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).
- An **action cost function**, denoted by ACTION-COST(s,a,s') that gives the numeric cost of applying action a in state s to reach state s'.

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

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.

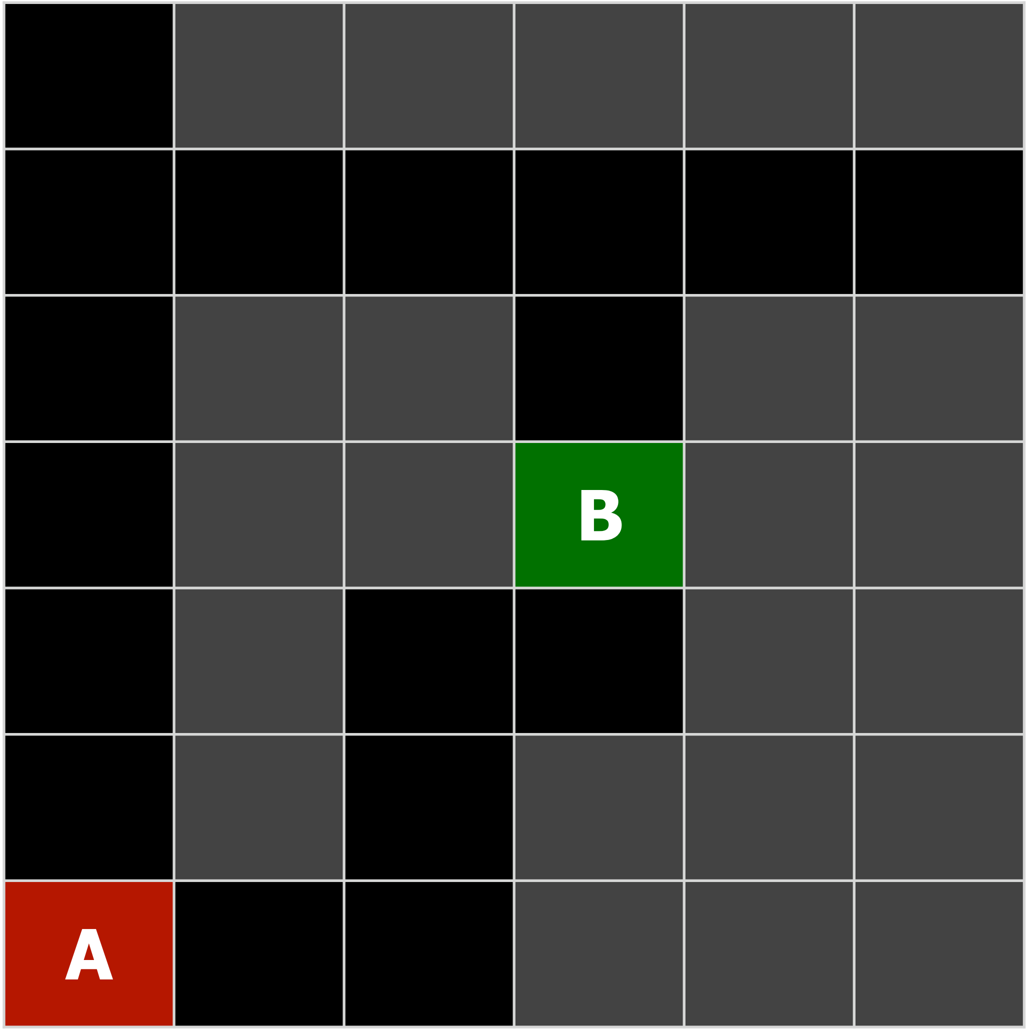
Breadth-First Search

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

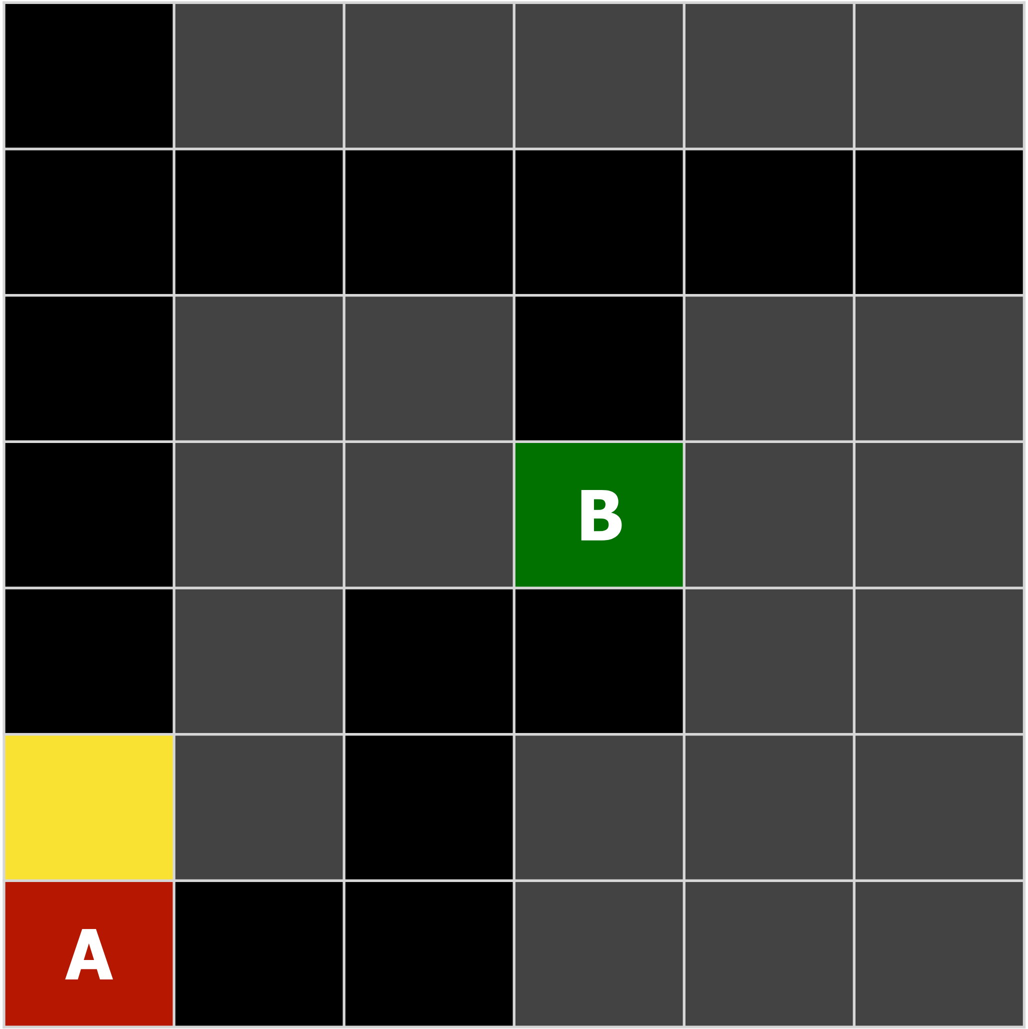
```
def function(search problem):
    frontier = []
    visited = []
    add initial state to frontier

    while frontier is not empty:
        remove first state 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
                for each neighbor of s:
                        add neighbor to frontier list
    return failure
```

