Lesson 5 Planning as Search

1. **Sebastian Introduction**

[**https://www.youtube.com/watch?v=2s3Pxc6AS8Y**](https://www.youtube.com/watch?v=2s3Pxc6AS8Y)

1. **Transition to Planning**

[**https://www.youtube.com/watch?v=0nyCrWuO1QM**](https://www.youtube.com/watch?v=0nyCrWuO1QM)

1. **The Planning Problem**

[**https://www.youtube.com/watch?time\_continue=4&v=oyPT3qirXPk**](https://www.youtube.com/watch?time_continue=4&v=oyPT3qirXPk)

1. **Search Space**

[**https://www.youtube.com/watch?v=vsMQtkOjT3c**](https://www.youtube.com/watch?v=vsMQtkOjT3c)

**There's a quiz below this video, scroll down to test your knowledge!**

**QUESTION 1 OF 2**

What are the problems with using continuous curves to represent paths through free space (choose all that apply)?

* \*\* The number of possible curves is large and approaches infinity for a complex search space.
* \*\* Constraints on processing time and computational resources will limit the number of paths you can consider.
* 

Real flying cars can't follow continuous curve trajectories.

* \*\* In general, continuous curves are unnecessarily complex representations of the problem.

SUBMIT

**QUESTION 2 OF 2**

What's the solution / alternative to using continuous curves in your representation of the planning problem?

* \*\* Discretize your search space
* 

Add extra dimensions to your search space

* 

Use a faster computer

SUBMIT

1. **Grid Representation**

[**https://www.youtube.com/watch?v=WcC-CfvQuJI**](https://www.youtube.com/watch?v=WcC-CfvQuJI)

**There's a quiz below this video, scroll down to test your knowledge!**

**QUIZ QUESTION**

Which of these things do you need in order to completely represent your planning problem (choose all that apply)?

* \*\* A state space
* \*\* Actions that define how the vehicle can move from state to state
* \*\* A start state and a goal state
* \*\* Costs associated with each possible action

SUBMIT

1. **Search**

[**https://www.youtube.com/watch?v=\_iJ\_pPZcLJc**](https://www.youtube.com/watch?v=_iJ_pPZcLJc)

**There's a quiz below this video, scroll down to test your knowledge!**

**QUIZ QUESTION**

Complete the following sentence: "A partial plan is..."

* 

A rough idea of how to get from the start state to the goal state

* A plan that does not yet reach the goal, but that contains a list of states and actions moving away from the start state
* 

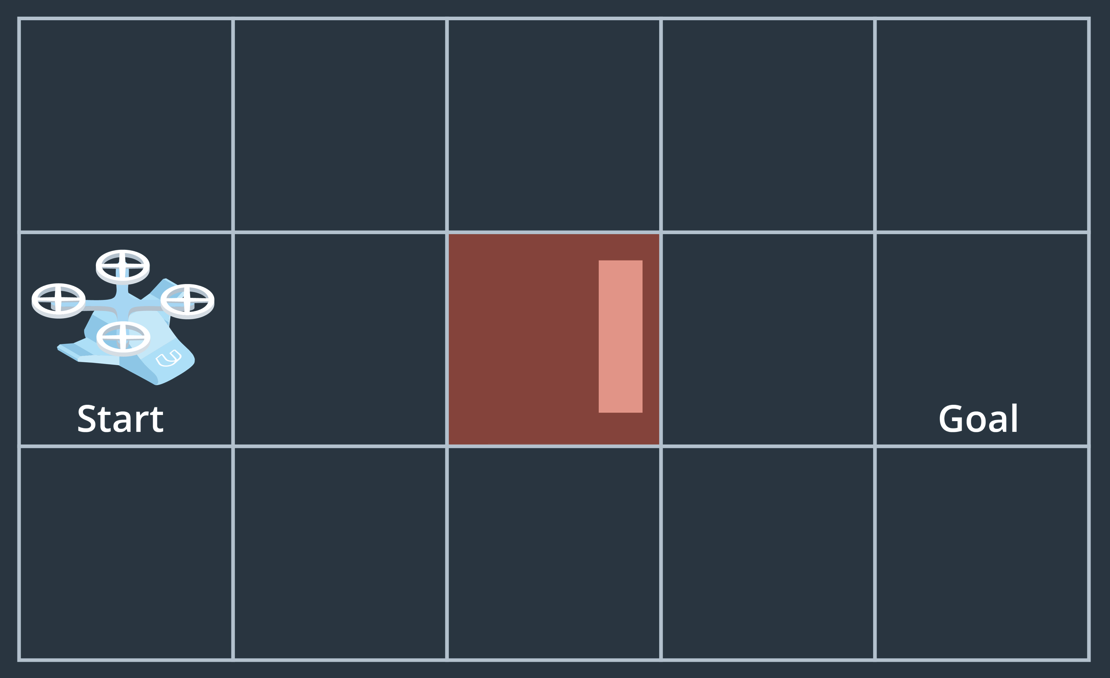
A plan that, even if it's not complete, at least makes progress toward the goal

