Design document

Potkuri-group

Helsinki December 12, 2008

Software Engineering Project

UNIVERSITY OF HELSINKI
Department of Computer Science

Course

581260 Software Engineering Project (6 cr)

Project Group

Veera Hoppula
Mikko Kuusinen
Jesse Paakkari
Tobias Rask
Timo Tonteri
Eero Vehmanen

Client

Valentin Polishchuk

Project Masters

Sampo Lehtinen

Homepage

`http://www.cs.helsinki.fi/group/potkuri`

Change Log

Version	Date	Modifications
1.0	12.12.2008	Final version
0.3	20.11.2008	Modified to correspond the latest version of the program
0.2	4.11.2008	Calculation part inspection
0.1	30.9.2008	Document created

Contents

1 Vocabulary **1**

2 General Architecture **2**

3 Chopper **3**

3.1 Chopper 3

4 General **4**

4.1 DataCollection 4

4.1.1 Method DataCollection 4

4.1.2 Method setAirControlClass 4

4.1.3 Method getAirControlClass 4

4.1.4 Method setArcPoints 4

4.1.5 Method getArcPoints 4

4.1.6 Method setCalculationControl 4

4.1.7 Method getCalculationControl 4

4.1.8 Method setFlagMap 4

4.1.9 Method getFlagMap 5

4.1.10 Method setRefreshRate 5

4.1.11 Method getRefreshRate 5

4.1.12 Method setInternalClock 5

4.1.13 Method getInternalClock 5

4.1.14 Method setGraph 5

4.1.15 Method getGraph 5

4.1.16 Method setGraphicsClass 5

4.1.17 Method getGraphicsClass 5

4.1.18 Method setWeatherMap 5

4.1.19 Method getWeatherMap 6

4.1.20 Method setTreeClass 6

4.1.21 Method getTreeClass 6

4.2 Parameter 6

4.2.1 Parameter 6

4.3 Parameters 6

4.3.1 Method getParameterValue 6

5 Calculation **7**

5.1 CalculationControl 7

5.1.1 Member variables 8

5.1.2 Method CalculationControl 8

5.1.3 Method runTimer 8

5.1.4 Method setTimerFlag 8

5.1.5 Method update 9

5.2 BuildTree 9

5.2.1 Member variables 9

5.2.2 Method BuildTree 9

5.2.3 Method getGraph 10

5.2.4 Method getTree 10

5.2.5 Method update 10

5.2.6 Method isChanged 10

5.2.7 Method aStar 10

5.2.8 Method calculate 10

5.2.9 Method checkTree 11

5.2.10 Method treeRecursion 11

5.2.11 Method setSlidingSafetyDistance 11

5.2.12 Method calculatePath 12

5.2.13 Method setSafetyDistance 12

5.2.14 Method testSide 12

5.2.15 Method setSafetyDistanceAirPlanes 12

5.2.16 Method halftable 12

5.3 Graph 13

5.3.1 Member variables 13

5.3.2 Method Graph 13

5.3.3 Method createGraph 14

5.3.4 Method getGraph 14

5.3.5 Method calculateArcPoints 14

5.3.6 Method getArcPoints 14

5.3.7 Method getOuterArc 14

5.3.8 Method getAirport 14
5.3.9 Method update 14
5.3.10 Method graphUpdate 14
5.3.11 Method checkSquare 15
5.4 Vertex 15
5.4.1 Member variables 15
5.4.2 Method Vertex 15
5.4.3 Method reset 16
5.4.4 Method distanceTo 16
5.4.5 Method graphDistanceTo 16
5.4.6 Method calculateDir 16
5.4.7 Method compareTo 16
5.5 Tree 16
5.5.1 Member variables 16
5.5.2 Method Tree 17
5.5.3 Method getRoot 17
5.5.4 Method getTreeMergePoints 17
5.5.5 Method setTreeMergePoints 17
5.5.6 Method getTree 17
5.5.7 Method getVertexTree 17
5.5.8 Method treeNodePath 17
5.5.9 Method addToTree 17
5.5.10 Method getLeaves 18
5.5.11 Method getVertexLeaves 18
5.5.12 Method toString 18
5.6 TreeNode 18
5.6.1 Member variables 18
5.6.2 Method TreeNode 19
5.6.3 Method setChild 19
5.6.4 Method setParent 19
5.6.5 Method getLeft 19
5.6.6 Method getRight 19
5.6.7 Method getParent 19

5.6.8 Method getVertex 19

6 AirSpace **20**

6.1 AirControl 20

6.1.1 Member variables 21

6.1.2 Method AirControl 21

6.1.3 Method update 21

6.1.4 Method setBuildTree 21

6.1.5 Method getAirportStatus 21

6.1.6 Method getMaxArrivingRate 22

6.1.7 Method getPlanes 22

6.1.8 Method getNumberOfPlanes 22

6.1.9 Method getArrivedPlanes 22

6.1.10 Method addPlane 22

6.1.11 Method controlPlanes 22

6.1.12 Method closestMergePoint 23

6.1.13 Method closestTNToPlane 23

6.1.14 Method safetyZoneCheck 23

6.1.15 Method flyPlane 23

6.1.16 Method newPath 23

6.1.17 Method sort 24

6.2 Plane 24

6.2.1 Member variables 24

6.2.2 Method Plane 25

6.2.3 Method fly 25

6.2.4 Method setColor 25

6.2.5 Method clearPath 25

6.2.6 Method setCurrentTreeNode 25

6.2.7 Method setLat 25

6.2.8 Method setLon 25

6.2.9 Method isOnArrivalTree 26

6.2.10 Method addToPath 26

6.2.11 Method setPath 26

6.2.12 Method setSpeed 26

6.2.13 Method setDistToAirport 26
6.2.14 Method setDir 26
6.2.15 Method getDistToAirport 26
6.2.16 Method getDir 26
6.2.17 Method getColor 27
6.2.18 Method getCurrentNode 27
6.2.19 Method getId 27
6.2.20 Method getLat 27
6.2.21 Method getLon 27
6.2.22 Method getPath 27
6.2.23 Method getSpeed 27
6.2.24 Method getSafetyZone 27
6.2.25 Method getNextArc 27
6.2.26 Method toString 27
6.2.27 Method distanceOfPoints 28
6.2.28 Method distaneTo 28
6.2.29 Method distanceTo 28
6.2.30 Method compareTo 28
6.2.31 Method checkArc 28
6.2.32 Method checkSpeeds 28

7 File Reading 29

7.1 FileController 29
7.1.1 Member variables 30
7.1.2 Method FileController 30
7.1.3 Method getErrorMessage 30
7.1.4 Method getStatus 30
7.1.5 Method getWeatherMap 30
7.1.6 Method update 30
7.1.7 Method prepare 31
7.2 FileInterface 31
7.2.1 Method FileController 31
7.2.2 Method getWeatherMap 31
7.2.3 Method FileController 31

7.2.4 update . . . 31

7.3 ReadImageFile . . . 31

7.3.1 Method ReadImageFile . . . 32

7.3.2 Method getErrorMessage . . . 32

7.3.3 Method getMap . . . 32

7.3.4 Method getMapMatrix . . . 32

7.3.5 Method getStatus . . . 32

7.3.6 Method update . . . 32

7.4 ReadPGMImageFile . . . 32

7.4.1 Method ReadPGMImageFile . . . 32

7.4.2 Method getErrorMessage . . . 32

7.4.3 Method getMap . . . 33

7.4.4 Method getMapMatrix . . . 33

7.4.5 Method getStatus . . . 33

7.4.6 Method update . . . 33

7.5 WeatherMap . . . 33

7.5.1 Member variables . . . 33

7.5.2 Method apply . . . 33

7.5.3 Method getMap . . . 33

7.5.4 Method getMapMatrix . . . 34

7.5.5 Method getScale . . . 34

7.5.6 Method getLat . . . 34

7.5.7 Method getLon . . . 34

8 Graphics 35

8.1 Graphics . . . 35

8.1.1 Method update . . . 35

8.1.2 Method showErrorMessage . . . 36

8.1.3 Method showMessage . . . 36

8.2 GraphicsWindow . . . 36

8.2.1 Method update . . . 36

8.2.2 Method buildWindow . . . 36

8.2.3 Method actionPerformed . . . 36

8.3 GraphicsEngine . . . 36

8.3.1 Method GraphicsEngine 37
8.3.2 Method buildNextFrame 37
8.3.3 Method getFrame 37
8.3.4 Method getGraphics 37
8.3.5 Method getStatus 37
8.3.6 Method setDebugMode 37
8.4 outputPane 37
8.4.1 Method update 37
8.4.2 Method paint 38
8.5 infoPane 38
8.5.1 Method update 38
8.5.2 Method paint 38

1 Vocabulary

Airport Airport is where arrival tree begins, in the middle of the map.

