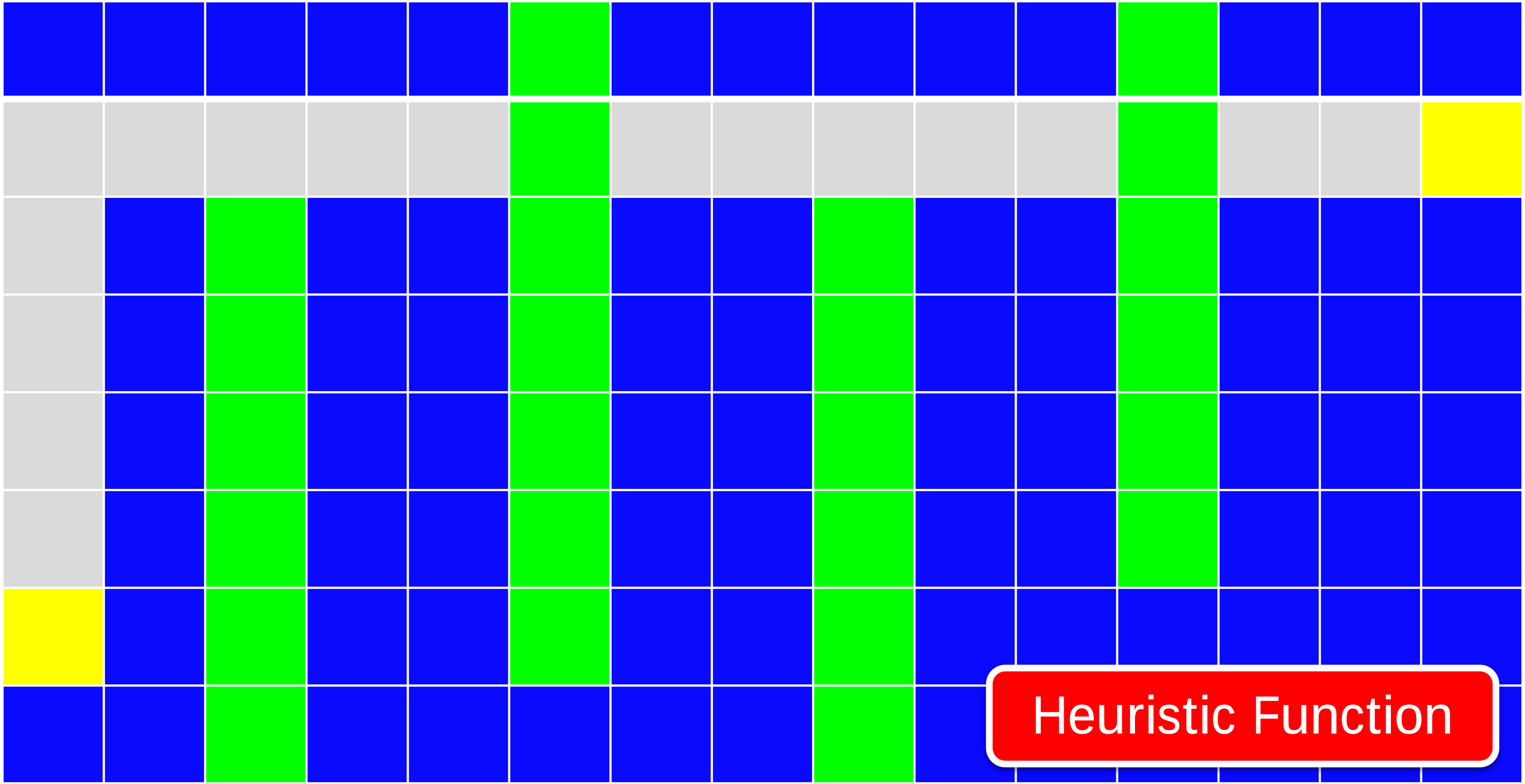

Heuristic Function

Heuristic Function



Heuristic Function

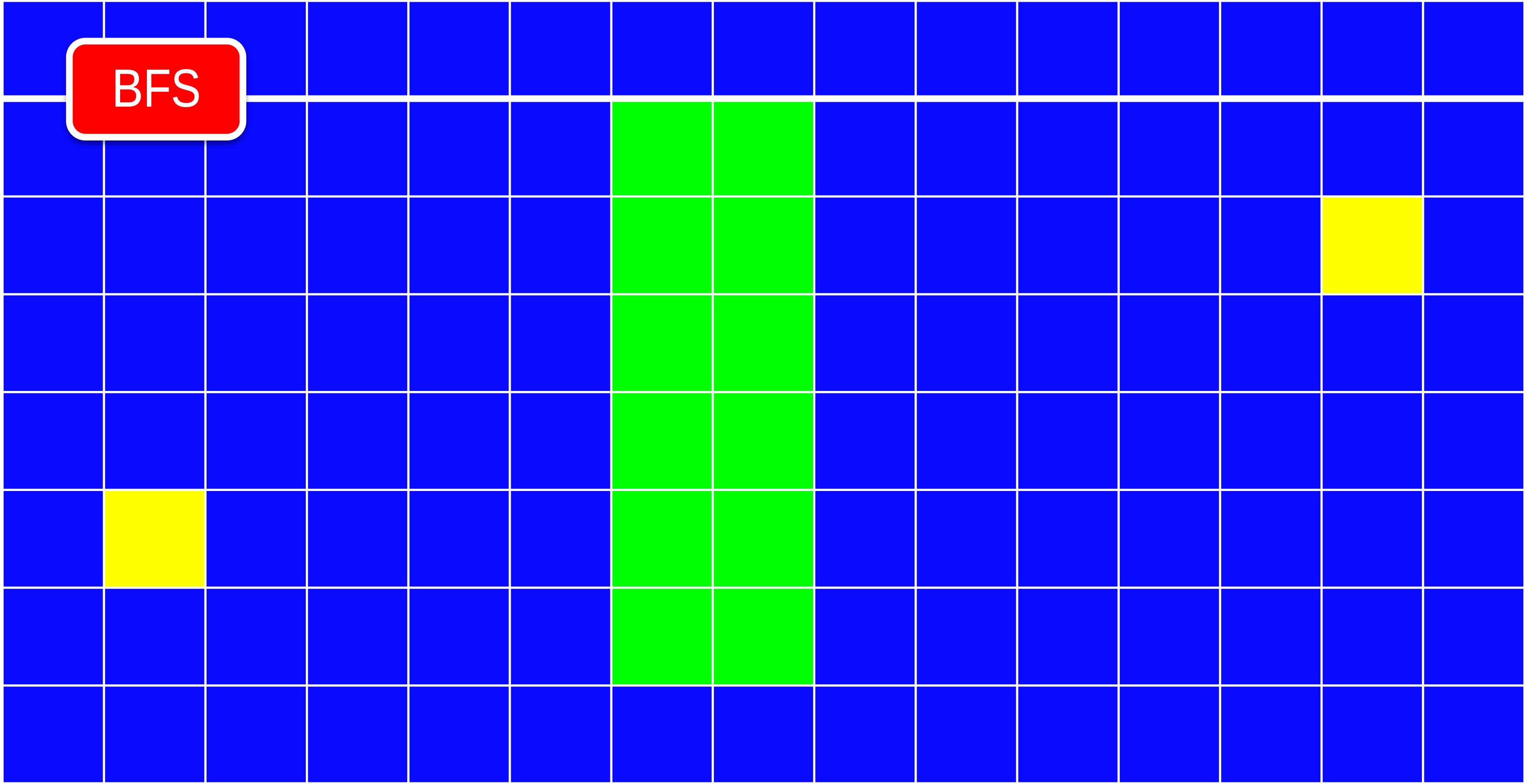


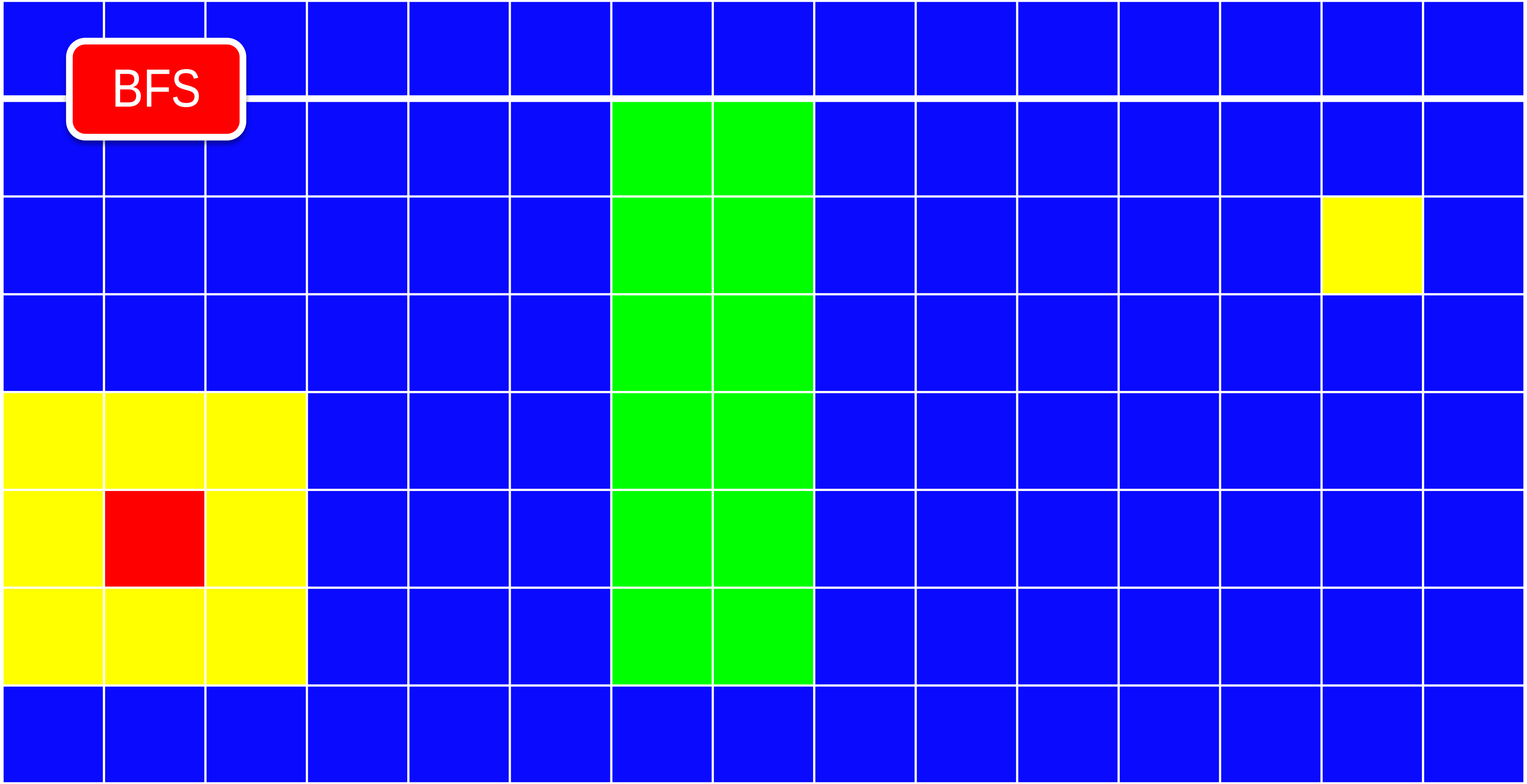


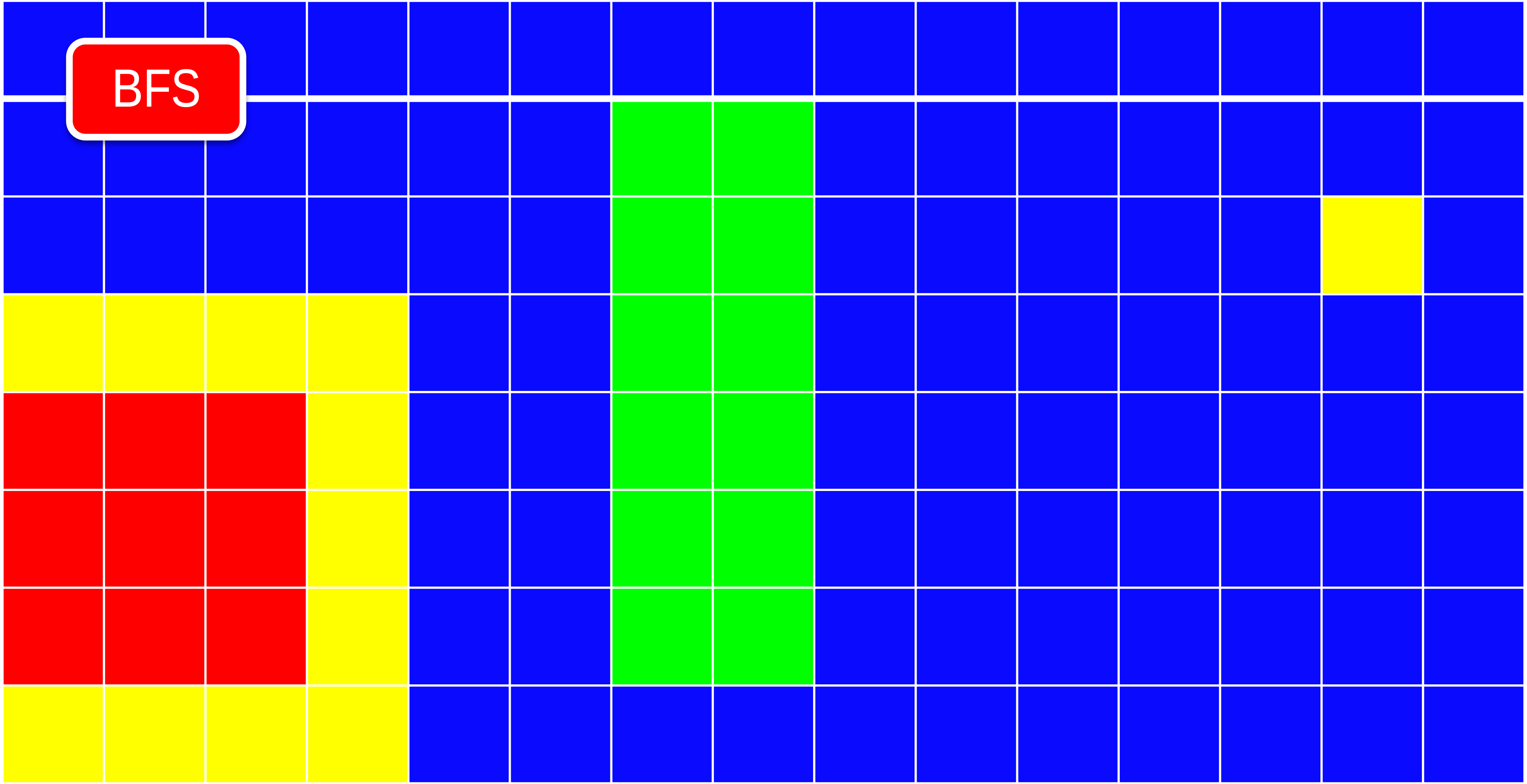
Heuristic Function

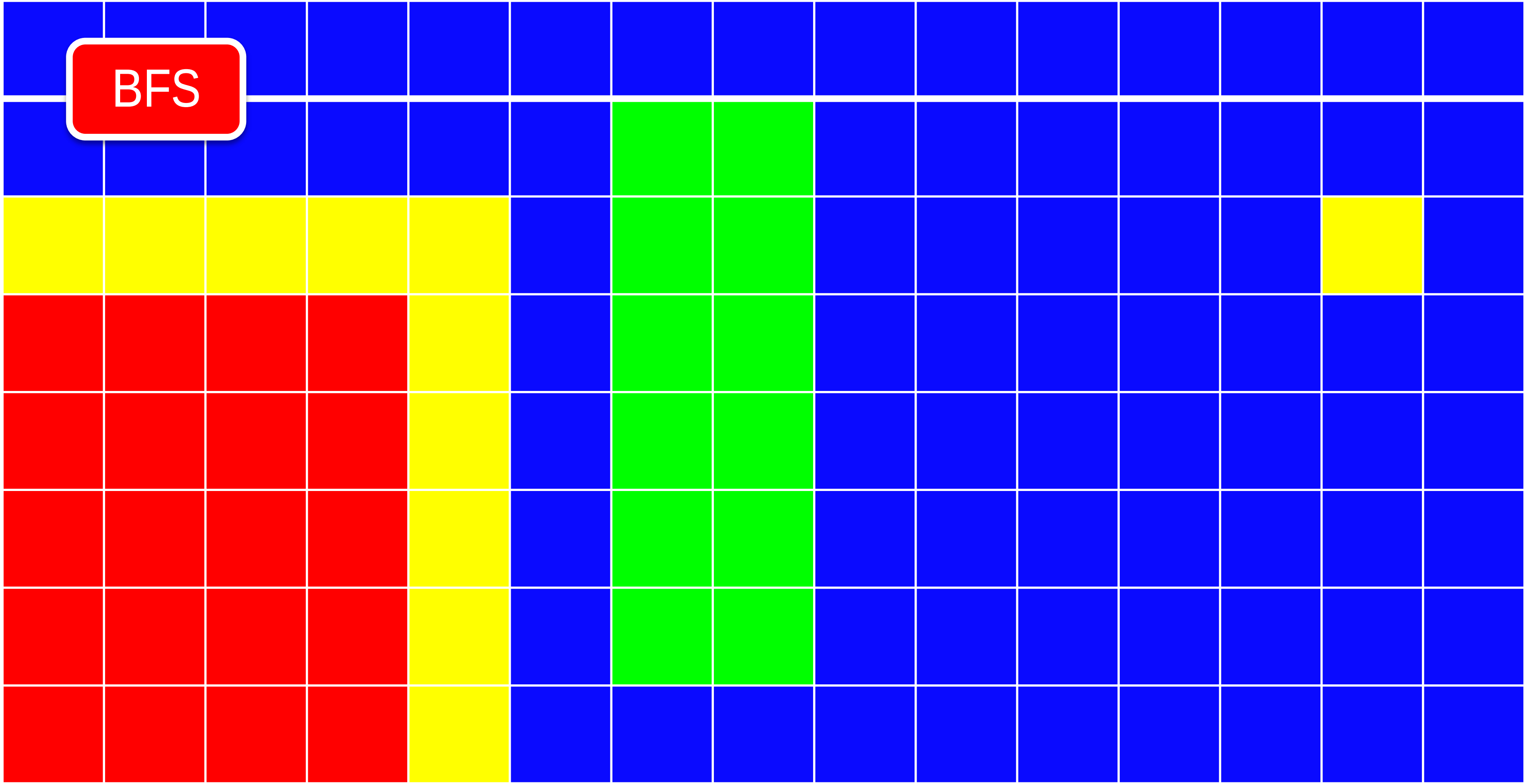
Greedy Best-First Search

