

```
#####
./addroutedialog.cpp
#####
1  /*
2   * Author: Lampalzer Konstantin
3   * Class: 5BHIF
4   * Date: 16.02.2019
5   */
6
7  #include "addroutedialog.h"
8  #include "ui_addroutedialog.h"
9  #include <QCompleter>
10
11  AddRouteDialog::AddRouteDialog(QWidget *parent) : QDialog(parent),
12                                                    ui(new Ui::AddRouteDialog)
13  {
14      ui->setupUi(this);
15      initGUI();
16  }
17
18  AddRouteDialog::~AddRouteDialog()
19  {
20      delete ui;
21  }
22
23  void AddRouteDialog::initGUI()
24  {
25      DbManager database = DbManager::getInstance();
26      QStringList airportList;
27      for (auto &airport : database.airports)
28      {
29          airportList << airport.name + " (" + airport.iata + ")";
30      }
31      QCompleter *airportCompleter = new QCompleter(airportList, this);
32      airportCompleter->setCaseSensitivity(Qt::CaseInsensitive);
33      ui->StartAirport->setCompleter(airportCompleter);
34      ui->EndAirport->setCompleter(airportCompleter);
35
36      // Airlines
37      QStringList airlineList;
38      for (auto &airline : database.airlines)
39      {
40          airlineList << airline.name;
41      }
42      QCompleter *airlineCompleter = new QCompleter(airlineList, this);
43      airlineCompleter->setCaseSensitivity(Qt::CaseInsensitive);
44      ui->Airline->setCompleter(airlineCompleter);
45  }
46
47  void AddRouteDialog::on_buttonBox_accepted()
48  {
49      DbManager database = DbManager::getInstance();
50      QString departure = ui->StartAirport->text().simplified();
51      QString destination = ui->EndAirport->text().simplified();
52      QString airline = ui->Airline->text().simplified();
53
54      departure = departure.left(departure.indexOf("(") - 1);
55      destination = destination.left(destination.indexOf("(") - 1);
56
57      int airport1 = database.getAirportId(departure);
58      int airport2 = database.getAirportId(destination);
59      int airlineId = database.getAirlineId(airline);
60
61      if (airport1 == 0 || airport2 == 0)
62      {
63          qDebug() << "Using debug mode";
64          airport1 = 4908; // Vienna
```

```

65      airport2 = 3699; // Palm Spings
66  }
67
68      DbManager::getInstance().addRoute(airport1, airport2, airlineId);
69
70      ui->StartAirport->setText("");
71      ui->EndAirport->setText("");
72      ui->Airline->setText("");
73  }
74
75  void AddRouteDialog::on_buttonBox_rejected()
76  {
77  }
78
79  #####
80  ./drawablemapwidget.cpp
81  #####
82  1  /*
83   * Author: Lampalzer Konstantin
84   * Class: 5BHIF
85   * Date: 16.02.2019
86   */
87
88  2  #include "drawablemapwidget.h"
89
90  3  #include "mainwindow.h"
91  4  #include <algorithm>
92
93  5  #include <cmath>
94
95  6  #include <QStringList>
96  7  #include <QPainter>
97  8  #include <QApplication>
98
99  9  DrawableMapWidget::DrawableMapWidget(QWidget *parent) : QWidget(parent)
100 {
101     resetPic();
102 }
103
104 10 void DrawableMapWidget::paintEvent(QPaintEvent *)
105 {
106     QPainter painter(this);
107     painter.drawPixmap(0, 0, pic);
108 }
109
110 11 QPoint DrawableMapWidget::airportToImg(Airport airport)
111 {
112     return latLonToPoint(airport.latitude, airport.longitude);
113 }
114
115 12 QPoint DrawableMapWidget::latLonToPoint(double lat, double lon)
116 {
117     return QPoint((lon + 180.0) * 4.0,
118                  (lat - 90.0) * -4.0);
119 }
120
121 13 void DrawableMapWidget::connectAirports(Airport from, Airport to, QColor color, Q
122 Painter &painter)
123 {
124     auto fromPoint = airportToImg(from);
125     auto toPoint = airportToImg(to);
126
127     auto directDistance = abs(from.longitude - to.longitude);
128     auto roundDistance = 360 - directDistance;
129
130     if (directDistance < roundDistance)
131     {
132         painter.setPen(QPen(QBrush{color}, 1));
133         painter.drawLine(fromPoint, toPoint);
134     }
135 }
136
137 #####
```

```

51     }
52     else
53     {
54         painter.setPen(QPen{QBrush{color}, 1});
55
56         auto leftSide = latLonToPoint((from.latitude + to.latitude) / 2, -180.0);
57         auto rightSide = latLonToPoint((from.latitude + to.latitude) / 2, 180.0);
58
59         if (from.longitude > 0)
60         {
61             painter.drawLine(fromPoint, rightSide);
62             painter.drawLine(leftSide, toPoint);
63         }
64         else
65         {
66             painter.drawLine(fromPoint, leftSide);
67             painter.drawLine(rightSide, toPoint);
68         }
69     }
70 }
71
72 void DrawableMapWidget::connectTheDots(std::vector<tuple<int, int>> routes, QColor
r color)
73 {
74     auto airports = DbManager::getInstance().airports;
75     QPainter painter(&pic);
76
77     // Need to go twice, as lines would be over text
78     for (auto route : routes)
79     {
80         auto fromAirport = airports[std::get<0>(route)];
81         auto toAirport = airports[std::get<1>(route)];
82
83         connectAirports(fromAirport, toAirport, color, painter);
84     }
85
86     for (auto route : routes)
87     {
88         auto fromAirport = airports[std::get<0>(route)];
89         auto toAirport = airports[std::get<1>(route)];
90
91         auto fromPoint = airportToImg(fromAirport);
92         auto toPoint = airportToImg(toAirport);
93
94         painter.setPen(QPen{QBrush{QColor{0, 0, 0}}, 6});
95         painter.drawText(fromPoint, fromAirport.iata);
96         painter.drawText(toPoint, toAirport.iata);
97     }
98
99     update();
100 }
101
102 void DrawableMapWidget::resetPic()
103 {
104     QPixmap l_pic(1440, 720);
105     QPainter painter(&l_pic);
106     QPixmap map(":/static/static/Earthmap.jpg");
107     map = map.scaled(1440, 720, Qt::KeepAspectRatioByExpanding, Qt::SmoothTransfo
rmation);
108     painter.drawPixmap(0, 0, map);
109
110     painter.setPen(QPen{QBrush{QColor{0, 255, 0, 200}}, 3});
111
112     for (auto airport : DbManager::getInstance().airports)
113     {
114         painter.drawPoint(airportToImg(airport));
115     }

