UDACITY

< Return to Classroom

Build an OpenStreetMap Route Planner

REVIEW

CODE REVIEW 3

HISTORY

▼ CppND-Route-Planning-Project/src/route_planner.cpp 2

```
#include "route_planner.h"
#include <algorithm>
#include "my_utility.h"

using std::cout;

bool flag_debug = false;

RoutePlanner::RoutePlanner(RouteModel &model, float start_x, float start_y, float end_x, float end_y): r
    // Convert inputs to percentage:
    start_x *= 0.01;
    start_y *= 0.01;
    end_x *= 0.01;
    end_y *= 0.01;

    // TODO 2: Use the m_Model.FindClosestNode method to find the closest nodes to the starting and end
    // Store the nodes you find in the RoutePlanner's start_node and end_node attributes.
    this->start_node = &m_Model.FindClosestNode(start_x, start_y);
    this->end_node   = &m_Model.FindClosestNode(end_x, end_y);
}


// TODO 3: Implement the CalculateHValue method.
// Tips:
// - You can use the distance to the end_node for the h value.
// - Node objects have a distance method to determine the distance to another node.

float RoutePlanner::CalculateHValue(RouteModel::Node const *node) {
    // h_valie = distance from end_node
    return (node->distance(*end_node));
}


// TODO 4: Complete the AddNeighbors method to expand the current node by adding all unvisited neighbor
// Tips:
// - Use the FindNeighbors() method of the current_node to populate current_node.neighbors vector with
// - For each node in current_node.neighbors, set the parent, the h_value, the g_value.
// - Use CalculateHValue below to implement the h-Value calculation.
```

```
// - For each node in current_node.neighbors, add the neighbor to open_list and set the node's visited a

void RoutePlanner::AddNeighbors(RouteModel::Node *current_node) {
    current_node->FindNeighbors();

    for(RouteModel::Node* neighbor : current_node->neighbors){
        if (neighbor->visited == false){
            neighbor->parent  = current_node;
            neighbor->h_value = CalculateHValue(neighbor); // h_valie = distance from end_node
            neighbor->g_value = current_node->g_value + current_node->distance(*neighbor); // g_valie =
            neighbor->visited = true;
            this->open_list.push_back(neighbor);
        }
    }
}


// TODO 5: Complete the NextNode method to sort the open list and return the next node.
// Tips:
// - Sort the open_list according to the sum of the h value and g value.
// - Create a pointer to the node in the list with the lowest sum.
// - Remove that node from the open_list.
// - Return the pointer.

// My Function to compare F Value
bool CompareFValue(const RouteModel::Node *a, const RouteModel::Node *b)
```

SUGGESTION

Alternatively, you could use a [lambda](https://en.cppreference.com/w/cpp/language/lambda] expression. Lambdas local functions in other functions. You could, for example, capture additional parameters into the lambda by utilizing arguments:

```
std::sort(open_list.begin(), open_list.end(),
        [&](const auto n1, const auto n2)
        { return n1->g_value + n1->h_value < n2->g_value + n2->h_value; });
```

The `[&]` means that you could reference all of the local variables of the enclosing function inside of your lambda (

```
{
    float f1 = a->g_value + a->h_value; // f1 = g1 + h1
    float f2 = b->g_value + b->h_value; // f2 = g2 + h2
    return f1 > f2;
}

// My Function to sort open_list
void *NodeSort(std::vector<RouteModel::Node *> *open)
{
    sort(open->begin(), open->end(), CompareFValue);
}

// Main Function of TODO 5
RouteModel::Node *RoutePlanner::NextNode() {
    // sort list with f value
    NodeSort(&this->open_list);

    // get pointer
    RouteModel::Node* current = this->open_list.back();
    this->open_list.pop_back();

    // return
    return current;
}


// TODO 6: Complete the ConstructFinalPath method to return the final path found from your A* search.
// Tips:
// - This method should take the current (final) node as an argument and iteratively follow the
//   chain of parents of nodes until the starting node is found.
```

```
// - For each node in the chain, add the distance from the node to its parent to the distance variable.
// - The returned vector should be in the correct order: the start node should be the first element
//   of the vector, the end node should be the last element.

std::vector<RouteModel::Node> RoutePlanner::ConstructFinalPath(RouteModel::Node *current_node) {
    // Create path_found vector
    distance = 0.0f;
    std::vector<RouteModel::Node> path_found;

    // TODO: Implement your solution here.
    while(true){
        if (current_node->parent != nullptr){
            distance += current_node->distance(*current_node->parent);
            path_found.push_back(*current_node);
        }
        else{
            break;
        }
        // For next step, change current_node.
        current_node = current_node->parent;
    }

    // add start point
    path_found.push_back(*current_node);
```

SUGGESTION

We know that the last node we add to the final path will be the `start_node`. You could check your assumptions here

```
assert(current_node == start_node);
```

