Breadth-First Search

Searching Problems

The following problems can all be solved the same way:

- finding shortest path(by distance or time) between two location(etc. Google maps)
- solving Rubik's cube, 8-puzzle
- word segmentation(given a sequence of letters without spaces, insert spaces to form words, e.g. "haveagoodday" -> "have a good day")

Each of these problems can be defined as a search problem and represented by a **graph**. A solution may be found by applying a graph search algorithm.

Searching Problem

A search problem can be defined formally as follows:

- A set of possible **states** that the environment can be in. We call this the **state space**. For example, each position of a Rubik's cube is a state. The state space is the set of all possible positions.
- The **initial state** that the agent starts in. For example: *initial position of the Rubik's cube*.
- A set of one or more **goal states**. For example: *solved state of the Rubik's cube*.
- The **actions** available to the agent. An example of an action: turn the front face 90 degrees clockwise.
- A **transition model**, which describes what each action does. RESULT(*s*,*a*) returns the state that results from doing action *a* in state *s*. For example, consider a state which is a location on the x-y plane and the action moveUp which moves up one unit. Then RESULT(*(3, 1)*, *moveUp*) = (3, 2).

The state space can be represented as a **graph** in which the **vertices** are states and the **directed edges** between them are actions.

Rubik’s Cube

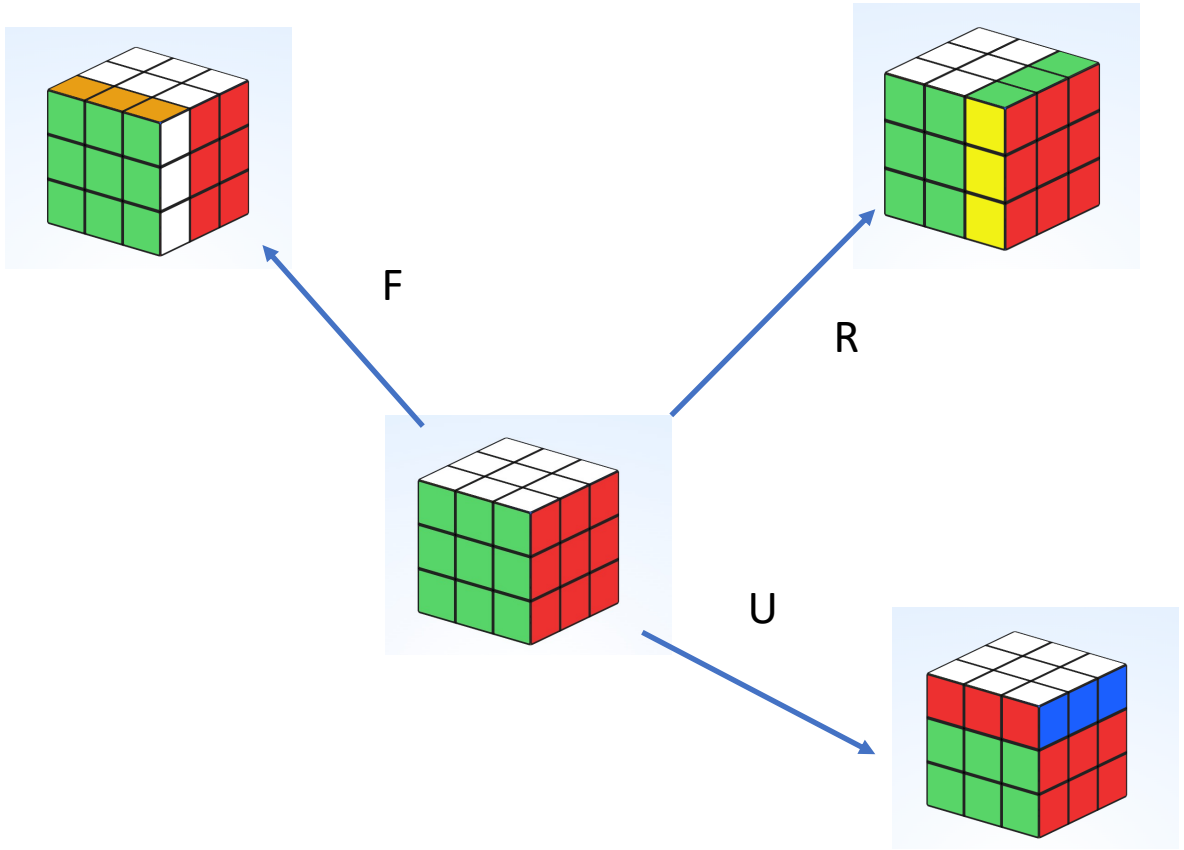
These four cubes represents four states of a Rubik’s cube.
Each state is a vertex in the graph.

An action is an allowed move that takes one
state to another state. An action is an edge in the graph.

For example, at the solved state, one action is F which is
rotating the front face(green) 90 degrees clockwise.

R action is rotating the right side(red) 90 degrees clockwise.
U action is rotating the up side(white) 90 degrees clockwise.

On the picture here, the three resulting states are
three neighbors of the solved state.



Search Algorithm

A **search algorithm** takes a search problem as input and returns a solution, or an indication of failure.

We consider algorithms that superimpose a **search tree** over the state-space graph, forming various paths from the initial state, trying to find a path that reaches a goal state.

Each **node** in the search tree corresponds to a state in the state space and the edges in the search tree correspond to actions. The root of the tree corresponds to the initial state of the problem.

Breadth-first search(BFS) is search algorithm which starts at the initial state and first search the state nearest neighbors and then expanding the search outward to other neighbors systematically.

BFS keeps track of nodes it has already visited as well as frontier nodes. Frontier nodes are nodes that are at the boundary of the search. Think of this algorithm as an expanding circle where the nodes in the circle have already been visited and the frontier nodes are the nodes at the perimeter of the circle.

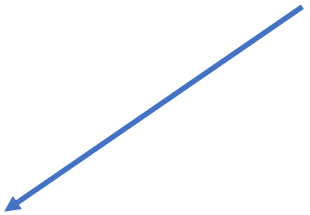
Breadth-First Search

Algorithm: Given a search problem, return the solution state or failure

```
def function(search problem):
    frontier = []
    visited = []
    add initial state to frontier

    while frontier is not empty:
        remove first node from frontier list, call it s.
        if s is goal state:
                we found our goal; return path to goal
        if s is not in visited list:
                add s to visited list
                expand at s by creating list of neighbor nodes to s
                for each neighbor of s:
                        add neighbor to frontier list
    return failure
```

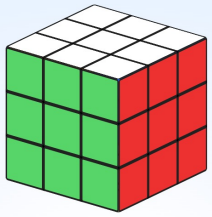
Expanding at s means to
create a list of neighbor nodes of s
where each neighbor node contains a pointer
to s. This allows us to recover the solution path
back to s once we find our goal state.



Expanding at a node(get neighbors)

Expanding at s means to create a list of neighbor nodes of s where each neighbor node contains a pointer to s.

This allows us to recover the solution path back to s once we find our goal state.



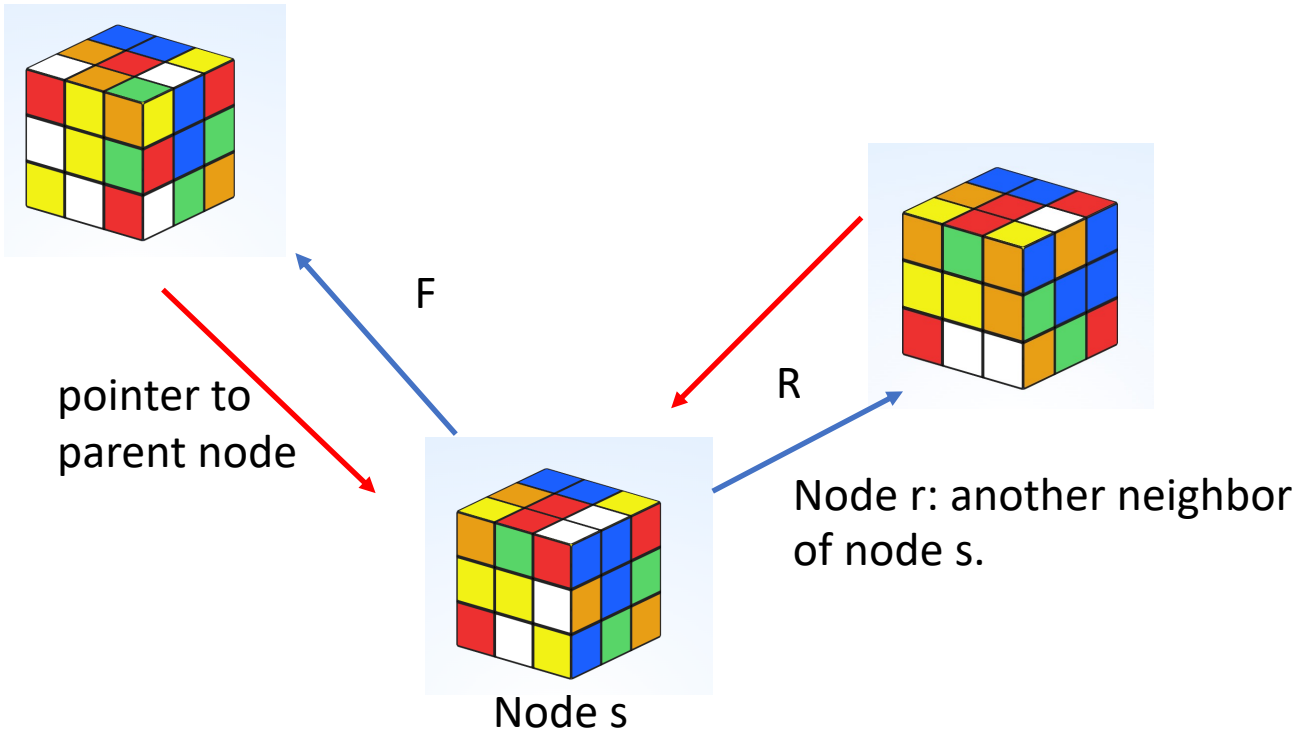
Expanding at a node(get neighbors)

Expanding at s means to create a list of neighbor nodes of s.

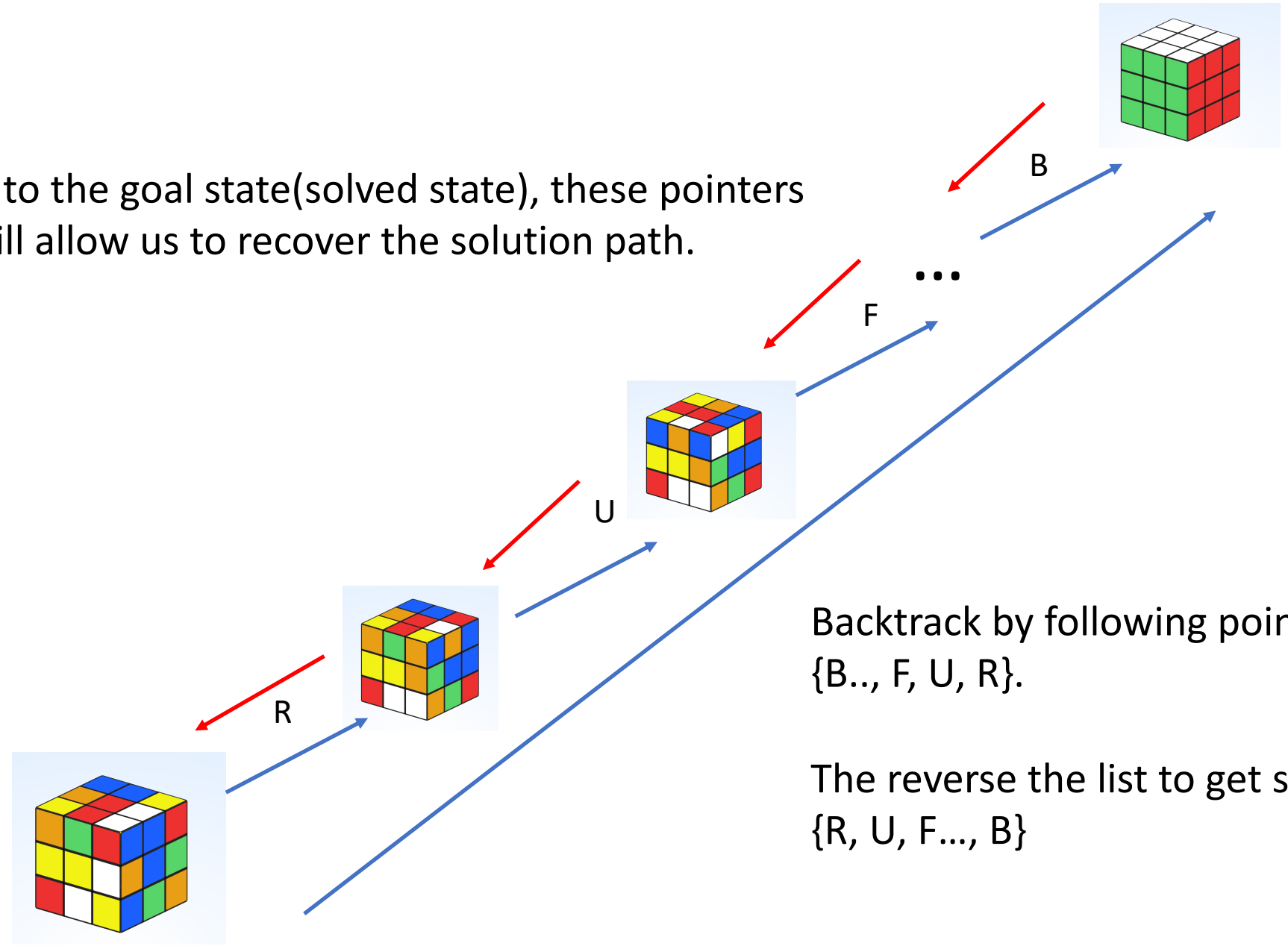
Each neighbor node contains the following information:
- state of node
- parent node(in this case, node s)
- action that gets to this node

For example, the node s in the picture is the current node.

Node t is a neighbor of node s and contains:
- state of node t(see cube position in picture)
- pointer to parent which is node s
- action that gets from node s to node t is the action F.



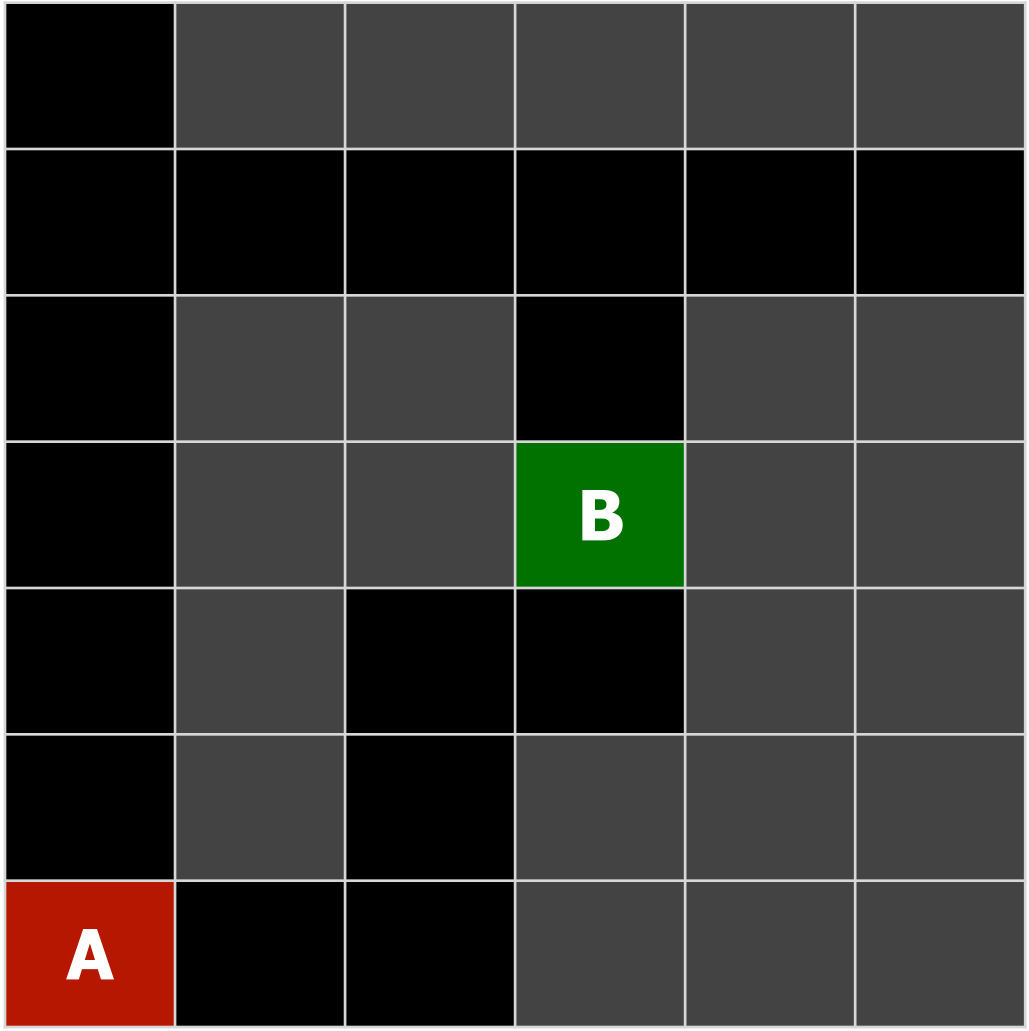
Once we get to the goal state(solved state), these pointers to parents will allow us to recover the solution path.



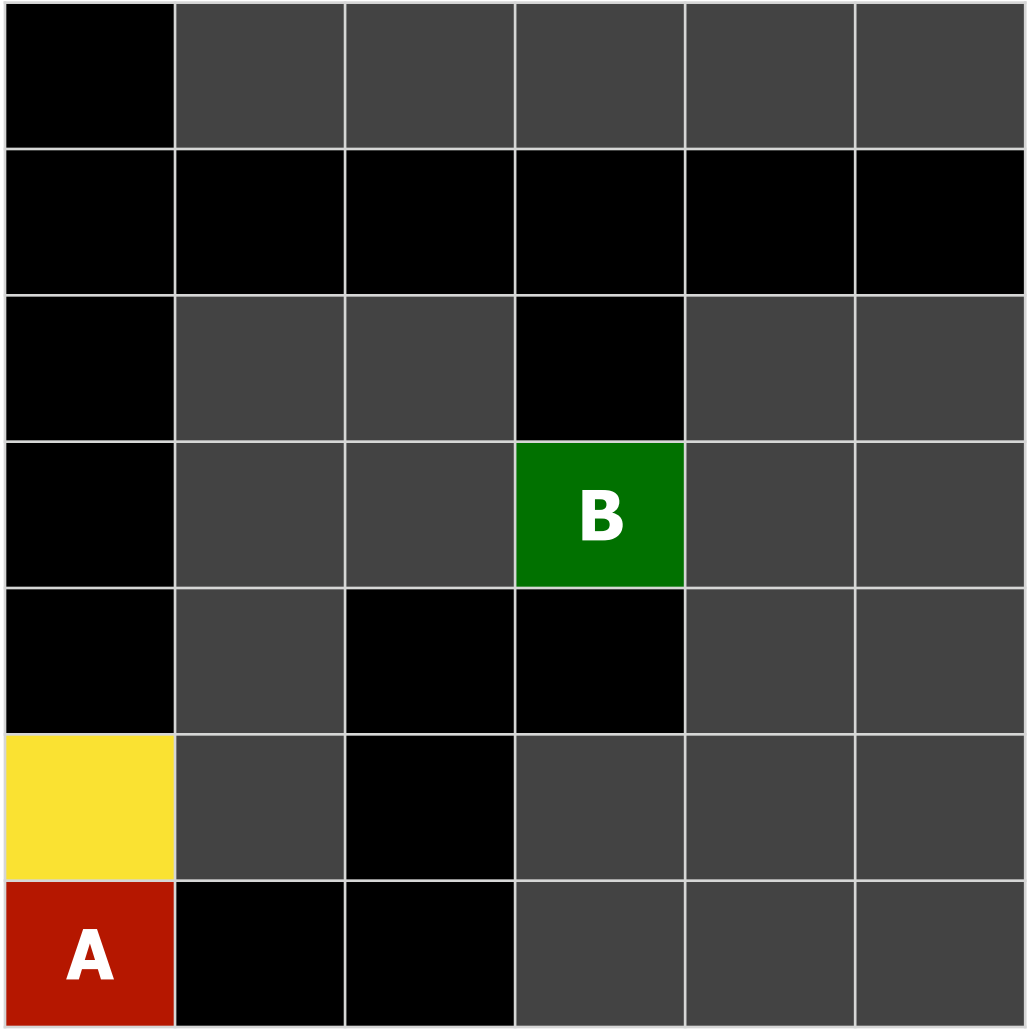
Backtrack by following pointers backwards: {B.., F, U, R}.

