Description

Project 2 is to write a simple discrete event simulation of aircraft landing at an airport. The assignment is organized into 3 phases, where each phase extends the simulation from the previous phase. The following sections provide background information.

Flight Phases

Aircraft flights are segmented into distinct phases. For this assignment, we will use a simplified set of flight phases that only deal with landing an aircraft.

> **Enroute** – we will consider aircraft flying to the airport that are not yet in a holding pattern to be *enroute*.
>
> **Holding** – aircraft that are circling awaiting clearance to land will be in the *holding* phase. All enroute aircraft will be required to enter the holding phase (perhaps briefly) before being cleared to land.
>
> **Landing** – once a runway becomes available, a holding aircraft can be cleared to land on a runway and enters the *landing* phase.
>
> **Standing** – once an aircraft has landed, it immediately enters the *standing* phase at its gate.

Aircraft

We will only consider the following models of aircraft with the following characteristics:

Table 1 - Aircraft Types

Type	Seats	Fuel Capacity (gallons)
A320	150	7,835
A330	253	36,750
A340	380	54,023
A380	555	81,890
B737	162	6,875
B747	450	60,125
B767	245	23,980
B777	365	47,890
RJ145	53	5,187
CRJ1000	100	2,897

The type, or model, of an aircraft determines the number of seats on the aircraft as well as its maximum fuel capacity in gallons.

Runways

Runways are named based on their compass alignment. For example, a runway aligned towards 90° is also aligned to 270° (90° + 180°) and would be named 09/27. We will assume two runways 09/27 and 13/31 at our airport.

Simulation Inputs

Inputs to the simulation will be provided in text files with the following information:

Table 2 – Simulation Inputs

Arrival Time	Flight Id	Aircraft Type	Passengers	Fuel on Arrival
32.54	AF276	A330	174	5540
86.51	CO943	A320	114	1221

The arrival time is the simulation time in minutes when the aircraft is scheduled to arrive at the airport. The flight id is the identification number of the flight. The aircraft type is one of the aircraft types given in the previous table. Passengers is the number of passengers on the aircraft, which will always be less than the number of seats for that aircraft type. The fuel on arrival is the number of gallons of fuel the aircraft will have when it arrives at the airport and enters a holding pattern.

The arrival time and the flight id will be unique for each flight in an input file. Entries on a single input line will be separated by an arbitrary but non-zero amount of white space. Input files will be named "arrivalsNinT.txt" where N is the number of arrivals in the file and T is number of minutes those arrivals are distributed over. Entries in an input file are not ordered by arrival time.

Phase 1 – Landing

At the conclusion of Phase 1, your simulation should be able to read a file of input events and initialize your simulation, transition the aircraft from the enroute phase to the holding phase, assign each holding aircraft to a runway, and land the aircraft.

Your solution to Phase 1 should be submitted in a file called **`project2Phase1.py`**. It should include a function **`simulation(filename, timeStep=0.01)`** that takes as arguments a string identifying the name of the input file and an optional timeStep that defaults to 0.01.

Your simulation function should first read the input file and initialize several data structures. Suggested data structures include:

> **`aircraftType`** – a dictionary that maps the flight id of each flight to the aircraft type.
> **`numberPassengers`** – a dictionary that maps the flight id of each flight to the number of passengers on that flight.

`arrivalTime` – a dictionary that maps the flight id of each flight to the arrival time of the aircraft (i.e., the arrival time in the input record).

`phaseStartTime` – a dictionary that maps the flight id of each flight to the simulation time the current flight phase began.

`runways` – a dictionary that maps a runway id (09/27 or 13/31) to the flight id of an aircraft currently assigned to that runway. If the runway is not in use, the value should be None.

`enroute` – a list of all flights currently enroute.

`holding` – a list of all flights currently holding.

`standing` – a list of all flights currently standing at their destination gate.

Your simulation should then initialize the simulation time `t` to 0.0 and enter a while loop that terminates when all aircraft are standing at their destination gate (i.e., all aircraft are in the `standing` list. The following changes should occur during each iteration of the while loop:

1. The flights in the `enroute` list should be checked to determine if their arrival time has become less than the simulation time. If so, the flight should be moved to the `holding` list, the `phaseStartTime` for that flight should be set to the current simulation time, and message "`<id> arriving`" should be printed. For example, if flight AF267 has arrived, "`AF267 arriving`" should be printed.

2. For each runway that is not in use, if any flights are holding, the flight that has been holding the longest should be assigned to the runway by updating the `runways` dictionary. The `phaseStartTime` for that flight should be set to the current simulation time, the flight should be removed from the `holding` list and a message "`<id> cleared to land on runway <r>`" printed. For example, if flight AF267 has been assigned to runway 09/27, a message "`AF267 cleared to land on runway 09/27`" should be printed.

