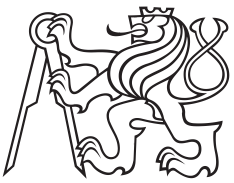
Bachelor's Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Airport Aircraft Movement and Control at Airport Including Airport Data Representation

Petr Hrych
Kybernetika a robotika, Systémy a řízení

2018
Supervisor: Mgr. Přemysl Volf Ph.D.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Hrych** Jméno: **Petr** Osobní číslo: **457434**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Systémy a řízení**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Pohyb a řízení letadel na letišti včetně datové reprezentace

Název bakalářské práce anglicky:

Airport Aircraft Movement and Control at Airport Including Airport Data Representation

Pokyny pro vypracování:

1. Seznamte se s problémem řízení letového provozu na letišti.
2. Seznamte se s problémem reprezentací dat na letišti.
3. Navrhněte reprezentaci dat pro letiště vhodnou pro pohyb letadel a řízení letového provozu.
4. Navrhněte základní model chování řídícího letového provozu na letišti.
5. Implement ujte navržené metody do systému AgentFly.
6. Proveďte experimenty s různými konfiguracemi.

Seznam doporučené literatury:

1. Aeronautical Information Exchange Model (AIXM) Official webpage - http://aixm.aero/
2. Nolan, M.: Fundamentals of air traffic control, Cengage learning, 2010
3. Atkin, J., Burke, E., and Ravizza, S.: The airport ground movement problem: Past and current research and future directions, ICRAT, 2010
4. Sislak D., Volf P., Pavlicek D., Pechoucek M.: AGENTFLY: multi-agent simulation of air-traffic management, Proceedings of the 20th European Conference on Artificial Intelligence, 2012

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Mgr. Přemysl Volf, Ph.D., centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **16.01.2018** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

Mgr. Přemysl Volf, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Acknowledgement / Declaration

I would like to thank my supervisior Mgr. Přemysl Volf, Ph.D., for his advice and support during the work. I would also like to express my sincere gratitude to Ing. Lukáš Koranda for his guidance and patience during consultations.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

In Prague, 25.5.2018

.....

Abstrakt / Abstract

Bakalářská práce se zabývá problémem řízení letového provozu na letišti. V úvodu je představena struktura letiště a způsob pozemního řízení. Dále je provedena rešerše možných zdrojů dat pro reprezentaci letiště a vytvořena datová struktura repezentující letiště, která umožňuje plánování trajektorií pro jednotlivá letadla. Za účelem plánování trajektorií je implementován algoritmus schopný plánovat trajektorie v závislosti na trajektoriích již naplánovaných. Pro účely simulace jsou naimplementovány moduly simulující pozemního řídího a pilota. Oba tyto moduly obsahují nástroje pro vyhýbání se kolizím a jsou spolu schopny komunikovat. Činnost plánovacího alogoritmu i obou modulů je testována pomocí různých simulačních scénárů. Celá implementace je zaintegrována do systému AgentFly.

Klíčová slova: Řízení letového provozu, simulace pozemního řídícího, datová reprezentace letiště, plánování trajektorií, AgentFly

In this Bachelor's thesis is introduced a problem with air traffic control at the airport. At the begging of this thesis the airport structure together with the ground contol is described. For the purposes of this thesis a research about a possible data sources have been made. A data structure representing an airport is designed. Trajectory planning is ensured by planning algorithm. This algorithm is able to find a trajectory, while avoiding collision with already planned trajectories. For an air traffic simulation a controller module and a pilot module are implemented. Both of these modules implement mechanisms for collision avoidance and comunication. The planning algorithm and both modules are tested in several simulation scenarios. This whole project is implemented and integrated into the AgentFly system.

Keywords: Air traffic control, simulation of the ground control, airport representation, trajectory planning, AgentFly

Contents /

Tables / Figures

Chapter 1
Introduction

In 2017 over 3.8 billion people used air transport. The number of flights performed globally by the airline industry in 2017 was 36.8 million[1]. Both these numbers are growing each year. As the number of passenger increases, airports need to increase their capacity and have to deal with problems associated with air traffic. Ground controllers handle air traffic on the airport. They are responsible for the navigation of all airplanes to runways, gates or other airport areas. Their work requires quick decision making, perfect knowledge of the airport and many other things. One wrong decision can lead to severe problems. In a better case, a problem leads to delay at the airport, which might be unpleasant and expensive, but it is nothing that can not be solved. In a worse case, a ground controller's mistake can lead to a collapse of all traffic at the airport. In the worst case scenario, one wrong decision can result in a loss of life.

The main task of this thesis is to design and create an airport structure and implement modules for simulating a ground controller and pilots. An airplane movement simulation and a pathfinding algorithm with a collision avoidance mechanism are necessary for a simulation. To fulfill these tasks, following problems must be solved:

1. Find a data format containing necessary information about airport

The basis of all of the simulations is an appropriate source of information. A data set must contain some specific information such as GPS positions of runways and other elements. Last but not least data must be in a format that can be easily loaded and processed.

2. Apropriate airport data representation

An airport can be represented in many forms. For planning algorithm, it is necessary to have a robust data structure representing airport.

3. Pathfinding algorithm

Implementation of pathfinding algorithm is crucial for this thesis. Appropriate planning of airport routes is the main task of ground controller. Moreover, pathfinding algorithm must be able to calculate noncollision ways. With this extension, the problem with pathfinding is no longer trivial, and a more sophisticated algorithm must be implemented.

4. Ground controller and pilot implementation

Pathfinding algorithm serves as a tool for the implemented ground controller. Ground controller is responsible for planning routes and must be able to respond to unexpected events. In case of problems, the ground controller must be able to replan routes.

Implementation of pilot modules includes movement at the airport and communication with the ground controller. Finally, pilots must be able to solve possible conflict situation on the airport.

5. Testing simulation on different scenarios

To verify the correct functionality of all modules, it is necessary to implement testing scenarios.

Chapter 2
Airport Definition

An airport is a place from which aircraft operate that usually has paved runways and maintenance facilities and often serves as a terminal[2].

The airport is composed of several elements. Main elements which appear in the simulation are runways, taxiways, guidance lines and aprons. All these elements can be seen at the figure 2.1.

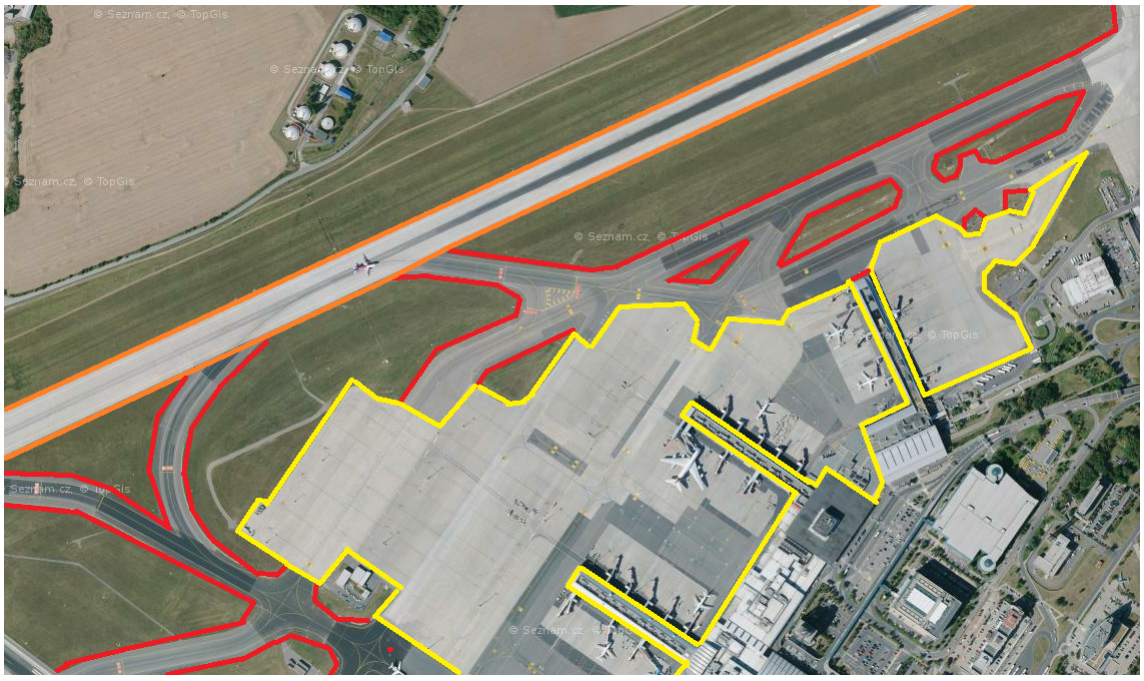


Figure 2.1. Václav Havel Airport Prague[1], runways - orange, taxiways - red, aprons - yellow

2.1 Runway

International Civil Aviation Organization (ICAO) defines runway as:

A defined rectangular area on a land aerodrome prepared for the landing and takeoff of aircraft.

Every runway must have a unique identification number. These numbers must meet following requirements [3]. Runways are identified by the number ranging from 01 to 36. This number stands for the magnetic azimuth of the runway's heading in dec degrees multiplied by 10. Numbers from 1 to 9 usually have 0 before them to avoid crosstalks. Because the runway can be used in both directions, it has to have a number for the opposite direction. The difference between these numbers must be 18 ($\sim 180°$). These

[1] `https://mapy.cz/letecka?x=14.2609674&y=50.1092952&z=16`

numbers may be followed by a letter. This letter (L - left, R - right, C - center) is used on larger airports with more parallel runways (e.q., runway named 05L in one direction is named 23R in the other direction). In a case of more than three parallel runways on one airport, the closest highest number is used as the identification number. For example, the airport in Los Angeles has 4 parallel runways named 6L-24R, 6R-24L, 7L-25R, 7R-25L.

The runway surface is often asphalt or concrete, but some runways have a natural surface such as dirt or gravel.

2.2 Guidance Line

A guidance line is a line drawn directly on the surface. Guidance lines help pilots with orientation at the airport. As can be seen at the figure 2.2, guidance lines are usually drawn in yellow. Guidance lines are on the lighter part of the figure. Lines on the darker part are called taxiway shoulder markings, and they are painted on areas which cannot take the weight of an airplane.

Figure 2.2. Guidance lines on airport [1]

2.3 Taxiway

Taxiways [4] are paths at an airport, which connect other elements such as runways, aprons, gates, hangars, terminals, and others. A movement on taxiways is restricted only to guidance lines.

Similarly to runways, also taxiways have conventions for names[5]. Taxiway's name is usually one letter. The letters I, O and Z are not used, to avoid confusion with the numbers 1, 0 and 2. The letter X is not used, because it indicates closure of portions of taxiway or runway. One taxiway can be divided into several parts. Every part has

[1] https://www.google.com/maps/@41.9742094,-87.9111392,90m/data=!3m1!1e3

its unique number which follows taxiway letter (e.q., A3). The naming of the taxiways begins on one side of the airport and carries on to the other extremity (e.g., from the east to the west or from the north to the south).

A typical speed on a taxiway is 20-30 knots (37-56 km/h).

Some taxiways have restrictions on maximum wingspan, that airplane driving through this taxiway can have. ICAO (International Civil Aviation Organisation) splits aircrafts into 6 categories [6]. These categories are presented in the table 2.1.

