Assignment 2: Planning and Search

In the field of Artificial Intelligence (AI), planning and search are fundamental concepts that enable systems to make decisions, solve problems, and navigate complex environments. These techniques are the backbone of many AI applications, from game playing and robotics to scheduling and autonomous navigation.

In this assignment, we will begin by introducing the concept of planning domains using a simple game environment called Grid World. In this context, we will learn how to define a planning problem and understand the solutions to these problems. Following this, we learn a fundamental search algorithm for solving planning problems called Breadth First Search (BFS).

Part 1: Planning Domains and Planning Problems

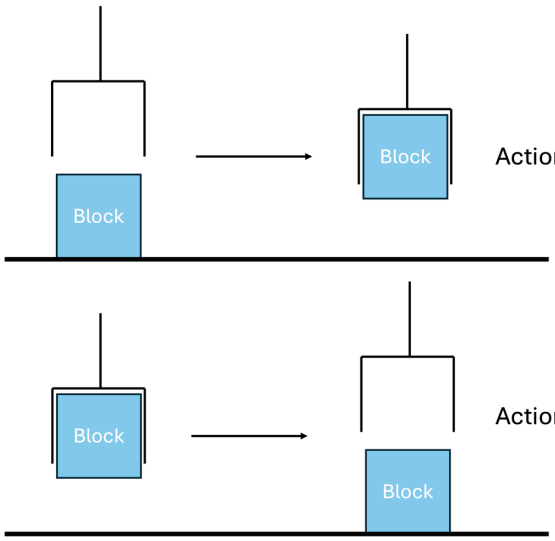
Planning Domain

A planning domain is a structured way to describe the world, enabling us to define and solve planning problems using standard algorithms. It consists of a set of possible states, representing different configurations of the world, and a set of actions that describe how these states can be changed. Each action has specific prerequisites that must be satisfied before it can be executed, as well as defined effects that describe the resulting changes in the world state. For instance, in a simple planning domain called "Having Dinner," the states "hungry" and "full" represent whether you are hungry or full. The action "eat" can only be performed when you are in the "hungry" state, and its effect is to change the state to "full".

Let’s look into a classical planning domain called 'Pick and Place' as illustrated in the figure below. This domain involves taking actions to change the state of a block using a hand gripper. There exist two possible block states: *block in hand* and *block on table*. As the name suggested, two actions are available: *pick* and *place*.

- Action *pick* requires the state of *block on table* and changes the state to *block in hand* as the effect.
- Action *place* requires the state of *block in hand* and changes the state to *block on table* as the effect.

Planning Domain: Pick and Place



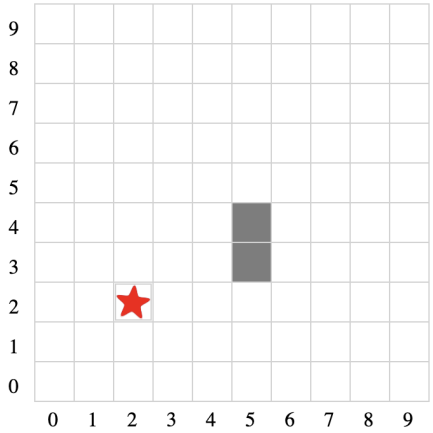
	Requirement	Effect
Action: *Pick*	*Block on table*	*Block in hand*
Action: *Place*	*Block in hand*	*Block on table*

Now assume there exists another block and it is possible that the hand gripper is occupied when it is trying to pick a block. Let's call the two blocks block A and block B. Now the available states are *block A in hand* or *block A on table*; and *block B in hand* or *block B on table*. Please fill out the blanks as follows to modify the actions

[Student need to fill :

- Action *pick A* requires the state of (1) *block A on table,* and *(2)* *block B on table*. It will change the state to (1) *block A in hand* as the effect.
- Action *pick B* requires the state of (1) *block B on table,* and *(2)* *block B on table*. It will change the state to (1) *block B in hand* as the effect.