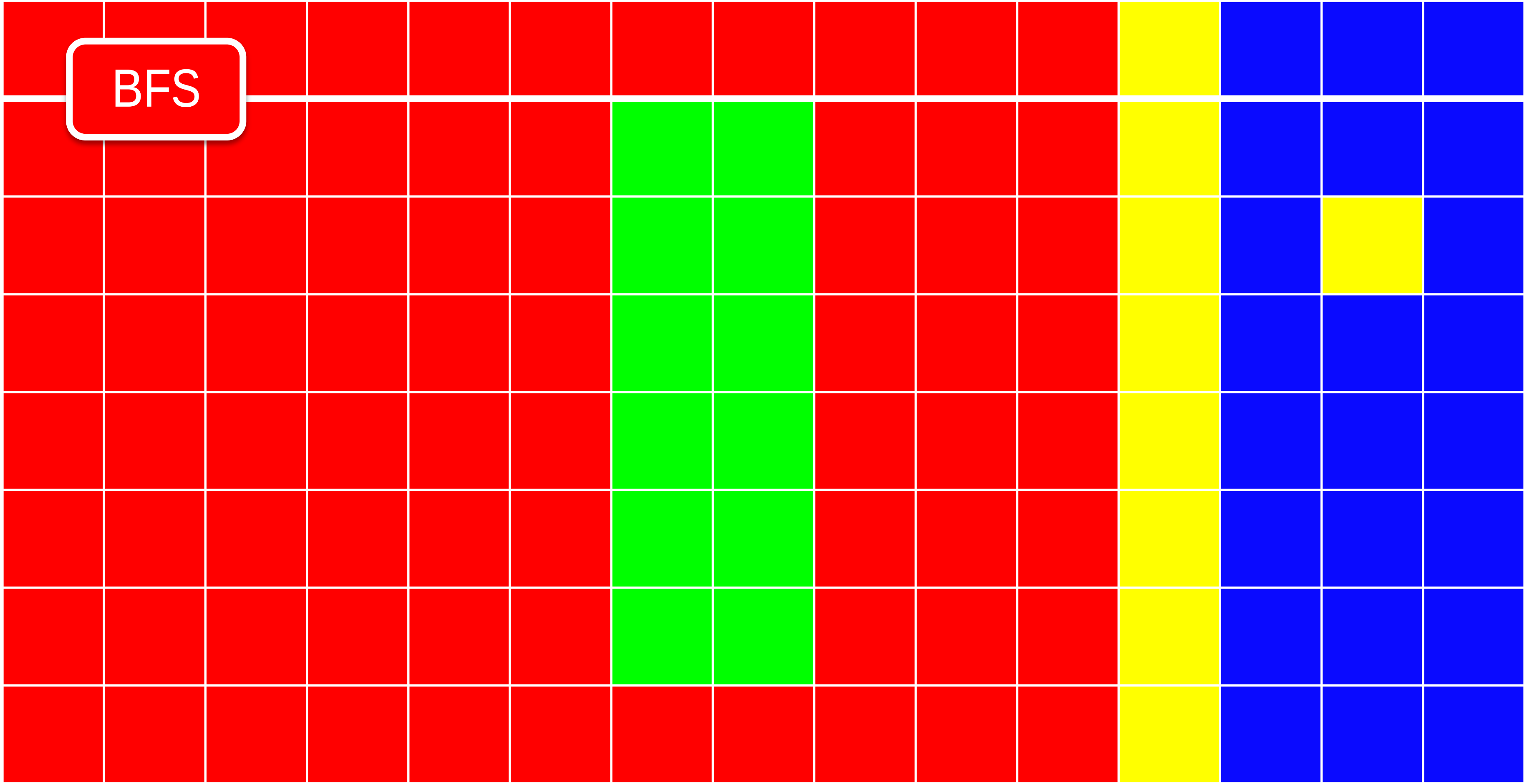
- Expands the node on the open list that it thinks is closest to the goal node
- This will hopefully lead us to the goal
- Thus, for GBeFS: $f(n) = h(n)$
- Resembles DFS
  - Tries a single path all the way to a goal
  - Backs up when it hits a ‘dead-end’