SUBMIT

1. **Partial Plans**

[**https://www.youtube.com/watch?time\_continue=1&v=zcuT9e4TgKo**](https://www.youtube.com/watch?time_continue=1&v=zcuT9e4TgKo)

**There's a quiz below this video, scroll down to test your knowledge!**

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)**

**[Your task is to navigate through the grid world shown above by exploring partial plans from start to goal.](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)**

**QUIZ QUESTION**

**Here are the rules:**

* you can't move off the grid and you can't move diagonally
* you can't move through the obstacle cell in the middle
* A single plan can branch into two or more plans, where more than one option exists for expansion, but separate plans may not merge together to occupy the same grid cell.

How many partial plans do you end up with after exploring all possible paths?

* 

15

* 

5

* 

4

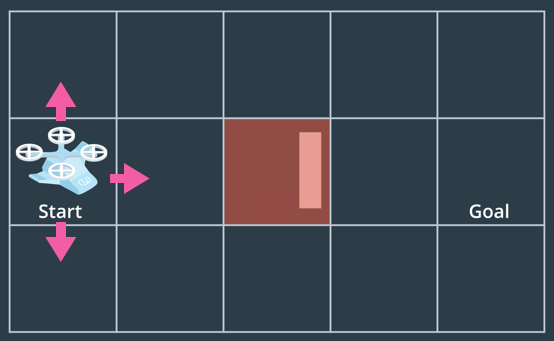
* it depends

SUBMIT

**Partial Plans Explained**

Figuring out how many partial plans are possible given a particular search space can be confusing. Here we'll step through the logic of how you can arrive at an answer for the example above.

Given that you can't move off the grid, and you can't move diagonally, from the start location you have three options, up, down or right like this:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)**

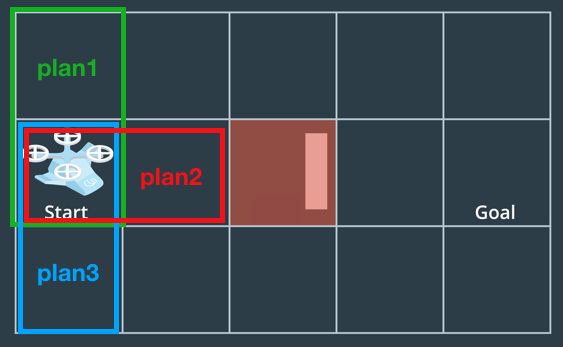
Each of these moves represents a unique partial plan. You could think of keeping track of the moves in each partial plan in lists like this:

plan1 = ['Start', 'up']

plan2 = ['Start', 'right']

plan3 = ['Start', 'down']

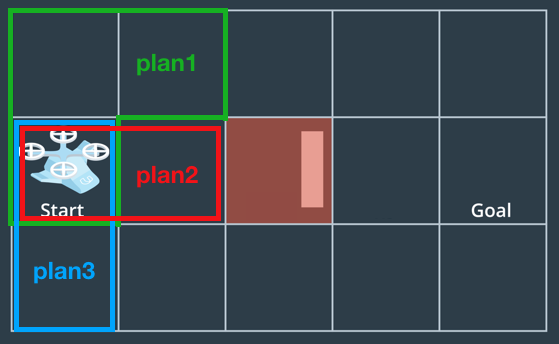
You can visualize each of these partial plans as now including specific cells in the grid like this:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)**

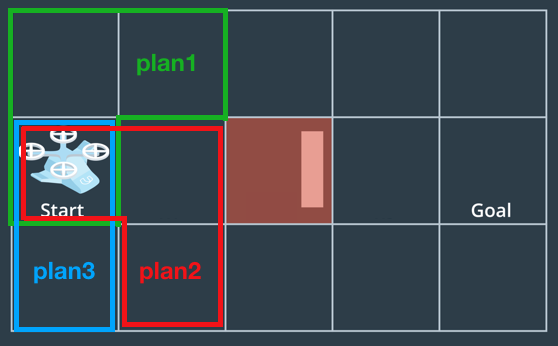
All plans include the starting cell and each includes one other cell as well at this point, the one above Start for plan1 (in green), to the right of Start for plan2 (in red) and down from Start for plan3 (in blue).