```

```

116
117     pic = l_pic;
118 }
#####
./main.cpp
#####
1  /*
2  * Author: Lampalzer Konstantin
3  * Class: 5BHIF
4  * Date: 16.02.2019
5  */
6
7 #include "mainwindow.h"
8 #include <QApplication>
9
10 int main(int argc, char *argv[])
11 {
12     QApplication a(argc, argv);
13     MainWindow w;
14     w.show();
15
16     return a.exec();
17 }
#####
./mainwindow.cpp
#####
1  /*
2  * Author: Lampalzer Konstantin
3  * Class: 5BHIF
4  * Date: 16.02.2019
5  */
6
7 #include "mainwindow.h"
8 #include "ui_mainwindow.h"
9 #include <vector>
10 #include <future>
11 #include <QCompleter>
12 #include <chrono>
13 #include "customsearchalgorithm.h"
14 #include "breadthfirstsearchalgorithm.h"
15
16 MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent),
17                                         ui(new Ui::MainWindow)
18 {
19     ui->setupUi(this);
20     initGUI();
21 }
22
23 void MainWindow::initGUI()
24 {
25     QStringList airportList;
26     for (auto &airport : database.airports)
27     {
28         airportList << airport.name + " (" + airport.iata + ")";
29     }
30     QCompleter *airportCompleter = new QCompleter(airportList, this);
31     airportCompleter->setCaseSensitivity(Qt::CaseInsensitive);
32     ui->FromSearch->setCompleter(airportCompleter);
33     ui->ToSearch->setCompleter(airportCompleter);
34
35     // Airlines
36     QStringList airlineList;
37     for (auto &airline : database.airlines)
38     {
39         airlineList << airline.name;
40     }
41     QCompleter *airlineCompleter = new QCompleter(airlineList, this);

```

```

42     airlineCompleter->setCaseSensitivity(Qt::CaseInsensitive);
43     ui->AirlineSearch->setCompleter(airlineCompleter);
44 }
45
46 MainWindow::~MainWindow()
47 {
48     delete ui;
49 }
50
51 void MainWindow::fillFlightTable(vector<vector<int>> routes, bool sort)
52 {
53     vector<QString> flights;
54     for (auto &vec : routes)
55     {
56         QString flight;
57         for (int i{0}; i <= vec.size() - 1; i++)
58         {
59             if (i != 0)
60             {
61                 flight += " -> ";
62             }
63             QString airportName = database.getAirportName(vec[i]);
64             flight += airportName;
65         }
66         flights.push_back(flight);
67     }
68     if (sort) {
69         std::sort(flights.begin(), flights.end());
70     }
71
72     for (auto &flight : flights)
73     {
74         ui->flighttable->addItem(new QListWidgetItem(flight));
75     }
76 }
77
78 void MainWindow::on_pushButton_clicked()
79 {
80     ui->flighttable->clear();
81     ui->map->resetPic();
82
83     QString departure = ui->FromSearch->text().simplified();
84     QString destination = ui->ToSearch->text().simplified();
85     QString airline = ui->AirlineSearch->text().simplified();
86
87     departure = departure.left(departure.indexOf("(") - 1);
88     destination = destination.left(destination.indexOf("(") - 1);
89
90     int airport1 = database.getAirportId(departure);
91     int airport2 = database.getAirportId(destination);
92     int airlineId = database.getAirlineId(airline);
93
94     if (airport1 == 0 || airport2 == 0)
95     {
96         qDebug() << "Using debug mode";
97         airport1 = 4908; // Vienna
98         airport2 = 3699; // Palm Spings
99     }
100
101     BreadthFirstSearchAlgorithm searchAlgorithm;
102
103     auto start = std::chrono::high_resolution_clock::now();
104     vector<vector<int>> routes = searchAlgorithm.getRoutes(airport1, airport2);
105     auto finish = std::chrono::high_resolution_clock::now();
106     auto microseconds = std::chrono::duration_cast<std::chrono::microseconds>(fin
ish - start);
107