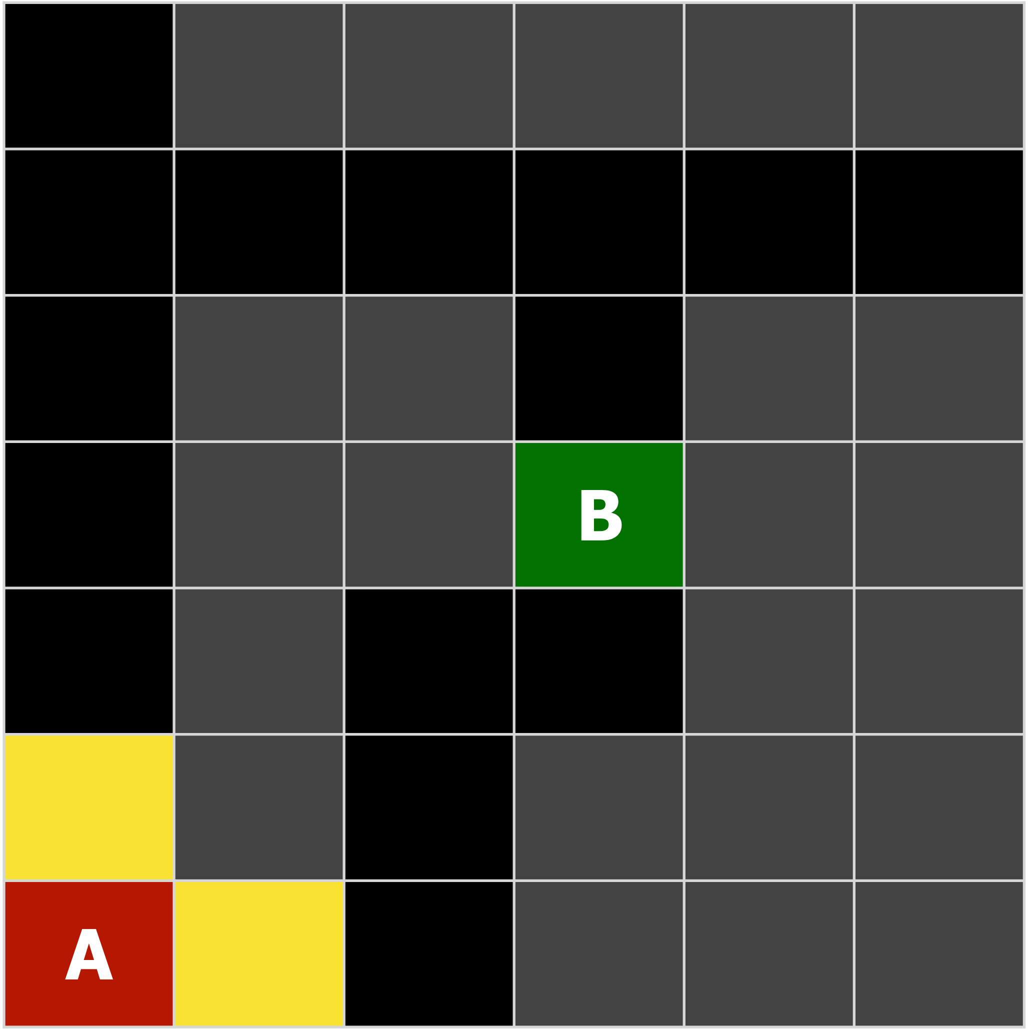
Breadth-First Search



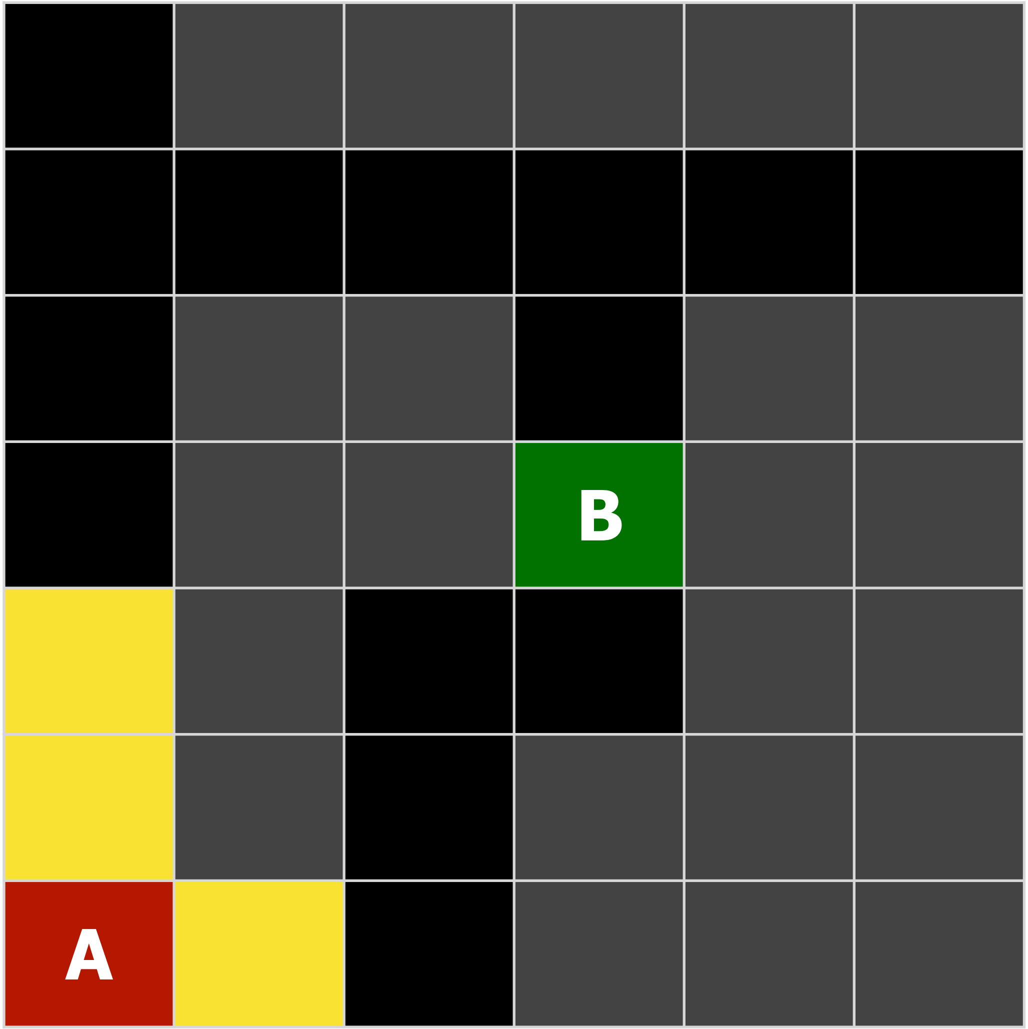
Breadth-First Search



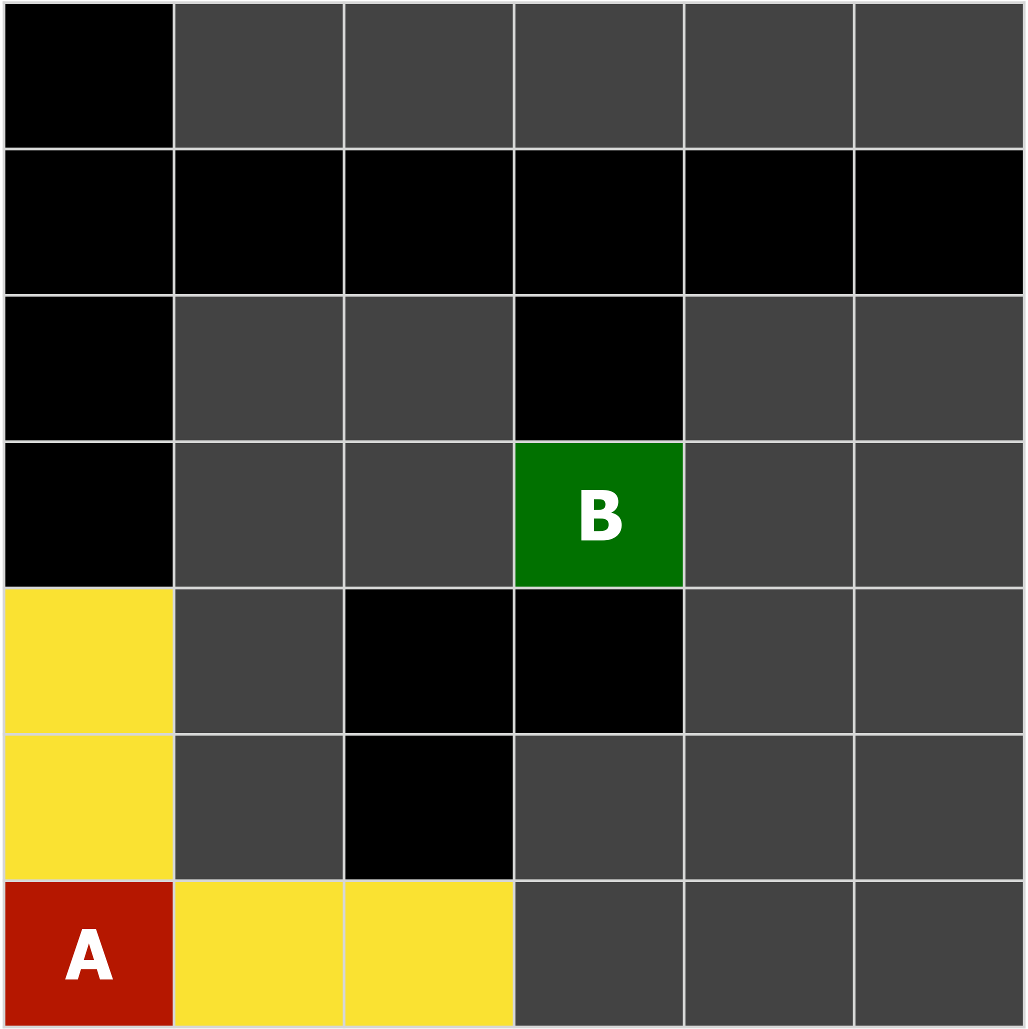
Breadth-First Search



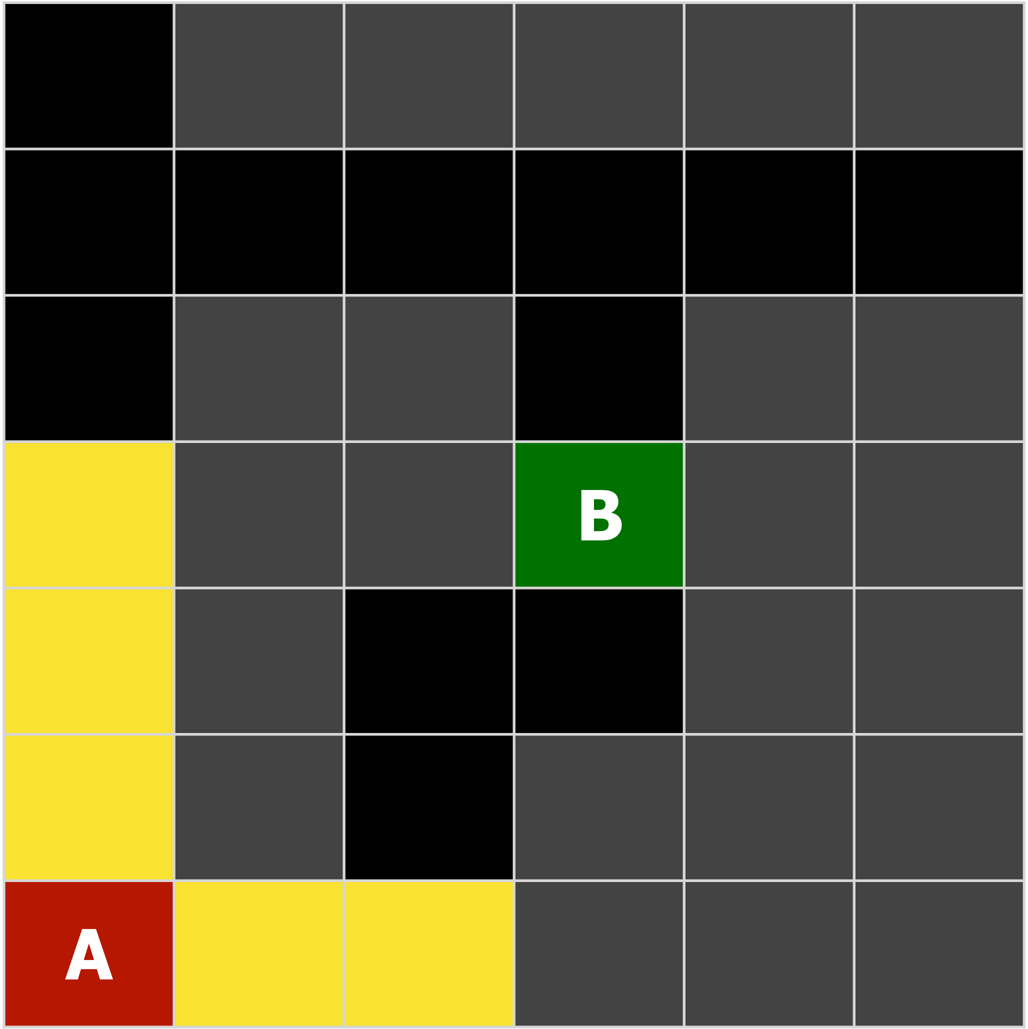
Breadth-First Search



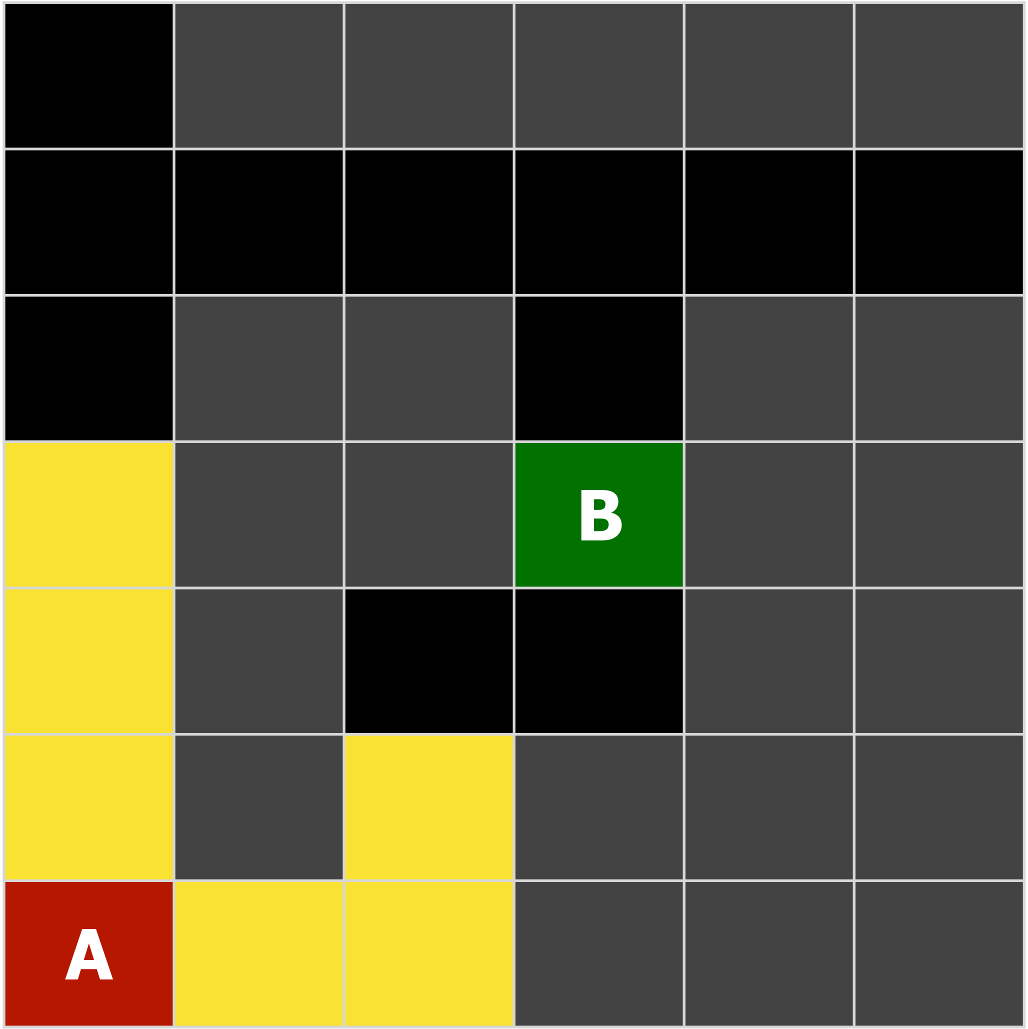
Breadth-First Search



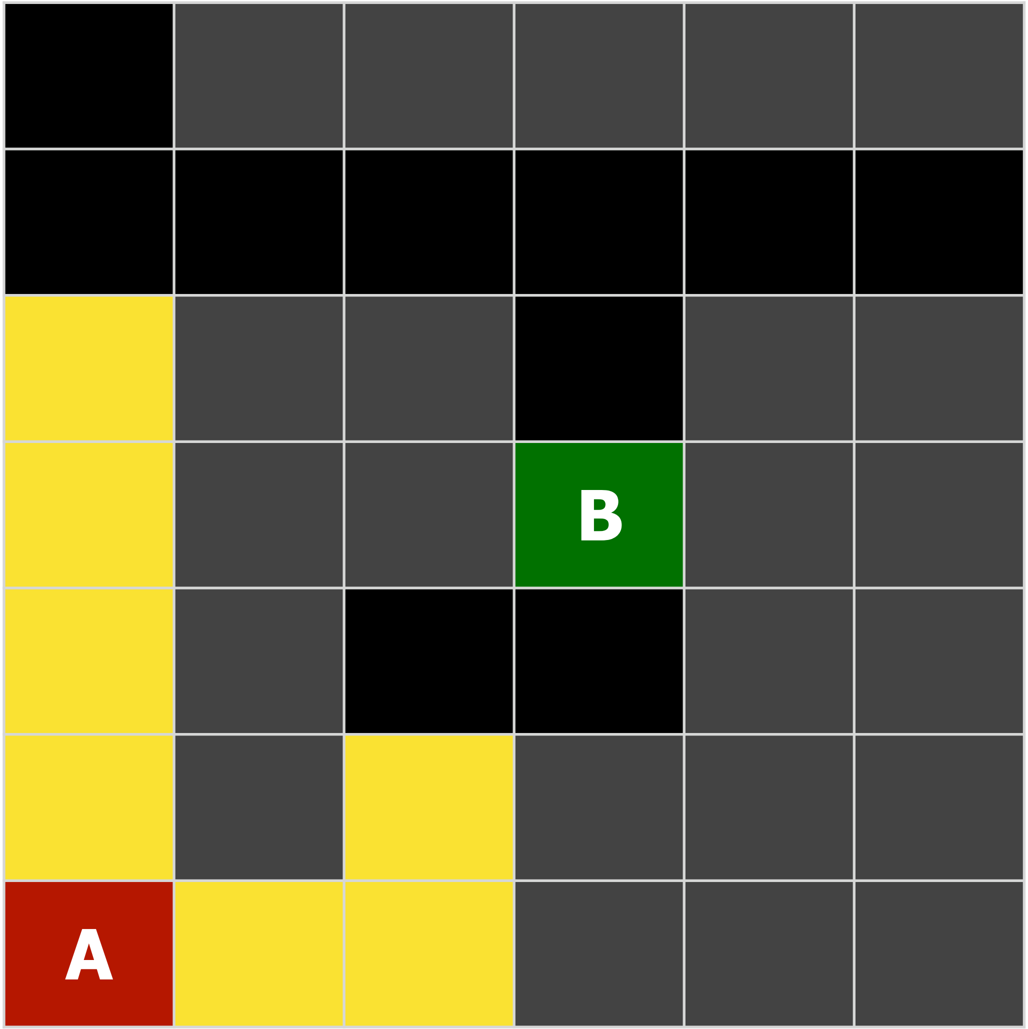
Breadth-First Search



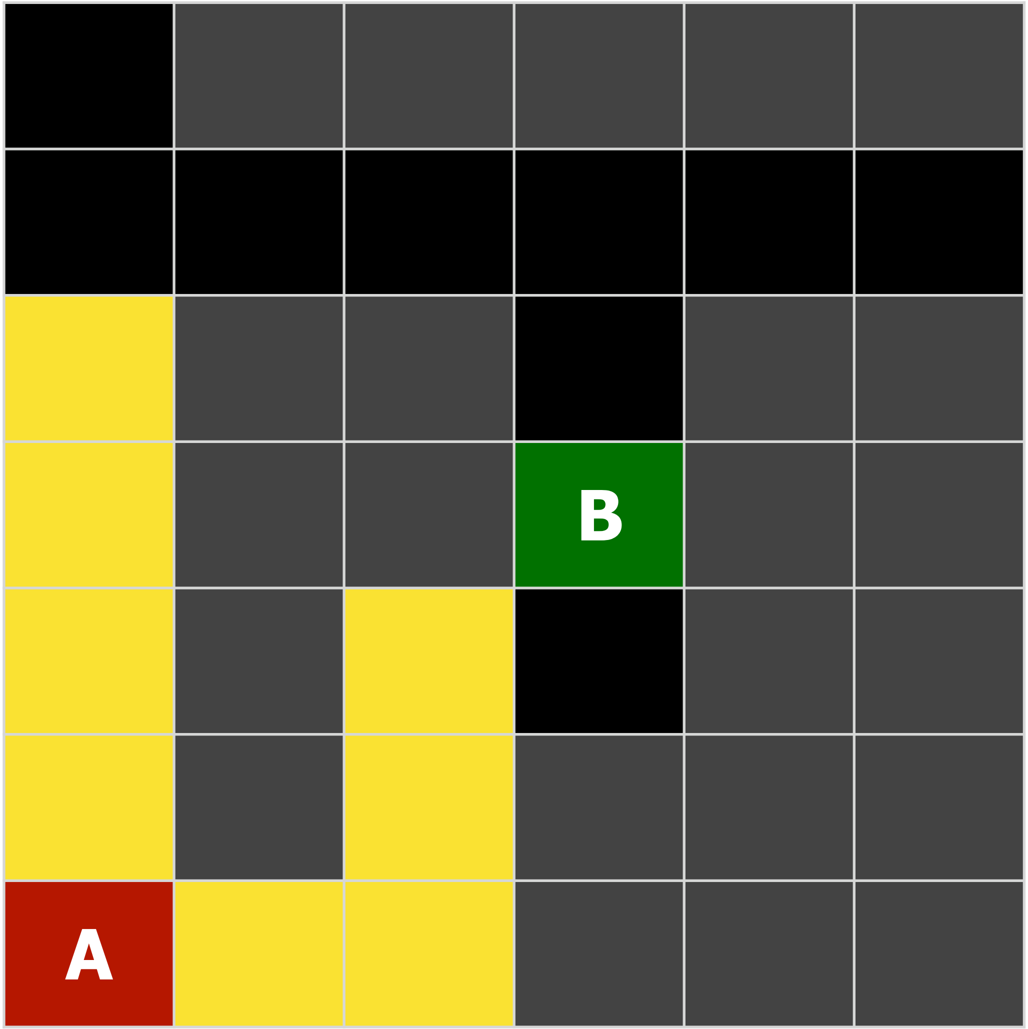
Breadth-First Search



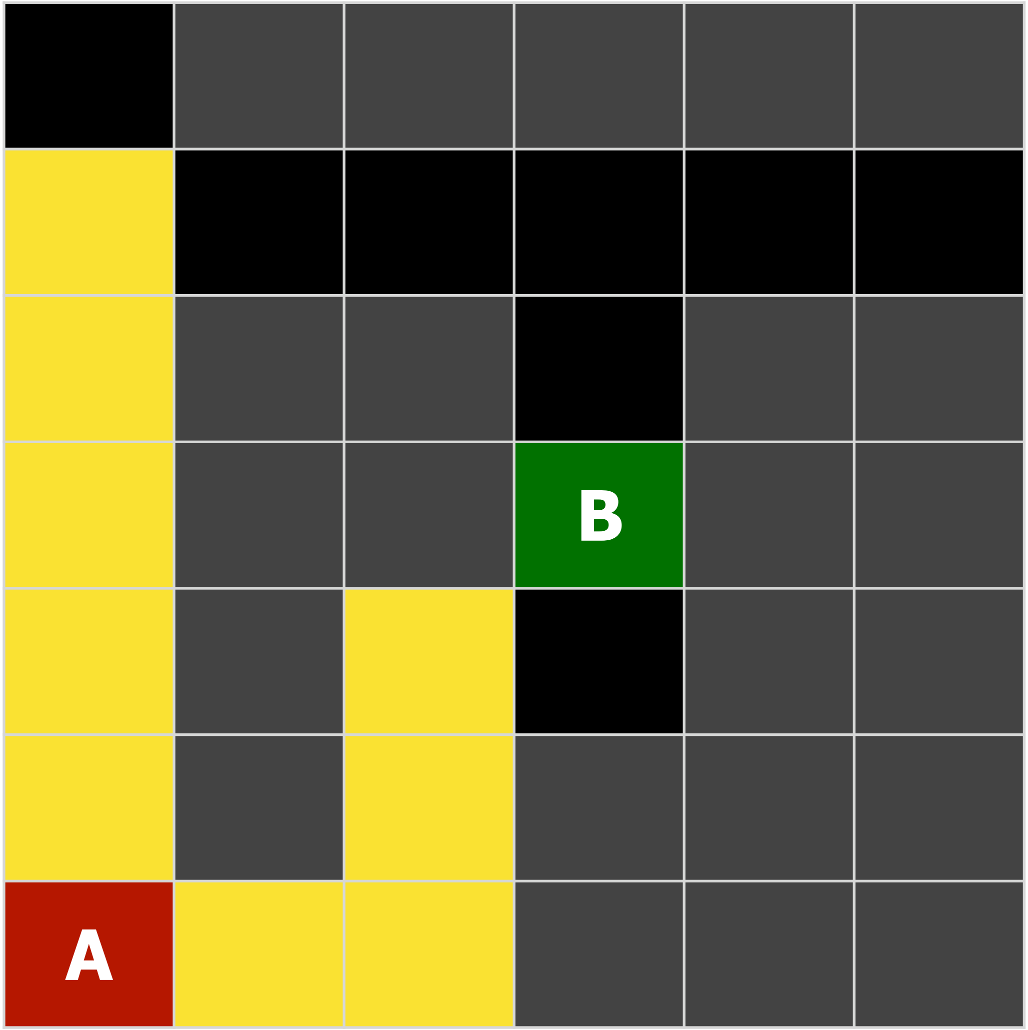
Breadth-First Search