Code letter	Wingspan	Outer main gear wheel span	Typical aeroplane
A	$<$ 15 m	$<$ 4.5 m	PIPER PA-31
B	15 m but $<$ 24 m	4.5 m but $<$ 6 m	ATR42
C	24 m but $<$ 36 m	6 m but $<$ 9 m	BOEING 737-700
D	36 m but $<$ 52 m	9 m but $<$ 14 m	AIRBUS A-310
E	52 m but $<$ 65 m	9 m but $<$ 14 m	BOEING 777
F	65 m but $<$ 80 m	14 m but $<$ 16 m	BOEING 747-8

Table 2.1. Wingspan cathegories

2.4 Apron

The apron is an area at the airport and serves as a parking place for airplanes. The apron can be entered via taxiways, and as well as on them, a movement is limited only by guidance lines. On the apron, airplanes can be loaded, unloaded, boarded or refueled. Some airports have de-iced zones on aprons. A movement on the apron is directed by the ground controller. Larger airports (e.q., Charles de Gaulle Airport) have a unique controller for and apron area (Apron controller 2.5.3). Main elements located on aprons are stands and gates.

Stand is a designated, usually numbered, piece of prepared & marked ground where an aircraft parks.

Gate is the point at the terminal where a passenger commences boarding from. It does not matter, whether it is via the airbridge, walk up or bus transfer.

Figure 2.3. Apron on an airport [1]

[1] `https://www.airplane-pictures.net`

2.5 Ground Controller

The main job of the ground controller is guiding the airplane to the gate (in case of arrival) or the runway (in case of departure) [8]. When airplane approaches the airport, it is directed to the runway. The ground controller then guides the airplane from runway via taxiways to the gate. Because there are several airplanes moving to gates (or runways) at the same time, the ground controller must plan accordingly to avoid a collision or a "deadlock" situation (for example when the aircraft are on the same taxiway but in the opposite direction). On larger airports (e.g., John F. Kennedy International Airport) this is a complicated problem and can not be handled by a single person.

Large airports are divided into smaller parts. The ground controller is then responsible for this smaller part. An airport can be divided into three main parts (runways, taxiways, aprons). Each of these parts can have its controller or even more controllers.

2.5.1 Tower Control

The tower control main task is planning a runway traffic. That means timing departures, arrivals and communication with pilots, which need to cross an active runway. In most cases, departures and arrivals are alternating one at a time usually with the minimum period of two minutes.

Sometimes pilots must cross an active runway. In this case, the tower controller must be contacted, and the active runway cannot be crossed without permission. An airplane entering an active runway without a permission can lead to a collision with another departing or arriving airplane.

2.5.2 Ground Control

The ground control is responsible for navigating airplanes through the airport via taxiways. Ground control is usually located on the highest tower at the airport to have a good look at the whole airport.

2.5.3 Apron Control

Apron controllers are responsible for guiding an airplane from an apron entry points to their parking positions. Simultaneously when an airplane is ready for a departure, they guide it through the apron area to ground controllers.

On small airports (e.q., Václav Havel Airport Prague) the apron and the ground control can be handled by a single person. Bigger airport (e.q., Heathrow Airport) can have multiple apron controllers for a single apron.

2.6 Pilot

The main work of the pilots before or after landing is to drive through the airport on the desired point, where he can take off, pick up passengers, etc. Pilots communicate only with ground controllers. Pilots can communicate with each other only in emergency situations.

Chapter 3
Data Structures

The basis for creating a system that can simulate an airport traffic is an appropriate data structure. Unfortunately, there is no convention or a uniform format. For a simulation, several requirements for a data set are necessary. A data set must contain information about:

Runways - shape, ID, constraints, length, width

Taxiways - shape, ID, centerline coordinates, information about crossings with other taxiways, information about priorities at crossroads, crossings with runways or aprons, constraints

Aprons and Stands - position, constraints

3.1 Information Sources

For this thesis, several possible data sources have been searched.

`http://lis.rlp.cz` [9]
This is a website used by ATC (air traffic control) in the Czech Republic. It contains GPS coordinates of runways in the Czech Republic, and it lacks information about taxiways or other airport structures.

`https://openflights.org/data.html` [10]
OpenFlight is a tool for searching and calculating flights. It contains data about a majority of airports, but only basic information such as coordinates, name, state are provided.

`https://ext.eurocontrol.int/airport_corner_public/` [11]
The excellent source of information about airports in Europe. It contains a lot of information about airports, such as Airport Capacity, Traffic Forecast, Ongoing a Planned Activities, General Information, Weather Management, etc. However, this source does not contain information about structures.

`https://aeronavdata.com/what-we-do/aeronautical-navigation-database-andb/` [12]
Website of a navigation database creating company Aeronavdata. It offers data from more than 700 airports in the United States and the UK mostly in AIXM format. Except for a few examples, all data is paid.

`https://www.faa.gov/air_traffic/flight_info/aeronav/aero_data/` [13]
This website contains basic information about airports mainly in the USA. Information can be downloaded as HTML.

`https://ext.eurocontrol.int/aixmwiki_public/bin/view/Main/` [14]
This site contains a few examples of data in AIXM format, which have only limited amount of data.

`https://www.ead.eurocontrol.int/cms-eadbasic/opencms/en/home/` [15]

This website is a database of Eurocontrol. It contains different types of data sets. There are two types of data sets. Free data (only free account is needed - EAD Basic) or non-free data (EAD Pro). Access to this data is only for "Professional users". Prior to connection, an EAD Data User Agreement needs to be signed between organization and EUROCONTROL.

`http://aixm.aero` [16]

This is a website with information about the AIXM format. This site also contains demo data from Chicago O'Hare airport. AIXM format meets the requirements, so this data was selected for future work.

3.2 Data Format

Most data are available in HTML or pdf format. These formats are not suitable for this application, because they cannot be easily loaded and compiled.

Individual airports usually have their internal structures in XML, JSON or other formats. For example, Václav Havel Airport Prague has detailed construction plans of runways, taxiway, etc. in AutoCAD. The primary task of this project is creating a structure, that can handle all possible formats.

Main reasons why AIXM format is used in this simulation and its specifications are described in section 3.2.1.

3.2.1 AIXM Format

AIXM (Aeronautical Information Exchange Model) is a project of EUROCONTROL (European Organisation for the Safety of Air Navigation) and FAA (Federal Aviation Administration). The main objective of this project is creating a data structure describing a whole airport and relations between individual elements. There is a wiki with a detailed description of every single element on the following website.

https://ext.eurocontrol.int/aixmwiki_public/bin/view/Main/

The main advantage of AIXM format is a detailed description of a relationship between elements. For example, every guidance line has a list of other guidance lines connected with this one. The guidance line is in relation with another element, on which this guidance line is "locatedOn".

The AIXM format as it is known today was published in 2005 in version 4.5. Previous versions were used only for the needs of the European AIS Database. The big upgrade came in 2008 with version 5.0. Nowadays the most common version is AIXM 5.1 which was published in 2010. In mid-2018 the newest version - 5.2 should be released.

Since version 5.0 version AIXM format has three main components:

1. Data Model expressed in UML (Unified Modeling Language)

Unified Modeling Language is a modeling language used in a software engineering, that is intended to provide a standard way to visualize a design of a system. A system is usually interpreted as a diagram with relations between individual elements. The AIXM format has a documentation and related specifications for all main elements such as Runway, Taxiway, Apron, GuidanceLine.

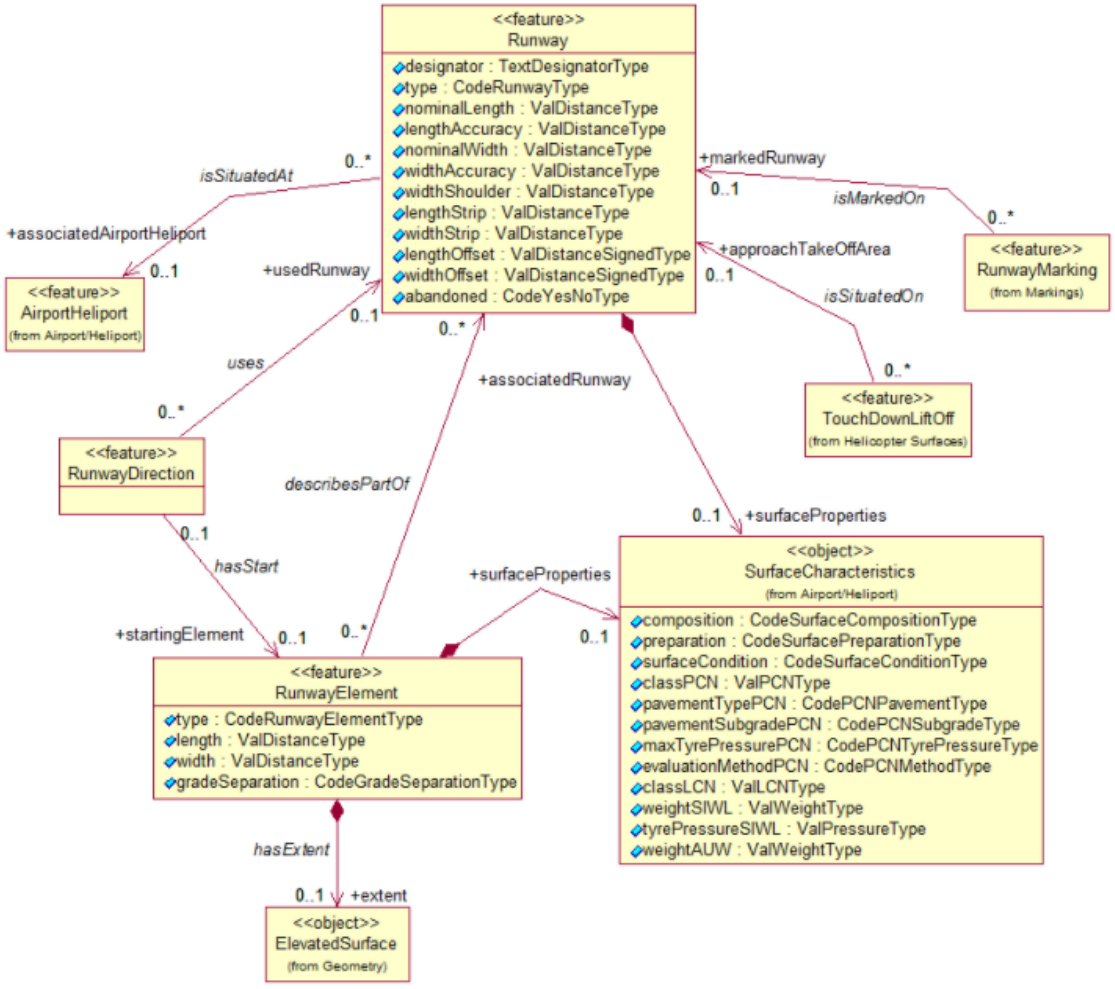


Figure 3.1. Runway described in UML [1]

2. GML-compliant XML Schema (Geography Markup Language)

All information about elements is saved in XML files. The structure of these files is defined by XSD (XML Schema Definition). XSD files can be found in attached files. Structure of XSD responds with the UML structure.

3. Temporality Concept that enables the encoding of both static and dynamic AIS data.

3.3 Airport Elements Used in Simulation

One of the main tasks of this thesis is to design a structure, which would be appropriate for trajectory planning, pathfinding algorithms, and a graphics visualization. Variables with $\star$ are loaded directly from the data set. Other variables are either parameters, or they have to be calculated in the process of creating the structure.

3.3.1 AixmAirportElement

This class is used for representing elements, whose coordinates form a plane, such as a runway, taxiway or apron. Each AixmAirportElement has following parameters:

[1] `http://aixm.aero/sites/aixm.aero/files/imce/AIXM51HTML/AIXM/Diagram_Runway.html`

id⋆ - Unique ID of this element.

coordinates⋆ - List of coordinates defining this element.

polygon - The unique element used in the AgentFly project for representing polygons (created from coordinates).

wingSpan⋆ - Airplanes are divided into classes based on their wingspan. Only airplanes with same or lower wingspan can enter this AixmAirportElement.