```

```

108     qDebug() << "TIME: " << microseconds.count() * 0.001;
109     fillFlightTable(routes, true);
110
111     auto newRoutes = splitRoutes(routes, airlineId);
112     if (airlineId == -1)
113     {
114         ui->map->connectTheDots(get<2>(newRoutes), QColor{82, 82, 255});
115     }
116     else
117     {
118         ui->map->connectTheDots(get<0>(newRoutes), QColor{255, 82, 82});
119         ui->map->connectTheDots(get<1>(newRoutes), QColor{82, 82, 255});
120         ui->map->connectTheDots(get<2>(newRoutes), QColor{82, 82, 82});
121     }
122 }
123
124 std::tuple<vector<tuple<int, int>>, vector<tuple<int, int>>, vector<tuple<int, in
t>>> MainWindow::splitRoutes(vector<vector<int>> routes, int airline)
125 {
126     vector<tuple<int, int>> airlineRoutes;
127     vector<tuple<int, int>> allianceRoutes;
128     vector<tuple<int, int>> otherRoutes;
129
130     int alliance = database.airlines[airline].alliance;
131
132     for (auto &route : routes)
133     {
134         for (int i{0}; i <= route.size() - 2; i++)
135         {
136             if (database.isConnected(route[i], route[i + 1], airline))
137             {
138                 airlineRoutes.push_back(make_tuple(route[i], route[i + 1]));
139             }
140             else if (database.isConnectedViaAlliance(route[i], route[i + 1], alli
ance))
141             {
142                 allianceRoutes.push_back(make_tuple(route[i], route[i + 1]));
143             }
144             else
145             {
146                 otherRoutes.push_back(make_tuple(route[i], route[i + 1]));
147             }
148         }
149     }
150
151     return std::make_tuple(airlineRoutes, allianceRoutes, otherRoutes);
152 }
153
154 void MainWindow::on_actionAbout_triggered()
155 {
156     QMessageBox qMessageBox;
157     qMessageBox.setText("<h3>Name:</h3> Konstantin Lampalzer<br>"
158                       "<h3>Klasse:</h3> 5BHIF");
159     qMessageBox.exec();
160 }
161
162 void MainWindow::on_actionAdd_Route_triggered()
163 {
164     addRouteDialog.show();
165 }
166
167 void MainWindow::on_multiSearchButton_clicked()
168 {
169     QStringList airportNames = ui->multiSearch->text().split(' ');
170     QString airline = ui->AirlineSearch->text().simplified();
171     int airlineId = database.getAirlineId(airline);
172

```

```

173     vector<int> airports;
174     for (QString airport : airportNames)
175     {
176         airport = airport.simplified();
177         int airportId = database.getAirportId(airport);
178         if (airportId == 0 || airportId == -1)
179         {
180             // TODO Add alert!
181             qDebug() << "Error in airport " << airport;
182             return;
183         }
184         airports.push_back(airportId);
185     }
186
187     BreadthFirstSearchAlgorithm searchAlgorithm;
188     vector<vector<int>> routes;
189
190     for (int i{0}; i <= airports.size() - 2; i++)
191     {
192         qDebug() << airports[i] << " " << airports[i+1];
193         auto singleSearch = searchAlgorithm.getRoutes(airports[i], airports[i+1])
;
194         routes.push_back(singleSearch[0]);
195     }
196     auto singleSearch = searchAlgorithm.getRoutes(airports[airports.size() - 1],
airports[0]);
197     routes.push_back(singleSearch[0]);
198
199     fillFlightTable(routes, false);
200
201     auto newRoutes = splitRoutes(routes, airlineId);
202     if (airlineId == -1)
203     {
204         ui->map->connectTheDots(get<2>(newRoutes), QColor{82, 82, 255});
205     }
206     else
207     {
208         ui->map->connectTheDots(get<0>(newRoutes), QColor{255, 82, 82});
209         ui->map->connectTheDots(get<1>(newRoutes), QColor{82, 82, 255});
210         ui->map->connectTheDots(get<2>(newRoutes), QColor{82, 82, 82});
211     }
212 }
#####
./addroutedialog.h
#####
1  /*
2  * Author: Lampalzer Konstantin
3  * Class: 5BHIF
4  * Date: 16.02.2019
5  */
6
7  #ifndef ADDRoutedIALOG_H
8  #define ADDRoutedIALOG_H
9
10 #include <QDialog>
11 #include "dbmanager.h"
12
13 namespace Ui
14 {
15     class AddRouteDialog;
16 }
17
18 class AddRouteDialog : public QDialog
19 {
20     Q_OBJECT
21
22 public:

```

```

23     explicit AddRouteDialog(QWidget *parent = 0);
24     ~AddRouteDialog();
25
26 private slots:
27     void on_buttonBox_accepted();
28
29     void on_buttonBox_rejected();
30
31 private:
32     Ui::AddRouteDialog *ui;
33
34     void initGUI();
35 };
36
37 #endif // ADDRoutedIALOG_H
#####
./airline.h
#####
1  /*
2  * Author: Lampalzer Konstantin
3  * Class: 5BHIF
4  * Date: 16.02.2019
5  */
6
7  #ifndef AIRLINE_H
8  #define AIRLINE_H
9
10 #include <QString>
11
12 class Airline
13 {
14 public:
15     int id;
16     QString name;
17     int alliance;
18
19     Airline(int _id, QString _name, int _alliance)
20     {
21         id = _id;
22         name = _name;
23         alliance = _alliance;
24     }
25
26     Airline() {}
27 };
28
29 #endif // AIRLINE_H
#####
./airport.h
#####
1  /*
2  * Author: Lampalzer Konstantin
3  * Class: 5BHIF
4  * Date: 16.02.2019
5  */
6
7  #ifndef AIRPORT_H
8  #define AIRPORT_H
9
10 #include <QString>
11
12 class Airport
13 {
14 public:
15     int id;
16     double latitude;
17     double longitude;

