Einführung in C++ – Übung 12
Testatgruppe A (Isaak)

Rasmus Diederichsen

18. Januar 2015

Aufgabe 12.1 Autopilot für den Fighter

src/util/PathPlanner.hpp

```
1  #ifndef PATH_H_
2  #define PATH_H_
3
4  #include <string>
5  #include <list>
6  #include "math/Vertex.hpp"
7
8  #include <boost/graph/adjacency_list.hpp>
9  #include <boost/graph/graphml.hpp>
10 #include <boost/graph/astar_search.hpp>
11
12
13 namespace asteroids
14 {
15
16    using namespace boost;
17    using namespace std;
18
19    typedef float cost;
20    typedef adjacency_list< listS, vecS, undirectedS, no_property,
          property<edge_weight_t, cost > > MutableGraph;
21    typedef property_map<MutableGraph, edge_weight_t>::type
          WeightMap;
22    typedef property_map<MutableGraph, vertex_index_t>::type
          IndexMap;
23    typedef MutableGraph::vertex_descriptor Vertex_t;
24    typedef MutableGraph::vertex_iterator vertex_iterator;
25    typedef MutableGraph::edge_descriptor Edge_t;
26    typedef pair<unsigned int, unsigned int> graph_edge;
27
28    /**
29     * @struct found_goal
30     * @brief 'Exception' to be thrown when the goal is found
31     */
32    struct found_goal
```

```
33      {
34         string s = "hooray!";
35      };
36
37      /**
38       * @class astar_goal_visitor
39       * @brief Visitor used during A* search
40       */
41      template<typename Vertex>
42         class astar_goal_visitor : public default_astar_visitor
43      {
44         public:
45            /**
46             * @brief Construct the visitor
47             * @param goal Vertex descriptor representing the goal
48             */
49            astar_goal_visitor(Vertex goal) : m_goal(goal) {}
50
51            /**
52             * @brief Called during the search when deciding whether a
                   vertex is
53             * part of the optimal path.
54             * @param u The current vertex in the search.
55             * @param g The graph containing it.
56             * @throws found_goal type to indicate abortion of the
                  search
57             */
58            void examine_vertex(Vertex u, const MutableGraph& g) {
59               if(u == m_goal)
60                  throw found_goal();
61            }
62         private:
63            Vertex m_goal; //< the goal vertex
64      };
65
66      /**
67       * @class distance_astar_heuristic
68       * @brief Functor to compute h(x) in astar search given vertex x
69       */
70      class distance_astar_heuristic : public astar_heuristic<
            MutableGraph, float>
71      {
72         public:
73            /**
74             * @brief Contruct the functor object
75             * @param goal Vertex descriptor representing the goal
76             * @param points Vector of all vertices to map descriptors
                   to their
77             * vectors.
78             */
79            distance_astar_heuristic(Vertex_t goal,vector<asteroids::
                  Vertex<float>> points) : points(points), m_goal(goal)
                  {}
80
81            /**
82             * @brief Compute distance heuristic for a vertex to the
                   goal.
```

```
83            * @param v Vertex descriptor for which the heuristic
                  should be
84            * computed.
85            * @return The distance heuristic (euclidean distance to
                  gaol)
86            */
87           float operator()(Vertex_t v)
88           {
89              asteroids::Vertex<float> diff = points[v] - points[
                    m_goal];
90              return sqrt(diff[0] * diff[0] + diff[1] * diff[1] +
                    diff[2] * diff[2]);
91           }
92        private:
93           Vertex_t m_goal; //< the goal
94           vector<asteroids::Vertex<float>> points; //< vector of all
                 vertices
95     };
96
97     /**
98      * @class edge_writer
99      * @brief A class to use during graph-to-dot-writing so that
          edge costs
100     * are printed
101     */
102    template <class WeightMap>
103       class edge_writer {
104          public:
105             /**