3. For each runway that is in use, if 10 minutes have elapsed since the flight started landing, the runway should be freed by updating the `runways` dictionary with `None`, the flight id should be added to the `standing` list, the `phaseStartTime` for that flight should be set to the current simulation time, and a message "`<id> standing at gate`" should be printed.

4. The simulation time `t` should be incremented by the amount `timeStep`.

Rather than coding all these directly in the while loop, you should implement items 1-3 as functions that are called from the while loop.

Phase 2 – Fuel Optimization

Significant changes from original posting are in red. Additional clarifications are in green.

Phase 2 extends the simulation so that each flight consumes fuel while holding and landing. It also changes the policy on which flight should be selected next to land so that flights that are running low on fuel are always given priority. Finally, it allows an option to be specified so that when selecting which flight is to land next, priority is given to those flights that consume the most fuel.

Your solution to Phase 2 should be submitted in a file called **`project2Phase2.py`**. It should include a function **`simulation(filename, timeStep=0.01, fuelOptimization=False)`** that takes as arguments a string identifying the name of the input file, an optional timeStep that defaults to 0.01, and a flag fuelOptimization that defaults to False.

You will also need to add and initialize some new data structures. These include:

> **`arrivalFuel`** – a dictionary that maps the flight id of each flight to the fuel the aircraft carried when it arrived.
>
> **`remainingFuel`** – a dictionary that maps the flight id of each flight to the fuel currently carried on the aircraft.
>
> **fuelCapacity** – this dictionary is no longer needed.
>
> **`seats`** – a dictionary that maps the aircraft type into the number of seats on it from Table 1. You may hard code this dictionary into your program.

The following changes also need to be made inside the while loop of the simulation.

1. Just prior to incrementing the simulation time, the `fuelRemaining` of each flight in the `holding` list or assigned to a runway should be decreased. Fuel consumption is a complex function of several factors, so we will make the simplifying assumption that while holding or landing, fuel is consumed at the rate 0.03 gallons / seat / minute.[1] . For example, an A340 with 380 seats would consume $0.03 \times 380 = 11.4$ gallons fuel/minute.

2. If a flight runs out of fuel, your simulation should print a message

   "<flight id> out of fuel"

[1] This does not represent the actual fuel consumption of any aircraft – your mileage may vary.

and terminate the simulation. An easy way to do this is to raise an exception when you discover a flight is out of fuel. This can be done placing a statement

```
raise Exception(id + ' out of fuel')
```

at the point in your code at which you discover the condition, where id is the flight id of the aircraft out of fuel. Then place a try block around the body of your simulation function to catch the exception of the form

```
try :
    …
 except Exception as e:
     print e
```

3. The choice of which holding aircraft is selected when a runway becomes available should also be changed. If one or more flights have less than 100 minutes of fuel left, the flight with the least minutes of fuel should be selected to land next. In the unlikely event of a tie (i.e., two or more flights with the least minutes of fuel have exactly the same minutes of fuel), select the flight that has been holding the longest.

   If all flights have 100 or more minutes of fuel, the selection of the next flight to land should reflect the fuelOptimization option. If it is False, the original algorithm from Phase 1 that selected the aircraft that had been waiting the longest should be used. If it is True, the aircraft in the holding list that consumes the most fuel per minute should be selected. In the event of a tie (i.e., two ore more flights that consume fuel at the fastest rate consume fuel at exactly the same rate), select the flight that has been holding the longest.

Finally, at the end of the simulation, the following two quantities should be computed and printed:

1. The total amount of fuel consumed landing by all aircraft from the time they arrived to the time they landed. This should be printed as

   "<total fuel consumed> gallons of fuel consumed landing"

   where <total fuel consumed> is an integer.

2. The total passenger minutes spent landing (i.e., the time from arrival to landing × the number of passengers on the aircraft) for all aircraft. This should be printed as

   "<total passenger mintues> passenger minutes spent landing"

   where <total passenger minutes> is an integer.

Debugging Hint

In can be helpful in debugging your program to identify the exact lines in your output that differ from the expected output in the unit test. An easy way to do this is to copy the expected output from the unit test into one file and the output from your program into another file (both are produced when the unit test is executed) and compare them using a file difference utility. A free web based file difference utility can be found at http://www.diffchecker.com

Extra Credit (up to 10 points maximum)

Write a function that generates arrival events like those shown in Table 2. The function should take an integer time T and integer number N as parameter. It should generate N arrival events with arri times randomly distributed between 0 and T. The aircraft type should be randomly selected from among the types in Table 1. The number of passengers should be a random fraction of the number of seats on that type of aircraft. The arrival fuel should be a random number that will provide 100 to 150 minutes of fuel for that type of aircraft.

Output the arrival events in a file named "`arrivalsNinT.txt`". The arrival time and flight id of each event in the file should be unique.

Use your program to generate varying numbers of arrival events over an 8-hour period. Modify your simulation to output a message when a flight is chosen as the next flight to land because it is low on fuel. At approximately what number of arrivals in an 8-hour period do flights start running low on fuel before they can land due to contention for a runway?