Arc Arcs are circles at a determined radius distance of the airport. The merge points are located into these arcs.

Arrival tree A binary tree consisting of paths. Has a root at the airport.

Checkstyle Java code review for Eclipse.

dbZ dBZ stands for decibels of Z. It is a meteorological measure of equivalent reflectivity (Z) of a radar signal reflected off a remote object.

EclEmma Java Code Coverage for Eclipse.

Flight plan Every plane has a flight plan which describes its path.

FMI Finnish Meteorological Institute.

Integration Testing Integration testing purpose is to assure that integrated classes do all those services they are planned to do in requirement document.

Java2D Display and print 2D graphics in Java programs.

JAR Runnable Java archive, which based on the ZIP file format.

JUnit JUnit testing framework.

Map A map from somewhere in the world used in this product.

Merge point A point on the map where two paths merge into one path.

nmi nautical mile (=1,8520km)

Path A route to the airport that should avoid storms.

PGM Portable Gray Map, a graphics file format.

Plane An airplane that tries to land at an airport along a path avoiding storms.

PMD Java code review for Eclipse.

Storm A set of pixels with a dBZ-value over a certain threshold (that is a parameter) close each other on the map. Indicated with red color on the map.

System testing System testing purpose is to assure that software corresponds it?s requirements.

User A person using the product to watch animations on aircrafts landing at an airport in presence of hazardous weather systems.

Unit testing Unit testing purpose is to assure that certain class or unit do all those services it is planned to do in requirement document.

2 General Architecture

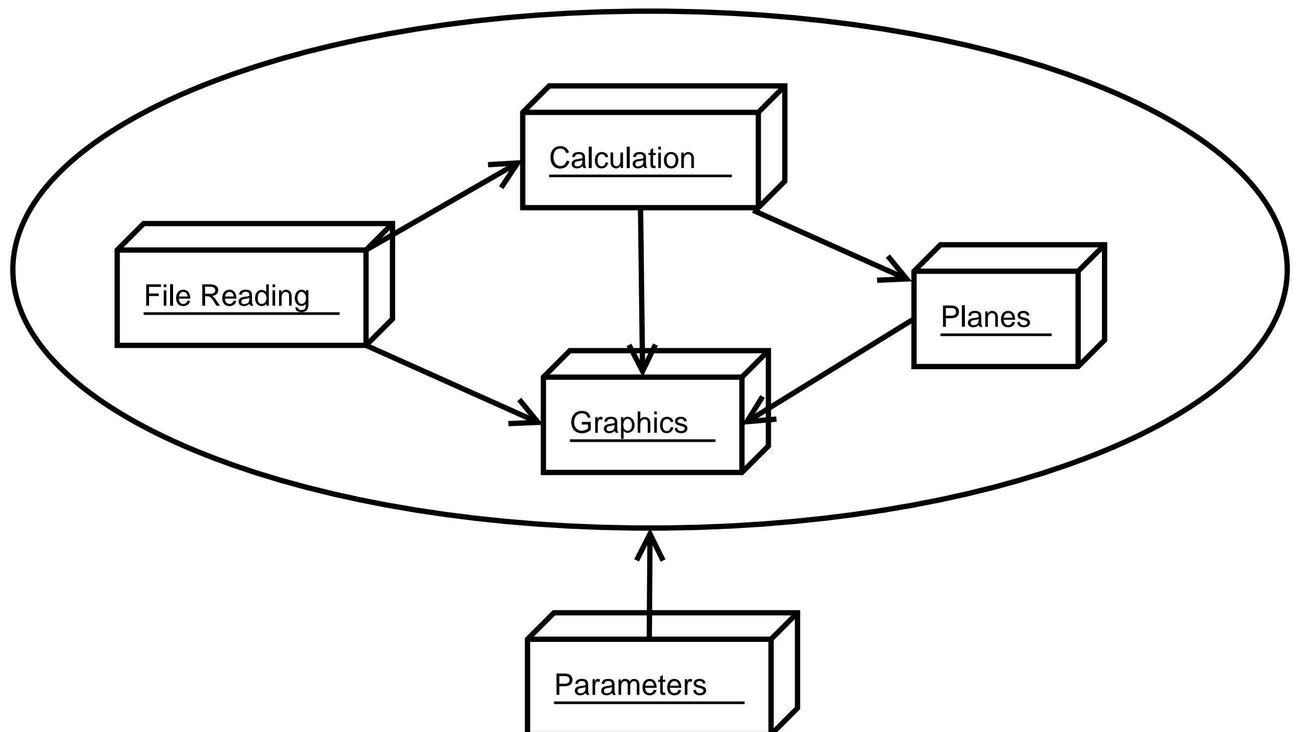
This section describes the architecture design of the program. Some changes are still possible during the implementation of this computer system.

The four parts of the system are *file reading*, *calculation*, *graphics presentation* and *planes*. All of these will get their parameters from the programs *parameters* class. The interface between these parts is the *DataCollection* class. In the *file reading* part, the data files will be read, and converted to a format suitable for the calculation to use. It is also responsible for updating the weather data.

The *calculation* system works as an independent part. It reads the current weather matrix, and updates the graph. After that it calculates (if needed) the new arrival tree for air planes. A recursive algorithm using A* algorithm will be used to find the arrival tree with the shortest paths.

An interface between the user and program is the *graphics* part. It will use the weather map read by the file reading part, and the tree from the calculation. Together these will form a map for the airplanes to follow.

Airplanes part is a set of planes following the tree. It does not really have any intelligence in it, other than the planes following parent nodes.



3 Chopper

This section describes the main part of the program and the start is performed.

3.1 Chopper

Chopper is a class for starting the program and reading parameter data into memory to be used by other classes. *Chopper* includes the main-function.

4 General

The general section is the package of the program, which contains all miscellaneous and common classes.

4.1 DataCollection

All information between graphics and calculation -packages is moved using DataCollection -class.

4.1.1 Method DataCollection

Constructor of the class, which takes no parameters.

4.1.2 Method setAirControlClass

Public method which takes AirControl-class as a parameter and store it.

4.1.3 Method getAirControlClass

Puhlic method which returns current instance of AirControl-class.

4.1.4 Method setArcPoints

Public method which takes Vertex ArrayList as parameter and store it.

4.1.5 Method getArcPoints

Public method which returns current instance Vertex ArrayList.

4.1.6 Method setCalculationControl

Public method which takes CalculationControl-class as parameter and store it.

4.1.7 Method getCalculationControl

Public method which returns current instance CalculationControl -class.

4.1.8 Method setFlagMap

Public method which takes boolean value as parameter and store it.

4.1.9 Method getFlagMap

Public method which returns current boolean value of flagMap.

4.1.10 Method setRefreshRate

Public method which takes numerical long value as parameter and store it.

4.1.11 Method getRefreshRate

Public method which returns value of refreshRate.

4.1.12 Method setInternalClock

Public method which takes numerical integer value as parameter and store it.

4.1.13 Method getInternalClock

Public method which returns value of internalClock.

4.1.14 Method setGraph

Public method which takes Vertex-list as parameter and store it.

4.1.15 Method getGraph

Public method which returns Vertex-list.

4.1.16 Method setGraphicsClass

Public method which takes Graphics-class as parameter and store it.

4.1.17 Method getGraphicsClass

Public method which returns current instance of Graphics-class.

4.1.18 Method setWeatherMap

Public method which takes WeatherMap-class as parameter and store it.

4.1.19 Method getWeatherMap

Public method which returns current instace WeatherMap-class.

4.1.20 Method setTreeClass

Public method which takes Tree-class as parameter and store it.

4.1.21 Method getTreeClass

Public method which returns current instance of Tree-class.

4.2 Parameter

This section describes the programs parameter class part.

4.2.1 Parameter

Parameter is a class for storing parameter values from parameter file given as a program parameter in the command-line.