]



Now let's consider a new planning domain called Grid World shown in the figure above. The Grid World domain involves moving an agent, represented by a red star, in a 10-by-10 grid by

four possible actions that move the agent by one grid cell in each of the four directions: *move up*, *move down*, *move left*, and *move right*. The action is not possible if it moves the agent into a wall, represented by gray cells, or out of the grid. The states of the domain are locations of the agent represented as the x-y coordinate. For example, the agent's current location in the figure above is denoted as (2, 2). Now, describe the requirements and the effects of the four actions:

[Student answers

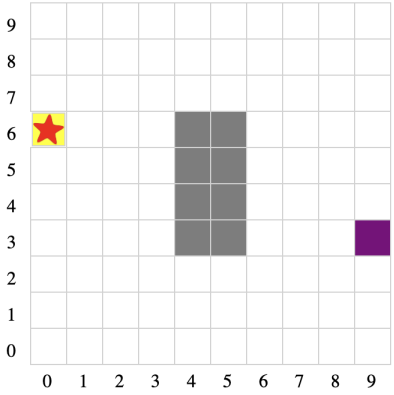
Assuming the agent is at a state of (x, y)

1. Move up:
    1. Requirement: (x, y+1) is not wall and not out of the grid
    2. Effect: agent state changes to (x, y+1)
2. Move down:
    1. Requirement: (x, y-1) is not wall and not out of the grid
    2. Effect: agent state changes to (x, y-1)
3. Move left:
    1. Requirement: (x-1, y) is not wall and not out of the grid
    2. Effect: agent state changes to (x-1, y)
4. Move right:
    1. Requirement: (x+1, y) is not wall and not out of the grid
    2. Effect: agent state changes to (x+1, y)

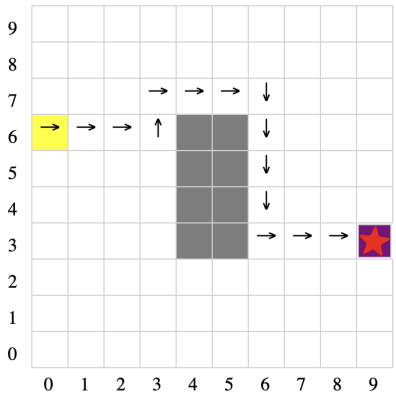
]

Planning Problem

A planning problem can be defined in a planning domain by specifying an initial state and a goal state. The figure below shows a well-defined planning problem with the initial state represented by a yellow cell and the goal state represented by a purple cell.



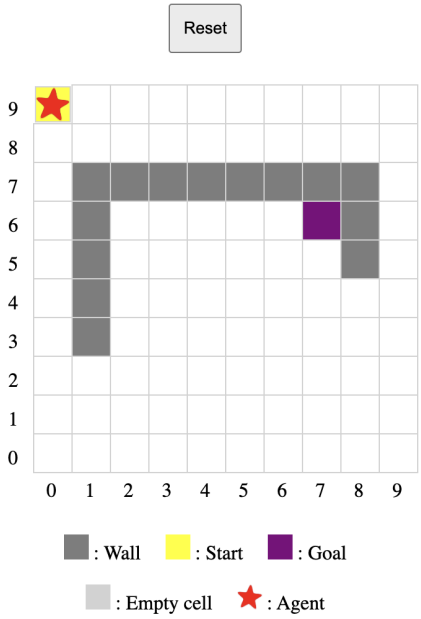
The solution to a planning problem is a sequence of actions that brings the world from the initial state to the goal state. If using up, down, left, and right arrows to represent the actions that a solution takes at different cells, a solution to the above planning problem can be visualized as follows:



Now, please create a solution to the planning problem defined in this Grid World sandbox.

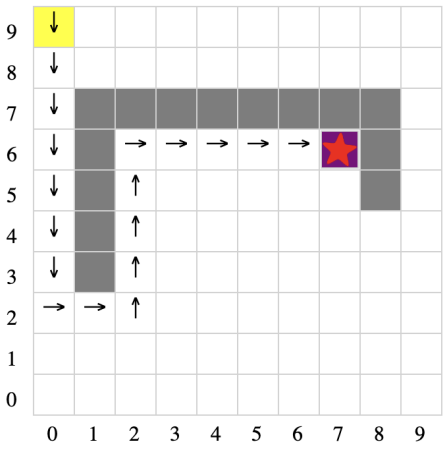
- You can move the agent around using “up”, “down”, “left”, and “right” keys on your keyboard. You cannot move the agent to previously visited cells or walls.
- Click the “reset” button to reset the agent to the “Start” location and clear the previous actions.