Now comes the choice of how to *expand* each of these plans. Given the rules of the game, you can see that you have one option with plan1, which is to move right. You have two options with plan2, which are to move up or down (can't move into the obstacle cell) and one option to move right from plan3 as well.

At this point, it doesn't matter which plan you expand first, so let's just go ahead and start by expanding plan1 in the only way possible, which is to move right like this:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)**

Now plan1 has been expanded to include a move to the right. What happens now if you try to expand plan2? Given that the cell above plan2 is already occupied by plan1, and separate plans can't merge together, there is only one remaining option for expanding plan2, which is to move down like this:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)**

Now your collection of partial plans looks like this:

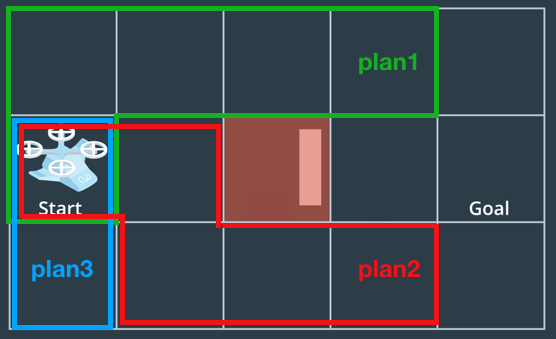
plan1 = ['Start', 'up', 'right']

plan2 = ['Start', 'right', 'down']

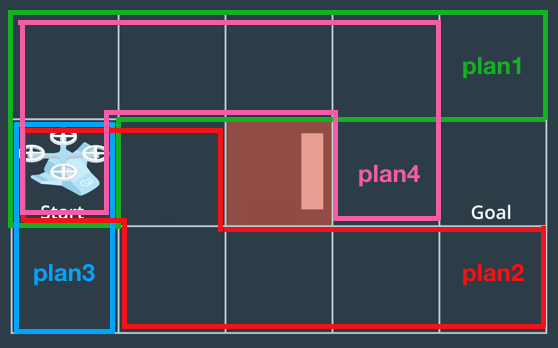
plan3 = ['Start', 'down']

What then becomes of plan3? At this point, the only direction (right) that plan3 had as an option to expand in is already occupied by plan2! This is not a problem for your search algorithm, it just means that there is nowhere else for plan3 to go. It's still a valid partial plan, but it's basically a dead plan because it can no longer be expanded.

plan1 and plan2 can continue marching happily toward the right for a couple moves like this:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)**

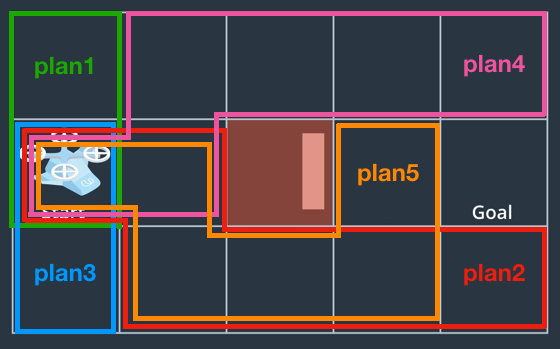
But now the landscape has changed! Both plan1 and plan2 have two options for directions to expand in. This means there is an opportunity for branching and forming a new partial plan. Suppose you first expand plan1 to the right. You could then add a new partial plan branching off of plan1 going down. Let's call that plan4. While we're at it let's also expand plan2 one more step to the right:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)**

Now, you have four plans in total and you've explored all cells in the grid. Any of these plans could be expanded to reach the goal and all of them represent valid paths that your vehicle could travel from start to goal.

You may have noticed, however, that there wasn't much rhyme or reason here in deciding which plan to expand next and in which direction. You can imagine a scenario where either plan1 or plan2 was left for dead instead of plan3.

In the case that you expand plan2 up and down before expanding plan1 or plan3, you'll end up with 5 total plans that look like this (plan5 could just as well take path above the obstacle):

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/d09207e7-f8d6-40a4-9147-9634d672e7cb)**

The truth is that there are many different strategies to consider when deciding how to expand your partial plans, and that's what we'll look at next!