```

```

18   QString name;
19   QString iata;
20
21   Airport(int _id, double _latitude, double _longitude, QString _name, QString _i
ata)
22   {
23       id = _id;
24       latitude = _latitude;
25       longitude = _longitude;
26       name = _name;
27       iata = _iata;
28   };
29   Airport() {}
30   };
31 };
32
33 #endif // AIRPORT_H
#####
./breadthfirstsearchalgorithm.h
#####
1  /*
2   * Author: Lampalzer Konstantin
3   * Class: 5BHIF
4   * Date: 16.02.2019
5   */
6
7 #ifndef BREADTHFIRSTSEARCHALGORITHM_H
8 #define BREADTHFIRSTSEARCHALGORITHM_H
9
10 #include "searchalgorithm.h"
11 #include "queue"
12 #include "map"
13
14 class BreadthFirstSearchAlgorithm : public SearchAlgorithm
15 {
16 public:
17     BreadthFirstSearchAlgorithm() {}
18     std::vector<std::vector<int>> getRoutes(int start, int end)
19     {
20         // Possible solution:
21         // Find shortest amount of hops
22         // Do BFS till depth = min - 1
23         // Keep Track of the predecessors of all neighbours
24         // Take pred. of end node and backtrack all routes recursively
25         // AKA. recursively go into all pred. .
26         // This works, as pred. have to have some route to the start node
27
28         std::vector<std::vector<int>> routes;
29         int depth = getShortestDepth(start, end);
30
31         auto predecessors = generatePredecessors(start, depth);
32         routes = backtrack({end}, depth, end, start, predecessors);
33
34         for (auto &route : routes)
35         {
36             std::reverse(route.begin(), route.end());
37         }
38
39         return routes;
40     }
41
42 private:
43     std::vector<std::vector<int>> backtrack(std::vector<int> prev, int depth, int
start, int end, std::map<int, std::vector<int>> &predecessors)
44     {
45         std::vector<std::vector<int>> ret;
46

```

```

47     if (depth == 0)
48     {
49         if (start == end)
50         {
51             std::vector<int> newPrev = prev;
52             //newPrev.push_back(end);
53             ret.push_back(newPrev);
54         }
55     }
56     else
57     {
58         auto nbs = predecessors[start];
59         for (auto &airport : nbs)
60         {
61             if (std::find(prev.begin(), prev.end(), airport) != prev.end())
62             {
63                 continue;
64             }
65
66             std::vector<int> newPrev = prev;
67             newPrev.push_back(airport);
68
69             auto toConcat = backtrack(newPrev, depth - 1, airport, end, prede
cessors);
70
71             if (toConcat.size() != 0)
72             {
73                 ret.insert(ret.end(), toConcat.begin(), toConcat.end());
74             }
75         }
76         return ret;
77     }
78
79     std::map<int, std::vector<int>> generatePredecessors(int start, int goalDepth)
80     {
81         int depth{0};
82         std::map<int, std::vector<int>> result;
83         std::queue<int> currentLayer;
84         std::queue<int> nextLayer;
85         std::map<int, bool> visited;
86
87         currentLayer.push(start);
88         int next = start;
89
90         while (depth != goalDepth)
91         {
92             if (!visited[next])
93             {
94                 visited[next] = true;
95                 for (auto &airport : database.getNeighbours(next))
96                 {
97                     if (visited[airport] != true)
98                     {
99                         nextLayer.push(airport);
100                         result[airport].push_back(next);
101                     }
102                 }
103             }
104
105             currentLayer.pop();
106             if (currentLayer.empty())
107             {
108                 depth += 1;
109                 currentLayer = nextLayer;
110                 if (currentLayer.empty())
111                 {

```

```

112         return result;
113     }
114 }
115 next = currentLayer.front();
116 }
117
118 return result;
119 }
120
121 int getShortestDepth(int start, int end)
122 {
123     int depth{0};
124     std::queue<int> currentLayer;
125     std::queue<int> nextLayer;
126     std::map<int, bool> visited;
127
128     currentLayer.push(start);
129     visited[start] = true;
130     int next = start;
131     while (next != end)
132     {
133         for (auto &airport : database.getNeighbours(next))
134         {
135             if (visited[airport] != true)
136             {
137                 visited[airport] = true;
138                 nextLayer.push(airport);
139             }
140         }
141
142         currentLayer.pop();
143         if (currentLayer.empty())
144         {
145             depth += 1;
146             currentLayer = nextLayer;
147             if (currentLayer.empty())
148             {
149                 return -1;
150             }
151         }
152         next = currentLayer.front();
153     }
154     return depth;
155 }
156 };
157
158 #endif // BREADTHFIRSTSEARCHALGORITHM_H
#####
./customsearchalgorithm.h
#####
1  /*
2   * Author: Lampalzer Konstantin
3   * Class: 5BHIF
4   * Date: 16.02.2019
5   */
6
7 #ifndef CUSTOMSEARCHALGORITHM_H
8 #define CUSTOMSEARCHALGORITHM_H
9
10 #include "searchalgorithm.h"
11 #include <future>
12
13 class CustomSearchAlgorithm : public SearchAlgorithm
14 {
15 public:
16     CustomSearchAlgorithm();
17