106              * @brief Construct an edge writer
107              * @param w A boost::property_map to map edge
                     descriptors to their
108              * weights.
109              */
110             edge_writer(WeightMap w) : wm(w) {}
111
112             /**
113              * @brief Write the weight of the current edge to a
                     stream
114              * @param out The stream the dotfile is written to
115              * @param e The current edge
116              */
117             template <class Edge> void operator()(ostream& out,
                    const Edge& e) const
118             {
119                out << "[label=\"" << wm[e] << "\"]"; // print in
                       dot syntax
120             }
121          private:
122             WeightMap wm;
123       };
124
125
126    /**
127     * @class PathPlanner
128     * @brief Class which can plan a shortest path in a weighted
          undirected
```

```
129      * graph.
130      */
131     class PathPlanner {
132
133        public:
134           /**
135            * @brief Construct a PathPlanner which reads from a file
                   containing a
136            * graph definition.
137            * @param mapfile The file containing the vertices and
                   edges
138            */
139           PathPlanner(string mapfile);
140
141           /**
142            * @brief Computes the shortest path from a start to a
                   goal vertex
143            * @param position The current fighter position
144            * @param s The start vertex
145            * @param e The goal vertex
146            * @return A list with float Vectors being the shortest
                   route.
147            */
148           list<Vertex<float> > getPath(Vertex<float> position,
                   string s, string e);
149           /**
150            * @brief Get the list of all vertices in the graph
151            * @return a Vector of all vertices in the graph in the
                   form of float
152            * Vetors.
153            */
154           vector<Vertex<float> > getNavpoints();
155
156           /**
157            * @brief Get the vector containing all edges as std::
                   pairs of indices.
158            * @param mapfile The file to read from
159            * @param pos Position in the file at which edge
                   definitions begin
160            * @return A vector containing all edges
161            */
162           vector<graph_edge> getEdgeList(const string mapfile,
                   streampos& pos) const;
163
164           /**
165            * @brief Print the graph to a dotfile
166            */
167           void printGraph();
168        private:
169           int num_vertices; //< number of vertices in graph
170           vector<Vertex<float> > navPoints; //< vector if all
                   vertices
171           map<string, int> nameMap; //< Map to get index of nav
                   point from name
172           map<int, string> indexMap; //< Map to get name of nav
                   point from index
173           vector<graph_edge> edges; //< All graph edged
```

```
174             MutableGraph g; //< The graph
175             WeightMap weightmap = get(edge_weight, g); //< The map
                    containing edge weights
176
177             /**
178              * @brief Compute and return all vertices in the graph
                     after their
179              * number has been determined.
180              * @param num Number of vertices (read from the first line
                     of the file)
181              * @param stream_pos Position in the file after number has
                     been read
182              * (will be updated to faciliate subsequent edge reading)
183              * @param mapfile The file containing the graph
184              * @return The vector of all vertices
185              */
186             vector<Vertex<float>> getVertexList(const int& num,
                    streampos& stream_pos, const string& mapfile);
187     };
188 }
189
190 #endif
```