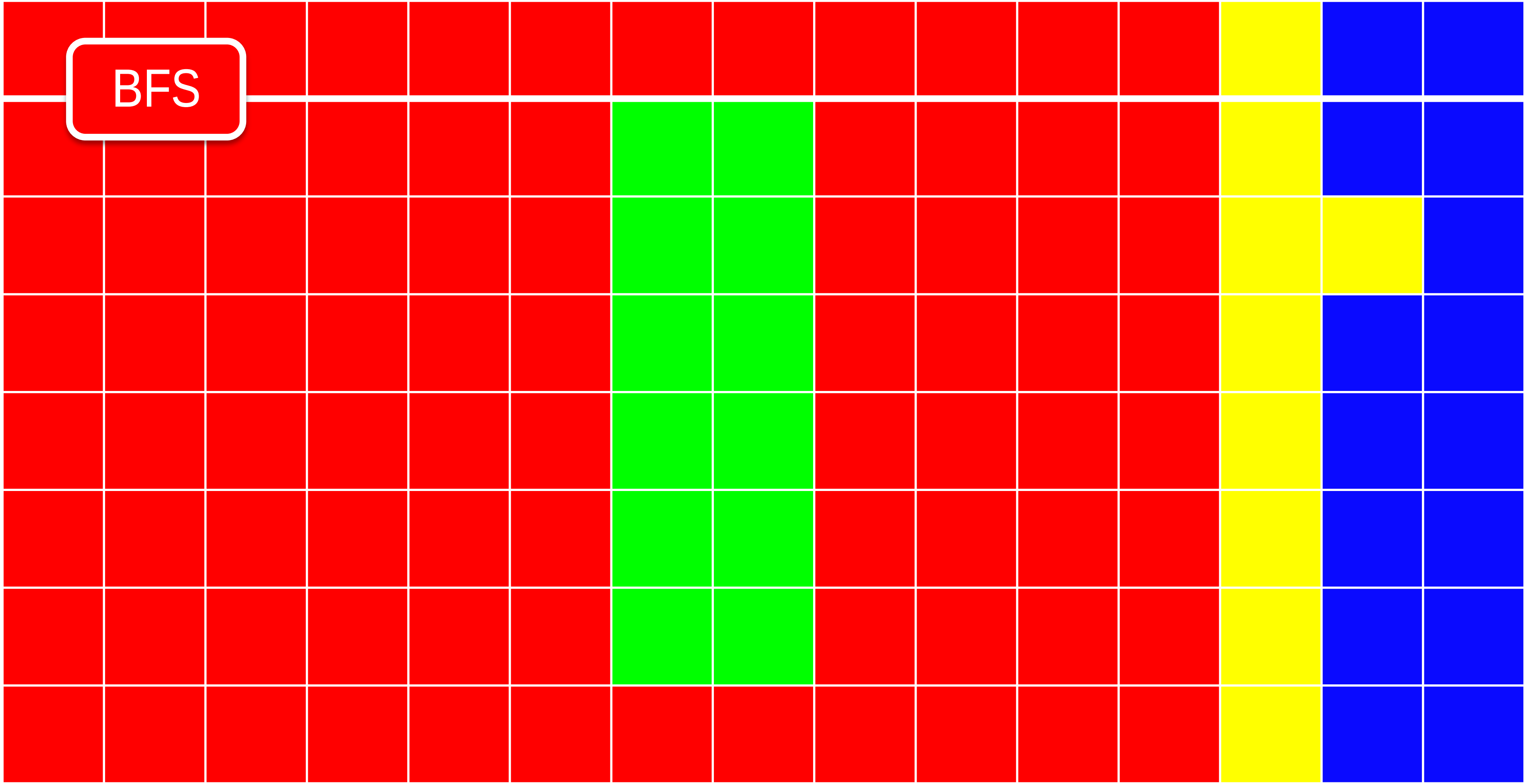

BFS

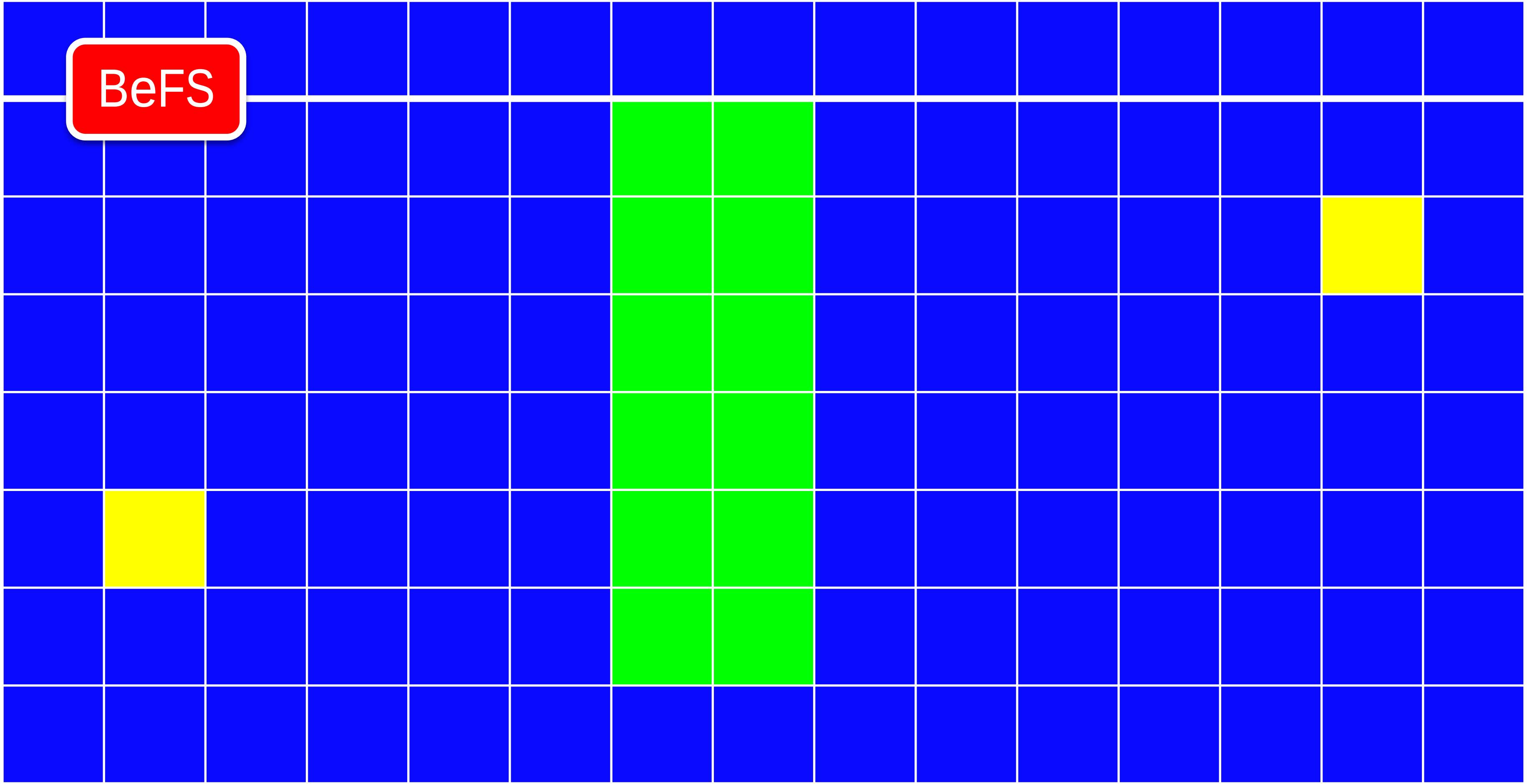

BFS

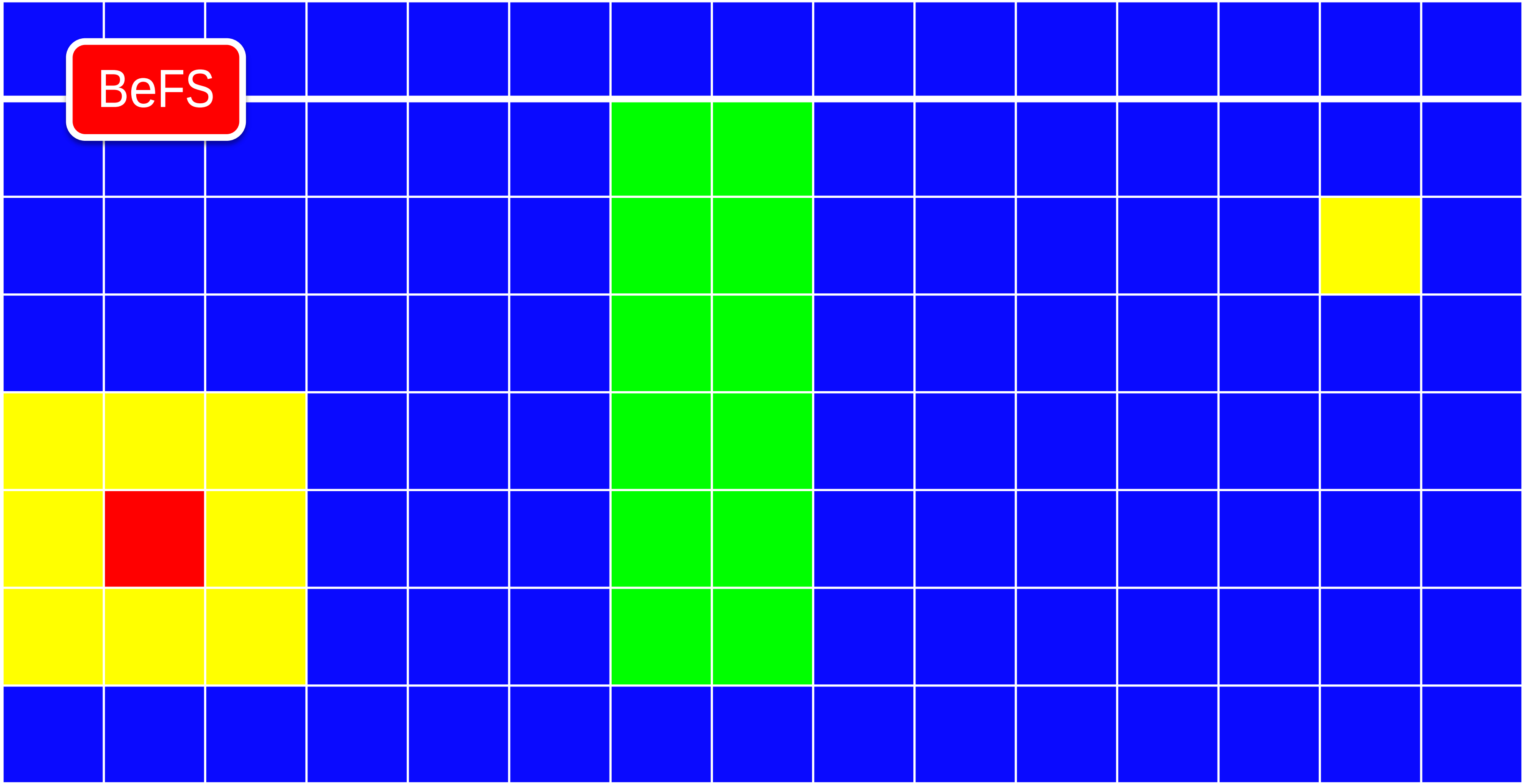

BFS

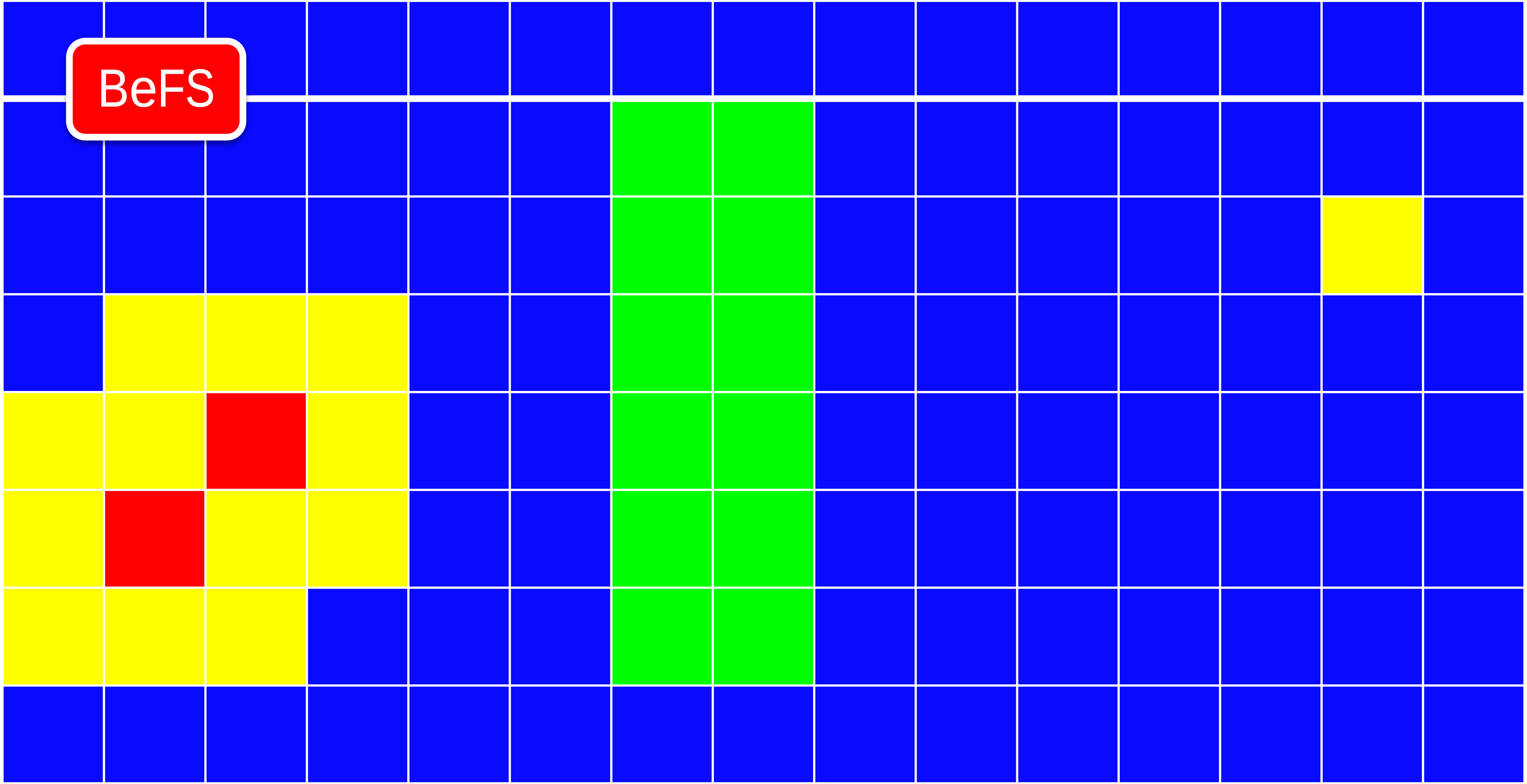

