Software Requirements Specification

Air Traffic Controller Simulator

Version: 1

Prepared by: Cam-air-os

Team Members:
Cosme Ochoa
Mycal Tucker

16.35 Real-Time Systems and Software

Approved By:

Signature:

1 Introduction

1.1 Purpose

Air traffic control is a service provided by ground-based air traffic controllers who direct the flight paths of airplanes in order to maintain a controlled airspace. The primary purpose of air traffic control is to prevent the collisions of airborne airplanes as they travel towards their destinations. Furthermore, air traffic control helps expedite the flow of air traffic and provide information to pilots about their suggested flight path.

To prevent the collision of airborne airplanes, air traffic control imposes rules on the flight paths in order to ensure the safety of all on board passengers. These traffic separation rules enforce that all airplanes shall maintain a minimum amount of empty space surrounding them at all times. To further insure safe flight paths, many airplanes provide their own collision avoidance systems, which provide additional safety by warning the pilots if another airplane gets too close.

Apart from these primary safety concerns, airplanes face day to day problems that alter the volume of air traffic. Several factors dictate the amount of traffic that can land at an airport during a particular time:

- Number of available runways
- Landing time for an airplane
- Max airplane capacity of particular airport
- Delays due to schedule changes
- Assigned Terminals
- Weather

In this project we will focus mainly in on the departure and arrival of airplanes as they travel to their destinations. Our airplanes shall abide by traffic separations rules that will help in collision avoidance. Moreover, airplanes arriving at their destination shall verify that runaway is available to commence its decent. Lastly, this simulator will keep track of the fuel levels of each airplane. This will help determine whether which airplane travel the proposed flight path to the desired destination with the amount of fuel available on the airplane. Lastly, an airplane may have to undergo a collision avoidance pattern due to the proximities of other airplanes or the holding pattern due to an unavailable runway. Keeping track of the fuel levels will allow to simulate whether an airplane can complete its flight due these disturbances.

1.2 System Overview

Our system simulates a 2D world in which airplanes periodically must fly routes between airports. Airports will have a maximum allowable number of airplanes that can be at the airport at any given time. Airports will keep track of the airplanes currently stationed at the airport as well of the time of the latest arrival. This will help avoid the collision of airplanes trying to arrive at the same airport at the same time within a small time window. In addition, when an airplane requests to land at an airport that is currently at full capacity, the airport will let the airplane know when if it can or cannot commence its landing sequence. Both of these cases will let the pilot know whether it should execute a holding pattern before it requests to land again if necessary.

Each airplane will be have a starting airport and a destination to which it wishes to arrive. Every airplane will be given a starting amount of fuel before departure which will deplete as it makes its journey. An airplane shall keep track of its current fuel level as it travel on its flight path. If at any point during its flight the airplane fuel level reaches 0, this will indicate a failed flight to the original destination and thus the airplane shall attempt an emergency crash landing.

Every airplane pilot will be depicted in our simulation as an airplane controller. The controller of the airplane shall keep track of the airplanes current pose, which will be updated by the controller as airplane travels along its determine flight path. The controller will help avoid collisions with with other nearby aircrafts as well as the execution of a holding pattern if the destination airport denies the request of the airplane to land.

The simulator for this project shall run the overall air traffic control simulation. It will keep track of the simulation time and updating all airplanes as necessary. It shall maintain a list of all airplanes and airports created in the simulation. Finally, the simulator shall update the current states of the aircrafts as they travel on their flight path by updating through a display server.

Baseline: A properly functioning system will plot airplanes travelling along their planned routes. Airports will keep track of the airplanes currently at the airport and deny a request to land by an airplane if the airport is at max capacity. In addition, if an airplane runs out of fuel at any point while travelling to an airport, it shall attempt an emergency crash landing. Users will not be able to change airplane routes once the simulation is actually running.

Final: Airports will force planes to enter a holding pattern while waiting a minimum time between landings. Planes will display their fuel levels as they travel and crash if they run out of fuel. Users will be able to add airplane routes while the simulation runs.

1.3 Pre-existing Groundwork

This project is built upon some of the code base developed earlier in the semester for 16.35. The code reuses DisplayClient, DisplayServer, Control, Simulator, VehicleController, and DeadLockAvoider to simulate multi-vehicle movement in a 2D environment. This system will not be running on a real-time operating system, so it will use slightly older versions of class code that use regular Java threads.

