Build an OpenStreetMap Route Planner

REVIEW | CODE REVIEW 2 | HISTORY

▸ src/main.cpp 1

▾ src/route_planner.cpp 1

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

RoutePlanner::RoutePlanner(RouteModel &model, float start_x, float start_y, float end_x, floa
    // 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 starti
    // Store the nodes you find in the RoutePlanner's start_node and end_node attributes.

    this->start_node = &model.FindClosestNode(start_x, start_y);
    this->end_node = &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) {
    return node->distance(*end_node);
}



// TODO 4: Complete the AddNeighbors method to expand the current node by adding all unvisite
// Tips:
// - Use the FindNeighbors() method of the current_node to populate current_node.neighbors ve
// - 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'

void RoutePlanner::AddNeighbors(RouteModel::Node *current_node) {
    current_node->FindNeighbors();
    for (auto &node : current_node->neighbors){
        node->parent = current_node;
        node->g_value = current_node->g_value + node->distance(*current_node);
        node->h_value = CalculateHValue(node);
        open_list.push_back(node);
        node->visited = true;
    }

}


// 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.

RouteModel::Node *RoutePlanner::NextNode() {
    std::sort(open_list.begin(), open_list.end(), [](const auto &a, const auto &b) {
        return a->g_value + a->h_value > b->g_value + b->h_value;
    });

    RouteModel::Node *node = open_list.back();
    open_list.pop_back();

    return node;
}


// TODO 6: Complete the ConstructFinalPath method to return the final path found from your A*
// Tips:
// - This method should take the current (final) node as an argument and iteratively follow t
//   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
// - The returned vector should be in the correct order: the start node should be the first e
//   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.
    path_found.push_back(*current_node);
    while (current_node->parent != nullptr)
    {
        path_found.push_back(*current_node->parent);
        distance += current_node->distance(*current_node->parent);
        current_node = current_node->parent;
    }

    std::reverse(path_found.begin(), path_found.end());
    distance *= m_Model.MetricScale(); // Multiply the distance by the scale of the map to ge
    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_
// - 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 th
// - Store the final path in the m_Model.path attribute before the method exits. This path wi

void RoutePlanner::AStarSearch() {
    RouteModel::Node *current_node = nullptr;

    // TODO: Implement your solution here.
    this->start_node->visited = true;
    open_list.push_back(this->start_node);
    while (open_list.size() > 0)
    {
        current_node = NextNode();
        if (current_node == this->end_node)
        {
            this->m_Model.path = ConstructFinalPath(current_node);
            return;
        }
        AddNeighbors(current_node);

    }



}
```

AWE SOME

You did good job on implementing these methods. Well done! For checking structure of code. Here is the link to some good practices for reference: https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-it

- thirdparty/pugixml/src/pugixml.hpp
- thirdparty/pugixml/src/pugiconfig.hpp
- thirdparty/googletest/googletest/src/gtest_main.cc
- thirdparty/googletest/googletest/src/gtest.cc
- thirdparty/googletest/googletest/src/gtest-typed-test.cc
- thirdparty/googletest/googletest/src/gtest-test-part.cc
- thirdparty/googletest/googletest/src/gtest-printers.cc
- thirdparty/googletest/googletest/src/gtest-port.cc
- thirdparty/googletest/googletest/src/gtest-matchers.cc
- thirdparty/googletest/googletest/src/gtest-internal-inl.h
- thirdparty/googletest/googletest/src/gtest-filepath.cc
- thirdparty/googletest/googletest/src/gtest-death-test.cc
- thirdparty/googletest/googletest/src/gtest-all.cc
- thirdparty/googletest/googlemock/src/gmock_main.cc
- thirdparty/googletest/googlemock/src/gmock.cc
- thirdparty/googletest/googlemock/src/gmock-spec-builders.cc

- thirdparty/googletest/googlemock/src/gmock-matchers.cc
- thirdparty/googletest/googlemock/src/gmock-internal-utils.cc
- thirdparty/googletest/googlemock/src/gmock-cardinalities.cc
- thirdparty/googletest/googlemock/src/gmock-all.cc
- src/route_planner.h
- src/route_model.h
- src/route_model.cpp
- src/render.h
- src/render.cpp
- src/model.h
- src/model.cpp
- build/thirdparty/googletest/googlemock/gtest/CMakeFiles/gtest_main.dir/src/gtest_main.cc.o
- build/thirdparty/googletest/googlemock/CMakeFiles/gmock_main.dir/src/gmock_main.cc.o
- build/CMakeFiles/route_planner.dir/src/route_planner.cpp.o
- build/CMakeFiles/OSM_A_star_search.dir/src/route_planner.cpp.o

RETURN TO PATH