BFS

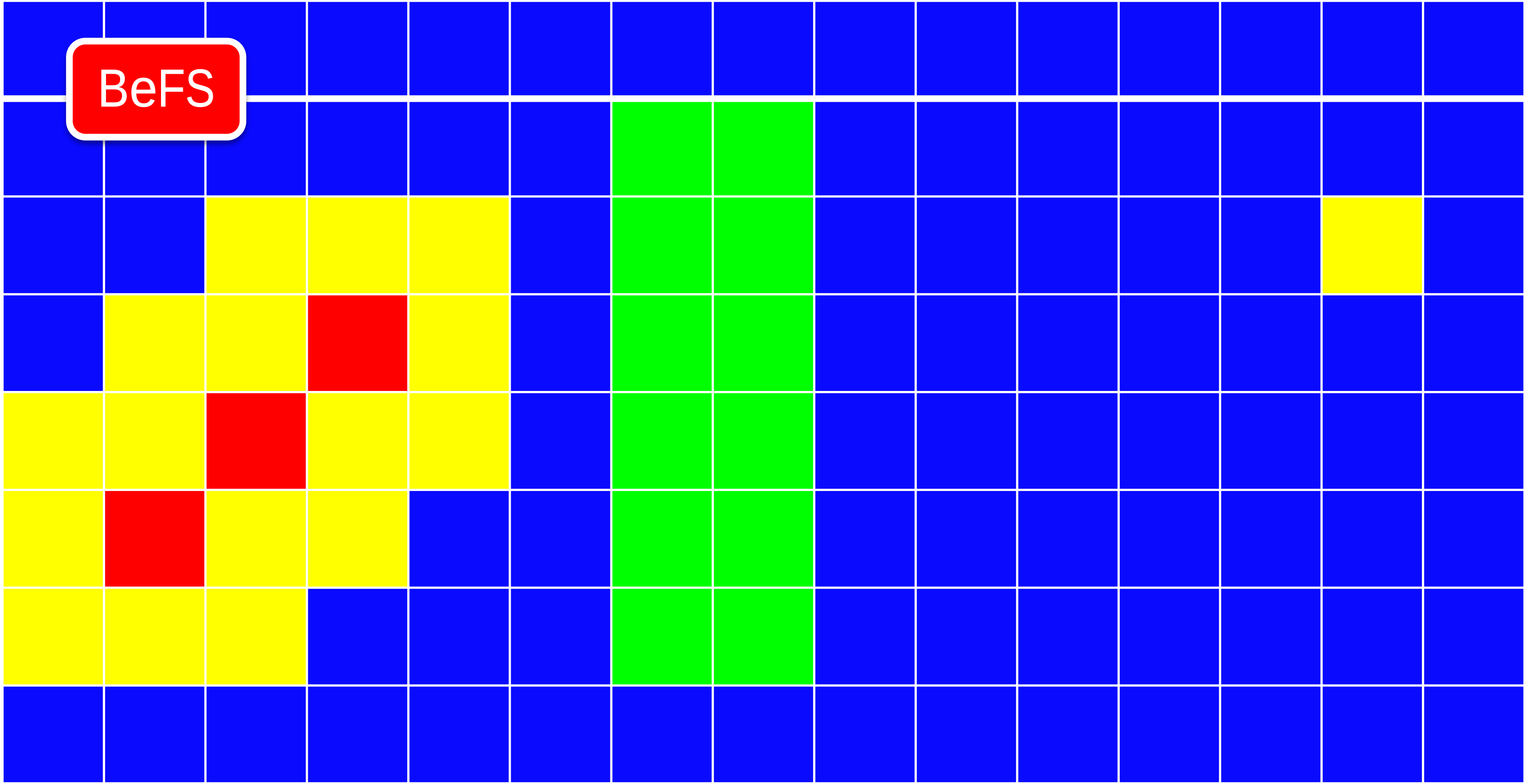

BFS

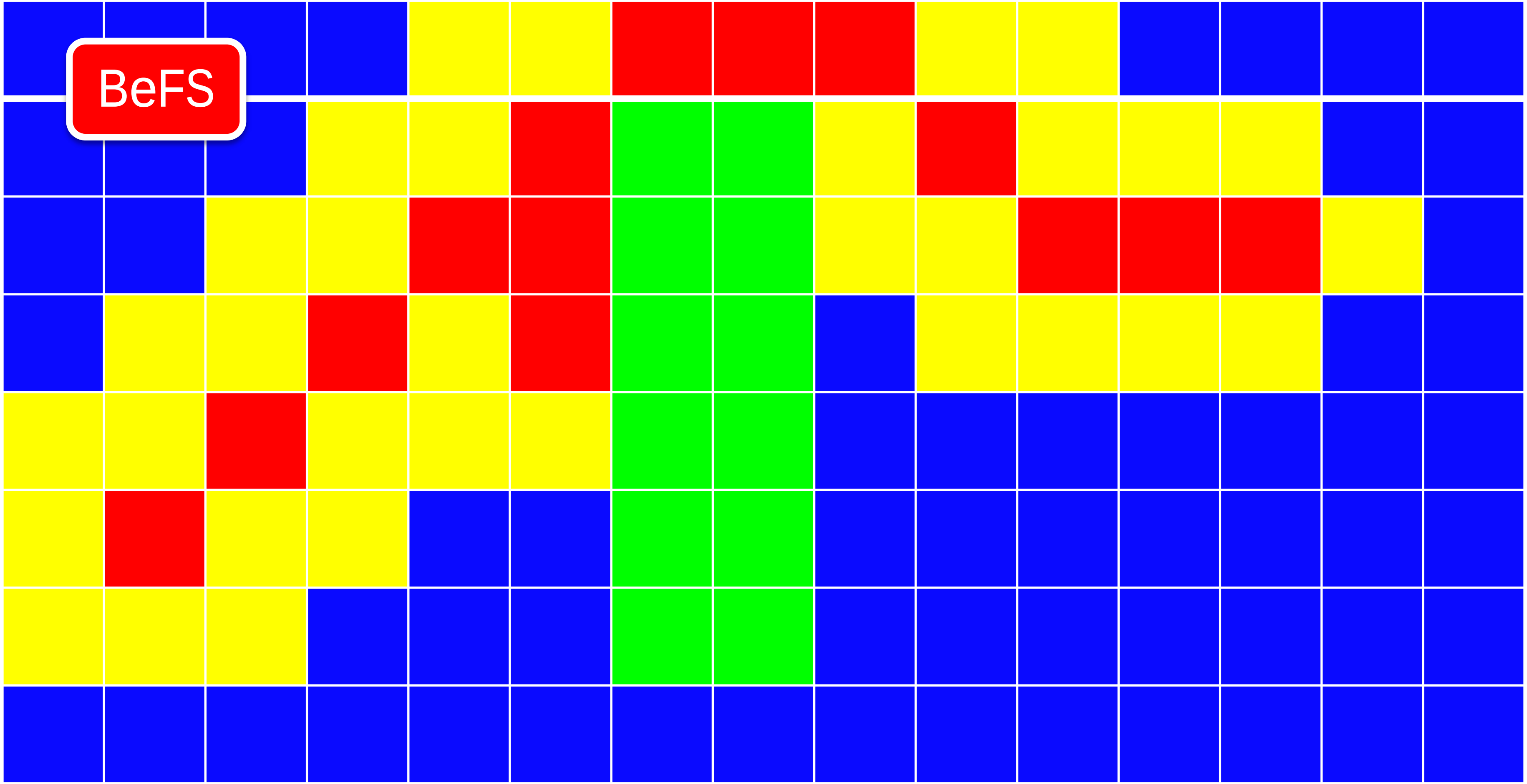

BFS

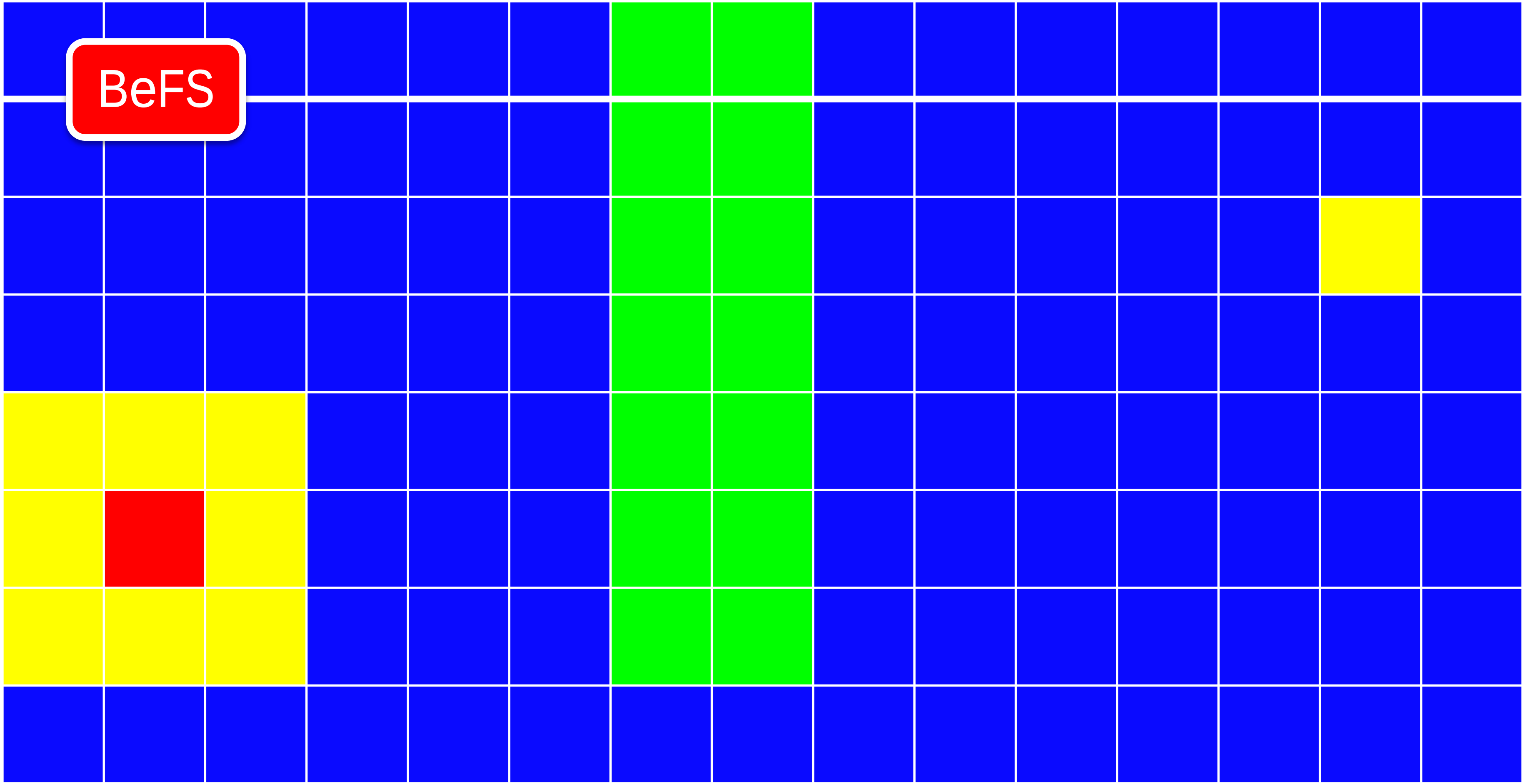

BeFS