```

```

18     std::vector<std::vector<int>> getRoutes(int start, int end)
19     {
20         std::vector<std::vector<int>> routes;
21         int depth{0};
22         do
23         {
24             routes = getRoutesInternal({start}, depth, start, end);
25             qDebug() << depth;
26             depth += 1;
27         } while (routes.size() == 0 && depth <= 4);
28         return routes;
29     }
30
31 private:
32     std::vector<std::vector<int>> getRoutesInternal(std::vector<int> prev, int de
pth, int start, int end)
33     {
34         std::vector<std::vector<int>> ret;
35
36         if (depth == 0)
37         {
38             if (database.isConnected(start, end))
39             {
40                 std::vector<int> newPrev = prev;
41                 newPrev.push_back(end);
42                 ret.push_back(newPrev);
43             }
44         }
45         else
46         {
47             auto nbs = database.getNeighbours(start);
48             std::vector<std::future<std::vector<std::vector<int>>>> futs;
49
50             for (auto &airport : nbs)
51             {
52                 if (std::find(prev.begin(), prev.end(), airport) != prev.end())
53                 {
54                     continue;
55                 }
56
57                 std::vector<int> newPrev = prev;
58                 newPrev.push_back(airport);
59
60                 if (depth == 4)
61                 {
62                     futs.push_back(std::async(&CustomSearchAlgorithm::getRoutesIn
ternal, this, newPrev, depth - 1, airport, end));
63                 }
64                 else
65                 {
66                     auto toConcat = getRoutesInternal(newPrev, depth - 1, airport
, end);
67
68                     if (toConcat.size() != 0)
69                     {
70                         ret.insert(ret.end(), toConcat.begin(), toConcat.end());
71                     }
72                 }
73
74                 for (auto &fut : futs)
75                 {
76                     auto toConcat = fut.get();
77                     ret.insert(ret.end(), toConcat.begin(), toConcat.end());
78                 }
79             }
80             return ret;
81         }

```

```

82  };
83
84  #endif // CUSTOMSEARCHALGORITHM_H
#####
./dbmanager.h
#####
1  /*
2   * Author: Lampalzer Konstantin
3   * Class: 5BHIF
4   * Date: 16.02.2019
5   */
6
7  #ifndef DBMANAGER_H
8  #define DBMANAGER_H
9
10 #include <QString>
11 #include <SqlDatabase>
12 #include <SqlQuery>
13 #include <SqlRecord>
14 #include <QDebug>
15 #include <vector>
16 #include <route.h>
17 #include <airport.h>
18 #include <airline.h>
19 #include <unordered_set>
20
21 class DbManager
22 {
23 public:
24     std::vector<std::vector<Route>> routes;
25     std::vector<std::map<int, bool>> connections;
26
27     std::vector<Airport> airports;
28     std::vector<Airline> airlines;
29
30     static DbManager &getInstance()
31     {
32         static DbManager instance;
33         return instance;
34     }
35
36     int getAirportCount()
37     {
38         QSqlQuery query("select count(*) as amount from Airport");
39         int amount = query.record().indexOf("amount");
40
41         while (query.next())
42         {
43             return query.value(amount).toInt();
44         }
45     }
46
47     int getAirlineCount()
48     {
49         QSqlQuery query("select count(*) as amount from Airline");
50         int amount = query.record().indexOf("amount");
51
52         while (query.next())
53         {
54             return query.value(amount).toInt();
55         }
56     }
57
58     int getAirportId(QString name)
59     {
60         std::vector<Airport> possibleAirports;
61         for (auto &airport : airports)

```

```

62     {
63         if (airport.name.contains(name) || airport.iata.contains(name))
64         {
65             possibleAirports.push_back(airport);
66         }
67     }
68
69     if (possibleAirports.size() != 1)
70     {
71         return -1;
72     } else {
73         return possibleAirports[0].id;
74     }
75 }
76
77 int getAirlineId(QString name)
78 {
79     for (auto &airline : airlines)
80     {
81         if (airline.name == name && airline.name != "")
82         {
83             return airline.id;
84         }
85     }
86     return -1;
87 }
88
89 bool isConnected(int id1, int id2)
90 {
91     if (connections[id1][id2] == true)
92     {
93         return true;
94     }
95     return false;
96 }
97
98 bool isConnected(int id1, int id2, int airline)
99 {
100     for (auto &route : routes[id1])
101     {
102         if (route.end == id2 && route.airline == airline)
103         {
104             return true;
105         }
106     }
107     return false;
108 }
109
110 bool isConnectedViaAlliance(int id1, int id2, int alliance)
111 {
112     for (auto &route : routes[id1])
113     {
114         if (route.end == id2 && airlines[route.airline].alliance == alliance)
115         {
116             return true;
117         }
118     }
119     return false;
120 }
121
122 QString getAirportName(int id)
123 {
124     return airports[id].name;
125 }
126
127 std::vector<int> getNeighbours(int id)
128 {