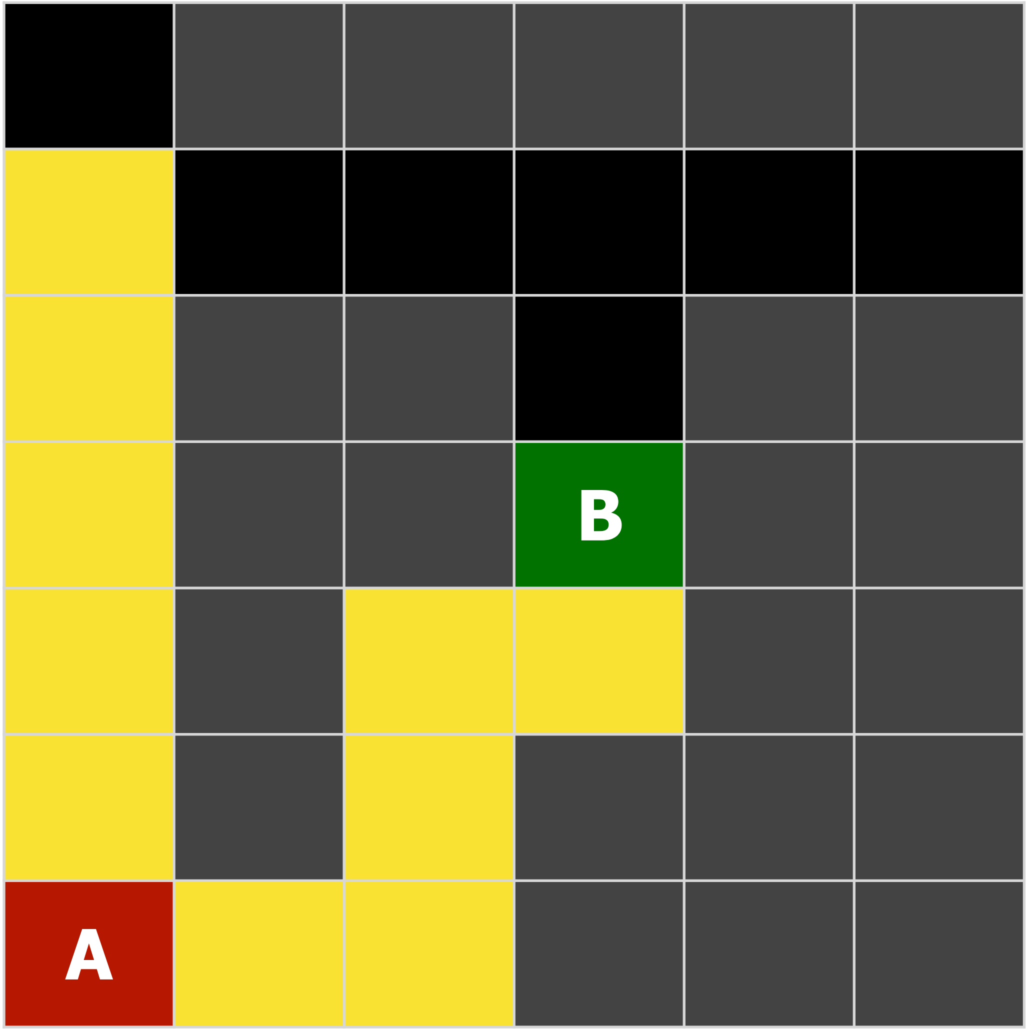
Breadth-First Search



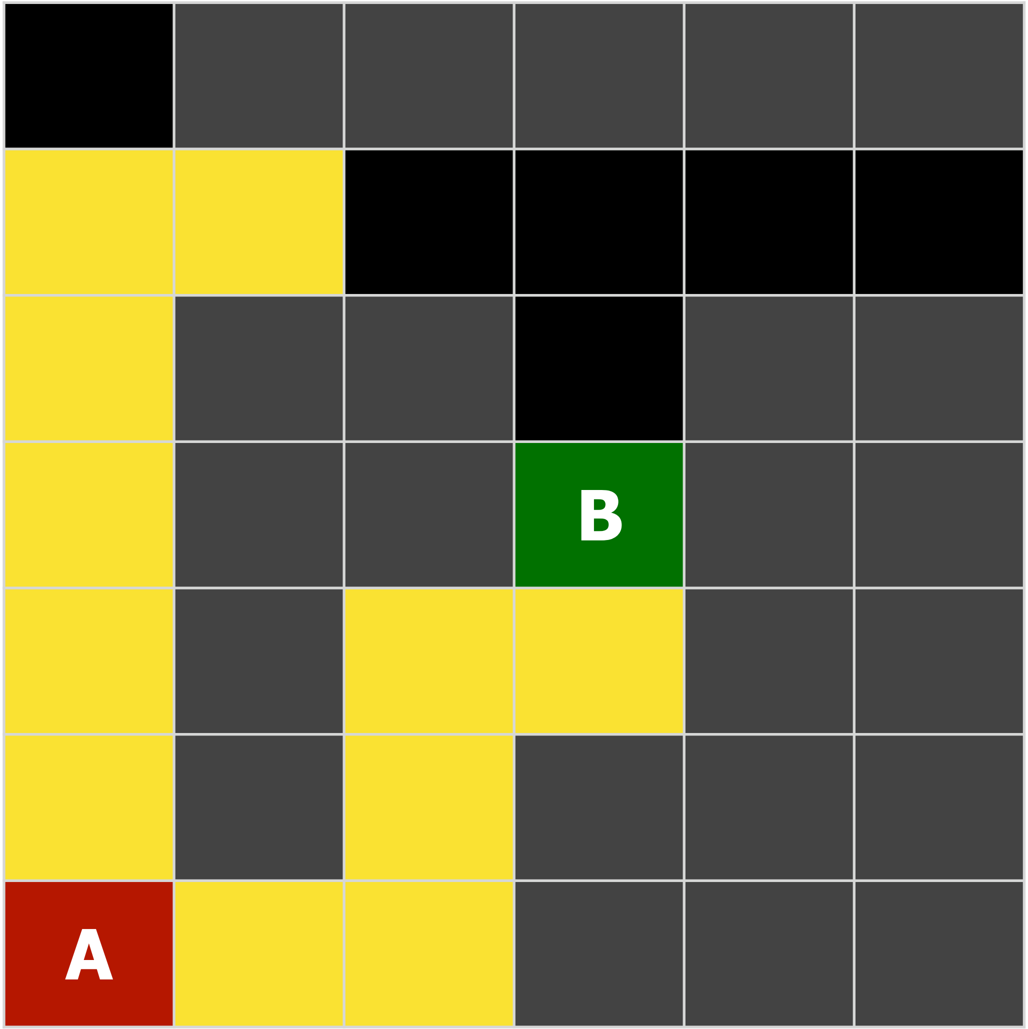
Breadth-First Search



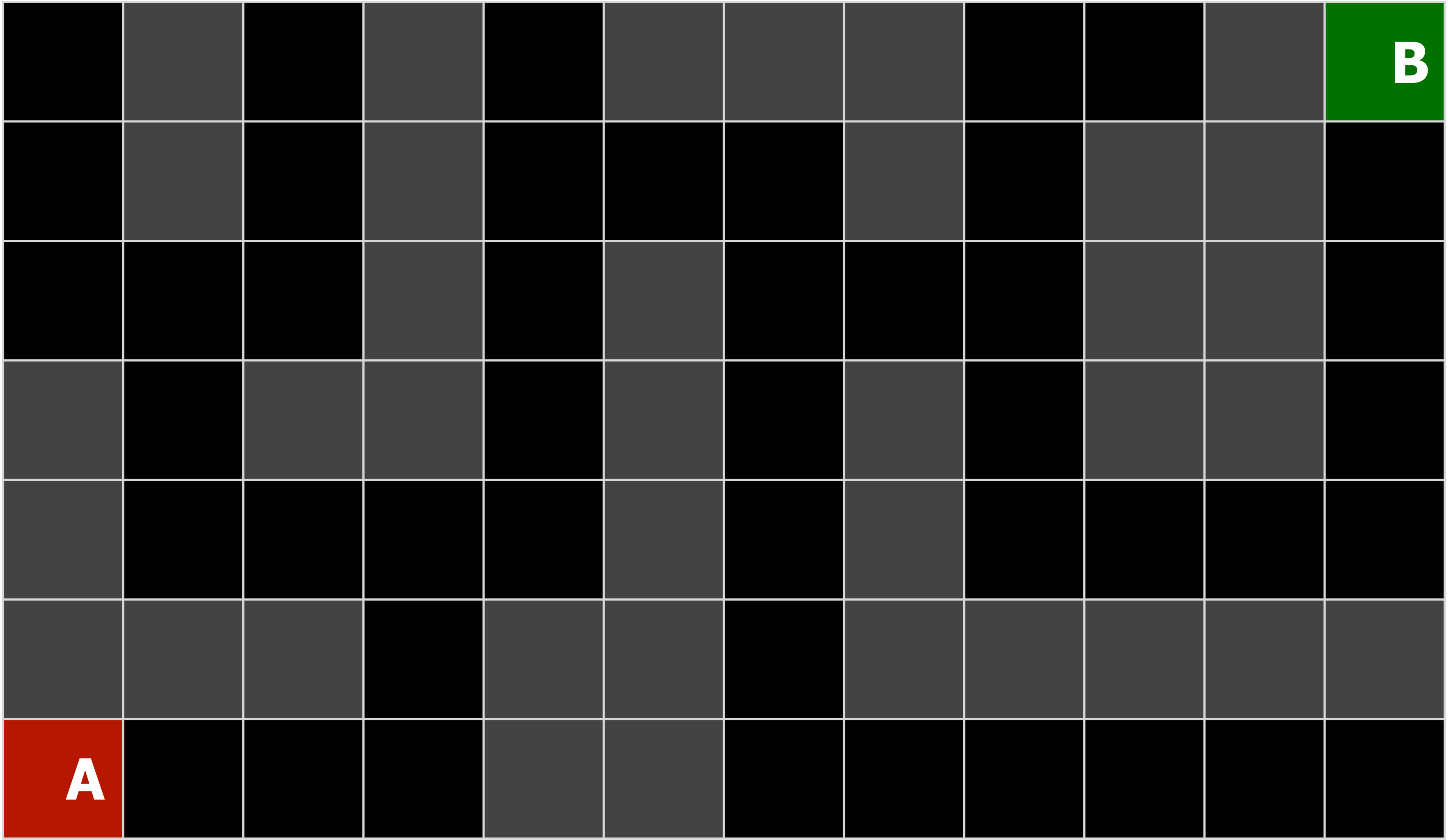
Breadth-First Search



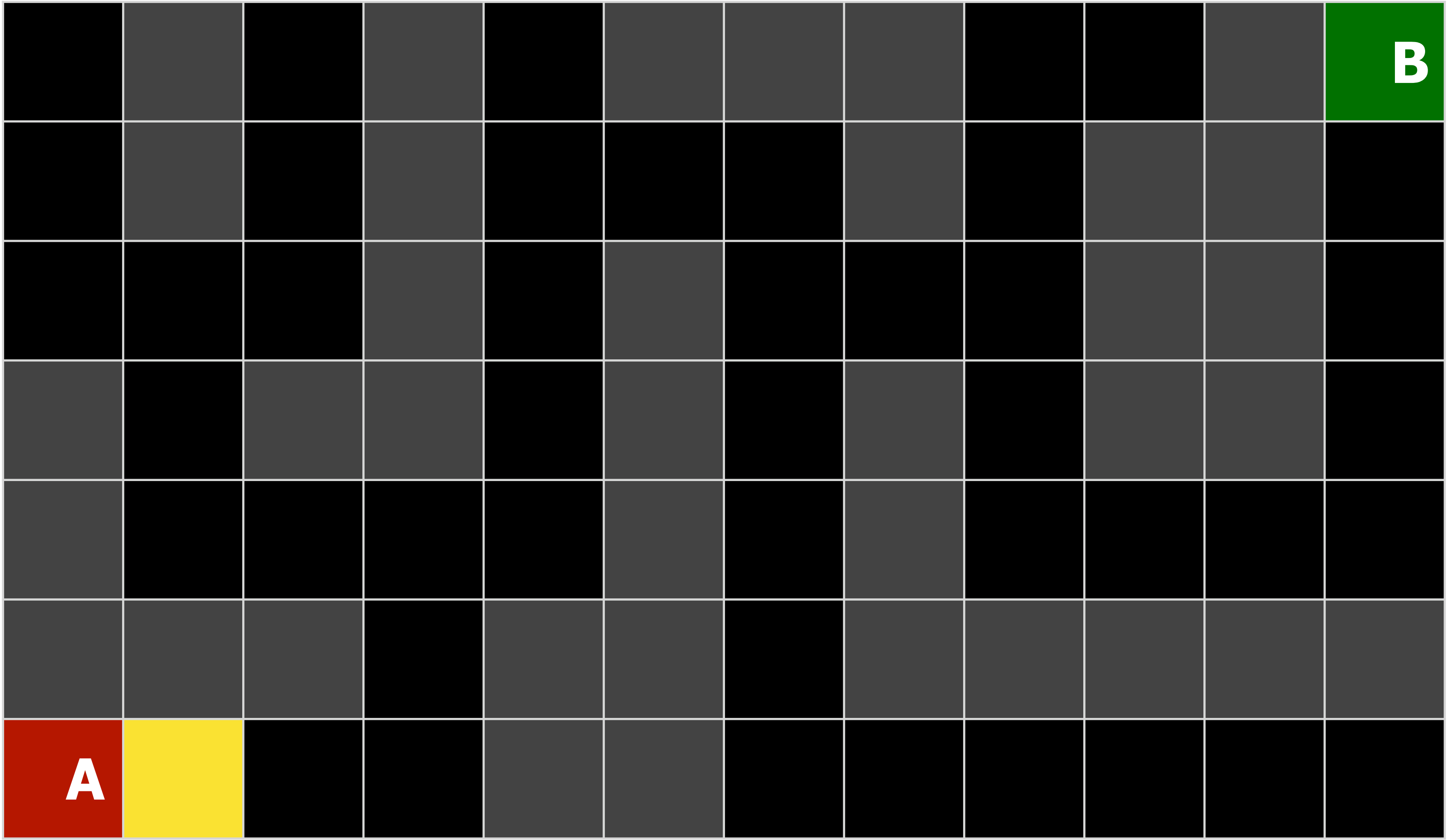
Breadth-First Search



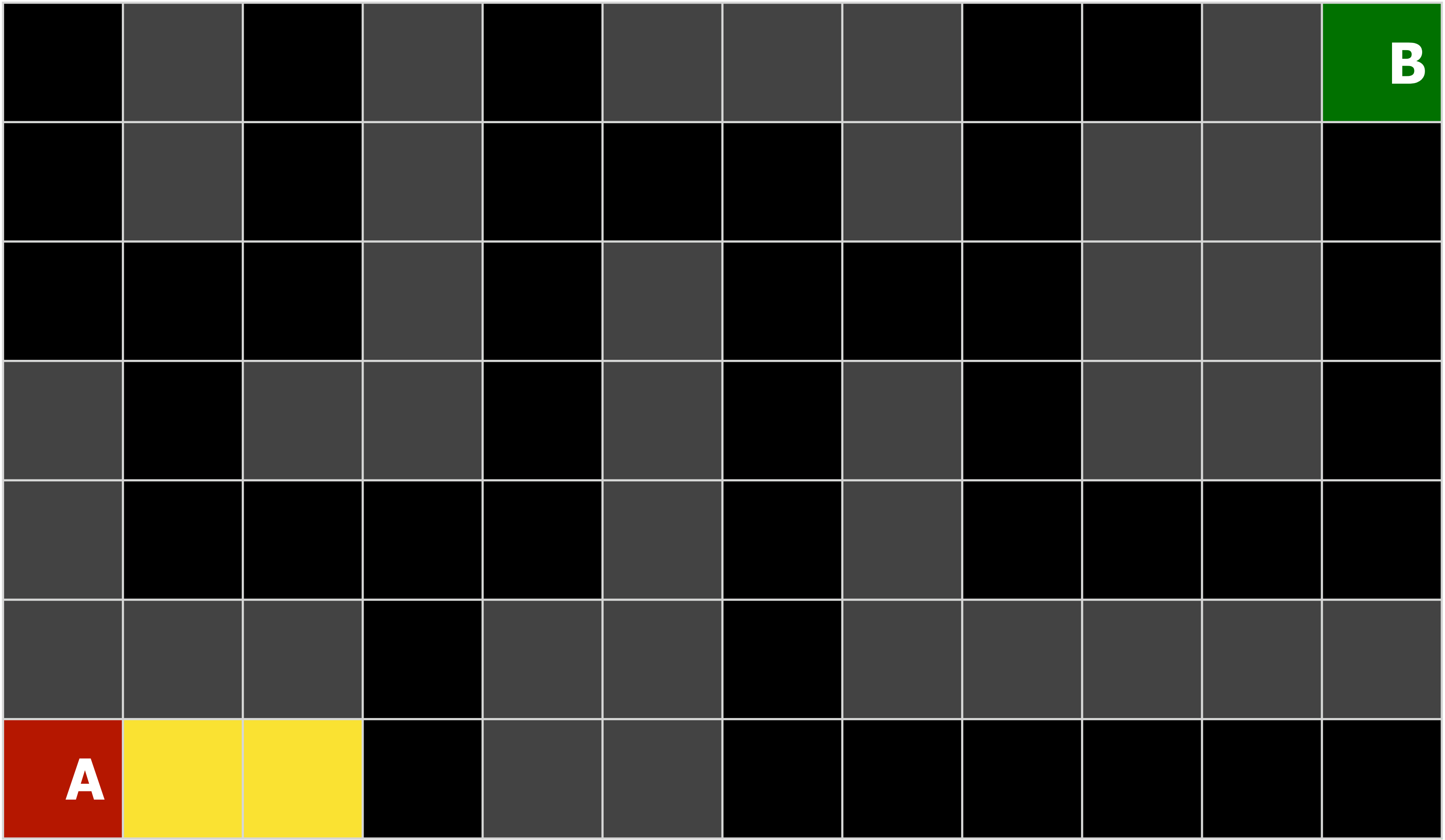
Breadth-First Search



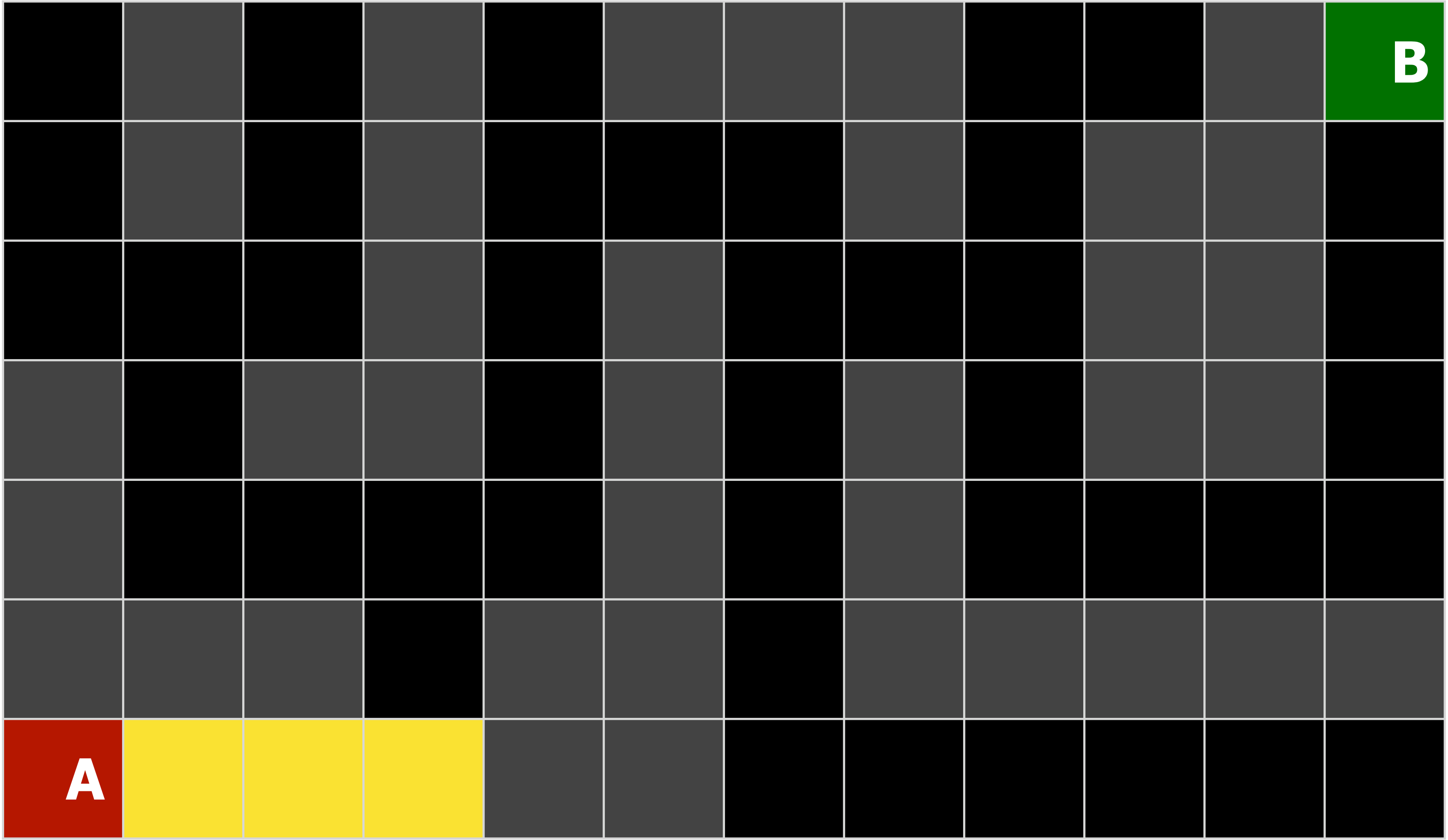
Breadth-First Search



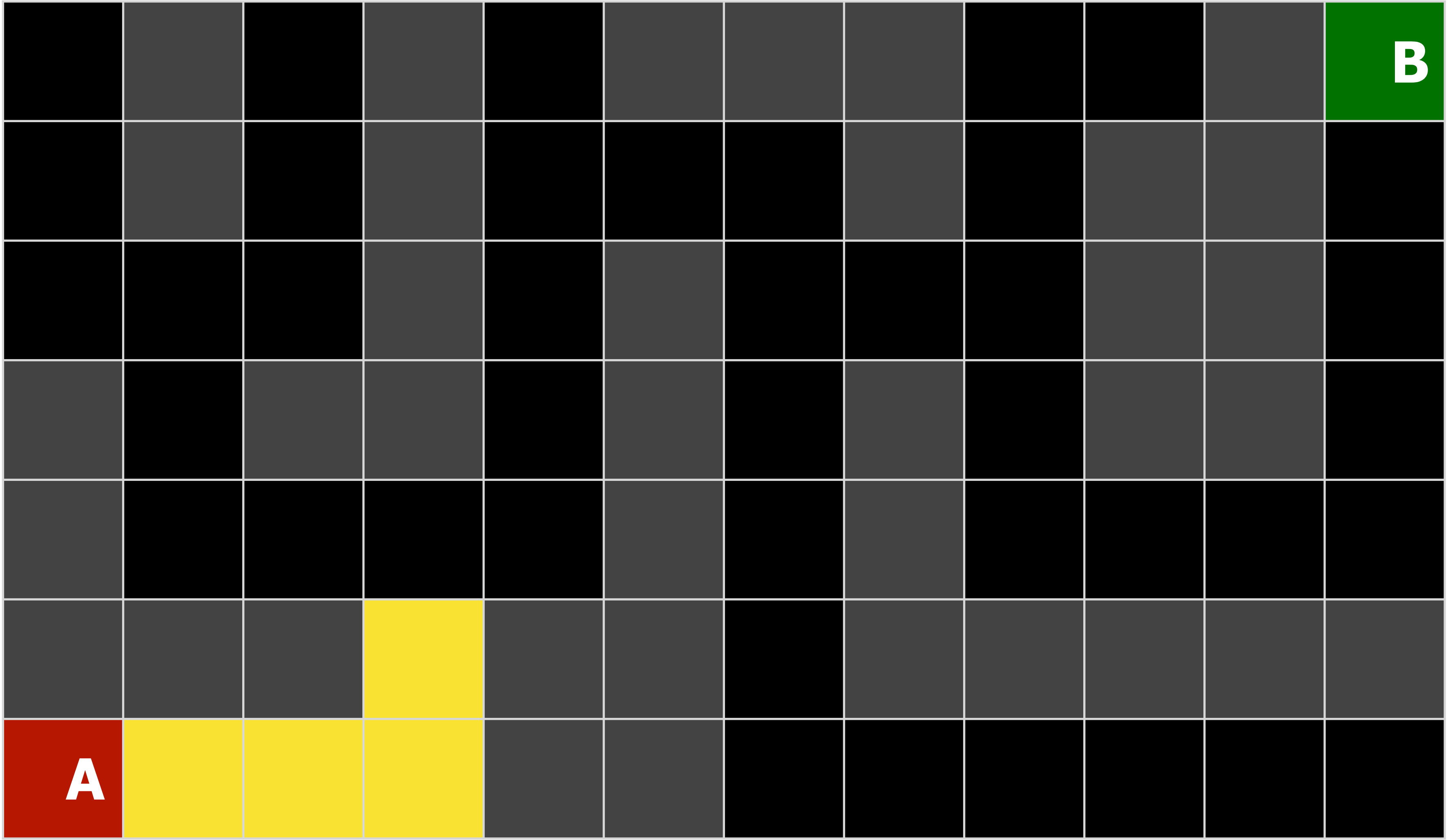
Breadth-First Search



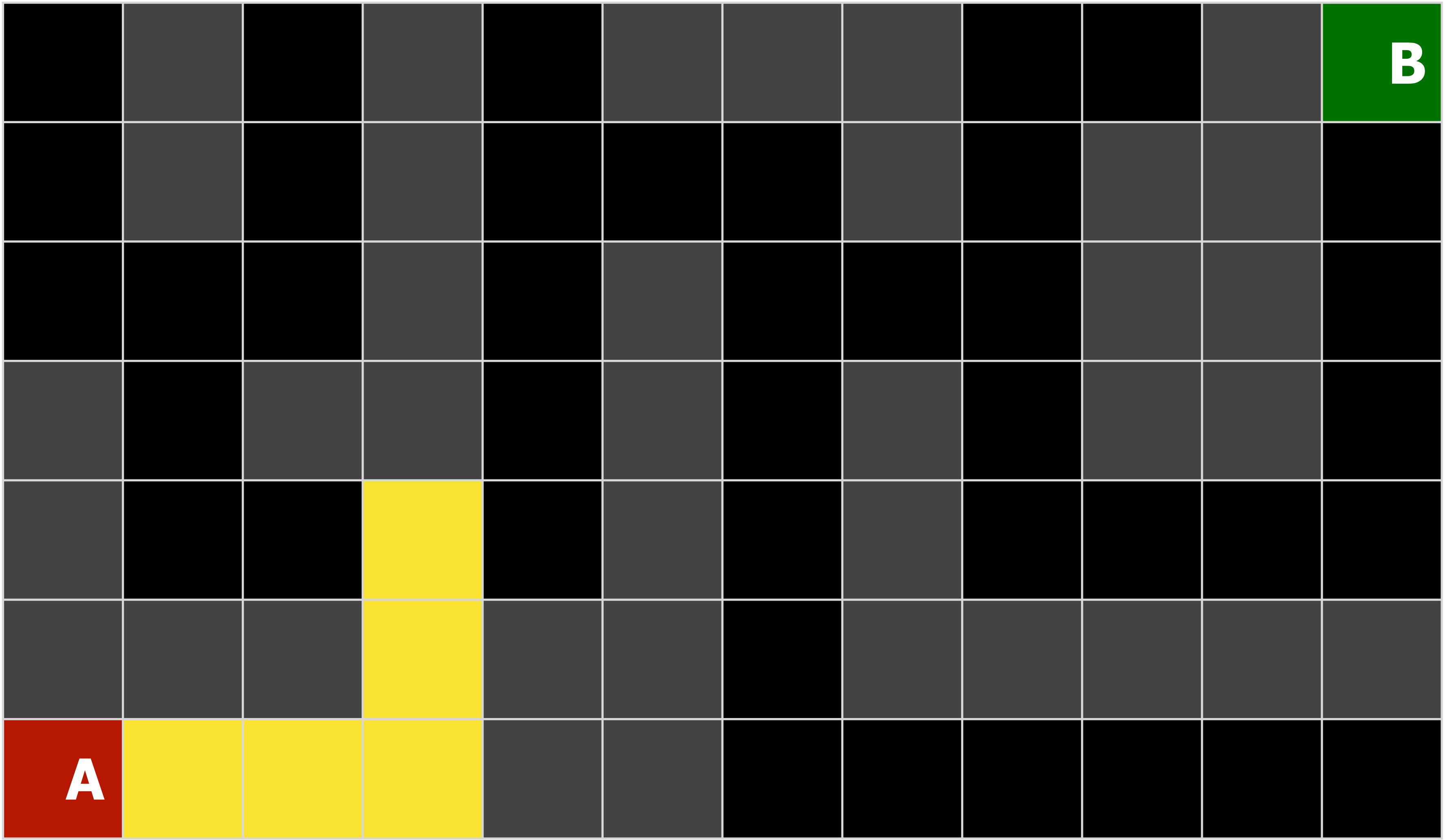
Breadth-First Search