The reverse the list to get solution path: {R, U, F…, B}

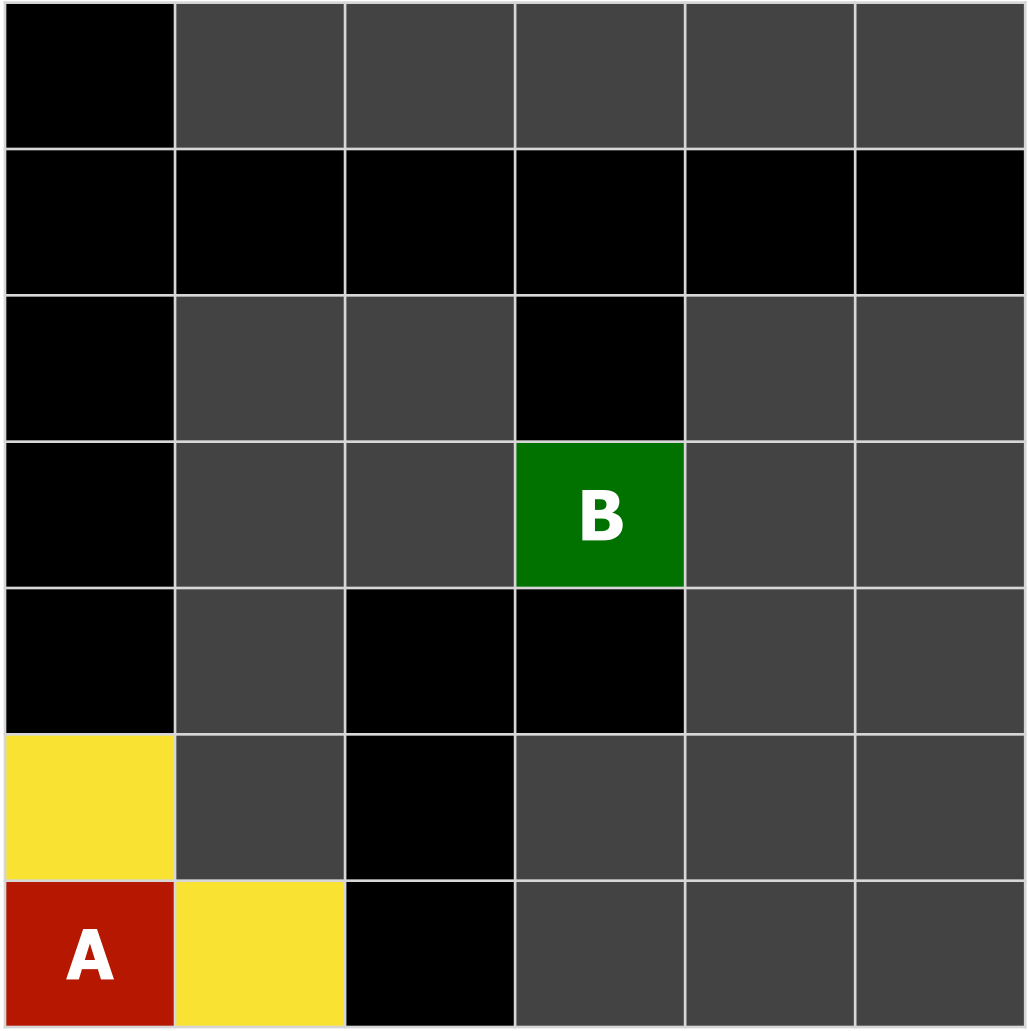
Breadth-First Search



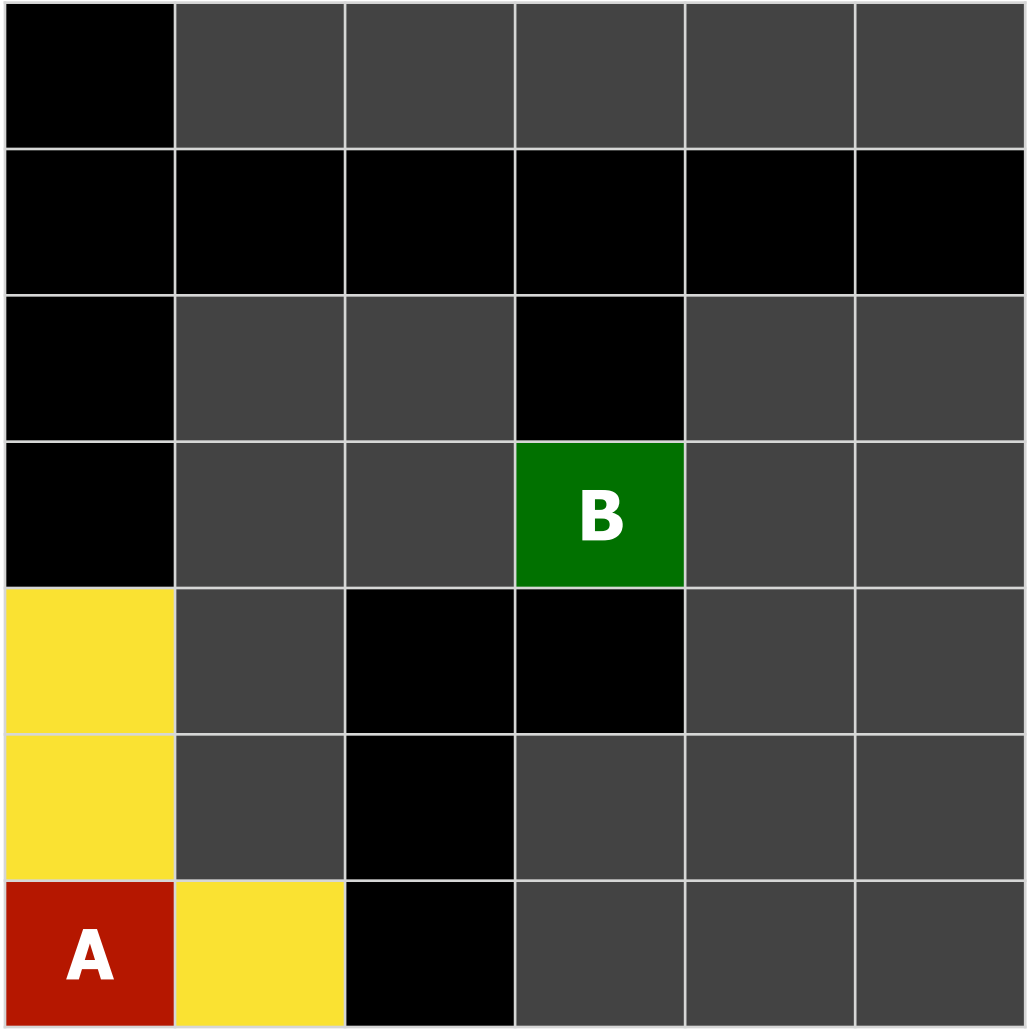
Breadth-First Search



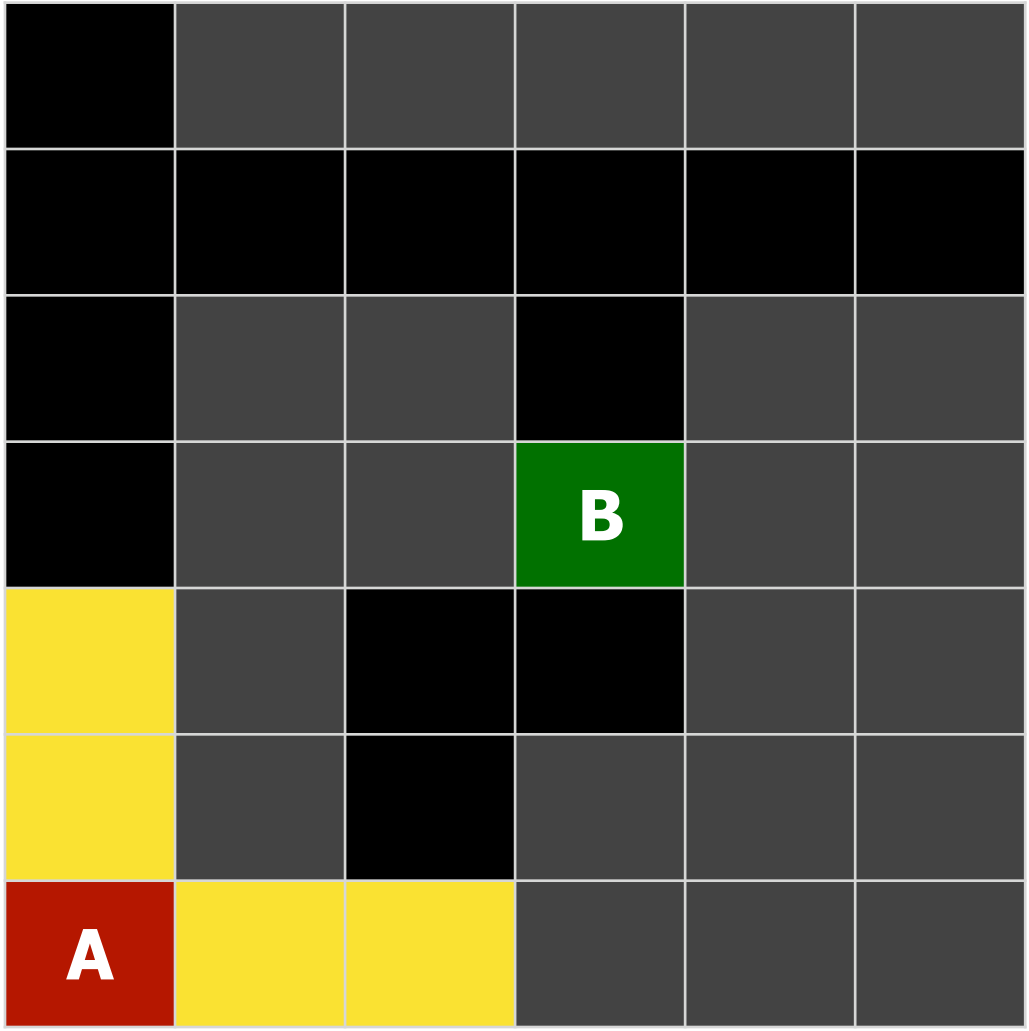
Breadth-First Search



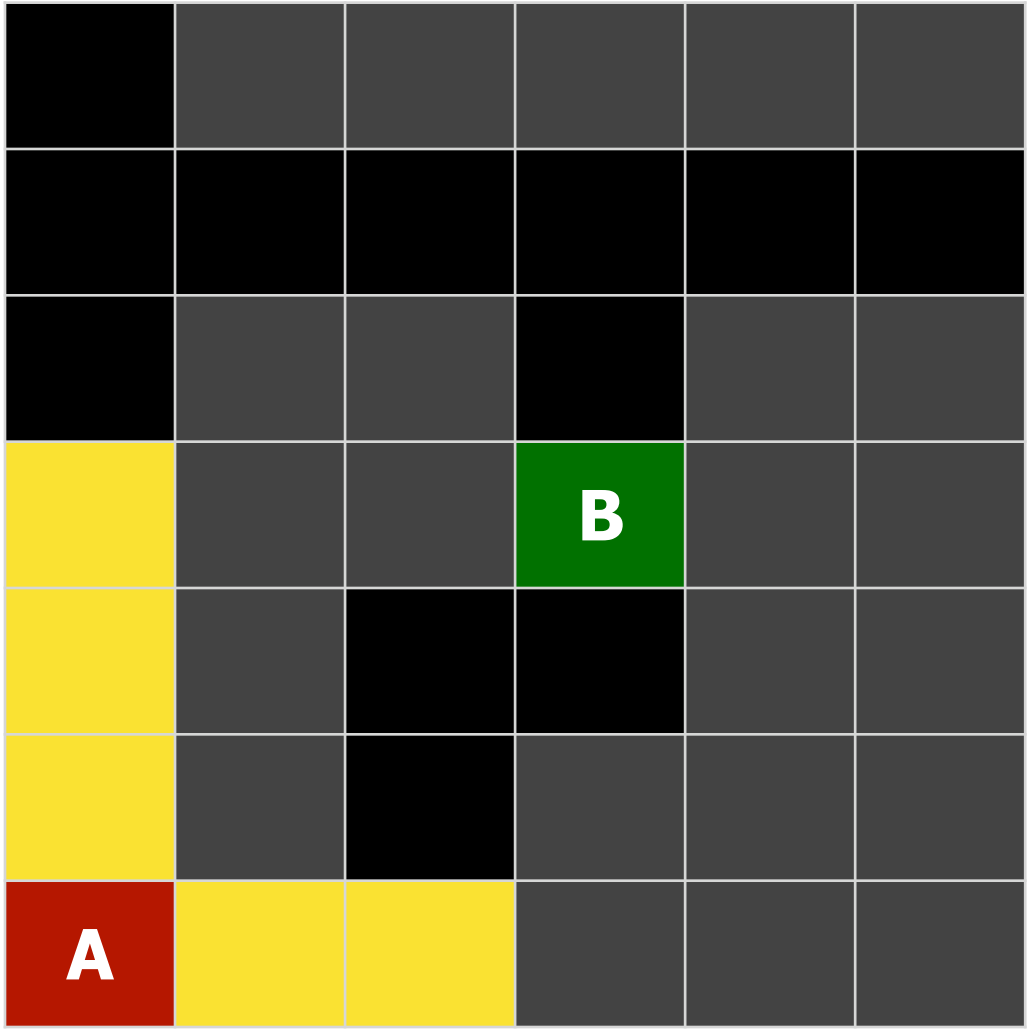
Breadth-First Search



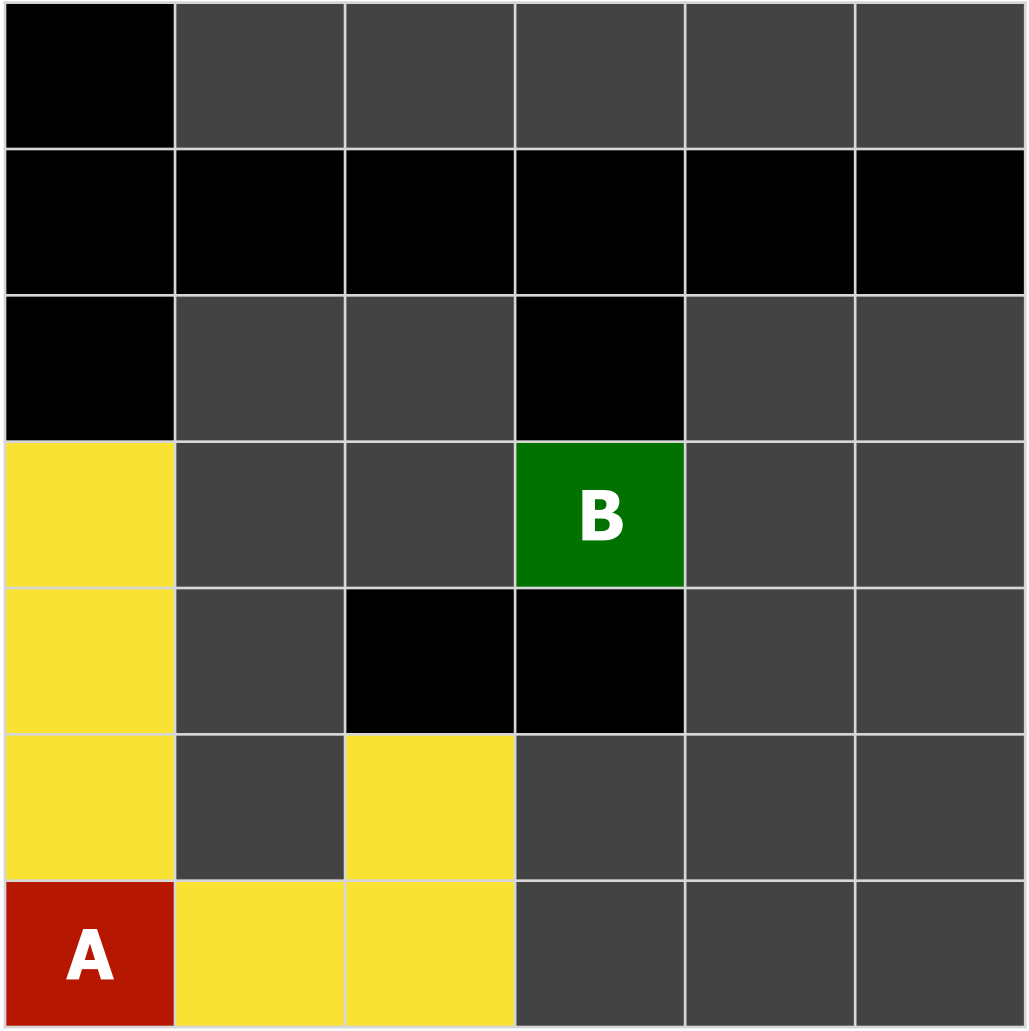
Breadth-First Search



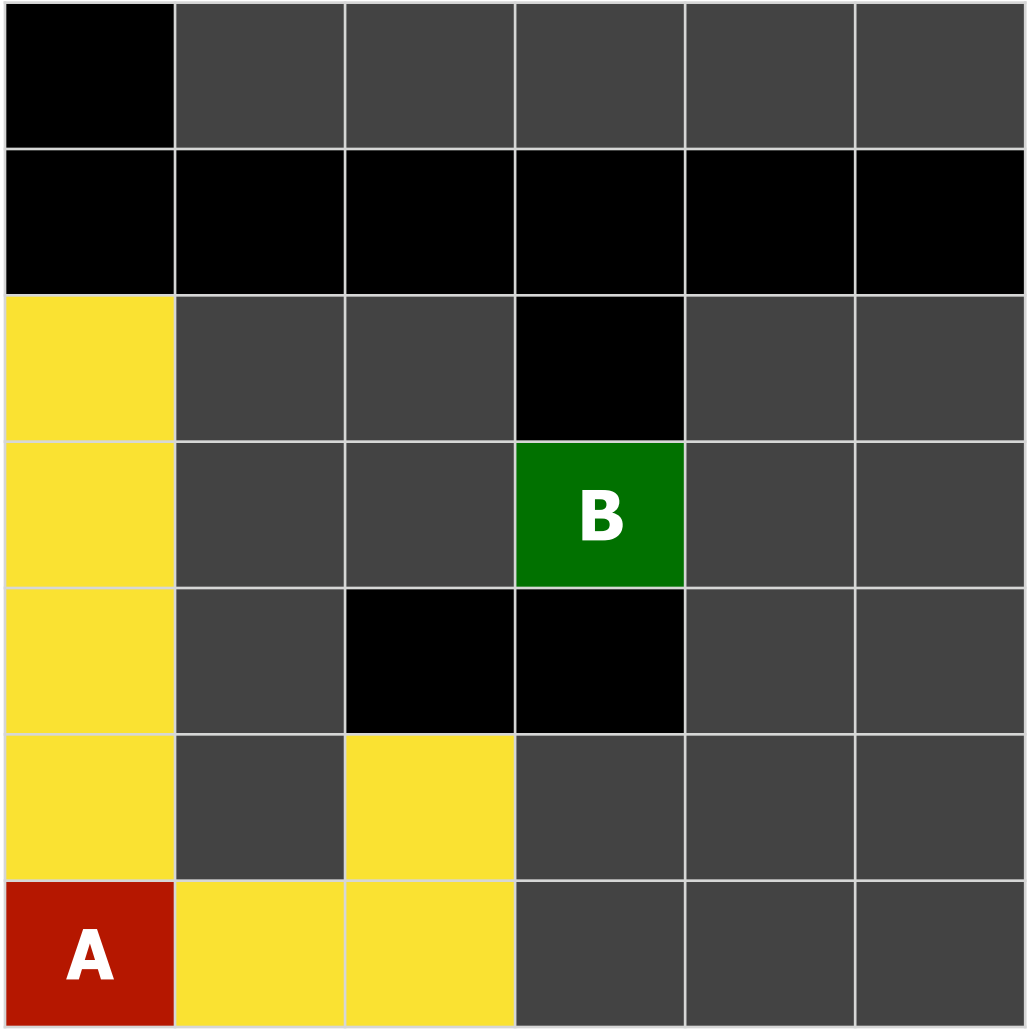
Breadth-First Search



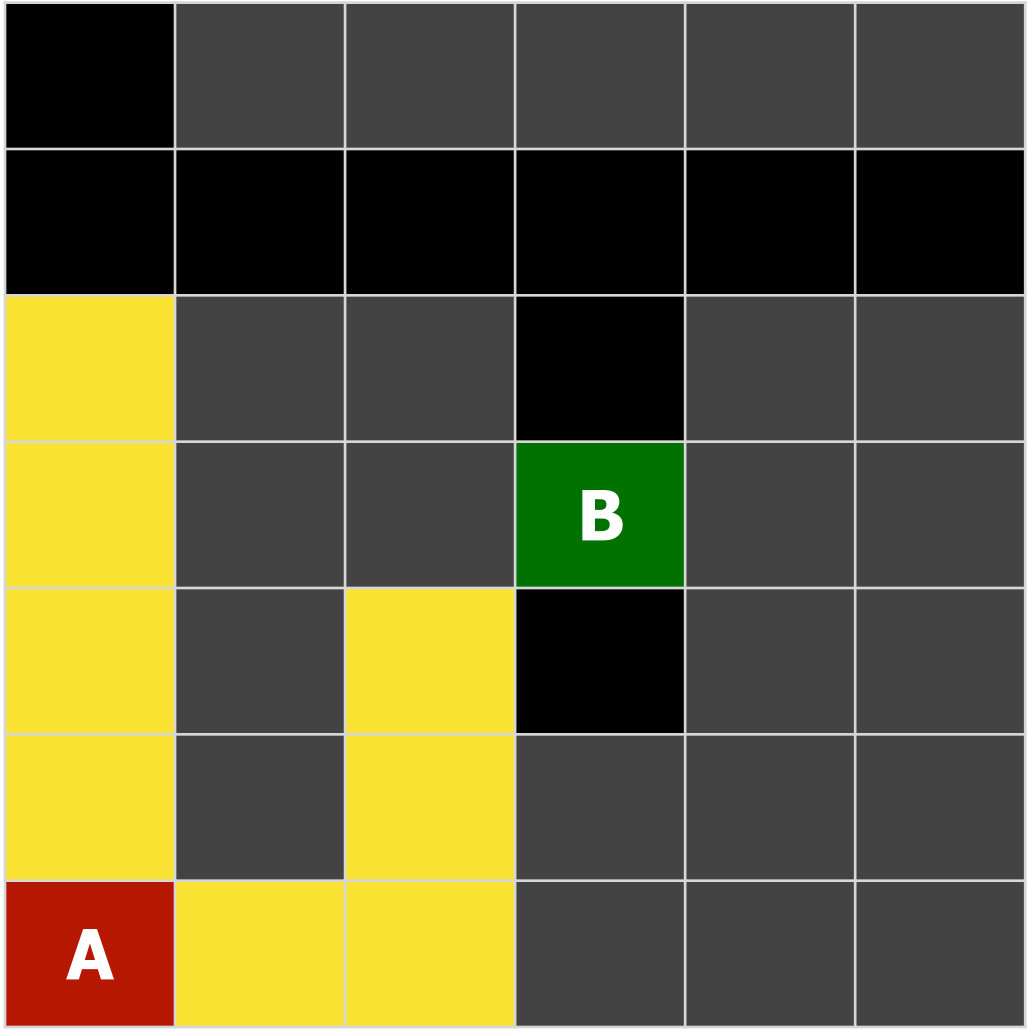
Breadth-First Search



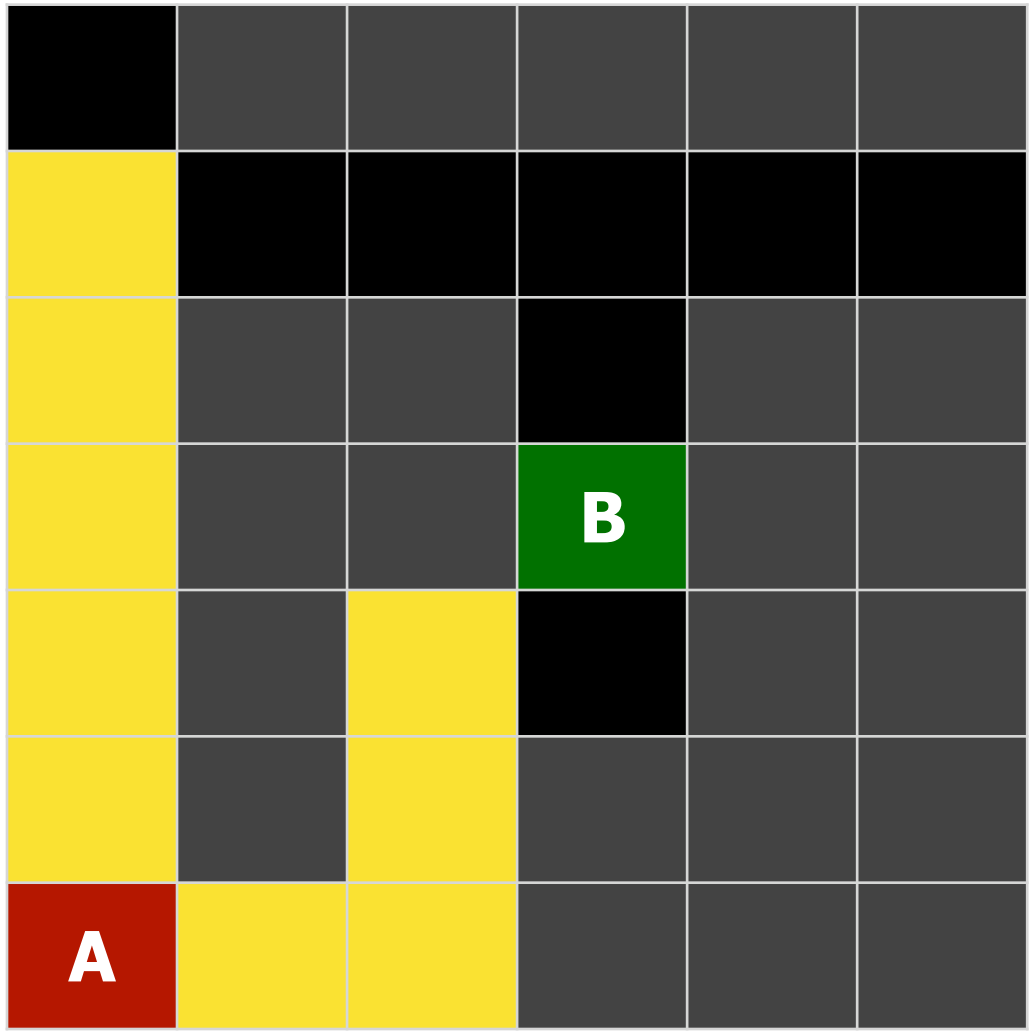