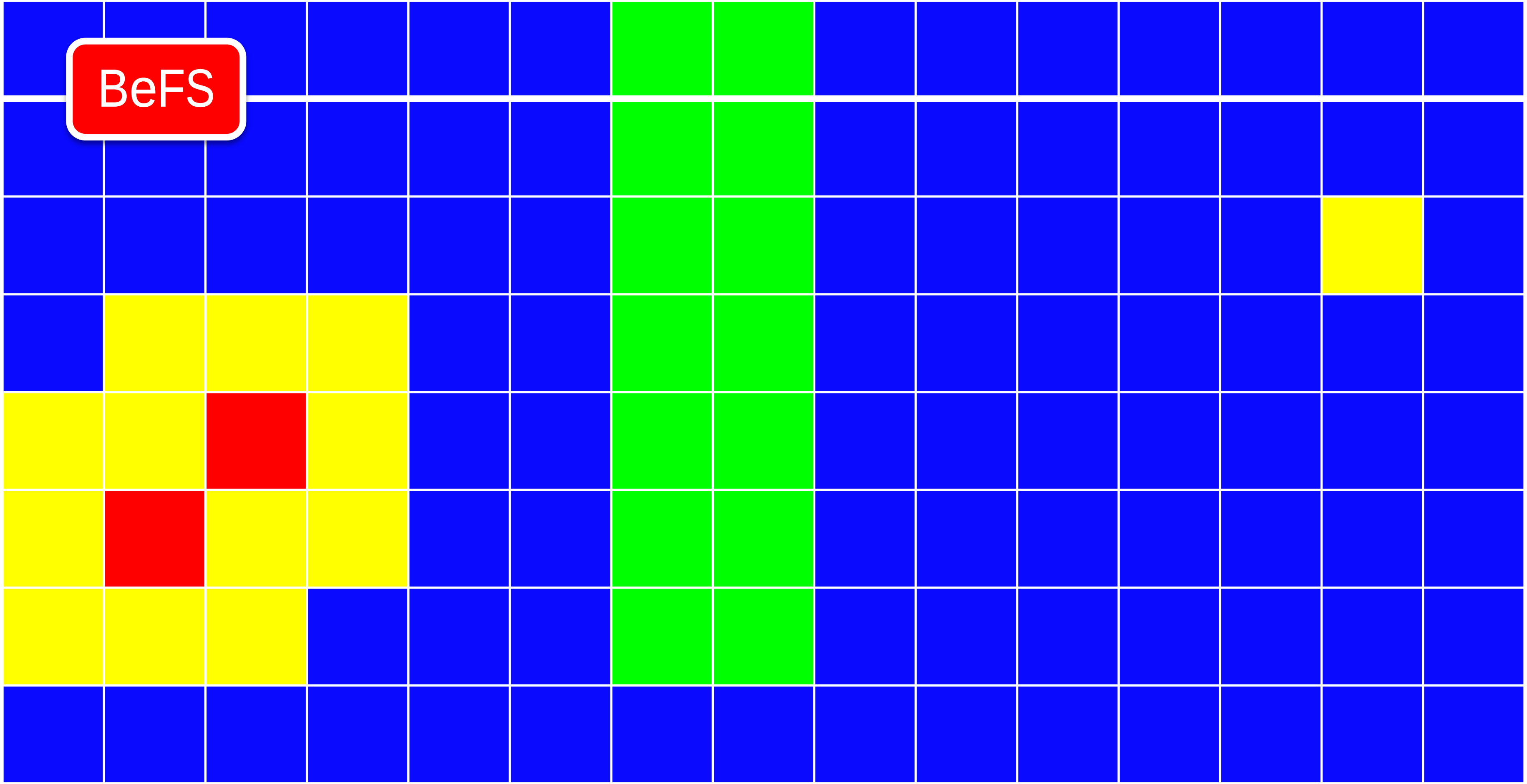

BeFS

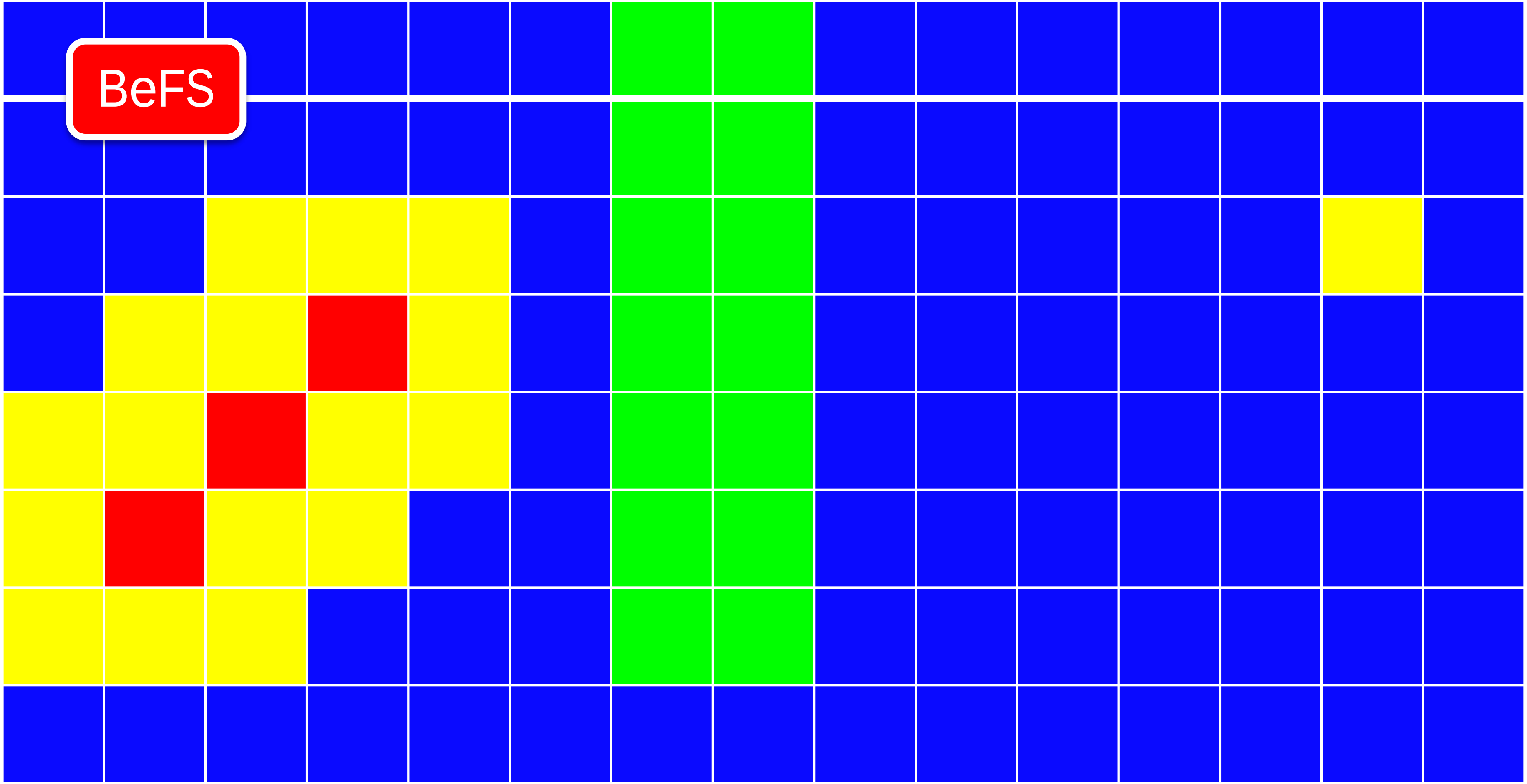

BeFS

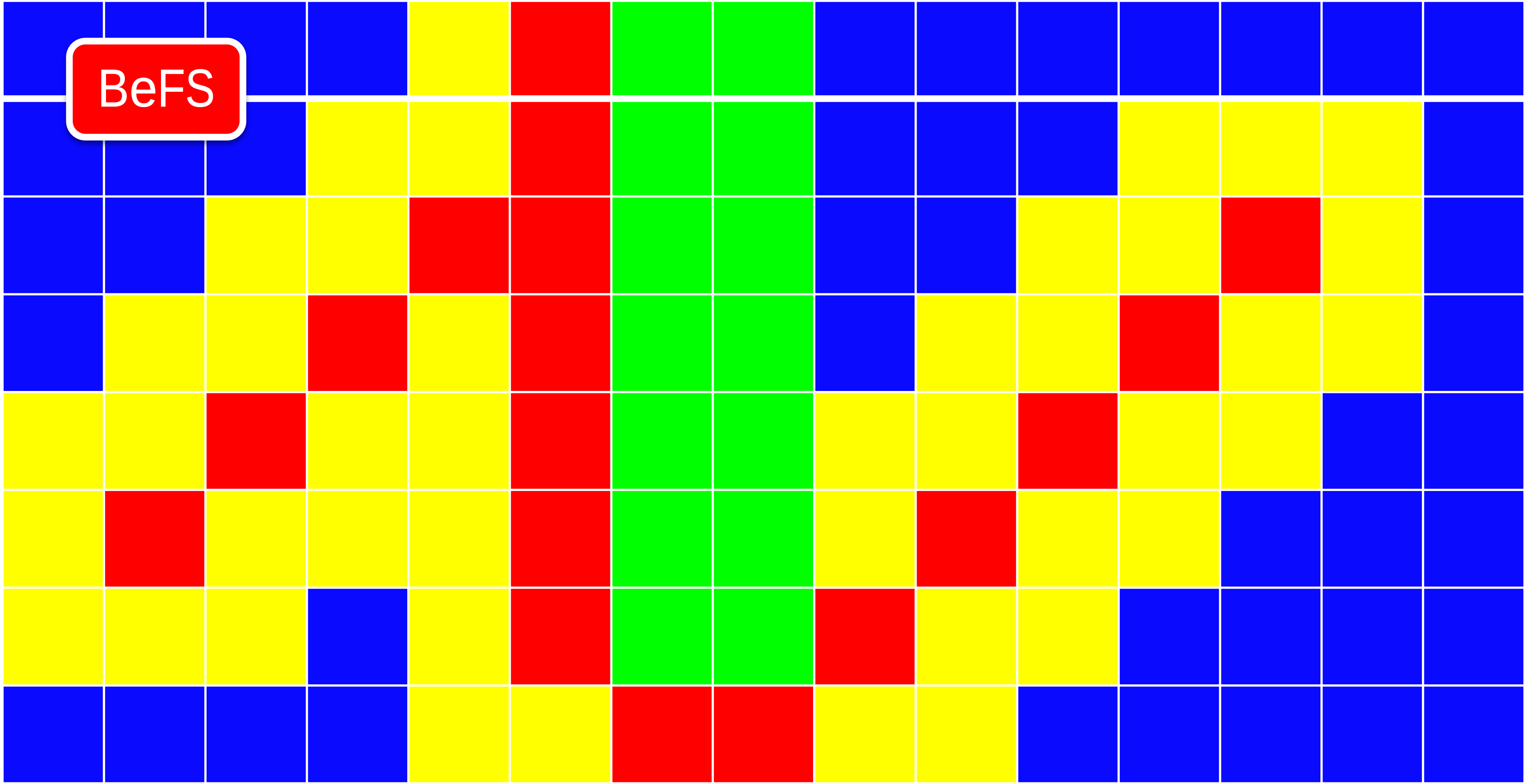

BeFS


BeFS


BeFS


BeFS


BeFS