src/util/PathPlanner.cpp

```
1  #include <iostream>
2  #include <map>
3  #include <math.h>
4  #include <string>
5  #include <sstream>
6  #include <boost/algorithm/string.hpp>
7
8  #include "PathPlanner.hpp"
9
10 namespace asteroids
11 {
12
13     vector<Vertex<float>> PathPlanner::getNavpoints()
14     {
15         return navPoints;
16     }
17     vector<graph_edge> PathPlanner::getEdgeList(const string mapfile
           , streampos& pos) const
18     {
19         ifstream file(mapfile);
20         vector<graph_edge> edges; // this will hold all the edges
21         if (file.good())
22         {
23             file.seekg(pos); // seek to beginning of edge definitions
24             while (!file.eof()) // BUG: Last edge added twice
25             {
26                 int u, v;
27                 file >> u >> v; // TODO: Some form of error checking is
                        needed
28                 edges.push_back(make_pair(u,v));
29             }
30         }
31         file.close();
```

```
32          return edges;
33      }
34
35      vector<Vertex<float>> PathPlanner::getVertexList(const int& num,
            streampos& stream_pos,
36           const string& mapfile)
37      {
38          vector<Vertex<float>> vertices;
39          ifstream file(mapfile);
40          if (file.good())
41          {
42              file.seekg(stream_pos); // first line after vertex number
                    definition
43              stringstream line_stream;
44              string line, name;
45              for (int i = 0; i < num ; i++)
46              {
47                  getline(file, line); // one vertex per line
48                  line_stream = stringstream(line); // wrap in stream to
                        extract name and coordinates
49                  line_stream >> name;
50                  nameMap[name] = i; // save name and index
51                  indexMap[i] = name;
52                  float x, y, z;
53                  line_stream >> x >> y >> z;
54                  vertices.push_back(Vertex<float>(x,y,z));
55              }
56          } else cerr << "Error. Could not read file." << endl;
57          stream_pos = file.tellg();
58          file.close();
59          return vertices;
60      }
61
62
63      void PathPlanner::printGraph()
64      {
65          ofstream o("graph");
66          if (o.good())
67          {
68              write_graphviz(o, g, default_writer(), edge_writer<
                    WeightMap>(get(edge_weight, g)));
69              o.close();
70              system("neato -Tsvg graph > graph.svg"); // TODO: Error
                    checking
71          }
72      }
73
74      std::list<Vertex<float> > PathPlanner::getPath(Vertex<float>
            position, std::string s, std::string e)
75      {
76          // Taken from
77          // http://www.boost.org/doc/libs/1_38_0/libs/graph/example/
                astar-cities.cpp
78          // example
79          vector<asteroids::Vertex_t> p(num_vertices);
80          vector<cost> d(num_vertices);
81          try
```

```
82          {
83              // call astar named parameter interface
84              astar_search
85                  (g, nameMap[s], // index of start vertex
86                   distance_astar_heuristic(nameMap[e], navPoints),
87                   predecessor_map(&p[0]).distance_map(&d[0]).visitor(
                         astar_goal_visitor<Vertex_t>(nameMap[e])));
88
89
90          } catch(found_goal f)
91          { // found a path to the goal
92              list<Vertex<float>> shortest_path;
93              list<string> shortest_path_names;
94              for(Vertex_t v = nameMap[e];; v = p[v])
95              {
96                  shortest_path.push_front(navPoints[v]);
97                  shortest_path_names.push_front(indexMap[v]);
98                  if(p[v] == v) break;
99              }
100             cout << "Path␣computed␣with␣vertices␣" << endl;
101             int index = 0;
102             for (list<Vertex<float>>::iterator it = shortest_path.
                    begin() ; it != shortest_path.end() ; it++, index++)
103                 cout << "\t" << indexMap[index] << endl;
104             shortest_path.push_front(position);
105             return shortest_path;
106         }
107     }
108
109
110     PathPlanner::PathPlanner (string mapfile)
111     {
112         ifstream file(mapfile);
113         stringstream s;
114         streampos pos;
115         string line;
116         if (file.good())
117         {
118             getline(file, line);
119             stringstream s(line);
120             s >> num_vertices;
121             pos = file.tellg();
122         }
123         file.close();
124         cout << "Number␣of␣vertices:␣" << num_vertices << endl;
125
126         navPoints = getVertexList(num_vertices, pos, mapfile);
127         edges = getEdgeList(mapfile, pos);
128         for (int i = 0; i < edges.size(); i++)
129         {
130             int u = edges[i].first;
131             int v = edges[i].second;
132             Edge_t e;
133             bool b;
134             tie(e,b) = add_edge(u, v, g);
135             Vertex<float> diff = navPoints[u] - navPoints[v];
136             weightmap[e] = sqrt(diff[0] * diff[0] + diff[1] * diff[1]
```

```
                + diff[2] * diff[2]);
137             cout << "Adding␣edge␣between␣" << indexMap[u] << "(" << u
                    << ")"
138                 << "␣and␣" << indexMap[v] << "(" << v << ")" << ",␣
                        weight␣is␣"
139                 << sqrt(diff[0] * diff[0] + diff[1] * diff[1] + diff[2]
                        * diff[2])
140                 << endl;
141         }
142         printGraph();
143     }
144
145 }
```