Breadth-First Search



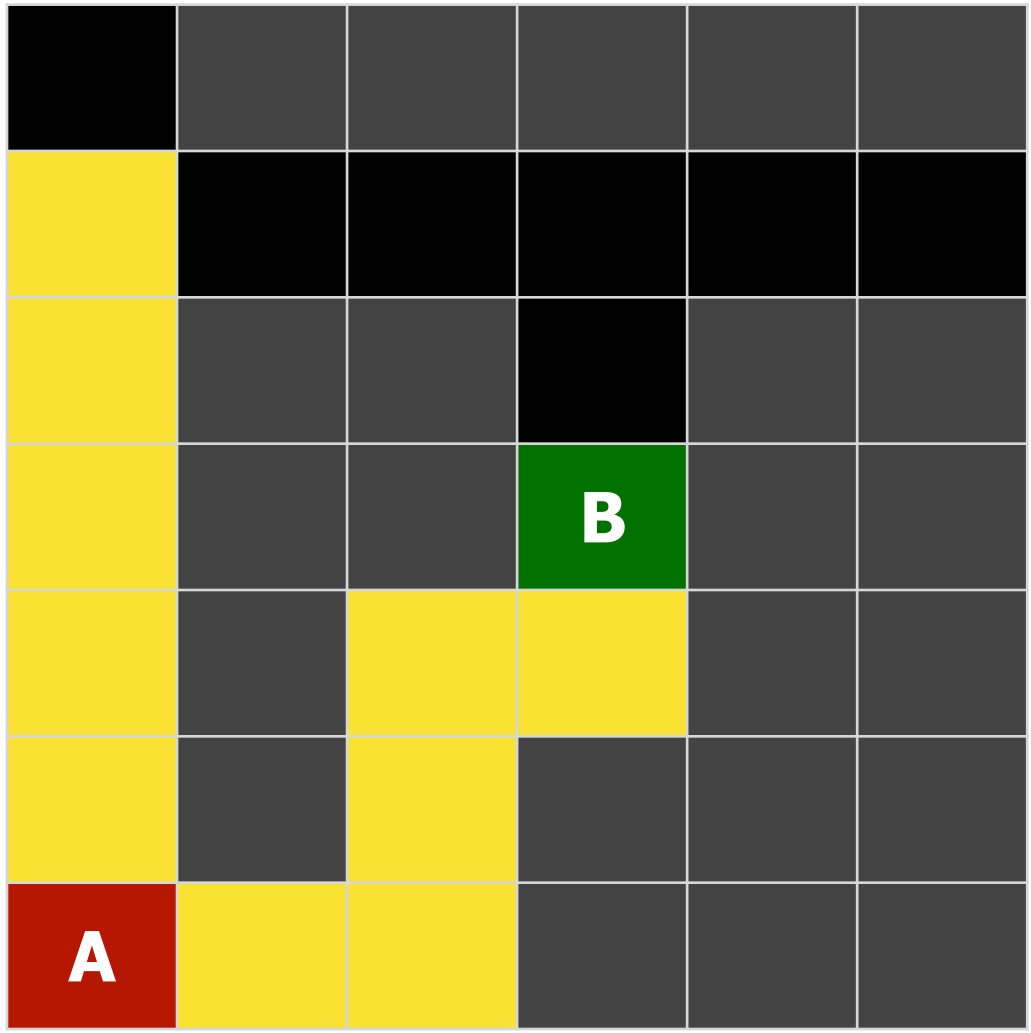
Breadth-First Search



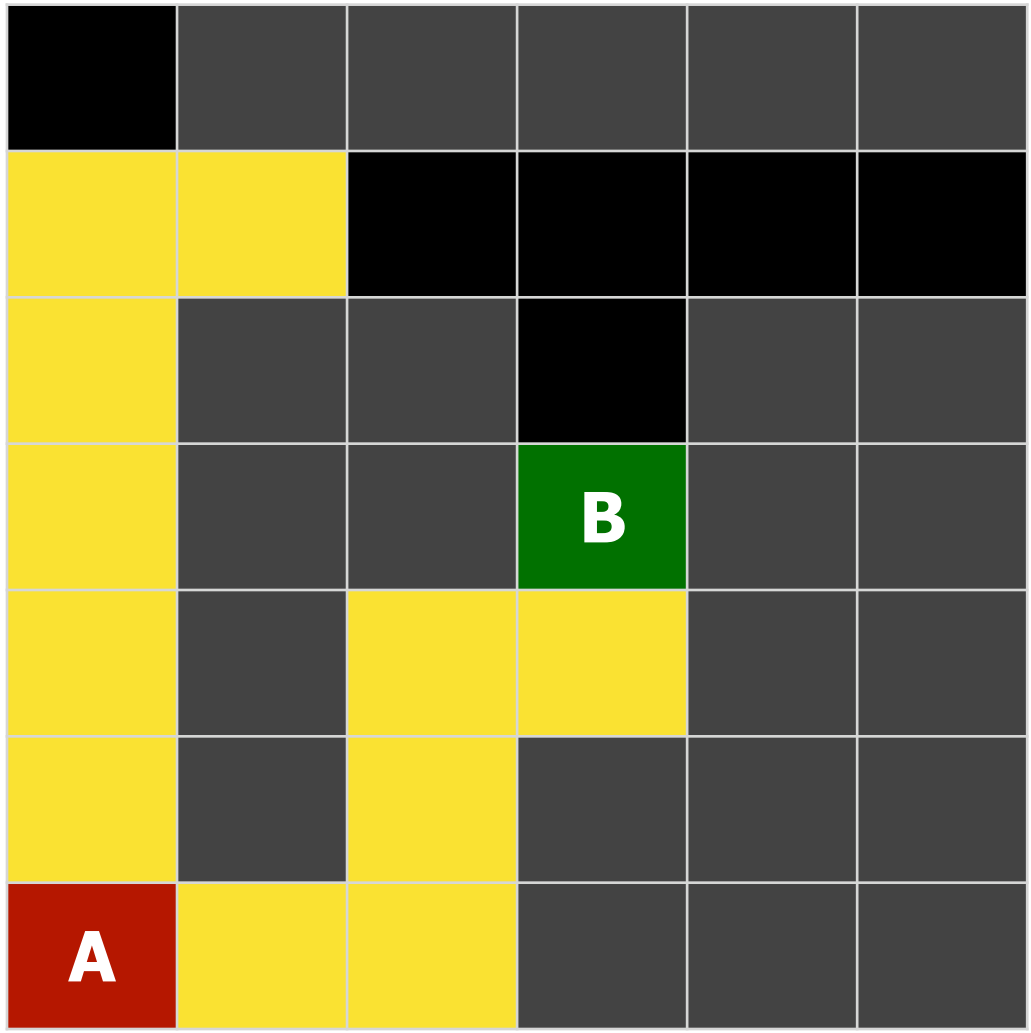
Breadth-First Search



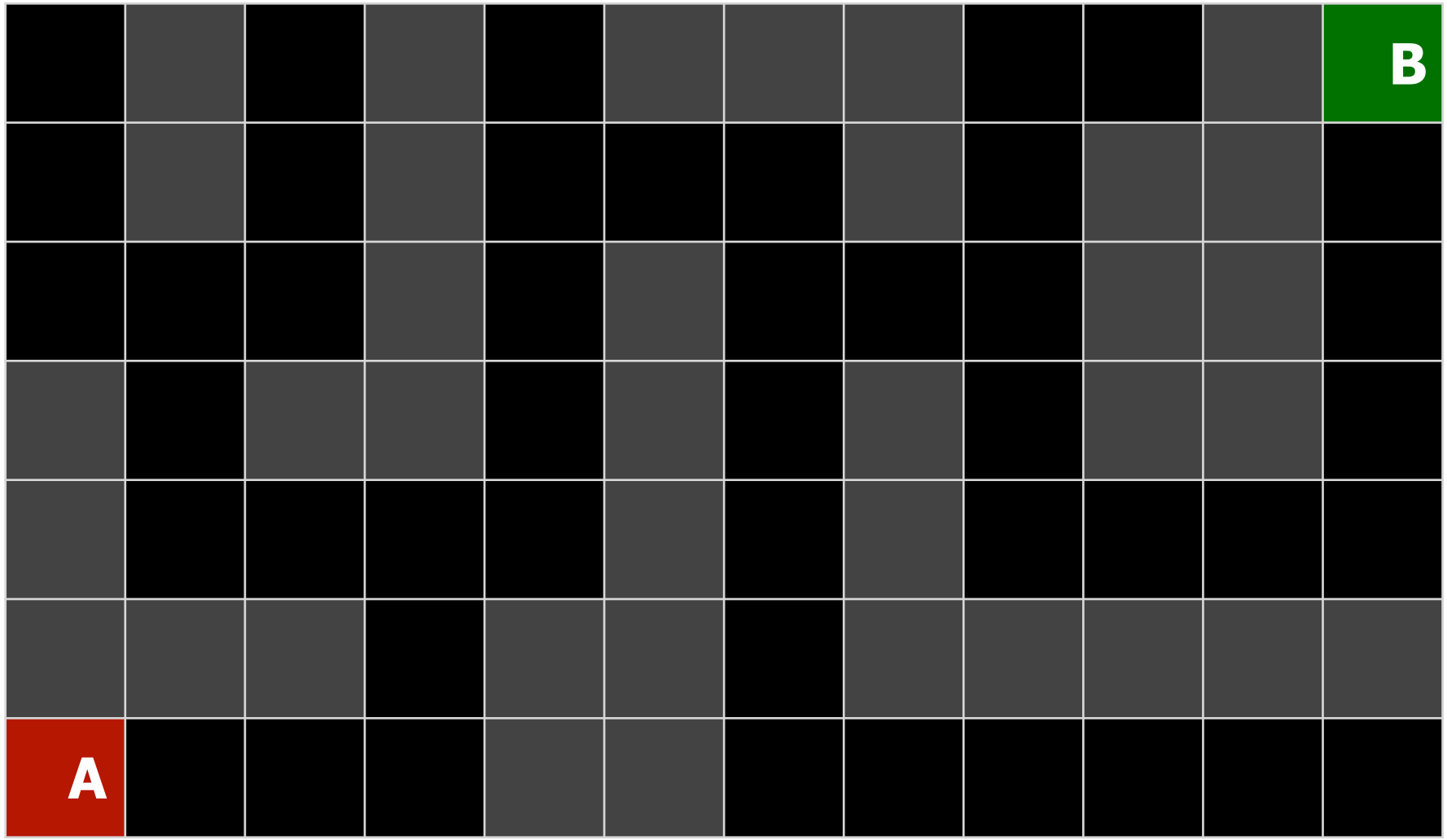
Breadth-First Search



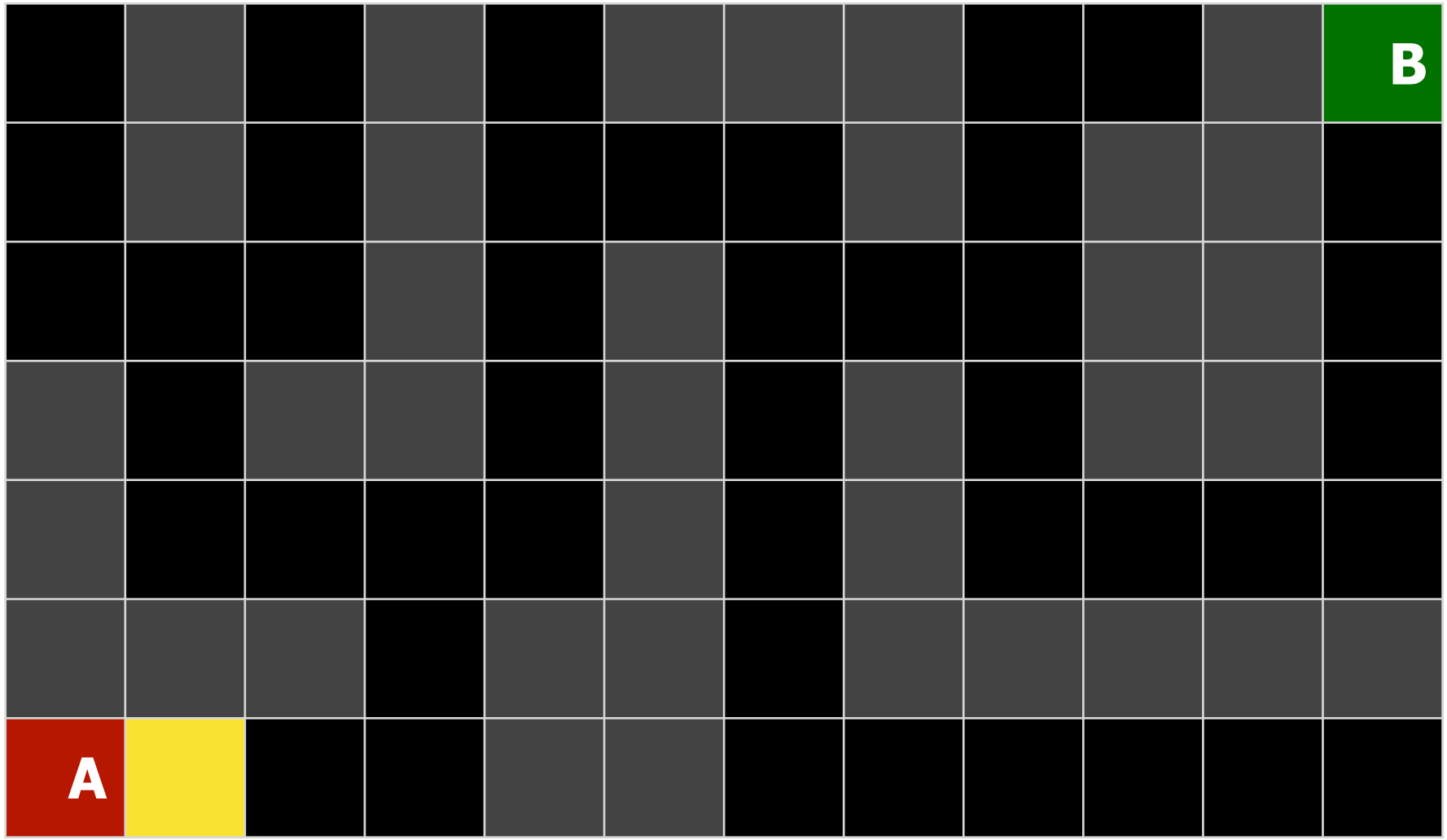
Breadth-First Search



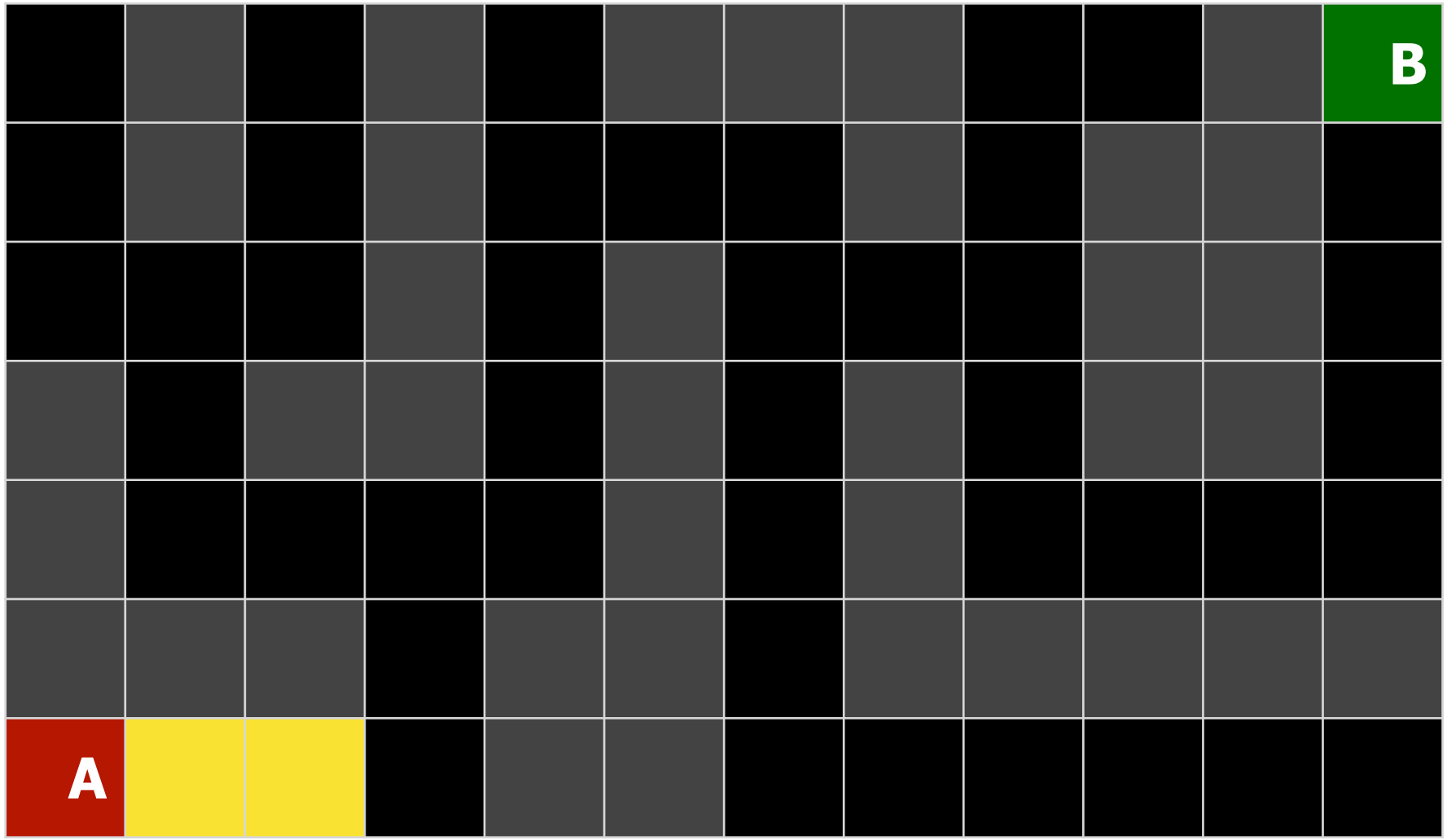
Breadth-First Search



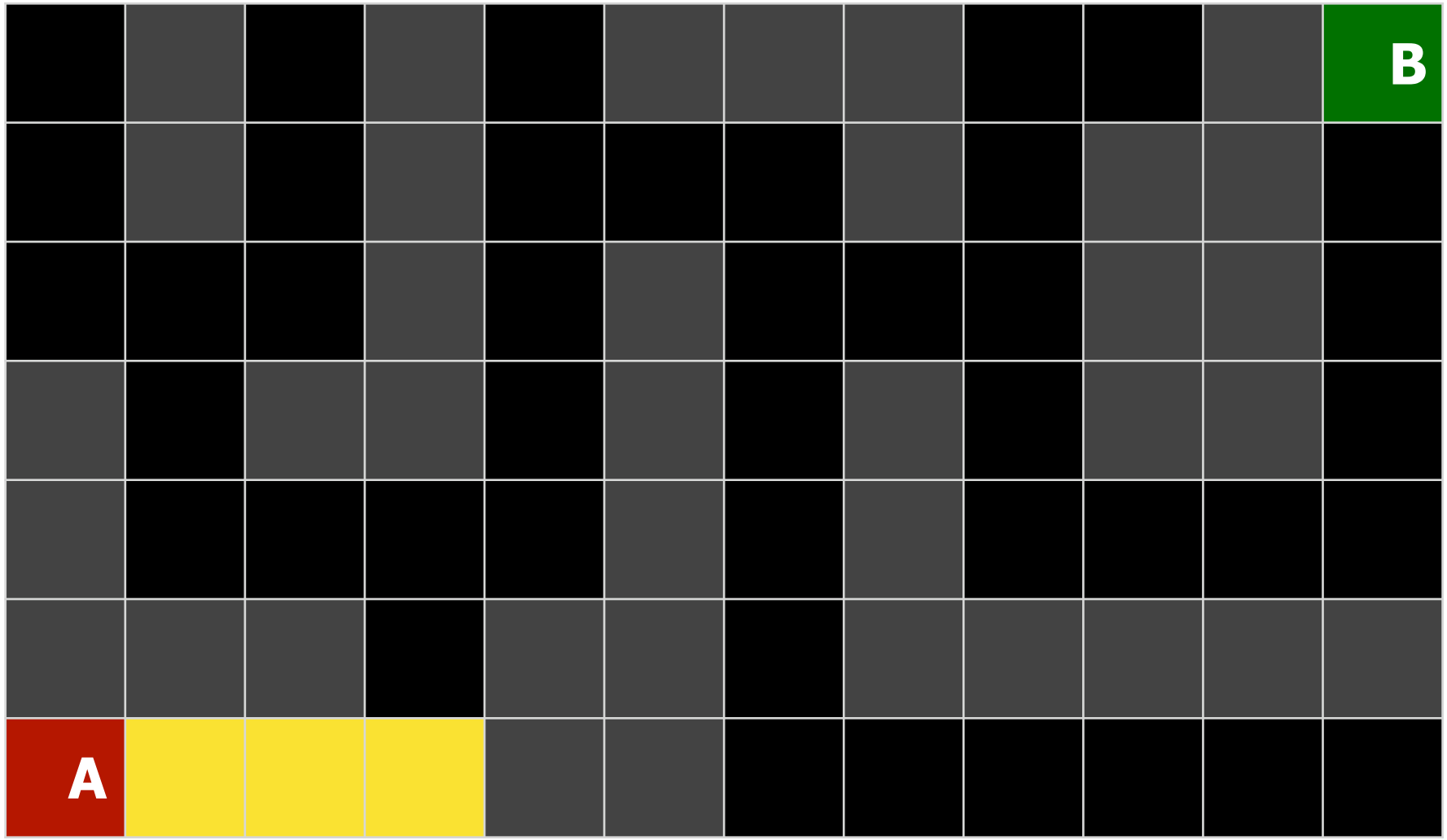
Breadth-First Search



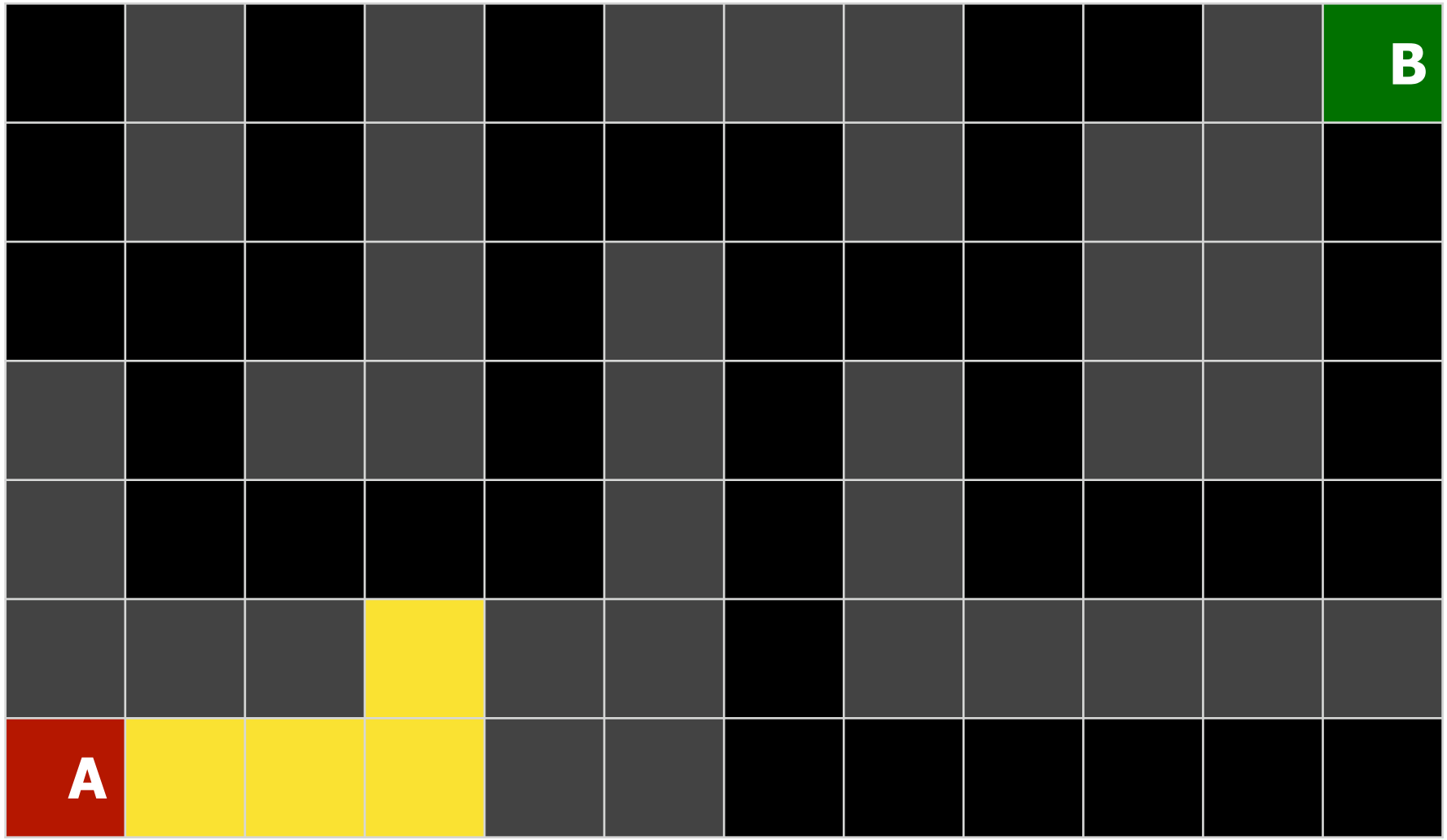
Breadth-First Search



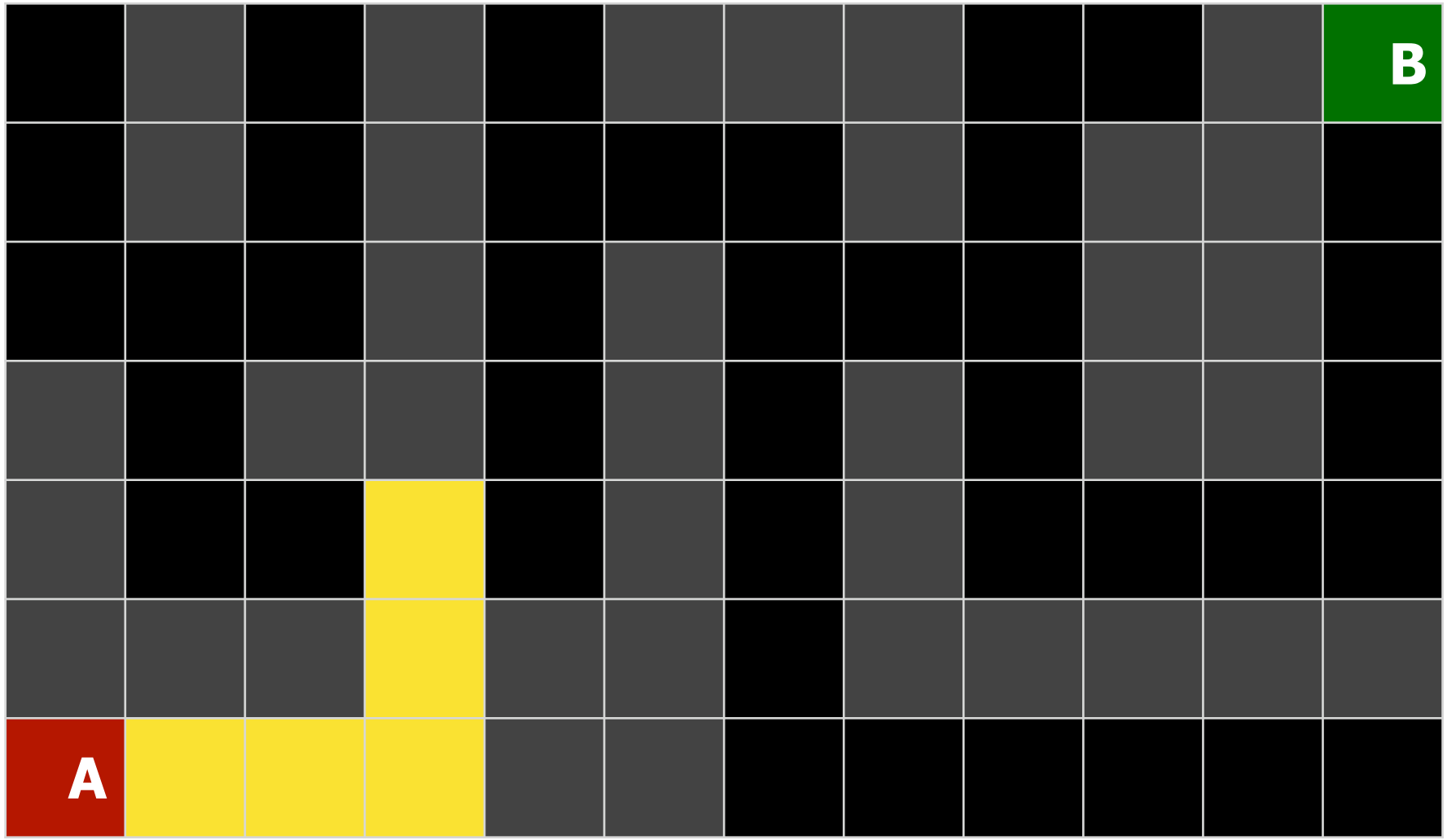