BeFS

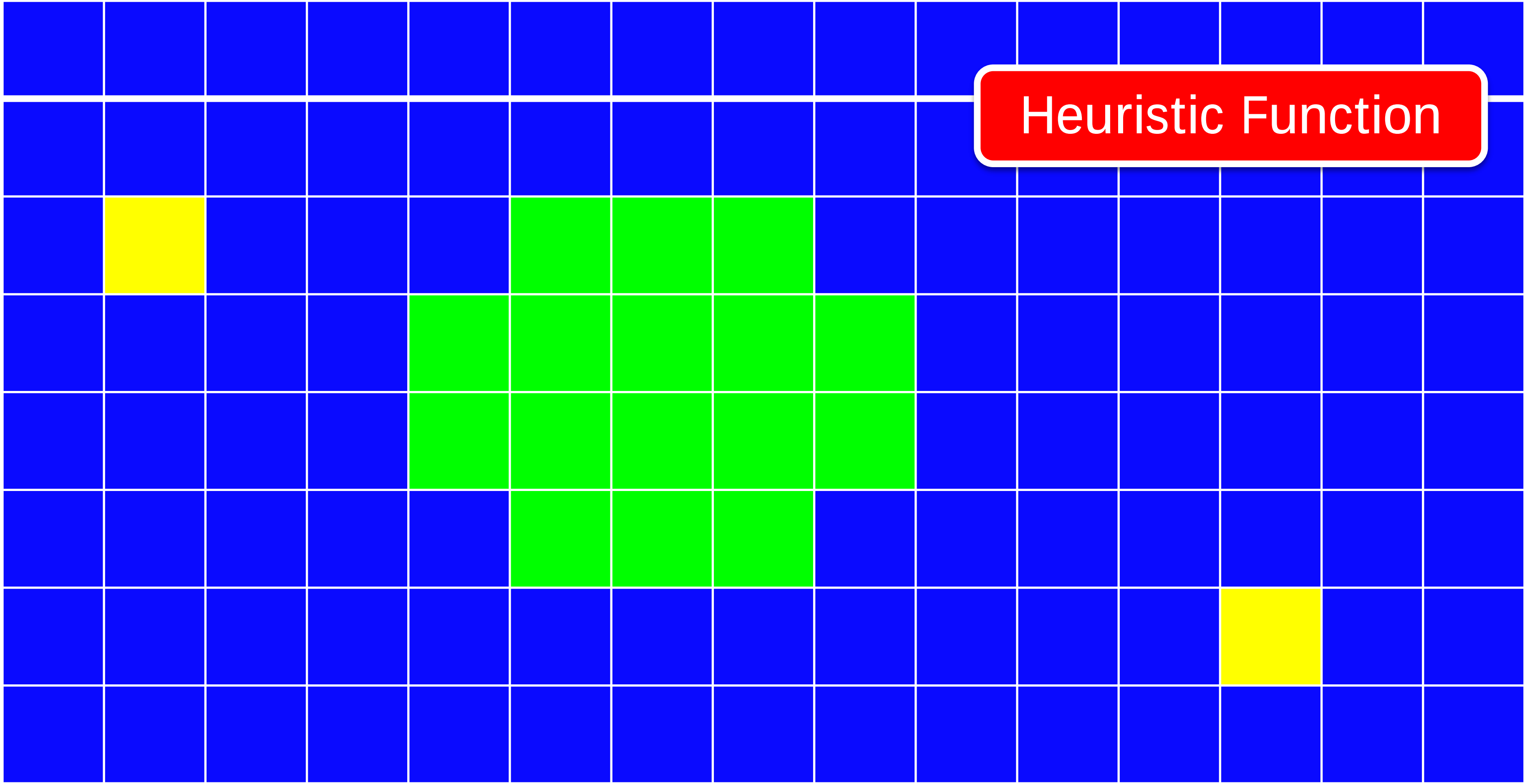
GBeFS Performance

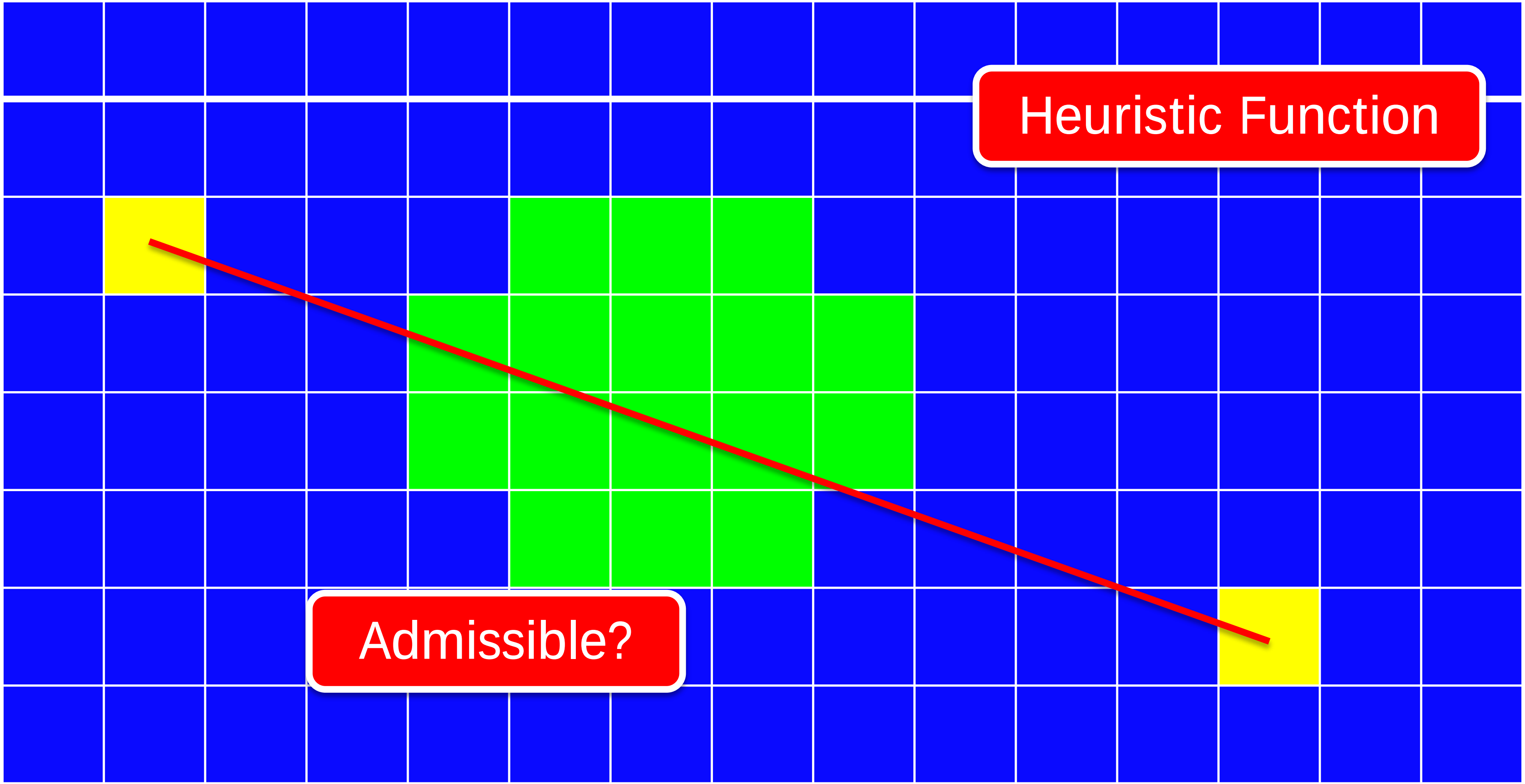
- Suffers similarly to DFS
- Incomplete in General
    - May not find a goal
    - May get lost in paths if heuristic is bad
- Not Optimal
    - May find a higher cost path than optimal
- Time complexity $O(b^m)$, m = max depth