3.3.2 AixmRunway

This class is used for representing runways. The AixmRunway extends the AixmAirportElement, but it has some methods and parameters that are unique.

shorterSideCenter - A GpsPosition, that is in the middle of one of the shorter sides

length - Length of runway.

RUNWAY_START - A parameter defining the maximum relative distance from shorterSideCenter that point on runway needs to have to be considered as the start of the runway. In default settings, it is set on 0.2.

RUNWAY_END - A parameter defining the minimum relative distance from shorterSideCenter that point on runway needs to have to be considered as the end of the runway. In default settings, it is set on 0.8.

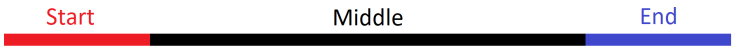


Figure 3.2. Runway divided into three parts

3.3.3 AixmTaxiway

Class representing taxiways. This class extends the AixmAirportElement class.

3.3.4 AixmApron

This class represents an apron. It extends the AixmAirportElement class and it has no unique parameters nor functions.

3.3.5 AixmVerticalStructure

This class represents vertical structures at the airport (e.g., terminals). As well as the AixmApron class, this class extends the AixmAirportElement class, but it adds neither parameters nor functions.

3.3.6 AixmGuidanceLine

Class representing guidance lines. They are crucial for creating a structure for navigation. Main parameters for AixmGuidanceLine are:

id⋆ - Unique ID of this AixmGuidanceLine.

coordinates⋆ - List of coordinates creating this AixmGuidanceLine.

owner[1] - An AixmAirportElement on which this AixmGuidanceLine is located.

connectedRunway - If an AixmGuidanceLine starts or ends on the runway this runway is saved in this variable.

[1] Data in AIXM format should provide this information. The data set from Chicago O'Haare lacked this information, and it had to be computed.

3.3.7 GraphSegment

Elements of this class are created from AixmGuidanceLines. The GraphSegment represents the part of the AixmGuidanceLine, which starts or ends in a point, where AixmGuidanceLine is connected to another AixmGuidanceLine. GraphSegments are cornerstones for a pathfinding algorithm. Their parameters are:

id - Unique ID of this GraphSegment.

segmentsTowardsStart - List of GraphSegments connected to this GraphSegment at its start.

segmentsTowardsEnd - List of GraphSegments connected to this GraphSegment at its end.

coordinates - List of coordinates creating this GraphSegment.

owner - An AixmGuidanceLine on which this GraphSegment is located.

length - Length of this GraphSegment in meters.

3.3.8 GraphSegmentOccupancy

Ground controller is responsible for proper use of GraphSegments. In no case, GraphSegment can be used by two airplanes in the same time and in the opposite direction. GraphSegmentOccupancy is used to avoid these situations. Parameters in this class are:

segment - Occupied GraphSegment.

direction - A direction in which is this GraphSegment used (0 if towards Start, 1 if towards End).

enterTime - The time when an airplane should enter this GraphSegment.

leaveTime - The time when an airplane should leave this GraphSegment.

entityID - ID of entity that uses this GraphSegment.

3.3.9 AixmAirport

The AixmAirport class represents the whole airport. This class contains information about all elements at the airport. Main parameters are:

taxiways - List of AixmTaxiways located at this airport.

runways - List of AixmRunways located at this airport.

aprons - List of AixmAprons located at this airport.

verticalStructures - List of AixmVerticalStructures located on this airport.

guidanceLines - List of AixmGuidanceLines located at this airport.

graphSegments - List of GraphSegments located on this airport (created from AixmGuidanceLines by the function **createConnections()** see section 3.4.3).

graphSegmentOccupancy - List of GraphSegmentOccupancies located on this AixmAirport (dynamically added and removed by the ground controller).

TOLERANCE - A parameter used to compensate inaccuracies in a data set. If a distance between two points is lower than this parameter, these points are considered as the same point.

ANGLE_TOLERANCE - A parameter used to compensate inaccuracies in a data set. If three points form an angle, this parameter determines the variance that this angle can have. For example, if three points should form 180° and ANGLE_TOLERANCE is 5, then the final angle can be between 175° - 185°.

RUNWAY_PATTERN - A string which contains a regular expression for runway name (used in the function **mergeRunways()**3.4.1).

TURNING_ANGLE - The maximum angle that airplanes can rotate on crossings.

At the figure 3.3 is a part of Chicago O'Hare International Airport created from the AIXM data set.

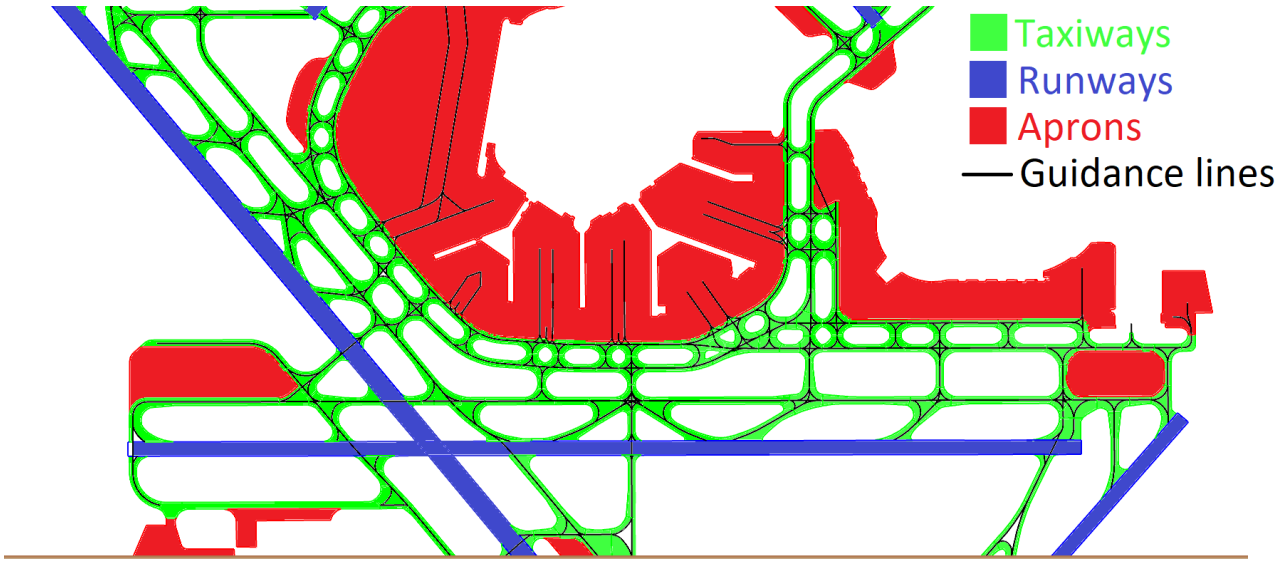


Figure 3.3. Chicago O'Hare International Airport created from the AIXM data set

Relations between elements can be seen at the figure 3.4. To make the figure more transparent, all constant parameters in the class AixmAirport are not shown.

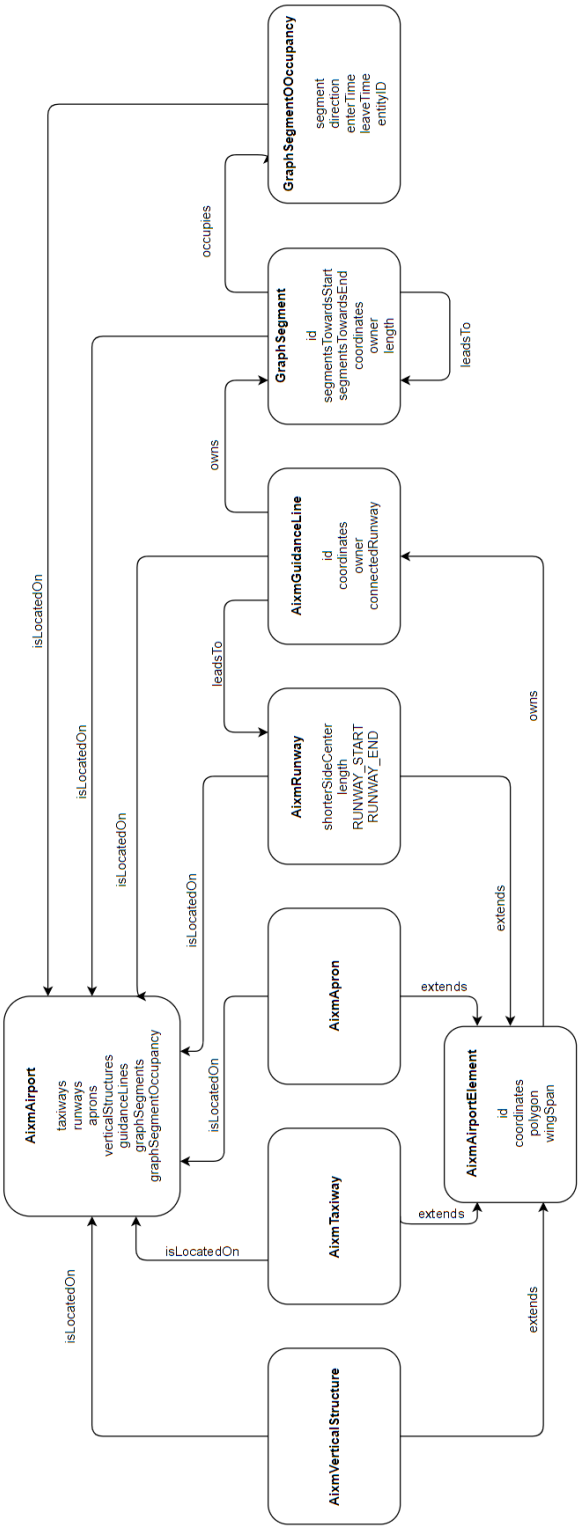


Figure 3.4. Relations between elements in a structure

3.4 Algorithm to Create the Structure

A raw data from a data set must be processed to create a structure suitable for a pathfinding algorithm. While following the pseudoalgorithm bellow, the data from the

AIXM format are processed, and the result is a structure formed by elements described in the section 3.3.

Pseudoalgorithm

1. load data from XML files
2. parse the XML file and create objects
3. create runways from runway elements (**mergeRunways()**)
4. find starts of all runways (**findRunwaysShorterSides()**)
5. create structure of GraphSegments for navigation (**createConnections()**)
6. assign Guidance lines to Taxiways, Runway or Aprons (**assignGuidanceLines()**)
7. make GraphSegments representing Stands available from all sides (**connectStands()**)
8. find which AixmGuidanceLines start or end at runways (**connectRunways()**)

3.4.1 mergeRunways()

The data in AIXM format contains information about runway elements (smaller parts of a runway, which are not overlapped by another runway element). Because of this, one runway can be made out of more runway elements. For easier work with runways, they have to be interpreted as one piece, with information about crossings with another runway.

In the AIXM data runway elements, that are attached directly to a runway have identification number identical to concerned runway followed by a number of the element (e.q., 9R-27L RE0). If an element is attached to more than one runway, his identification number is only a number of an element (e.q., RE30). This attribute of AIXM data is used in runway merging algorithm. All angles on the runway are expected to be approximately 90°.