4.3 Parameters

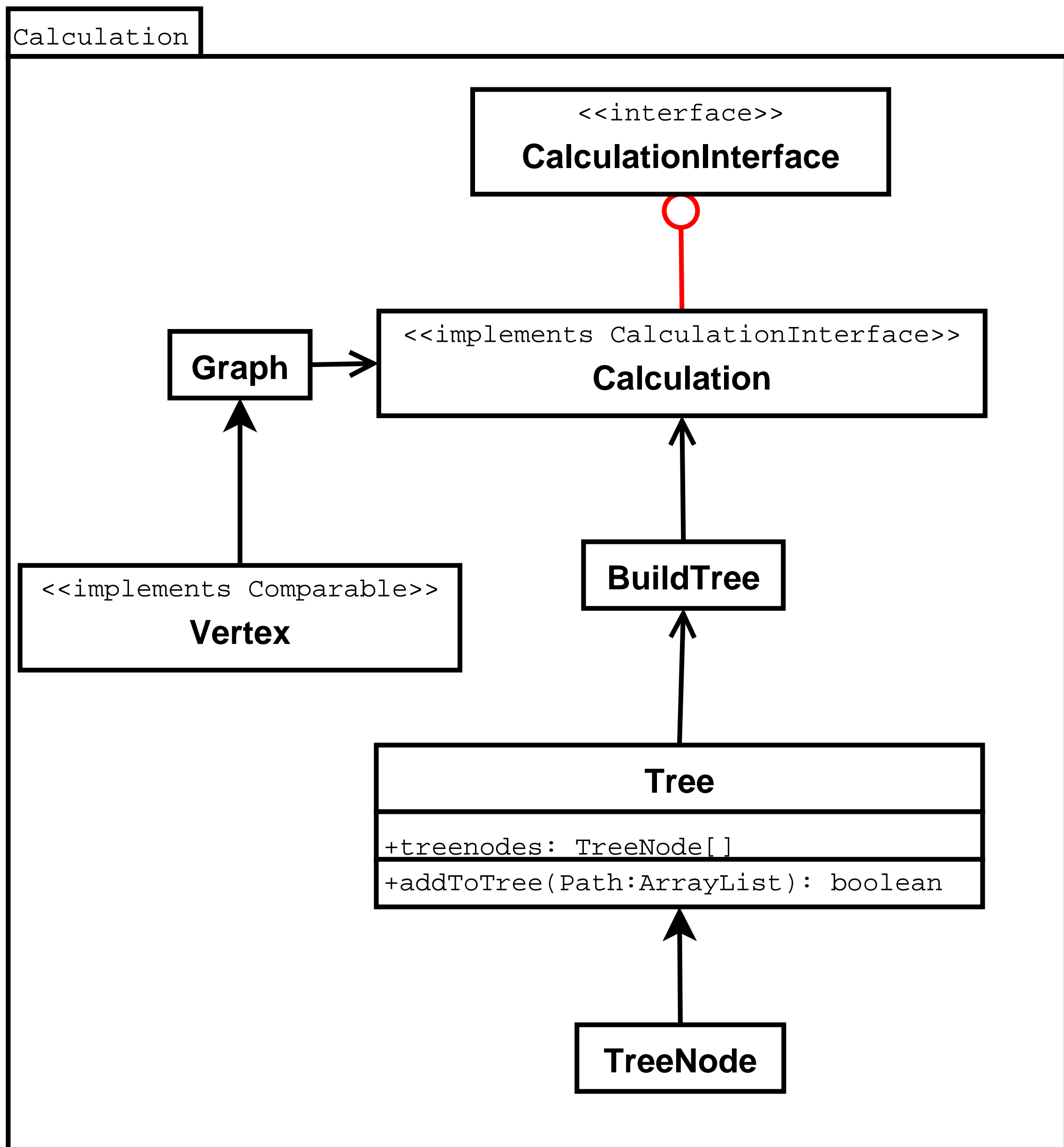
Parameters is a class for reading parameter values from parameter file given as a program parameter in the command-line. In this class all the parameters will be red into memory so that other classes can use them.

4.3.1 Method getParameterValue

Returns parameter value as string.

5 Calculation

The Calculation section is the part of the program, where the arrival tree for air planes is calculated.



5.1 CalculationControl

CalculationControl (or just Calculation) is the class that controls the calculation process of the arrival tree and changes the graph as the weather changes. When program starts, main method of class *Chopper* calls the constructor of CalculationControl, which creates

instances of classes needed in calculation.

5.1.1 Member variables

Variable	Type	Visibility	Short description
airControl	AirControl	private	Instance of class AirControl, which rules flying planes.
dataCollection	DataCollection	private	Parameter information.
fileController	FileController	private	Instance of class FileController, which rules reading weather files.
flagSystemTimer	boolean	private	When set to be true, method update loops.
graph	Graph	private	Graph used in calculation.
buildTree	BuildTree	private	Instance of the class BuildTree, which does the calculation process of the arrival tree.
internalClock	int	private	An update variable.
updateIntervalMap	int	private	An update variable.
updateIntervalAirspace	int	private	An update variable.
updateIntervalGraphics	int	private	An update variable.
timerUpdateMap	int	private	An update variable.
timerUpdateAirSpace	int	private	An update variable.
timerUpdateGraphics	int	private	An update variable.
testMode	boolean	private	Testing variable.

5.1.2 Method CalculationControl

The constructor method of the class. Gets DataCollection *dataCollection* as a parameter and sets member variable *dataCollection* to be as it. Sets some member variable values from class *Parameters*. Defines some update settings and does the first updating. Creates instances of classes *FileController*, *Graph*, *AirControl* and *BuildTree*. Sets the member variable *flagSystemTimer* to be true.

5.1.3 Method runTimer

Public method which takes no parameters. Repeats the same loop as long as the member variable *flagSystemTimer* is set to be true. On every loop method *update* is called. If update -procedure takes less than timerInterval -parameter defines, we perform some garbage collection and put our thread sleep for the duration.

5.1.4 Method setTimerFlag

Sets the member variable flagSystemTimer to be as boolean value given in parameters.

5.1.5 Method update

Private method which takes no parameters. Does the updating process of the program by updating member instances of classes *Graph*, *AirControl*, *BuildTree* and *FileController*. Update variables *timerUpdateMap*, *timerUpdateGraphics* and *timerUpdateAirspace* are increased by one in the beginning of the method, and it is checked by watching their values, which instances are to be updated.

5.2 BuildTree

BuildTree is the class, that calculates the arrival tree for air planes from given graph. When method *calculate* is called with graph as a parameter, the whole calculation is done immediately and the arrival tree is returned. Calculation is done with a recursive algorithm that uses the A* algorithm.

5.2.1 Member variables

Variable	Type	visibility	Short description
graph	Graph	private	Graph in which the arrival tree will be calculated.
arrivalTree	Tree	private	Arrival tree calculated for planes.
dataCollection	DataCollection	private	Data collection from where some parameters are taken.
safetyDistance	int	private	safety distance for arrival tree branches.
flagArrivalTreeUpdate	boolean	private	is set true when tree updates.
asked	boolean	private	An update variable.
pathCounter	int	private	Keeps count on "parent" paths while calculation is done.
finalPathCount	int	private	Keeps count on paths while calculation is done.
mergePoints	ArrayList<Vertex>	private	Merge points of the arrival tree.

5.2.2 Method BuildTree

The constructor method of the class. Takes DataCollection *dataCollection* as a parameter. Sets this.*graph* and *this.arrivalTree* to be as nulls. (A new tree will be created for every calculation when method *calculate* is called, because old trees will be needed in some other places in the program.) Sets *this.dataCollection* as parameter *dataCollection* and creates a new ArrayList for *this.mergePoints*. Sets *this.safetyDistance* using *dataCollection*.

5.2.3 Method getGraph

Returns *this.graph*.

5.2.4 Method getTree

Returns *this.arrivalTree*.

5.2.5 Method update

Public method, which takes Graph *graph* as a parameter. Sets *this.graph* to be as parameter *graph*. if parameter was null or *checkTree* returns true, calculation is done again (*calculate* method is called). Otherwise, it is not. Returns boolean value.

5.2.6 Method isChanged

Returns *this.flagArrivalTreeUpdated*.

5.2.7 Method aStar

Takes Vertex *start* and Vertex *goal* as parameters. Is an implementation of A* algorithm in Java. Calculates the shortest path in *this.graph* from *start* to *goal*.