Breadth-First Search



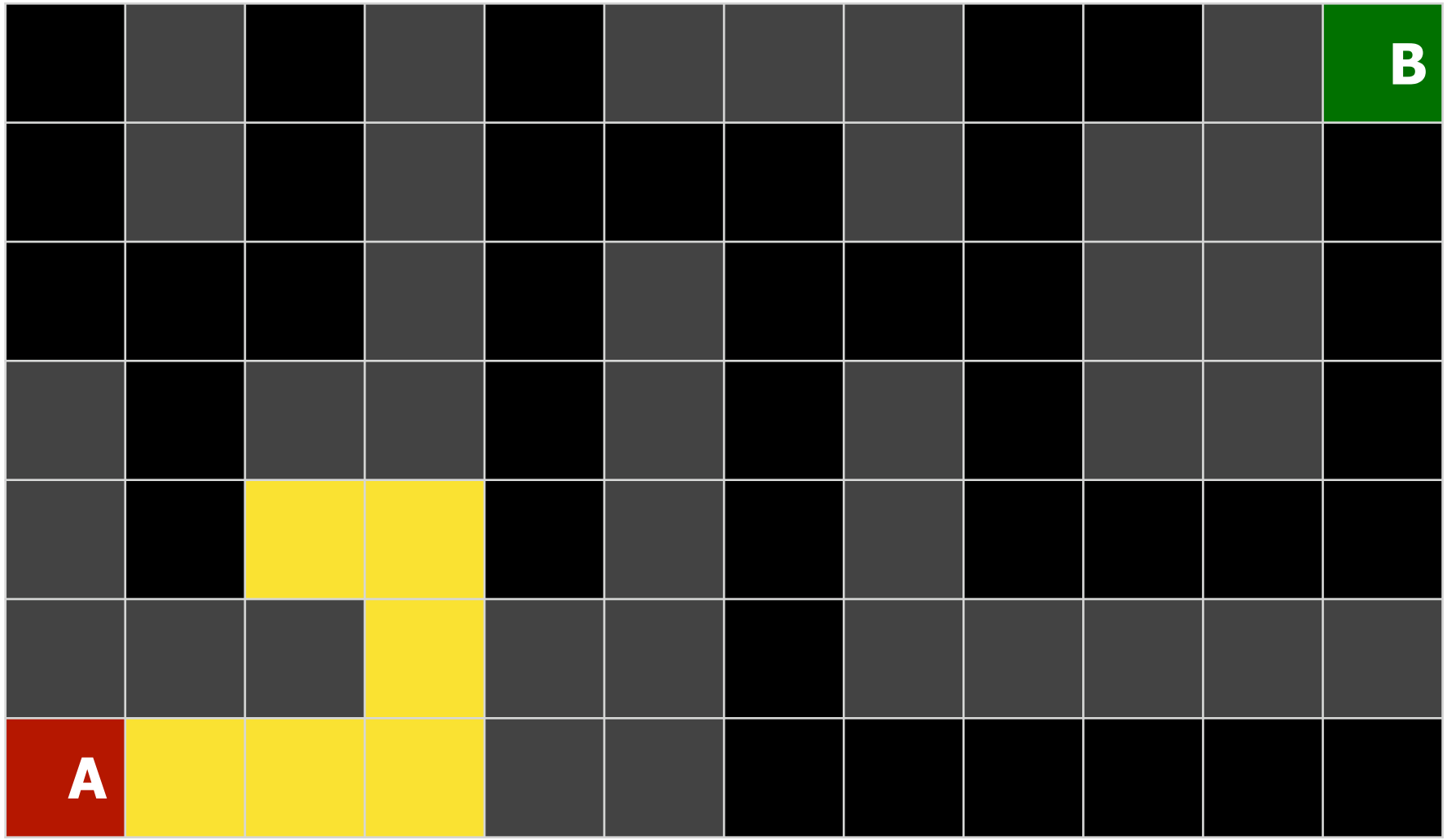
Breadth-First Search



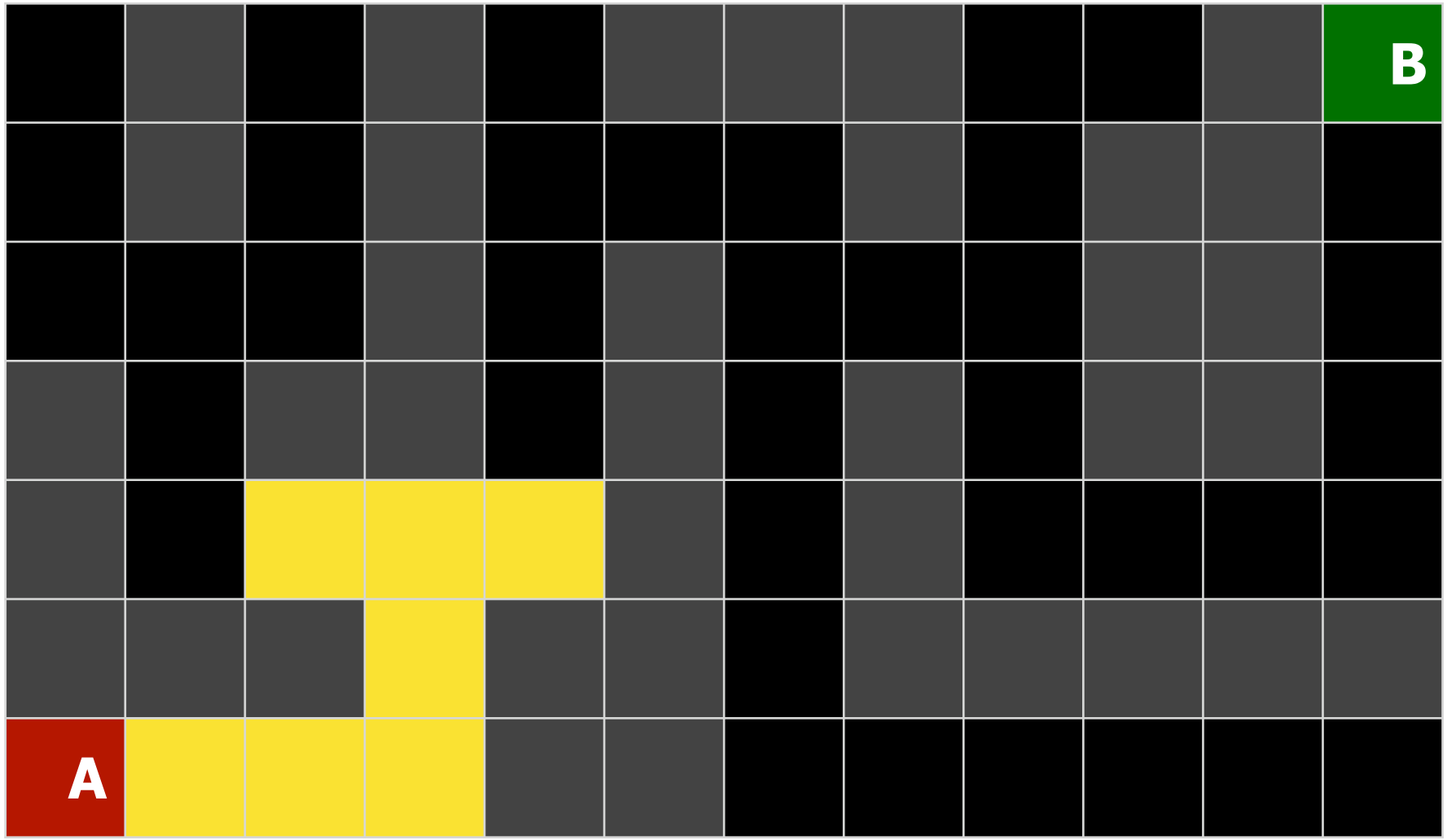
Breadth-First Search



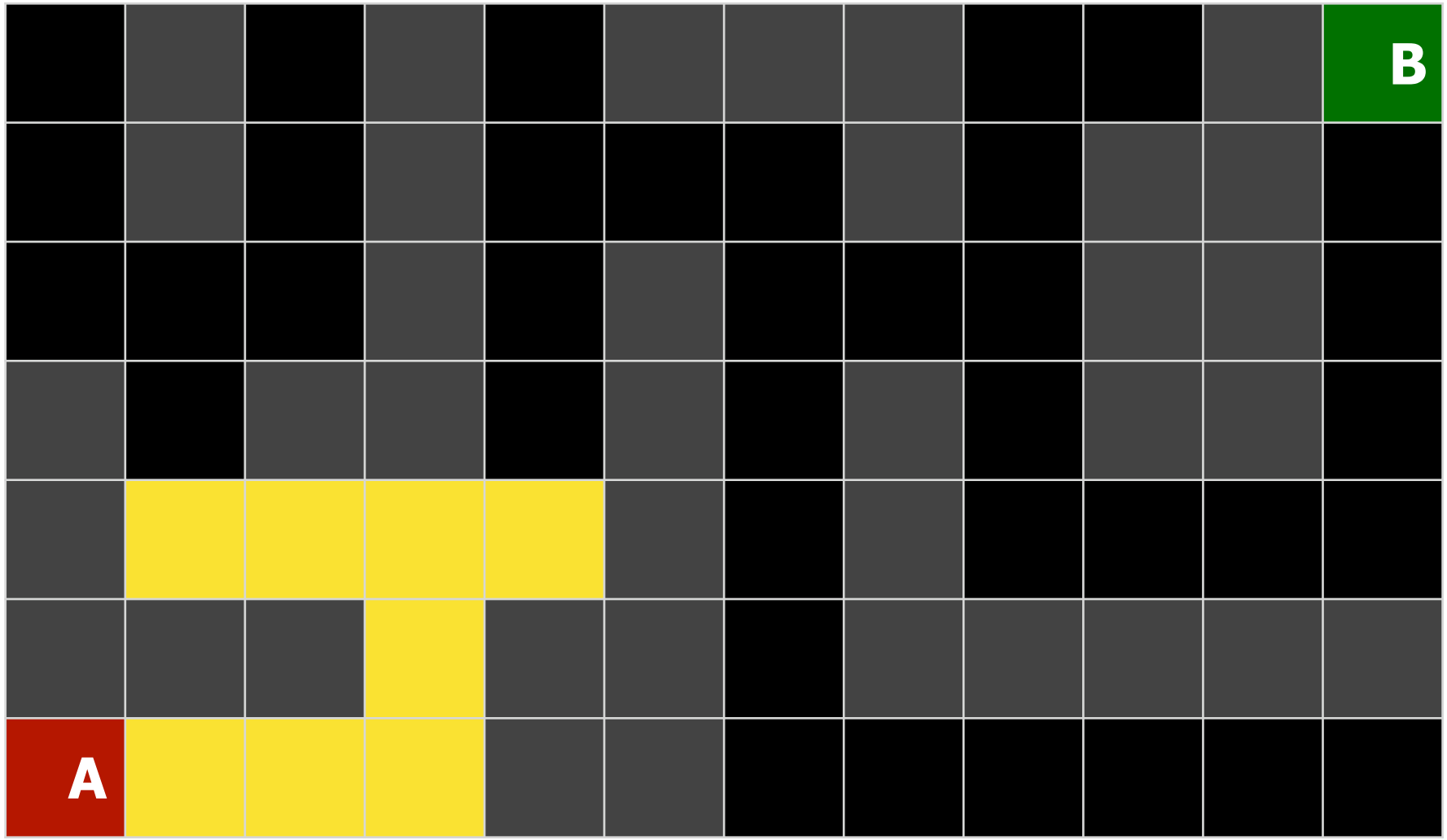
Breadth-First Search



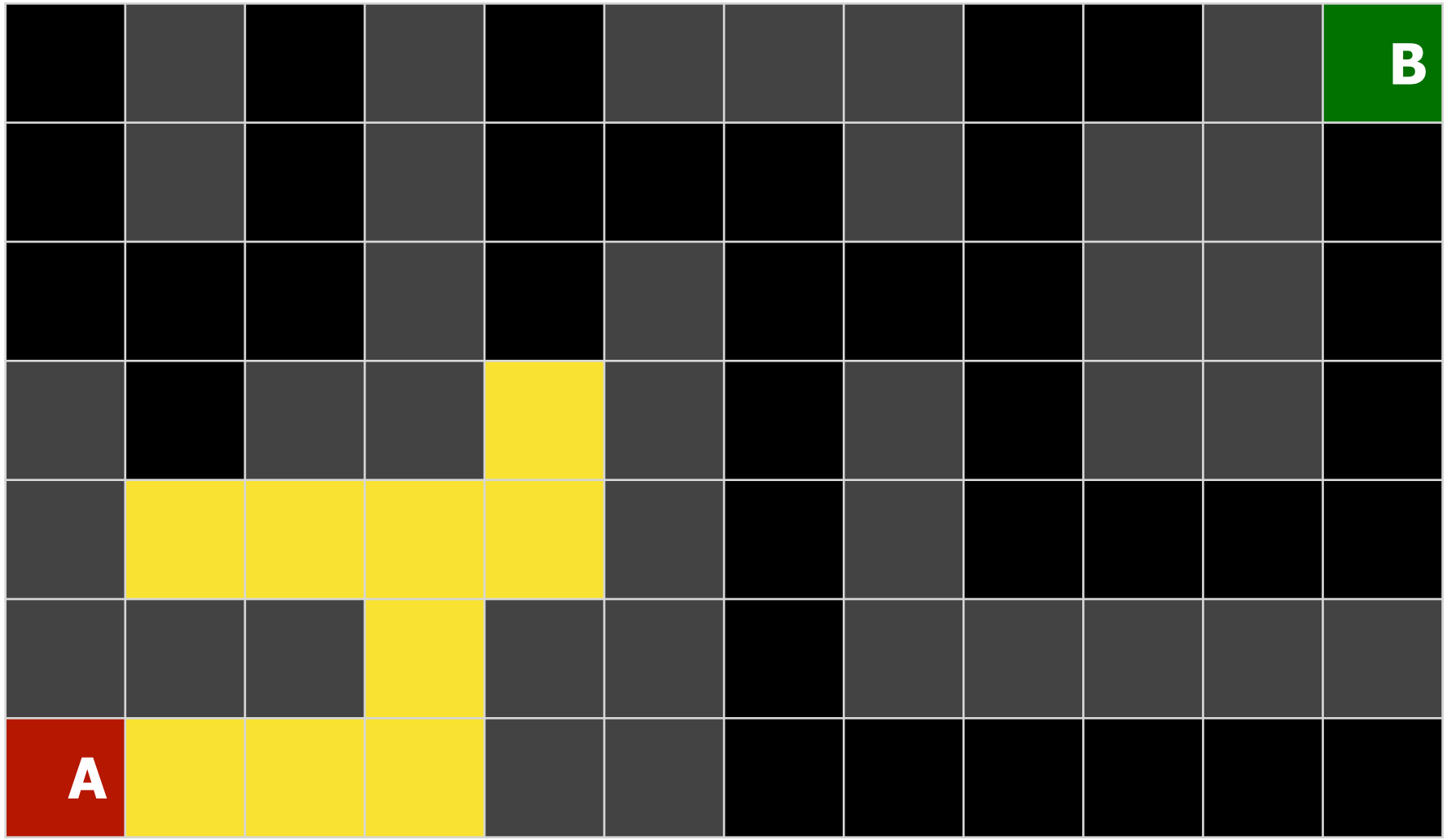
Breadth-First Search



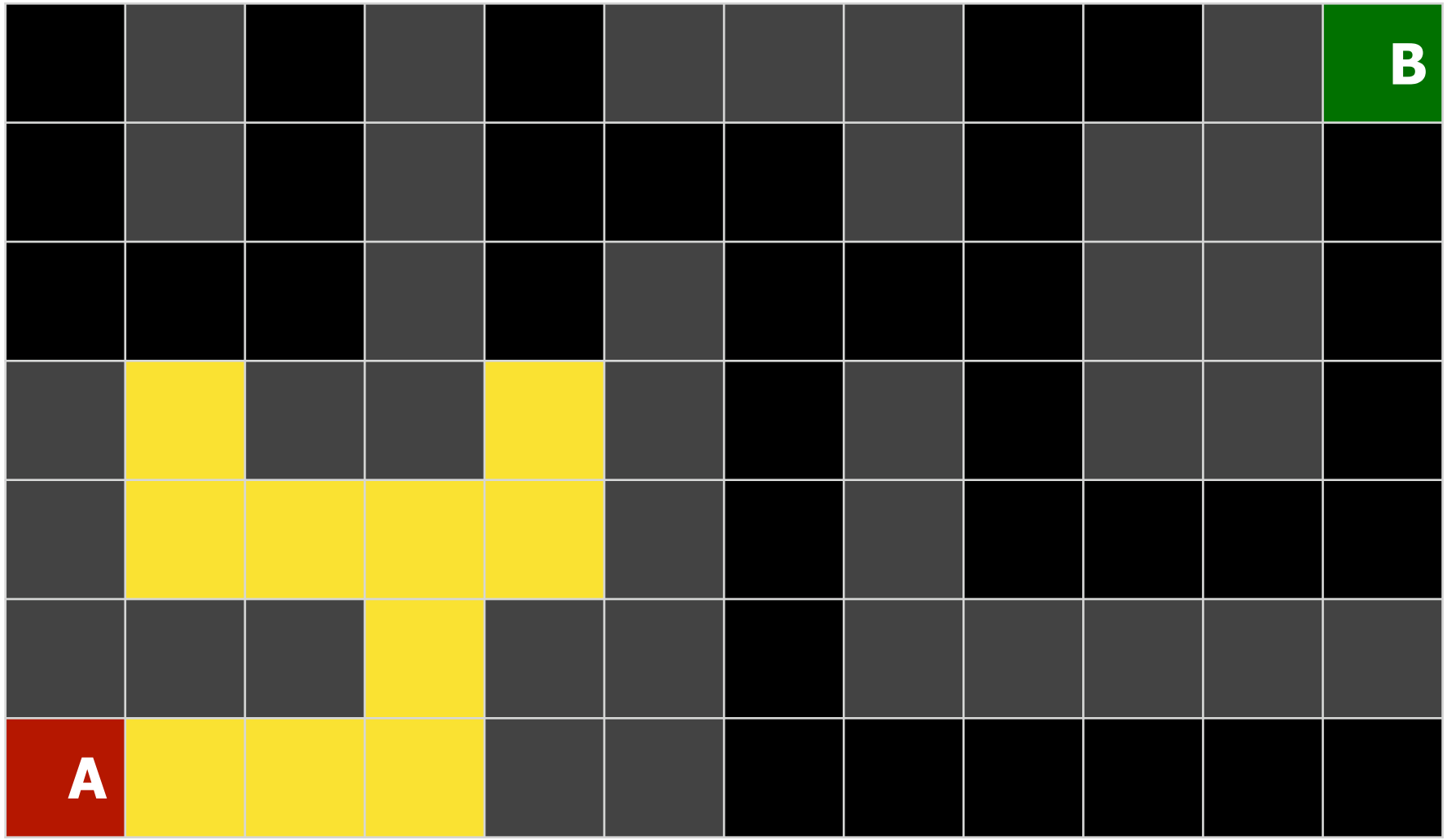
Breadth-First Search



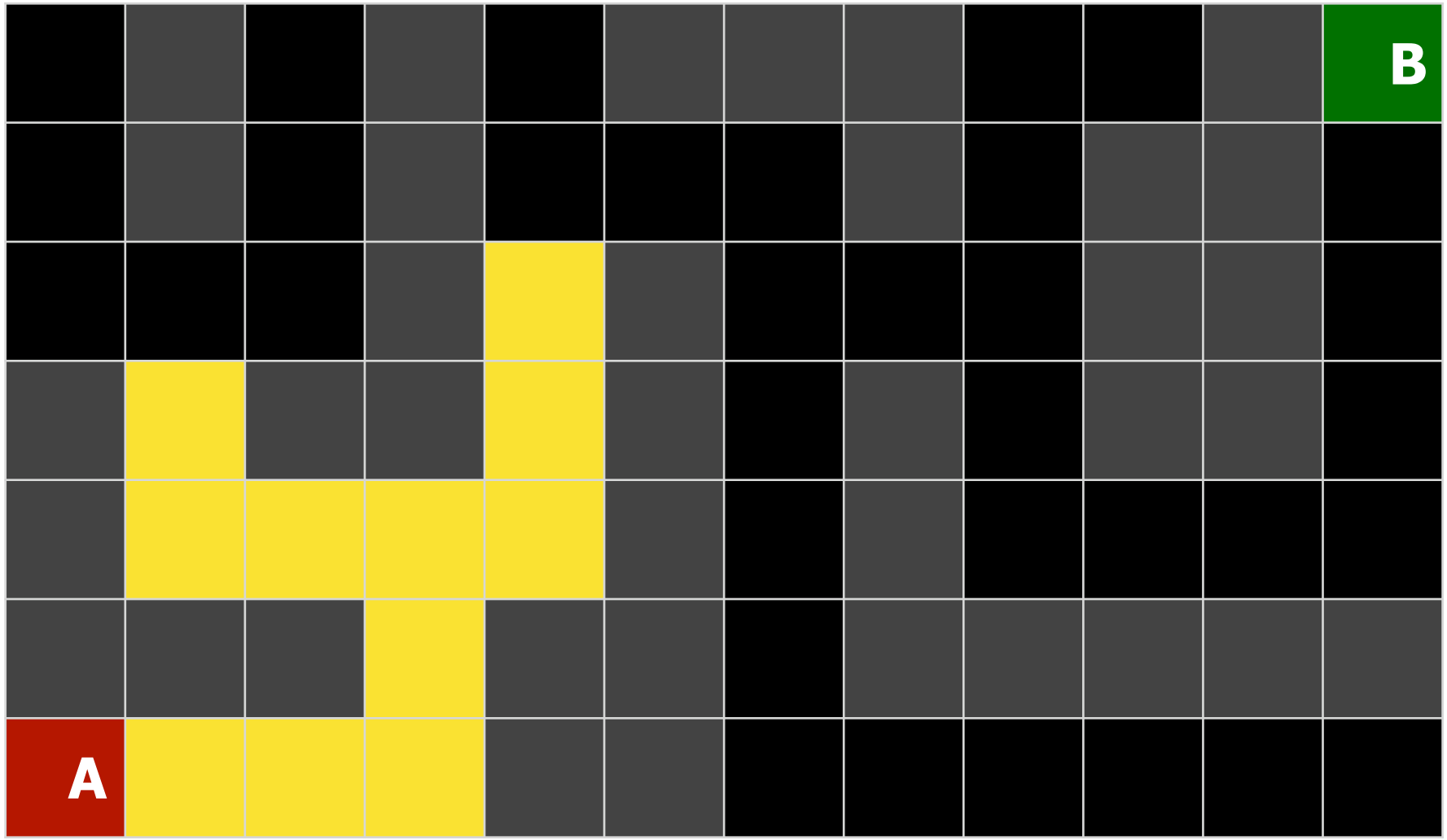
Breadth-First Search



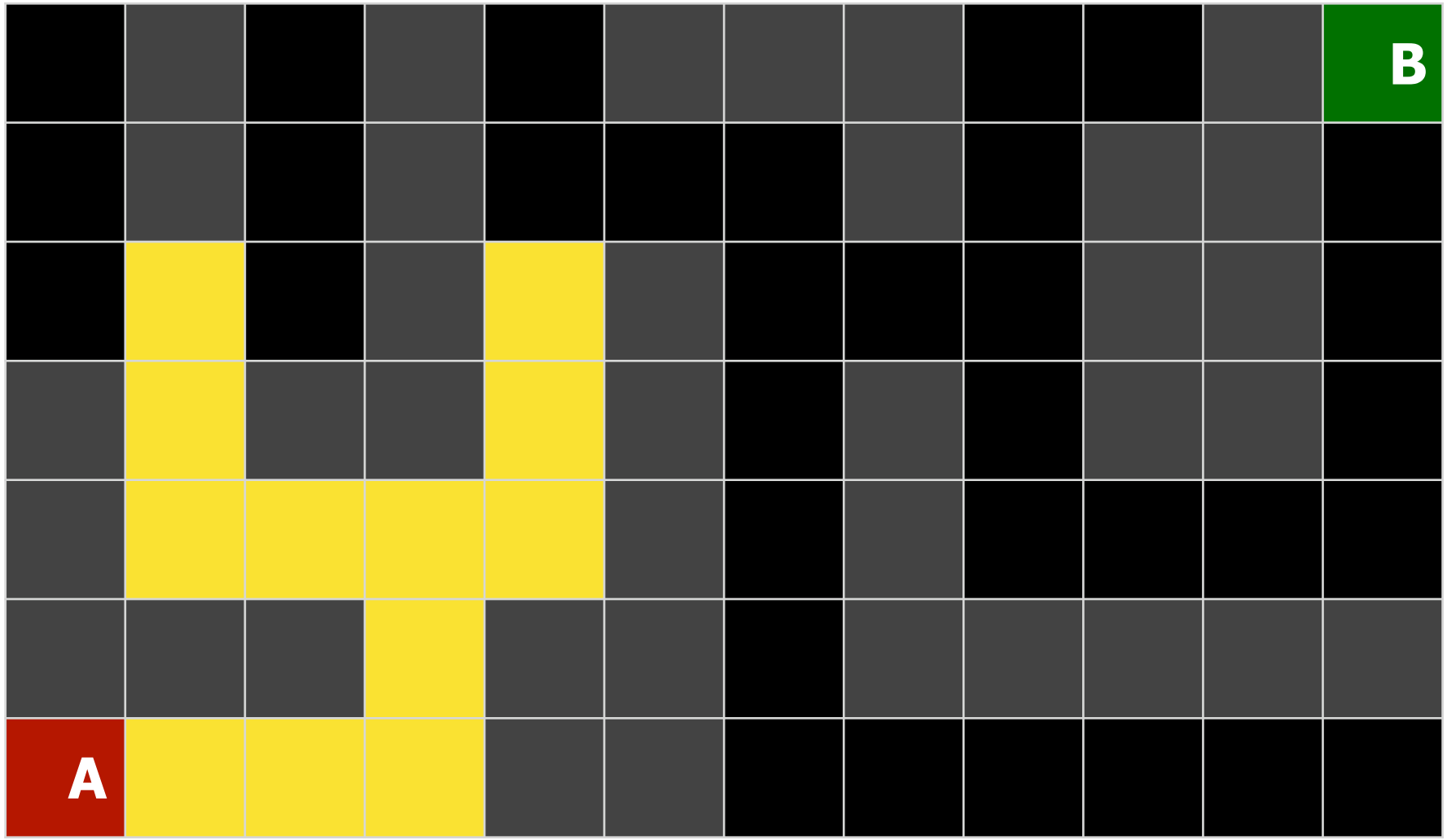
Breadth-First Search



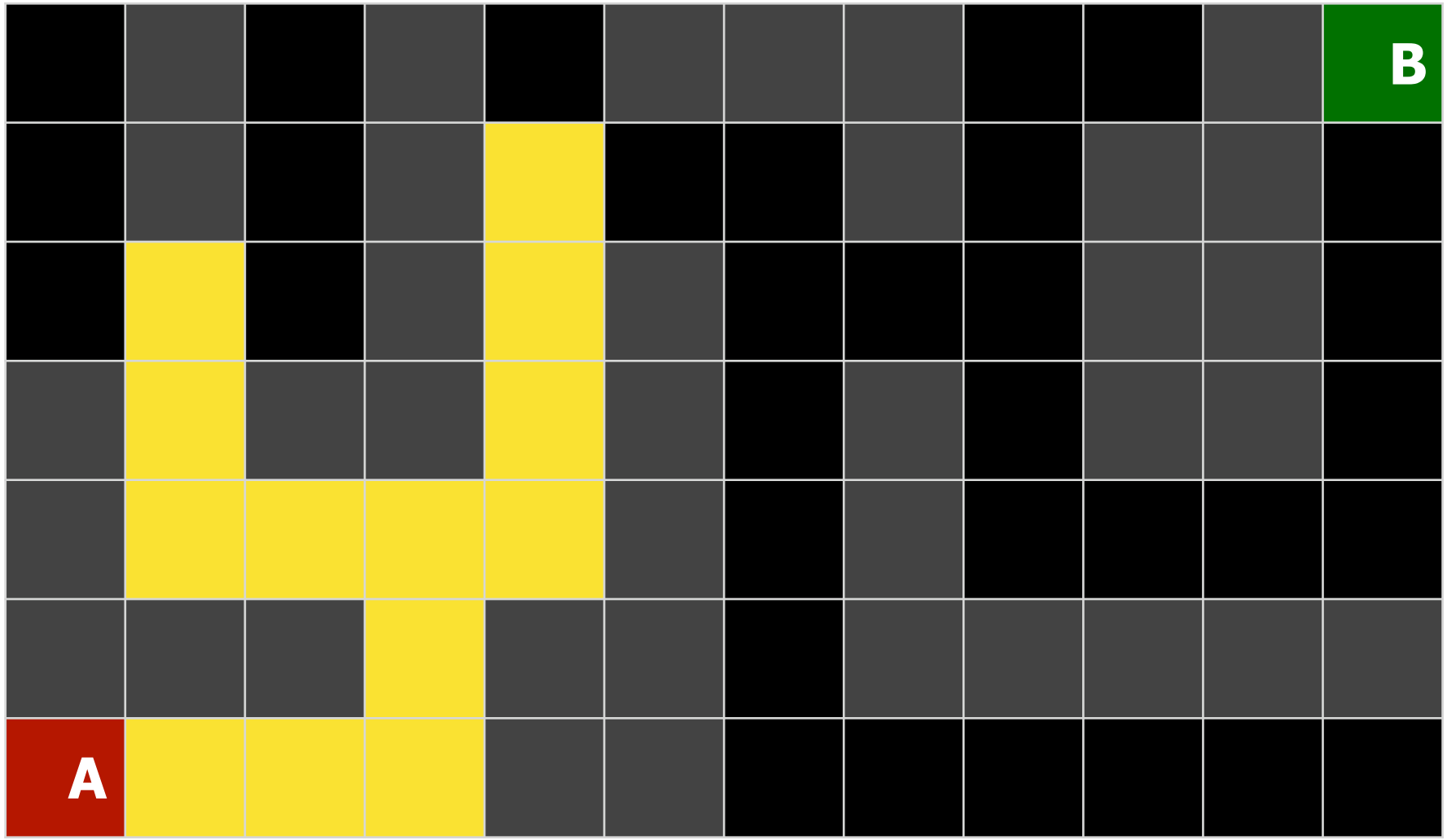