The main goal of this algorithm is to find a list of coordinates that form runway.

```
runwayElement = airport.getRunwayElement();
list.add(runwayElement.getFirstPoint());
list.add(runwayElement.getSecondPoint());
prevPoint = runwayElement.getFirstPoint();
curPoint = runwayElement.getSecondPoint();
nextPoint = runwayElement.getThirdPoint();
while (list.get(0) != nextPoint)
  angle = computeAngle(prevPoint,curPoint,nextPoint)
  if (angle == 180)
    list.add(nextPoint)
    prevPoint = curPoint;
    curPoint = nextPoint;
    nextPoint = runwayElement.getNextPoint;
  else
//find a point on other runway elements that with prevPoint and curPoint forms an angle 180°
    newPoint = findPointOnOtherRunways();
    if (newPoint == null)
      list.add(nextPoint)
      prevPoint = curPoint;
      curPoint = nextPoint;
      nextPoint = runwayElement.getNextPoint;
    else
      runwayElement = newPoint.getRunwayElement();
      list.add(newPoint);
      prevPoint = curPoint;
      curPoint = newPoint;
      nextPoint = runwayElement.getNextPoint;
    end
  end
end
return list;
```

Figure 3.5. Pseudoalgorithm for merging runways

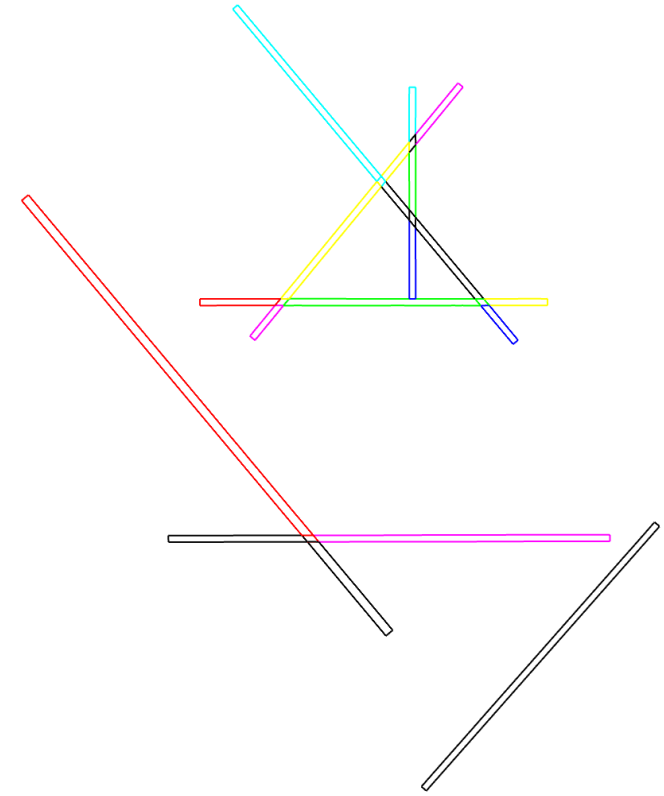


Figure 3.6. Runways before merging

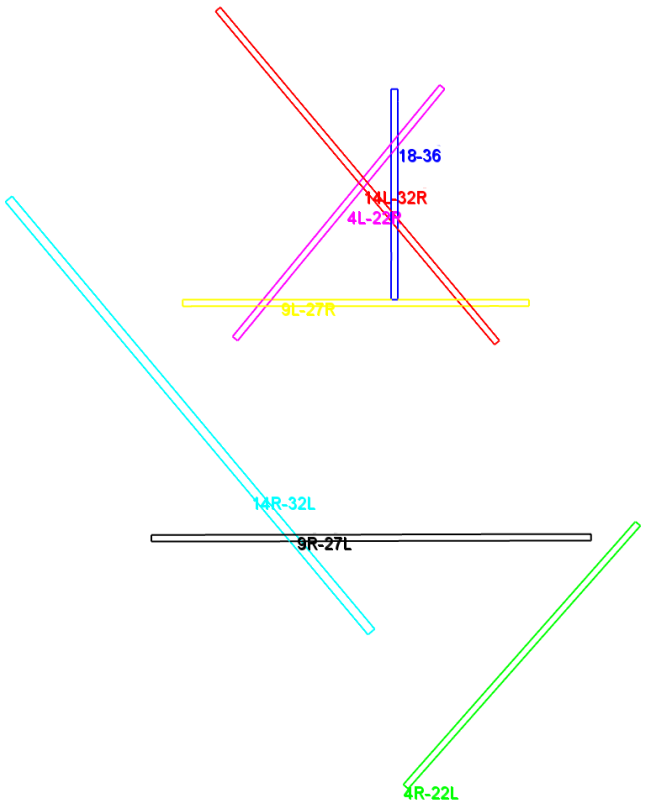


Figure 3.7. Runways after merging

3.4.2 findRunwaysShorterSides()

The developed algorithm requires an information about runway's start and end. At first, a runway must be merged to one piece by function **mergeRunways()**. This complete runway has a rectangular shape and is represented by a list of GPS coordinates. The function "findRunwaysShorterSides" searches for a point in the center of one of the shorter sides. With this information, measurements on this runway can be made.

Psudoalgorithm:

1. find the first corner
2. find the second corner
3. find the third corner
4. **firstDistance** = distance between the first and the second corner
5. **secondDistance** = distance between the second and the third corner
6. if **firstDistance** > **secondDistance**
   shorterSideCenter = point between the second and third corner
   else
   shorterSideCenter = point between the first and second corner
   end
7. return shorterSideCenter

3.4.3 createConnections()

In most situations, a movement on an airport takes place only on guidance lines. Because of that, the decision to create a structure for navigation and pathfinding from guidance lines was straightforward.

Guidance lines as are represented in the AIXM format are not directly suitable for trajectory planning algorithms for several reasons. The AIXM data contains only information about a guidance line, but no information about connections to other guidance lines. Inappropriate connections between AixmGuidanceLines are another reason why a new structure is created. Some AixmGuidanceLines have connections in the middle, which is not appropriate. It is easier to work with a structure, which has connections only at the beginning or at the end. If AixmGuidanceLines have a common point, it can be turned from one to the other one in this point, if and only if one of these AixmGuidanceLines begins or ends at this point (at the figure 3.8 is not possible to turn from yellow to purple guidance line). These reasons resulted in a decision to create a new structure based on AixmGuidanceLines.

A main element of a structure is a GraphSegment. Its main parameters are described in section 3.3. The GraphSegment is created directly from an AixmGuidanceLine, but its connections are only at the beginning or at the end. For this feature, the AixmGuidanceLine have to be split into more GraphSegments. GraphSegments need to fulfill two main conditions to be connected. They must have a common point, and they have to form an angle, that allows an airplane to move from one GraphSegment to the second one (an airplane can turn to a maximum angle TURNIG_ANGLE - in the default setting it is 90°). Sometimes data about GPS positions of guidance lines are not accurate. For these cases, there is a parameter TOLERANCE, which is used for finding points that are close enough to be considered to be the same point.

Algorithm for creating GraphSegments is quite complicated, and the whole code can be found in attached files.

To sum up, AixmGuidanceLines are searched and if any connected AixmGuidanceLine is found algorithm current GraphSegment ends, a new GraphSegment on founded AixmGuidanceLine is created, this new GraphSegment is added to the list of OpenGraphSegments and searching continues in this GraphSegment. When all AixmGuidanceLines are split, the algorithm ends.

Example of AixmGuidanceLines can be seen at figure 3.8. In can be seen, that some guidance lines have connections in the middle (for example blue and purple guidance lines)

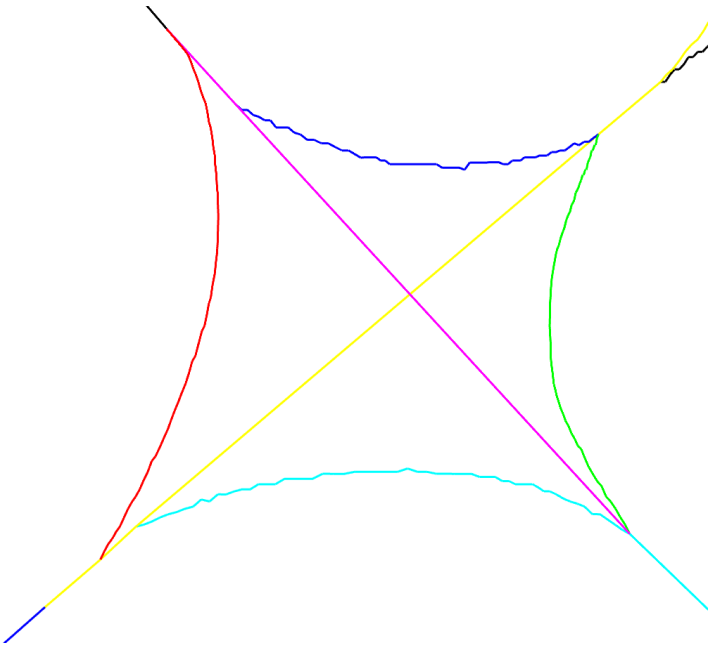


Figure 3.8. Guidance lines

At the figure 3.9 it can be seen that AixmGuidanceLines from the figure 3.8 are split into smaller parts with connections only at the begging or at the end. Crossing between the turquoise and the yellow guidance line is not considered, because it is in the middle of both guidance lines. The red separators were added for clarification. They separate individual GraphSegments.

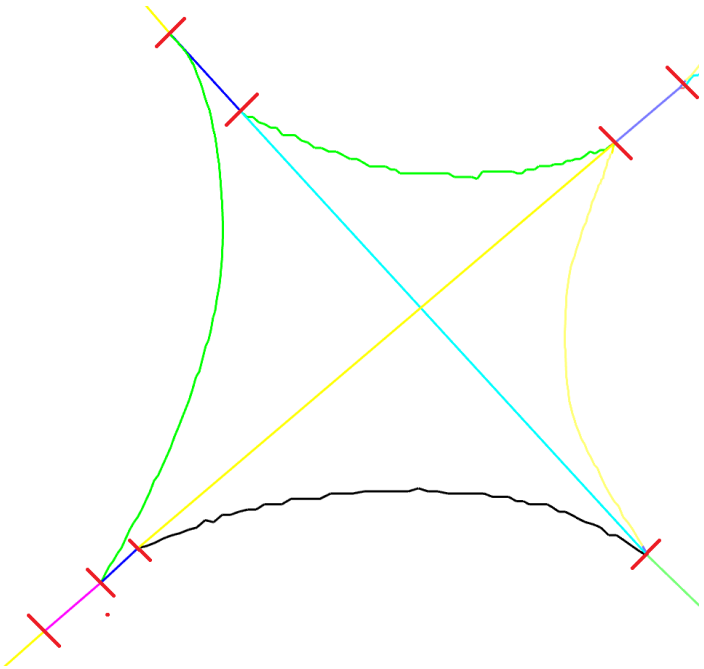


Figure 3.9. GraphSegments

3.4.4 assignGuidanceLines()

Data in AIXM format should contain an information about the owner of a AixmGuidanceLine. However, in some data sets, this essential information is missing. It is essential to know AixmGuidanceLine's owner for several reasons. In a real air traffic control, the pilot receives names of taxiways or runways from the ground controller. Another reason is that some guidance lines cross runways. This information is crucial for safe operation. The pilot must be aware when entering an active runway. Before entering, the permission (mainly from tower control) must be granted and only then, the airplane can enter a runway.

Individual AixmAirportElements are represented as a Poly (a class that AgentFly project uses for polygon representation). Class Poly implements the function isInside(GpsPosition), which returns true when the point is inside this polygon. Otherwise, it returns false. Function **assignGuidanceLines()** check all points that form the guidance line and set owner to the element, which has most points from this guidance line (because of inaccuracy some points may be located on the wrong AixmAirportElement). In case of a match, owner of this guidance line is then decided by a point created between the first and the second coordinate.

3.4.5 connectStands()

A stand is a place on an apron, which also serves as a parking place for airplanes. In the AIXM format, stands are represented as an AixmGuidanceLine with following properties:

1. Stand's owner is AixmApron.
2. Stands can be entered only from one side (segmentsTowardsStart or segmentsTowardsEnd is an empty list).

An airplane on its own is not able to turn from one guidance line to another if an angle between them is lower then TURNING_ANGLE (in default settings 90°). This

rule does not apply to stands. Airplanes can be towed to stands, so the condition about angle does not need to be met.