```

```

129         std::unordered_set<int> set;
130         std::vector<int> results;
131
132         for (auto &route : routes[id])
133         {
134             set.insert(route.end());
135         }
136         results.assign(set.begin(), set.end());
137
138         return results;
139     }
140
141     void addRoute(int startAirport, int endAirport, int airline)
142     {
143         routes[startAirport].push_back(Route(startAirport, endAirport, airline));
144         connections[startAirport][endAirport] = true;
145
146         //QSqlQuery query;
147         //query.prepare("insert into routes values (?, ?, ?)");
148         //query.bindValue(startAirport, endAirport, airline);
149         //query.exec();
150     }
151
152 private:
153     QSqlDatabase m_db;
154
155     DbManager()
156     {
157         m_db = QSqlDatabase::addDatabase("QSQLITE");
158         m_db.setDatabaseName("./AirlineRoutes.db");
159
160         if (!m_db.open())
161         {
162             qDebug() << "Error: connection with database fail";
163         }
164         else
165         {
166             qDebug() << "Database: connection ok";
167         }
168
169         loadRoutes();
170         loadAirports();
171         loadAirlines();
172     }
173
174     void loadRoutes()
175     {
176         routes.resize(getAirportCount());
177         connections.resize(getAirportCount());
178
179         QSqlQuery query;
180         query.prepare("select * from route order by airport1");
181         query.exec();
182
183         auto a1Col = query.record().indexOf("airport1");
184         auto a2Col = query.record().indexOf("airport2");
185         auto airlineCol = query.record().indexOf("airline");
186
187         while (query.next())
188         {
189             int a1{query.value(a1Col).toInt()};
190             int a2{query.value(a2Col).toInt()};
191             routes[a1].push_back(Route(a1, a2, query.value(airlineCol).toInt()));
192             connections[a1][a2] = true;
193         }
194     }
195

```

```

196     void loadAirports()
197     {
198         airports.resize(getAirportCount() + 1);
199         QSqlQuery query;
200         query.prepare("select * from airport");
201         query.exec();
202
203         auto idCol = query.record().indexOf("id");
204         auto latitudeCol = query.record().indexOf("latitude");
205         auto longitudeCol = query.record().indexOf("longitude");
206         auto nameCol = query.record().indexOf("name");
207         auto iataCol = query.record().indexOf("iata");
208
209         while (query.next())
210         {
211             auto id{query.value(idCol).toInt()};
212             airports.at(id) = Airport(id,
213                                     query.value(latitudeCol).toDouble(), query.
214                                     value(longitudeCol).toDouble(),
215                                     query.value(iataCol).toString());
216         }
217
218         void loadAirlines()
219         {
220             airlines.resize(getAirlineCount() + 1);
221             QSqlQuery query;
222             query.prepare("select * from airline");
223             query.exec();
224
225             auto idCol = query.record().indexOf("id");
226             auto nameCol = query.record().indexOf("name");
227             auto allianceCol = query.record().indexOf("alliance");
228
229             while (query.next())
230             {
231                 auto id{query.value(idCol).toInt()};
232                 airlines.at(id) = Airline(id, query.value(nameCol).toString(), query.
233                 value(allianceCol).toInt());
234             }
235         };
236
237     #endif // DBMANAGER_H
#####
./dijkstrasearchalgorithm.h
#####
1  /*
2  * Author: Lampalzer Konstantin
3  * Class: 5BHIF
4  * Date: 16.02.2019
5  */
6
7  #ifndef DIJKSTRASEARCHALGORITHM_H
8  #define DIJKSTRASEARCHALGORITHM_H
9
10 #include "searchalgorithm.h"
11 #include <map>
12
13 class DijkstraSearchAlgorithm : public SearchAlgorithm
14 {
15     DijkstraSearchAlgorithm() {}
16     std::vector<std::vector<int>> getRoutes(int start, int end)
17     {
18     }
19 }

```



```

20 #endif // DIJKSTRASEARCHALGORITHM_H
#####
./drawablemapwidget.h
#####
1 /*
2  * Author: Lampalzer Konstantin
3  * Class: 5BHIF
4  * Date: 16.02.2019
5  */
6
7 #ifndef DRAWABLEMAPWIDGET_H
8 #define DRAWABLEMAPWIDGET_H
9
10 #include <QWidget>
11 #include "dbmanager.h"
12 #include <QApplication>
13 #include <QPoint>
14
15 class DrawableMapWidget : public QWidget
16 {
17     Q_OBJECT
18 public:
19     explicit DrawableMapWidget(QWidget *parent = nullptr);
20
21     void paintEvent(QPaintEvent *e);
22     QPoint airportToImg(Airport airport);
23     QPoint latLonToPoint(double lat, double lon);
24
25     void connectAirports(Airport from, Airport to, QColor color, QPainter &painter)
26 ;
27     void connectTheDots(std::vector<std::tuple<int, int>> routes, QColor color);
28     void resetPic();
29 signals:
30
31 public slots:
32
33 private:
34     QPixmap pic;
35 };
36
37 #endif // DRAWABLEMAPWIDGET_H
#####
./mainwindow.h
#####
1 /*
2  * Author: Lampalzer Konstantin
3  * Class: 5BHIF
4  * Date: 16.02.2019
5  */
6
7 #ifndef MAINWINDOW_H
8 #define MAINWINDOW_H
9
10 #include <QMainWindow>
11 #include <QFont>
12 #include "dbmanager.h"
13 #include <QMessageBox>
14 #include <vector>
15 #include <QTableWidgetItem>
16 #include <memory>
17 #include "addroutedialog.h"
18
19 using namespace std;
20
21 namespace Ui
22 {