In this project, the possible presence of deadlock cannot be ignored when avoiding collisions. Deadlock might arise because every airplane will need to verify if any other airplane is to close in proximity. Therefore, every airplane will need to obtain in an orderly manner the pose of all other aircrafts and make sure that no other airplane is too close and whether an avoidance trajectory is necessary. Similar as during past assignments, the possibility of deadlock will be broken by eliminating the circular wait condition.

2 System Description

2.1 Classes

2.1.1 Airplane (shall be able to run in a separate thread)

An Airplane object shall be almost exactly like a GroundVehicle object. Its state shall contain information about pose, velocities, and fuel level. For the remainder of this specification, consider a sample Airplane object "*a.*"

2.1.1.1 Variables

pose: array of doubles that stores *a*'s x and y position and its heading
- x, y shall be within the 2D simulated world [0, 100]
- heading shall be within [0, 2*pi)

velocities: array of doubles that stores *a*'s linear speed and angular velocity
- linear speed shall be within [5, 10]
- angular velocity shall be within [-pi/4, pi/4]

fuelLevel: double that stores how much fuel *a* has left
- fuelLevel shall be within [0, infinity)

flying: boolean representing whether or not the airplane is flying

2.1.1.2 Methods

Airplane(double[] pose, double[] velocities, double initialFuel): the constructor shall create and return an Airplane object that has pose, velocity, and fuelLevel as specified by the arguments. All arguments shall be clamped within their legal bounds.

void setPose(double[] newPose): shall set pose to match newPose (after clamping to legal values)

void setVelocity(double[] newVelocities): shall set velocities to match newVelocities (after clamping to legal values)

double[] getPose(): shall return pose

double[] getVelocity(): shall return velocities

double getFuelLevel(): shall return fuelLevel void applyControl(Control c): shall set velocities to match *c* within legal bounds. If c is null, this method shall not change any of Airplane's variables.

void updateState(int msec): shall update pose, velocities, and fuelLevel to simulate time progressing for msec milliseconds. It shall clamp these updated values to within legal bounds.

void setFlying(boolean fly): shall set flying to match fly

boolean getFlying(): shall return flying

void run(): shall wait for time from sim to change and then call updateState()

2.1.2 AirplaneController

AirplaneController shall extend VehicleController. It shall keep track of an Airplane object and return Control objects for it when prompted. These Control objects shall cause the Airplane to fly from one Airport to another. AirplaneController shall also implement collision avoidance by causing Airplanes to fly avoid each other if they get too close together

2.1.2.1 Variables

plane: Airplane object that this AirplaneController shall generate Controls for
startAirport: Airport object that plane shall start from
endAirport: Airport object that plane shall land at
sim: Simulator object that shall run the entire simulation and keep track of time
otherAirplanes: list of all other Airplane objects currently flying

2.1.2.2 Methods

AirplaneController(Airplane plane, Airport startAirport, Airport endAirport, Simulator sim): constructor shall set plane, startAirport, endAirport, and sim as given in the arguments. otherAirplanes shall be created as an empty list.

Control getControl(): shall return a Control object that plane should obey to reach endAirport safely in as straight a line as possible. The Control object shall deviate from a straight line for four possible reasons:

Collision avoidance: if plane is within 10 pixels of any other airplanes, it shall fly slightly away from the nearest airplane while still making progress towards endAirport.

endAirport.requestLand(plane) returns false: if plane is within 20 pixels of endAirport but endAirport says that plane cannot land, getControl() shall return a Control to make plane follow a holding pattern around endAirport.

if plane has no fuel left, getControl() shall return null

if a.flying() is false, getControl() shall return null

void addAirplane(Airplane a): shall add *a* to otherAirplanes void setDestination(Airport): shall set endAiport to the argument

2.1.3 Airport
Airport objects shall represent an airport. It shall have a fixed capacity for Airplanes that can stay at the Airport at any given time. To ensure that this capacity shall never be exceeded, Airports shall implement a sort of two-phase commit when an Airplane tries to land.