Pseudoalgorithm:

```
for GraphNodes
  if GraphNode.isStand
    if GraphNode.getSegmentsTowardsEnd.isEmpty
      connections = GraphNode.getSegmentsTowardsStart
    else
      connections = GraphNode.getSegmentsTowardsEnd
    end
    for connections
      if connection.getSegmentsTowardsEnd contains GraphNode
        newConnections.add(connection.getSegmentsTowardsEnd)
      else
        newConnections.add(connection.getSegmentsTowardsStart)
      end
    end
    newConnections.removeDuplicates
    newConnections.remove(GraphNode)
    for newConnections
      newConnection.connect(GraphNode)
      GraphNode.connect(newConnection)
    end
    newConnections.removeAll
  end
end
```

Figure 3.10. Pseudoalgorithm of function connectStands()

Demonstration of the algorithm
Initial state:

GraphSegment1
segmentsTowardsEnd: **null**
segmentsTowardsStart: **GraphSegment2**
GraphSegment2
segmentsTowardsEnd: **GraphSegment1**, **GraphSegment3**
segmentsTowardsStart: **GraphSegment4**
GraphSegment3
segmentsTowardsEnd: **null**
segmentsTowardsStart: **GraphSegment2**
GraphSegment4
segmentsTowardsEnd: **null**
segmentsTowardsStart: **GraphSegment2**

Let's assume, that the **GraphSegment1** is a stand.

The algorithm then proceeds as follows:
1. stand - **GraphSegment1**
2. connections - **GraphSegment2**
3. newConnections - **GraphSegment3**

4. **GraphSegment1**.connect(**GraphSegment3**),
   GraphSegment3.connect(**GraphSegment1**)

Final state:

GraphSegment1
segmentsTowardsEnd: **null**
segmentsTowardsStart: **GraphSegment2**, **GraphSegment3**
GraphSegment2
segmentsTowardsEnd: **GraphSegment1**, **GraphSegment3**
segmentsTowardsStart: **GraphSegment4**
GraphSegment3
segmentsTowardsEnd: **null**
segmentsTowardsStart: **GraphSegment1**, **GraphSegment2**
GraphSegment4
segmentsTowardsEnd: **null**
segmentsTowardsStart: **GraphSegment2**

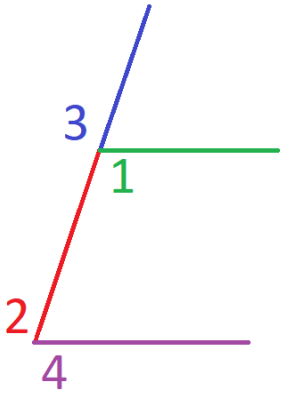


Figure 3.11. Because the angle between GraphSegment 1 and 3 is lower then 90°, they are not connected. However, GraphSegment1 is a stand, and because of that the connection between them is created.

3.4.6 connectRunways()

The majority of airports have more than one runway. These runways are rotated from each other usually by 45° or 90°. This layout provides flexibility to the airport. Wind conditions can be partly eliminated by using a runway that is rotated correctly. When an airport has more runways, route a finding algorithm needs to know, which guidance line leads to which runway.

Each AixmGuidanceLine has a parameter RunwayConnection. An AixmGuidanceLine is considered as an appropriate way to the runway when it starts or ends at runway and runway is not an owner of this guidance line. Another important parameter is a point where guidance line leads on the runway. For this need enum ConnectionDistance was created. This enum has three values (START, MIDDLE, END). A point found in function **findRunwaysShorterSides()** is considered to be a beginning of runway. The default setting of thresholds START or END is:

0 - 20% - START
20 - 80% - MIDDLE
80 - 100% - END

These thresholds can be changed in AixmAirport. Values are saved in RUNWAY_START and RUNWAY_END.

With the connected runway, the ground controller can easily find a way from or to that runway.

Chapter 4
Pathfinding algorithm

For planning airplane's trajectories through the airport, a pathfinding algorithm has to be implemented. The structure created from GraphSegments 3.3.7 allows usage of algorithms designed for working with graph composed of nodes and edges such as A*. Use the A* algorithm as a starting point turned out to be the best option. A GraphSegment is not a point but a line. Due to this fact, several changes compensating this difference are implemented 4.2.

Another restriction for pathfinding algorithm is that the newly planned routes must not collide with already planned routes 4.3.

4.1 A*

A* algorithm was described by Peter Hart, Nils Nilsson and Bertram Raphael of Stanford Research Institute in 1968. It is a heuristic algorithm mainly used for pathfinding in graphs. A* is an extension of Dijkstra's algorithm.

A* uses function f(n) (n is the last node on the path) to determine the cost of the current path. Function f(n) consists of two parts.

$$f(n) = g(n) + h(n) \tag{1}$$

Function g(n) is the real cost of path from start to last node. Function h(n) (heuristic function) is the estimated cost from last node to end. For finding optimal solution function h(n) must be admissible. Mathematically it means:

$$\forall n : 0 \leq h(n) \leq h^{\star}(n), \tag{2}$$

where $h^{\star}(n)$ is the actual cost for reaching the end[17].

In this project, the shortest trajectory is considered as the optimal one. This is not the only possible solution. Other possibilities could be for example the path which takes shorter time, the most fuel economic path or the path with least changes of direction.

Following function is used as heuristic function:

$$h(n) = \sqrt{((x(n)) - x(end))^2 + (y(n) - y(end))^2}. \tag{3}$$

This function is a two-dimensional Euclidian distance. This heuristic function is admissible because in best case scenario the shortest trajectory to the end is a straight line, which is the same as the estimate. In other cases, the estimation is always lower than the actual distance and obviously is higher than zero.

4.2 Input Data

A* is designed for finding best trajectory between two nodes in a graph. For finding the best trajectory between two GraphSegments, the planning algorithm must be able to find the best possible trajectory between two lines. This problem is solved by dividing GraphSegment's references to other GraphSegments into two groups (segmentsTowardsEnd and segmentsTowardsStart). When GraphSegment is expanded, algorithm checks which of these two groups contain the parent of this GraphSegment. If the parent is in the list segmentsTowardsEnd, an airplane is at the start of this GraphSegment, and expanded GraphSegments are GraphSegments from the list segmentsTowardsStart. If the parent is in the list segmentsTowardsStart, an airplane is at the end of the GraphSegment, and expanded GraphSegments are only from the list segmentsTowardsEnd. In input parameters it can be specificated, whether trajectory should be planned from the start or the end of a GraphSegment and in which direction should an airplane enter the final GraphSegment.

Demonstration of a GraphSegment expansion

Let's consider following situation (4.1):

The airplane is on the **GraphSegment3**. Previous GraphSegment was **GraphSegment1**.

Connections between GraphSegments:

GraphSegment1

segmentsTowardsEnd: **GraphSegment3**

segmentsTowardsStart: **null**

GraphSegment2

segmentsTowardsEnd: **GraphSegment3**

segmentsTowardsStart: **null**

GraphSegment3

segmentsTowardsEnd: **GraphSegment5**, **GraphSegment6**

segmentsTowardsStart: **GraphSegment1**, **GraphSegment2**

GraphSegment4

segmentsTowardsEnd: **GraphSegment5**, **GraphSegment6**

segmentsTowardsStart: **null**

GraphSegment5

segmentsTowardsEnd: **null**

segmentsTowardsStart: **GraphSegment3**, **GraphSegment4**

GraphSegment6

segmentsTowardsEnd: **null**

segmentsTowardsStart: **GraphSegment3**, **GraphSegment4**

The parent of the **GraphSegment3** is **GraphSegment1**, which is in the list segmentsTowardsStart. Because of that, expanded GraphSegments are **GraphSegment5** and **GraphSegment6** (all GraphSegments from the list segmentsTowardsEnd). The **GraphSegment4** is not connected with the **GraphSegment3**, because the angle between them is lower then a TURNING_ANGLE (90°).

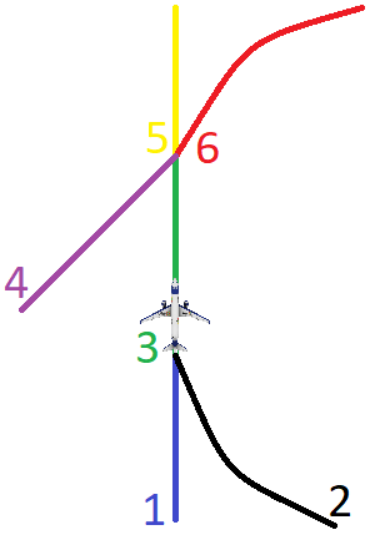


Figure 4.1. The airplane can continue only to the GraphSegments 5 or 6. The GraphSegment4 is no available becase the angle between the GraphSegments 3 and 4 is lower then 90°.

A* algorithm is designed to work in the Cartesian coordinate system, but the input data are GPS positions. Even though that airports are on the sphere, counting with GPS position is possible, because the size of airports is small, differences between altitude are negligible, and distances from poles are big enough. The inaccuracy resulting from these factors is negligible.

4.3 Noncollision Planning

The algorithm is designed to calculate the best possible trajectory between two lines. The possibility to plan a trajectory that would have some restriction from the previously planned trajectory is a new feature, which extends the pathfinding algorithm. In this task, airplanes cannot be on the same GraphSegment at the same time and in the opposite direction.

To resolve this problem, a new object is defined. **GraphSegmentOccupancy** described in section 3.3.8 is designed specifically for this problem. GraphSegmentOccupancy contains information about occupied **GraphSegment**, the time interval in which is this GraphSegment used (**enterTime**, **leaveTime**), which entity uses this GraphSegment **entityID** and in a which **direction**. One GraphSegment can be occupied by multiple airplanes, which means more GraphSegmentOccupancies. Information about an entityID is necessary for replaning route described in section 6.1.2. Input parameters must include a list of GraphSegmentOccupancies and the current time is milliseconds.

To determine whether there is a risk of collision, every GraphSegment must know when it will be occupied and in which direction. All nodes from A* algorithm contains the information about the cost (in this case is equal to distance) from start node to current node. With this information and with an assumption that airplane is moving with a constant velocity, the algorithm can estimate, when the airplane will be occuping this GraphSegment using the formula

$$t = \frac{s}{v}, \tag{4}$$

where s is the length of GraphSegment and v is the velocity of the airplane.

A direction can be determined thanks to the knowledge of this GraphSegment's parent. If the parent is in the list **segmentsTowardsStart**, the direction is 1 (airplane is driving through this GraphSegment from the start to the end). Conversely, if the parent is in the list **segmentsTowardsEnd**, the direction is 0 (airplane is driving through this GraphSegment from the end to the start).

After finding these two parameters (time and direction), the algorithm can determine, whether there is a risk of a collision. A collision is detected, just when the following conditions are met:

1. **The GraphSegment in GraphSegmentOccupancies is equal to the current GraphSegment.**

2. **The direction of this GraphSegmentOccupancies must be different (airplanes can be on the same GraphSegment in the same direction).**

3. **The calculated time is in the interval (enterTime,leaveTime).**

When a collision is detected, this GraphSegment is put in the closed list. Due to this operation it can no longer be claimed, that found trajectory is the shortest one. However, simulations show, that found trajectories are not too much different from the optimal ones.

4.4 Differences between Planning Algorithm for Simulation and A*

The planning algorithm works with GraphSegments (lines), while A* works with nodes. This feature requires changes in function which expands GraphSegment. The planning algorithm must consider the direction of an airplane. The direction of an airplane decides whether GraphSegments from list segmentsTowardsEnd or list segmentsTowardsStart are available for next movement.

A* algorithm can not plan trajectories, which would depend on previously planned trajectories. The planning algorithm can work with GraphSegmentOccupancies and can calculate trajectories without collisions. However, due to this feature, it can no longer be guaranteed, that found trajectory is the shortest one.