A* uses the evaluation function $f(n) = g(n) + h(n)$ to decide which vertex it visits next. $g(n)$ is the cost from *start* to Vertex n and $h(n)$ distance from n to *goal*. A* keeps vertices it has not visited but which are adjacent to visited vertices in a priority queue called *open set*. Each time it visits a new vertex it moves this from *open set* to *closed set* which is an *ArrayList* of visited vertices. The priority value of vertex n in *open set* is $f(n)$ so every time the vertex with the lowest evaluation value is the next one to be visited. If many vertices has the (same) lowest value, the one to be chosen is the vertex which is in the "right direction". A* stops when *goal* is added to *closed set* or when *open set* is empty (no path was found). Before stopping A* also resets the fields of vertices it has changed. A* returns the path as an *ArrayList* of vertices or *null* if no path was found.

5.2.8 Method calculate

Public method which takes no parameters. Creates a new Tree to be as member variable *arrivalTree*. Sets the airport node to be the start node and the list of all the points in outermost arc to be the list of end nodes. Sets some member variables to be right before new calculation process. Calls the method *treeRecursion* with these values. (Method *treeRecursion* constructs *this.arrivalTree* to be as a correct arrival tree for planes.)

5.2.9 Method checkTree

Private method which takes no parameters. Checks, if there are storms over *this.arrivalTree*. If so, returns true. Otherwise returns false.

5.2.10 Method treeRecursion

Private method which takes Vertex *startNode* and ArrayList<Vertex> *endNodeTable* as parameters. Is called by method *calculate*. Is a recursive function which, when called at first time, calls itself as long as there are elements in variable *queue*. While running, constructs the arrival tree for airplanes. Operates in a following way:

1. Sets variable *endNodes* to be as *endNodeTable* got as parameter. This is the end node table on which the calculation will be done in this round of recursion. Makes a new ArrayList<Vertex> as variable *parEndNodes*, which will be the end node table given as parameter to next recursive call. Defines boolean value *side*, which is value telling on which side of the *endNodes* did the end of the path just calculated go.

2. Sets area around *startNode* to be available "slidingly" using methods *setSafetyDistance* and *setSlidingSafetyDistance*. Defines variable *path* by finding the shortest path from *startNode* to *endNodes* using method *calculatePath*. Sets area around *startNode* to be unavailable again. Increases values *this.pathCounter* and *this.finalPathCount* both by one.

3. If path just found was not empty/null, adds all vertexes that are merge points from *path* to *queue*. Puts there only first ones from each arc found, so that in *queue* there is only one point from each arc. Adds *path* to *this.arrivalTree* using method *addToTree*. Sets safety distance around *path* using method *setSafetyDistance*.

4. As long as the *queue* is not empty, repeats the following:

Finds the side on which side of the *endNodeTable* did *path* go using method *testSide*.

Depending on which side did *path* go or was there only one node in *endNodeTable*, sets *endNodes* and *parEndNodes* to be as halves of the previous *endNodeTable*. Uses method *halftable* to divide *endNodeTable* in half.

Sets *parStartNode* to be as first node in *queue* and removes that node from *queue*.

If *parEndNodes* is not null, calls recursively method *treeRecursion* (=itself) with values *parStartNode* and *parEndNodes*. If *treeRecursion* is not called, decreases value of *this.pathCounter* by one, to know from which path does the next path start.

Last, allways decreases *pathCounter* value by one.

5.2.11 Method setSlidingSafetyDistance

Private method which takes Vertex *startNode*, boolean *sliding* and boolean *availability* as parameters. Sets a safety distance for nodes from *startNode* to as many as (1*x + 2*x + ... + x*x) nodes under it, when x = *this.safetyDistance*. If parameter *sliding* is true, the first nodes (having multiplier one) are set to have a safety distance of one, the second nodes

the safety distance of 2, etc., as long as safety distance is *this.safetyDistance*. If parameter *sliding* is false, safety distance of all these nodes is set to be *this.safetyDistance*.

5.2.12 Method calculatePath

Private method which takes Vertex *startNode* and ArrayList<Vertex> *endNodes* as parameters. Finds the shortest path from *startNode* to all *endNodes* using method *aStar*. Saves the shortest of them all in an ArrayList. Returns the path calculated as a ArrayList<Vertex> *shortestPath*.

5.2.13 Method setSafetyDistance

Public method which modifies availability values of vertexes in a particular area around the certain vertex. Takes Vertex *vertex*, boolean *availability*, int *distance* and boolean *set* as parameters. When the safety distance is set for the first time for a new path (its nodes *pathSafetyDistance* values are zero), each nodes *pathSafetyDistance* value in path and around it is set to be as *this.finalPathCount*. For every followers of the *vertex* (each 8 directions in graph) sets variable *parAdj* to be the start node of recursion. If *parAdj* is not null and its *pathSafetyDistance* value is same as *this.pathCounter* now (for avoiding setting wrong paths safety distance off), sets its availability to be as parameter *availability* and calls method *setSafetyDistance* (=itself) with parameter values *parAdj*, *availability* and *distance - 1* (if *distance* value is greater than 1).

5.2.14 Method testSide

Takes ArrayList<Vertex> *path* and ArrayList<Vertex> *endNodes* as parameters. Tests, whether do the last node of parameter *path* locate on left or right side of *endNodes*. If there happens to be a pairless amount of nodes in *endNodes* and last node of *path* went just in middle of *endNodes*, tells that path went to right. Returns integer value 0 if path went left, 1 if it went right. If there was no success, returns 2. If *endNodes* was null or size zero or one, returns 3.

5.2.15 Method setSafetyDistanceAirPlanes

Public method, which takes Vertex *vertex*, boolean *availability* and int *distance* as parameters. Acts very much like method *setSafetyDistance*, except does not mind about safety distances of other paths when removing safety distance.

5.2.16 Method halftable

Takes ArrayList<Vertex> *endNodes* and boolean *side* as parameters. Takes half of the table *endNodes* and returns it as a ArrayList<Vertex>. Parameter *side* defines, which side to take and return from *endNodes*. If there is a pairless amount of nodes in *endNodes* and

it is required to return the right side of it, puts the one node in a middle also to a return array. If left side is taken, returns only half or smaller half of the ArrayList.

5.3 Graph

Graph class is a net presentation of the image created. It consists of vertices, that are linked to adjacent ones. The list of the adjacent ones is a simple array. Every node has at least three (3), and a maximum of eight (8), neighbors. Simply having an array of adjacent nodes if quite enough for the A* algorithm. However, the location, that is, the coordinates (latitude and longitude), are known. Also the information about direction of neighbors is known by the array cell.

5.3.1 Member variables

Variable	Type	Short description
graph	Vertex[][]	A two dimensional matrix of the graph.
arcpoints	ArrayList[]	An *ArrayList* array of arcpoints.
startpoints	Vertex[]	An array of vertices, that are the starting points for the airplanes.
mapMatrix	int[][]	A two dimensional matrix of weather in integer format.
map	WeatherMap	A *WeatherMap* class address.
cornerlat	double	Map corner latitude.
cornerlon	double	Map corner longitude.
start	double	Arcpoints sector start angle.
end	double	Arcpoints sector end angle.
startPointCount	double	Number of startpoints in the sector.
nullVertex	Vertex	A nullVertex instance address.
param	Parameters	Parameters class.

5.3.2 Method Graph

The constructor method for this class creates the graph. It takes a DataCollection *data*, int *width* and int *height* as a parameters. The *data* knows the address of the *WeatherMap*. A two dimensional presentation of the map is included in the *WeatherMap*. All information is then available for the graph creation. The matrix can be read and converted to a graph, using the createGraph method. Also the airport location is set.

The reference to the graph is stored in the attributes of the class.

5.3.3 Method createGraph

This method makes a certain size graph. It takes int *width* and int *height* as parameters, and creates a two dimensional array of *Vertex* instances. After that, the vertices are linked to each other. If something goes wrong, the method returns null. Otherwise the address of the graph.

5.3.4 Method getGraph

Takes no parameters. Returns *this.graph*.

5.3.5 Method calculateArcPoints

Takes double *start*, double *end* and double *startPointCount* as parameters. Finds the *merge point* candidates from the graph. Goes thought vertices at a certain distance from the airport. All that are in defined range for a *merge point* will be listed in the *arc-points* two dimensional array. This will then be stored as a local variable *this.arcPoints*. *this.arcpoints* array will never have to be updated, even though the graph might change.

5.3.6 Method getArcPoints

Does not take parameters. Returns *this.arcPoints*.

5.3.7 Method getOuterArc

Takes no parameters. Returns the outermost arcs points, in an *ArrayList* instance.