```

```

23 class MainWindow;
24 }
25
26 class MainWindow : public QMainWindow
27 {
28     Q_OBJECT
29
30 public:
31     explicit MainWindow(QWidget *parent = nullptr);
32     void initGUI();
33     ~MainWindow();
34
35 private slots:
36     void on_pushButton_clicked();
37
38     void on_actionAbout_triggered();
39
40     void on_actionAdd_Route_triggered();
41
42     void on_multiSearchButton_clicked();
43
44 private:
45     Ui::MainWindow *ui;
46     AddRouteDialog addRouteDialog;
47     DbManager database = DbManager::getInstance();
48
49     QFont titleFont{"Helvetica", 18, QFont::Bold};
50     QFont standardFont{"Helvetica", 18};
51
52     void fillFlightTable(vector<vector<int>> routes, bool sort);
53     std::tuple<vector<tuple<int, int>>, vector<tuple<int, int>>, vector<tuple<int,
int>>> splitRoutes(vector<vector<int>> routes, int airline);
54 };
55
56 #endif // MAINWINDOW_H
#####
./route.h
#####
1 /*
2  * Author: Lampalzer Konstantin
3  * Class: 5BHIF
4  * Date: 16.02.2019
5  */
6
7 #ifndef ROUTE_H
8 #define ROUTE_H
9
10 #include <string>
11
12 class Route
13 {
14 public:
15     int start;
16     int end;
17     int airline;
18
19     Route(int _start, int _end, int _airline)
20     {
21         start = _start;
22         end = _end;
23         airline = _airline;
24     }
25
26     Route() {}
27 };
28 #endif // ROUTE_H
#####

```

```

./searchalgorithm.h
#####
1  /*
2   * Author: Lampalzer Konstantin
3   * Class: 5BHIF
4   * Date: 16.02.2019
5   */
6
7 #ifndef SEARCHALGORITHM_H
8 #define SEARCHALGORITHM_H
9
10 #include <vector>
11 #include "dbmanager.h"
12
13 class SearchAlgorithm
14 {
15     public:
16         DbManager database = DbManager::getInstance();
17         virtual std::vector<std::vector<int>> getRoutes(int start, int end) = 0;
18 };
19
20 #endif // SEARCHALGORITHM_H
#####
./addroutedialog.ui
#####
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ui version="4.0">
3     <class>AddRouteDialog</class>
4     <widget class="QDialog" name="AddRouteDialog">
5         <property name="geometry">
6             <rect>
7                 <x>0</x>
8                 <y>0</y>
9                 <width>400</width>
10                <height>300</height>
11            </rect>
12        </property>
13        <property name="windowTitle">
14            <string>Dialog</string>
15        </property>
16        <layout class="QGridLayout" name="gridLayout">
17            <item row="0" column="0">
18                <layout class="QFormLayout" name="formLayout">
19                    <item row="0" column="0">
20                        <widget class="QLabel" name="label">
21                            <property name="text">
22                                <string>Start Flughafen</string>
23                            </property>
24                        </widget>
25                    </item>
26                    <item row="1" column="0">
27                        <widget class="QLabel" name="label_2">
28                            <property name="text">
29                                <string>End Flughafen</string>
30                            </property>
31                        </widget>
32                    </item>
33                    <item row="0" column="1">
34                        <widget class="QLineEdit" name="StartAirport"/>
35                    </item>
36                    <item row="1" column="1">
37                        <widget class="QLineEdit" name="EndAirport"/>
38                    </item>
39                    <item row="2" column="0">
40                        <widget class="QLabel" name="label_3">
41                            <property name="text">
42                                <string>Fluglinie</string>

```

```

43         </property>
44     </widget>
45 </item>
46 <item row="2" column="1">
47     <widget class="QLineEdit" name="Airline"/>
48 </item>
49 </layout>
50 </item>
51 <item row="1" column="0">
52     <widget class="QDialogButtonBox" name="buttonBox">
53         <property name="orientation">
54             <enum>Qt::Horizontal</enum>
55         </property>
56         <property name="standardButtons">
57             <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
58         </property>
59     </widget>
60 </item>
61 </layout>
62 </widget>
63 </resources/>
64 <connections>
65 <connection>
66     <sender>buttonBox</sender>
67     <signal>accepted()</signal>
68     <receiver>AddRouteDialog</receiver>
69     <slot>accept()</slot>
70 </hints>
71 <hint type="sourcelabel">
72     <x>248</x>
73     <y>254</y>
74 </hint>
75 <hint type="destinationlabel">
76     <x>157</x>
77     <y>274</y>
78 </hint>
79 </hints>
80 </connection>
81 <connection>
82     <sender>buttonBox</sender>
83     <signal>rejected()</signal>
84     <receiver>AddRouteDialog</receiver>
85     <slot>reject()</slot>
86 </hints>
87 <hint type="sourcelabel">
88     <x>316</x>
89     <y>260</y>
90 </hint>
91 <hint type="destinationlabel">
92     <x>286</x>
93     <y>274</y>
94 </hint>
95 </hints>
96 </connection>
97 </connections>
98 </ui>
#####
./mainwindow.ui
#####
1 <?xml version="1.0" encoding="UTF-8"?>
2 <ui version="4.0">
3     <class>MainWindow</class>
4     <widget class="QMainWindow" name="MainWindow">
5         <property name="geometry">
6             <rect>
7                 <x>0</x>
8                 <y>0</y>

