CODE: Check for Valid Neighbors

Check for Valid Neighbors

Lesson 3:

A* Search

2. Motion Planning

5. Coding the Shortest Path Algorithm

SEARCH

RESOURCES

CONCEPTS

☑ 3. Maze

⊻ 4. Maze 2

7. Lesson Code Structure

8. CODE: Starting A* Search

9. CODE: Writing the A* Heuristic

10. Pass by Reference in C++

11. CODE: Adding Nodes to the Ope...

12. CODE: Initialize the Open Vector

13. CODE: Create a Comparison Fun...

14. CODE: Write a While Loop for the...

15. CODE: Check for Valid Neighbors

17. CODE: Expand the A* Search to ...

19. CODE: Adding a Start and End to ...

21. How to Become More Proficient ...

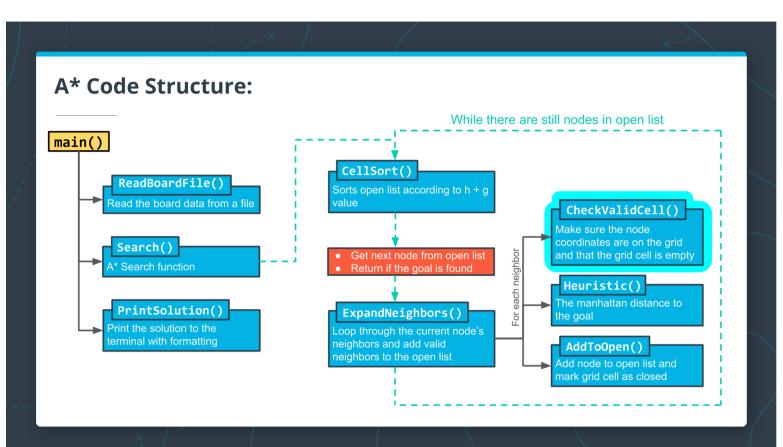
16. Constants

✓ 18. Arrays

☑ 20. Congratulations!!



https://video.udacity-data.com/topher/2019/February/5c7626b2_l2-check-for-valid-neighbors/l2-check-for-valid-neighbors_720p.mp4



Completing the CheckValidCell() function

Nice work, you are almost done with your program! The last part of the A* algorithm to be implemented is the part that adds neighboring nodes to the open vector. In order to expand your A* search from the current node to neighboring nodes, you first will need to check that neighboring grid cells are not closed, and that they are not an obstacle. In this exercise, you will write a function CheckValidCell that does exactly this.

To Complete This Exercise:

Write a function bool CheckValidCell that accepts two ints for the x and y coordinates and a reference to the grid. The function should do two things:
1. Check that the (x, y) coordinate pair is on the grid.
2. Check that the grid at (x, y) is kEmpty (this is the default case if the grid cell is not kClosed or a kObstacle). If both of these conditions are true, then CheckValidCell should return true. Otherwise, it should return false.

