Maintenance 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
0.1	8.12.2008	Document created.
1.0	12.12.2008	Final version.

Contents

1 Introduction

The purpose of this document is to help those who want to modify the program. It contains installing instructions, information about how this program has followed its requirements and design documentations, instructions to maintain code, bugs, enhancement suggestions etc.

2 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.

3 How to install the program

Click the link *The program* in the main menu to get the *chopper.zip* -file. Unzip *chopper.zip* to the directory you want to install the program to. After this the file Chopper.jar is the runnable program. Exact running instructions are in the user manual.

4 Unfulfilled requirements and parts of design

4.1 Requirements which were not implemented

Speed of the planes decelerates when approaching the airport.

F8 - Map can be zoomed

F10 - Wind has direction and speed

F11 - Weather data is generated randomly

F12 - User is able to give storm centers, their intensity and wind speed

F14 - Program stores every weather data picture from Testbed to hard drive

4.2 Other comments about requirements

A requirement was that planes should not fly too close to each others. In the program planes do slow down their speed when they are too close each others, but this is not sufficient condition to prevent plane crashes. This problem may be solved in future by control planes speed on the grounds of arriving time to merge point.

It is said in the requirements document (section 6, user requirements) that the purpose of the program is to count and model as safe as possible way to approaching planes to airport through changing weather conditions. However, the group accentuates that the program doesn't calculate the safest path but rather the optimal path with defined safety distance.

System requirement F2 claims that planes never fly over intense storm. In some situations, however, a plane may fly over storm. Basically there are two reasons: user has set too small value for StormSafetyDistance - parameter, or the storm suddenly appear from nowhere. The main point is that the current arrival tree never perch over the storm.

4.3 Unfulfilled parts of design

Everything that was designed has been implemented.

5 Code maintenance

A Javadoc documentation of the code is provided with the program. The meanings of every package, class, method and field are explained there.

The program has been coded following the the code conventions for the Java programming language by Sun microsystems: http://java.sun.com/docs/codeconv/html /CodeConvTOC.doc.html. Following these instructions is highly recommendable also during maintenance. A plugin for Eclipse called Checkstyle has been used to check that these instruction are followed. In addition to Checkstyle another Eclipse plugin, PMD, has been used to look for some potential problems in code.

For every class in the code there is a JUnit test case class which tests that this class works properly. These unit test classes can be found in the subpackages of the package called *test*. Moreover, every test package has a class that runs its tests (for example CalculationTests runs the tests of the package test.calculation). Classes AllTests and AllTests2 in the package *test* run all tests of the program. If changes to the code is made, it should be taken care of that unit tests pass after the changes. Possibly some new unit tests should also be added.

5.1 Bugs

Plane can in certain situations teleport backwards. This happens because Plane is not able to update it's TreeNode because of storm. If Plane gets inside storm it cannot create path of TreeNodes. Instead it is flying to TreeNode possibly very far away from it's current location without path. This can lead to aStar finding path from last place where TreeNode was updated and at same Plane also appears at this location.

Planes can sometimes be given a path that flies away from direction of airport. This can happen when closest TreeNode is found other side of storm and path that is found is u shaped going around the storm.

When Plane is out of storm no new path is given although it could be because there has not been need to calculate Tree again. This is caused by following: Plane is only given new path when Tree has changed.

Safety distance check of Planes doesn't work when there are plenty of Planes on close distance from airport. Reason for this is unknown.

6 Other things affecting maintenance

6.1 Enhancement suggestions

The arrival tree should be re-calculated regularly when a certain time passes by or otherwise prevent situations where the tree has useless curves even without storms nearby. Currently this could happen sometimes because the tree is calculated only when a storm comes over the tree.

The heuristics of the A* algorithm could be modified so that A* would always find a path (between two vertices of a graph) that looks as straight as possible. Currently the path can go very close to a storm before it makes a curve around it, even though it would look better if the path would change the direction as soon as possible so that it would not have to make a curve just before the storm. Currently the length of the path is, however, optimal. It is just a matter of how it looks on a screen.

Also, routes in a graph could be formed so that they wouldn't allways have to go to adjacent vertices but could make a straight line between two vertices that are not adjacent. This would reduce zigzags but doing this would require lots of changes to different classes in the code.

Planes are now following path of TreeNodes although Graph is made of Vertices. This creates a bug that might be avoided by maintaining path of Plane as Vertices instead of Treenodes.

Astar method can't currently search routes through storms which could be useful for certain situtations for Planes. For example to find shortest way a way from center of large storm.

Setting safety distances to paths of the arrival tree could have propably been done in an more efficient way. Now the safety distance can only be set to some five nodes away from the nodes without slowing down the program too much. The recursive algorithms without too much repeat were tried, but none did make the program significantly faster. There should anyway be a more efficient way to do this.

The paths of the arrival tree can be in a bit tricky angle, when thinking about planes flying, if there are many storms nearby. This could be eliminated from the program if wanted by setting the arc on where the path leaves to be not available during the calculation or by looking, if the path intersects the particular arc twice or more times.