5.3.8 Method getAirport

Takes no parameters. Returns *this.airPortVertex*.

5.3.9 Method update

Requires no parameters. It is used for changing the weather. As the time passes, the program calls for update, to re-read the weather to the graph, due to possible changes. This is done using the *graphUpdate* method. If updating was failed, the method returns false, else true.

5.3.10 Method graphUpdate

This method takes four parameters: int *scale*, int[][] *matrix*, Vertex[][] *graph* and int *safety*. The matrix is scaled down to a smaller graph, using the *checkSquare* method.

Returns boolean value about succession.

5.3.11 Method checkSquare

Takes int *size*, int *x*, int *y* and int[][] *matrix* as parameters. Looks for some integer value at some part of the matrix. If such value is found, true is returned. The parameters size, x and y define the coordinates where to look for. The size tells how large square to check.

5.4 Vertex

The *Vertex* class is made to be a graph node. It is designed to support the A* algorithm for path finding.

5.4.1 Member variables

Variable	Type	Visibility	Short description
adj	Vertex[]	private	a list of adjacent *Vertices*.
state	int	private	This tells if this *Vertex* is in A* closed set or open set or if it has not been visited yet.
available	boolean	private	Tells if this *Vertex* is passable or not.
lat	double	private	Latitude coordinate.
lon	double	private	Longitude coordinate.
x	int	private	X-coordinate.
y	int	private	Y-coordinate.
gCost	double	private	distance from start *Vertex* of A* to *this*.
hCost	double	private	distance from this to goal *Vertex*.
parent	Vertex	private	Parent of this vertex in a path that A* creates.
distCost	double	private	Cost to goal *Vertex* of A* as straight line distance .
bestDir	int	private	Tells if this *Vertex* is in the "best direction" for A*.
storm	boolean	private	Is this a storm vertex.
stormSafety	booleac	private	Tells if this vertex is in storm safety distance area.
pathSafety	int	private	Tells if this vertex is in path safety area, value identifies path number..

5.4.2 Method Vertex

The constructor method. Takes double *lon*, double *lat*, int *x*, int *y*, boolean *available* and Vertex *nullVertex* as parameters.

5.4.3 Method reset

Takes no parameters. Resets the fields *this.state, this.gCost, this.hCost, this.parent, this.distCost* and *this.BestDir* (the fields A* changes) so that when A* is called again, these fields have their default values.

5.4.4 Method distanceTo

Takes Vertex *other* as a parameter. Returns the straight line distance to *other* in a graph.

5.4.5 Method graphDistanceTo

Takes Vertex *other* as a parameter. Returns the distance to *other* in a graph.

5.4.6 Method calculateDir

Takes Vertex *goal* as parameter. Returns the direction from *this* to *goal* as radians.

5.4.7 Method compareTo

Takes Vertex *other* as parameter, and compares the evaluation function value f (= gCost + hCost) of *this* to *other*. Because of this method *Vertex* class can implement Comparable interface and vertices can be compared. A* needs this property because it keeps vertices in a priority queue.

5.5 Tree

Tree is the class that implements the tree that will be returned from calculation.

5.5.1 Member variables

Variable	Type	Visibility	Short description
treeNodes	ArrayList<TreeNode>	private	List of Vertexes the tree includes. (No matter in what order they are presented.)
root	TreeNode	private	Root of the tree.
treeMergePoints	ArrayList<TreeNode>	private	List of merge points in a tree.
vertexTree	ArrayList<Vertex>	private	List of tree vertexes in same order tree Nodes are in this.*treeNodes*.

5.5.2 Method Tree

The constructor method of the class. Creates empty ArrayList for *this.treeNodes*, *this.treeMergePoints* and for *this.vertexTree*. Sets *this.root* to be as null.

5.5.3 Method getRoot

Returns *this.root*.

5.5.4 Method getTreeMergePoints

Returns this.*treeMergePoints*.

5.5.5 Method setTreeMergePoints

Public method which gets ArrayList<Vertex> *MergePoints* as parameter. Finds corresponding tree nodes from *this.treeNodes* for those vertexes and sets them to be in *this.mergePoints*.

5.5.6 Method getTree

Returns *this.treeNodes*.

5.5.7 Method getVertexTree

Returns *this.vertexTree*.

5.5.8 Method treeNodePath

Public method which takes ArrayList<Vertex> *path* as a parameter. Creates a new ArrayList<TreeNode> *treeNodeTable*. Makes new TreeNodes of given vertexes in a path. Sets parenthood -relationships correct in the *treeNodeTable*. (Does not care about childhood -relationships.) Returns *treeNodeTable*.

5.5.9 Method addToTree

Takes ArrayList<Vertex> *path* as a parameter. Operates in a following way:

1. If *this.treenodes* is empty, makes corresponding TreeNodes for the first and the second one vertex in a path and adds them in *this.treeNodes*. Adds those vertexes to *this.vertexTree*. Makes parenthood-childhood -relationships between first two in *this.treeNodes* correct.

2. If *this.treeNodes* is not empty, finds the first vertex of the path from *this.treenodes*. If the first one is found, makes corresponding TreeNode for the second one vertex in

path and adds it to *this.treeNodes*. Adds corresponding vertex to *this.vertexTree*. Makes parenthood-childhood -relationships between the one allready in *this.treeNodes* and the one added to tree, correct.

3. Finally, makes corresponding TreeNodes of all the rest vertexes from path and adds them to *this.treeNodes* and adds vertexes from path to *this.vertexTree*. Makes parenthood-childhood -relationships between nodes correct.

When setting left or right child of some node, first inserted child is always left, second is allways right. It does not matter if the children are not geographically correctly left or right, because nodes have knowledge about place. (They are left or right only because it is easy to get tree printed in graphics if there are children, planes need only information about parents.)

Returns true if there was success, otherwise returns false.

5.5.10 Method getLeaves

Takes no parameters. Checks all the leaves from tree by finding all the nodes without any children. Returns ArrayList of these tree nodes.

5.5.11 Method getVertexLeaves

Takes no parameters. Uses method getLeaves to take leaves from *this.treeNodes* as a vertexes. Puts these vertexes in a ArrayList<Vertex> *treeLeaves* and returns it.

5.5.12 Method toString

Public method which prints out all the tree nodes x and y values.

5.6 TreeNode

TreeNode is the class that implements a node of a tree in calculation of flight paths.

5.6.1 Member variables

Variable	Type	Short description
left	TreeNode	Left child of a node in a tree.
right	TreeNode	Right child of a node in a tree.
parent	TreeNode	Parent of a node in a tree.
vertex	Vertex	Tells the id of the Vertex that this tree node represents.

5.6.2 Method TreeNode

The constructor method of the class. Gets Vertex *nodeId* as a parameter. Sets all values to be nulls, except *vertex*, which will be set to be the same as a given parameter value.

5.6.3 Method setChild

Takes TreeNode *child* as a parameter. If tree has not got left child yet, sets *child* to be *this.left*. If *this.left* is allready there, sets *child* to be *this.right*.

5.6.4 Method setParent

Takes TreeNode *parent* as a parameter. Sets *this.parent* to be as *parent*.

5.6.5 Method getLeft

returns *this.left*.

5.6.6 Method getRight

returns *this.right*.

5.6.7 Method getParent

returns *this.parent*.

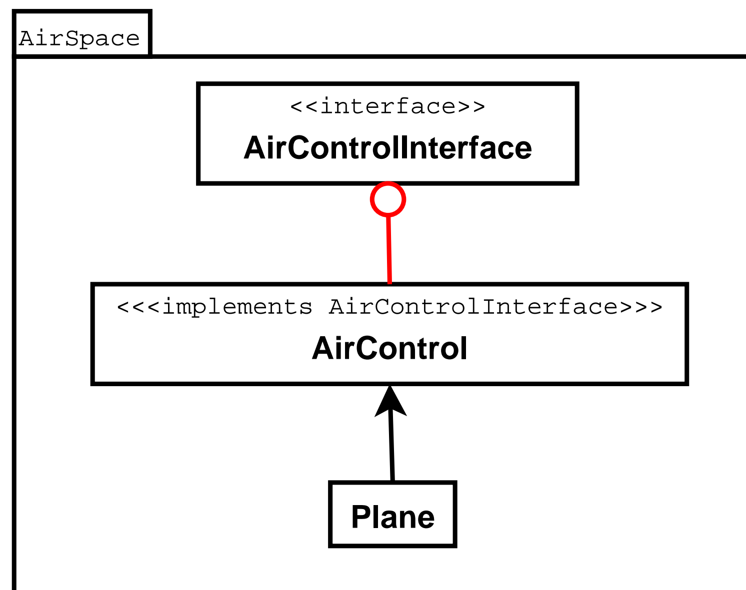
5.6.8 Method getVertex

returns *this.vertex*.