Breadth-First Search



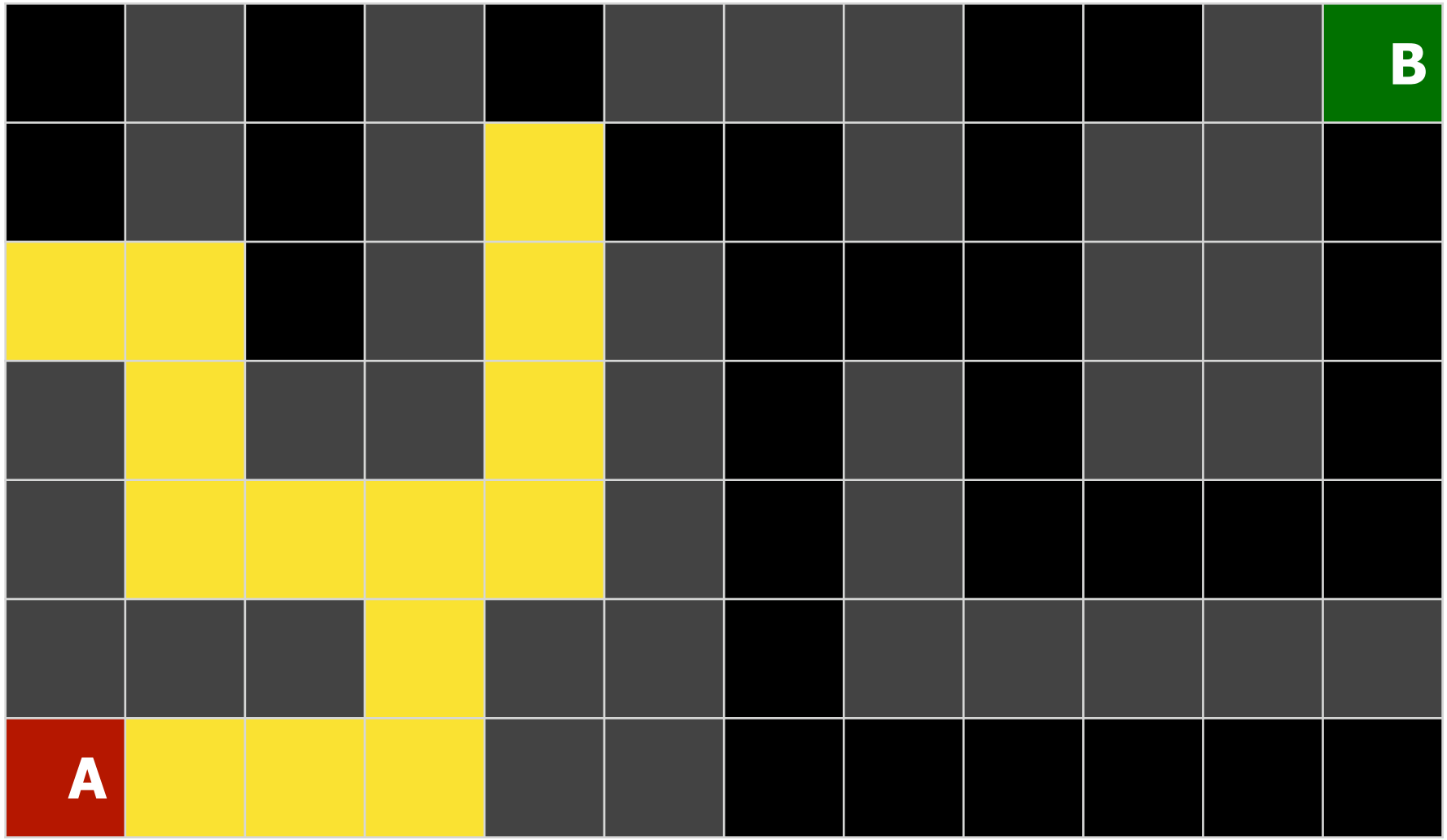
Breadth-First Search



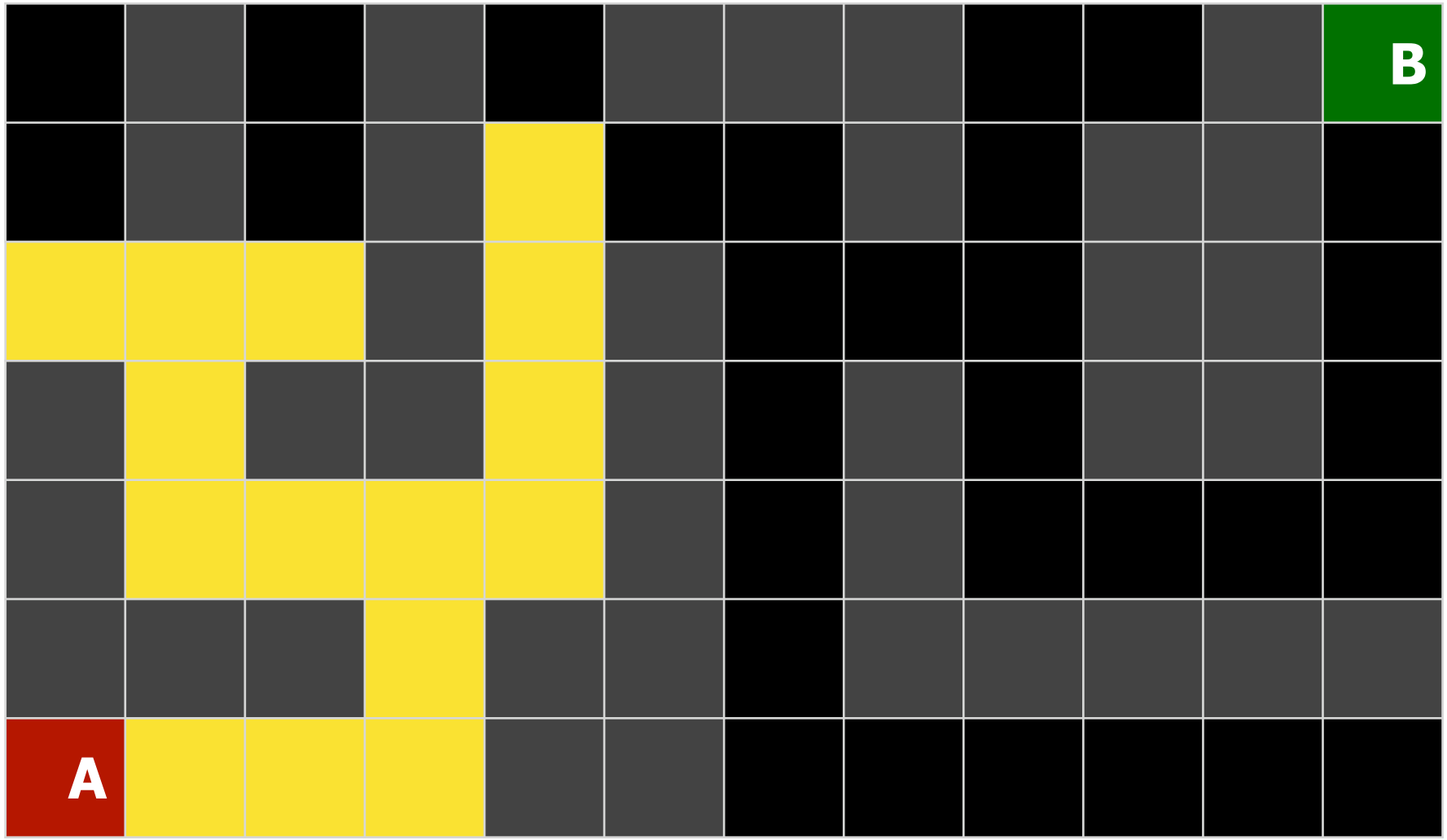
Breadth-First Search



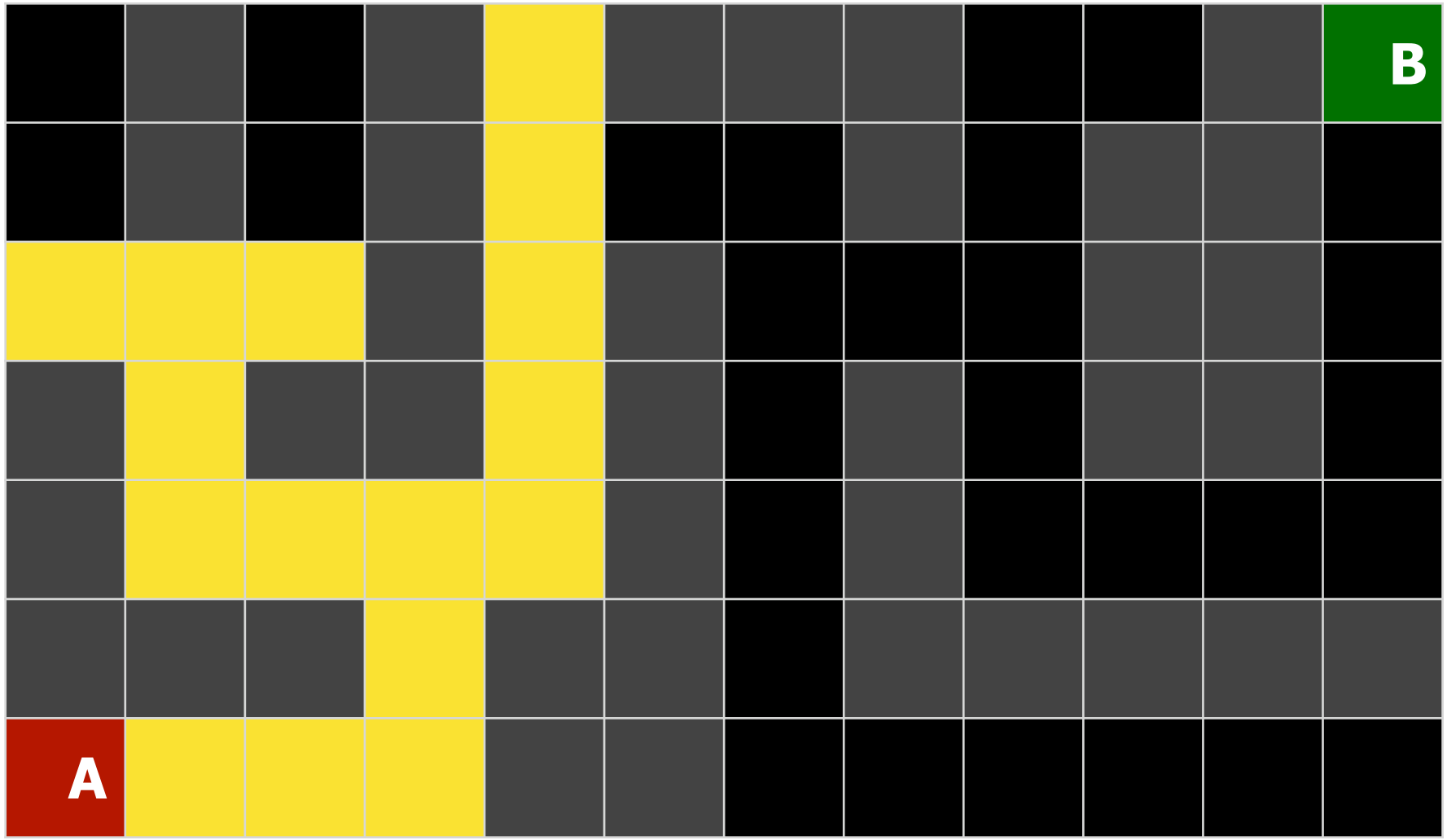
Breadth-First Search



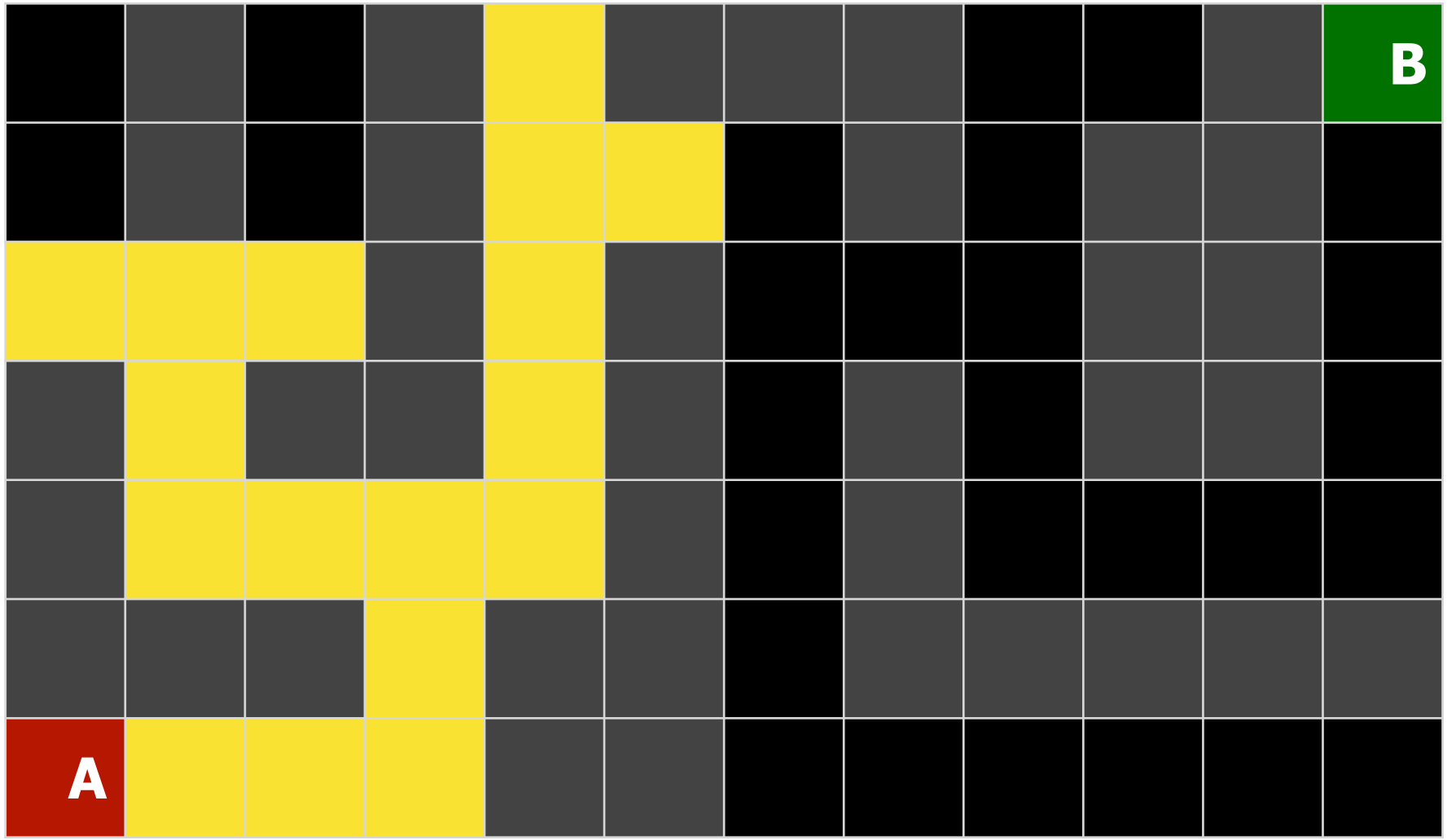
Breadth-First Search



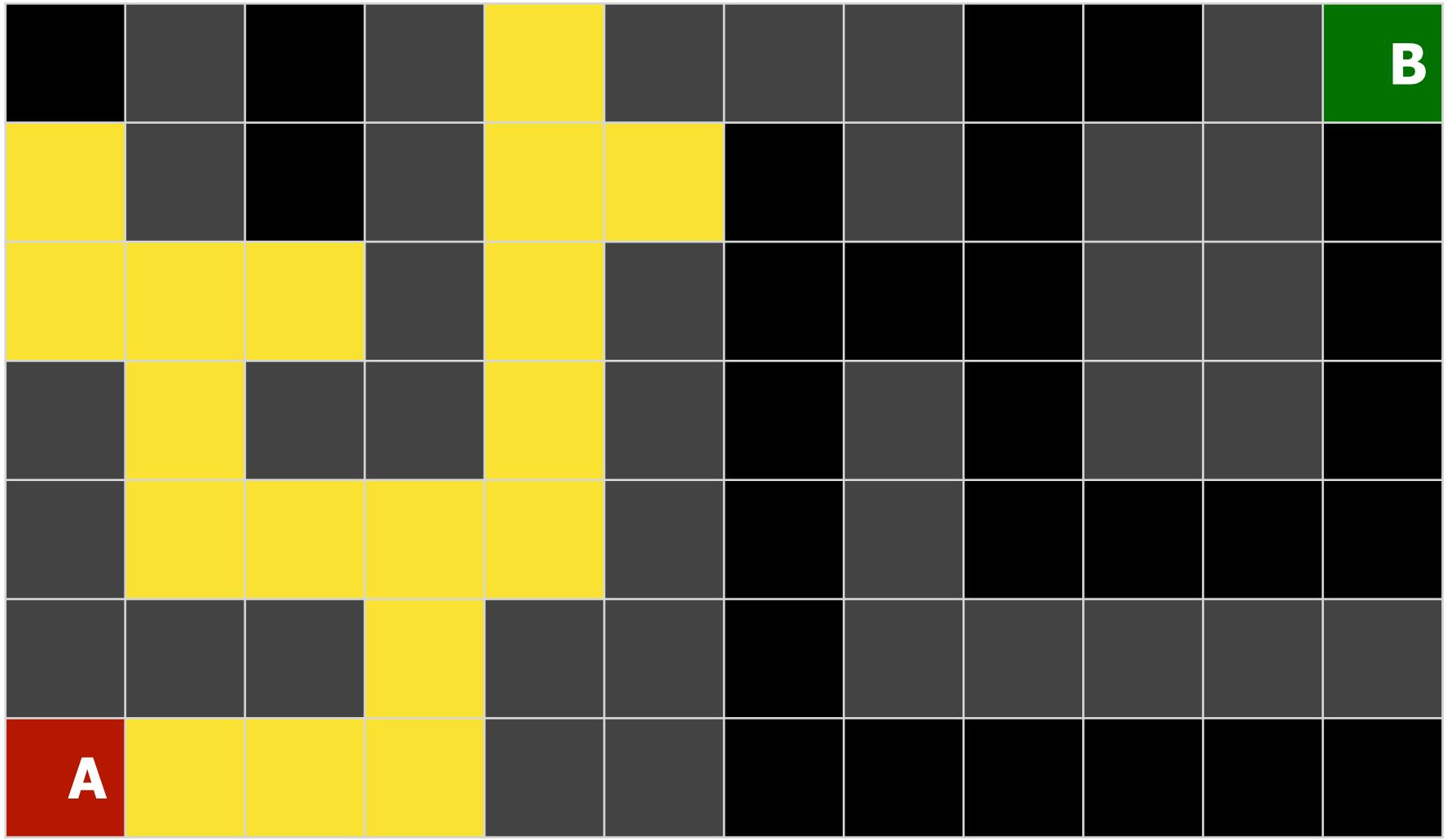
Breadth-First Search



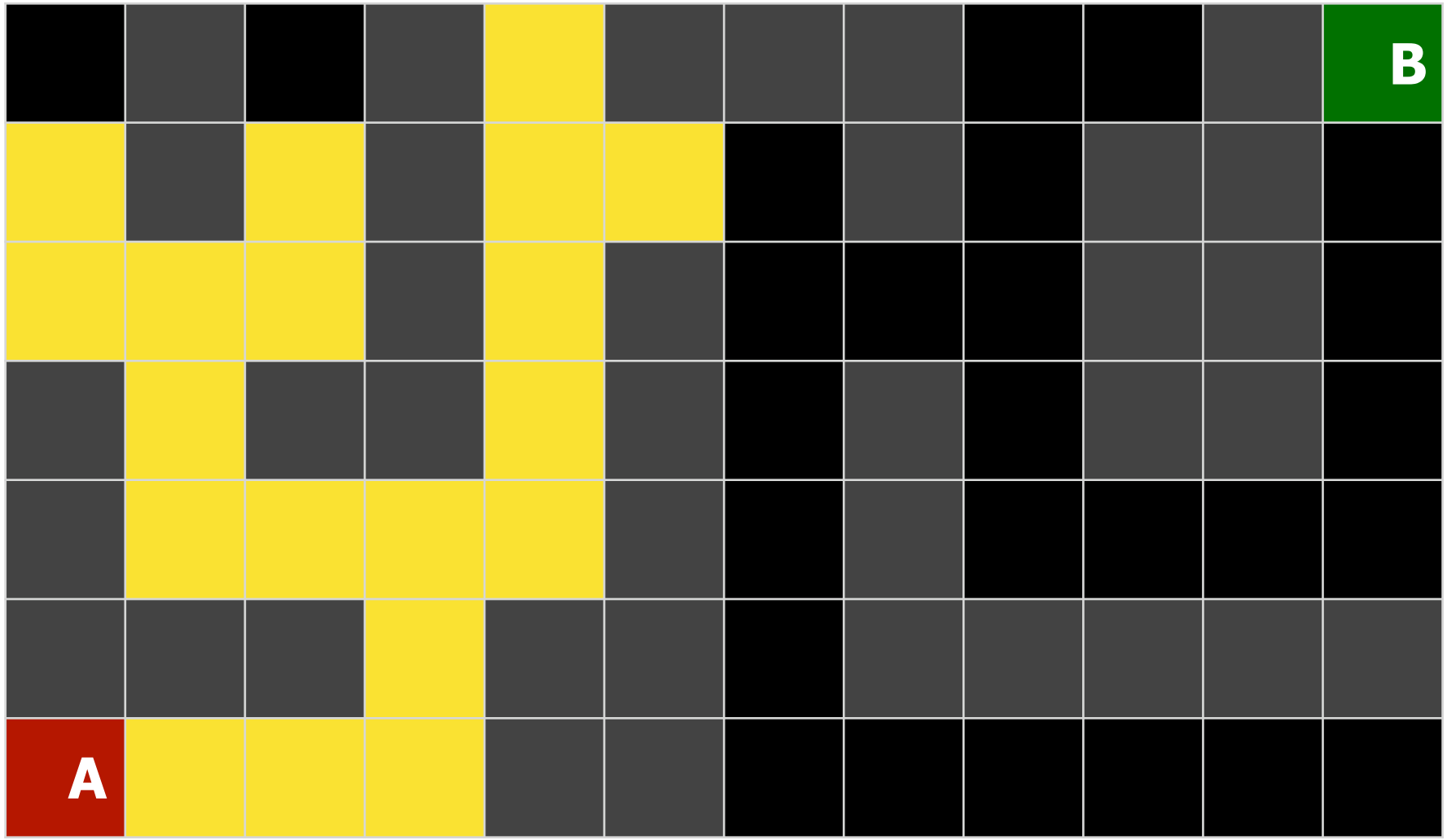
Breadth-First Search



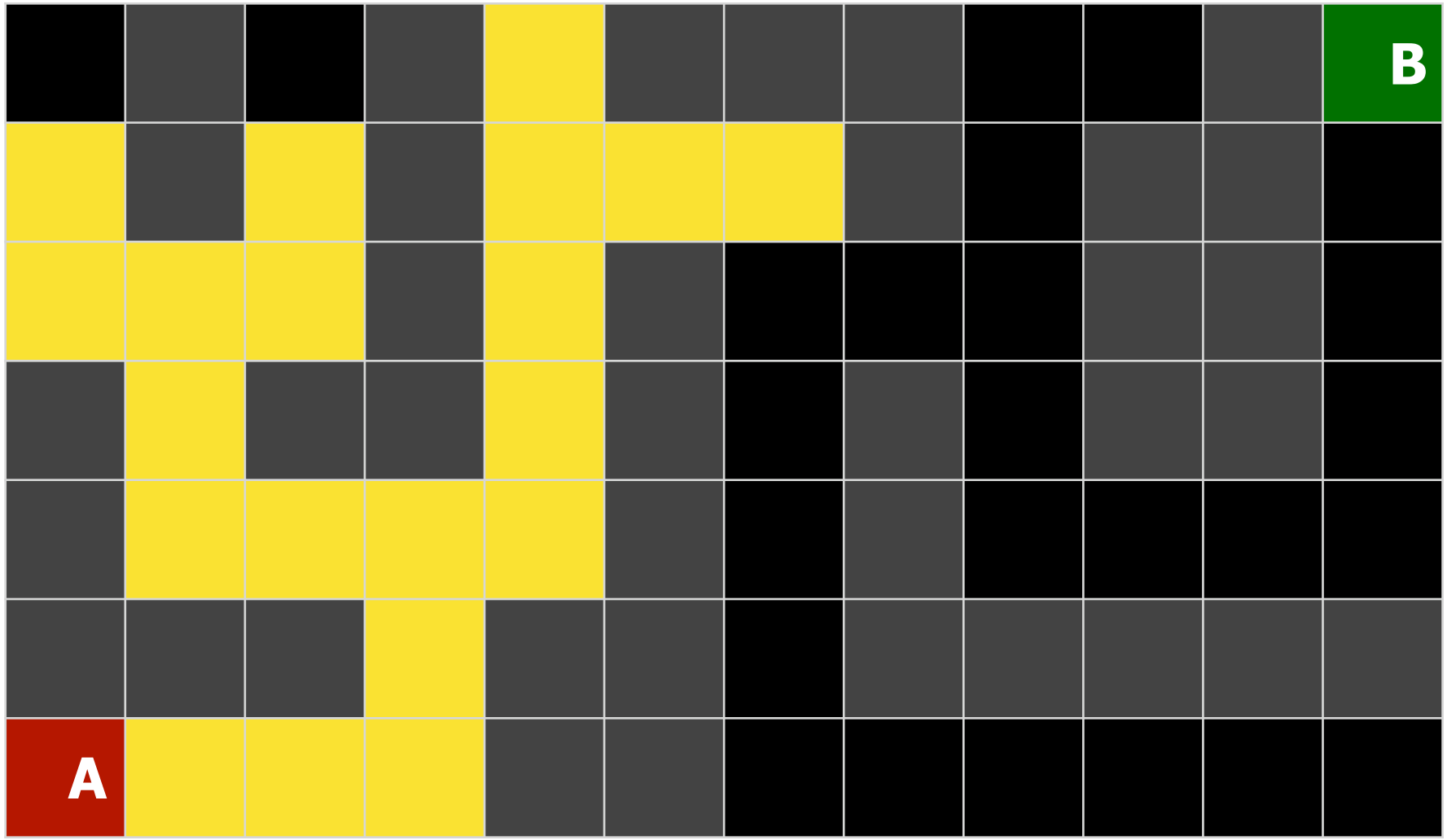
Breadth-First Search



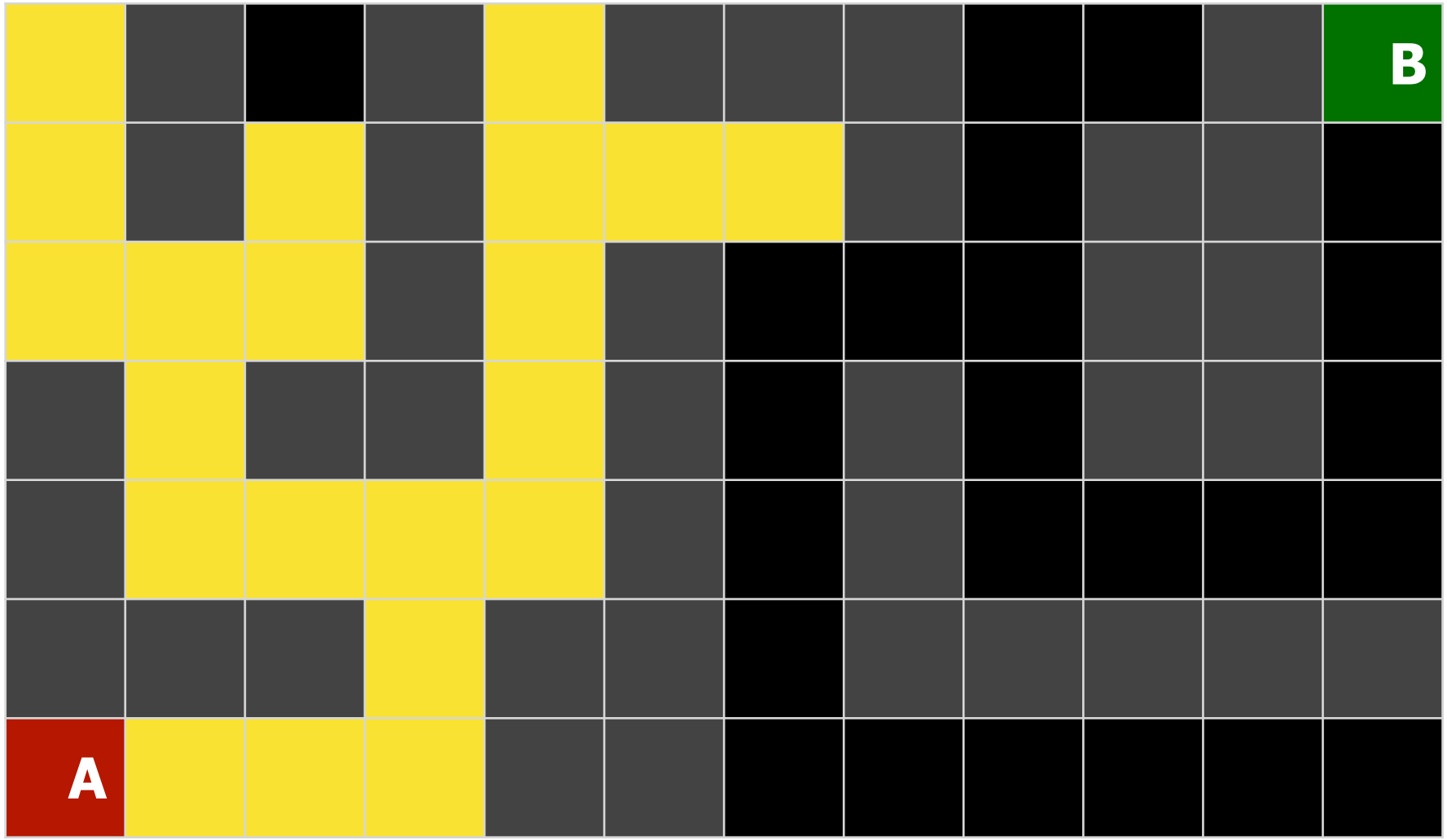
Breadth-First Search