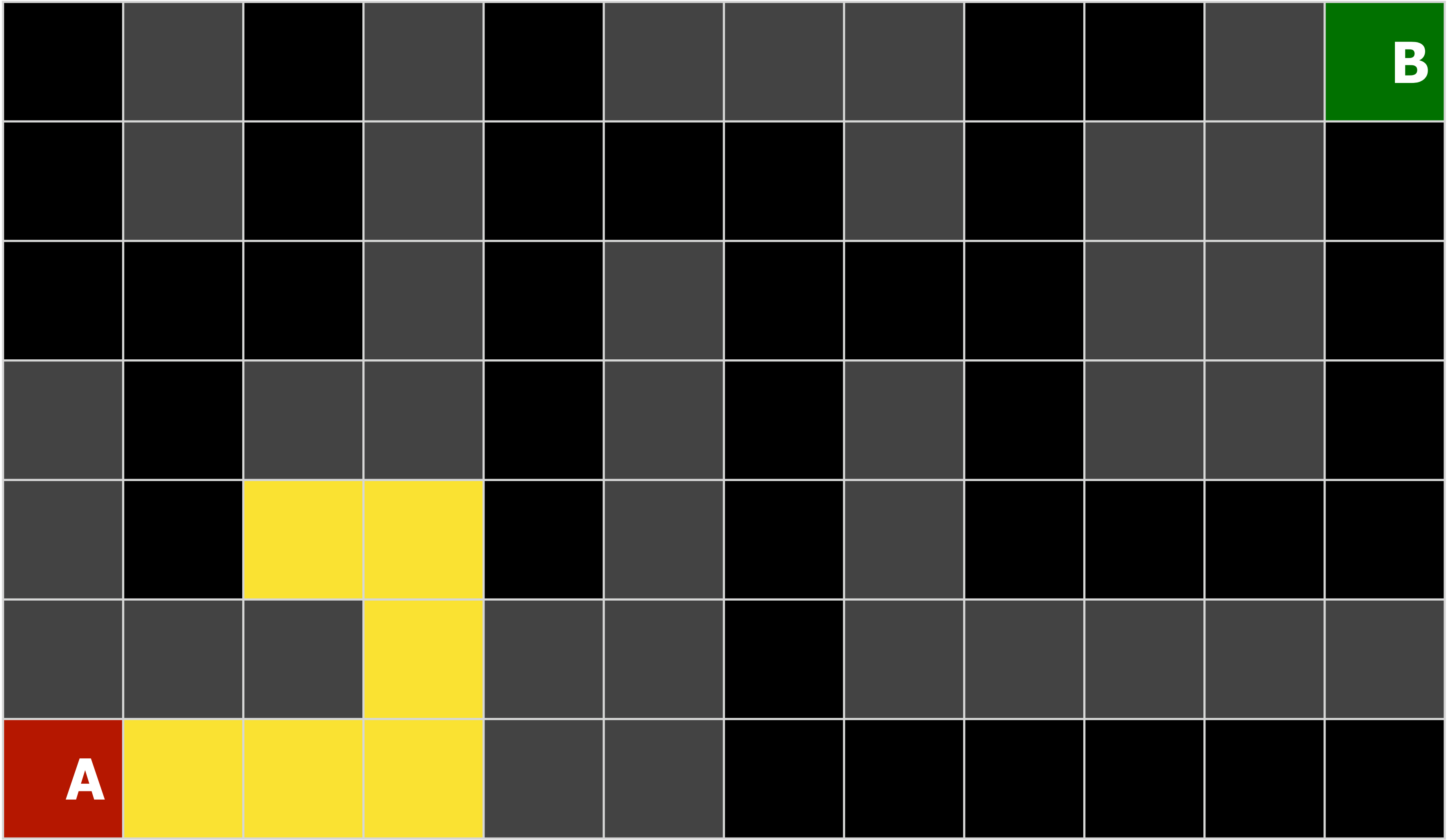
Breadth-First Search



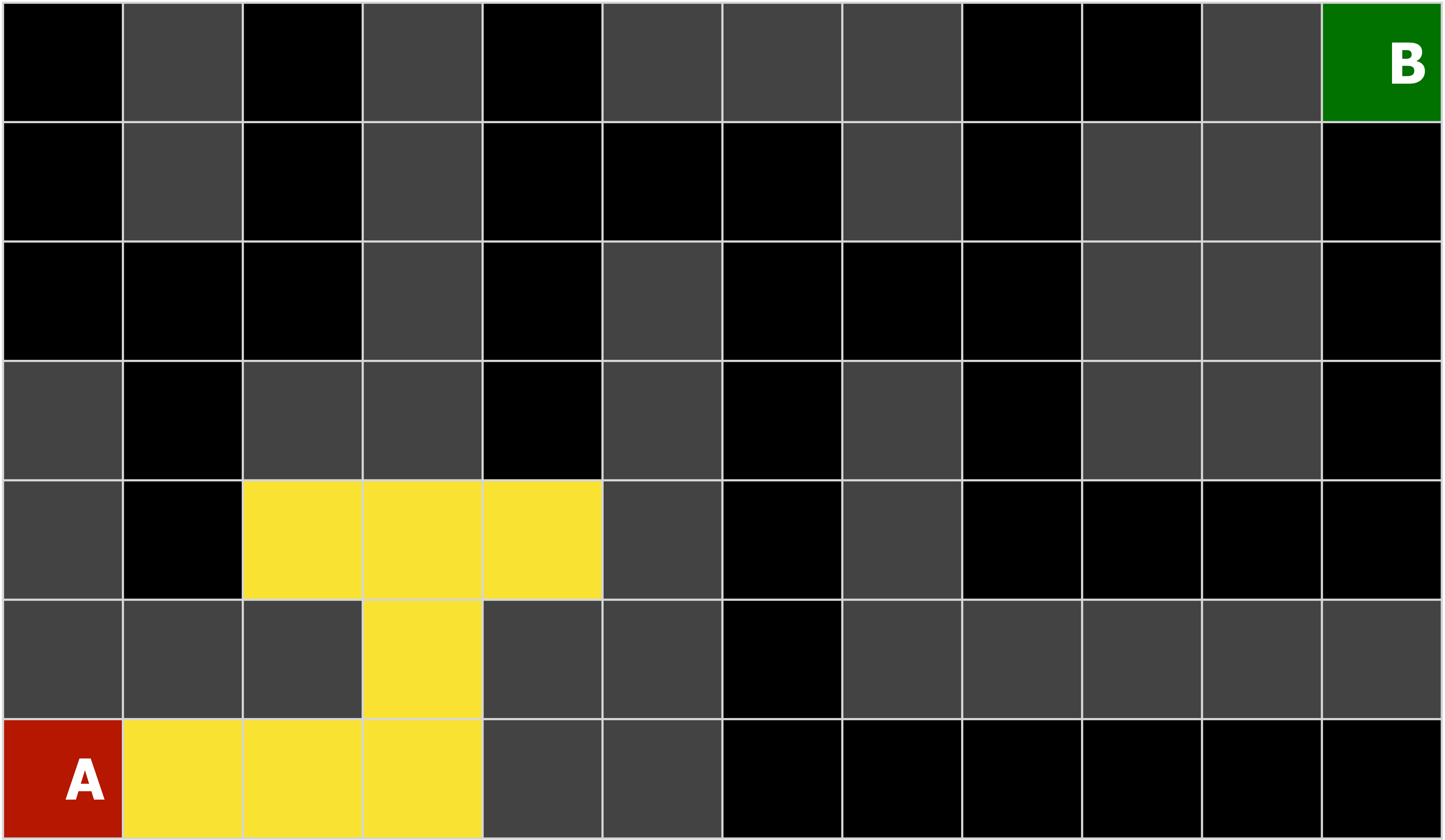
Breadth-First Search



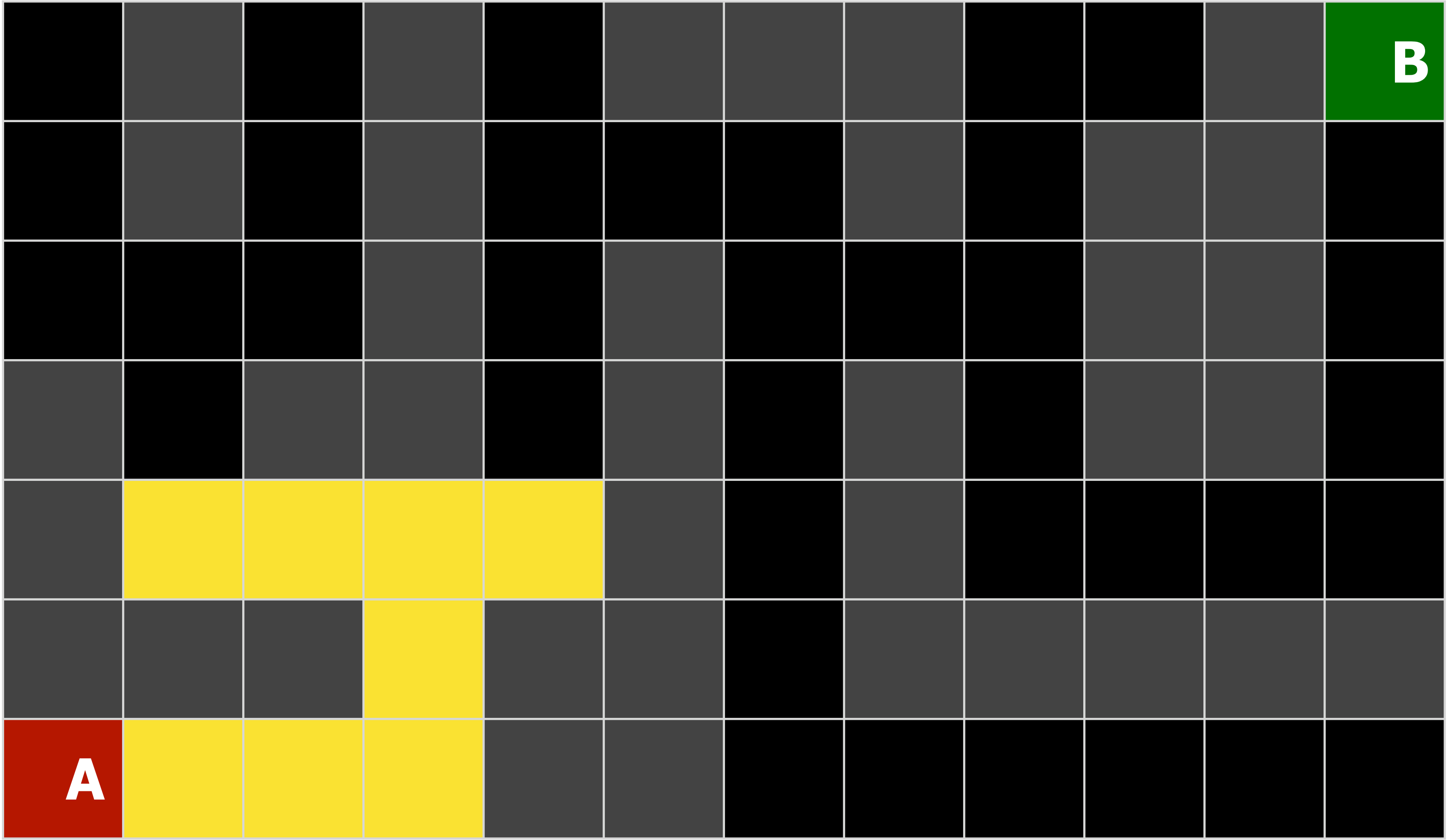
Breadth-First Search



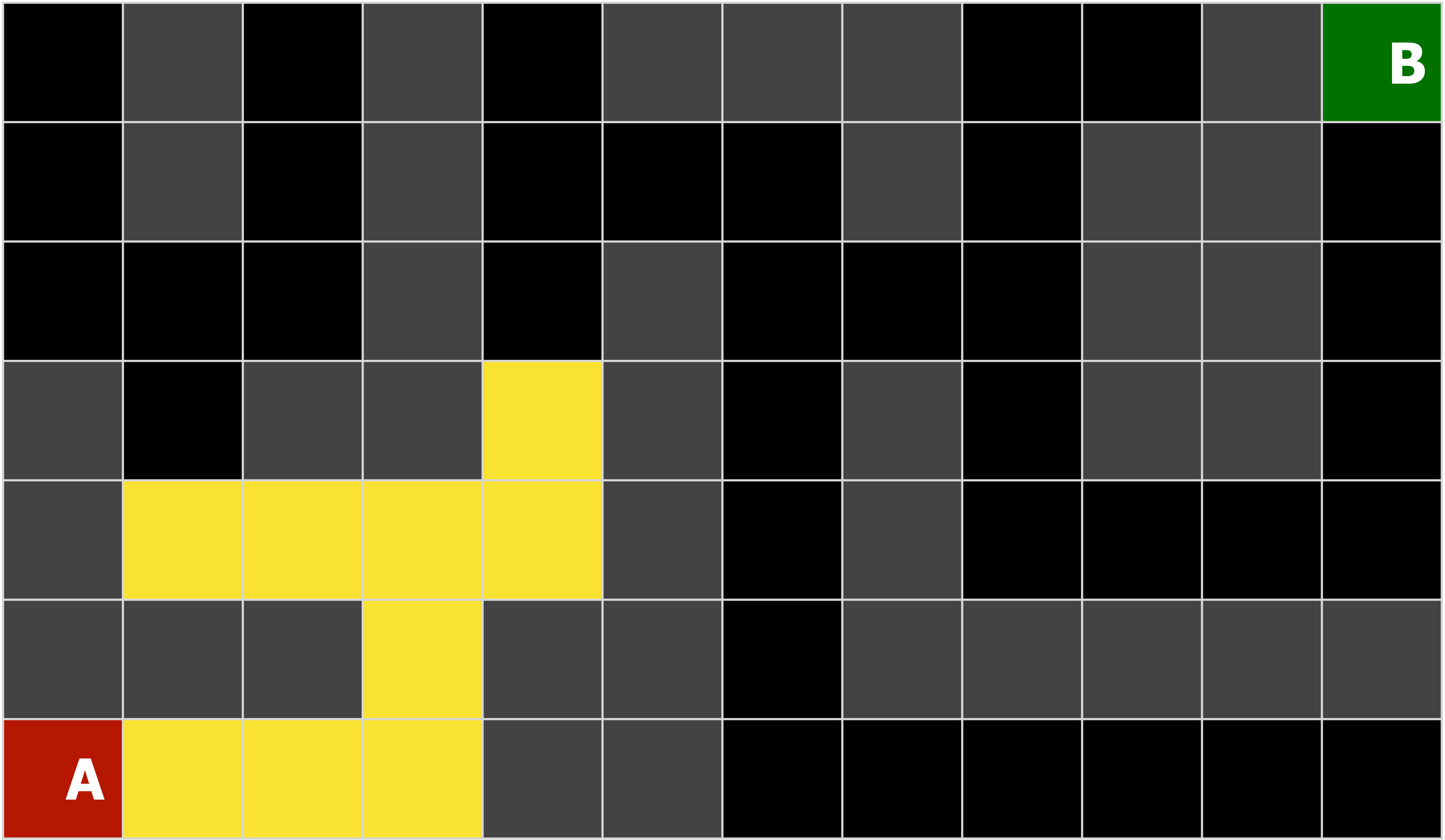
Breadth-First Search



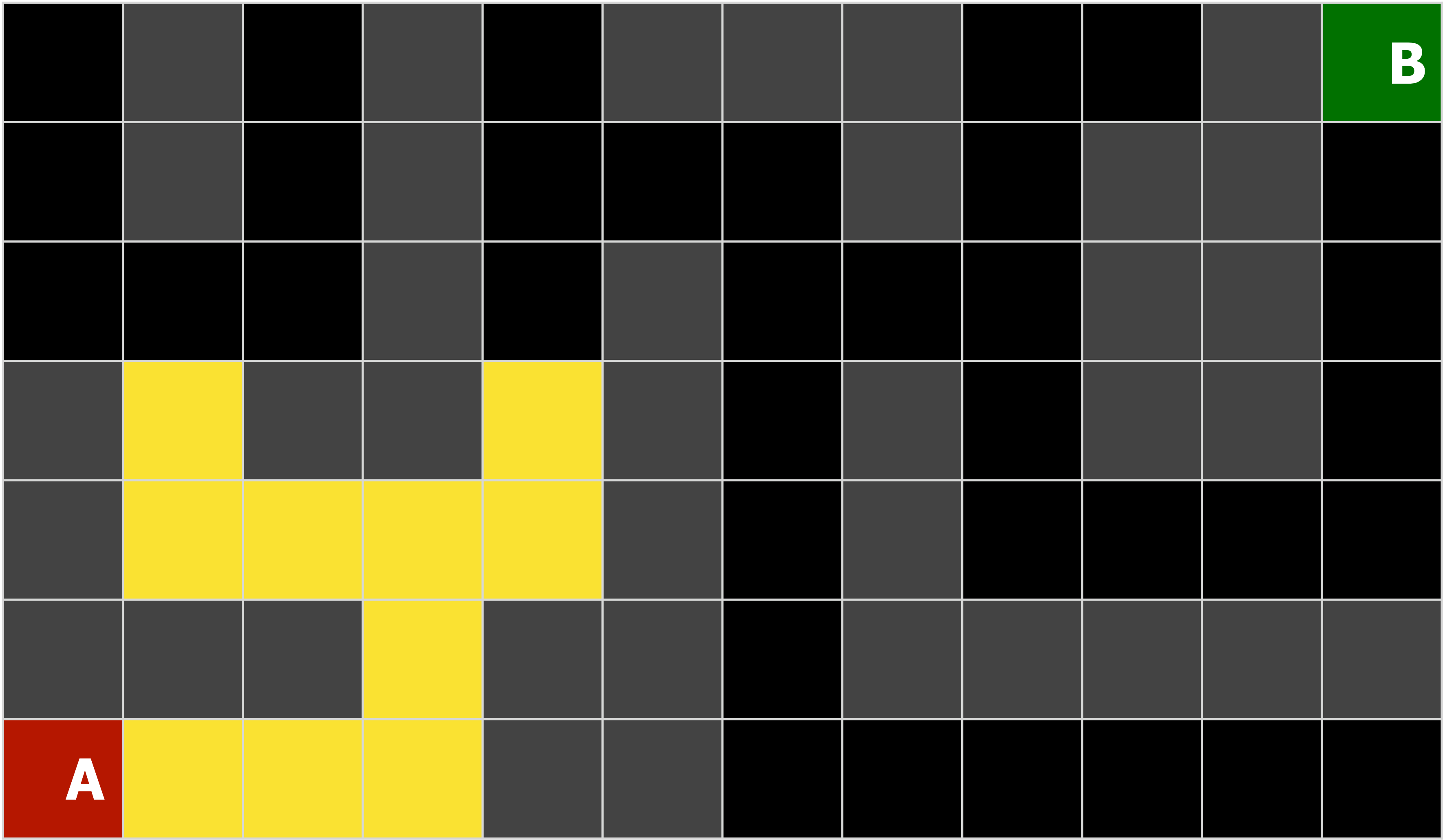
Breadth-First Search



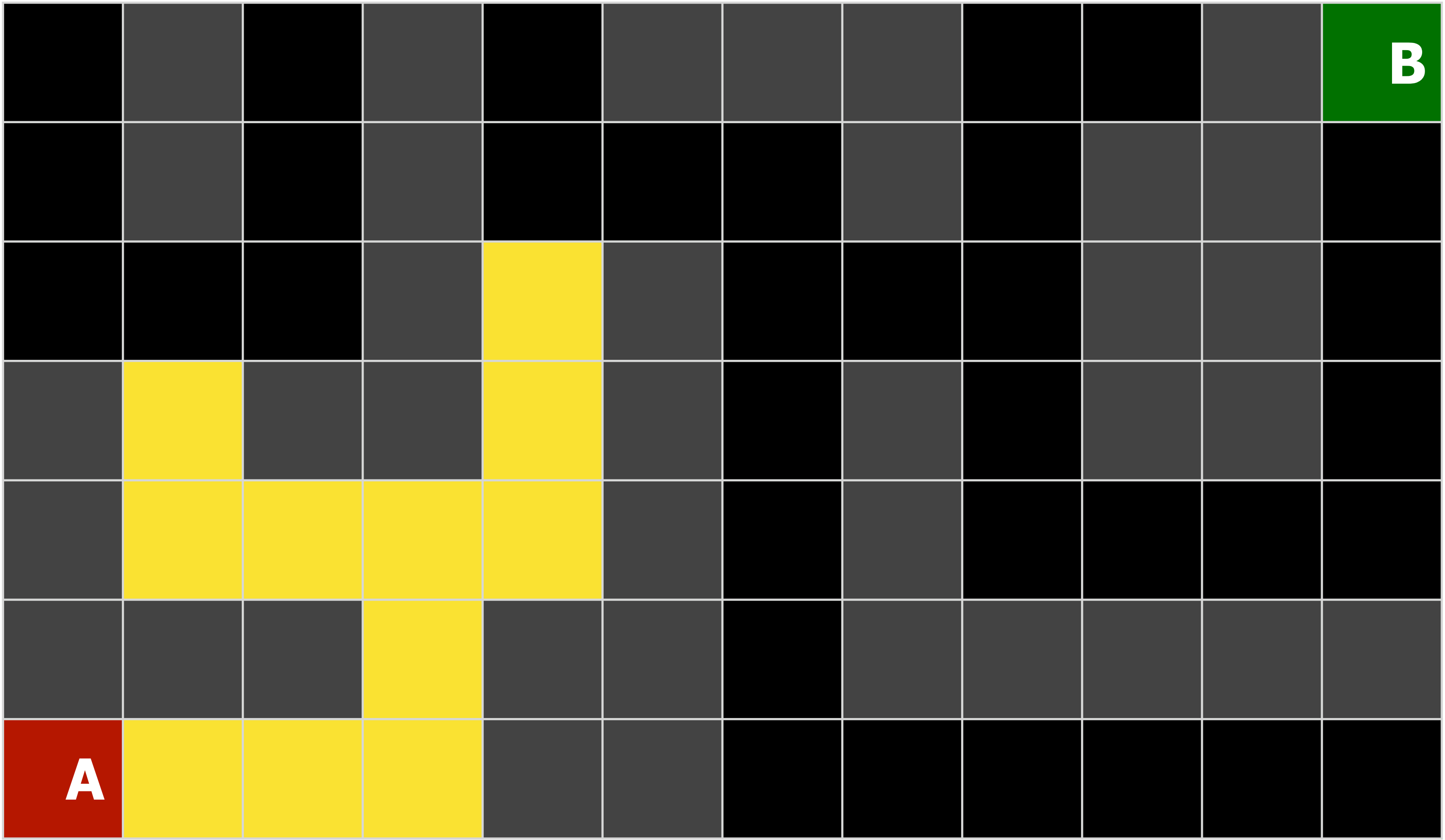
Breadth-First Search



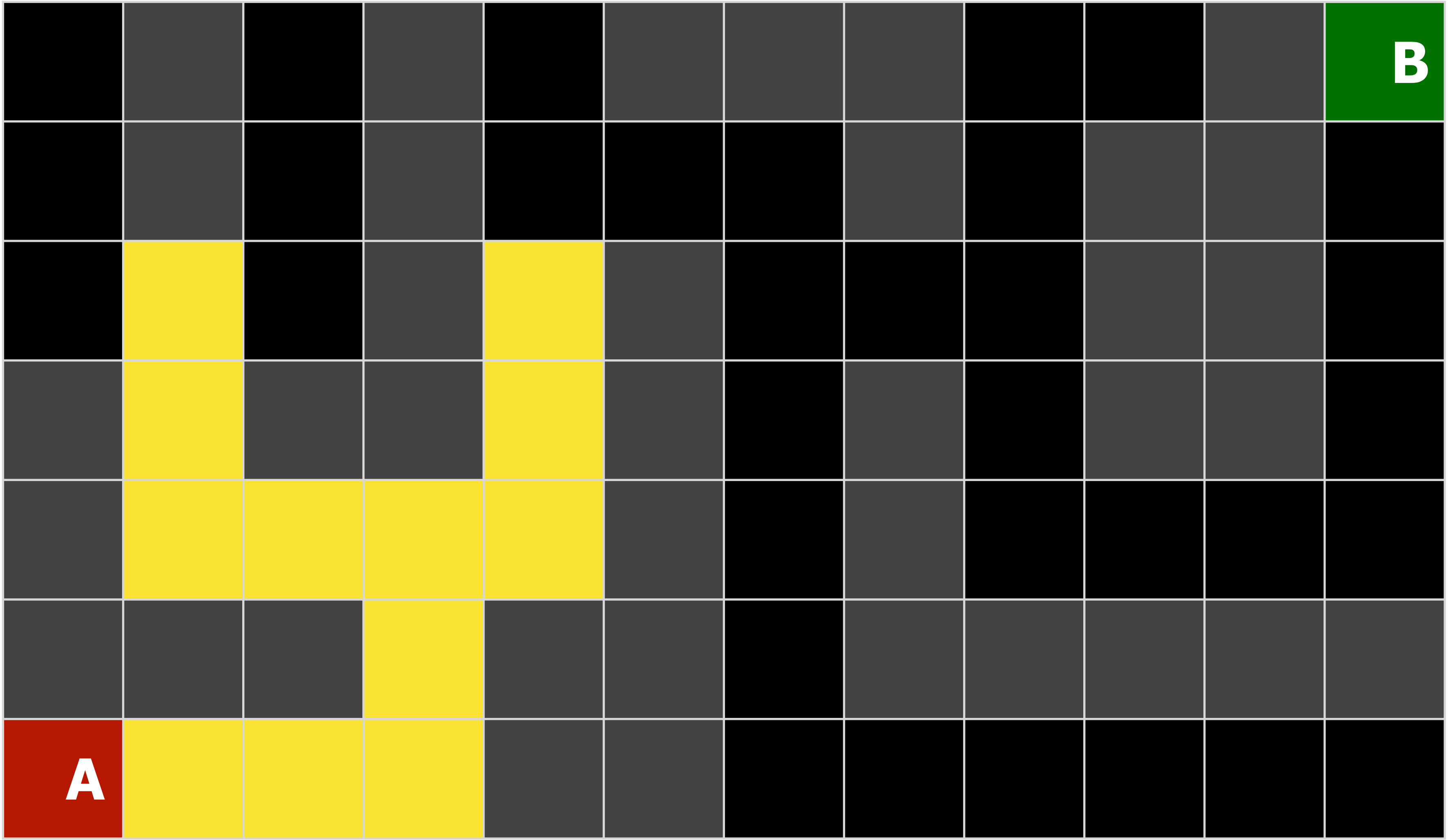
Breadth-First Search



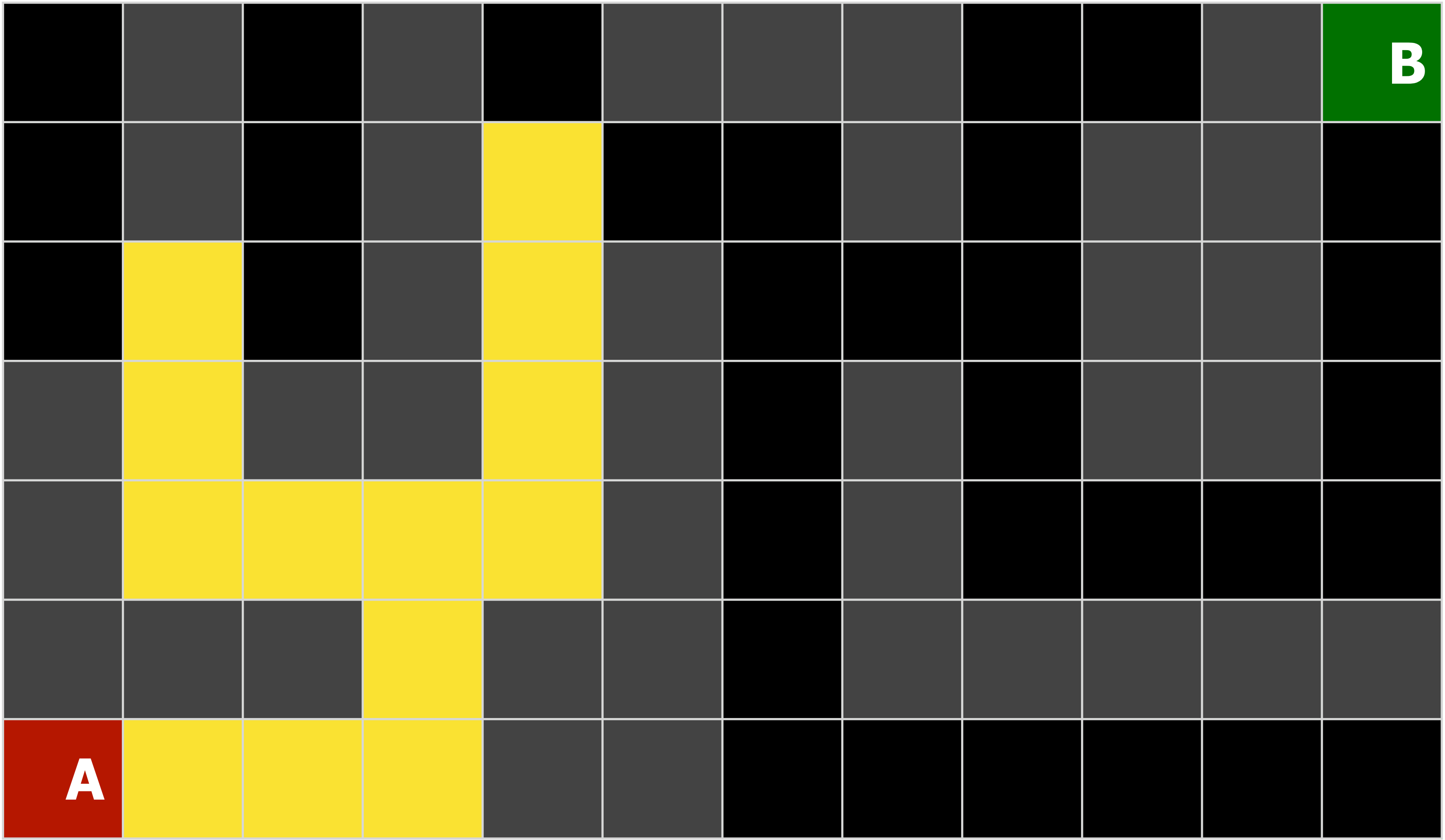
Breadth-First Search