2.2.3.1 Variables
name: a String representing the name of the airport
groundedAirplanes: a list of Airplanes currently stationed at this Airport
landingAirplanes: a list of Airplanes that have been approved to land at this Airport
lastLandingTime: Simulator time at which the most recently landed Airplane in airplanes landed at the airport
landingTimeWindow: integer representing the minimum time that must elapse between Airplanes landing at this Airport
sim: the Simulator running the simulation
capacity: an integer representing how many Airplanes can stay grounded at this Airport
location: an array of doubles storing the x and y location of the Airport
x shall be a double that shall fall within [0, 100]
y shall be a double that shall fall within [0, 100]
x and y shall never change after initialization (a.k.a. the Airport shall not move)

2.1.3.2 Methods
Airport(x, y, capacity, sim): shall construct an Airport object with variables set according to the arguments passed in. groundedAirplanes and landingAirplanes shall be set to empty lists. lastLandingTime shall be set to sim.getCurrentTime()

boolean spawnAirplane(Airplane a): shall add Airplane *a* to groundedAirplanes if adding *a* shall not cause getNumFreeSpots() to equal zero. In that case, this method shall return true. Otherwise (if getNumFreeSpots() returns zero), this method shall not change any of its variables and shall return false.

boolean requestLand(Airplane): this method shall first check if getNumFreeSpots() returns zero. It shall also check if sim’s current time is less than lastLandingTime + landingTimeWindow. If either of these conditions is true, this method shall return false. Otherwise, this method shall add Airplane to landingAirplanes and return true.

boolean commitLand(Airplane): shall check if Airplane is in landingAirplanes. If it is not, this method shall return false. Otherwise, this method shall remove Airplane from landingAirplanes and add it to groundedAirplanes. This method shall then return true.

void unrequestLand(Airplane): shall remove Airplane from landingAirplanes. (An Airplane may crash before actually managing to land. The Airport should therefore not “save a spot" for it by keeping it in landingAirplanes. As a result, Airports rely on Airplanes to call unrequestLand if they crash.)

void takeoff(Airplane): shall remove Airplane from groundedAirplanes (if Airplane is not in the list, this method shall not perform any action). In addition, this method shall call Airplane.setFlying(true)

int getNumFreeSpots(): shall return capacity minus (length of groundedAirplanes + length of landingAirplanes)

double[] getLocation(): shall return an array of doubles containing x and y

String getName(): shall return name

2.1.4 Simulator
Simulator shall run the overall simulation by keeping track of simulation time and updating Airplanes as necessary. It shall maintain a list of all Airplanes in the world. In the baseline deliverable, these Airplanes shall be hardcoded in. In the final deliverable, users shall be able to add Airplanes to the simulation via mouse clicks.

2.1.4.1 Variables
currentTime: integer representing how long the simulation has run in milliseconds
running: boolean representing whether the simulation is running or not
airplanes: list of Airplanes in the simulation
airplaneControllers: list of all the AirplaneControllers in the simulation
airports: list of Airports in the simulation

2.1.4.2 Methods
int getCurrentTime(): shall return currentTime by default or 0 if running is false

void run(): shall update the simulation time in 10 millisecond increments and alert any threads waiting on it that the time has changed. run() shall also update the destinationAirports of the AirplaneControllers as necessary (either specified by users or hadcoded in). run() shall never finish executing.

void main(argv []): shall accept arguments from the command line to set up a connection between a DisplayClient and a DisplayServer. In the baseline deliverable, main() shall be hardcoded to add some Airplanes and Airports to the Simulation. main() shall also start all the necessary threads

void addAirplane(Airplane): shall add Airplane to airplanes list boolean addAirport(Airport): shall add Airport to airports list and return true if Airport isn't within 20 pixels of all other existing Airports in airports. Otherwise this method shall not change anything and return false.

2.2 User Interface

Baseline:

Users shall only be able to observe the simulation running. Airports and Airplane flights shall have been hardcoded into the Simulator.

Final Deliverable:

Users shall be able to add Airplanes that fly between Airports they specify via textboxes. Airplanes shall change color during the simulation as they run lower on fuel.

3. Appendix I: Extras

3.1 Not Enough Fuel:

If an Airplane runs out of fuel during a flight, it shall crash and be removed from the simulation. An extra deliverable that would be very nice but very complicated would allow Airplanes to land at nearby Airports if they calculate that they will not be able to reach their destination given their current fuel level.

3.2 Runway Configuration (Takeoff and Landing):

In the real world, airports have multiple runways. Adding in further landing capabilities to Airports to allow many Airplanes to land at the same time would be nice. Furthermore, airports have runways at fixed angles. We could conceivably force Airplanes leaving Airports to match their pose to that of the runway they are leaving.

3.3 Schedulability Test:

It would be nice if users were warned when they added a flight that it was not schedulable. Given our limited knowledge of schedulability tests, though, we decided that we could not confidently implement one. In a world where planes have to wait for others and can land on multiple runways, we have violated both our independent tasks and our uniprocessor assumptions. Perhaps after completing the course we can revisit this problem.