The planning algorithm allows determining starting and final heading of an airplane.

4.5 Airplane Movement Implementation

For a movement of an airplane, a class PlanWrapper is used in the AgentFly project. For this simulations a new class **GpsFlightPlanWrapperAixmAirports** which inherits from the PlanWrapper is created. This class is responsible for creating PlanWrapper suitable for movement on an airport.

The whole plan is created from smaller elements - GpsTaxiElements. These elements are created between two points and contain information about initial state, final state, velocity, heading, etc. A final plan is a linked list of GpsTaxiElements.

4.5.1 Start of the Airplane

A main parameter for creating GpsFlightPlanWrapperAixmAirports is a list of GraphSegments which determines the trajectory. These GraphSegments are used in function **createTrajectory()**. This function creates GpsTaxiElements from GraphSegments.

The function **createTrajectory()** iterates through all GraphSegments and all their coordinates. It creates GpsTaxiElements between adjacent coordinates and set its parameters such as heading, velocity, etc.

4.5.2 Stopping the Airplane

For situations where an airplane needs to be stopped is used GpsStopElement. A function **stop()** is used when an airplane is on a collision course or should stop for other reasons. This function removes all GpsTaxiElements from the plan and replaces them with GpsStopElement. Before it happens, the old trajectory is saved, and when the path is clear, this trajectory is loaded, and airplane continues in its previous trajectory.

Chapter 5
Ground Controller

The ground controller is a person responsible for an airplane movement. When an airplane arrives at the airport, a pilot must tune his radio frequency to be able to communicate with the ground controller. The ground controller then must navigate a pilot through the airport to a stand. Similarly, when an airplane is ready to depart, the ground controller must navigate pilot on a runway.

5.1 Airport Controller

In a simulation, all controllers described in section 2.5 are replaced by one called Airport controller. This controller is responsible for all sectors at ones. In a future work, simulation can be extended, and more controllers that are communicating with themselves can be added. The airport controller is mainly responsible for planning routes, while avoiding possible conflicts or collisions. The planning algorithm is described in section 4.

5.2 Communication with Pilot

The ground controller communicates directly to pilots. Because a slip of the tongue or mishearing can lead to serious problems, a communication between pilot and a ground controller must be safe. For this reason is in aviation used "pilot alphabet".

The oficial name of this spelling alphabet is ICAO Spelling Alphabet, and it uses words for representing letters. Because of this, the risk of being misunderstood is minimal. Moreover, some words have unique pronunciation which also reduces the risk. All letters and their representations can be seen at the figure 5.1.

A–ALFA	B–BRAVO
C–CHARLIE	D–DELTA
E–ECHO	F–FOXTROT
G–GOLF	H–HOTEL
I–INDIA	J–JULIETT
K–KILO	L–LIMA
M–MIKE	N–NOVEMBER
O–OSCAR	P–PAPA
Q–QUEBEC	R–ROMEO
S–SIERRA	T–TANGO
U–UNIFORM	V–VICTOR
W–WHISKEY	X–X–RAY
Y–YANKEE	Z–ZULU

Figure 5.1. Alphabet [1]

Another thing to ensure maximum security of communication is repeating received message. The ground controller and pilots repeat any received message, or at least they repeat the most valuable information [7].

[1] http://2.bp.blogspot.com/-jGaFgK9QGUU/TZdBDiWqxpI/AAAAAAAAAG8/gz31iCmhBE0/s1600/nato.jpg

Communication can look like this:
Ground controller: Delta 443, right Alpha, right Bravo, then Papa.
Pilot: Roger, right Alpha, right Bravo, then Papa, Delta 443.

5.2.1 Communication in Simulation

In the simulation, individual agents (pilots and the airport controller) communicate over the radio. At first, pilot sends a message with current position and desired position. Airport controller then finds a way (list of GraphSegments) and sends it back to the pilot. The pilot then repeats this message to the controller and creates PlanWrapper (for more information see section 4.5).

Airport controller can send a message to pilot with the requirement to change his current path.

Pilots do not communicate with each other.

In a real situations, the controller sends to pilots only information about taxiways, which should be used. In simulations, the whole way (list of GraphSegments) is sent. Main reasons for this solutions are following:

1. The ground controller knows where the airplane will be at a certain time and can use this information to planning ways for other airplanes.

2. Pilots do not need to use pathfinding algorithm for finding a way through taxiways. This saves a calculation time.

In the simulation, a communication between the pilot and the ground controller could be handled by sending simple events. However, communication over the radio is one of the activities that have an impact on a ground controller's cognitive load. Despite the fact that message sent by the ground controller is a list of GraphSegments, the radio displays only names of taxiways. This makes the simulation more realistic.

At the figure 5.2 is shown the cut out of the ground controllers display. On display can be seen guidance lines, runways, and airplanes.

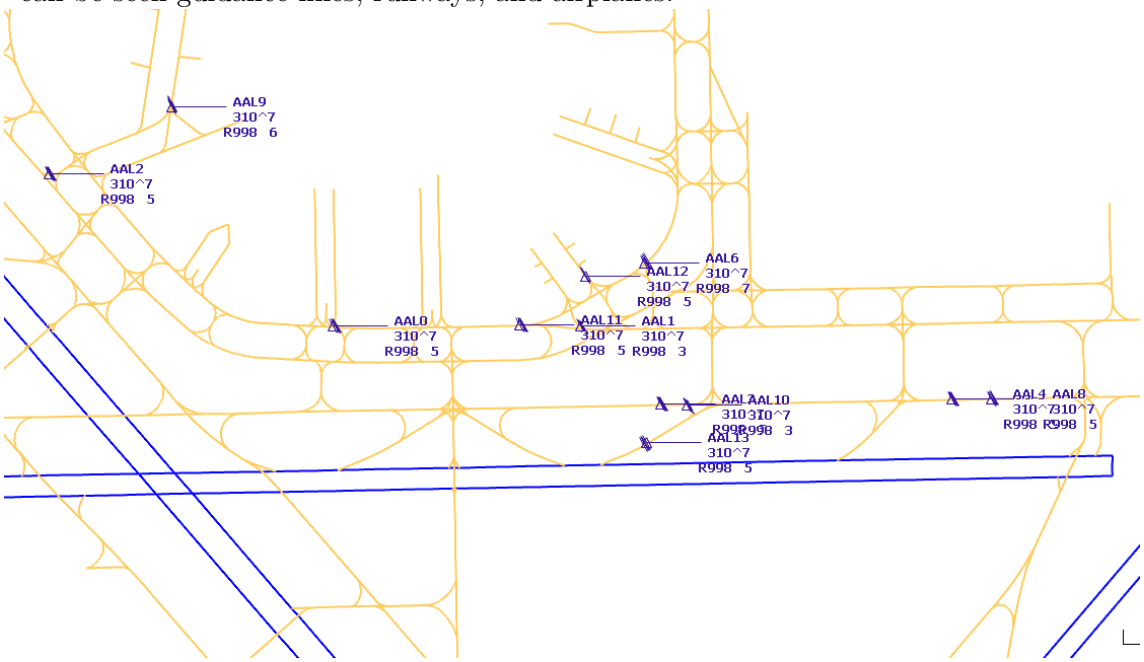


Figure 5.2. Ground controllers display

Chapter 6
Collision Avoidance Mechanisms

Collisions at the airport can occur for many different reasons. The majority of them can be sorted into following groups.

1. Airplanes on the same taxiway in the opposite direction
2. Airplanes on the same taxiway in the same direction
3. Airplanes on crossings
4. Airplanes moving through the active runway

Some problems can be solved by pilots directly in a traffic, but some problems must be solved by the ground controller in advance.

6.1 Airplanes on the Same Taxiway in the Opposite Direction

The ground controller is responsible for solving this problem. To handle this problem, the ground controller uses pathfinding algorithm described in section 4. With this algorithm, noncollision trajectory can be calculated.

6.1.1 Planning Process

There are two possible ways for trajectory planning.

1.Plan all trajectories at the same time
This solution is based on the idea of planning trajectories for all airplanes at the same time. Results of this solution would be probably better concerning the total length of all trajectories, but it requires recompute the plan of all airplanes whenever a new airplane arrives or when an unexpected situation occurs. Also, it would require an unreasonable amount of messages between ground controller and pilots.

2.Plan trajectory only for a new airplane
The main advantage of planning a trajectory for one airplane is a fast calculation. Even though that found trajectory does not have to be optimal, simulations show, that in most cases this trajectory is a realistic and reasonable. Another advantage of this solution is the similarity with a real situations.

6.1.2 Position Checking

A GraphSegmentOccupancy contains the estimation when an airplane should be on particular GraphSegment. However, in real situations, movement on the airport is influenced by many aspects, and an airplane is sometimes forced to slow down or even stop. Because of that, the ground controller regularly makes sure, that all airplanes are on expected GraphSegments. If any airplane is on the GraphSegment in the unexpected time, the ground controller must recompute a new way for this airplane.

Using known GpsPosition of an airplane, the ground controller can find on which GraphSegment the airplane currently is. This GraphSegment is compared with GraphSegmentOccupancies, and if the airplane should not be on this GraphSegment, then the ground controller deletes all GraphSegmentOccupancies for this airplane (while using entityID from GraphSegmentOccupancy)and recomputes a new trajectory for this airplane. This is necessary because other airplanes trajectories were computed with the assumption that this airplane will follow its plan.

6.2 Airplanes on the Same Taxiway in the Same Direction

This type of collisions can be solved by a pilot. The pilot can check all airplanes before him, and he can slow down or stop. The detailed description of other airplane detection is in section 6.3.

6.3 Airplanes on Crossings

In a situation when two airplanes are on a crossing, one airplane must give a right of way to the other airplane. This situation can be handled by the ground controllers, but sometimes they are handled by pilot themselves. In this simulation, pilots are responsible for solving this situation. Possible heuristics for solving this problem are for example:

First in first out
The most intuitive solution. However, in some situations, it is not certain who came first. In this situations, some other method must be used.

Heavier airplane priority
The main advantage of this solution is fuel saving. A smaller airplane can be slowed down much easier than a big airplane. The disadvantage is a requirement for another heuristic when two airplanes with the same size meet each other.

Right-hand rule
Right-hand rule is a system, in which the pilot of an airplane is required to give way to airplanes approaching from the right at crossings. The main advantage of this rule is that almost every common situation can be handled by this rule (situations when airplanes are on all entries from crossing, but this situation cannot happen at the airport). This was the main reason, why it was selected as a heuristic to determine which airplane has right of way.

6.3.1 Right-hand Rule Implementation

For following the Right-hand rule, pilots need to know where other airplanes are, and they must find out if any of them is on a collision course. If the pilot finds an airplane that is coming from right side, he must stop or slow down, let that airplane go, and then he can continue in his previous plan.