1. **Breadth vs Depth**

[**https://www.youtube.com/watch?time\_continue=2&v=mtBKgqxjCyQ**](https://www.youtube.com/watch?time_continue=2&v=mtBKgqxjCyQ)

1. **Jupyter Notebooks**

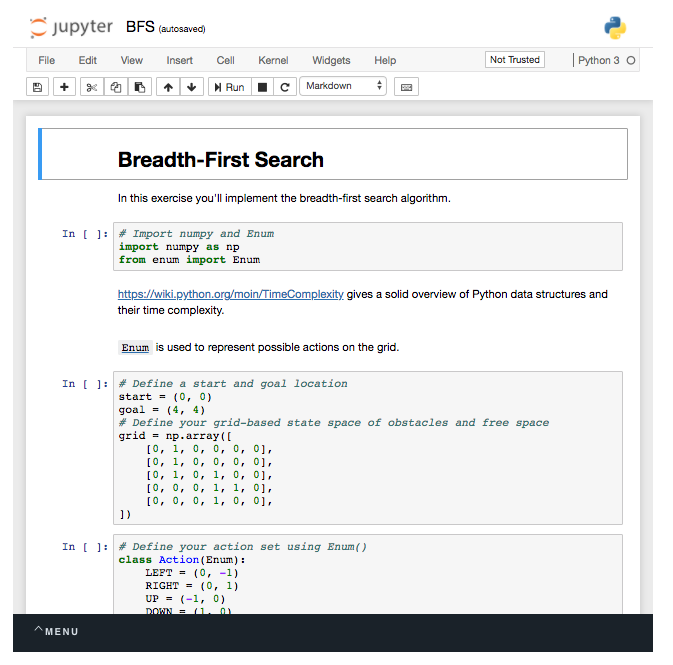
**Jupyter Notebooks**

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)**

For the next exercise, and many more to come, we'll be using **[Jupyter](http://jupyter.org/" \t "_blank)** notebooks embedded right in your browser. These notebooks are a great way to test code, plot up results and add notes, links and other supporting documentation right alongside your code in markdown cells.

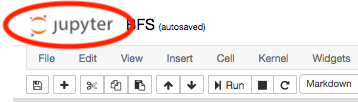
If you haven't used Jupyter notebooks before, check out this [**free Udacity lesson**](https://classroom.udacity.com/courses/ud1111/lessons/b15ba0a2-015d-4c5a-87ae-9efba2cabb43/concepts/75e1eee0-5f81-4d5b-a1ca-eaebe3c91759) as a way to get started. If you're interested, you can also check out the [**detailed docs**](https://jupyter-notebook.readthedocs.io/en/stable/notebook.html)on notebooks and all their functionality.

Notebooks in the classroom will look like this:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)**

You can select a cell by clicking on it, or by using the arrow keys to move up and down between cells. To run a cell either use shift-enter or click the "run" button in the toolbar. To add a new cell use the "+" icon and choose to make it code or markdown using the pulldown menu in the toolbar. To edit the contents of a markdown (text) cell, double click on it. You can download the entire notebook by selecting "File --> download as".

To see what else is in the directory containing the notebook (like a solution notebook or other supporting files), you can click on the Jupyter icon to show the contents of the home directory:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)**

**[Click here to see the contents of the home directory.](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)**

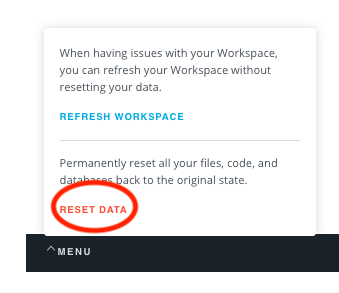
**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)**

**[Contents of the home directory](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)**

If at any point you get a 404 error in your notebook that looks like this:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)**

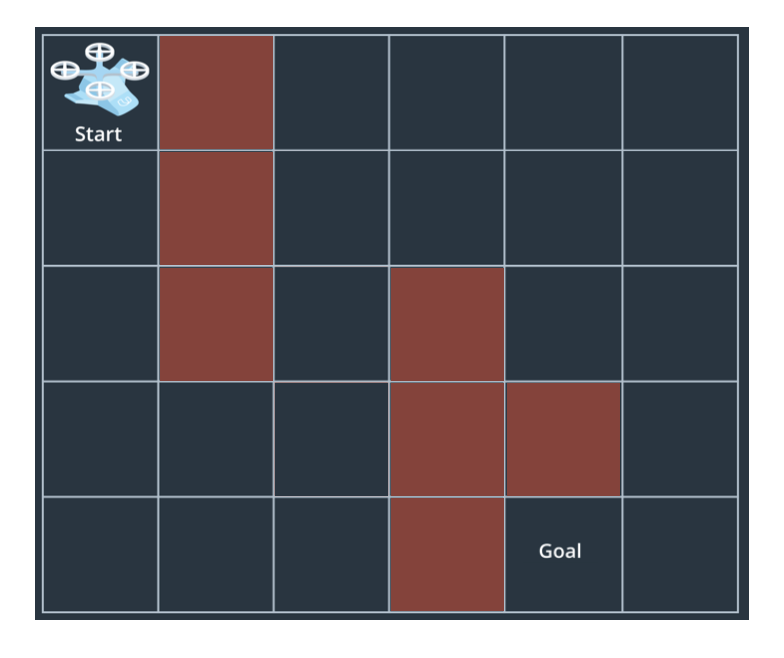
This most likely means that the notebook has been replaced with an updated version and you need to "Reset data" using the menu in the lower lefthand corner:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/f3b50398-54fa-40ef-bd60-4b6392f06a74)**

