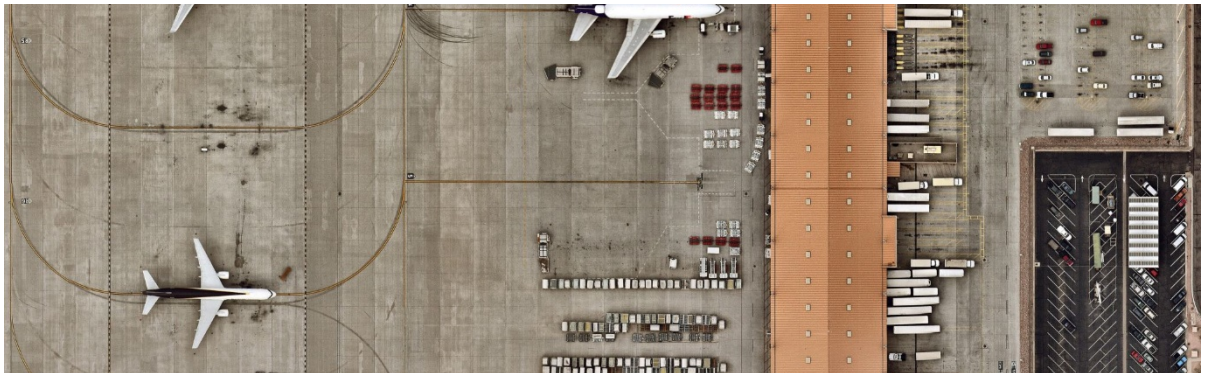


Airport Simulation

Use Case Description

The purpose of the Airport Simulation project is to create a realistic and interactive simulation of an airport's operations using the programming language C++. This simulation is designed to provide a virtual environment that closely replicates the activities and challenges encountered by airports in real world scenarios while maintaining a manageable scale for the user. It will serve as a valuable tool for educational purposes, research, and training within the aviation industry.

Class List

1. **Simulation:** Creates new vehicles and gives the user control on when and where planes/helicopters land and direct themselves to. Also saves and loads simulations.

2. **Vehicle:** Generalization of any given class that can be described as a vehicle

3. Plane: Inherits from vehicle class, and actively interacts with InAir, Runways, Taxiways, TerminalGates, and Hangers.

4. **Helicopter:** Inherits from vehicle class, works similarly to planes but doesn't require runways / taxiways and can head to their terminal type

5. **InAir:** Inherits from Plane and helicopter class to determine type of vehicle in the air

6. **TerminalGate:** Represents an airport terminal and can contain 1 plane that is using the terminal.

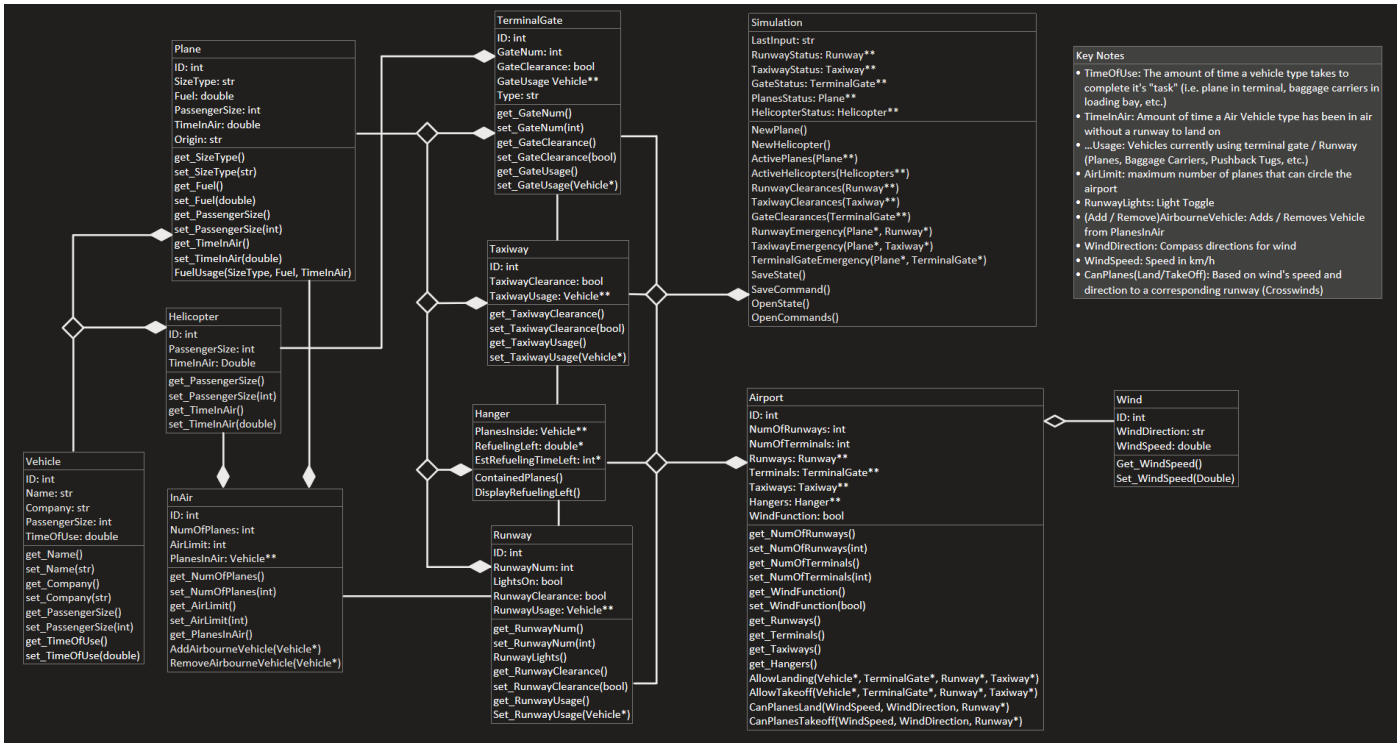
7. **Taxiway:** Connections between Runways and TerminalGates, can contain 1 plane per taxiway