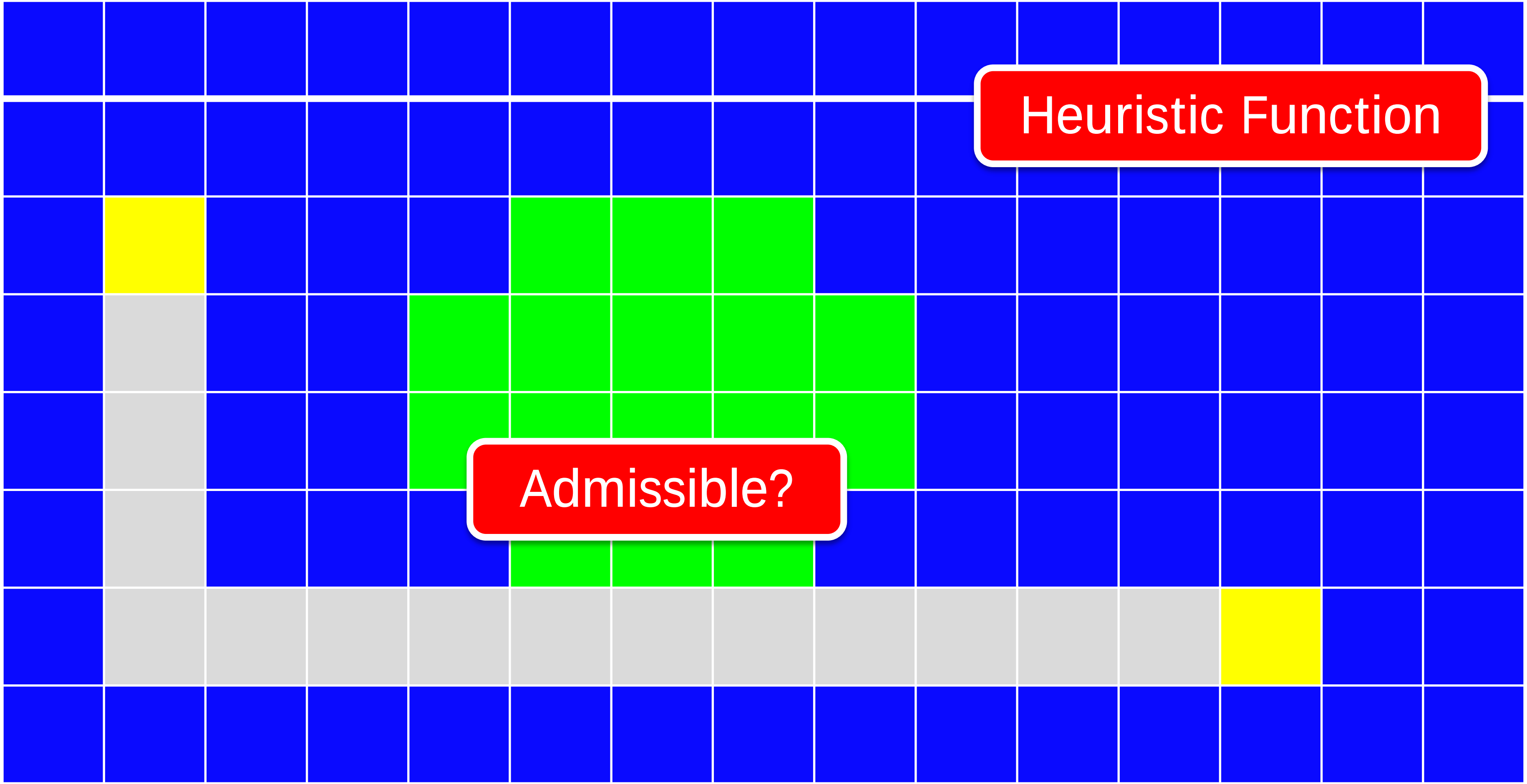
Admissible Heuristic

- Never overestimates distance to goal
- Can be seen as ‘optimistic’ guesses
- $h(n) <= h^*(n)$
- $f(n) = g(n) + h(n)$ never overestimates true cost of a path through n when h(n) is admissible

Heuristic Function

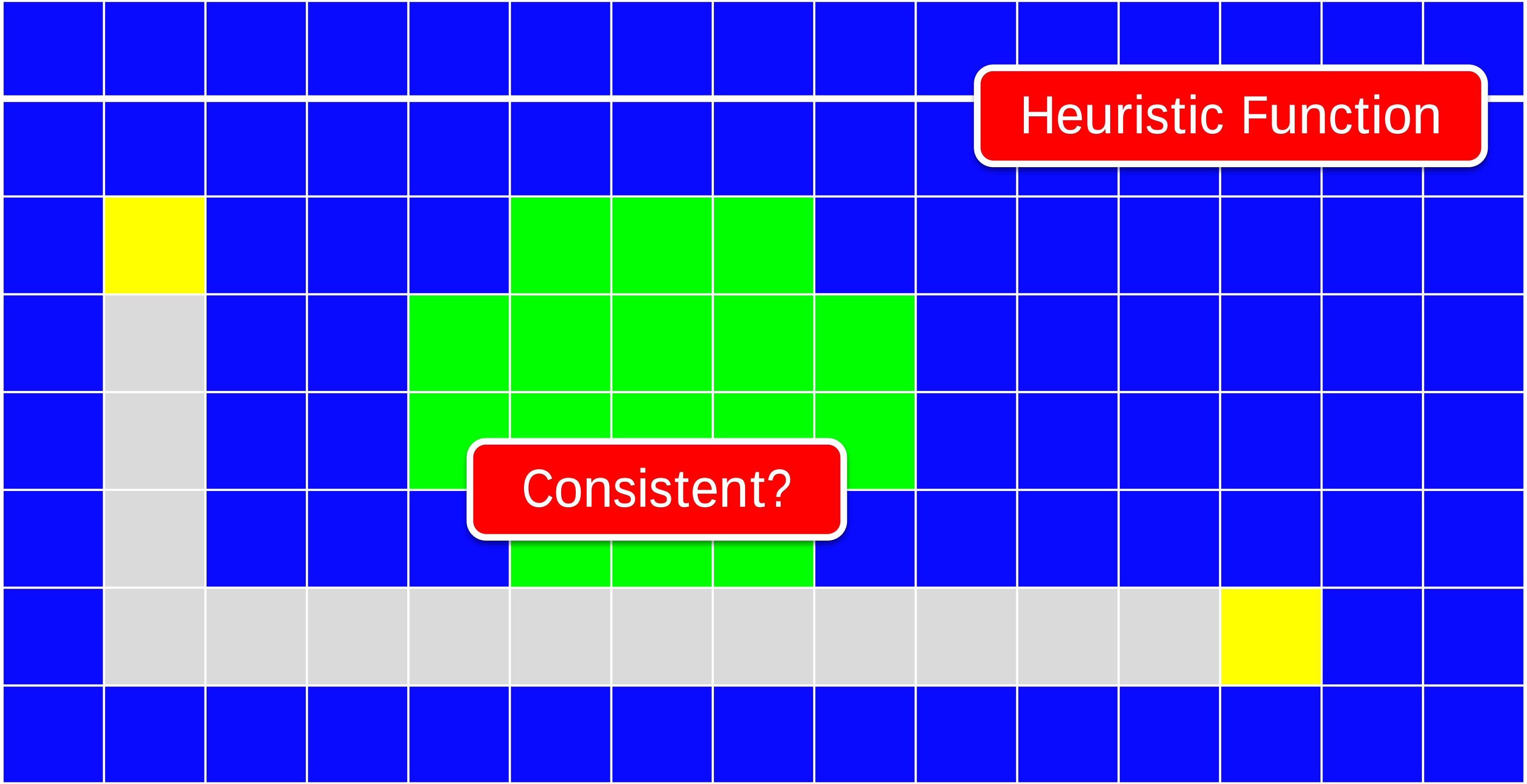



Heuristic Function
Admissible?


Heuristic Function
Admissible?

Consistent Heuristic

- Also called monotone heuristic
- Consistent if $h(n) \leq c(n,a,n') + h(n')$
  - $h(n)$ = estimate path cost from n to goal
  - $c(n,a,n')$ = cost of action a (transitions node n to n')
  - $h(n')$ = estimate path cost from n' to goal
- Estimate of reaching goal from n is always less than the estimate of reaching the goal from n' plus the cost of getting to n' from n


Heuristic Function
Consistent?

Consistent Heuristic

- Every consistent heuristic is also admissible
- In practice, it is hard to construct an admissible heuristic that isn't consistent
- If h(n) is consistent, the values of f(n) along any path are non-decreasing
- The sequence of nodes expanded by A* using Graph-Search is in non-decreasing order of f(n)
- Therefore, first goal node selected for expansion is optimal, since all later nodes are at least as expensive
- A* using Graph-Search is optimal if h(n) is consistent