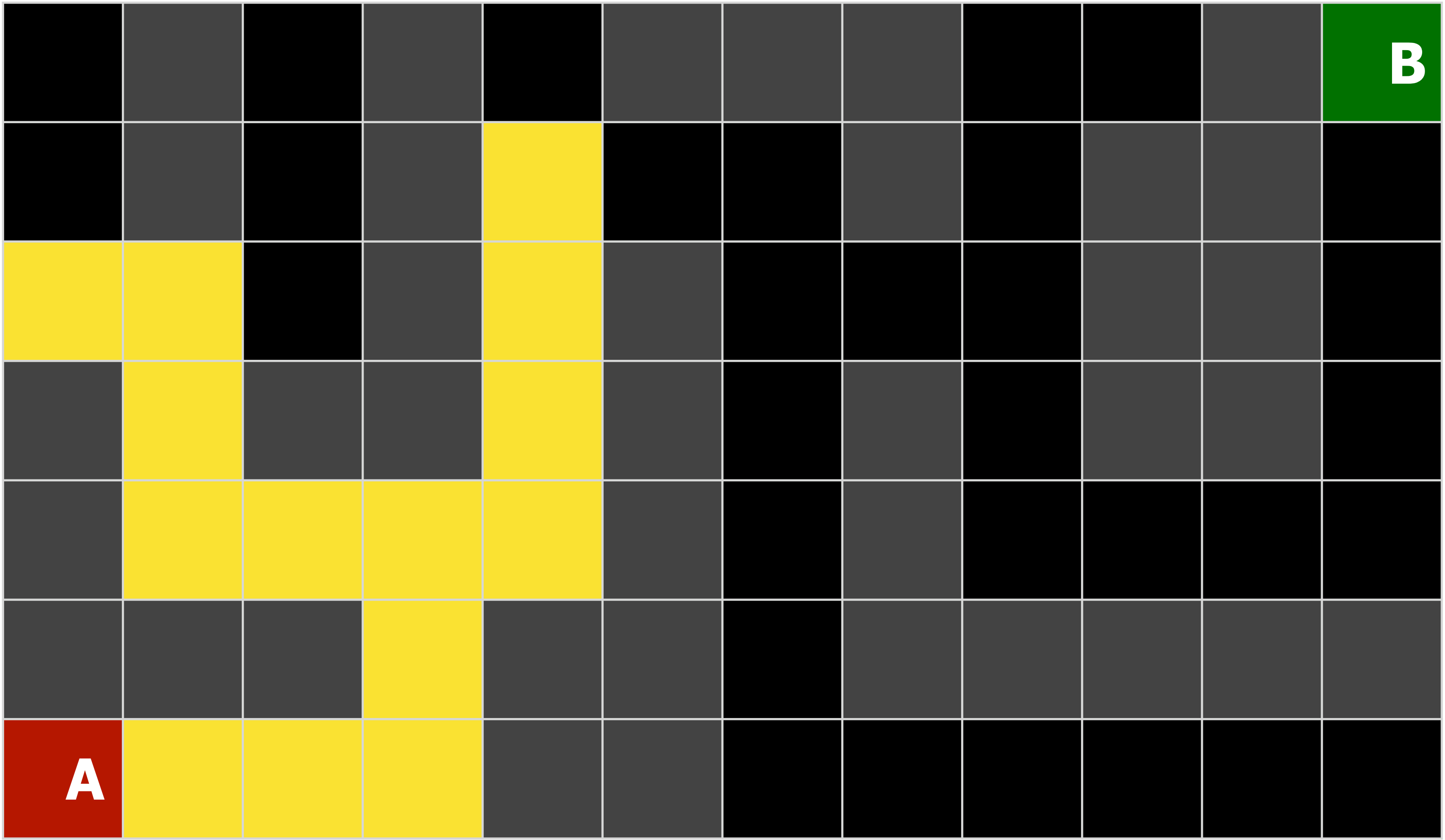
Breadth-First Search



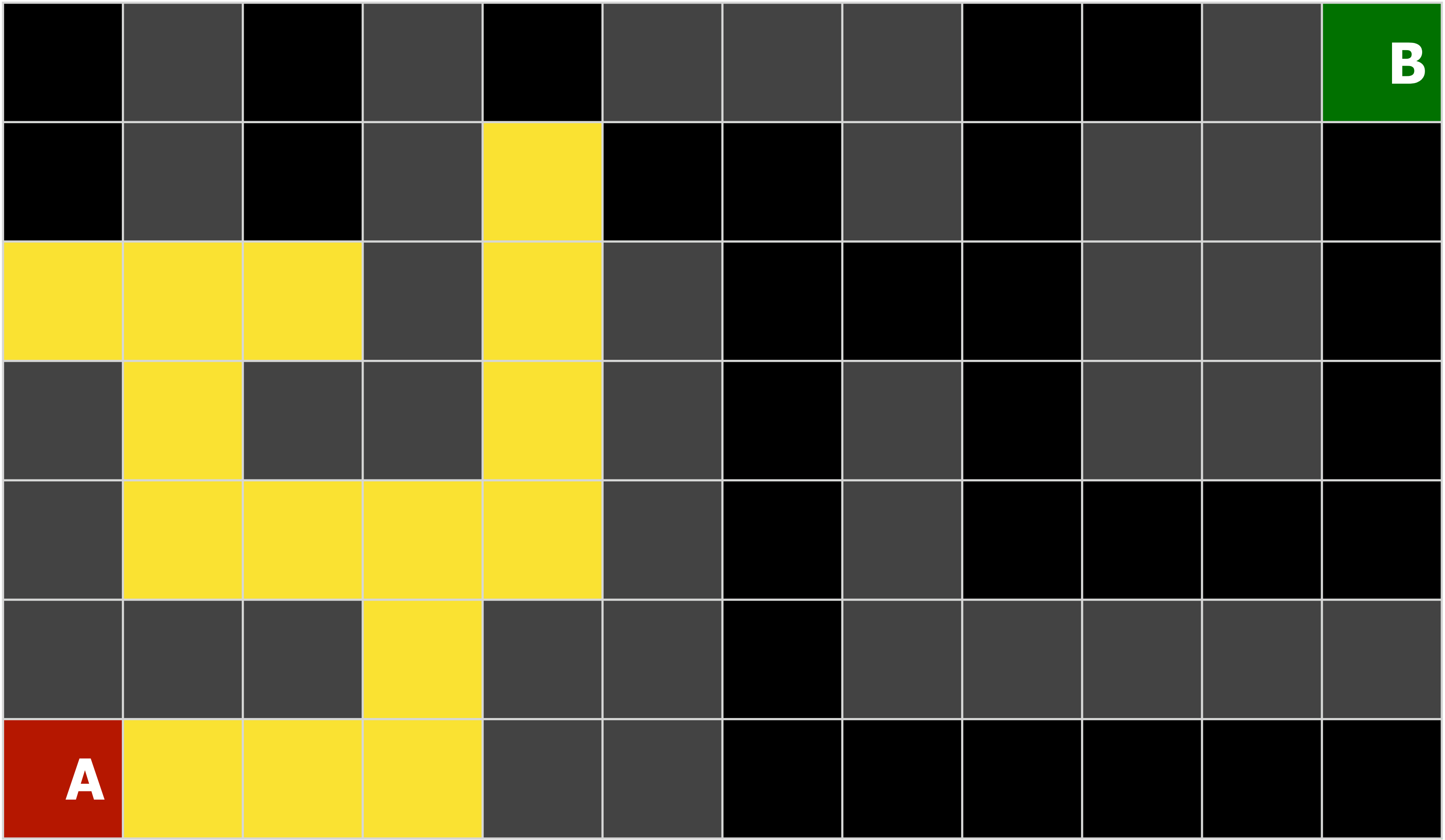
Breadth-First Search



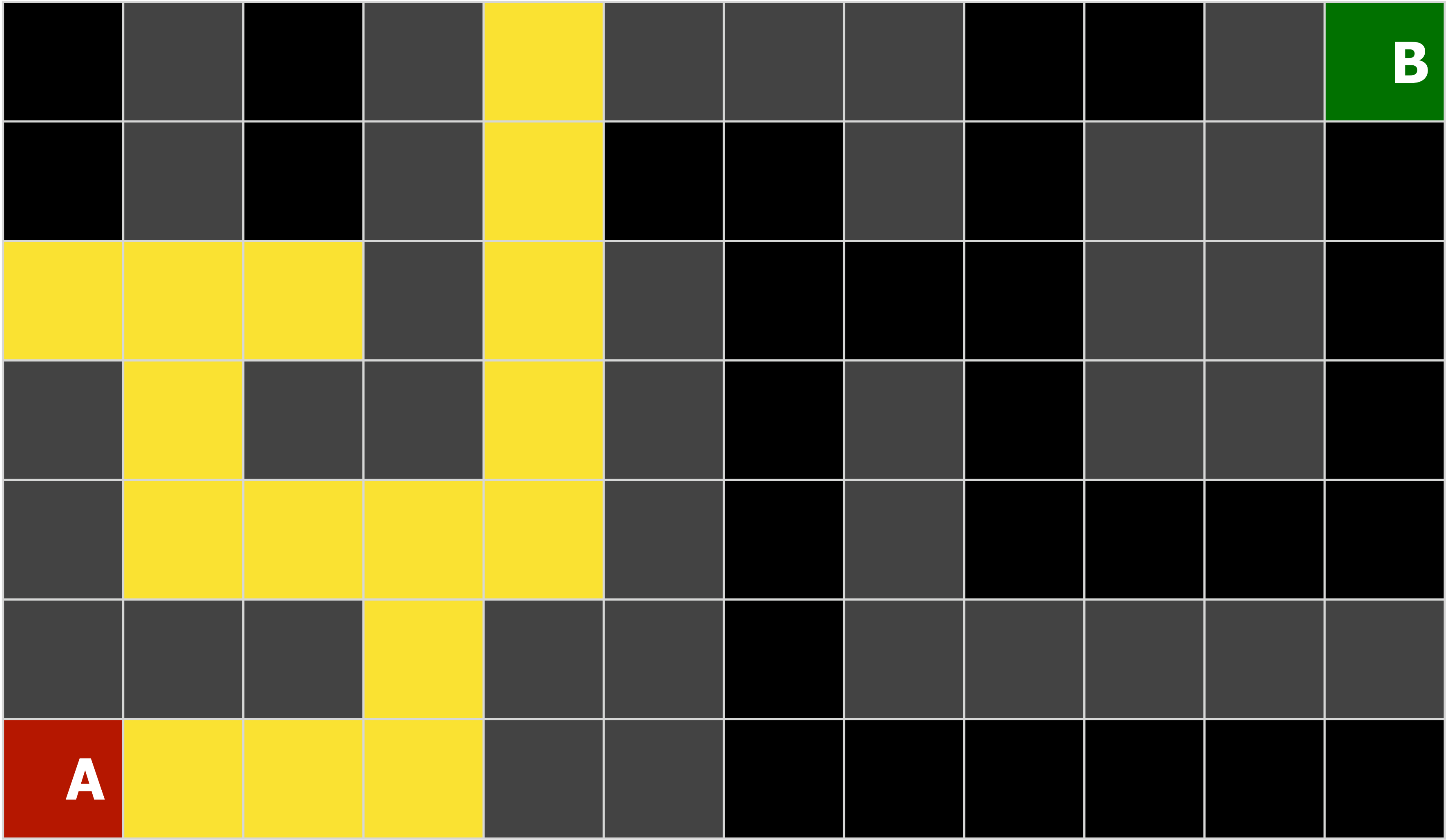
Breadth-First Search



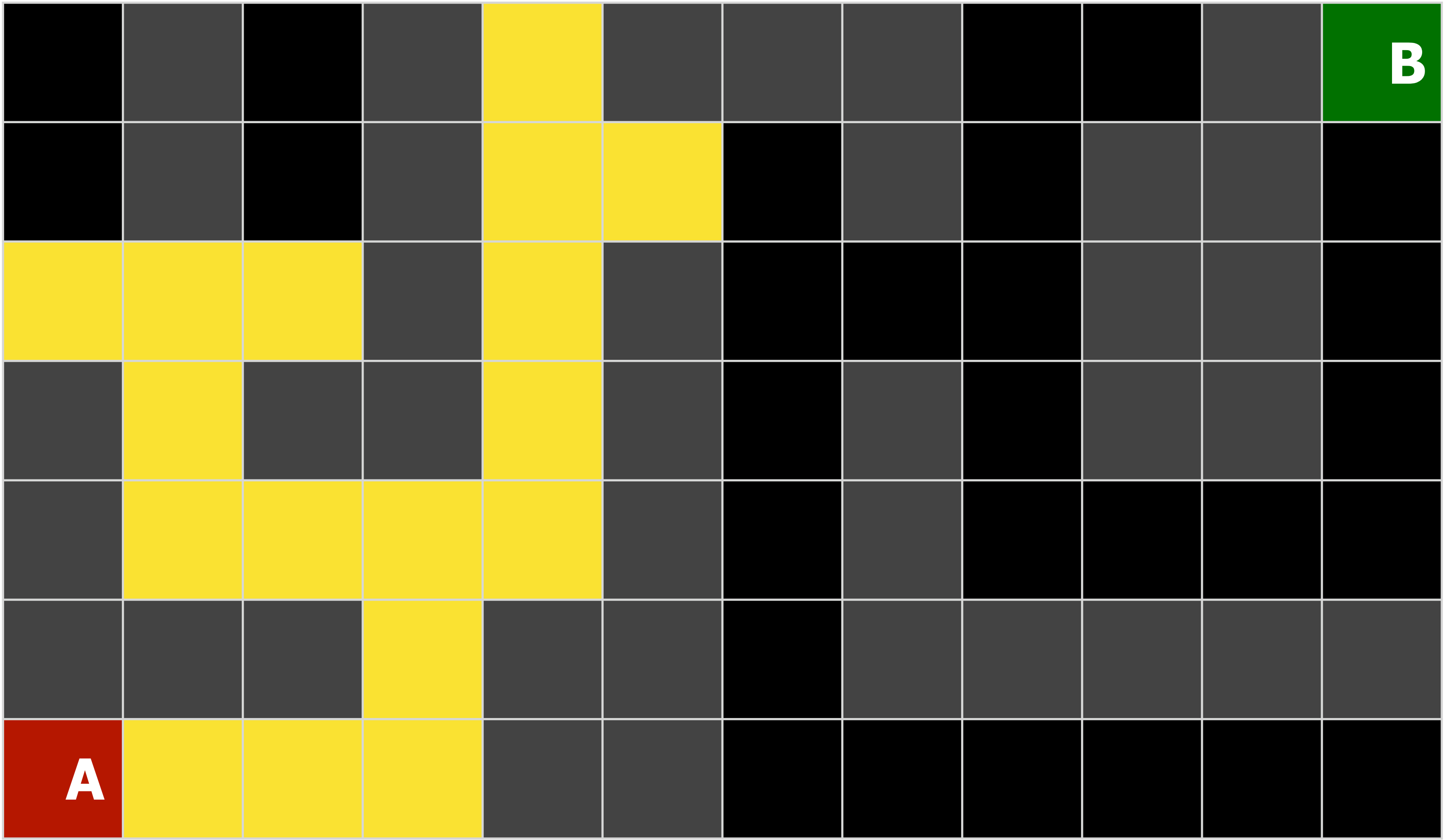
Breadth-First Search



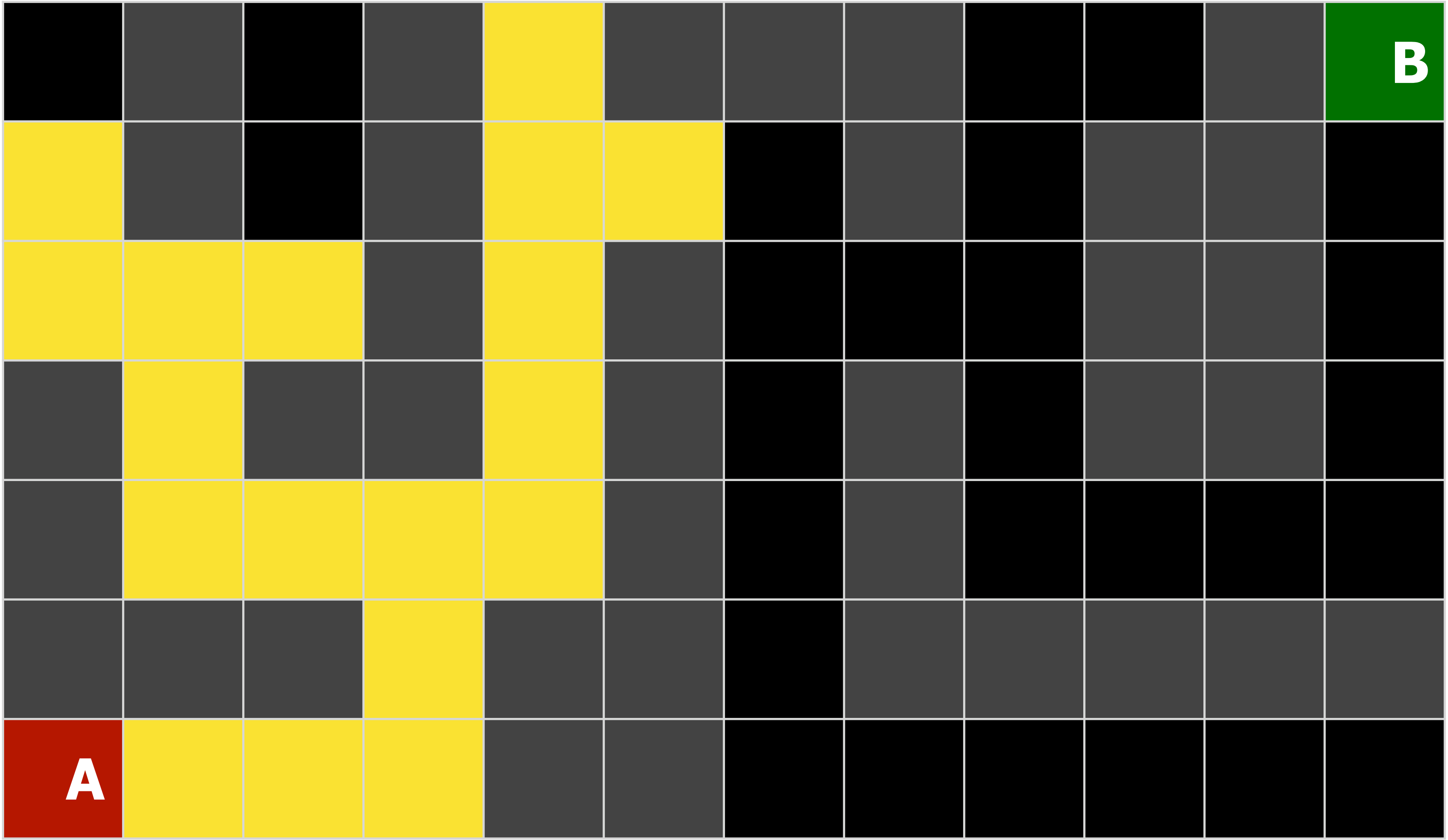
Breadth-First Search



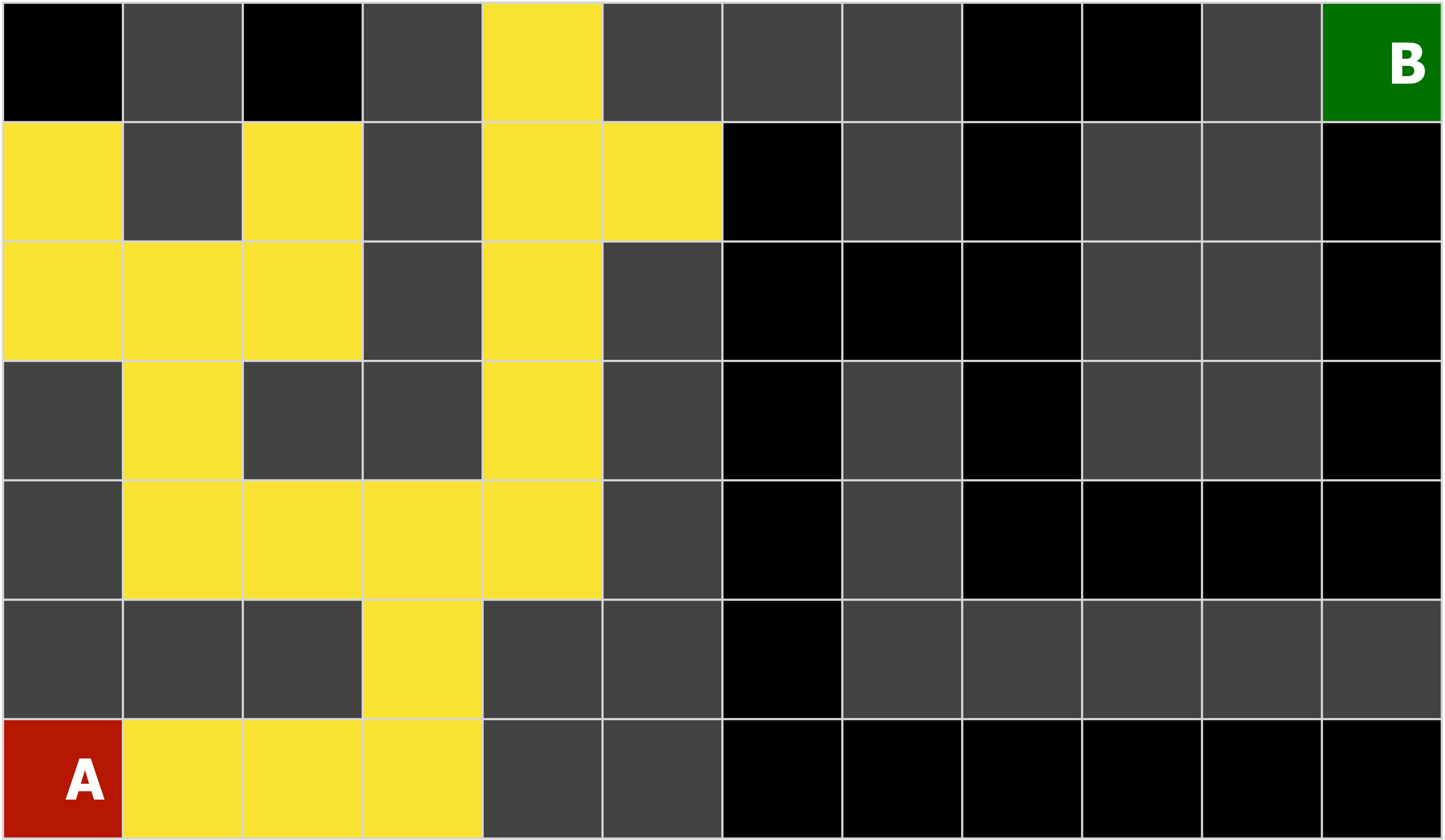
Breadth-First Search



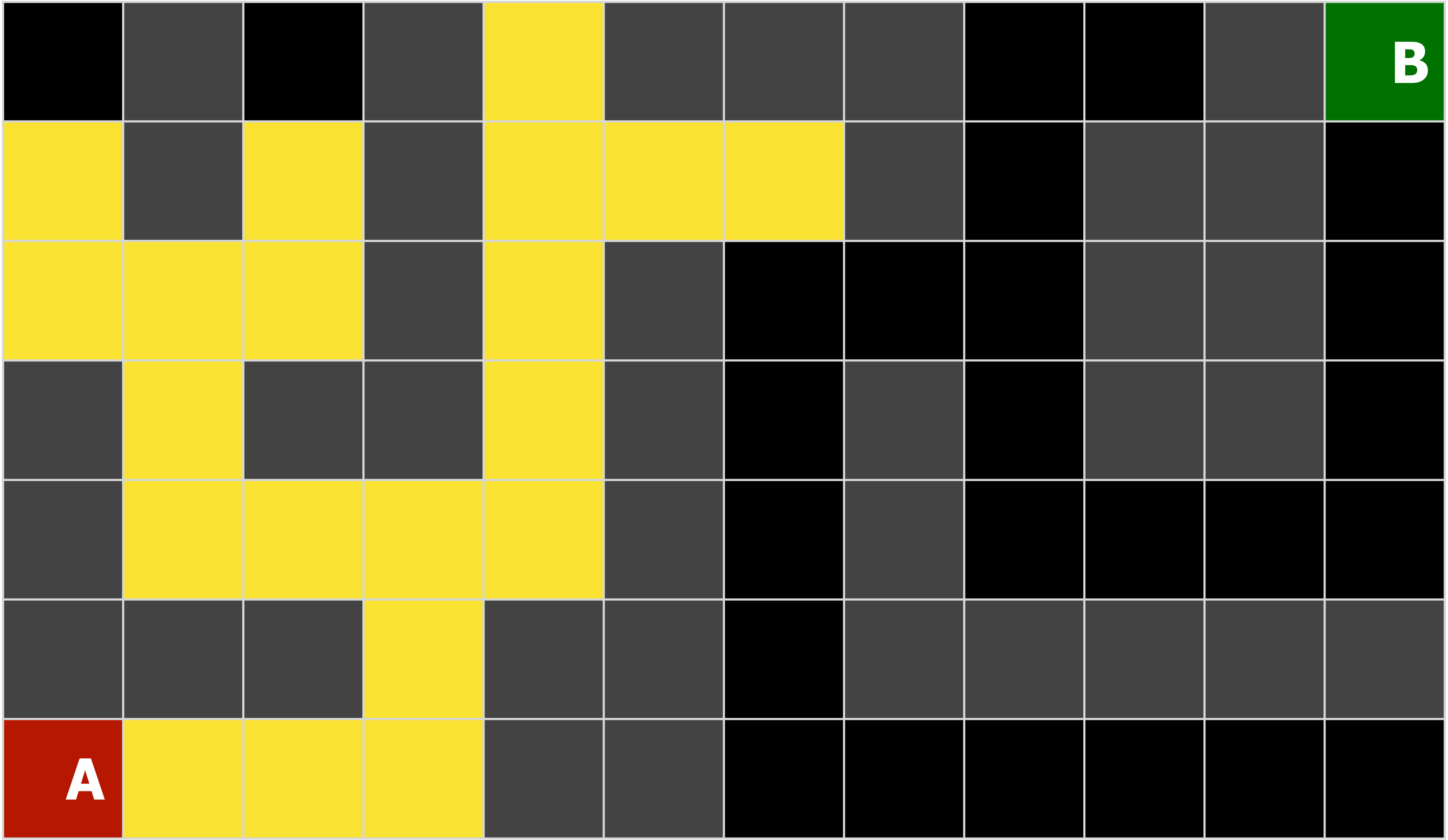
Breadth-First Search



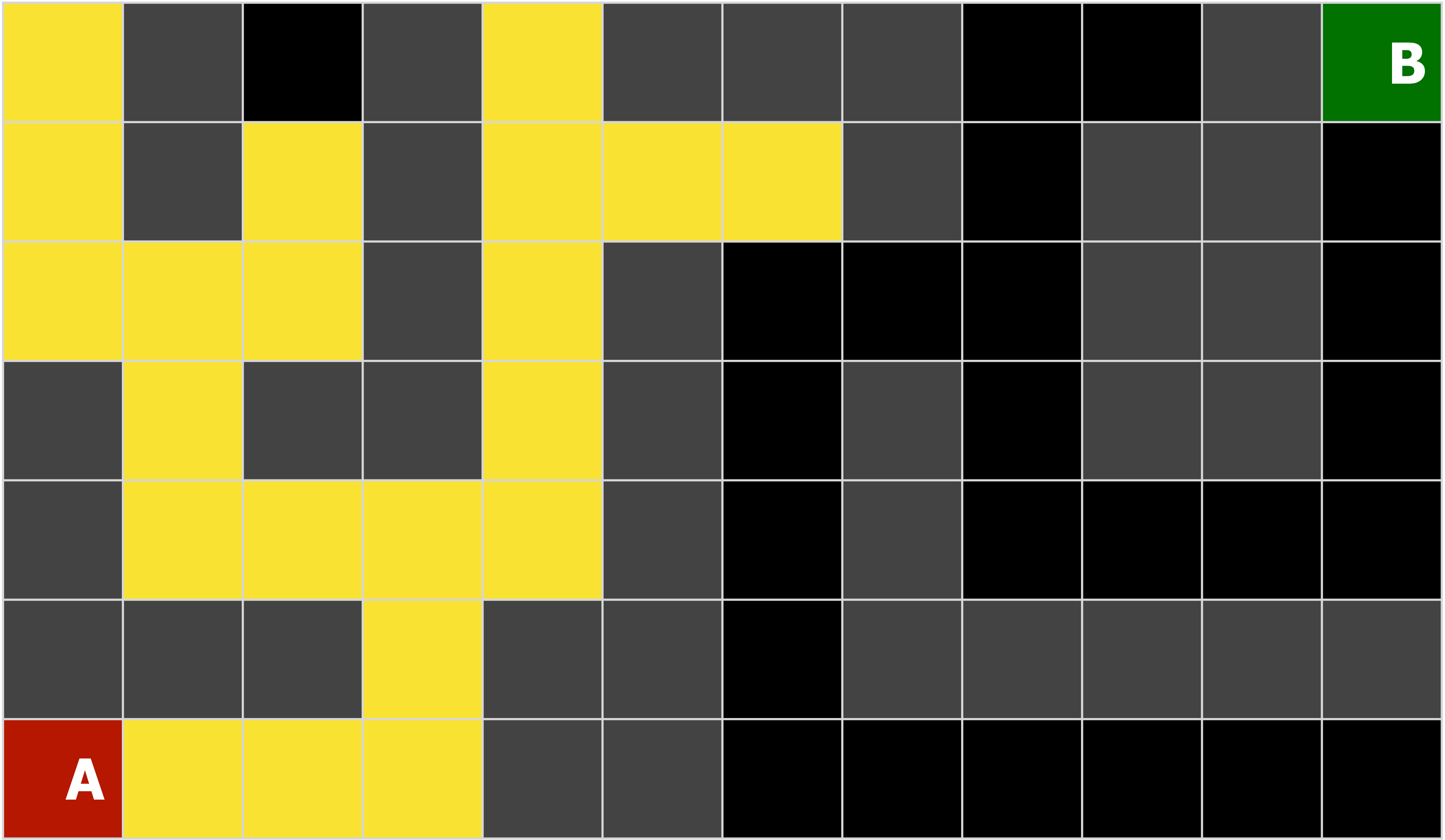
Breadth-First Search