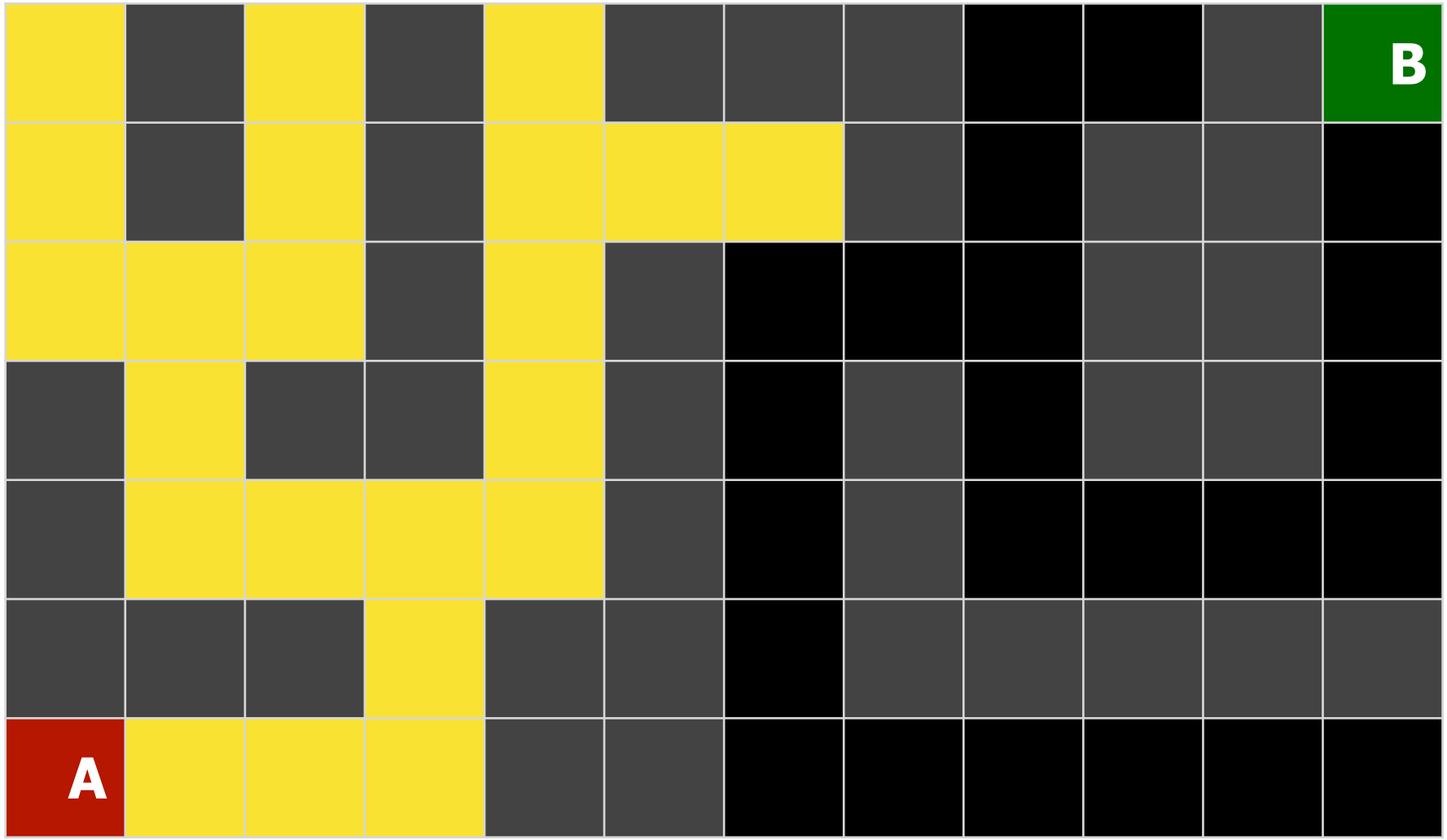
Breadth-First Search



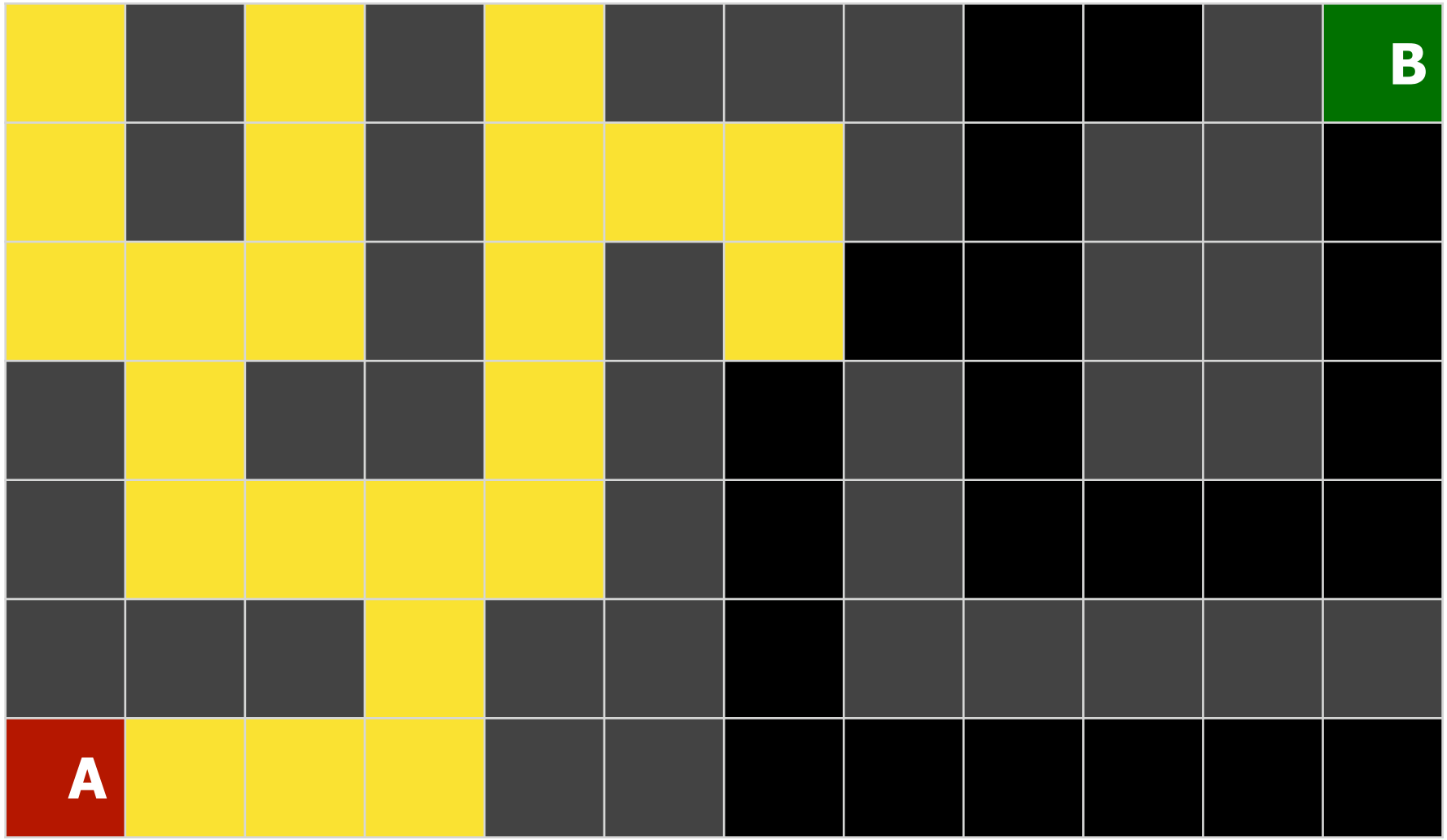
Breadth-First Search



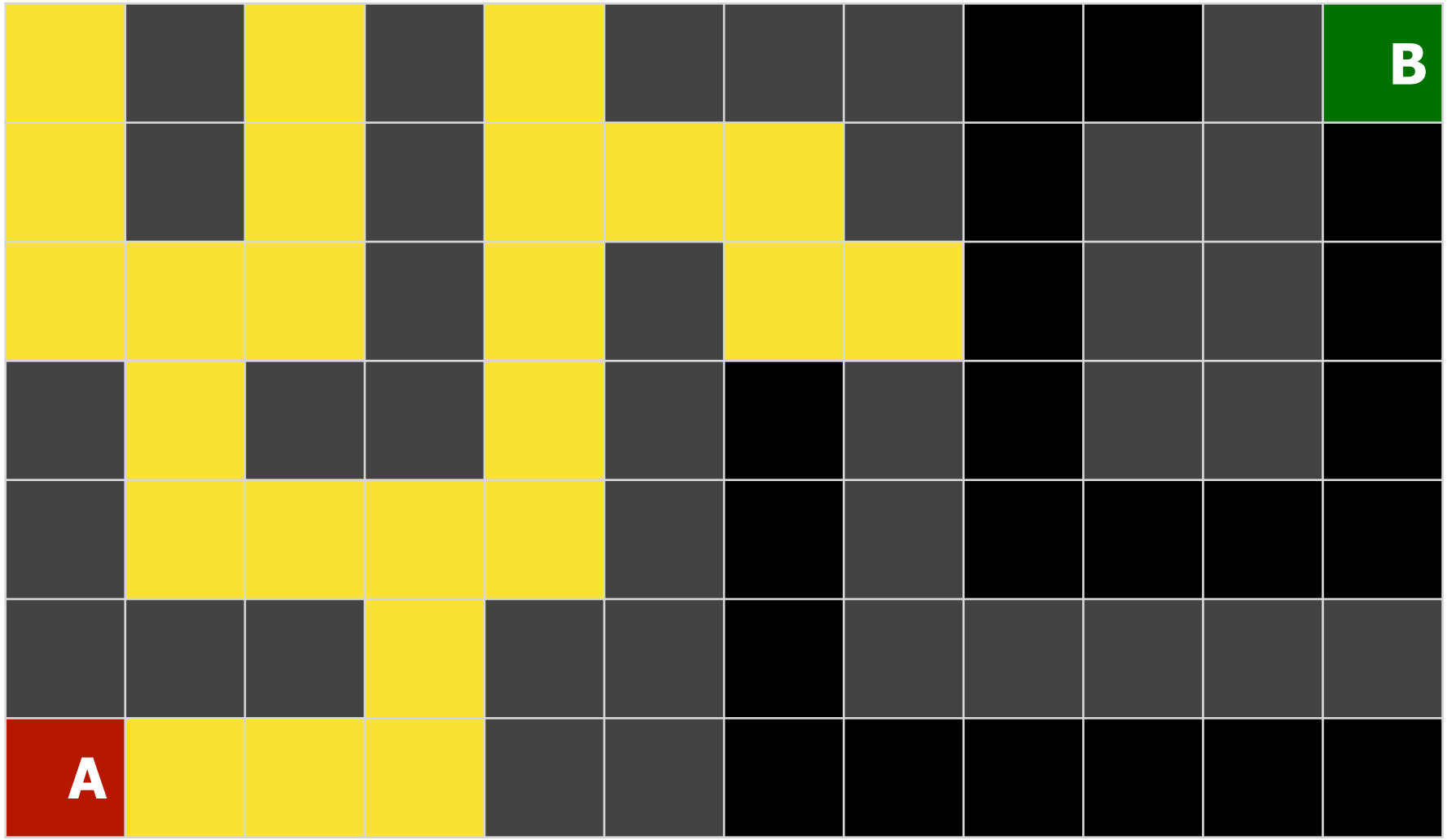
Breadth-First Search



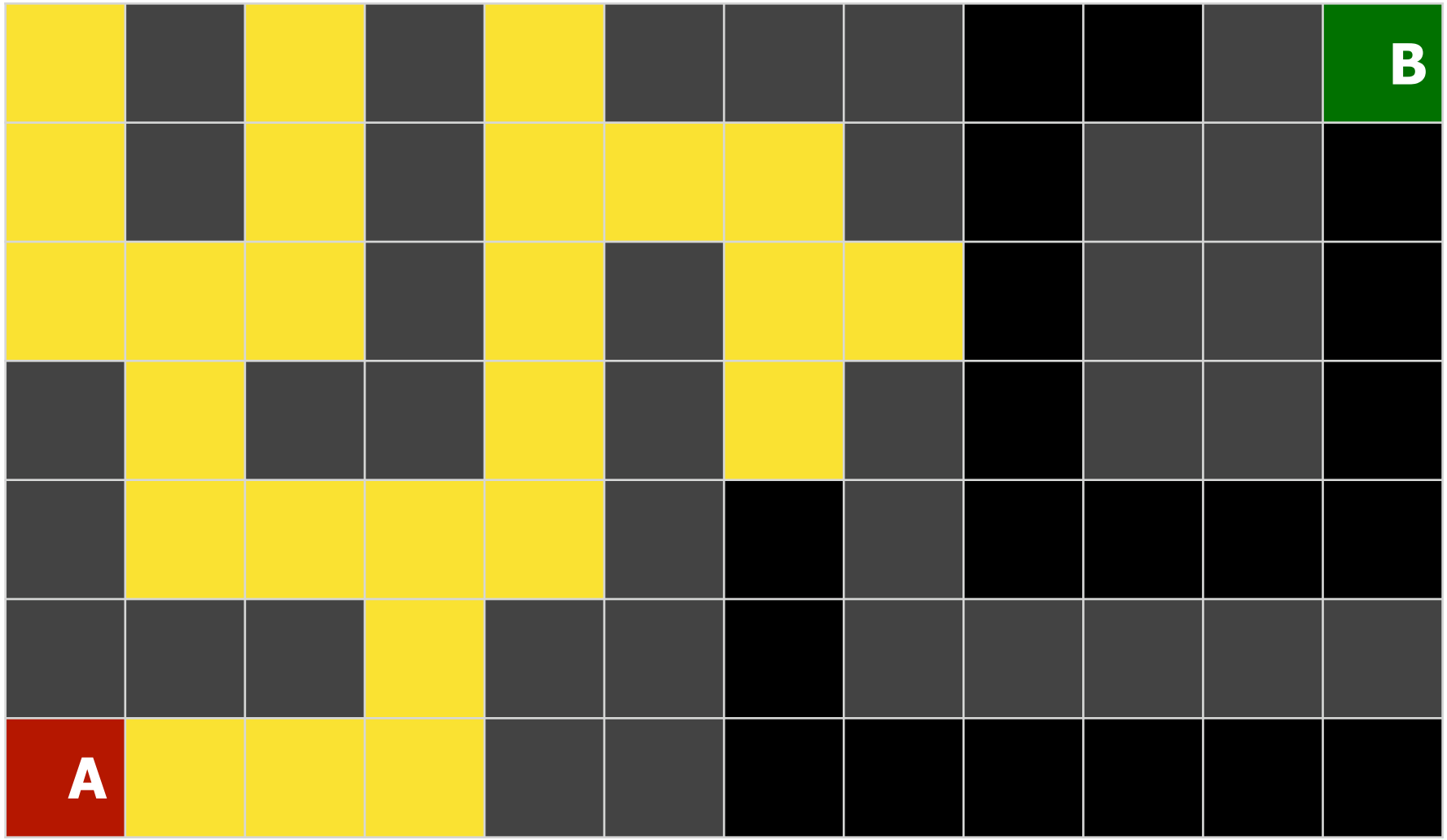
Breadth-First Search



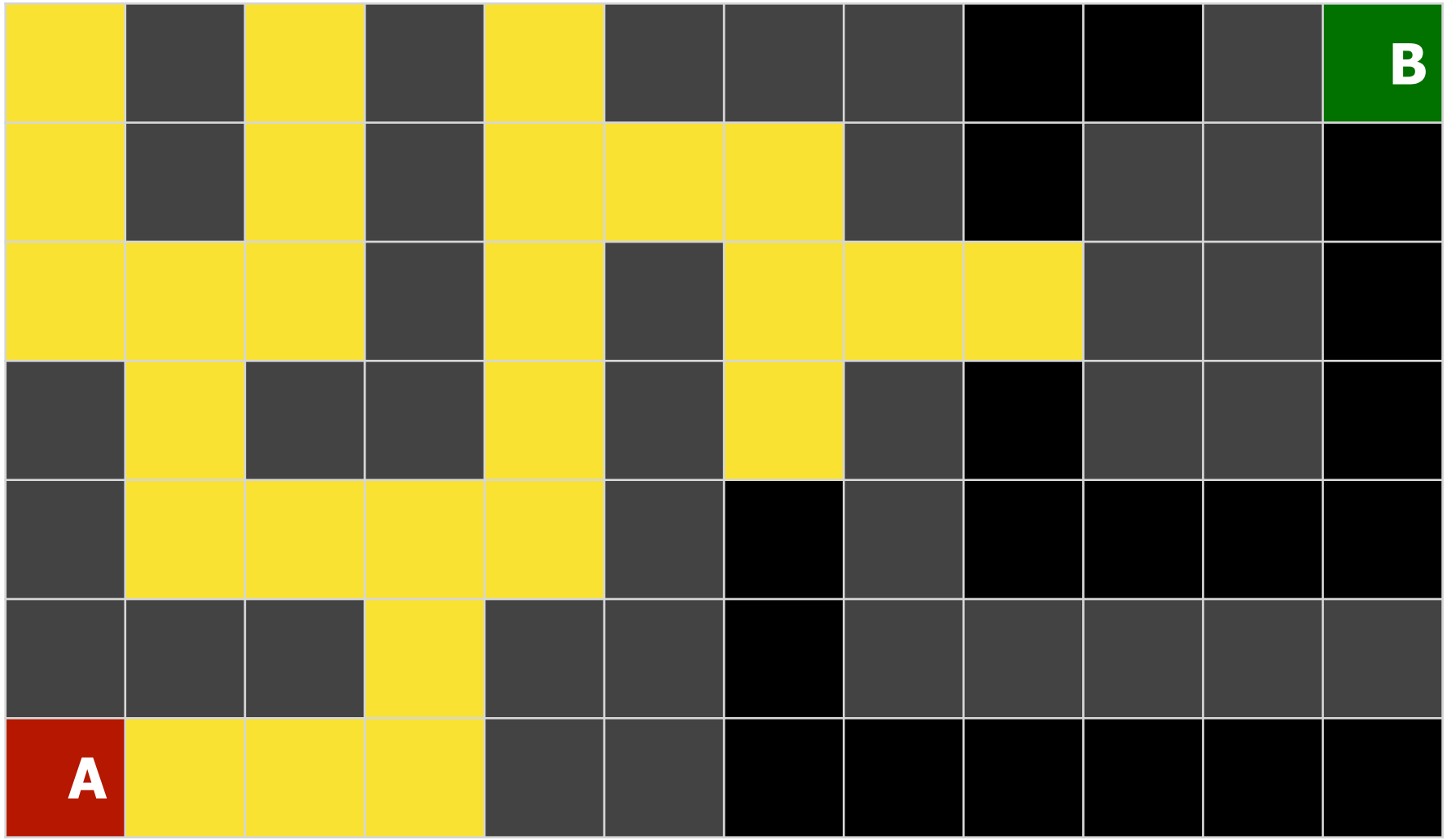
Breadth-First Search



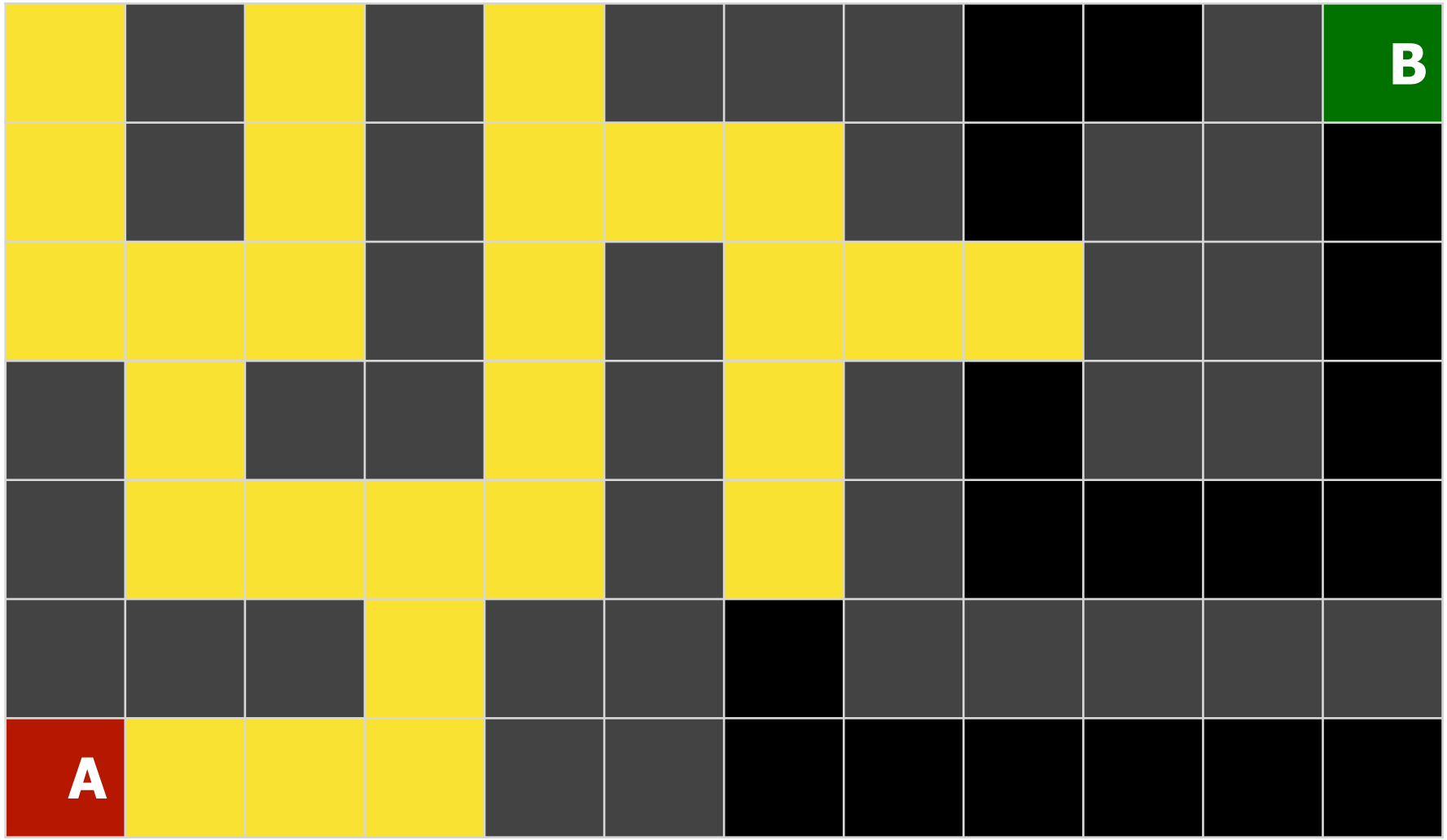
Breadth-First Search



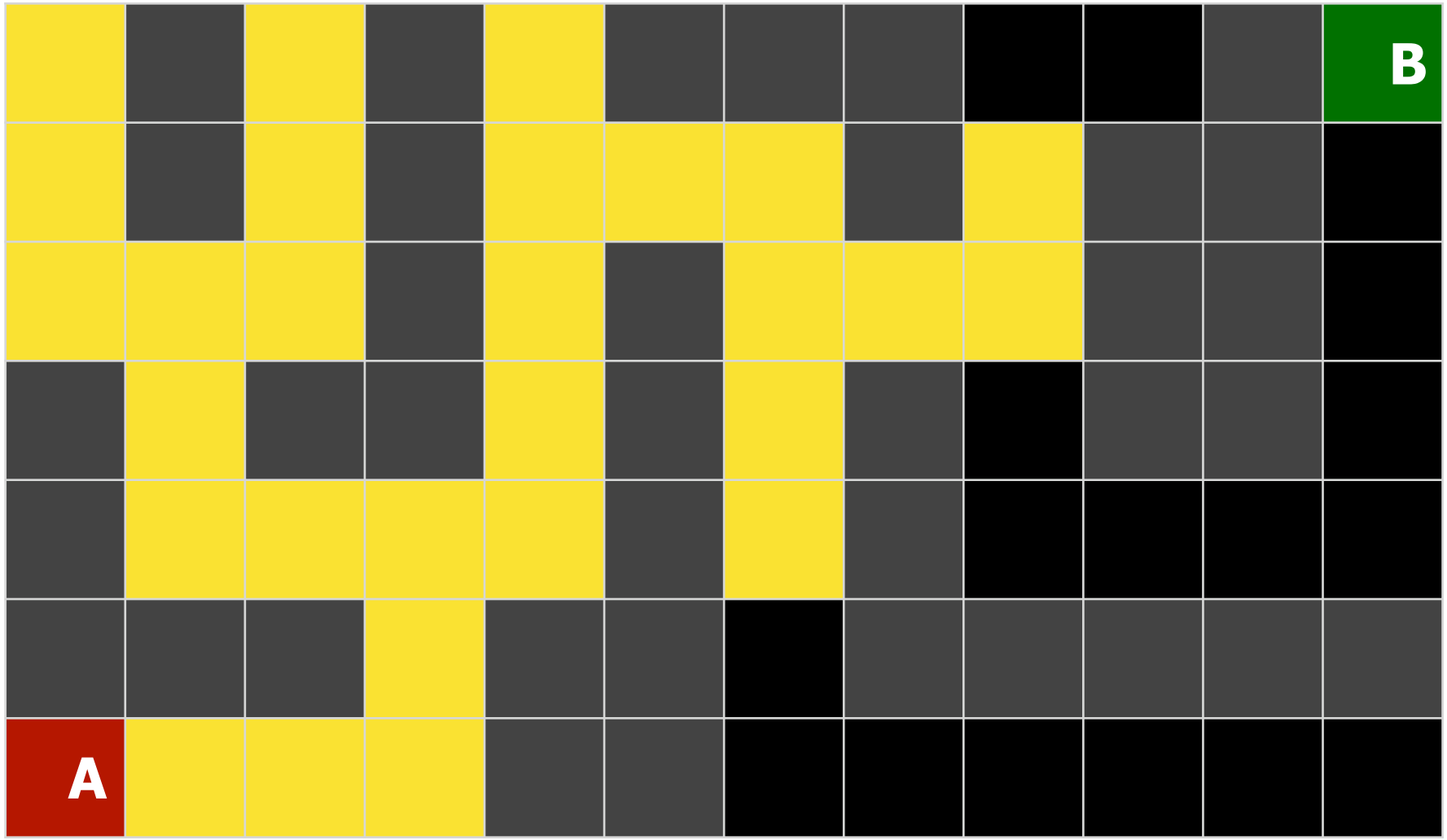
Breadth-First Search



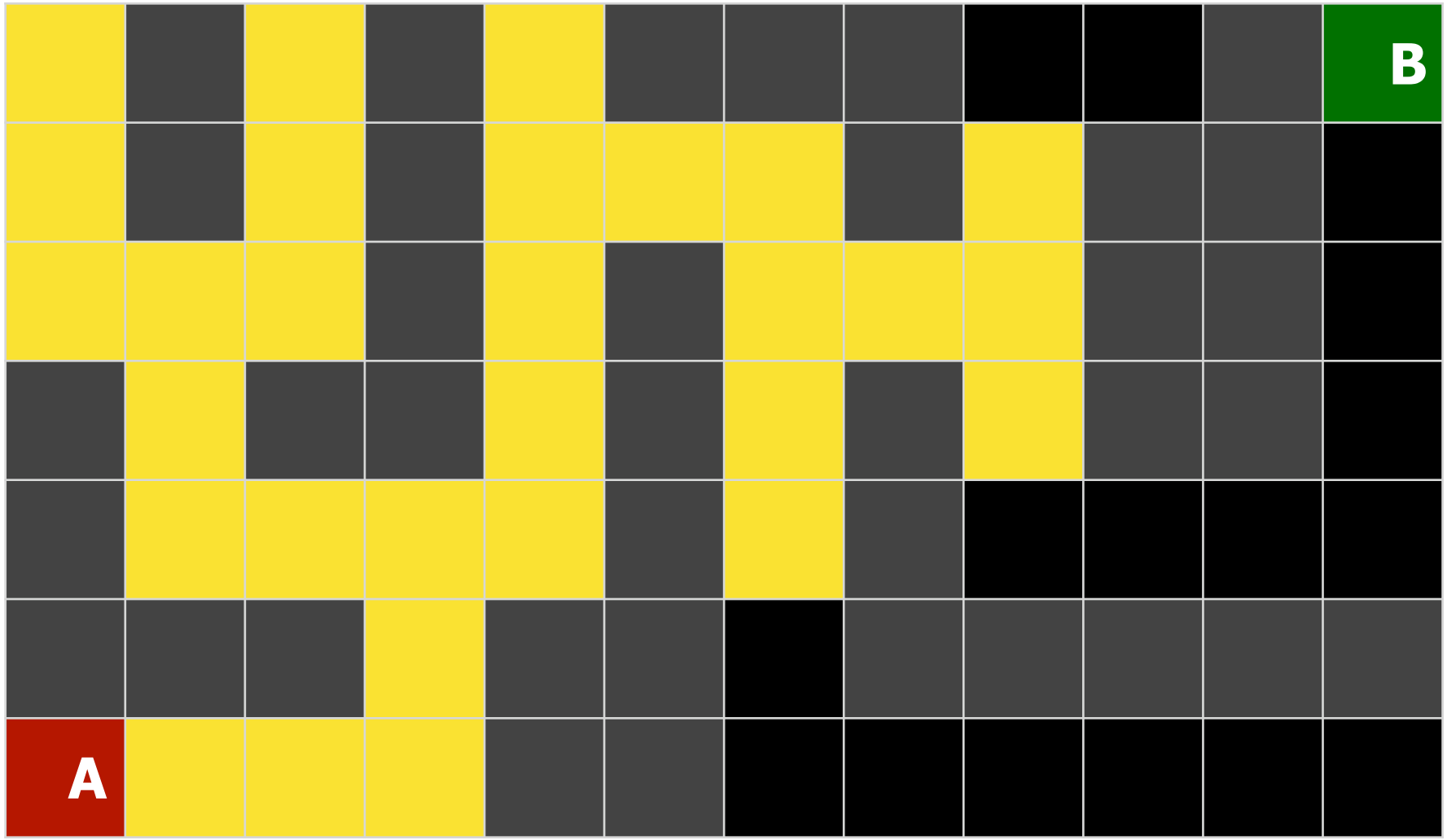
Breadth-First Search