Grid World Environment 1



Upload a screenshot of your solution to this planning problem:

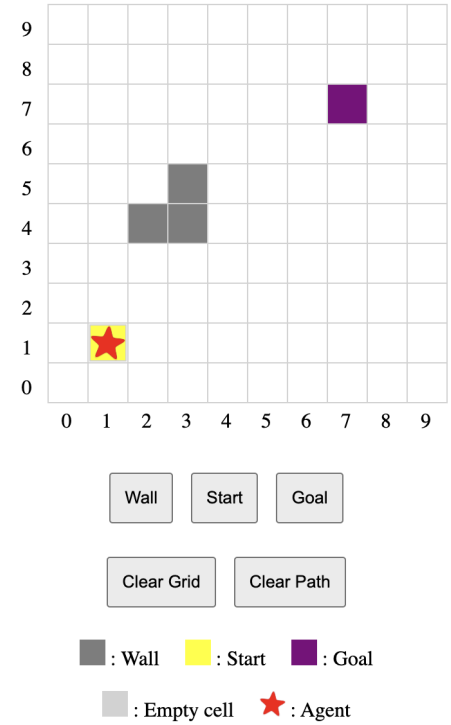
[Student answer:



]

Now try to build your own planning problem domain using this Grid World environment builder:

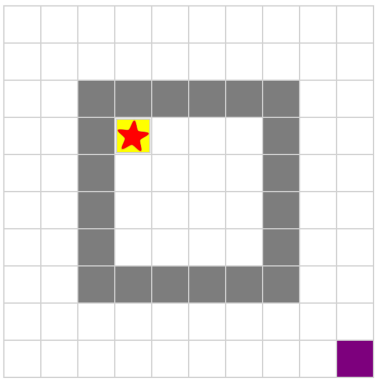
Grid World Environment Builder



- Click the “Wall”, “Start”, or “Goal” button to activate the corresponding mode.
- Click on a cell to set it to the currently active mode.
- Click “Clear grid” to set all cells to be empty.
- Click “Clear path” to remove previous actions and bring the agent back to the “Start” cell.

Now use the Grid World environment builder to create a planning problem that does not have a solution. Upload a screenshot of your planning problem.

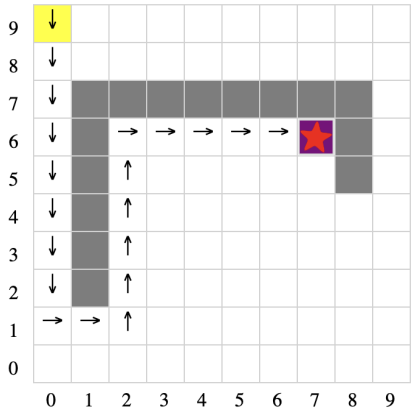
[Student fill: (below is an example)



]

A planning problem might have multiple valid plans. A domain may optionally have costs associated with each action, and a planning problem requires finding a valid plan with minimum cost. For example, the possible cost may be +1 for each move and a plan with the minimum cost corresponds to the shortest path(s) connecting start and goal locations.

Can you count the total cost for the plan shown in the following map?



[Students fill:

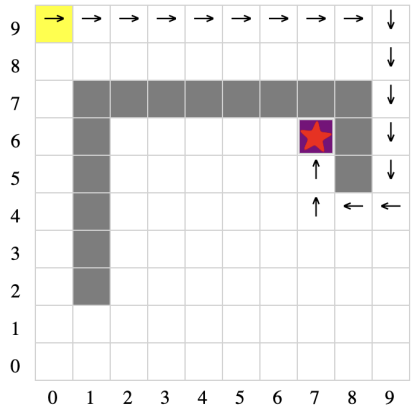
20

]

Does this plan minimize the cost? If not, try to create your own plan in this Grid World Sandbox and calculate the cost. Upload a screenshot of your minimal cost solution:

[Student fill:

No.



]

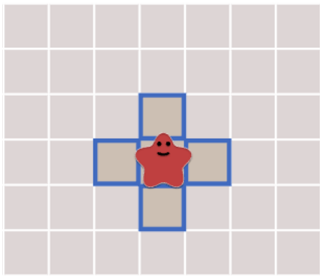
Part 2: Search Algorithms for Planning Problems

One ambitious goal of AI researchers is to find an algorithm that can solve any problem. As a small step towards that goal, let's find an algorithm that can solve any planning problem in the Grid World domain.

A search algorithm is a general approach for solving planning problems. One of the most basic search algorithms is Breadth-First Search (BFS). BFS systematically explores all possible states in a domain, starting from the initial state. It does so by progressively reaching states in order of increasing action sequence length—moving from closer states to more distant ones, where closer is defined as requiring less actions to reach. But how does BFS accomplish this? Beforing diving into the following parts, read Breadth First Search Section from this tutorial for detailed explanation of BFS. Ignore the Python codes if you have trouble understanding them.