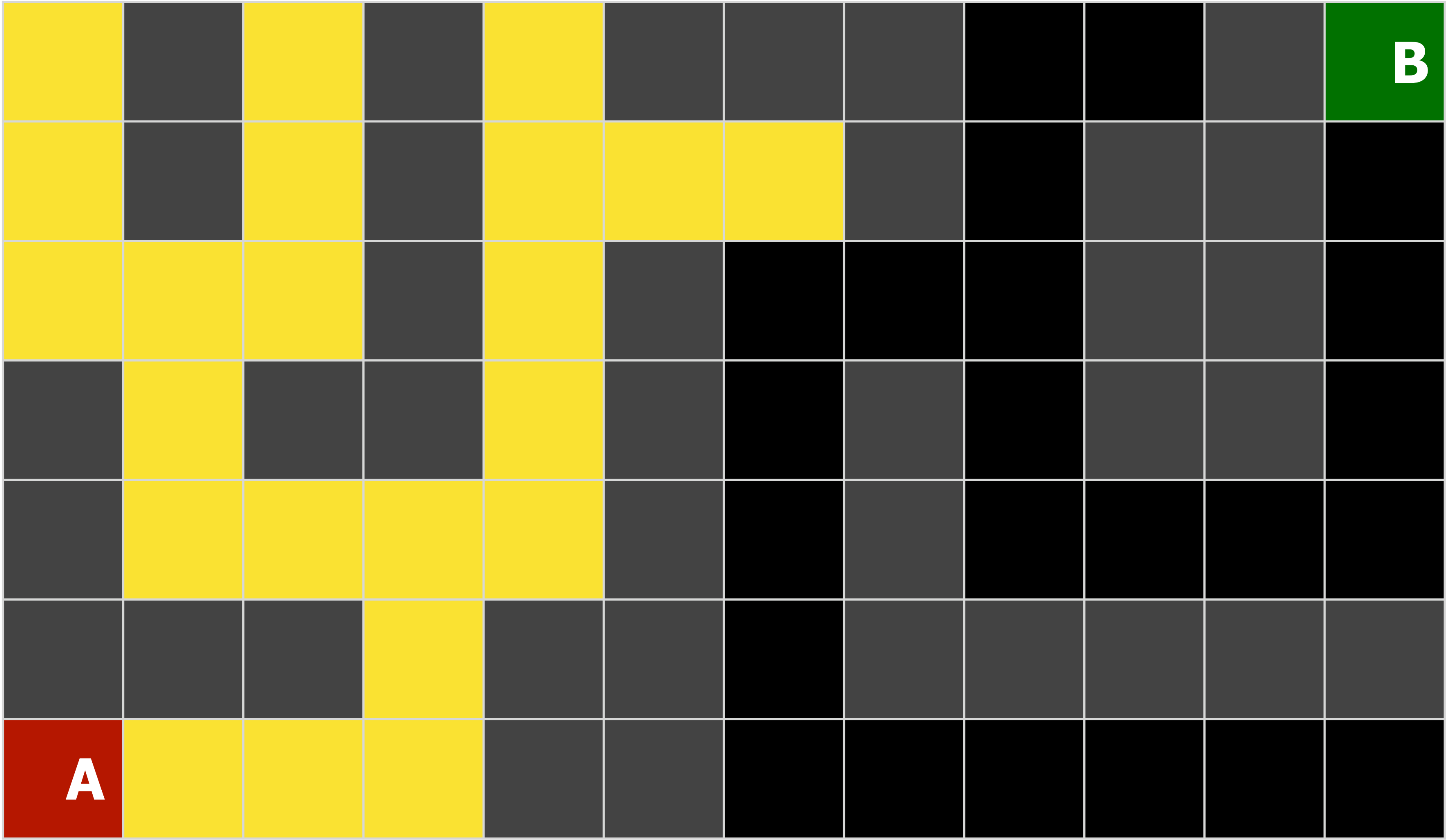
Breadth-First Search



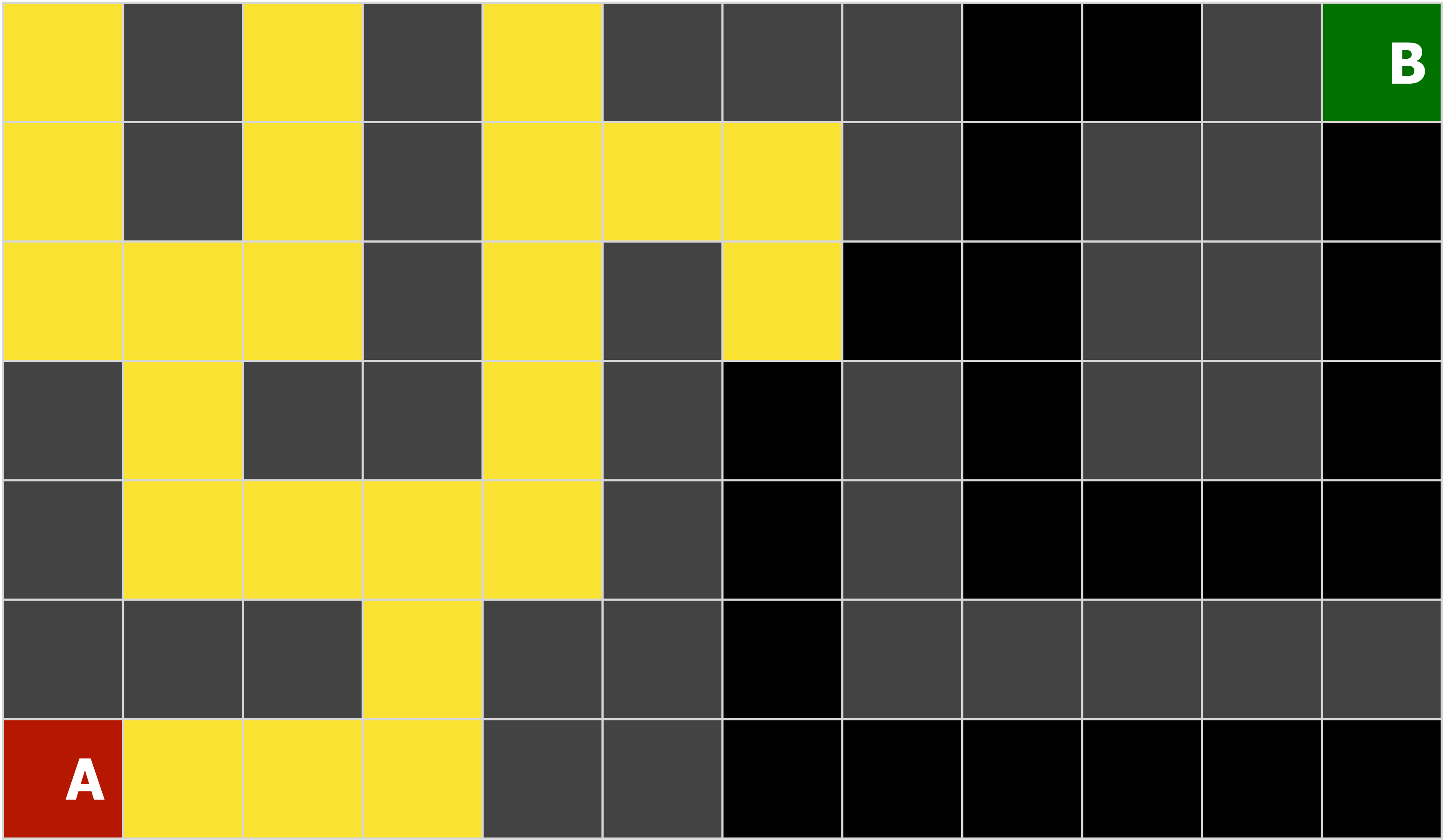
Breadth-First Search



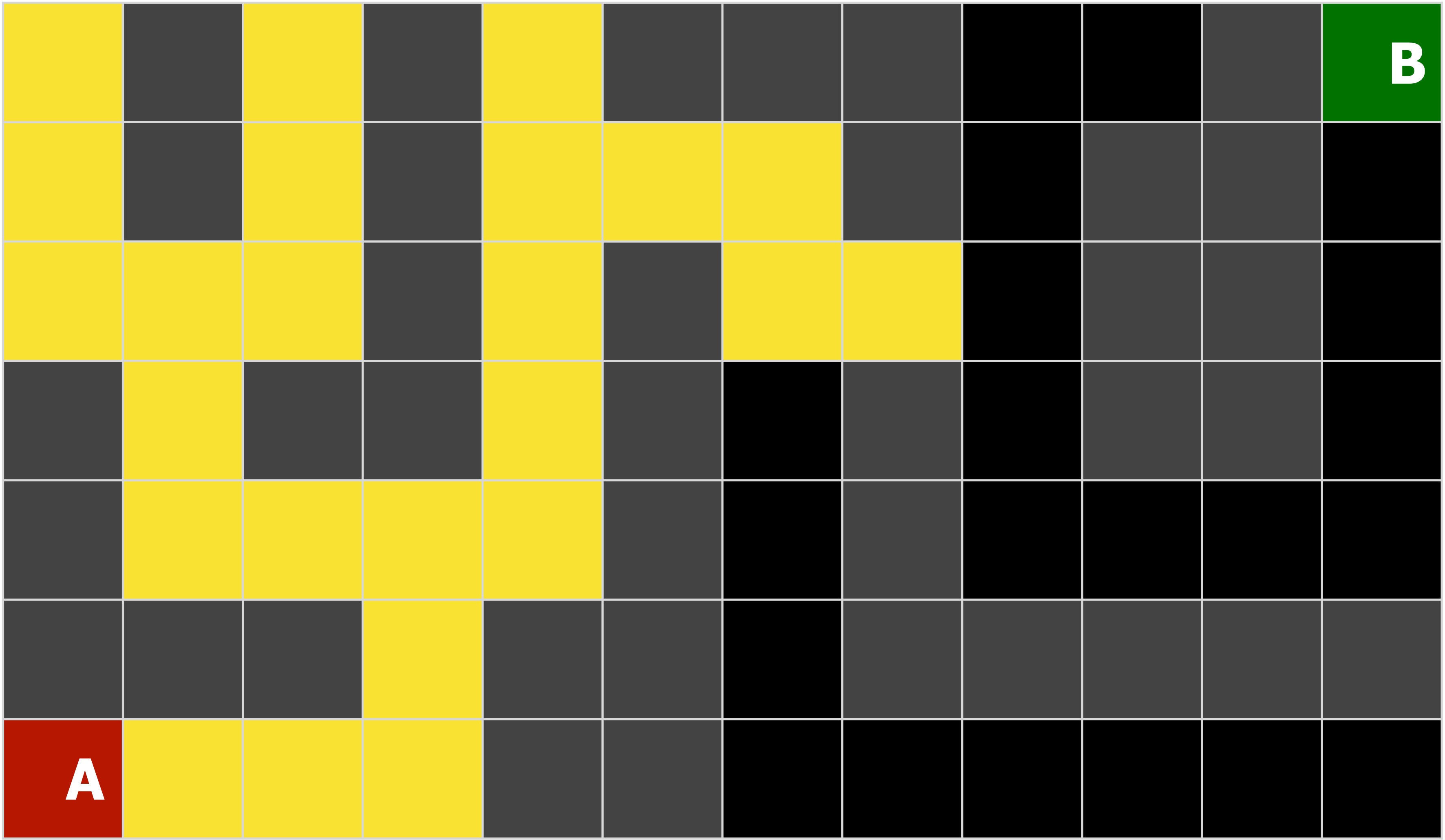
Breadth-First Search



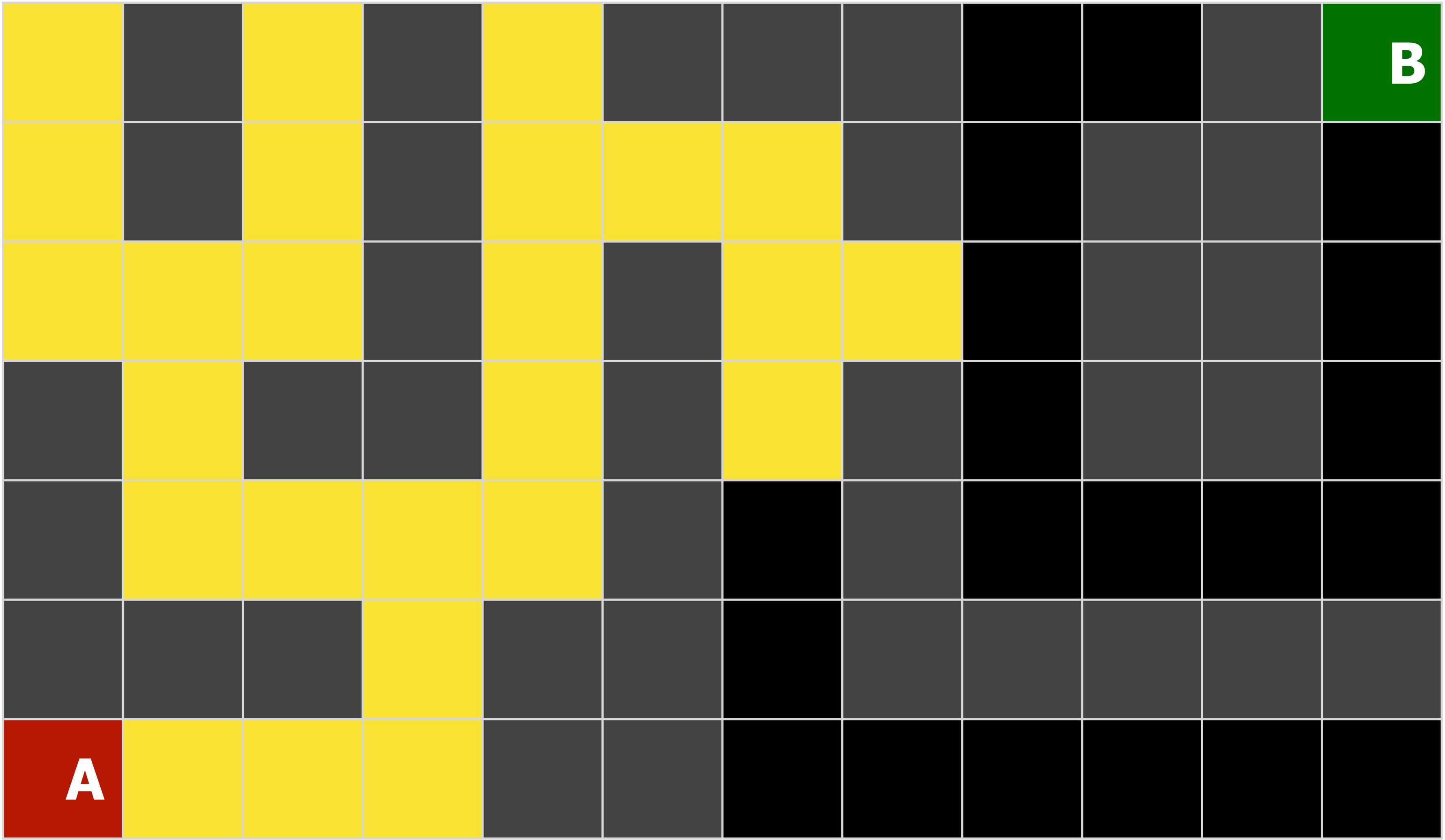
Breadth-First Search



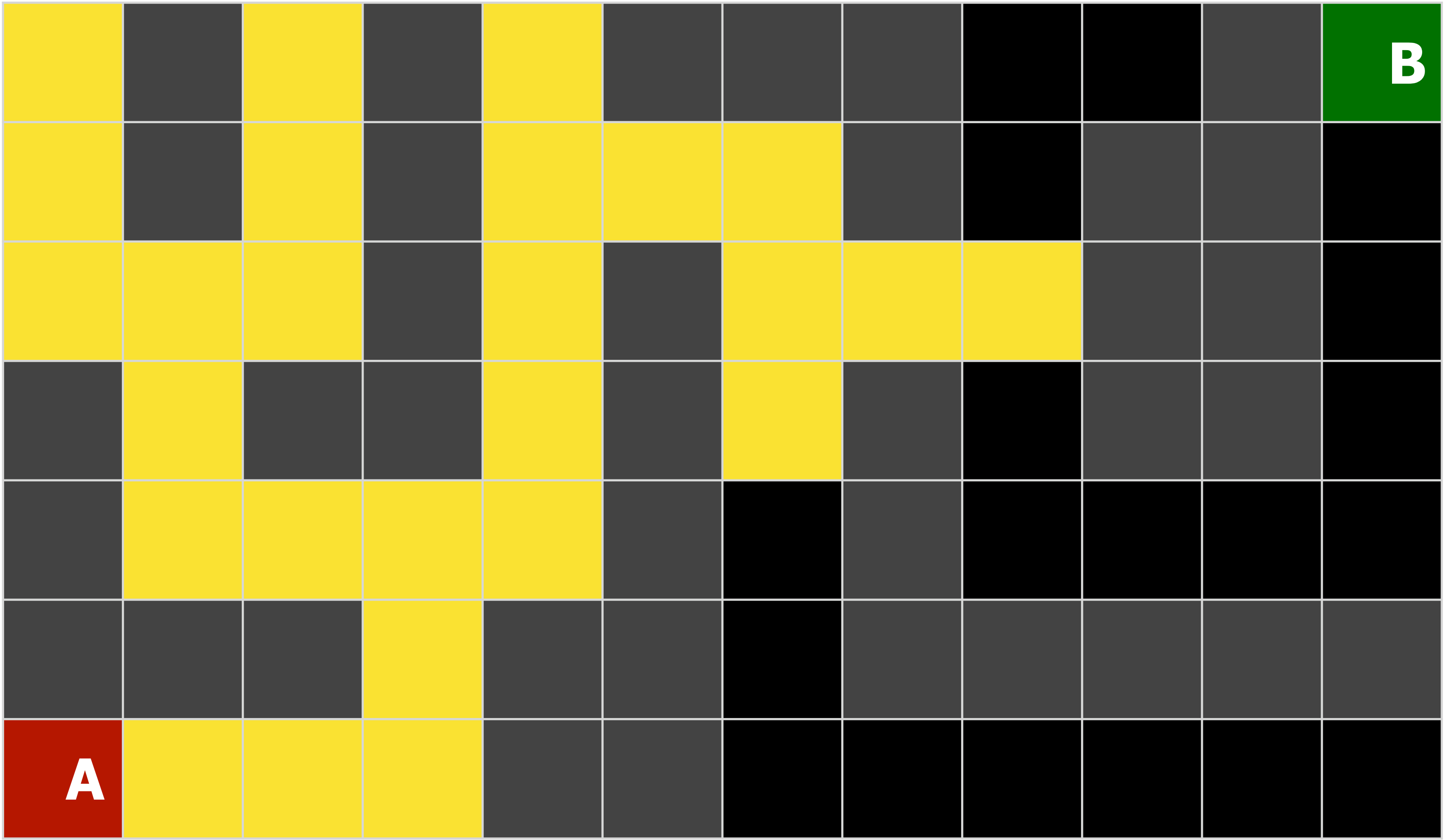
Breadth-First Search



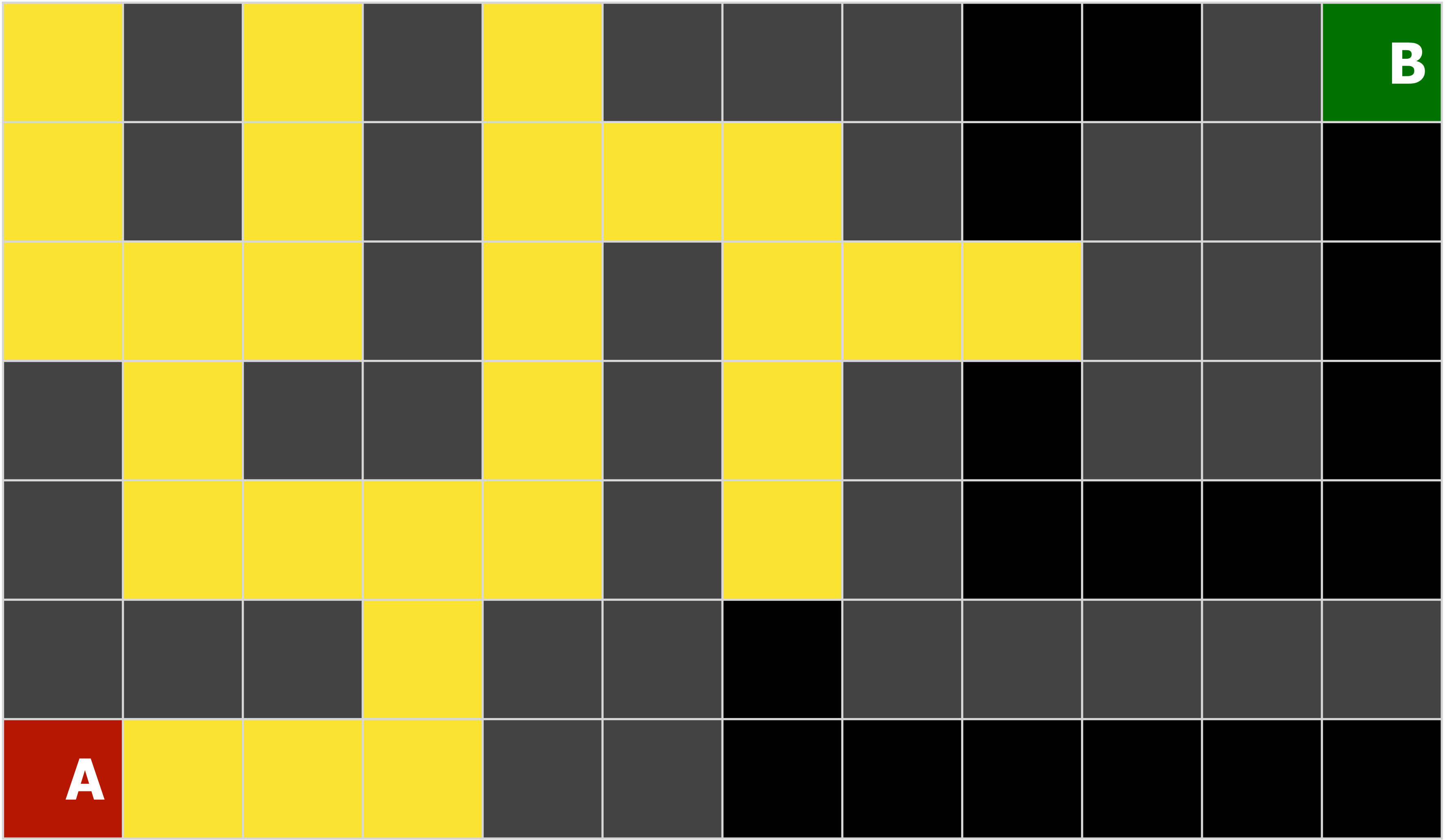
Breadth-First Search



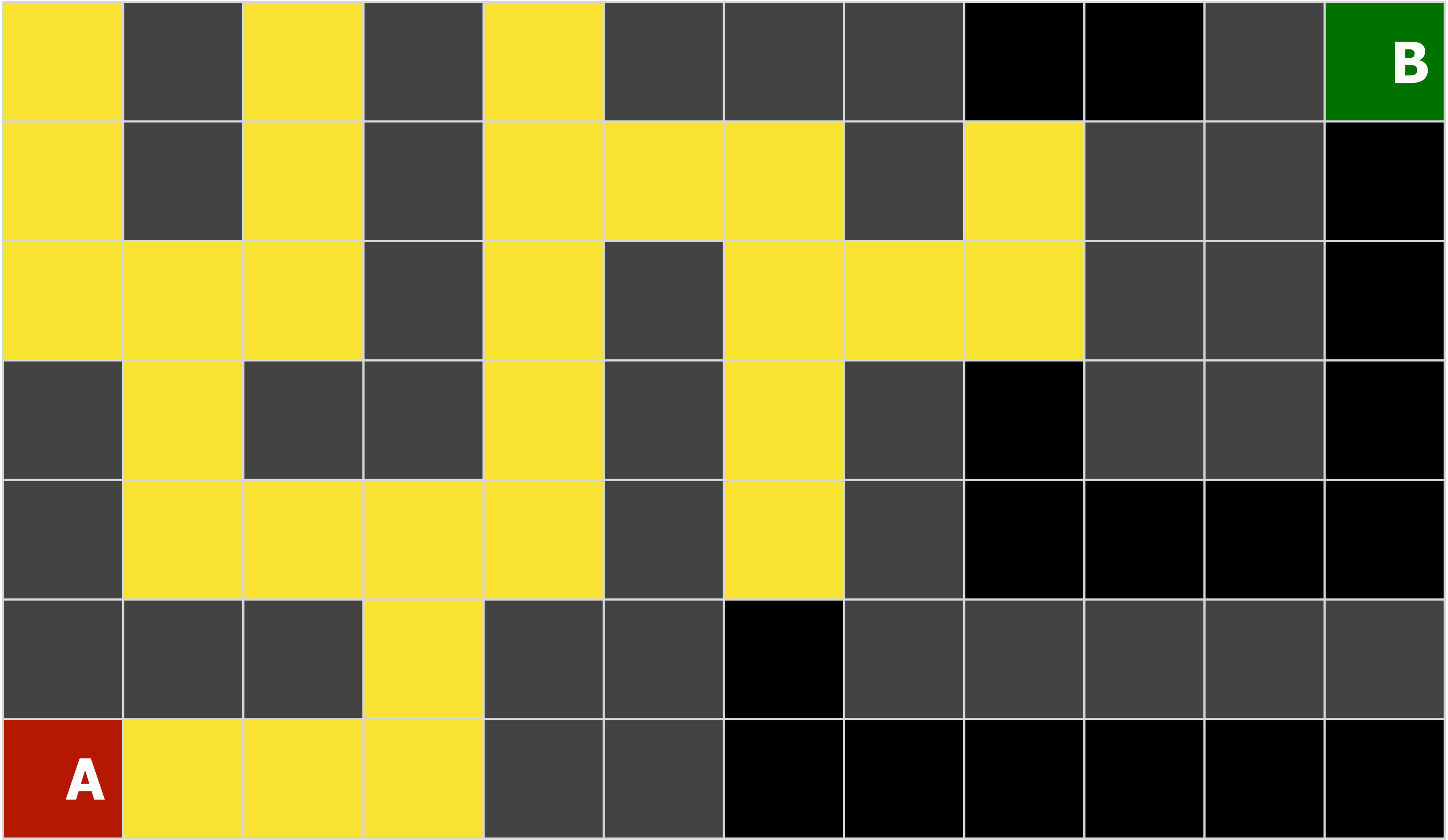
Breadth-First Search



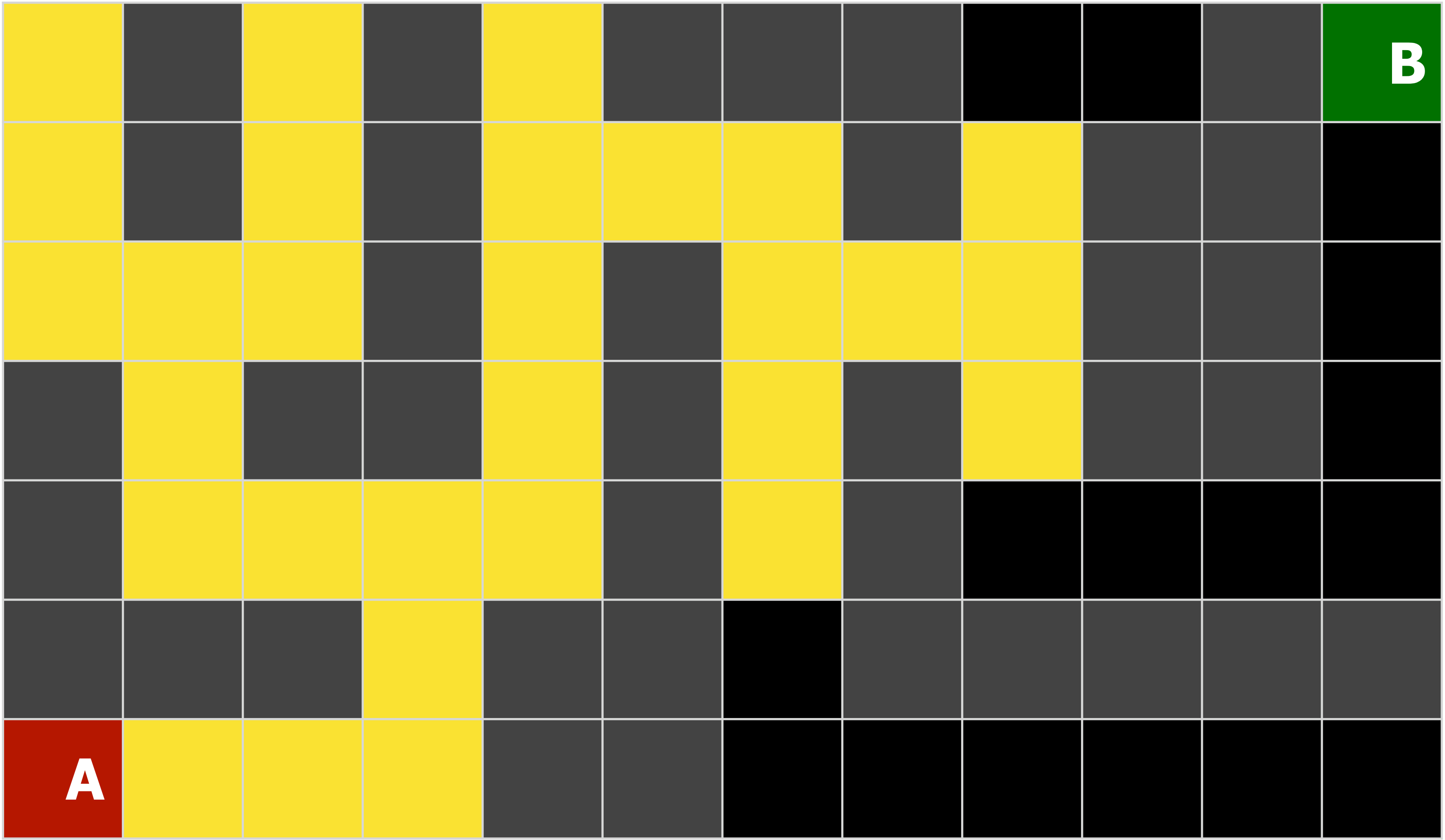
Breadth-First Search