In a simulation, every pilot receives an event when any entity on the airport moves. The content of this event is the information about all other airplanes, their positionw, and headings. The controlled area is represented by a polygon (a section bounded by a red line at Figure 6.1), which is calculated from the airport position and from the heading. If any of other airplanes is heading into this polygon and is moving, the pilot

stops and waits, until no one is heading into this polygon. Then he can start moving. A size and an angle of this polygon can be set in the pilot module by following parameters:

POLYGON_SIZE - the size of the polygon
POLYGON_ANGLE - the angle of the controlled area

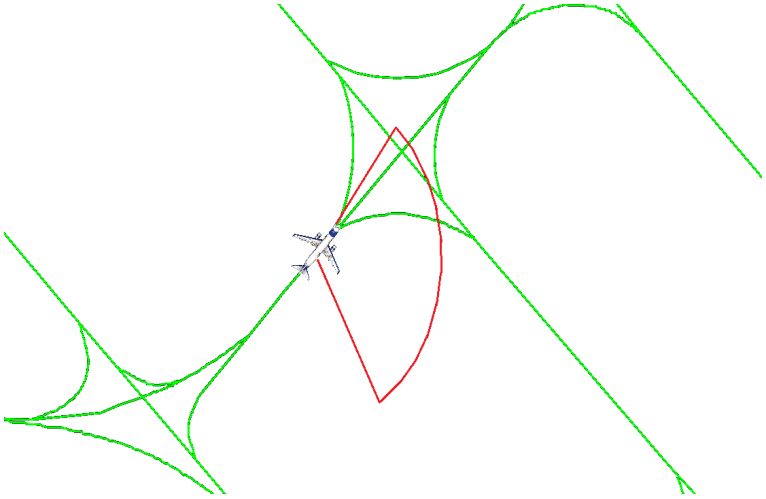


Figure 6.1. Polygon with controlled area

6.4 Airplanes Moving Through the Active Runway

In some situations, it is necessary for the airplane to cross the active runway. When the airport has more controllers, pilot switches on the frequency of responsible controller and asks for permission to cross runway. This simulation contains only one controller, because of that this problem can be solved in route planning process.

Chapter 7
Simulation Tests

To verify that all modules work as intended, several tests are implemented. Each module is tested individually in smaller tests, and then their ability to cooperate is tested on large-scale simulations.

7.1 Trajectory Planning

Trajectory planning algorithm evaluation also served as evaluation for an algorithm that creates the structure of GraphSegments from AixmGuidanceLines.

At the figure 7.1 is shown a way from runway 9R to the stand. The airplane can be seen at the starting point (right bottom part of the figure), and the calculated route to the stand is the red line. Green lines are other guidance lines, and yellow lines are runways.

Already at first glance can be seen, that computed trajectory follows the rules described in sections 3.4.3 and 4. The trajectory is the shortest one and angles, in turn, are lower then 90°.

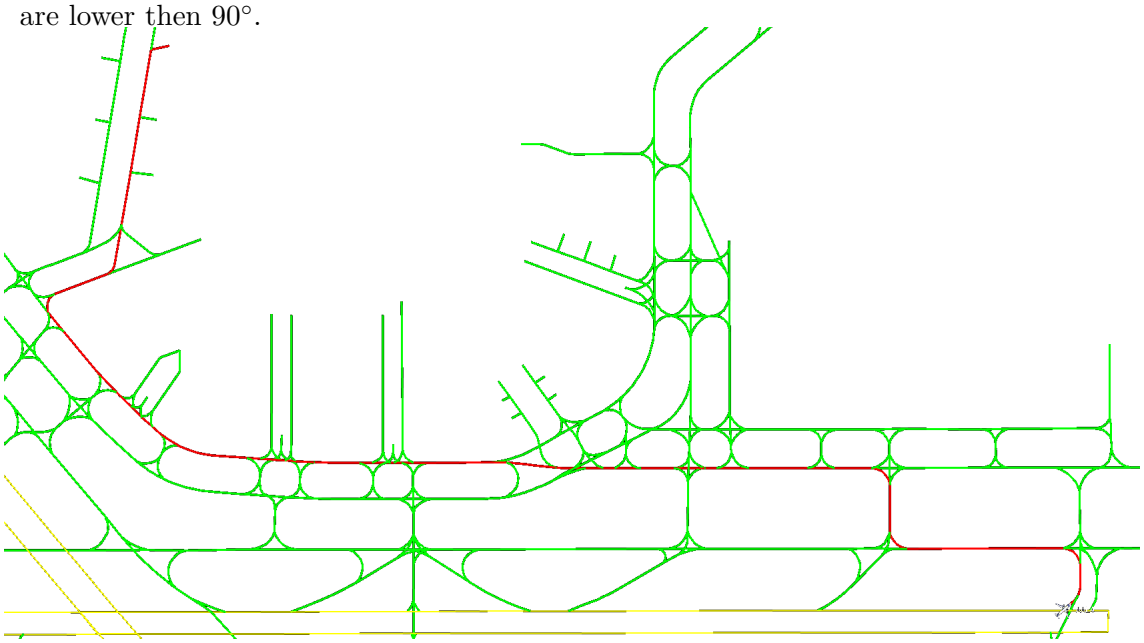


Figure 7.1. Trajectory planned from the runway to the stand

7.2 Right-hand Rule

Right-hand rule is a heuristic used to determine which airplane has the right of way when two airplanes meet on a crossing. The detailed description is in the section 6.3.

In this simulation, two airplanes are on the crossing. Airplane arriving from the west is supposed to stop and let airplane from southeast drive through the crossing. Then airplane should continue in its way.

At the figure 7.2 both airplanes are approaching towards the crossing.

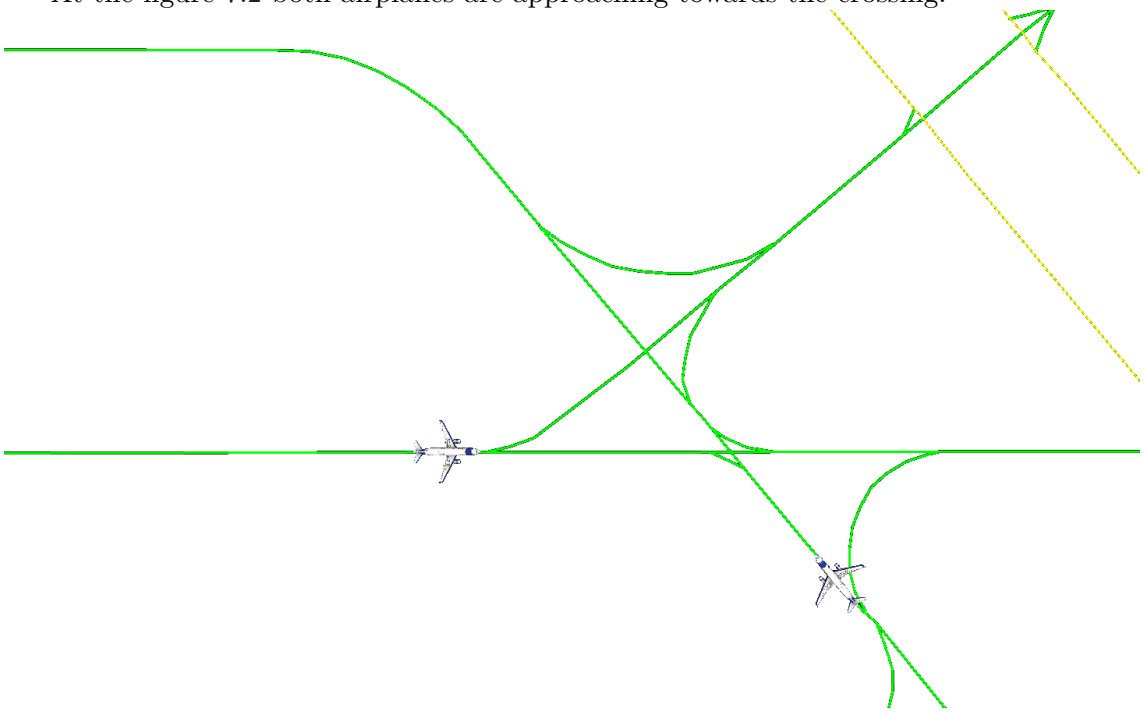


Figure 7.2. Right-hand rule - airplanes are approaching towards the crossing.

At the figure 7.3 the airplane from west stopped (signalized by the blue bubble) and the other one drives through.

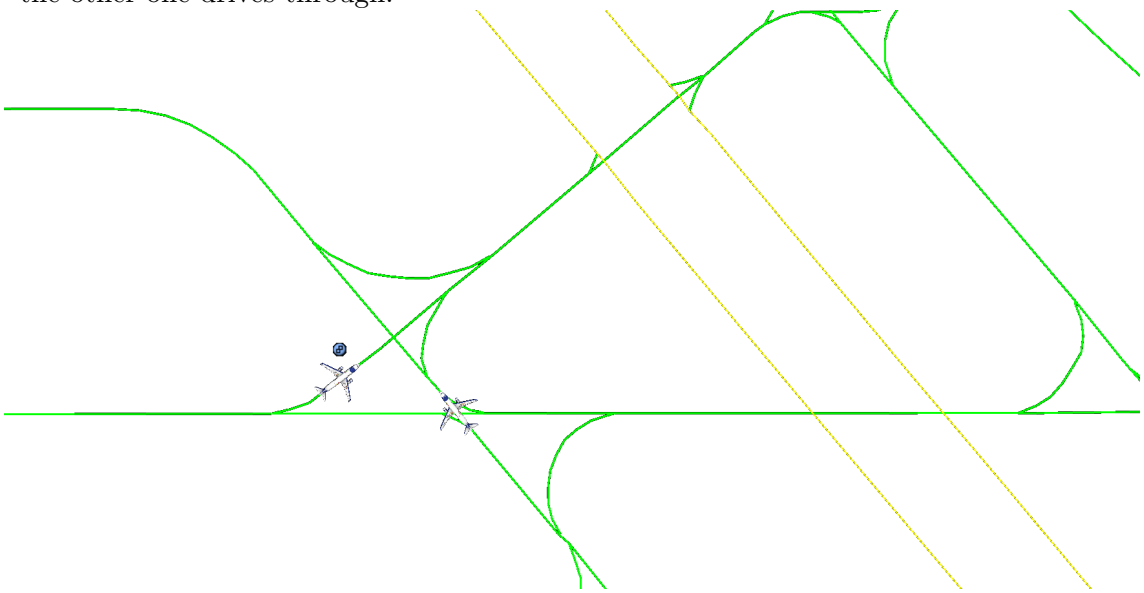


Figure 7.3. Right-hand rule - the airplane with the blue bubble stoped

At the last figure 7.4 both airplanes are moving and have sufficient distance between them.

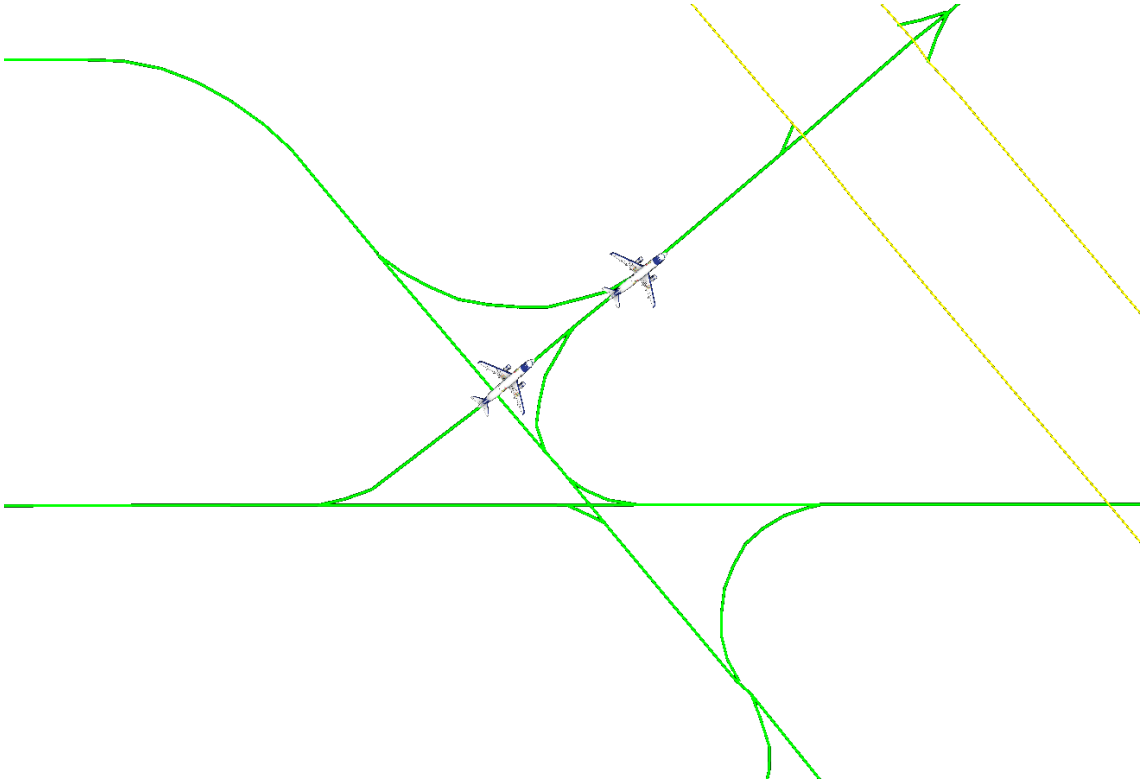


Figure 7.4. Right-hand rule - both airplanes are moving again

7.3 Collision Avoidance

The collision avoidance mechanism tested by this simulation is described in section 6.1.