```

```
9      <width>1720</width>
10     <height>1100</height>
11   </rect>
12 </property>
13 <property name="sizePolicy">
14   <sizepolicy hsizeType="Expanding" vsizeType="Expanding">
15     <horstretch>0</horstretch>
16     <verstretch>0</verstretch>
17   </sizepolicy>
18 </property>
19 <property name="minimumSize">
20   <size>
21     <width>1720</width>
22     <height>1100</height>
23   </size>
24 </property>
25 <property name="windowTitle">
26   <string>MainWindow</string>
27 </property>
28 <widget class="QWidget" name="centralWidget">
29   <property name="sizePolicy">
30     <sizepolicy hsizeType="Expanding" vsizeType="Expanding">
31       <horstretch>0</horstretch>
32       <verstretch>0</verstretch>
33     </sizepolicy>
34   </property>
35   <layout class="QGridLayout" name="gridLayout">
36     <item row="0" column="0">
37       <layout class="QHBoxLayout" name="horizontalLayout" stretch="0,720">
38         <item>
39           <layout class="QFormLayout" name="formLayout">
40             <item row="2" column="0">
41               <widget class="QLabel" name="fromLabel">
42                 <property name="minimumSize">
43                   <size>
44                     <width>30</width>
45                     <height>0</height>
46                   </size>
47                 </property>
48                 <property name="text">
49                   <string>From:</string>
50                 </property>
51               </widget>
52             </item>
53             <item row="2" column="1">
54               <widget class="QLineEdit" name="FromSearch">
55                 <property name="text">
56                   <string/>
57                 </property>
58               </widget>
59             </item>
60             <item row="3" column="0">
61               <widget class="QLabel" name="toLabel">
62                 <property name="minimumSize">
63                   <size>
64                     <width>30</width>
65                     <height>0</height>
66                   </size>
67                 </property>
68                 <property name="text">
69                   <string>To:</string>
70                 </property>
71               </widget>
72             </item>
73             <item row="3" column="1">
74               <widget class="QLineEdit" name="ToSearch">
75                 <property name="text">
```

```
76     <string/>
77   </property>
78 </widget>
79 </item>
80 <item row="4" column="0">
81   <widget class="QLabel" name="airlineLabel">
82     <property name="minimumSize">
83       <size>
84         <width>30</width>
85         <height>0</height>
86       </size>
87     </property>
88     <property name="text">
89       <string>Airline:</string>
90     </property>
91   </widget>
92 </item>
93 <item row="4" column="1">
94   <widget class="QLineEdit" name="AirlineSearch">
95     <property name="text">
96       <string/>
97     </property>
98   </widget>
99 </item>
100 <item row="5" column="0">
101   <widget class="QLabel" name="label">
102     <property name="text">
103       <string/>
104     </property>
105   </widget>
106 </item>
107 <item row="5" column="1">
108   <widget class="QPushButton" name="pushButton">
109     <property name="text">
110       <string>Search</string>
111     </property>
112   </widget>
113 </item>
114 <item row="7" column="0">
115   <widget class="QLabel" name="label_2">
116     <property name="text">
117       <string>Multisearch</string>
118     </property>
119   </widget>
120 </item>
121 <item row="7" column="1">
122   <widget class="QLineEdit" name="multiSearch">
123     <property name="text">
124       <string>VIE; JFK</string>
125     </property>
126   </widget>
127 </item>
128 <item row="8" column="1">
129   <widget class="QPushButton" name="multiSearchButton">
130     <property name="text">
131       <string>Multi-search</string>
132     </property>
133   </widget>
134 </item>
135 <item row="6" column="0" colspan="2">
136   <spacer name="horizontalSpacer">
137     <property name="orientation">
138       <enum>Qt::Horizontal</enum>
139     </property>
140     <property name="sizeHint" stdset="0">
141       <size>
142         <width>40</width>
```

```
143         <height>20</height>
144     </size>
145 </property>
146 </spacer>
147 </item>
148 </layout>
149 </item>
150 </item>
151 <layout class="QVBoxLayout" name="verticalLayout_2" stretch="0,0">
152     <item>
153         <widget class="DrawableMapWidget" name="map" native="true">
154             <property name="minimumSize">
155                 <size>
156                     <width>1440</width>
157                     <height>720</height>
158                 </size>
159             </property>
160         </widget>
161     </item>
162     <item>
163         <widget class="QListWidget" name="flighttable">
164             <property name="minimumSize">
165                 <size>
166                     <width>720</width>
167                     <height>360</height>
168                 </size>
169             </property>
170         </widget>
171     </item>
172 </layout>
173 </item>
174 </layout>
175 </item>
176 </layout>
177 </widget>
178 <widget class="QMenuBar" name="menuBar">
179     <property name="geometry">
180         <rect>
181             <x>0</x>
182             <y>0</y>
183             <width>1720</width>
184             <height>22</height>
185         </rect>
186     </property>
187     <widget class="QMenu" name="menuMenu">
188         <property name="title">
189             <string>Menu</string>
190         </property>
191         <addaction name="actionAbout"/>
192         <addaction name="actionAdd_Route"/>
193     </widget>
194     <addaction name="menuMenu"/>
195 </widget>
196 <action name="actionAbout">
197     <property name="text">
198         <string>About</string>
199     </property>
200 </action>
201 <action name="actionAdd_Route">
202     <property name="text">
203         <string>Add Route</string>
204     </property>
205 </action>
206 </widget>
207 <layoutdefault spacing="6" margin="11"/>
208 <customwidgets>
209     <customwidget>
```

```
210     <class>DrawableMapWidget</class>
211     <extends>QWidget</extends>
212     <header>drawablemapwidget.h</header>
213     <container>1</container>
214 </customwidget>
215 </customwidgets>
216 <resources/>
217 <connections/>
218 </ui>
```