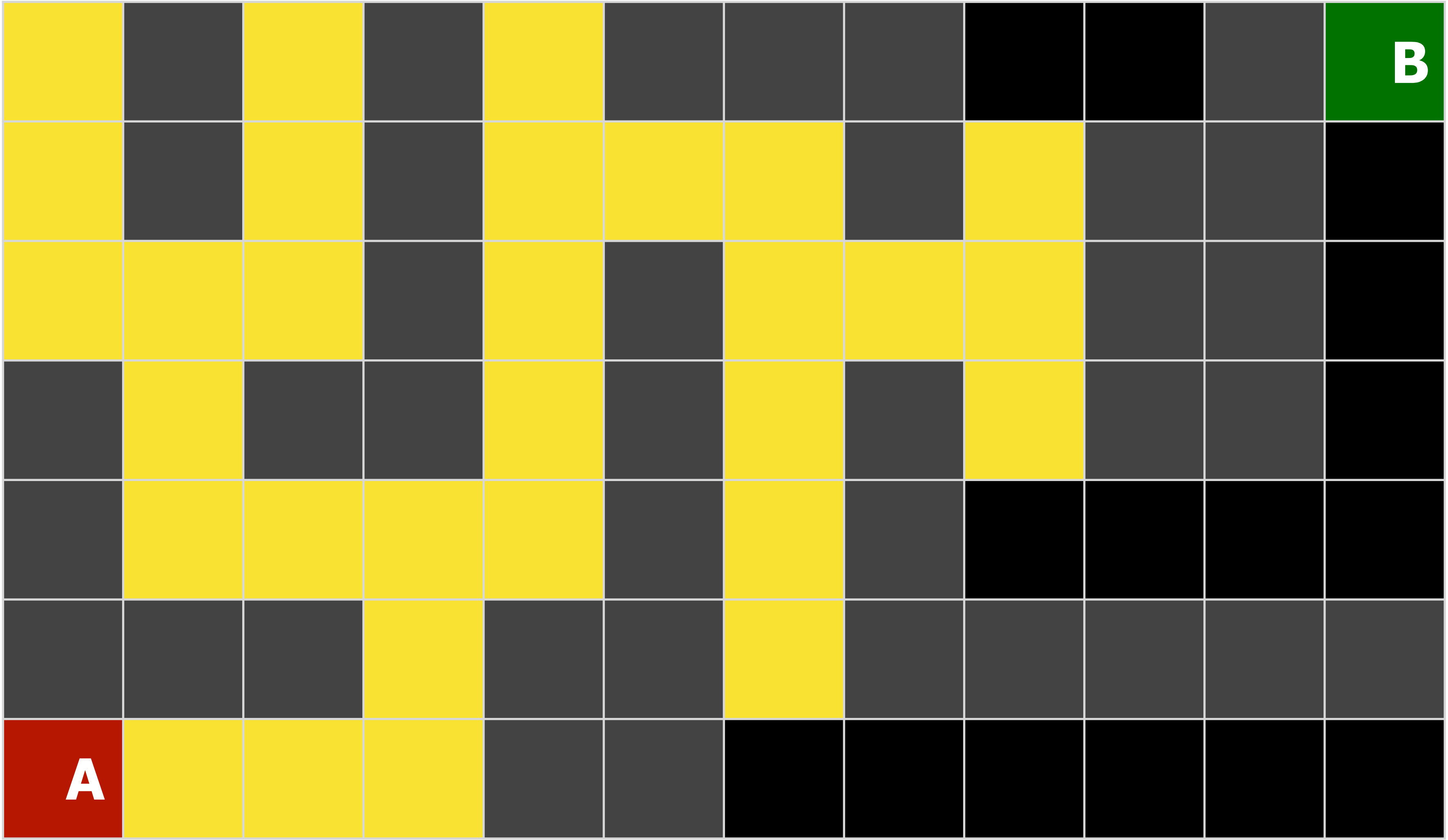
Breadth-First Search



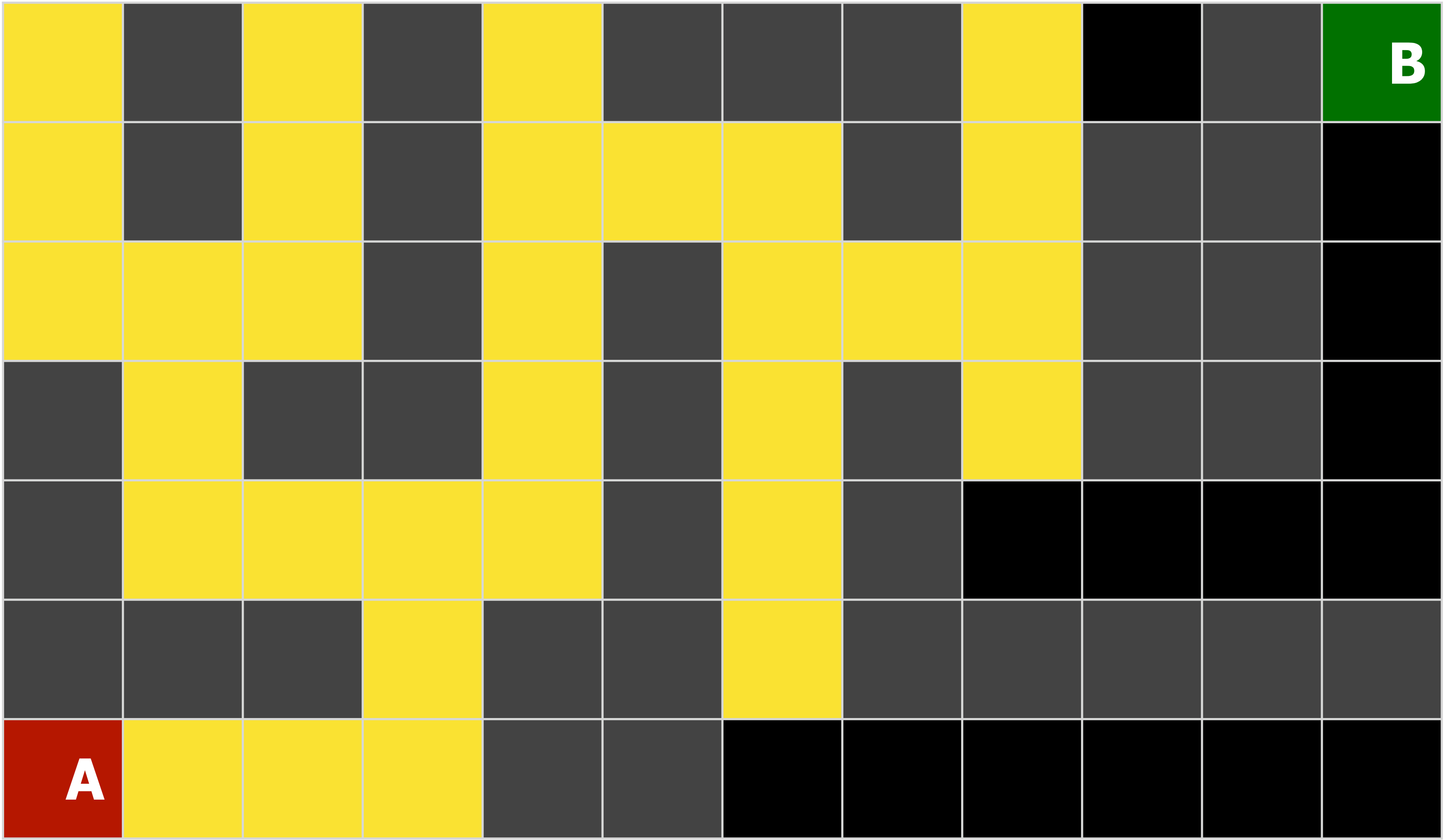
Breadth-First Search



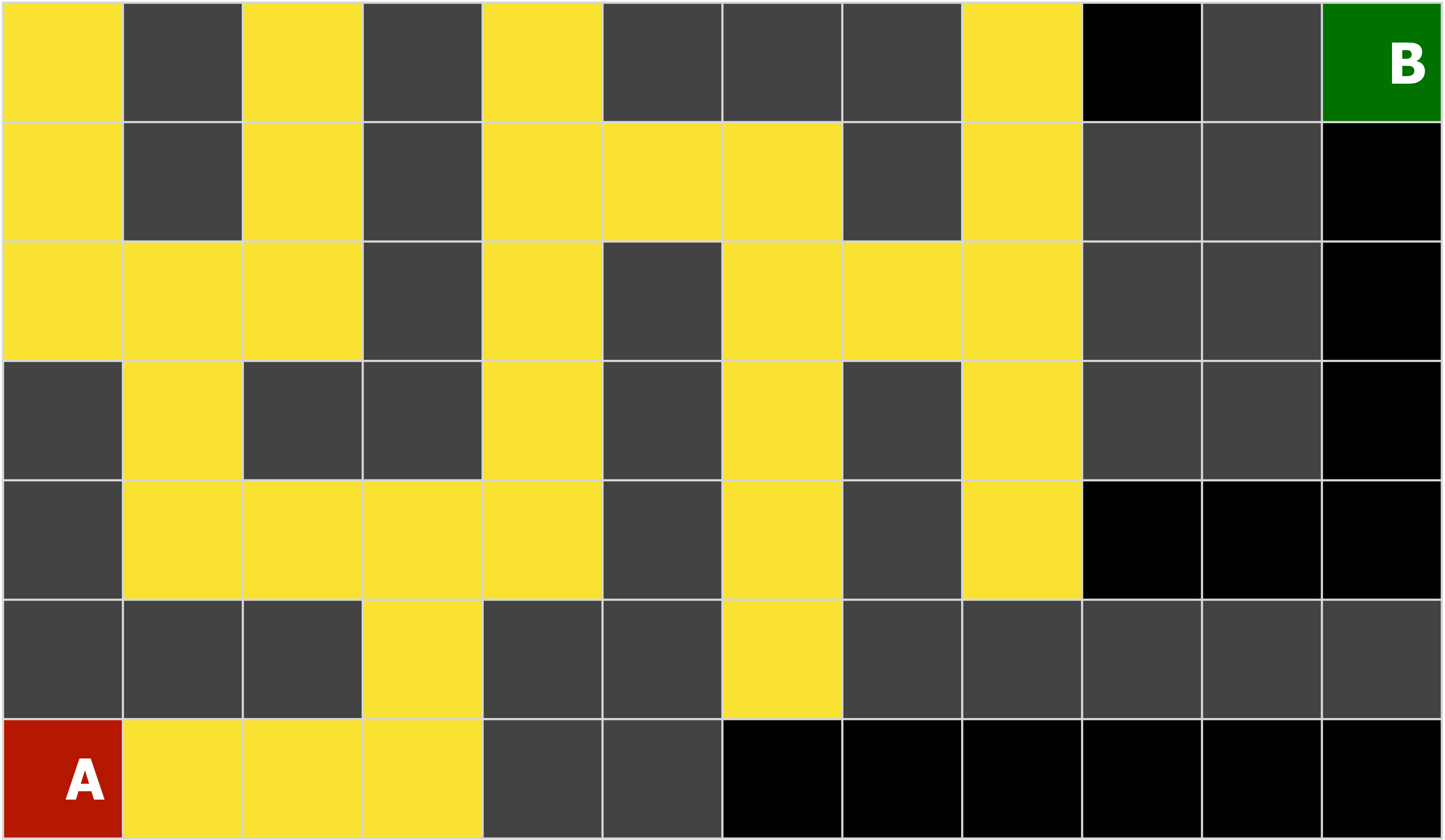
Breadth-First Search



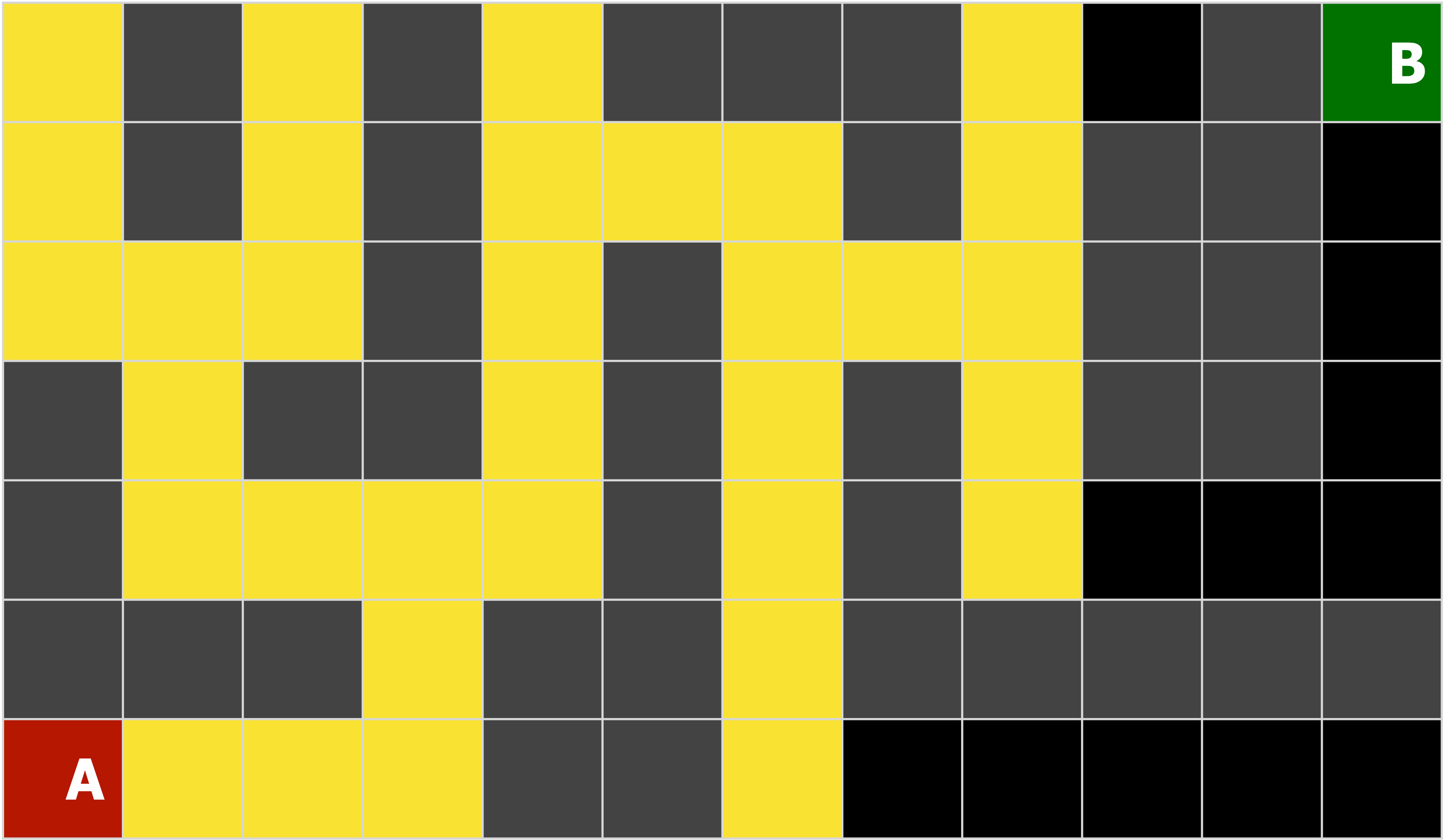
Breadth-First Search



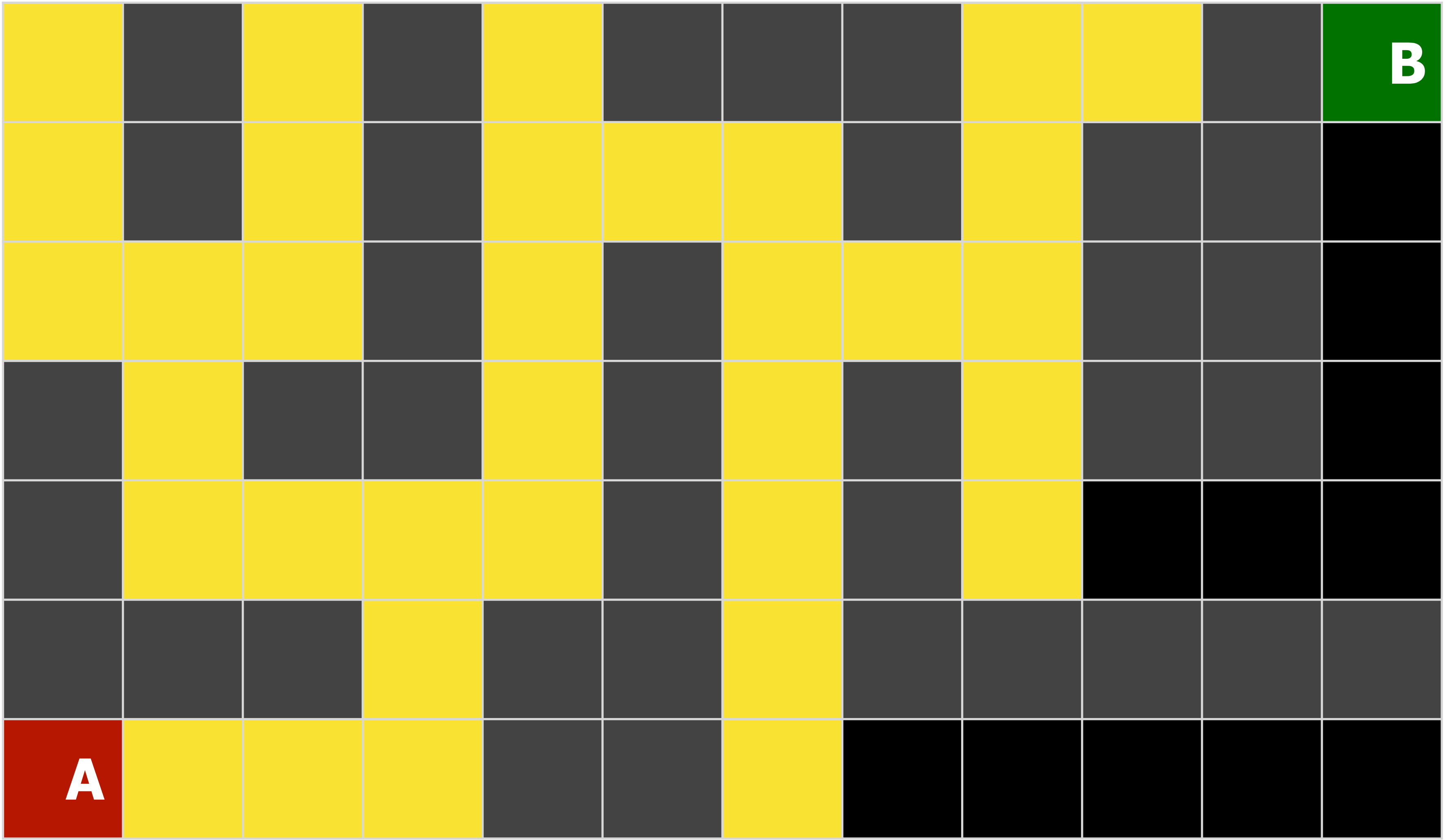
Breadth-First Search



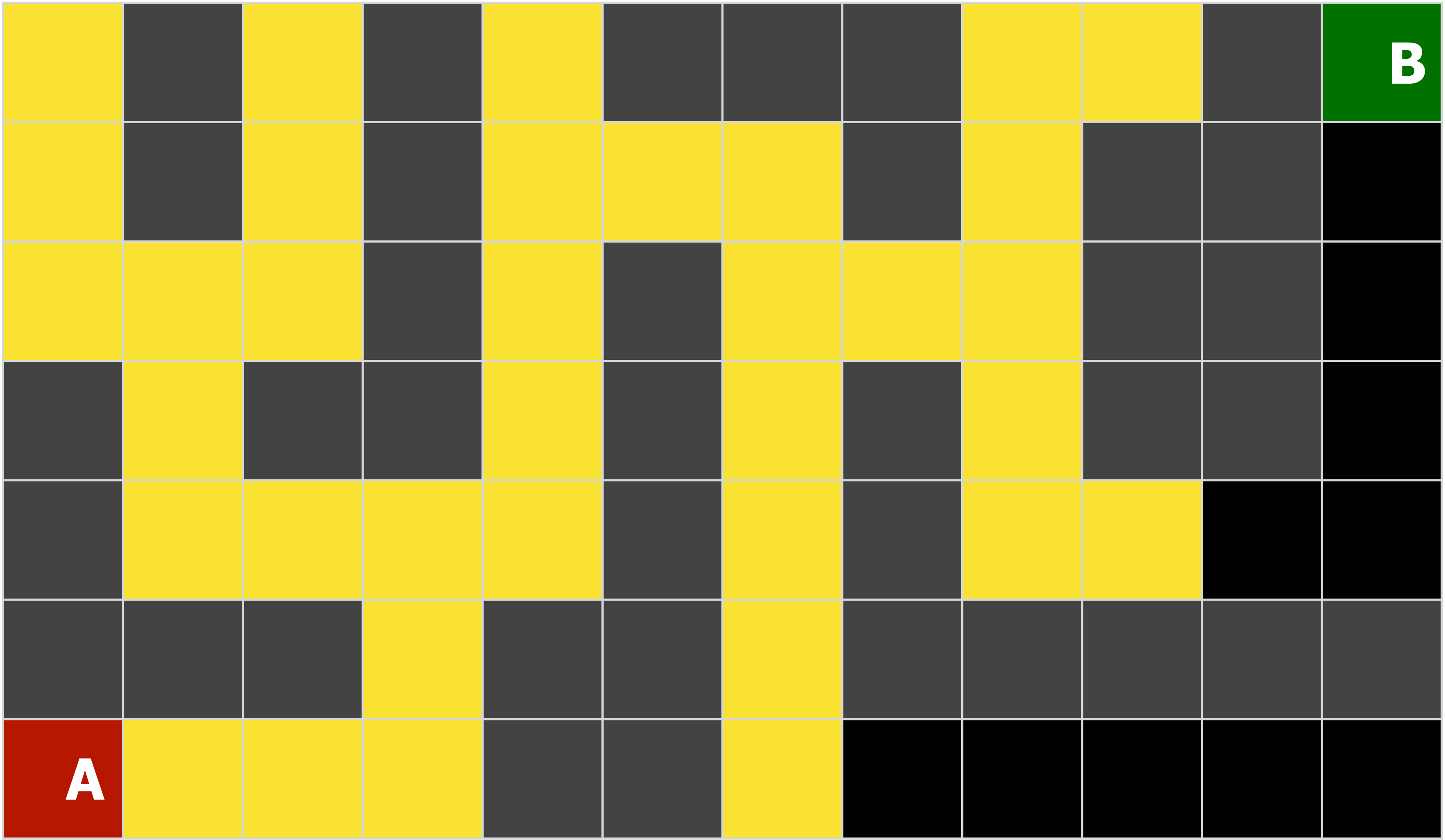
Breadth-First Search



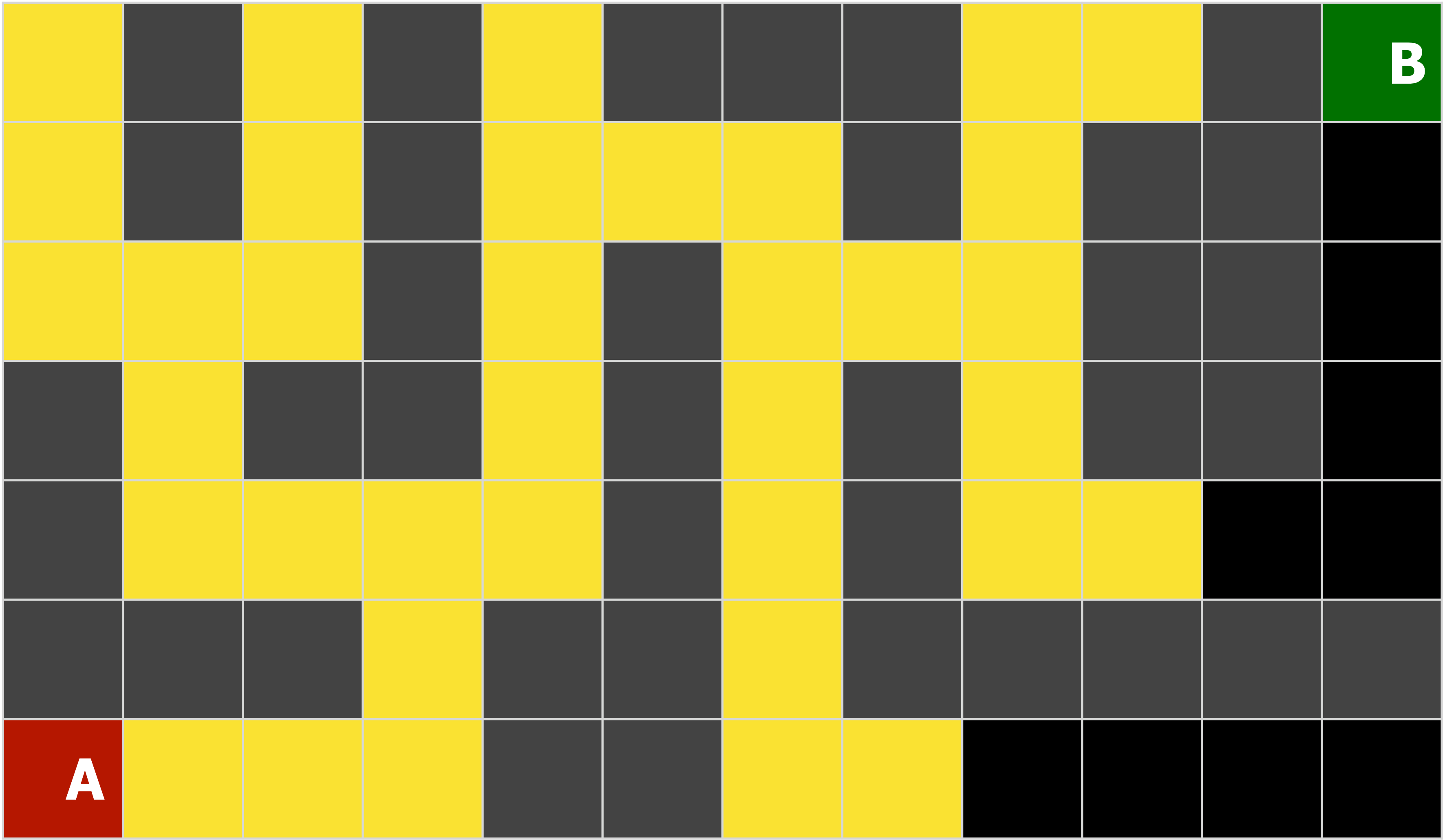
Breadth-First Search



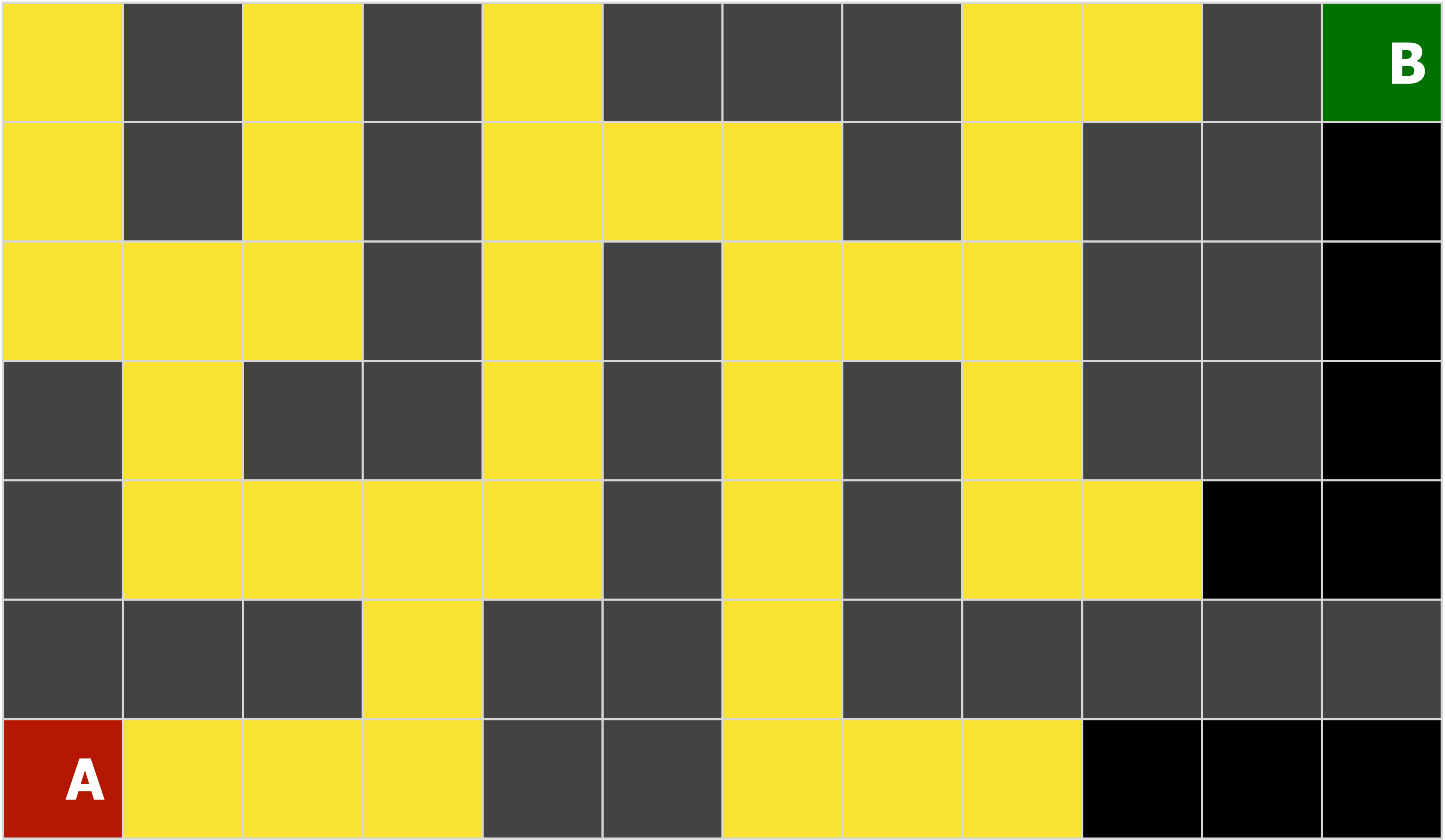
Breadth-First Search