6 AirSpace

6.1 AirControl

AirControl class takes care of the planes (eg.. creates, removes and calls fly method for Planes). It controls Planes on map and provides up to date information of planes locations and numbers to Graphics for drawing and displaying.



6.1.1 Member variables

Variable	Type	Visibility	Short description
leafNumber	int	private static	Number of leaf where to create Plane.
planeCreateCounter	int	private static	Counter for often create plane(s).
arcPoints	ArrayList[]	private	Arc Points.
arrivedPlanes	int	private	How many planes has landed.
buildTree	BuildTree	private	Instance of BuildTree.
colorAlert	Color	private	Alert color of Plane.
colorNormal	Color	private	Normal color of Plane.
createmultiplier	int	private final	How many planes create at a time.
minSafeDistance	int	private	Minimun safe distance between Planes.
numberOfPlanes	int	private	How many planes does AirControl have.
parameters	Parameters	private	Instance of Parameters class.
planeCreateInterval	int	private	How often plane is created.
planes	ArrayList<Plane>	private	ArrayList of planes on map.
tree	Tree	private	Instance of Tree class.

6.1.2 Method AirControl

Constructor method of the class. Takes Parameters *parameters* and ArrayList[] *arcPoints* as parameters. Sets *this.planes* to be as ArrayList for Planes. Sets *this.parameters* to be as parameters as given and same for *this.arcPoints*. *This.planeCreateInterval*, *this.planeMaxSpeed* and *this.minSafeDistance* are taken from Parameters. *This.buildTree* and *this.tree* are set to be as nulls. *This.numberOfPlanes* and *this.arrivedPlanes* are set to be as 0.

6.1.3 Method update

Public method that initiates update of the planes' positions on map. This happens by calling method *controlPlanes*.

6.1.4 Method setBuildTree

Public method that sets *this.buildTree* from parameters.

6.1.5 Method getAirportStatus

Public method that returns true if there is an arrival tree, otherwise false.

6.1.6 Method getMaxArrivingRate

Public method that returns maximun value of how many Planes can land in an hour in theory.

6.1.7 Method getPlanes

Public method that returns ArrayList of Planes.

6.1.8 Method getNumberOfPlanes

Public method that returns number of Planes landing.

6.1.9 Method getArrivedPlanes

Public method that returns number of Planes that have already landed.

6.1.10 Method addPlane

Private method that creates Planes on current tree's leaves. All Planes are added to *this.planes* and number of planes landing is increased.

6.1.11 Method controlPlanes

Private method that controls Planes. Operates in a following way:

1. Gets newest Tree.

2. Checks if root is null and if it is calls flyPlane method for every plane.

3. if tree is not null calls addPlane that creates Planes.

4. Goes through Planes one at a time if tree has changed and Plane is not on tree Plane calls newPath for every Plane.

5. Tells all Planes to move by calling flyPlane.

6. Checks if any Planes are too close to some other Plane.

6.1.12 Method closestMergePoint

Private method that finds closest merge point to Plane. Takes Plane *plane* and int *arc* as parameters. If arrival tree doesn't exists returns null and if *arc* is 0 returns root of tree. Sorts merge points list and goes through this list to find closest one to Plane. Closest TreeNode found is returned.

6.1.13 Method closestTNToPlane

Private method that finds closest merge point to Plane. Takes Plane *plane* as parameter. Returns root of arrival tree if next arc is 0. Method loops all TreeNodes on tree and finds one closest to Plane. Then method travels TreeNodes parents up certain value and returns that.

6.1.14 Method safetyZoneCheck

Private method that checks if any two planes are too close to each other. If two planes are too close the further away from airport will be colored and it's speed is reduced. If no Plane is found too close to current Plane it is colored blue and speed is increased.

6.1.15 Method flyPlane

Private method that calculates the distance how long a plane can travel and then tells plane to fly that amount. Takes Plane *plane* to fly as parameter and returns boolean value. Method is also used in setting direction of the plane.

6.1.16 Method newPath

Private method that checks what is the best way to guide a plane to closest merge point and asks *aStar* from *this.buildTree* to calculate this. Takes Plane *plane* as parameter. Operates in a following way:

1. Find closest TreeNode to Plane.

2. Check if Plane is closer to airport than found TreeNode and if it is find closest merge point.

3. Check if Plane is closer to found merge point if it is find merge point from inner arc.

4. Check if current TreeNode is same as found.

5. Check if Plane is in storm and clears path of Plane if true.

6. Make all Vertices's on arrival tree available for A*.

7. Call *aStar* method to calculate path from Plane to found TreeNode.

8. Create TreeNodes to Vertex path given by *aStar*.

9. Gives Plane it's new path.

6.1.17 Method sort

Private method that sorts ArrayList of TreeNodes to ArrayList that contains ArryLists. Method takes ArrayList<TreeNode> *al* as parameter. In *al* every TreeNode is then checked for distance to airport and categorized accordingly. Returns created ArrayList.

6.2 Plane

Instance of Plane class holds all the information about planes.

6.2.1 Member variables

Variable	Type	Visibility	Short description
color	Color	private	Color value of Plane.
currentNode	TreeNode	private	TreeNode where the plane has last been.
direction	double	private	direction of Plane in degrees.
distToAirport	double	private	Distance between Plane and airport.
lat	double	private	Latitude of the plane.
lon	double	private	Longitude of the plane.
maxspeed	double	private	Maximum speed of Planes.
minspeed	double	private	Minimum speed of Planes.
nextArc	int	private	Number of arc where plane is heading.
path	ArrayList<Vertex>	private	ArrayList of Vertexes that create flight path of Plane.
planeId	int	private	ID number of Plane.
root	TreeNode	private	Root of arrival tree.
safetyzone	double	private	Safety zone of the plane.
speed	double	private	Speed of the plane.

6.2.2 Method Plane

Constructor method of the class. Takes TreeNode *startNode* (as first TreeNode), ArrayList[] *arcPoints* (as arcpoints), Parameters *parameters* and TreeNode *root* as parameters. StarNode is set as *this.currentNode*. *This.lat* and *this.lon* are taken from currentNode. *This.safetyDistance* and *this.speed* are taken from *parameters*. *this.parameters* and *this.arcPoints* are taken from given parameters. *This.nextArc* is given arcPoints minus one. *This.path* is set as new ArrayList and after that path is created from currentNode to the airport. *This.color* is set blue and *this.planeId* is given.

6.2.3 Method fly

Public method that Calculates planes new location and checks if we arrive on merge point or airport. Takes double *travelDistance* (distance to travel) as parameter. Checks if path is only one in size and if it is returns false as we are at airport. If that is not true then gets coordinates of Plane and then the TreeNodes coordinates we are arriving in. Calculates distance between these points and if Plane travels this distance checks if Plane gets to arc, updates *this.currentNode* with next TreeNode of path and removes the TreeNode from path that it came from. Finally it calls this method again with remaining travel distance. If Plane doesn't get to next TreeNode it's coordinates are updated and true is returned.

6.2.4 Method setColor

Public method that sets given parameter Color *color* as Planes new *this.color*.

6.2.5 Method clearPath

Public method that clears path of Plane by calling ArrayLists method *clear*.

6.2.6 Method setCurrentTreeNode

Public method that takes TreeNode node as parameter. Sets *node* as *this.currentNode* for plane and updates planes coordinates from *this*.

6.2.7 Method setLat

Public method that takes double *latitude* as parameter. Sets latitude information of plane.

6.2.8 Method setLon

Public method that takes double *longitude* as parameter. Sets longitude information of plane.

6.2.9 Method isOnArrivalTree

Public method that checks if Plane still is on current Tree even it has changed. Takes TreeNode *node* and int hashCode as parameters. HashCode of Given TreeNode is checked if it matches given int. If it is not method is called again from TreeNodes child. Tree is walked from root to leaves until match is found. If match is found returns true else false.