And with that you're ready to start using Jupyter notebooks to perform breadth-first search in the next exercise!

1. **Breadth-First Exercise**

**Breadth-First Search**

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/be3741ea-17f0-47c2-a405-3c6cd7e50019)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/be3741ea-17f0-47c2-a405-3c6cd7e50019)**

In this exercise, you'll implement breadth-first search to find a path from start to goal in a grid world like the one shown above. In this case, the grid represents your state space and the individual states that your vehicle can be in are simply positions within the grid. Using numpy in Python, you can represent this grid in the following manner:

**import** numpy **as** np

grid = np.array([

[0, 1, 0, 0, 0, 0],

[0, 1, 0, 0, 0, 0],

[0, 1, 0, 1, 0, 0],

[0, 0, 0, 1, 1, 0],

[0, 0, 0, 1, 0, 0],

])

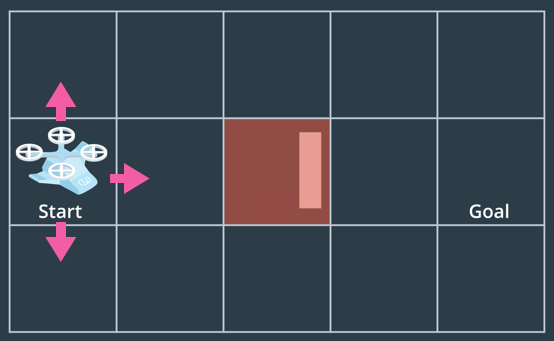
Within this numpy array, zeros represent the free space and ones represent obstacles. Positions within the grid are denoted just like you would in any matrix (or image) representation where (i, j) indicates row i column j, and the position (0, 0) is in the upper lefthand corner.

**Python data structures for search**

The search process involves keeping track of things like all your partial plans and positions visited so far. [**Python features a number of data structures**](https://docs.python.org/3/tutorial/datastructures.html) that you could use for keeping track of actions, partial plans and visited locations, but not all data structures are created equal! Depending on your use case, different structures will perform more or less efficiently.

In this exercise, you'll keep track of which cells you can expand into, your visited list and all your partial plans using a combination of a Python [**queue**](https://docs.python.org/3/library/queue.html), a [**set**](https://docs.python.org/3/tutorial/datastructures.html#sets) and a [**dictionary**](https://docs.python.org/3/tutorial/datastructures.html#dictionaries). The way this will work is that you'll keep track of all the cells that are possible to expand into within the queue, all the cells you've already visited in the set, and how you moved through the grid (your partial plans) in the dictionary.

For example, using the grid world from the previous exercise, you have three possible actions from the start location:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/be3741ea-17f0-47c2-a405-3c6cd7e50019)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/be3741ea-17f0-47c2-a405-3c6cd7e50019)**

The first step in the process is then to initialize a Queue() object and add the start location to it:

**from** queue **import** Queue

start = (1, 0) *# Location in (i, j) of the start location in the image above*

q = Queue()

q.put(start)

Next, initialize a set() object for your visited list and add the start location to it.

visited = set()

visited.add(start)

print(visited)

>>> {(1, 0)}

Then define an empty dictionary, where you'll record how you moved through the grid and a goal location, which in this example is (1, 4).

branch = {}

goal = (1, 4)

Next, you'll explore which actions are valid given your current position in the grid. In the first project, you used the **[Enum](https://docs.python.org/3/library/enum.html" \t "_blank)** class to keep track of the state of your vehicle and here we'll use it to keep track of the action set like this:

**from** enum **import** Enum

**class** **Action**(Enum):

LEFT = (0, -1)

RIGHT = (0, 1)

UP = (-1, 0)

DOWN = (1, 0)