For this simulation, two airplanes are changing their positions (the first one is moving from the runway to stand, the second one is moving from stand to runway). This situation is not realistic, because airplanes usually arrive from one runway and departe from the other one, but for a demonstration of the collision avoidance mechanism is this situation most transparent.

At the figure 7.5, are highlighted both ways from airplanes. Purple lines are common to both airplanes. The blue line is for the airplane coming from the south, and the red one is for the other one. As can be seen, the airplane approaching from the north takes the longer way to avoid the collision.

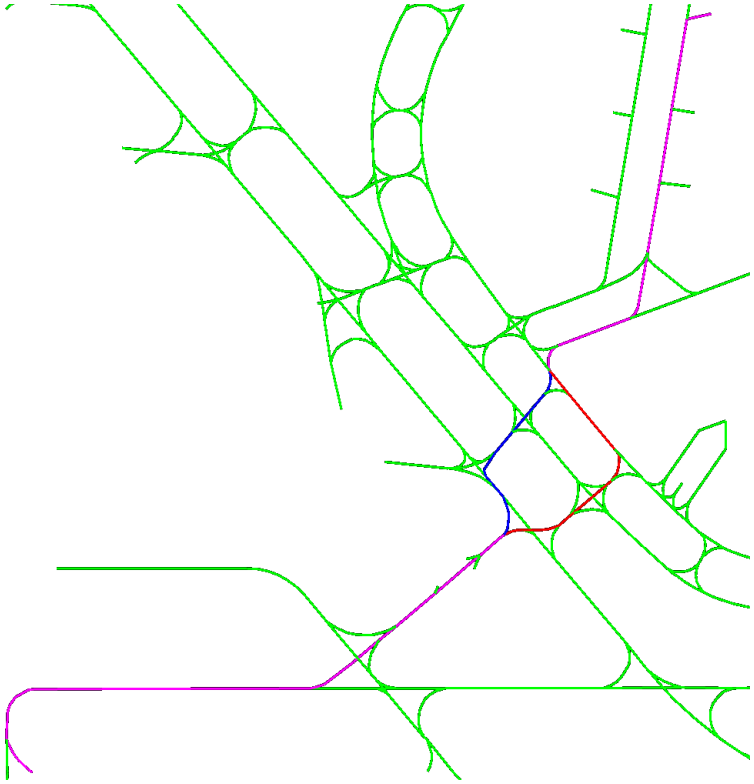


Figure 7.5. For airplanes which are changing their positions, are calculated different trajectories.

7.4 Large Scale Simulation

This scenario simulates a real situation with two active runways (9L and 9R). One runway serves for departures (9L - on the north), and the other one serves for arrivals (9R - on the south).

Besides this particular simulation, various simulations with different runways or a traffic density are implemented. They examine the behavior of individual modules and how they work together. Even though that this simulations are not based on real data, they are based on real pattern situations.

At the figure 7.6, can be seen a situation from this scenario. It must be noted, that airplanes are for simulation purposes displayed bigger.

Figure 7.6. Large scale simulation

Chapter 8
Conclusion

Main tasks of this thesis is to simulate an air traffic at the airport, pilots and a ground controller.

All tasks of this bachelor's thesis were fulfilled.

1.Study the problem of air traffic control at the airport
The main source of information is a book Fundamentals of Air Traffic Control. A Youtube channel VASAviation, which is providing records from radio communication can be really helpful for understanding how pilots and controller communicates.

2.Study the problem of airport data representation.
Several sources of data have been found, but the majority of them lack some sort of information that was necessary for simulation. The AIXM format is the best option for the airport representation. This format is expanding around the world and it is the result of teamwork of the Federal Aviation Administration and Eurocontrol, so it can be assumed, that airports will start providing information in this format.

3.Design airport data representation suitable for aircraft movement and air traffic control.
The AIXM data format contains a large amount of information but is not directly suitable for simulating air traffic at the airport. For this purposes, a new structure is designed and implemented as well as the algorithm for processing AIXM data into this structure.

4.Design initial behavior model of airport air traffic controller a pilot.
The appropriate model of air traffic controller requires a pathfinding algorithm for noncollision planning. This algorithm can find a trajectory while avoiding collisions and works with the data structure created from the AIXM data set. Another collision avoiding mechanism is implemented in the pilot module - rule of the right hand. Communication between a pilot and a ground controller is transmited over the radio. Radio is used as a component for workload model.

5.Implement designed methods into AgentFly system.
All methods are implemented and integrated in AgentFly system while respecting all principles of this project.

6.Perform experiments using various configurations.
Several simulation scenarios is used to evaluate all models and functions. Simulations for testing individual components as well as large-scale simulations are implemented.

8.1 Future Work

On following list can be found proposals for upgrading simulation.

Adding more controllers

As it is described in section 2.5, air traffic at the airport is handled by more ground controllers. Adding more controllers and implementing communication between them would make the simulation more realistic.

Simulate more actions on the airport

Movement on the airport is not only from a runway to stand and vice versa. Airplanes sometimes must stop at the de-icing area; they need to be refueled etc. Adding these actions would make the simulation more realistic.

Improve the pathfinding algorithm

Pathfinding algorithm developed for this simulation can find an only direct way to the designated point. The possible extension would be planning with stop possibility. In some situations, it could be possible that way with few stops could be faster.

References

[1] Airline industry worldwide - number of flights 2017 - Statistic. *Statista - The Statistics Portal for Market Data, Market Research and Market Studies* [online]. Statista, 2018. [Acessed 23 May 2018]. Avaible at: `https://www.statista.com/statistics/564769/airline-industry-number-of-flights/`

[2] Airport - Definition of Airport by Merriam-Webster. *Dictionary by Merriam-Webster: America's most-trusted online dictionary* [online]. Merriam, 2018. [Acessed 21 May 2018]. Avaible at: `https://www.merriam-webster.com/dictionary/airport`

[3] Runway selection - Airservices. *Airservices* [online]. Airservices Australia, 22 April 2014. [Acessed 19 May 2018]. Avaible at: `http://www.airservicesaustralia.com/wp-content/uploads/12-139FAC_NCIS-Runway-selection_P2.pdf`

[4] Z. K. CHUA, M. COUSY, F. ANDRÉ, and M. CAUSSE. *Simulating Air Traffic Control Ground Operations:Preliminary Results from Project Modern Taxiing* [online]. In: *4th SESAR Innovation Days 2014*. Madrid: Universidad Politécnica de Madrid, 2014. [Acessed 19 May 2018]. HAL ID: hal-01132451. Avaible at: `https://hal-enac.archives-ouvertes.fr/hal-01132451/file/ChuaCousyAndreCausse2014.pdf`

[5] ACI WORLD SAFETY, and TECHNICAL STANDING COMMITTEE (SUB-GROUP). *Runway Safety Handbook*. 1st ed. Montreal: ACI World, Montreal, Canada, 2014. ISBN 978-1-927907-31-3.

[6] ICAO Aerodrome Reference Code - *SKYbrary Aviation Safety*. SKYbrary Aviation Safety [online]. SKYbrary, 2 August 2017. [Accessed 19 May 2018]. Avaible at: `https://www.skybrary.aero/index.php/ICAO_Aerodrome_Reference_Code`

[7] *VASAviation. YouTube* [online]. [Accessed 19 May 2018]. Avaible at: `https://www.youtube.com/channel/UCuedf_fJVrOppky5gl3U6QQ`

[8] M. S. NOLAN. *Fundamentals of air traffic control*. 5th ed. Clifton Park, N.Y.: Delmar Cengage Learning, c2011. ISBN 14-354-8272-7.

[9] Letecká informační služba. *Letecká informační služba* [online]. Letecká informační služba, Řízení letového provozu ČR, s.p., 2018. [Accessed 24 May 2018]. Avaible at: http://lis.rlp.cz

[10] OpenFlights: Airport and airline data. *OpenFlights.org: Flight logging, mapping, stats and sharing* [online]. [Accessed 24 May 2018]. Avaible at: `https://openflights.org/data.html`

[11] Public Airport Corner. *Moved Temporarily* [online]. EUROCONTROL, 2018. [Accessed 24 May 2018]. Avaible at: `https://ext.eurocontrol.int/airport_corner_public/`

[12] Aeronautical Navigation Database (ANDB) - AeroNavData. *Home - AeroNavData* [online]. AeroNavData, 2018. [Accessed 24 May 2018]. Avaible at: `https://aeronavdata.com/what-we-do/aeronautical-navigation-database-andb/`

[13] Aeronautical Data, National Flight Data Center (NFDC). *Federal Aviation Administration* [online]. Federal Aviation Administration, 3 May 2018. [Accessed 24 May 2018]. Avaible at: `https://www.faa.gov/air_traffic/flight_info/aeronav/aero_data/`

[14] AIXM Wiki - Home (Main.WebHome) - XWiki. *Moved Temporarily* [online]. EUROCONTROL, 9 September 2009. [Accessed 24 May 2018]. Avaible at: `https://ext.eurocontrol.int/aixmwiki_public/bin/view/Main/`

[15] EUROCONTROL - The European Organisation for the Safety of Air Navigation - Home. *EUROCONTROL - The European AIS Database: Introduction to EAD Basic - Home* [online]. EUROCONTROL, 2001. [Accessed 24 May 2018]. Avaible at: `https://www.ead.eurocontrol.int/cms-eadbasic/opencms/en/home/`

[16] AIXM. *AIXM* [online]. EUROCONTROL, 2018 [Acessed 19 May 2018]. Avaible at: `http://aixm.aero`

[17] W. ZENG and R. L. CHURCH. Finding shortest paths on real road networks: the case for A*. *International Journal of Geographical Information Science* [online]. 2009, **23**(4), 531-543 [Acessed 2018-05-19]. DOI: 10.1080/13658810801949850. ISSN 1365-8816. Avaible at: `http://www.tandfonline.com/doi/abs/10.1080/13658810801949850`

Appendix A
Abbreviations

GPS	Global Positioning System
AIXM	Aeronautical Information Exchange Model
FAA	Federal Aviation Administration
ICAO	International Civil Aviation Organization
EAD	European AIS Database
HTML	HyperText Markup Language
XML	eXtensible Markup Language
XSD	XML Schema Definition
JSON	JavaScript Object Notation
UML	Unified Modeling Language

Appendix B
List of Attachments

Replays from simulations:
- CollisionAvoidance.mp4
- LargeScaleAirport.mp4
- LargeScaleController.mp4
- Right-hand rule.mp4

AIXM XSD files:
- AIXM_AbstractGML_ObjectTypes.xsd
- AIXM_DataTypes.xsd
- AIXM_Features.xsd

AIXM data
- Chicago_Aprons.xml
- Chicago_Runways
- Chicago_Taxiways.xml
- Chicago_VerticalStructures.xml

Source codes
- JavaClasses.zip