This will help to find bugs in complex projects by continuously checking your assumptions during the development p

```

    // reverse vector order
    std::reverse(begin(path_found), end(path_found));;

    // Multiply the distance by the scale of the map to get meters.
    distance *= m_Model.MetricScale();
    return path_found;

}


// TODO 7: Write the A* Search algorithm here.
// Tips:
// - Use the AddNeighbors method to add all of the neighbors of the current node to the open_list.
// - Use the NextNode() method to sort the open_list and return the next node.
// - When the search has reached the end_node, use the ConstructFinalPath method to return the final pa
// - Store the final path in the m_Model.path attribute before the method exits. This path will then be

void RoutePlanner::AStarSearch() {
    // in the main(), initial points are already set by { start_x, start_y, end_x, end_y }.
    RouteModel::Node *current_node = nullptr;

    // TODO: Implement your solution here.

    // current_node = this->start_node;
    // current_node->visited = true;
    // open_list.push_back(current_node);

    // while(this->open_list.size() > 0){
    //     AddNeighbors(current_node);
    //     current_node = NextNode();

    //     if ((current_node->x == end_node->x) && (current_node->y == end_node->y))
    //     {
    //         m_Model.path = ConstructFinalPath(current_node);
    //         return;
    //     }
```

```
    // }

    start_node->visited = true;
    open_list.push_back(start_node);

    while(open_list.size() >0){
        current_node = NextNode();
        if (current_node->distance(*end_node) == 0){
            m_Model.path = ConstructFinalPath(current_node);
            return;
        }
        else{
            AddNeighbors(current_node);
        }
    }

}
```

▸ **CppND-Route-Planning-Project/src_original/route_model.cpp** 1

▸ **CppND-Route-Planning-Project/build/thirdparty/googletest/googlemock/gtest/CMakeFiles/gtest_main.dir/src/gtest_main.cc.o**

▸ **CppND-Route-Planning-Project/build/thirdparty/googletest/googlemock/CMakeFiles/gmock_main.dir/src/gmock_main.cc.o**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googletest/src/gtest_main.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googletest/src/gtest.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googletest/src/gtest-typed-test.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googletest/src/gtest-test-part.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googletest/src/gtest-printers.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googletest/src/gtest-port.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googletest/src/gtest-matchers.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googletest/src/gtest-internal-inl.h**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googletest/src/gtest-filepath.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googletest/src/gtest-death-test.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googletest/src/gtest-all.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googlemock/src/gmock_main.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googlemock/src/gmock.cc**

▸ **CppND-Route-Planning-Project/thirdparty/googletest/googlemock/src/gmock-spec-builders.cc**

- CppND-Route-Planning-Project/thirdparty/googletest/googlemock/src/gmock-matchers.cc
- CppND-Route-Planning-Project/thirdparty/googletest/googlemock/src/gmock-internal-utils.cc
- CppND-Route-Planning-Project/thirdparty/googletest/googlemock/src/gmock-cardinalities.cc
- CppND-Route-Planning-Project/thirdparty/googletest/googlemock/src/gmock-all.cc
- CppND-Route-Planning-Project/build/CMakeFiles/route_planner.dir/src/route_planner.cpp.o
- CppND-Route-Planning-Project/build/CMakeFiles/OSM_A_star_search.dir/src/route_planner.cpp.o
- CppND-Route-Planning-Project/build/CMakeFiles/OSM_A_star_search.dir/src/my_utility.cpp.o
- CppND-Route-Planning-Project/thirdparty/pugixml/src/pugixml.hpp
- CppND-Route-Planning-Project/thirdparty/pugixml/src/pugiconfig.hpp
- CppND-Route-Planning-Project/src_original/route_planner.h
- CppND-Route-Planning-Project/src_original/route_planner.cpp
- CppND-Route-Planning-Project/src_original/route_model.h
- CppND-Route-Planning-Project/src_original/render.h
- CppND-Route-Planning-Project/src_original/render.cpp
- CppND-Route-Planning-Project/src_original/model.h
- CppND-Route-Planning-Project/src_original/model.cpp
- CppND-Route-Planning-Project/src_original/main.cpp
- CppND-Route-Planning-Project/src/route_planner.h
- CppND-Route-Planning-Project/src/route_model.h
- CppND-Route-Planning-Project/src/route_model.cpp
- CppND-Route-Planning-Project/src/render.h
- CppND-Route-Planning-Project/src/render.cpp
- CppND-Route-Planning-Project/src/my_utility.h
- CppND-Route-Planning-Project/src/my_utility.cpp

- CppND-Route-Planning-Project/src/model.h
- CppND-Route-Planning-Project/src/model.cpp
- CppND-Route-Planning-Project/src/main.cpp

RETURN TO PATH