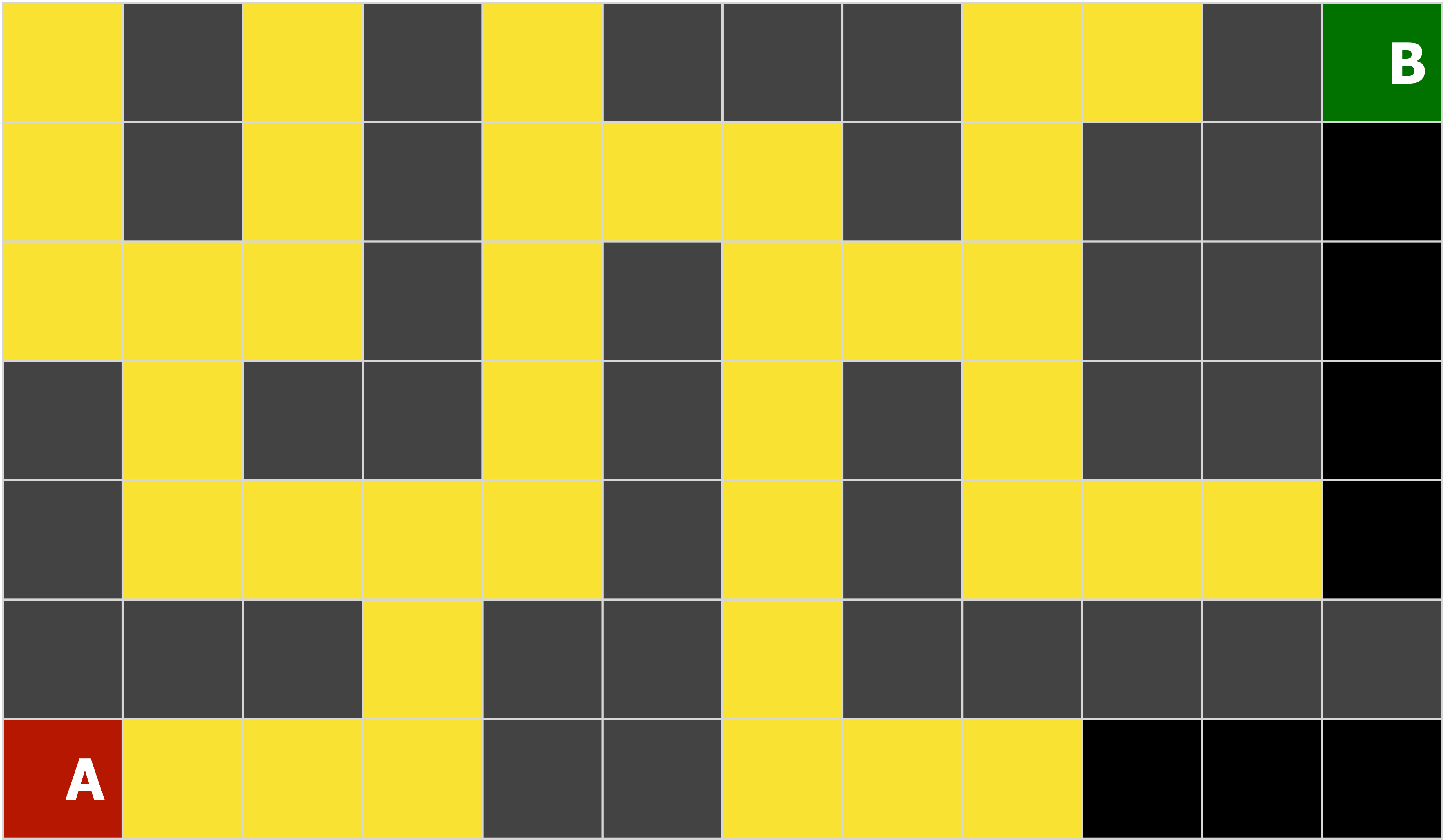
Breadth-First Search



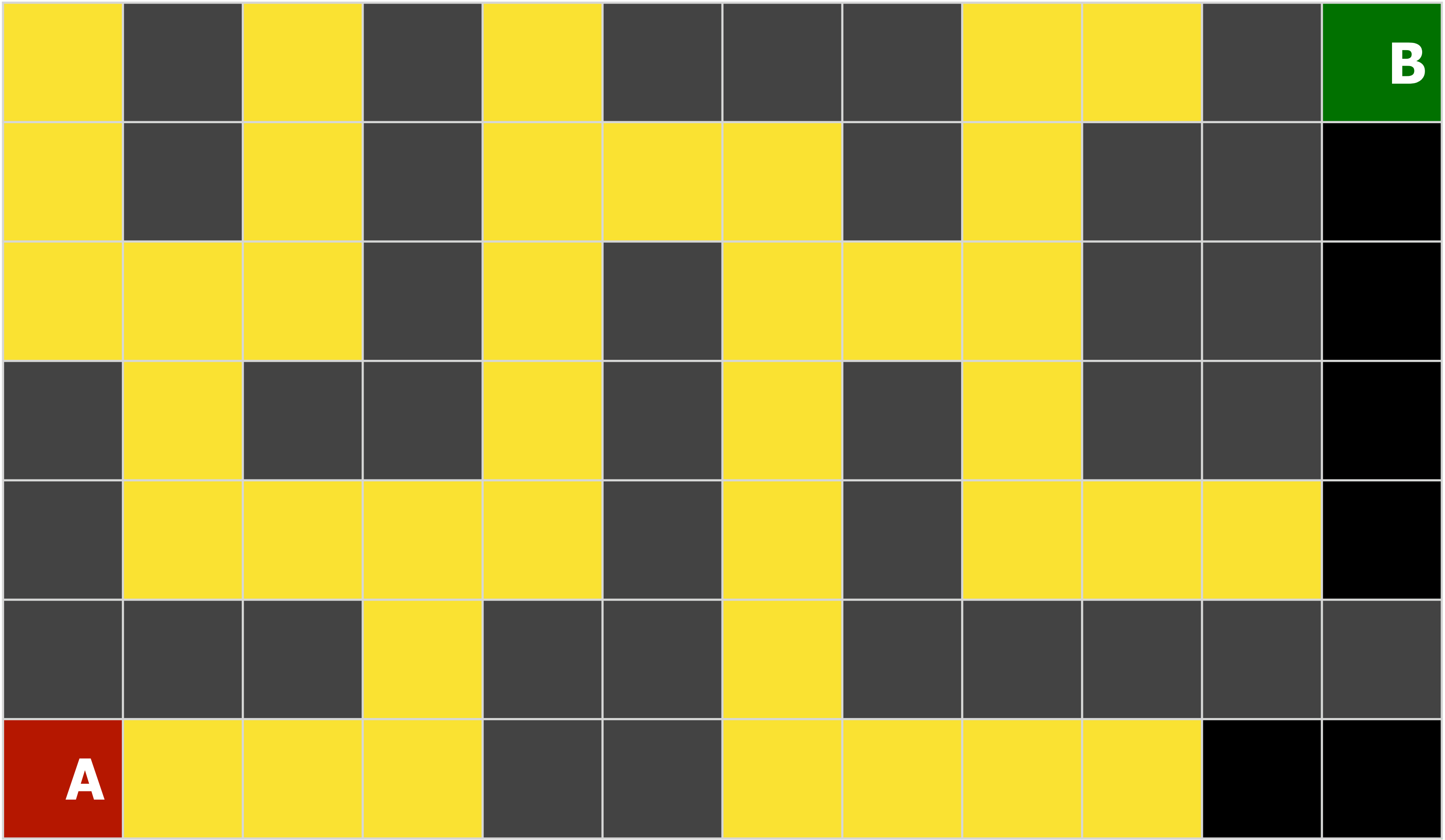
Breadth-First Search



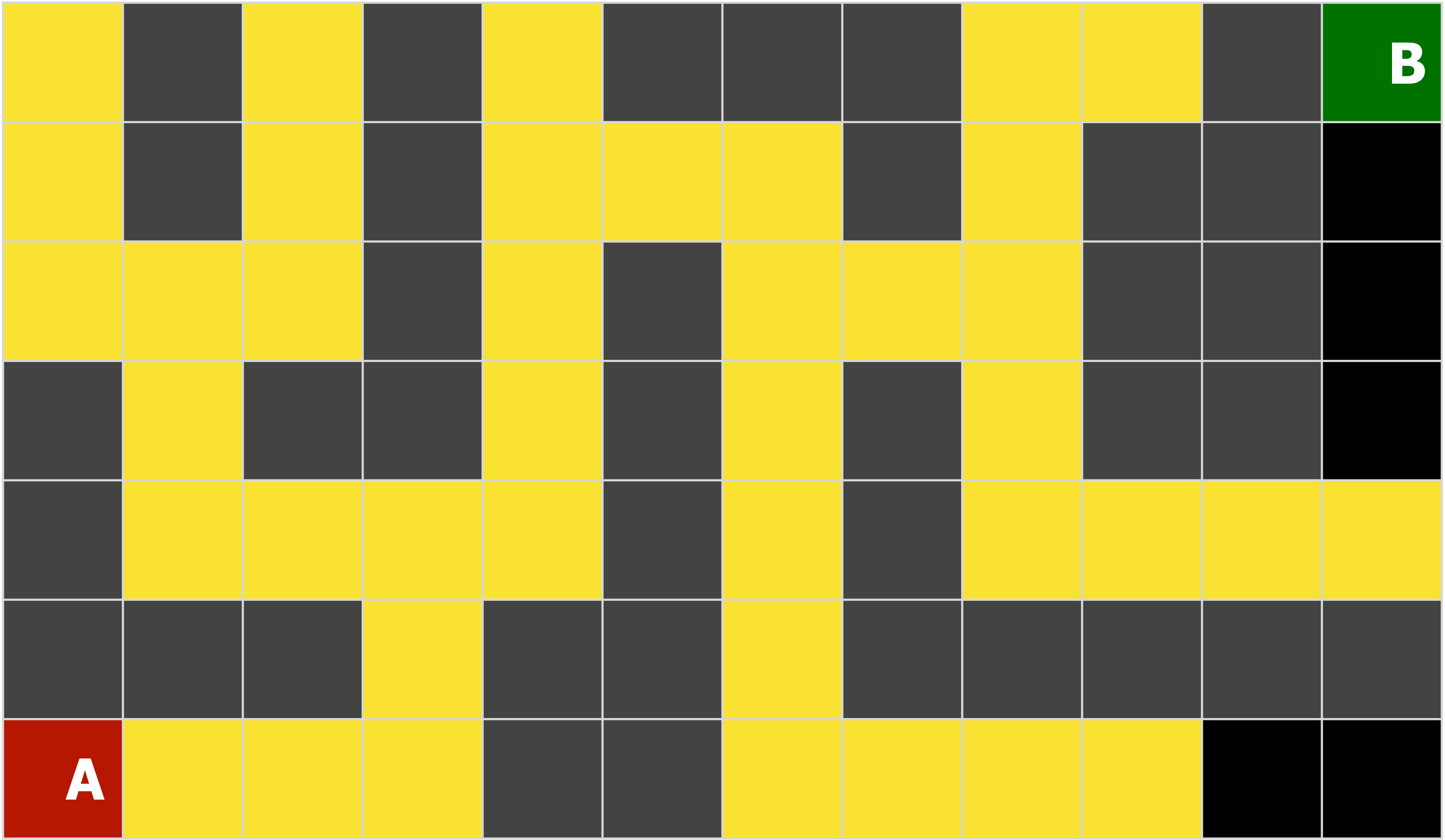
Breadth-First Search



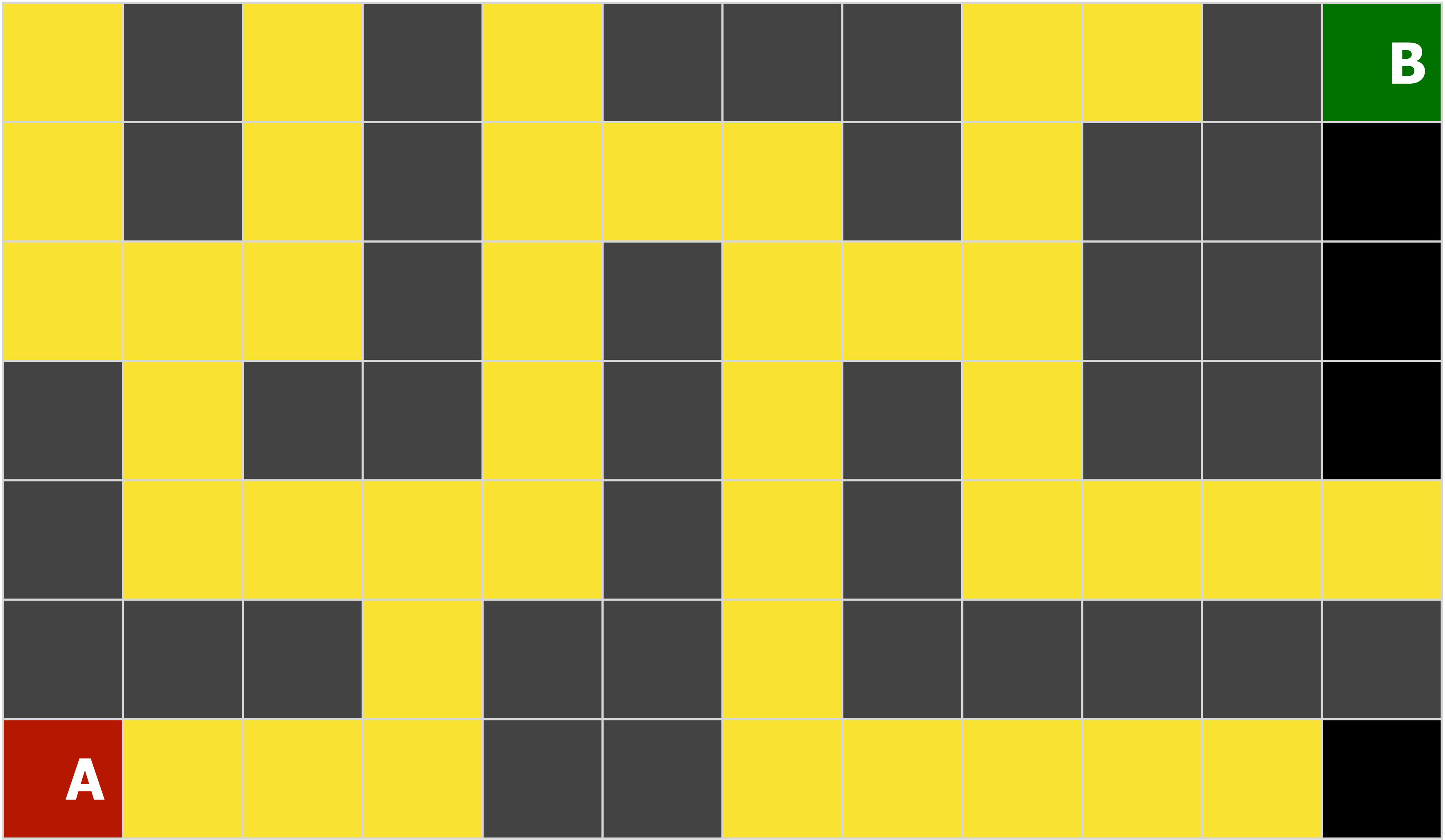
Breadth-First Search



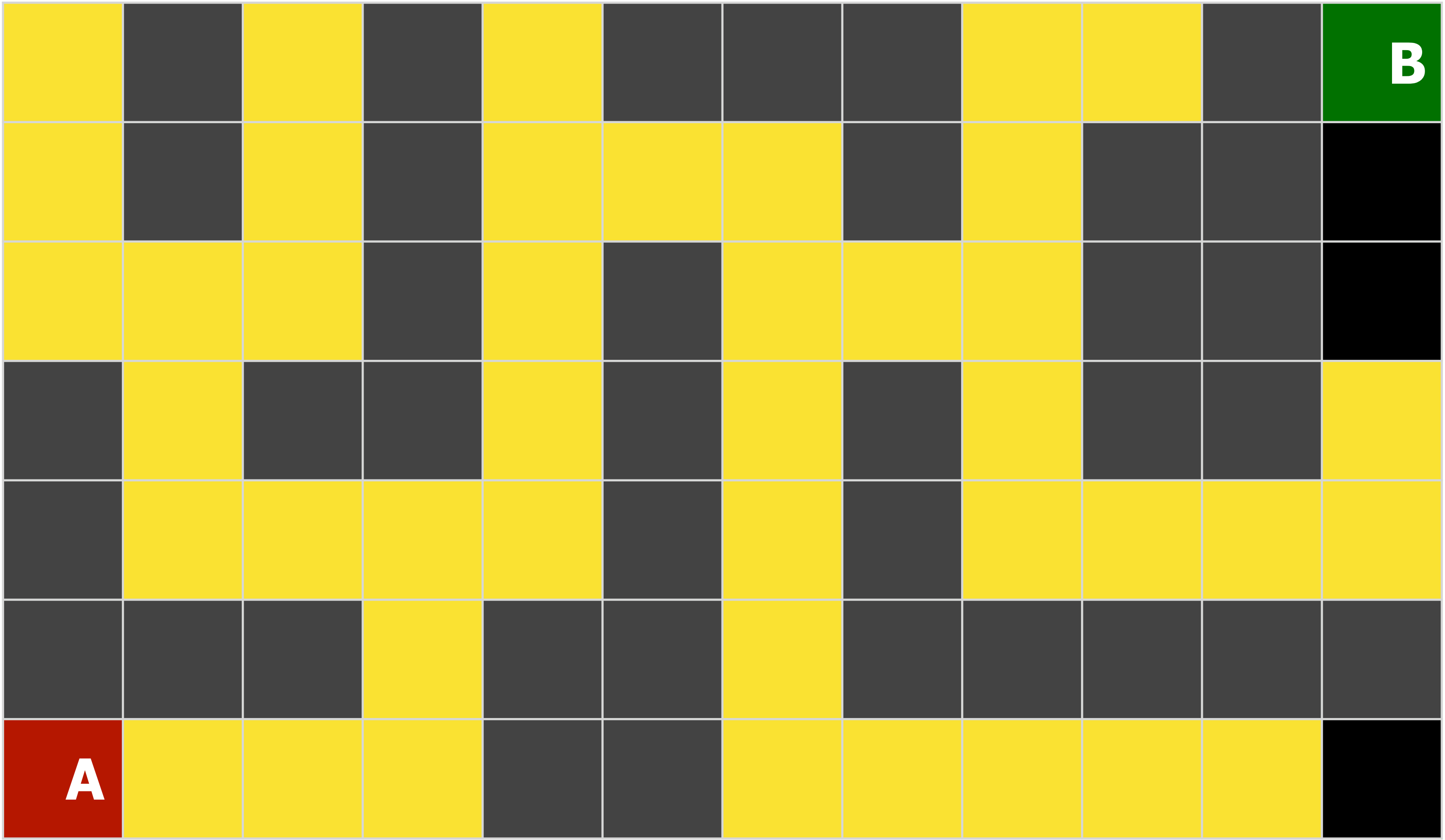
Breadth-First Search



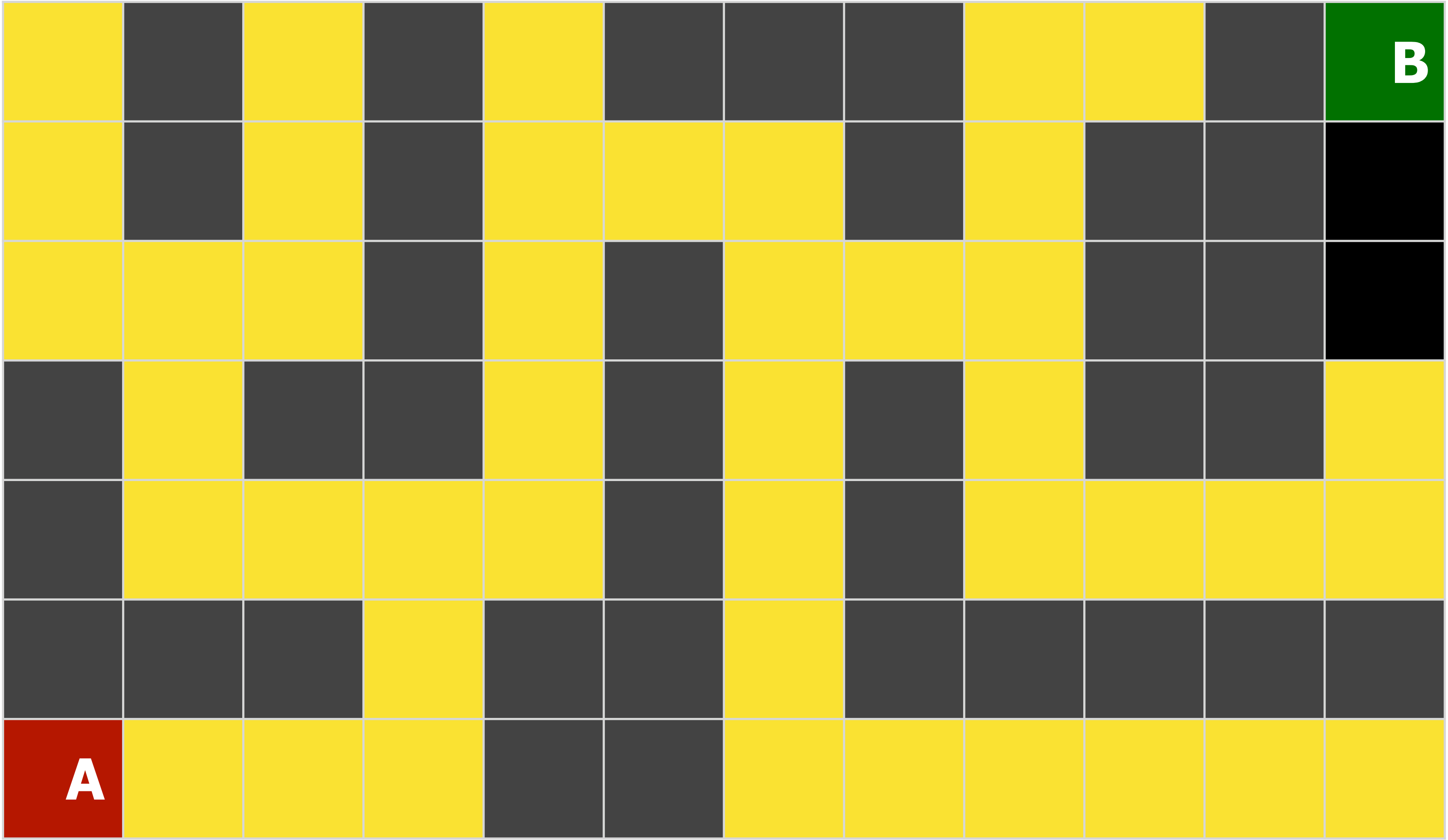
Breadth-First Search



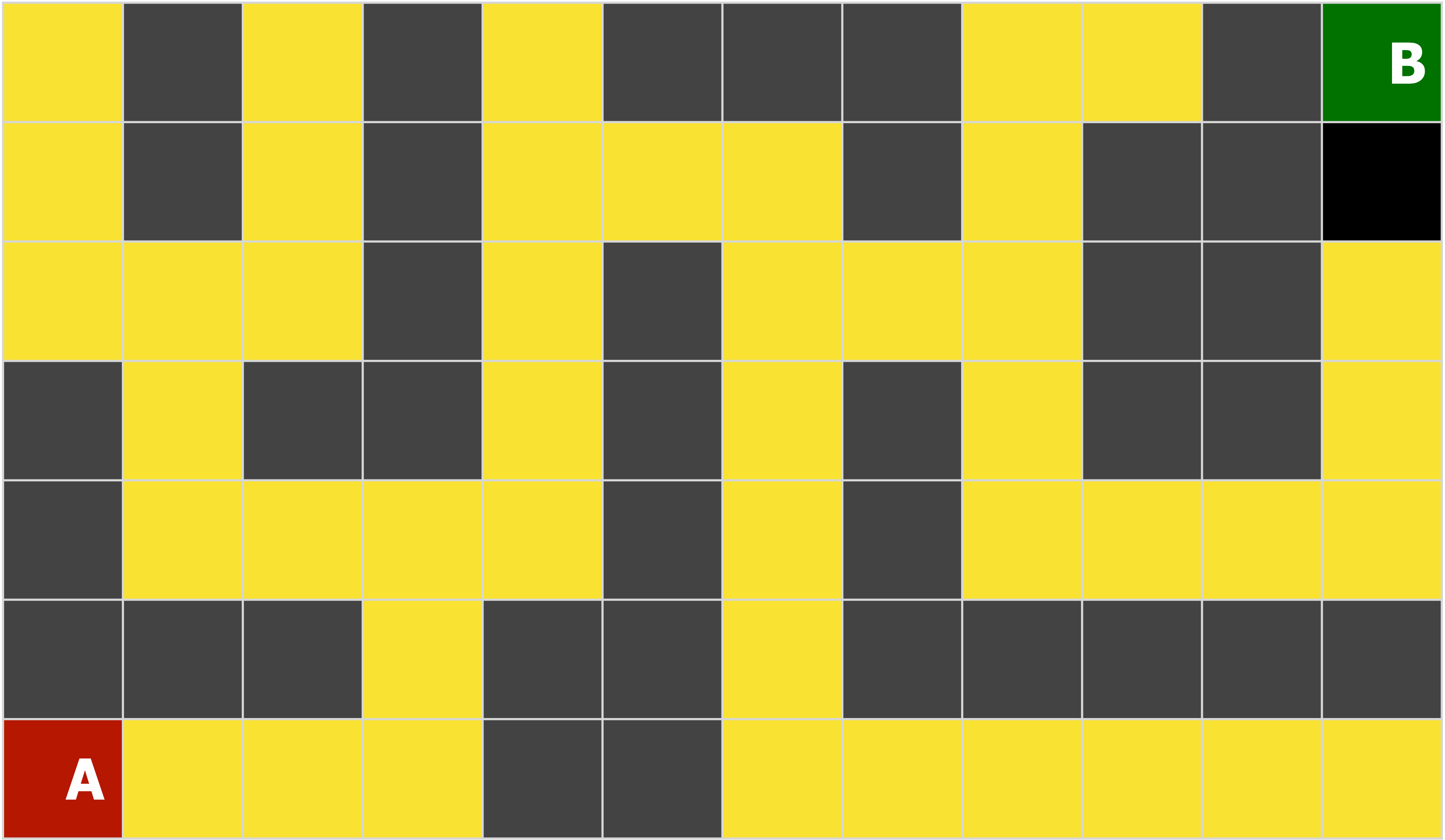
Breadth-First Search



Breadth-First Search



Breadth-First Search



Breadth-First Search