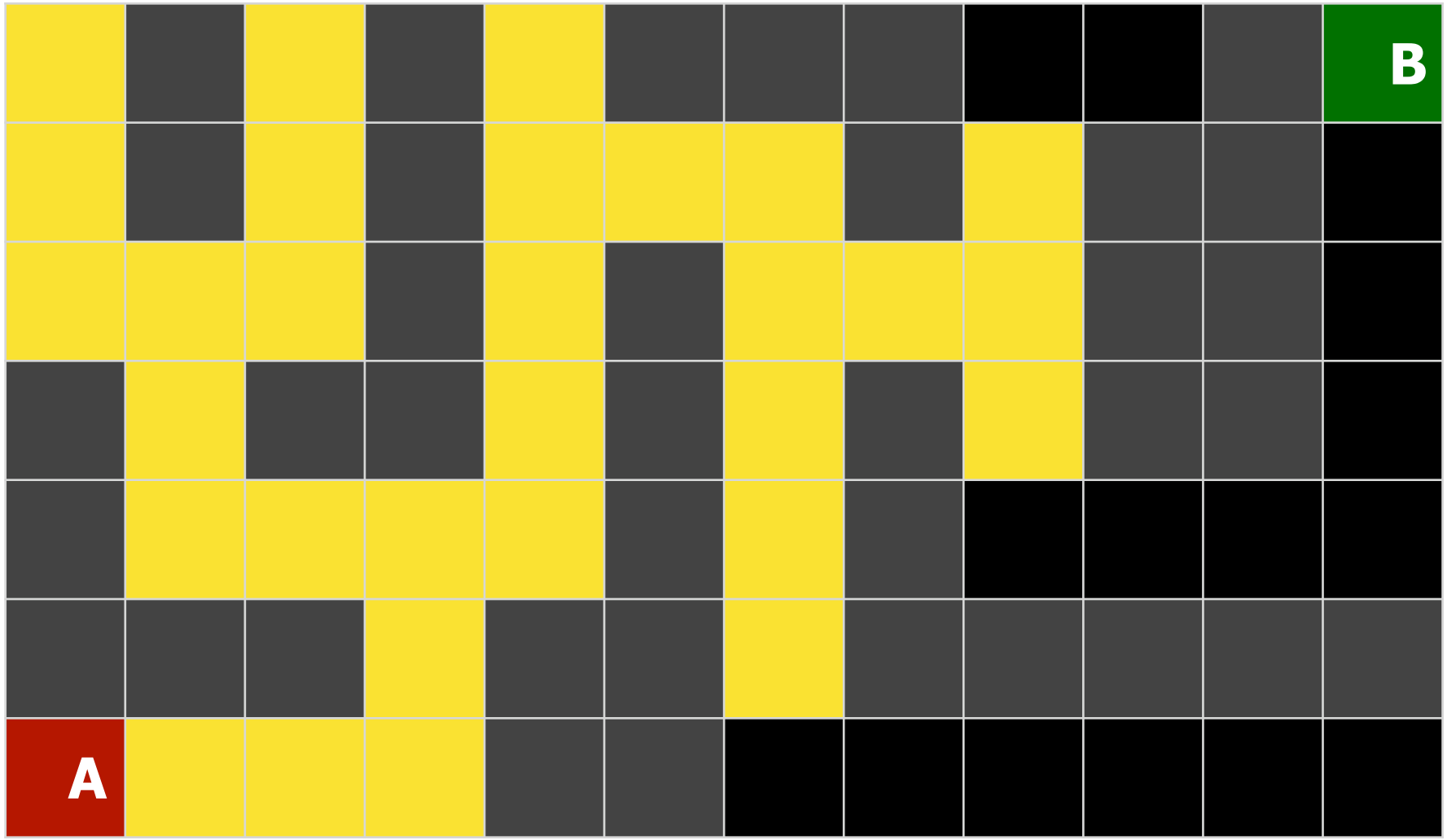
Breadth-First Search



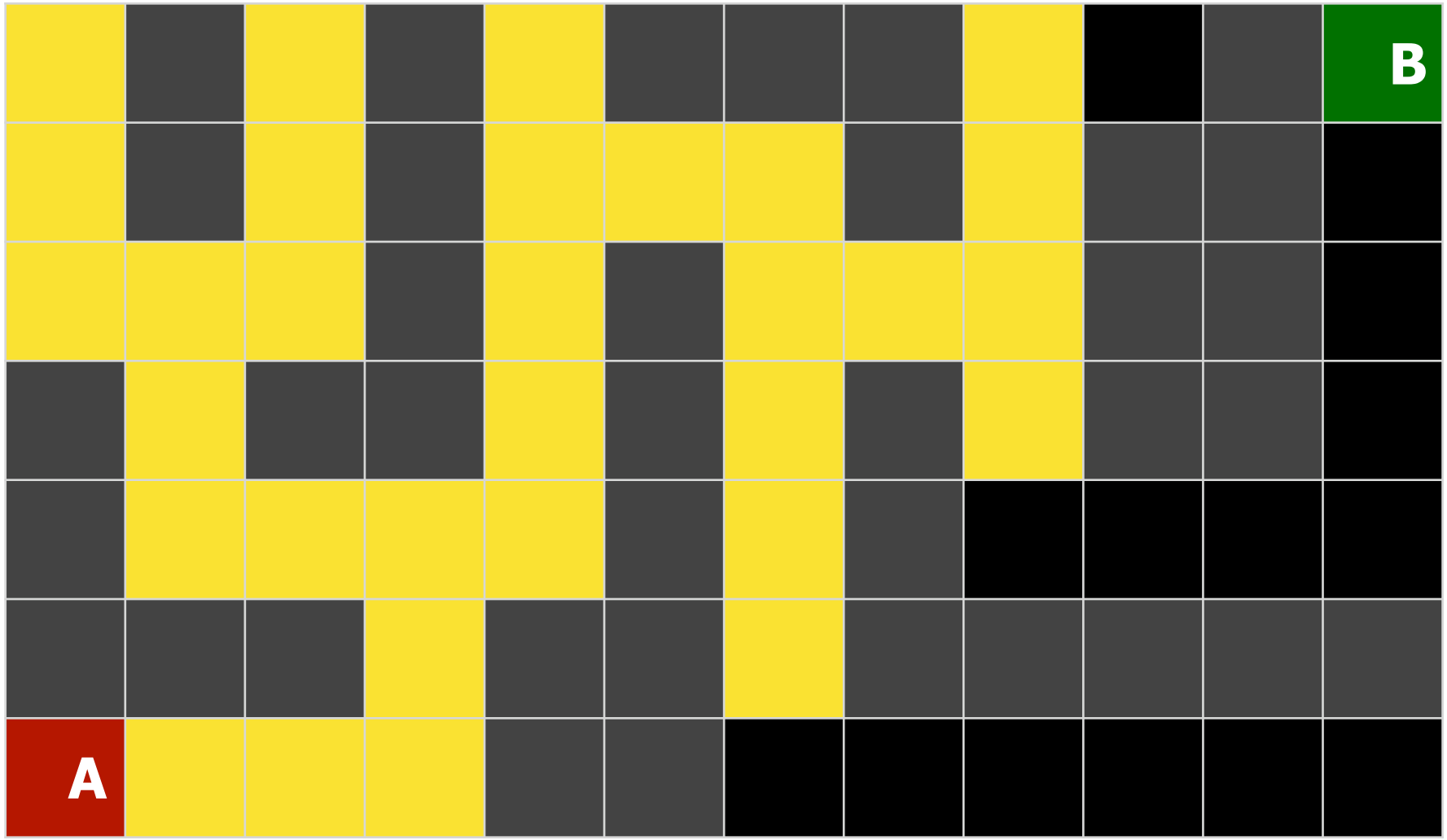
Breadth-First Search



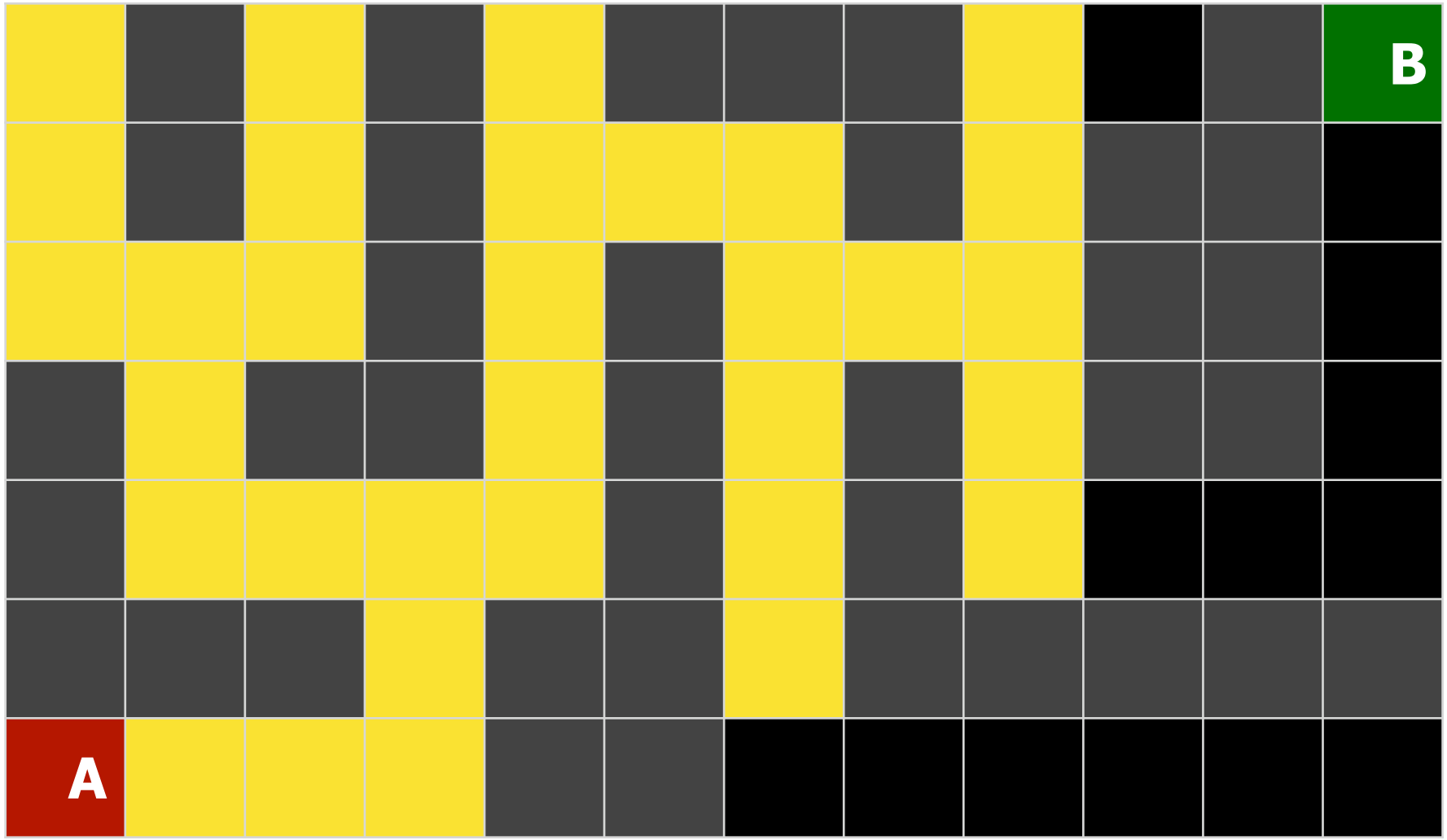
Breadth-First Search



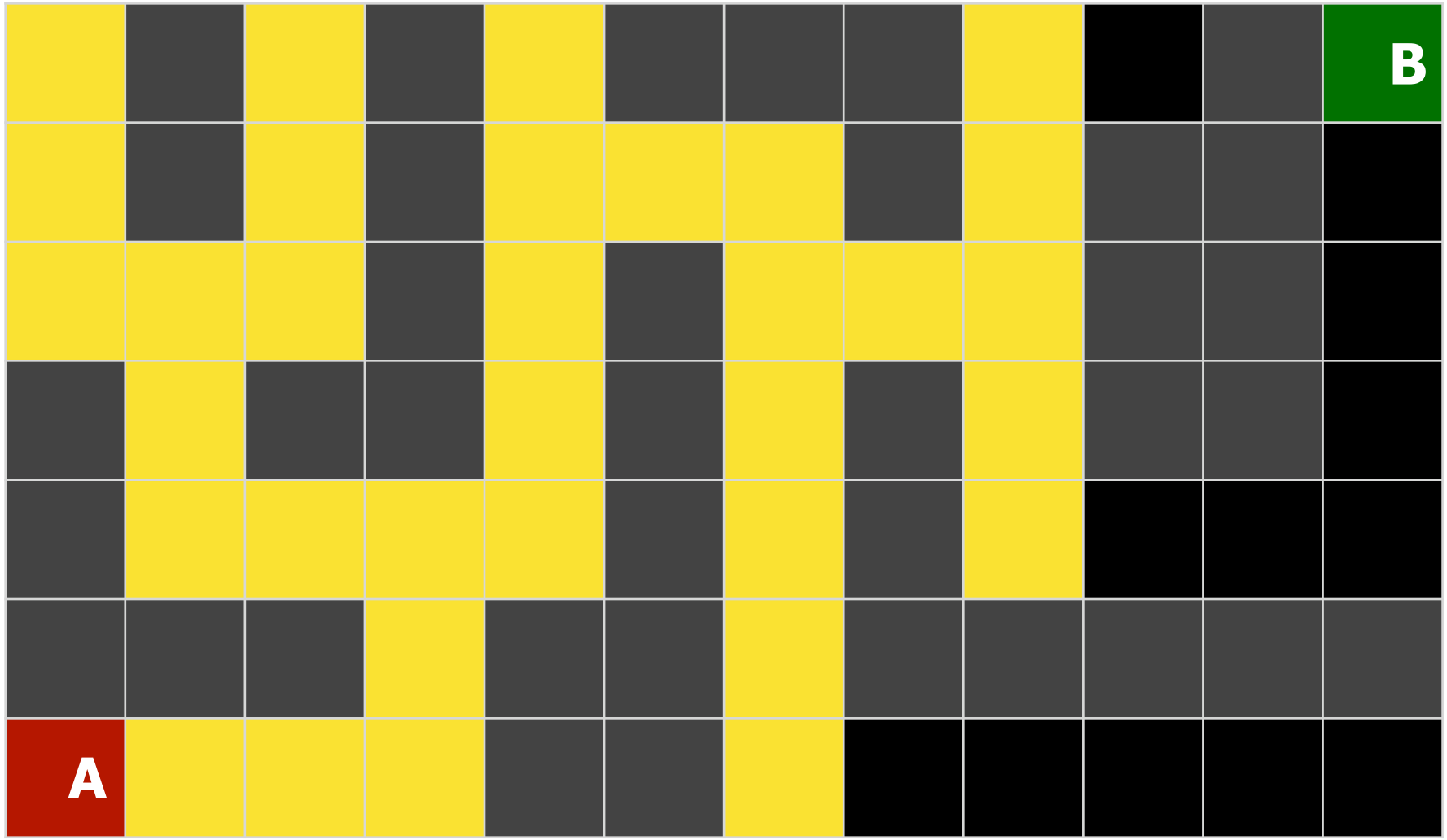
Breadth-First Search



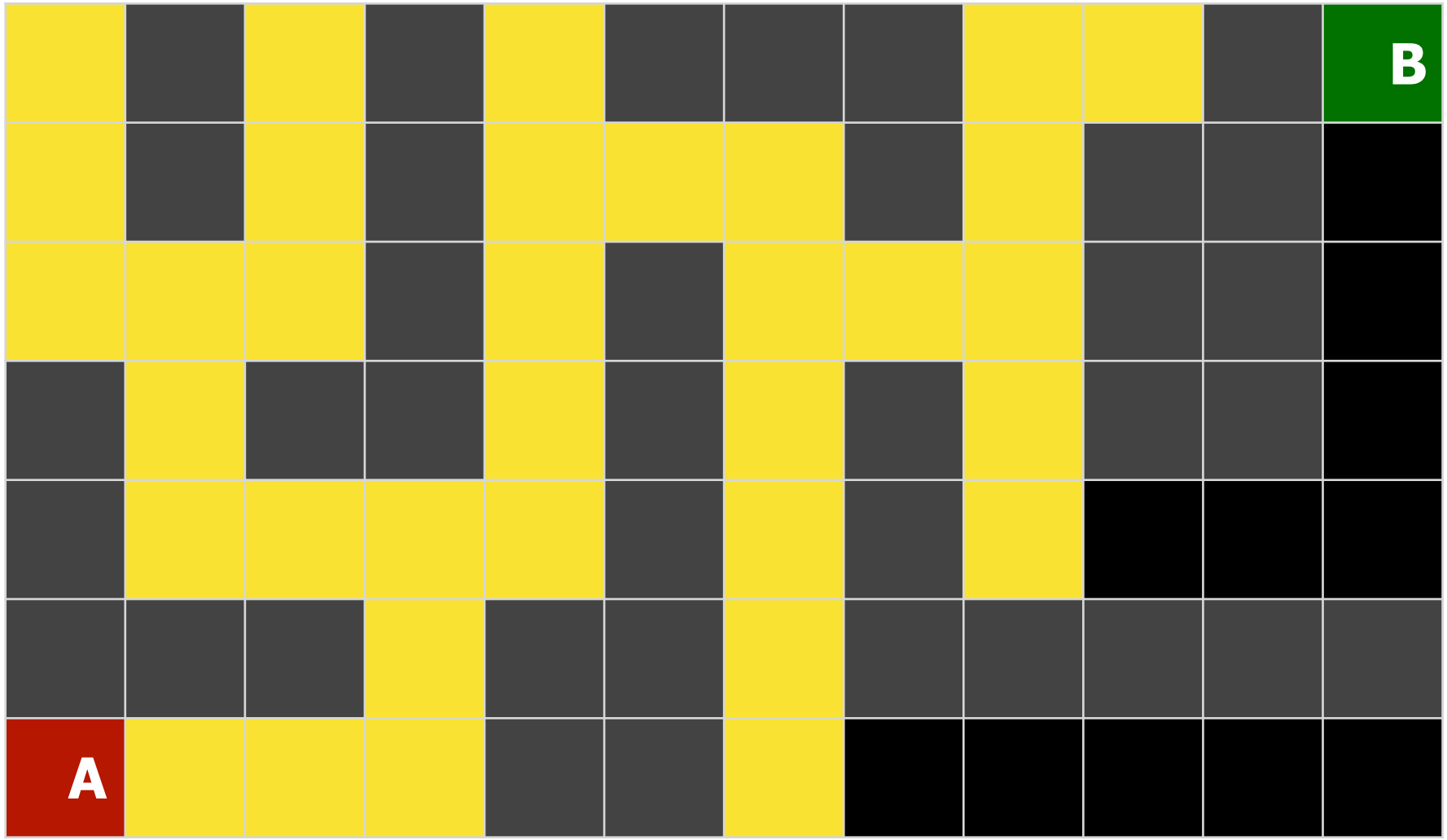
Breadth-First Search



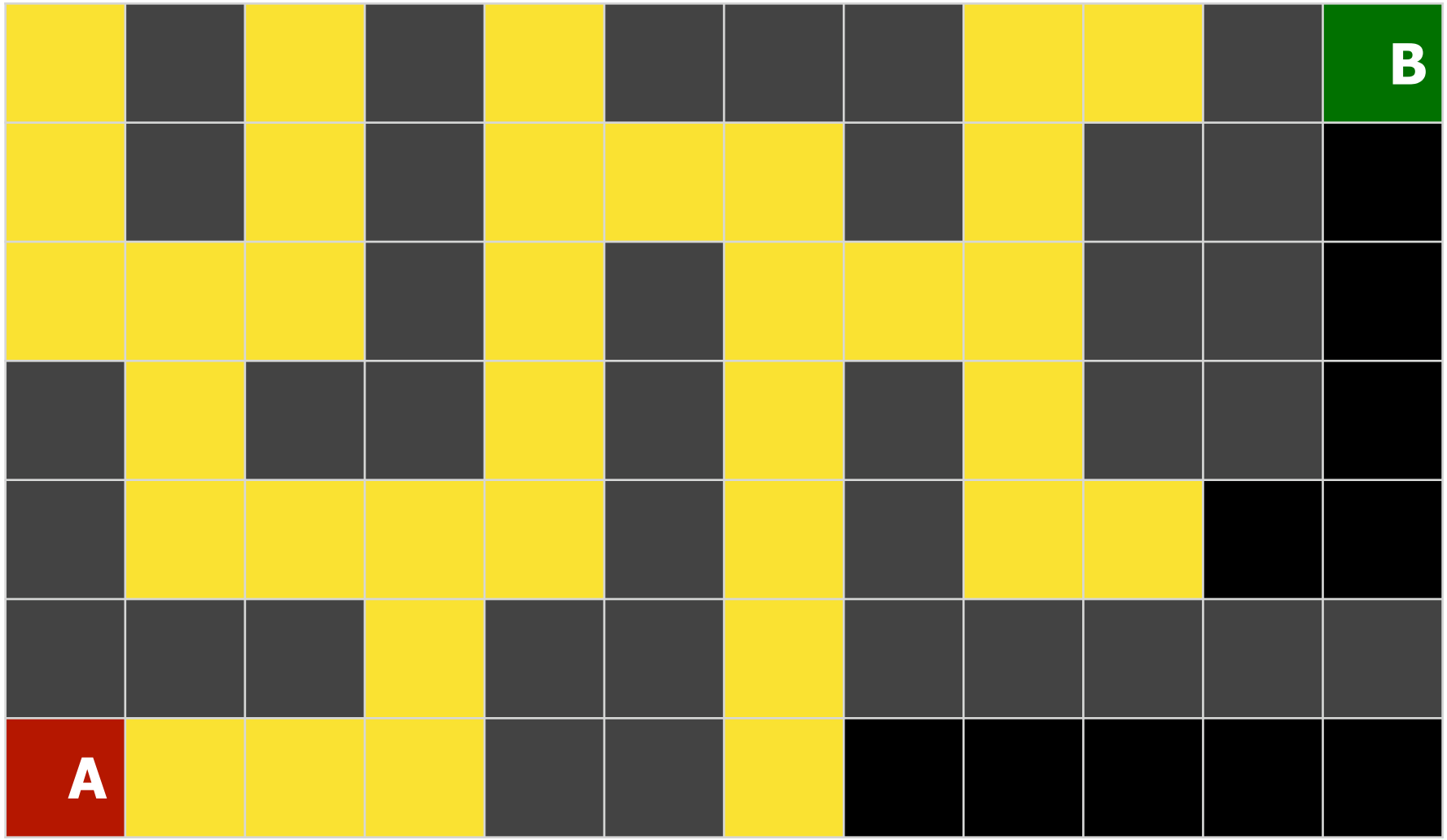
Breadth-First Search



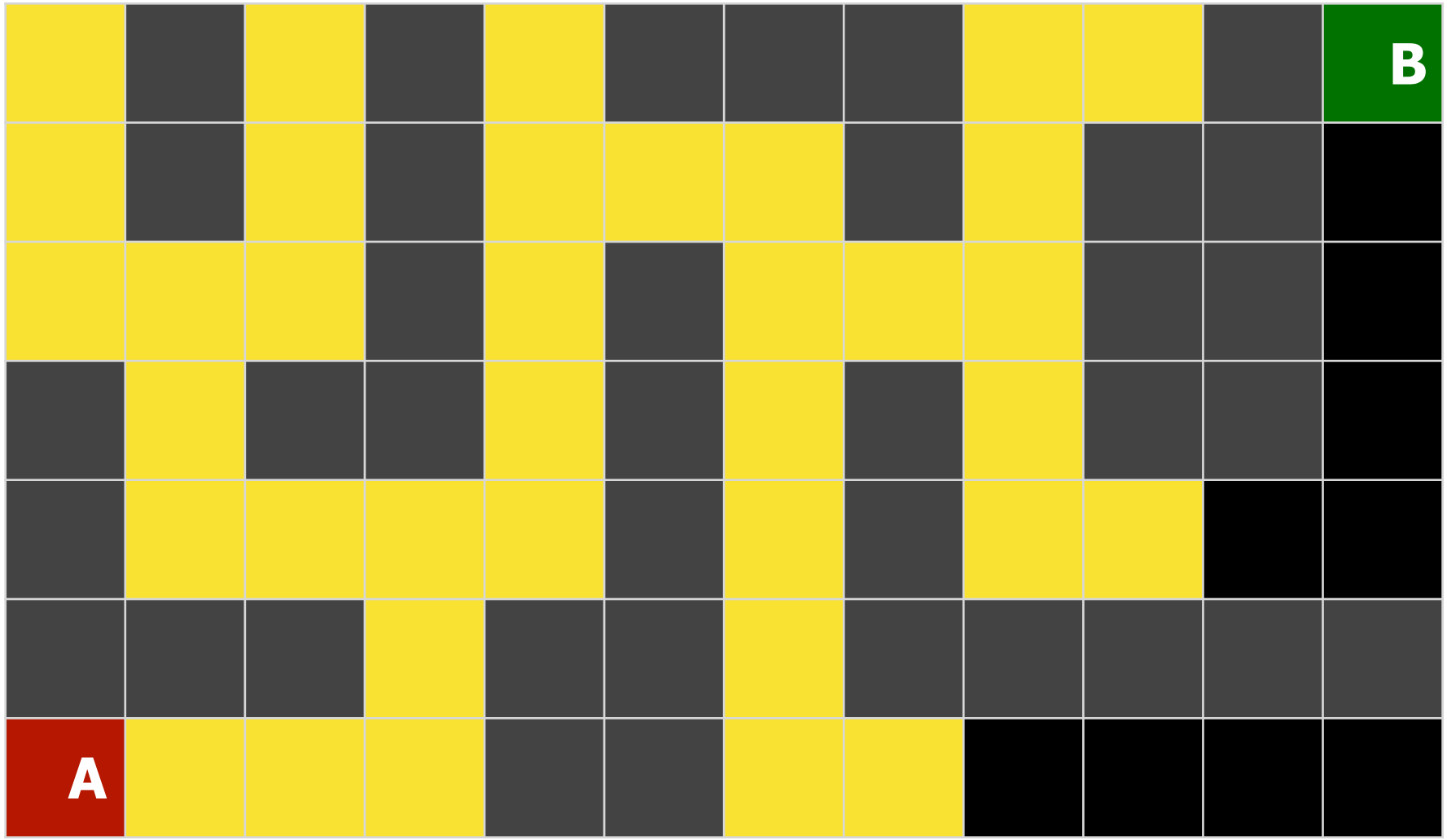
Breadth-First Search



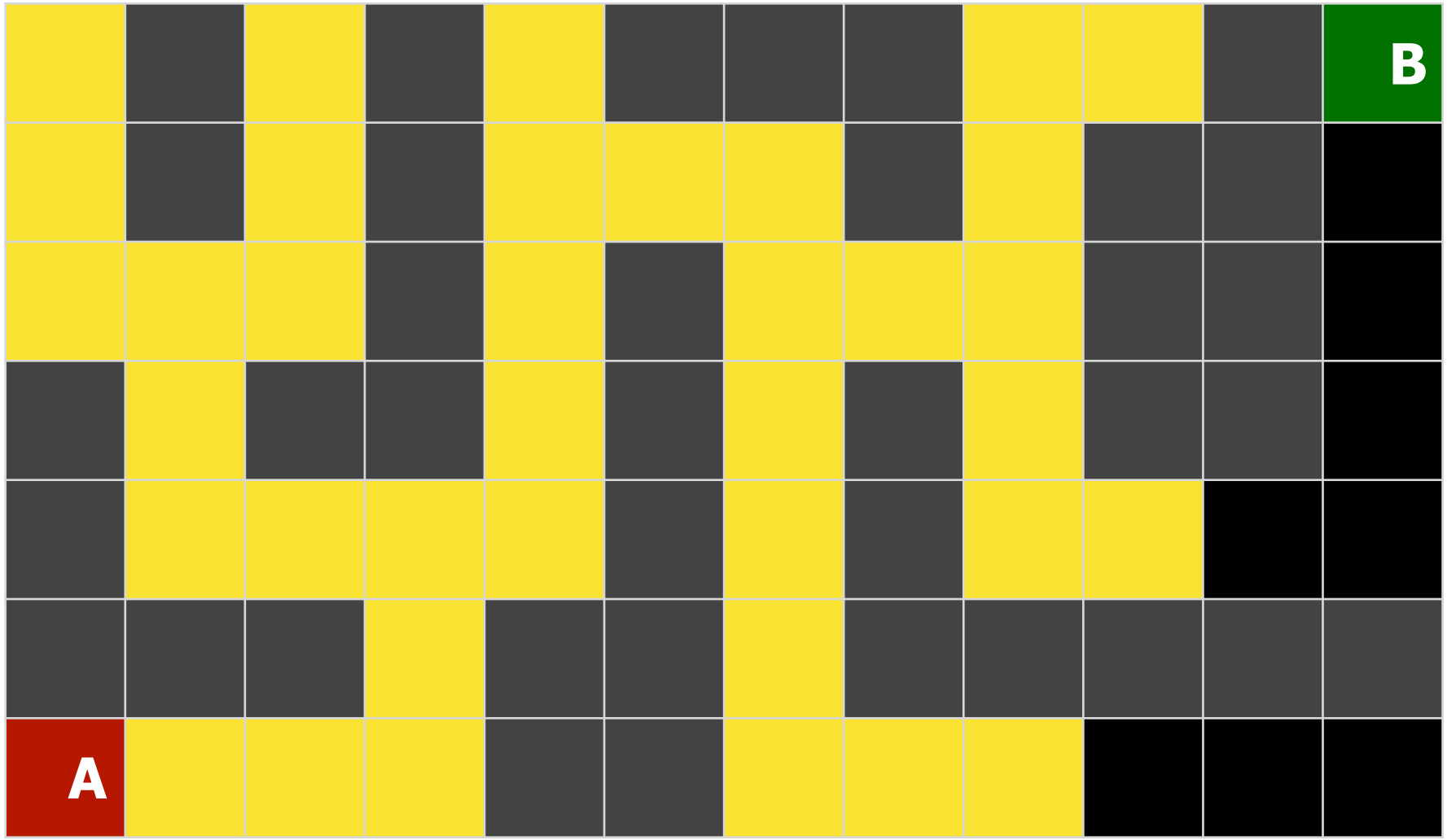
Breadth-First Search



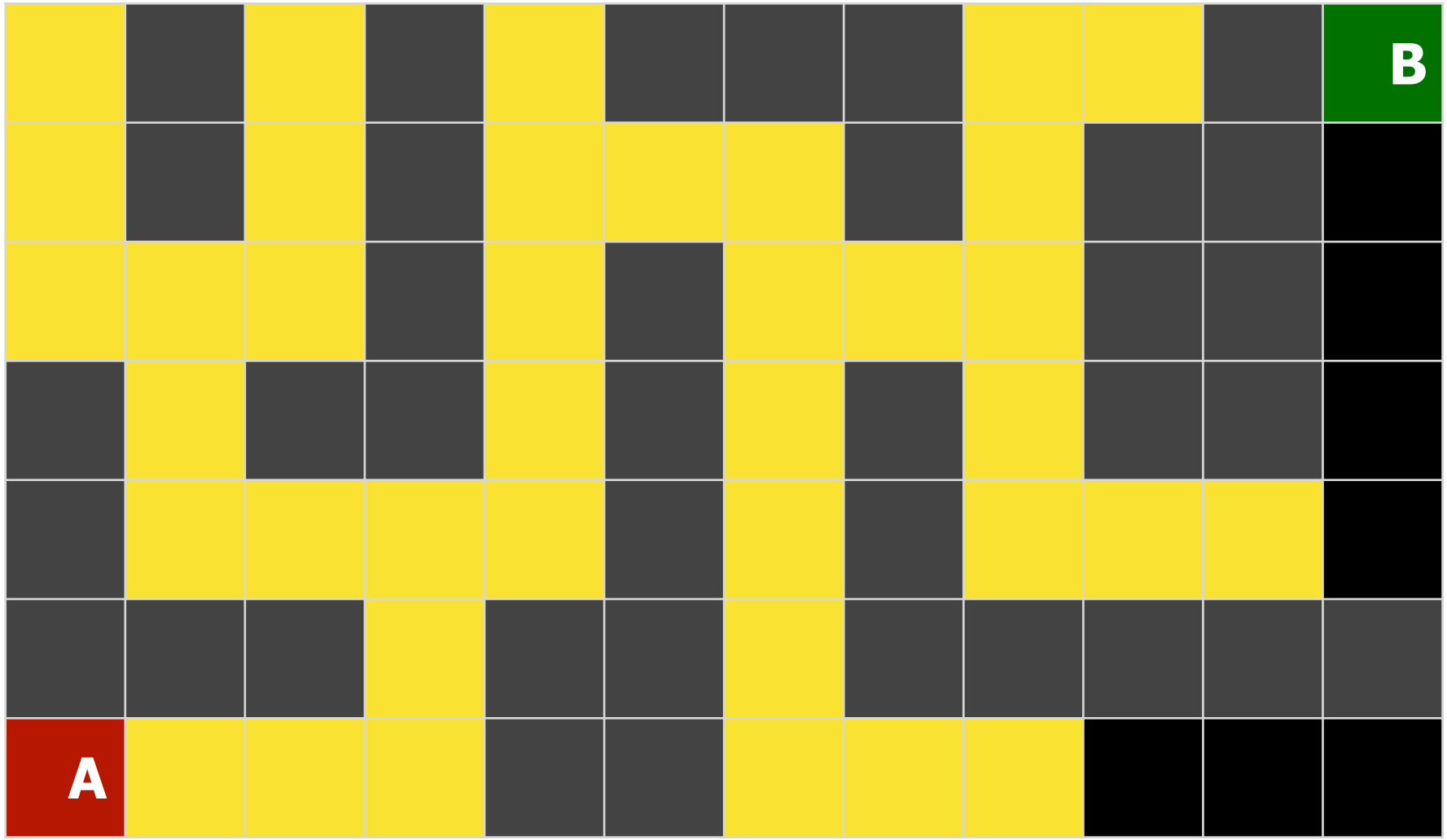