6.2.10 Method addToPath

Public method that adds given path to path. Takes ArrayList<TreeNode> *pathToAdd* and TreeNode *joinPoint* as parameters. Method first clears the old path. Then it calls method *setPath* for first TreeNode of given path and again *setPath* for *joinPoint*.

6.2.11 Method setPath

Public method that sets flying path of plane. Takes TreeNode *currentNode* as a parameter. If *currentNode* is not null it is added to path and if it's parent is not null method is called again with parent of this TreeNode.

6.2.12 Method setSpeed

Public method that sets flying speed of Plane. Takes parameter double *speed*. *Speed* can't be lower than minimun speed nor greater than maximum speed.

6.2.13 Method setDistToAirport

Public method that sets distance of Plane to airport. Takes parameter double *dist* as distance.

6.2.14 Method setDir

Public method that sets the direction of the plane as degrees (0 - 360) based on latitude and longitude coordinates of two points. Takes four doubles *startLon, startLat, goalLon, goalLat* as parameters. Returns direction where Planes is flying as double.

6.2.15 Method getDistToAirport

Public method that returns distance of Plane from airport.

6.2.16 Method getDir

Public method that returns direction of Plane.

6.2.17 Method getColor

Public method that returns color information of Plane.

6.2.18 Method getCurrentNode

Public method that returns currentNode of Plane.

6.2.19 Method getId

Public method that returns ID information of Plane.

6.2.20 Method getLat

Public method that returns latitude information of Plane.

6.2.21 Method getLon

Public method thatReturns longitude information of Plane.

6.2.22 Method getPath

Public method that returns path of Plane.

6.2.23 Method getSpeed

Public method that returns speed information of Plane.

6.2.24 Method getSafetyZone

Public method that returns safety zone information of Plane.

6.2.25 Method getNextArc

Public method that returns next arc of Plane.

6.2.26 Method toString

Public method that returns String presentation of Plane.

6.2.27 Method distanceOfPoints

Public method that returns distance of two coordinates. Takes four doubles *currLat, currLon, targLat, targLon* as parameters. Returns distance between first pair and second pair of coordinates as double.

6.2.28 Method distaneTo

Public method that returns distance of two Planes on map. Takes Plane *other* as parameter. Returns distance between this and *other* as double.

6.2.29 Method distanceTo

Public method that returns distance of Plane and point on map. Takes two doubles *targLat* and *targLon* as parameters. Returns distance between this and given lat and lon coordinates as double.

6.2.30 Method compareTo

Public method that compares two Planes. Takes Plane *other* as parameter. Returns -1 if this is closer, 1 if other Plane is closer and otherwise 0.

6.2.31 Method checkArc

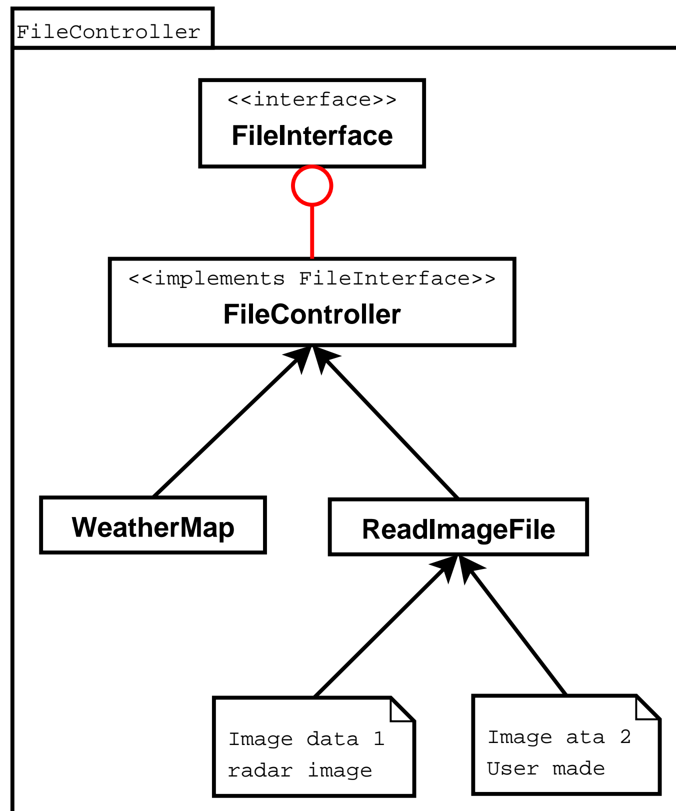
Private method that checks if is Plane is close enough to airport to be on inner arc than current.

6.2.32 Method checkSpeeds

Private method that corrects min and max speed values so that minspeed is at least 1 and maxspeed is at least minspeed +1.

7 File Reading

This section describes the programs file input part. The purpose of file reading class is to get the weather data into memory. This is done by reading text or image files in some format, and converting them to a *WeatherMap* class.



7.1 FileController

The *FileController* class handles the creation of all the classes in file reading package. It makes an instance of the *WeatherMap*, *ReadImageFile* and *ReadTextFile* classes, so it can use their methods when needed. This class also implements the *FileInterface* interface.

7.1.1 Member variables

Variable	Type	visibility	Short description
errorMessage	String	private	Error message for user.
mode	String	private	Tells which ReadImageFile - class is to be used.
parameters	Parameters	private	Instance of the class parameters.
readImageFile	ReadImageFile	private	Instance of the class ReadImageFile.
readPGMImageFile	ReadPGMImageFile	private	Instance of the class ReadPGMImageFile.
status	boolean	private	Status reveal if constructor fails.
weatherMap	WeatherMap	private	Instance of the class WeatherMap.

7.1.2 Method FileController

The constructor method of the class, which takes *Parameters* class as a parameter.

7.1.3 Method getErrorMessage

Method takes no parameters. Returns the latest error message.

7.1.4 Method getStatus

Method takes no parameters. Returns the status of class. Status is false if constructor or update fails.

7.1.5 Method getWeatherMap

This method returns the address of a *WeatherMap* class. This is needed for the calculation part of the program to create a graph.

7.1.6 Method update

The update method is used to command the lower level classes to update the map picture to a newer one.

7.1.7 Method prepare

Method pre-calculates data if it is possible. At the moment this is used only in "radar" mode.

7.2 FileInterface

FileController implements *FileInterface* class, which is an interface class between the file reading and the calculation packages. It defines the methods required to create a graph.

7.2.1 Method FileController

Defines the constructor method. During this, the *FileController* creates a *WeatherMap*, and decides which file mode to use, according to parameters.

7.2.2 Method getWeatherMap

Defines a method to return a *WeatherMap* class instance.

7.2.3 Method FileController

Defines a method for constructor.

7.2.4 update

Defines a method for updating weather images.

7.3 ReadImageFile

ReadImageFile is a class for reading weather data. It reads Portable GreyMap (PGM) images from Finnish Meteorological Institute's (FMI) radar, and converts them to form this program uses. Data is rather a set of radar pictures. Class has methods for converting the data into integer array and a *BufferedImage* format.

Note that the actual data is in polar coordinates, which is converted into PGM-images. In other words, every line in data represent here measurements from a specific direction. Measurements are calculated at intervals of 500 meters, and the furthest measure is as far as 250km. This means 500 measurement on one line. The single measurement is represented as a value between 0-254. Value 255 is a special case, it means that there is no measurement at this certain point. We convert the value x from PGM image to dBZ scale by formula: $Z[dBZ] = 0.5 * x - 32$

7.3.1 Method ReadImageFile

The constructor method for image file reading class. Method takes *Parameters* class as a parameter.

7.3.2 Method getErrorMessage

Method takes no parameters. Returns the latest error message.

7.3.3 Method getMap

Method takes no parameters. Returns the created map, in *BufferedImage* format.

7.3.4 Method getMapMatrix

Method takes no parameters. Returns the created map information converted into integer array.

7.3.5 Method getStatus

Method takes no parameters. Returns the status of class. Status is false if constructor or update fails.

7.3.6 Method update

Takes no parameters. Method is called to update the weather data. Returns boolean value succeed.

7.4 ReadPGMImageFile

ReadPGMImageFile is a class for reading weather data in PGM image format. This class has methods for converting the image to a *BufferedImage* class instance.

7.4.1 Method ReadPGMImageFile

Takes The constructor method for image file reading class. Method takes *Parameters* class as a parameter.

7.4.2 Method getErrorMessage

Method takes no parameters. Returns the latest error message.

7.4.3 Method getMap

Method takes no parameters. This method returns the created map, in *BufferedImage*.