**def** **\_\_str\_\_**(self):

**if** self == self.LEFT:

**return** '<'

**elif** self == self.RIGHT:

**return** '>'

**elif** self == self.UP:

**return** '^'

**elif** self == self.DOWN:

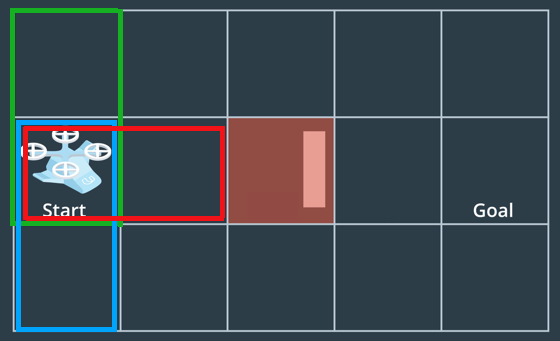
**return** 'v'

Here we've defined each action as a tuple containing the indices (i, j)corresponding to how that action moves you within the grid. We've also included a string representation for each action to be used later in visualizing the path. You could do something similar with a dictionary but using an Enum object is a nice clean way of keeping track of your actions and other associated properties like, in this case, a string representation of each action.

So in the example case, valid actions are UP, DOWN and RIGHT, corresponding to movements of (-1, 0), (1, 0) and (0, 1), respectively. Or in code:

valid = [Action.UP, Action.RIGHT, Action.DOWN]

The next thing to do is expand using each of these actions:

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/be3741ea-17f0-47c2-a405-3c6cd7e50019)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/be3741ea-17f0-47c2-a405-3c6cd7e50019)**

You'll find the grid locations of these new cells one at a time based on the original cell (start in this case) and the actions that took you to get there. You'll then step through and determine whether each new cell is already on your visited list. If so, ignore it, if not, add it to the queue and visited list, and record in your branchdictionary the cell you came from and action that took you there.

current\_node = start

**for** action **in** valid:

*# delta of performing the action*

da = action.value

next\_node = (current\_node[0] + da[0], current\_node[1] + da[1])

*# Check if the new node has been visited before.*

*# If the node has not been visited you will need to*

*# 1. Mark it as visited*

*# 2. Add it to the queue*

*# 3. Add how you got there to branch*

**if** next\_node **not** **in** visited:

visited.add(next\_node)

q.put(next\_node)

branch[next\_node] = (current\_node, action)

print(q.queue)

print(visited)

print(branch)

*# And this output looks like:*

>>> deque([(1, 0), (0, 0), (1, 1), (2, 0)])

>>> {(2, 0), (1, 0), (0, 0), (1, 1)}

>>> {(0, 0): ((1, 0), <Action.UP: (-1, 0)>), (1, 1): ((1, 0), <Action.RIGHT: (0, 1)>), (2, 0): ((1, 0), <Action.DOWN: (1, 0)>)}

You can employ the methods above to implement breadth-first search and when you finally arrive at the goal, it's time to retrace your steps through the branchdictionary to figure out how you got there! That looks something like this:

*# Retrace your steps*

path = []

n = goal

**while** branch[n][0] != start:

*# Append each new node to the path as you work your way back*

path.append(branch[n][1])

n = branch[n][0]

*# One last time to append the start location*

path.append(branch[n][1])

*# And reverse the order to make it a path from start to goal*

path = path[::-1]

**Breadth-first search exercise**

Now it's your turn! Remember the name of the game for breadth-first search is to keep track of visited cells and all your partial plans and always expand the shortest partial plan first. Check out the notebook below and complete the TODOs in the breadth\_first() function. Is your search method successful? What about if you modify the grid? Can you always find the shortest path? As an extra challenge, figure out how to convert your implementation from breadth-first search to depth-first search!

Good luck! And for a peek at our breadth-first solution, scroll to the link at the bottom of the notebook.