8. **Runway:** Landing and take off areas for planes, with 1 plane being able to make either action per runway at any given time

9. **Hanger:** A place some planes temporarily go into for refuelling

10. **Airport:** Inherits from Taxiway and runway and is used to specify the features of the airport such as the number of runways, number of terminals, windspeed at the airport, and if whether he planes could take-off or land within the specified wind speed.

11. **Wind:** Refers to wind speed and direction, to determine if there are cross winds



Data and Function Members

- Simulation:

- Attributes:

- LastInput: str

- RunwayStatus: Runway**

- TaxiwayStatus: Taxiway**

- GateStatus: TerminalGate**

-PlanesStatus: Plane**

- HelicopterStatus: Helicopter**

- Functions:

- NewPlane(): Creates new planes to fly in from a randomly selected origin.

- NewHelicopter(): Creates new helicopter to fly in from a randomly selected origin

- DeleteVehicle(Vehicle**): Removes plane / helicopter once it has left the simulation area

- ActivePlanes(Plane**): Lists all planes in simulation

- ActiveHelicopters(Helicopters**): Lists all helicopters in simulation

- RunwayClearances(Runway**): Lists all runways in airport

- TaxiwayClearances(Taxiway**): Lists all taxiways in airport

- GateClearances(TerminalGate**): Lists all terminal gates in airport

- RunwayEmergency(Plane*, Runway*): Creates an emergency that requires certain user commands to avoid further problems. As this is the runway emergency, all landing and take off planes would need to be canceled / rerouted.

- TaxiwayEmergency(Plane*, Taxiway*): Creates an emergency that requires certain user commands to avoid further problems. As this is the taxiway emergency, all planes need to be routed around the taxiway.

- TerminalGateEmergency(Plane*, TerminalGate*): Creates an emergency that requires certain user commands to avoid further problems. As this is the terminal gate emergency any planes / helicopters need to be routed elsewhere if they were going to said terminal gate.

- SaveState(): Creates a save file of current time, planes, helicopters and their locations and states.

- OpenState(): Opens a previous save file of a simulation.

- Vehicle:

- Attributes:

  - ID: int

  - Name: str

  - Company: str

  - PassengerSize: int

  - TimeOfUse: double

- Functions:

  - get_name(): returns vehicle name

  - set_name(str): forced way for setting a vehicle's name

  - get_company(): returns vehicle company

- set_company(str): forced way for setting a vehicle's company

- get_passengerSize(): returns vehicle passenger size

- set_passengerSize(int): forced way for setting a vehicle's passenger size

- get_timeOfUse(): returns a value of how long the vehicle takes to do a stationary process

- set_timeOfUse(double): forced way for setting a vehicle's stationary process time

- Plane:

- Attributes:

  - ID:int

  - Size type: str

  - Fuel: double

  - PassengerSize:int

  - TimeinAir: double

  - Origin:str

- Functions:

  - get_sizeType(): returns a string of planes size (small, medium, large)

  - set_sizeType(str): forced way for setting a plane's size

  - getFuel(): returns remaining fuel (percentage based on time implemented, starting fuel and size)