7.4.4 Method getMapMatrix

Method takes no parameters. Returns the created map information converted into integer array.

7.4.5 Method getStatus

Method takes no parameters. Returns the status of class. Status is false if constructor or update fails.

7.4.6 Method update

Takes no parameters. Method is called to update the weather data. Returns boolean value succeed.

7.5 WeatherMap

The *WeatherMap* class retrieves information about the weather, as *BufferedImage* class. It also gets the two-dimensional array presentation of the weather. Then it stores them to its private variables.

7.5.1 Member variables

Variable	Type	visibility	Short description
width	double	private	map image width
height	double	private	map image height
scale	double	private	maps scale
cornerlat	double	private	map corner latitude coordinate
cornerlon	double	private	map corner longitude coordinate

7.5.2 Method apply

This method takes a *BufferedImage* and *MapMatrix* as parameters and stores them as local variables.

7.5.3 Method getMap

Method takes no parameters. This method returns the created map, in *BufferedImage*.

7.5.4 Method getMapMatrix

Method takes no parameters. Returns the created map information converted into integer array.

7.5.5 Method getScale

Method takes no parameters. Returns the scale of weather map. This property is not implemented in this version.

7.5.6 Method getLat

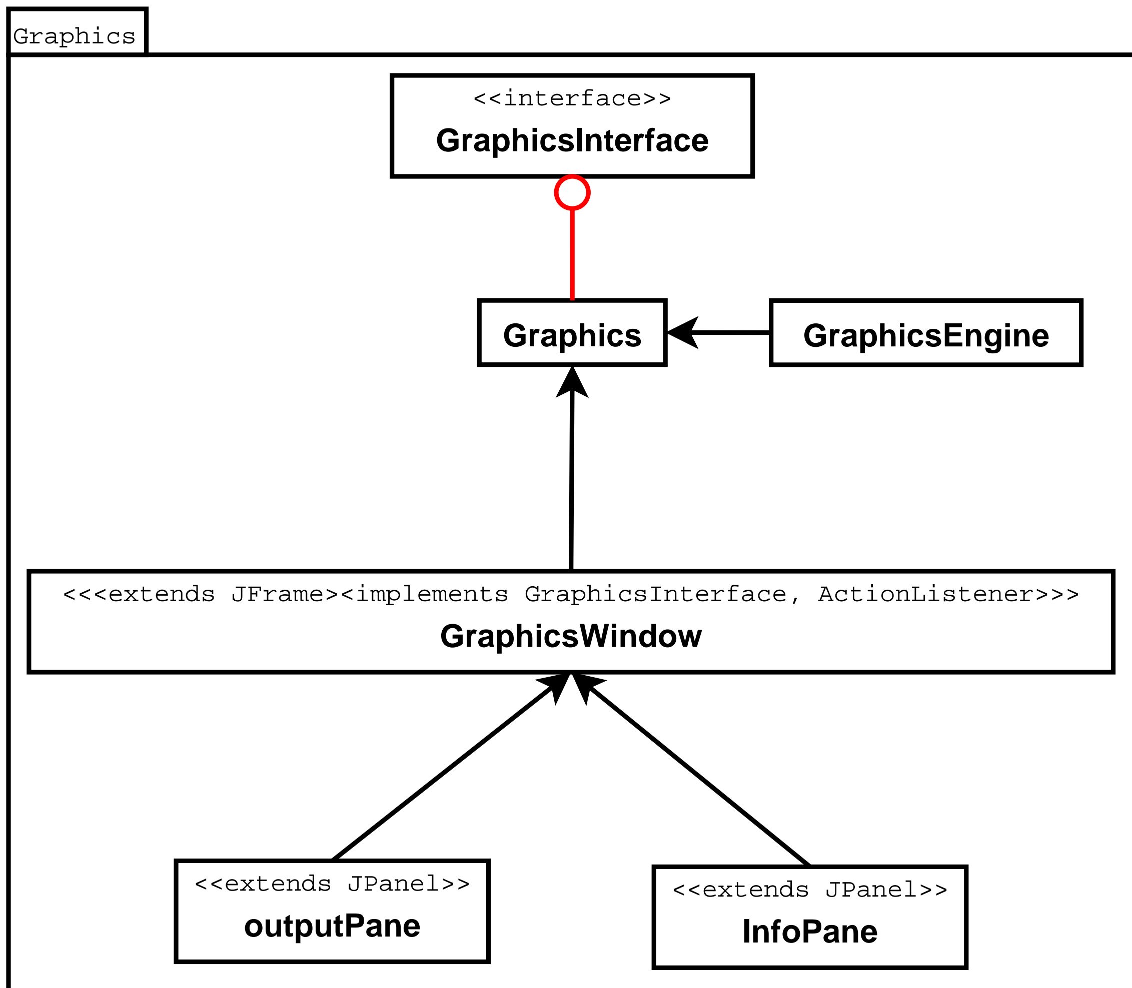
Method takes no parameters. Returns latitude of map. This property is not implemented in this version.

7.5.7 Method getLon

Method takes no parameters. Returns longitude of map. This property is not implemented in this version.

8 Graphics

This section describes the programs Graphics package. The purpose of package is to display the programs data structure on screen.



8.1 Graphics

Graphics class maintains the package classes and it implements GraphicsInterface. Constructor takes DataCollection -class as parameter.

8.1.1 Method update

Update is a public method, which takes no parameters. Updates graphics on screen.

8.1.2 Method showErrorMessage

ShowErrorMessage is a public method, which takes message and boolean value as parameter. Method shows error message in pop-up-window. True value as parameter reveal that program shall terminate execution.

8.1.3 Method showMessage

ShowMessage is a public method, which takes message as parameter. Method shows message in pop-up-window.

8.2 GraphicsWindow

GraphicsWindow class builds program's windows in new thread. Constructor is a public method, which takes DataCollection and GraphicsEngine- classes as parameters.

8.2.1 Method update

Update is a public method, which takes no parameters. Method updates GraphicsEngine, InfoPane and OutputPane. Returns boolean value.

8.2.2 Method buildWindow

BuildWindow is a public method, which takes no parameters. Method builds the main window and includes OutputPane, InfoPane and menus in it.

8.2.3 Method actionPerformed

ActionPerformed is a public method, which takes events as parameter. Method handles ActionEvents.

8.3 GraphicsEngine

GraphicsEngine is responsible for rendering graphics viewed on screen. GraphicsEngine class gets all information about program's state from DataColletion -class and it renders weather, planes and routes in one offScreen -image. GraphicsEngine uses Java 2D and buffered offScreen-images. Next frame will be rendered when buildNextFrame() -method is called. We render the main background image just when weather changes. Background image contains weather map, airport and arrival tree. When weather stays the same, we just render airplanes and airplane paths on the top of old background. At the beginning we show the Chopper -logo. GraphicsEngine also takes care of loading public graphics from hard disk. This feature is rather used by window and JPanels.

8.3.1 Method GraphicsEngine

The constructor of the class, that initializes render parameters. Takes DataCollection class as a parameter.

8.3.2 Method buildNextFrame

BuildNextFrame is public method, which takes no parameters. Method uses Java 2D and renders buffered off-screen images. Returns boolean value.

8.3.3 Method getFrame

GetFrame is public method, which returns next buffered off-screen image.

8.3.4 Method getGraphics

GetGraphics is public method, which takes index as parameter. Method returns asked bufferedImage. This feature is rather used by JPanels.

8.3.5 Method getStatus

GetStatus is public method, which takes no parameters. Method returns current class status. Status is false until first offScreen image is ready.

8.3.6 Method setDebugMode

SetDebugMode is public method, which takes boolean value as parameter. Method purpose is to make it easier to test graphicsEngine.

8.4 outputPane

OutputPane uses Java 2D Graphics to display and print graphics on screen. Constructor gets the GraphicsEngine class as a parameter.

8.4.1 Method update

Update is public method, which takes no parameters. Method updates outputPane and calls the paint -method. Returns boolean value.

8.4.2 Method paint

Paint is a public method, which takes graphics -object as parameter. Method paints off-Screen images to JPanel.

8.5 infoPane

InfoPane -class extends JPanel and it shows numerical information about programs state. Constructor gets the GraphicsEngine -class as a parameter.

8.5.1 Method update

Update is public method, which takes no parameters. Method updates infoPane and calls the paint -method. Returns boolean value.

8.5.2 Method paint

Paint is public method, which takes graphics -object as parameter. Method paints information to JPanel.