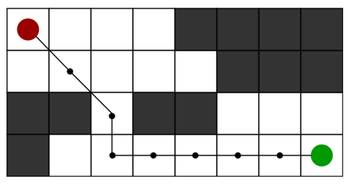
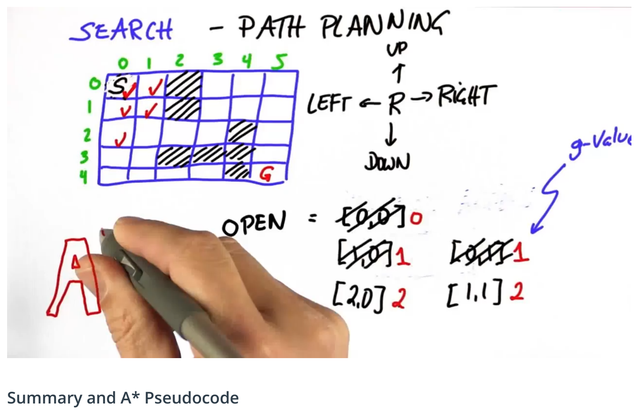
A* Search



A* Overview

In the previous lesson, Sebastian described how a general path search worked between two cells on a board, and you wrote C++ code to implement the board. In the next video, Sebastian will describe an improved way of searching, using an algorithm called A* search. This is the algorithm you will use for the implementation of your project.

After the video, there is an additional pseudocode outline of the A* algorithm that you will be following as you work through each exercise. Don't worry about remembering it all now, as the exercises will guide you through each step!



https://video.udacity-data.com/topher/2016/September/57d16349_a-artificial-intelligence-for-robotics/a-artificial-intelligence-for-robotics_720p.mp4

Summary and A* Pseudocode

This algorithm described by Sebastian is very similar to other search algorithms you may have seen before, such as **breadth-first search**, except for the additional step of computing a heuristic and using that heuristic (in addition to the cost) to find the next node.

The following is psuedocode for the algorithm described in the video above. Although the pseudocode shows the complete algorithm in a single function, we will split parts of the algorithm into separate functions in this lesson so you can implement them step-by-step in a sequence of exercises:

> **Search**(*grid*, *initial_point*, *goal_point*) :
>
> 1. Initialize an empty list of open nodes.
> 2. Initialize a starting node with the following:
>   - x and y values given by *initial_point*.
>   - g = 0, where g is the cost for each move.
>   - h given by the heuristic function (a function of the current coordinates and the goal).
> 3. Add the new node to the list of open nodes.
> 4. **while** the list of open nodes is nonempty:
>   1. Sort the open list by f-value
>   2. Pop the optimal cell (called the *current* cell).
>   3. Mark the cell's coordinates in the grid as part of the path.
>   4. **if** the *current* cell is the goal cell:
>   - return the *grid*.
>   5. **else**, expand the search to the *current* node's neighbors. This includes the following steps:
>   - Check each neighbor cell in the *grid* to ensure that the cell is empty: it hasn't been closed and is not an obstacle.
>   - If the cell is empty, compute the cost (g value) and the heuristic, and add to the list of open nodes.
>   - Mark the cell as closed.
> 5. If you exit the while loop because the list of open nodes is empty, you have run out of new nodes to explore and haven't found a path.

Summary

The A* algorithm finds a path from the start node to the end node by checking for open neighbors of the current node, computing a heuristic for each of the neighbors, and adding those neighbors to the list of open nodes to explore next. The next node to explore is the one with the lowest total cost + heuristic (g + h). This process is repeated until the end is found, as long as there are still open nodes to explore.