Breadth-First Search



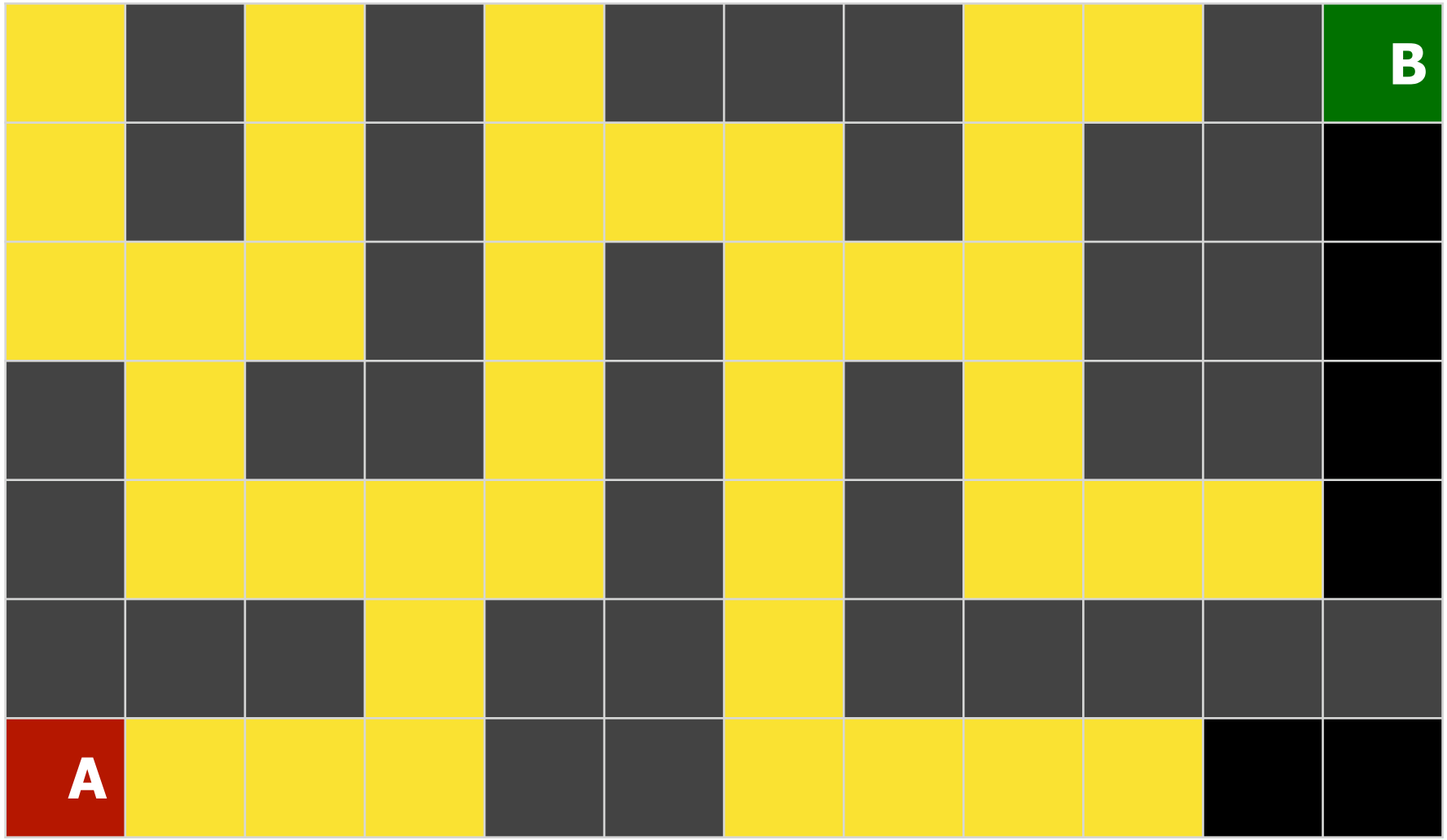
Breadth-First Search



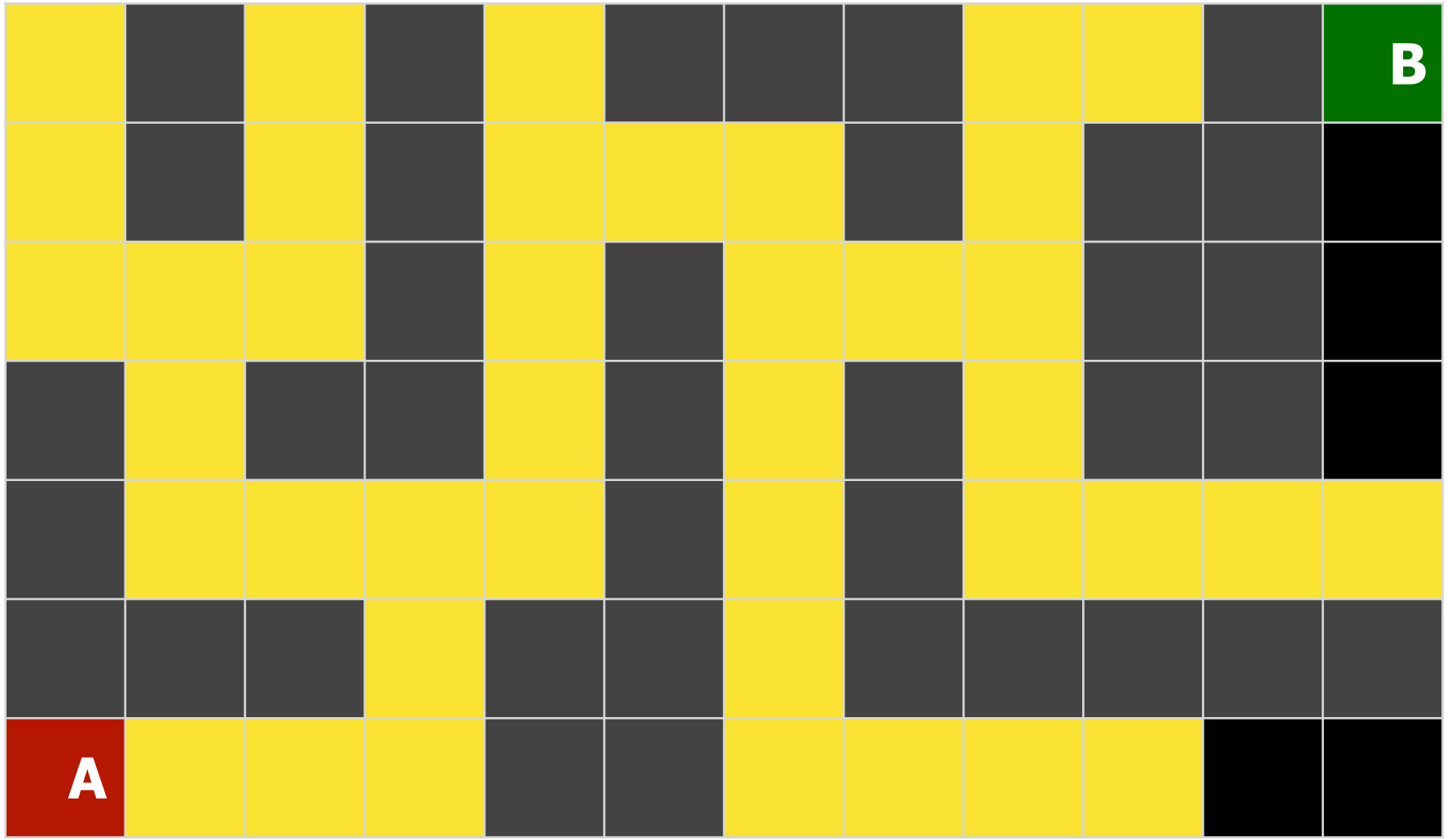
Breadth-First Search



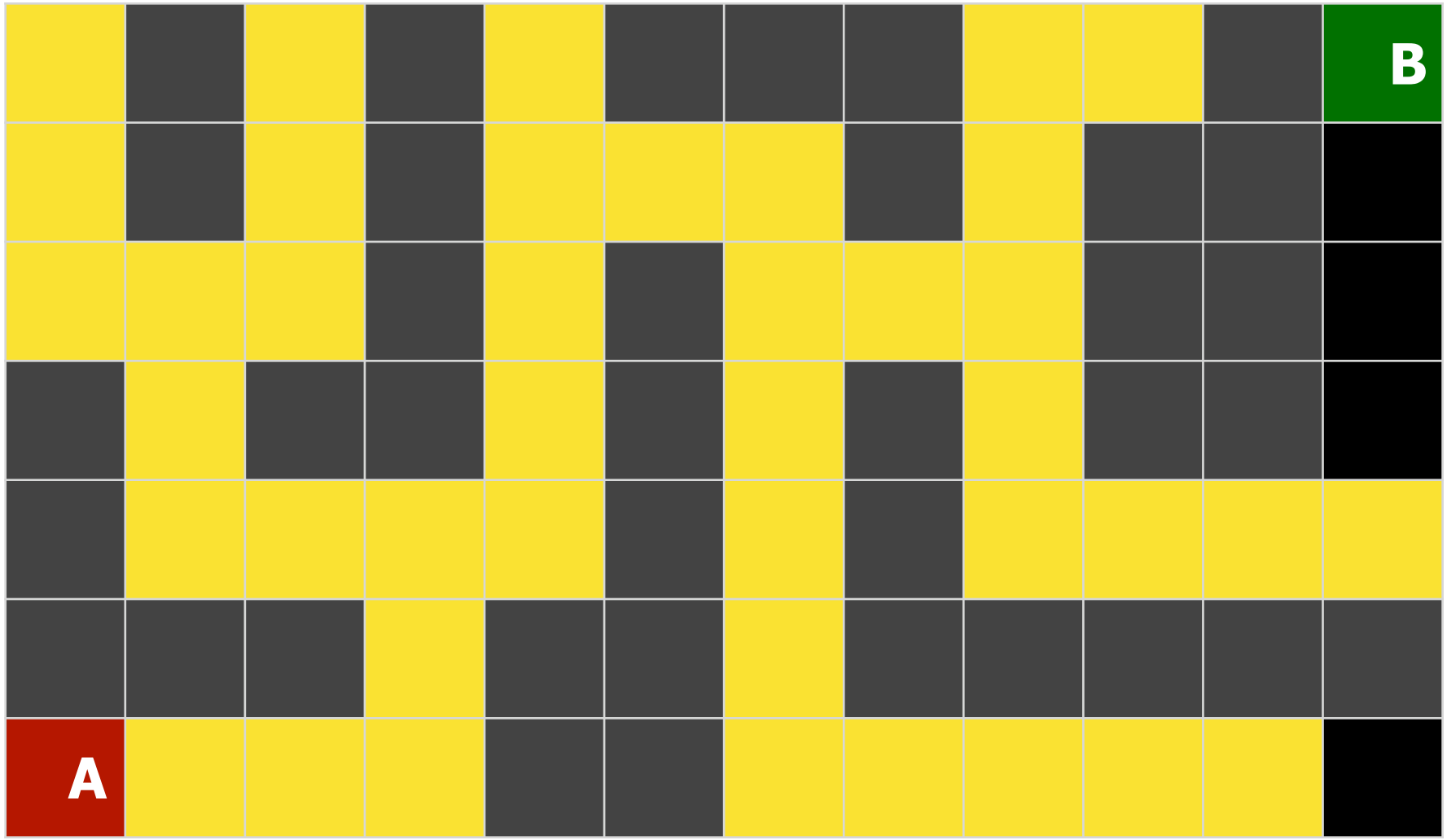
Breadth-First Search



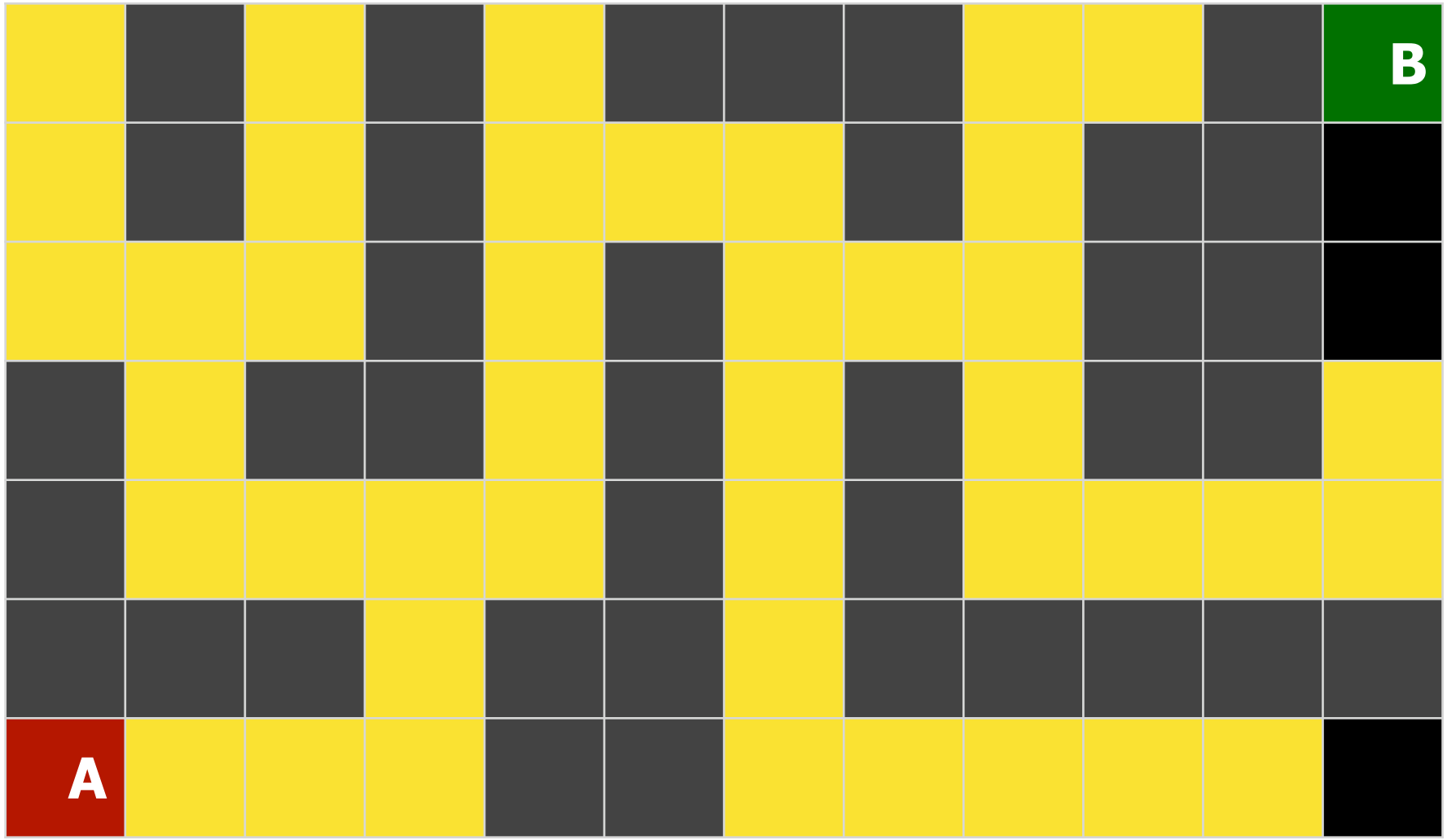
Breadth-First Search



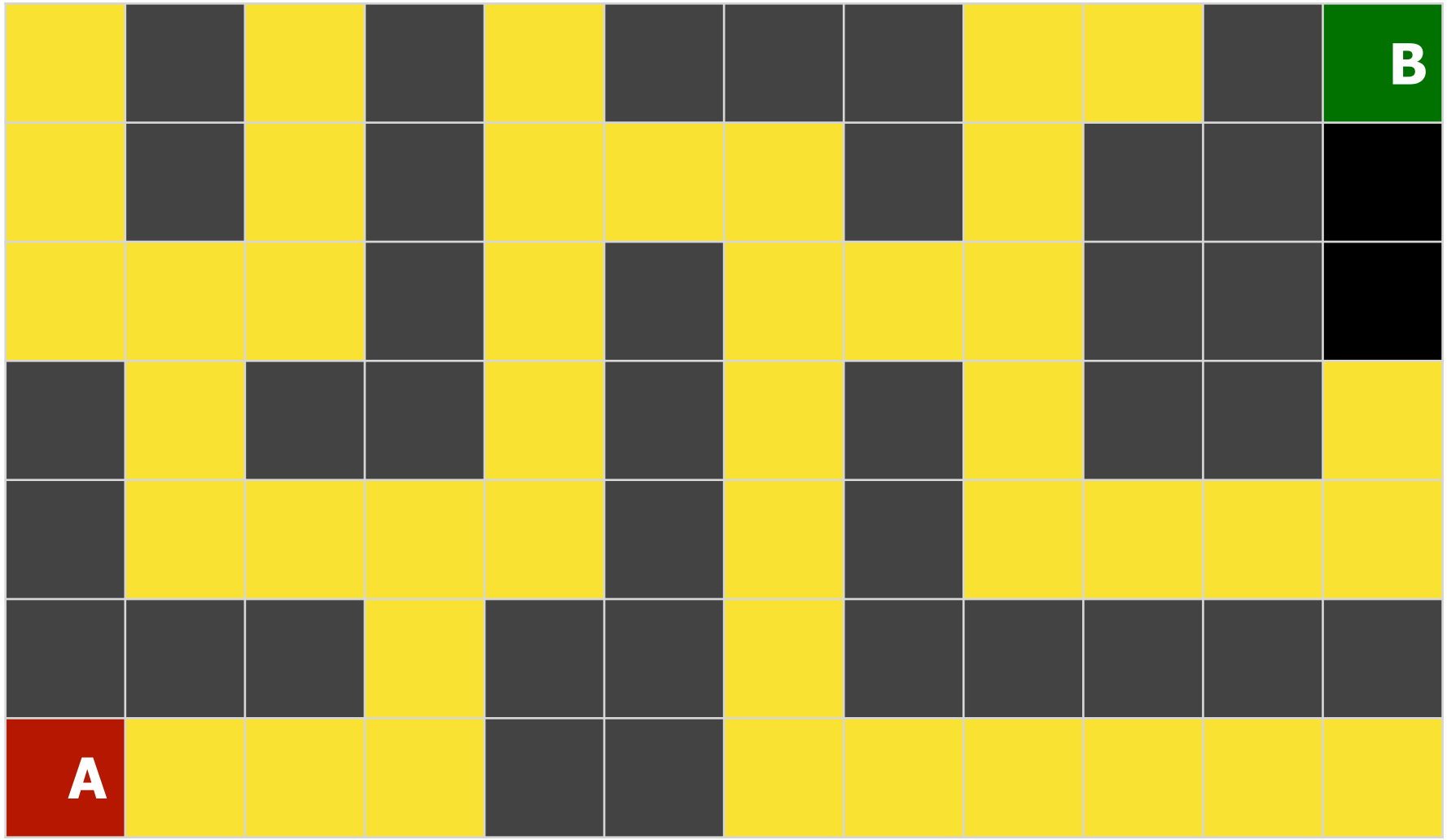
Breadth-First Search



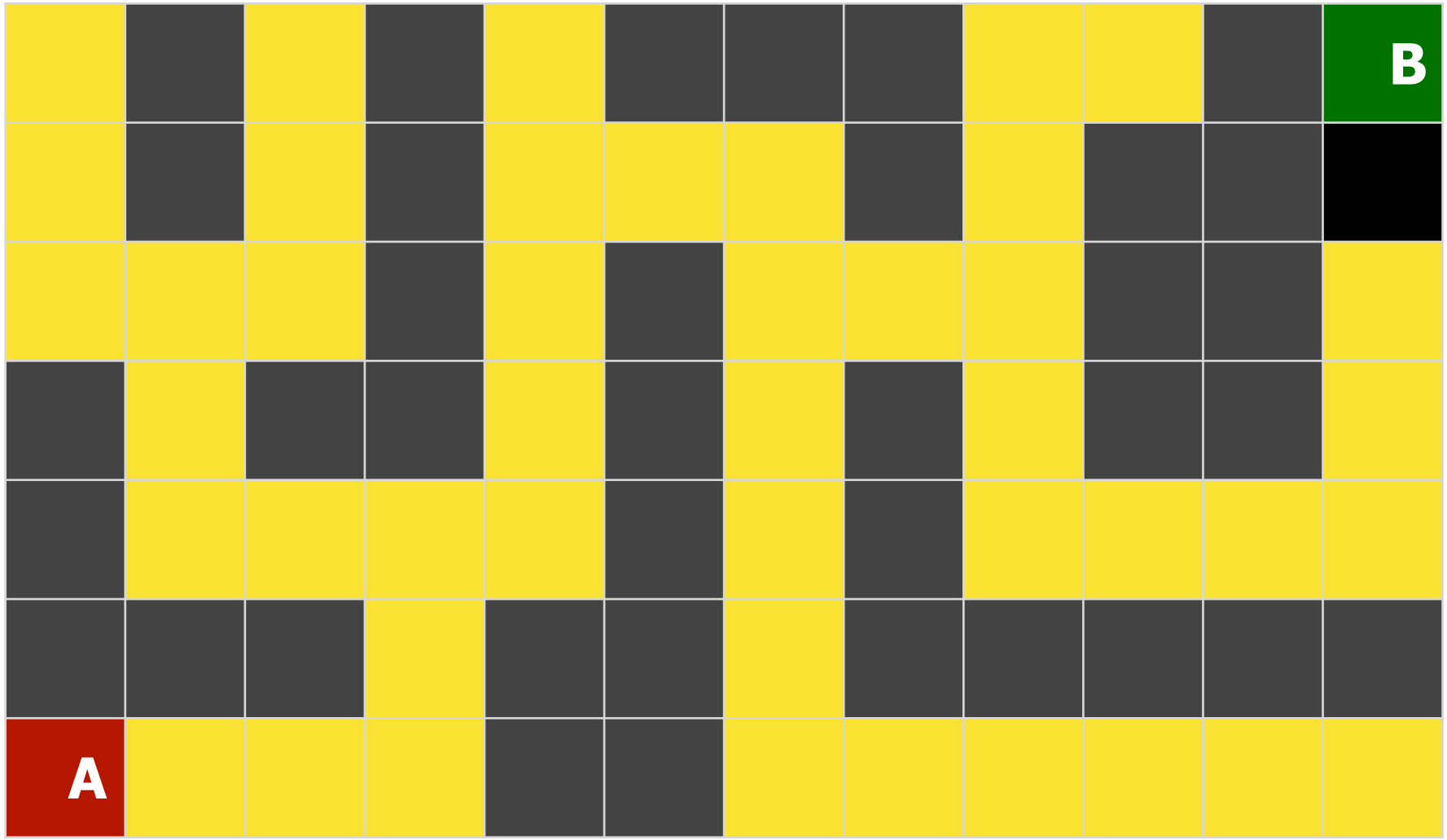
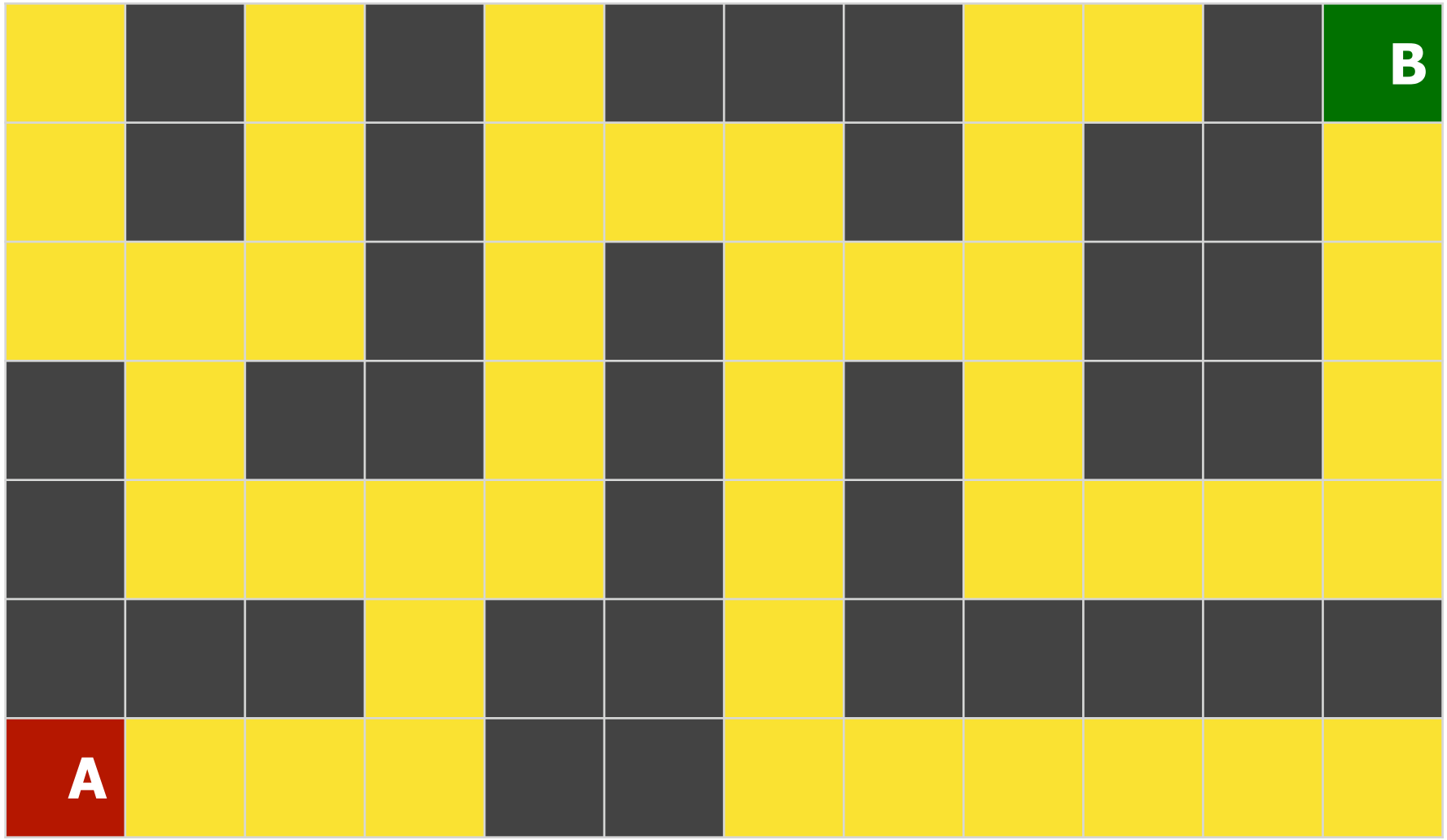
Breadth-First Search



Breadth-First Search



Breadth-First Search



Breadth-First Search