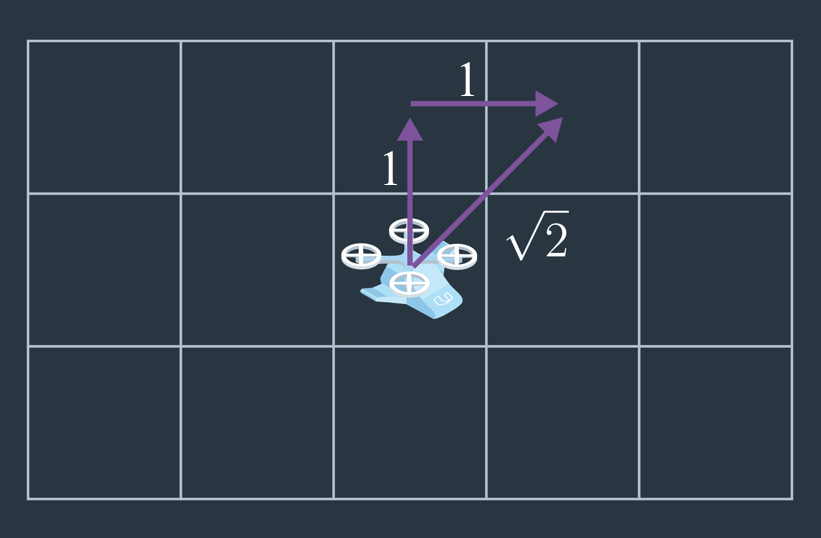
**BFS Workspace**

1. **Cost**

[**https://www.youtube.com/watch?v=qEYbn-jq7Qg**](https://www.youtube.com/watch?v=qEYbn-jq7Qg)

1. **Cost Exercise**

**The Cost of Action**

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/44fbeb21-92d8-4bee-97a8-5fc160e159ee)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/44fbeb21-92d8-4bee-97a8-5fc160e159ee)**

Before we get into incorporating a map into our planning solution, we'll delve a little deeper into the notion of cost. It's relatively intuitive to think of diagonal actions as having a higher cost than lateral or vertical motions between grid cells because the distance travelled is slightly longer. However, actions might have associated costs for other reasons as well.

Here, we'll expand the functionality of our Action() class to include the cost of each action using the following modifications:

**from** enum **import** Enum

**class** **Action**(Enum):

*# Assign the cost of each action as the third element in the tuple*

LEFT = (0, -1, 1)

RIGHT = (0, 1, 1)

UP = (-1, 0, 1)

DOWN = (1, 0, 1)

**def** **\_\_str\_\_**(self):

**if** self == self.LEFT:

**return** '<'

**elif** self == self.RIGHT:

**return** '>'

**elif** self == self.UP:

**return** '^'

**elif** self == self.DOWN:

**return** 'v'

*# Assign a new property that returns the cost of an action*

@property

**def** **cost**(self):

**return** self.value[2]

*# Assign a property that returns the action itself*

@property

**def** **delta**(self):

**return** (self.value[0], self.value[1])

Here we've added a third element each tuple defining actions, which is the cost of each action, and in this case is set to a value of one for all actions. We've also defined some new properties of the Action() class that return the cost and delta (the action itself). With this you have everything you need to perform uniform cost search!

**Cost Exercise**

In this exercise you'll assign a cost to each action and you can even expand your action set to include diagonal motions if you like!

Good luck! And for a peek at our solution scroll down to the link at the bottom of the notebook.