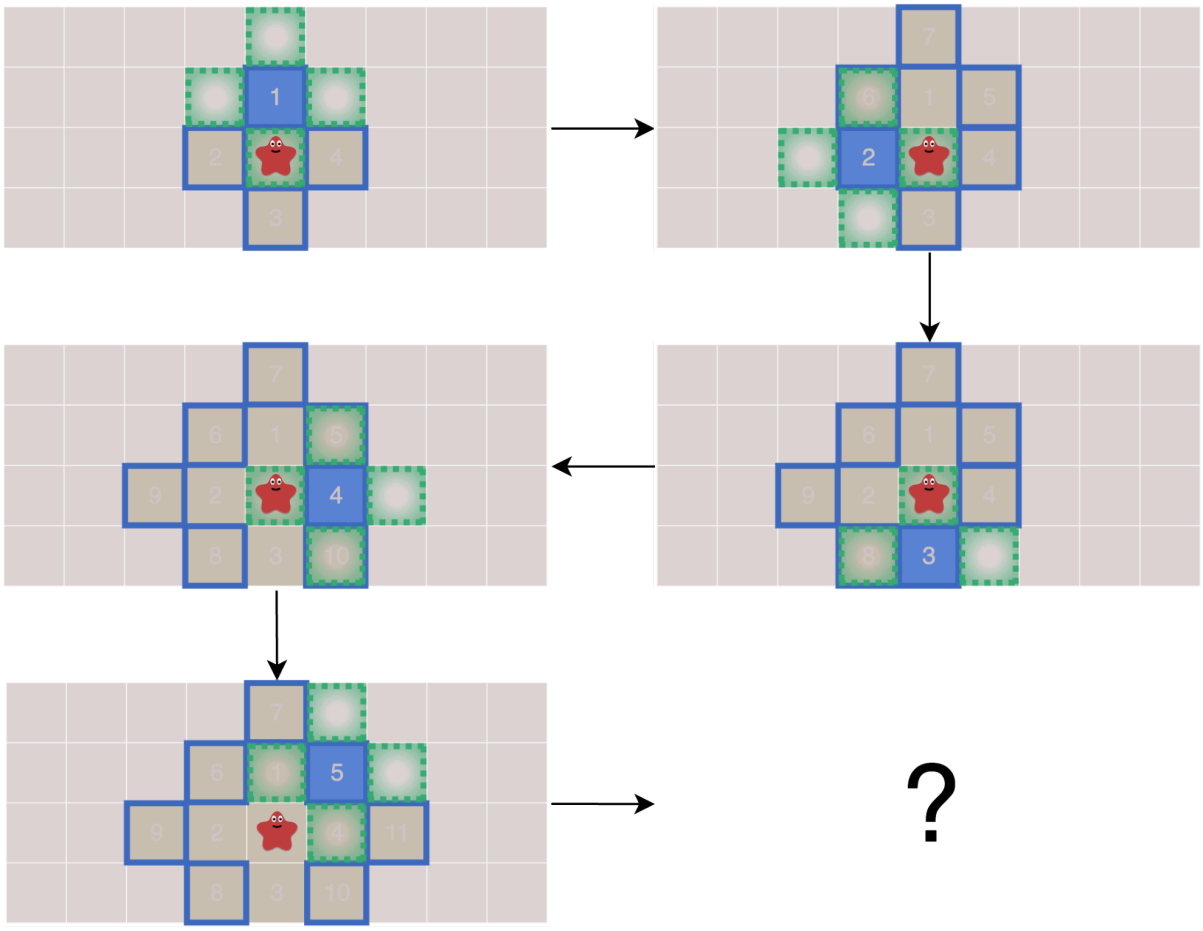
Frontier

As we learned from the tutorial, the frontier is the set of states that are one action away from the states that have been visited so far. In the figure below, at the initial state, the frontier consists of four states (highlighted in blue frames) that can be reached by moving up, down, left, or right from the initial state.



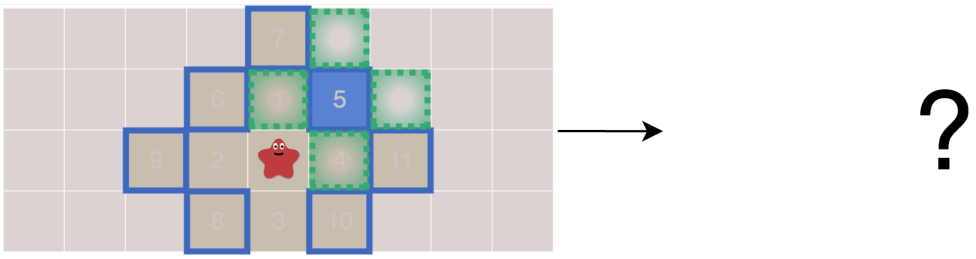
BFS operates by repeatedly executing the following steps to expand the frontier:

1. Pick and remove a state (or grid cell) from the frontier. In the figure, this is represented by turning the blue-framed states into bold blue blocks.
2. Expand it by finding states that are one action away from the picked state. These new states, shown as green blocks in the figure, are then added to the frontier.



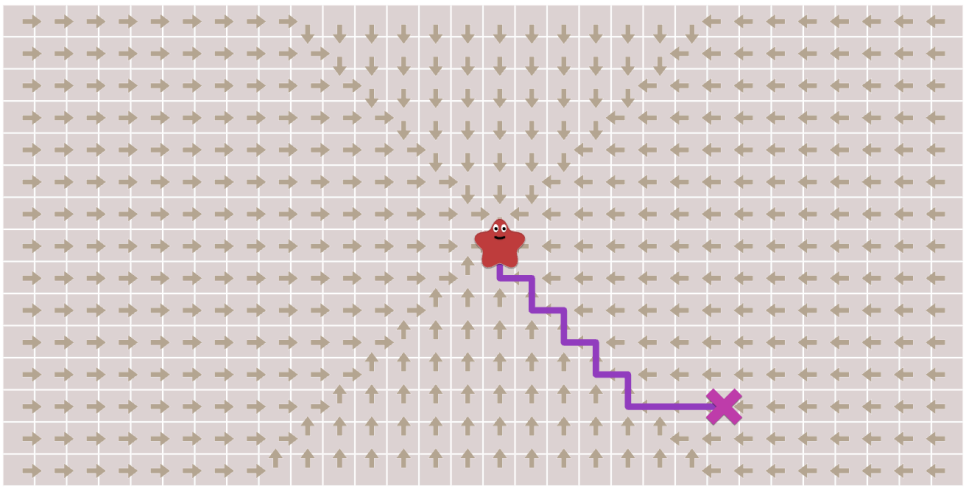
Each subfigure represents the execution of steps 1 and 2, progressively expanding the frontier one action away from the initial state. BFS guarantees that all states will be visited by continually expanding the frontier.

[Which of the grid cells will be visited next according to the figure below, and how many new grid cells will be added to the frontier?



a. Grid cell 6 will be visited next, and 2 new grid cells will be added to the frontier;
b. Grid cell 7 will be visited next, and 3 new grid cells will be added to the frontier;
c. Grid cell 6 will be visited next, and 4 new grid cells will be added to the frontier;
d. Grid cell 7 will be visited next, and 0 new grid cells will be added to the frontier;
]

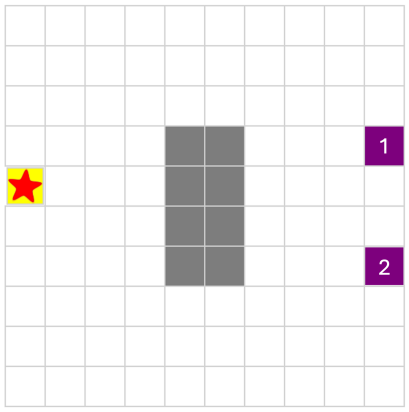
Now that we understand how BFS can find the goal state by exploring all possible states, the next question is: how does it find the sequence of actions (or the path) to reach the goal state? Notice that each state only enters the frontier once, and it does so by taking one action from a previously visited state. If we keep track of the action that brings a state into the frontier and the state from which it originated, the process would look like this:



For a goal state indicated by a purple cross, we can trace the path back to the initial state by following the arrows that show the state from which it was added to the frontier.

Now try to answer the following questions:

1. Assume two different goals exist in the grid shown. A path to either goal is considered a valid solution to this planning problem. Which goal will the BFS algorithm reach first?



[Students fill:

Goal 1

]

2. Does BFS guarantee to find the goal with minimum cost (assuming each action cost +1)? Explain your answer.

[Student fill:

Yes.

Explain: When BFS finds the goal cell, it is guaranteed to have tested all possible cells with fewer actions (lower cost) to reach before considering cells requiring more actions.

]