**Cost Workspace**

1. **Heuristics**

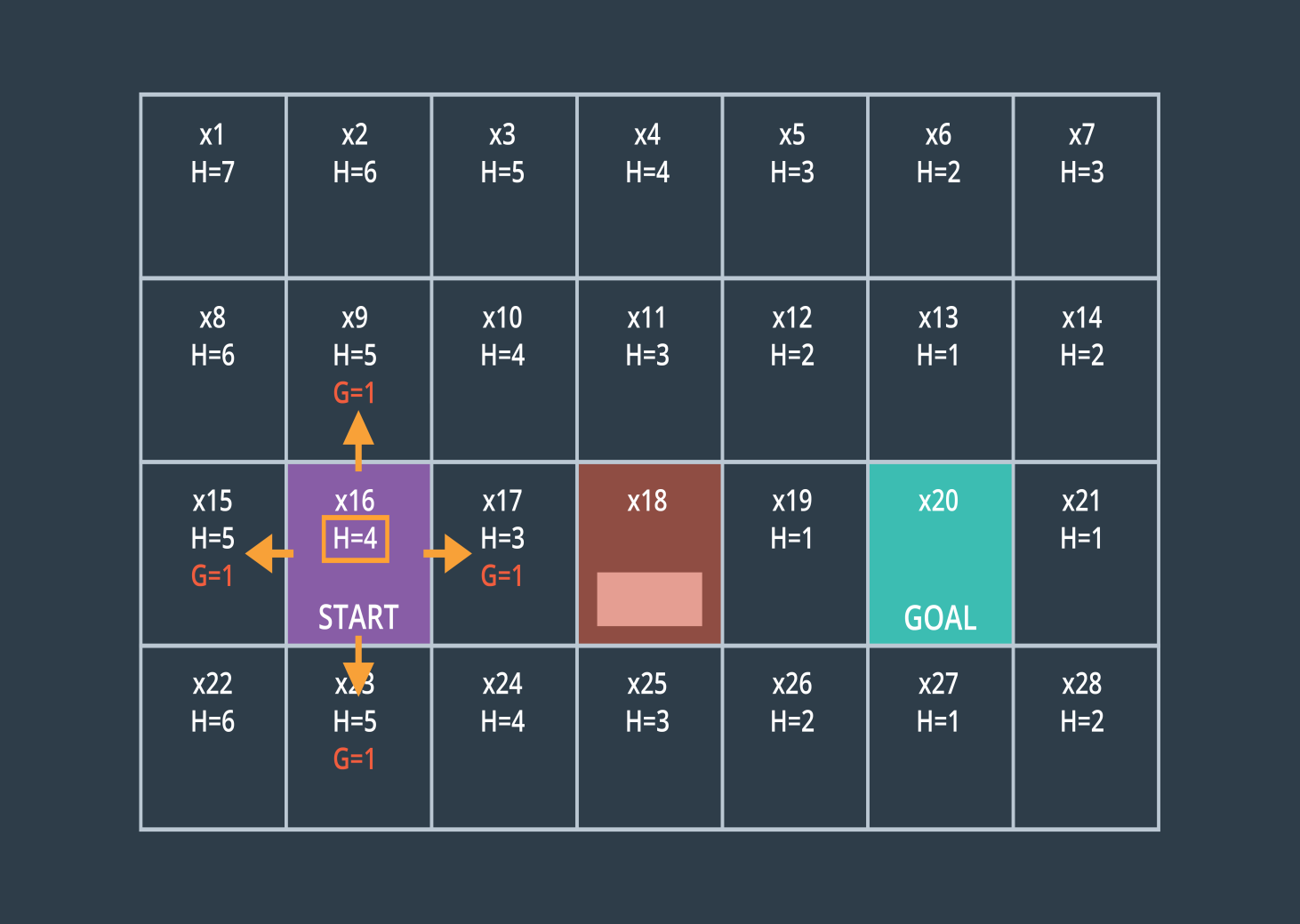
[**https://www.youtube.com/watch?v=qBAwN2r573Q**](https://www.youtube.com/watch?v=qBAwN2r573Q)

1. **A\***

[**https://www.youtube.com/watch?v=yT-8YRxSkqk**](https://www.youtube.com/watch?v=yT-8YRxSkqk)

1. **A\* Exercise**

**A\***

**[[](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/32aa8e06-004a-4138-801d-ef8228a8658c)](https://classroom.udacity.com/nanodegrees/nd787/parts/5aa0a956-4418-4a41-846f-cb7ea63349b3/modules/0c12632a-b59a-41c1-9694-2b3508f47ce7/lessons/c9b7c43f-6c90-401a-a2ce-ac10613e34f1/concepts/32aa8e06-004a-4138-801d-ef8228a8658c)**

The A\* algorithm is a simple yet elegant way of efficiently finding the lowest cost path from start to goal. In this exercise, you'll find the lowest cost plan by considering the cost of each partial plan and the value of the heuristic for all possible destinations at each step along the way.

In this exercise, you'll use a [**Python priority queue**](https://docs.python.org/3/library/queue.html#queue.PriorityQueue), which is a convenient way of maintaining a sorted queue. The PriorityQueue data structure allows you to quickly and efficiently select the lowest cost partial plan from your queue of all partial plans.

**A\* Exercise**

Your first TODO in the exercise below is to define a heuristic. You can use the Euclidean or Manhattan distance described in the previous video or think of another alternative but keep in mind your heuristic must be [***admissible* and *consistent***](https://en.wikipedia.org/wiki/Consistent_heuristic). You'll also define a cost for each action and modify to include diagonal actions if you like. Finally, you'll update the cost of each partial plan to reflect the sum of the cost of all actions in the plan plus the heuristic value.

Good luck! And for a peek at our solution scroll down to the link at the bottom of the notebook.

**A\* Workspace**

1. **Summary**

[**https://www.youtube.com/watch?time\_continue=1&v=MlvfEd8REJI**](https://www.youtube.com/watch?time_continue=1&v=MlvfEd8REJI)

**Additional Resources**

* Udacity's free [**Artificial Intelligence for Robotics Course**](https://classroom.udacity.com/courses/cs373) (taught by Sebastian Thrun) includes a lesson called "Search" which gives a slower-paced introduction to breadth first search and A star search.
* [**This web app**](https://qiao.github.io/PathFinding.js/visual/) provides nice interactive visualizations of various path planning algorithms in two dimensions.