  - setFuel(double): forced way for setting a plane's current fuel

  - get_PassengerSize(): returns plane's passenger size

  - set_PassengerSize(int): forced way for setting a plane's passenger size

  - get_TimeInAir(): set time from initialisation until the plane needs to make a forced landing (based on fuel, initialisation time and size)

  - set_TimeInAir(double): forced way for setting a plane's remaining time in air

- Helicopter:

- Attributes:

  - ID: int

- PassengerSize: int

- TimeInAir: double

- Functions:

- getFuel(): returns remaining fuel (percentage based on time implemented, starting fuel and size)

- setFuel(double): forced way for setting a helicopter's current fuel

- get_PassengerSize(): returns helicopter's passenger size

- set_PassengerSize(int): forced way for setting a helicopter's passenger size

- get_TimeInAir(): set time from initialisation until the helicopter needs to make a forced landing (based on fuel, initialisation time and size)

- set_TimeInAir(double): forced way for setting a helicopter's remaining time in air

- InAir:

- Attributes:

- ID: int

- NumofPlanes: int

- AirLimit: int

- PlanesInAir: Vehicle**

- Functions:

- get_NumofPlanes(): returns the number of planes and helicopters waiting to land

- get_AirLimit(): returns the number of planes and helicopters that can wait in the air

- set_AirLimit(int): forced way for setting the limit of planes and helicopters waiting for clearance

- get_PlanesInAir(): returns list of all planes / helicopters in air

- AddAirborneVehicle(Vehicle*): adds new vehicle into the airspace

- RemoveAirborneVehicle(Vehicle*): removes a vehicle from airspace

- TerminalGate:

- Attributes

- ID: int

- GateNum: int
- Gateckearance: bool
- GateUsage Vehicle**
- Type: str

- Functions:
  - get_GateNum(): Returns a number that represents the gate
  - set_GateNum(int): Sets the gate's number
  - get_GateClearance(): returns true or false if the gate is not in use
  - set_GateClearance(bool): overrides if the gate is in use or not
  - get_GateUsage(): returns current plane / helicopter using gate
  - AddGateVehicle(Vehicle*): adds new vehicle to the gate
  - RemoveGateVehicle(Vehicle*): removes a vehicle from the gate

- Taxiway:

- Attributes:
  - ID: int
  - GateNum: int
  - TaxiwayClearance: bool
  - TaxiwayUsage: Vehicle**
- Functions:
  - get_TaxiwayNum(): Returns a number that represents the gate
  - set_TaxiwayNum(int): Sets the gate's number
  - get_TaxiwayClearance(): returns true or false if the gate is not in use
  - set_TaxiwayClearance(bool): overrides if the gate is in use or not
  - get_TaxiwayUsage(): returns current plane / helicopter using gate
  - AddTaxiwayVehicle(Vehicle*): adds new vehicle to the gate
  - RemoveTaxiwayVehicle(Vehicle*): removes a vehicle from the gate

- Runway:

- Attributes:
  - ID: int

- RunwayNum: int
- LightsOn: bool
- RunwayClearance: bool
- RunwayUsage: Vehicle**

- Functions:
  - get_RunwayNum(): Returns a number that represents the gate
  - set_RunwayNum(int): Sets the gate's number
  - get_RunwayClearance(): returns true or false if the gate is not in use
  - set_RunwayClearance(bool): overrides if the gate is in use or not
  - get_RunwayUsage(): returns current plane / helicopter using gate
  - AddRunwayVehicle(Vehicle*): adds new vehicle to the gate
  - RemoveRunwayVehicle(Vehicle*): removes a vehicle from the gate

- Airport:

- Attributes:
  - ID: int
  - NumOfRunways: int
  - NumOfTerminals: int
  - NumOfTaxiways: int
  - Runways: Runway**
  - Terminals: TerminalGate**
  - Taxiways: Taxiway**
  - WindFunction: bool
  - Hangers: Hanger
- Functions:
  - get_NumOfRunways(): Returns number of total runways
  - set_NumOfRunways(int): sets number of total runways
  - get_NumOfTerminals(): Returns number of total terminal gates
  - set_NumOfTerminals(int): sets number of total terminal gates

- get_NumOfTaxiways(): Returns number of total taxiways

- set_NumOfTaxiways(int): sets number of total taxiways

- get_WindFunction(): Returns true or false if wind is enabled for the simulation

- set_WindFunction(bool): sets the wind function as on or off

- get_Runways(): Returns all runways

- get_Terminals(): Returns all terminals

- get_Taxiways(): Returns all taxiways

- get_Hangers(): Returns all Hangers

- Wind:

- Attributes:

  - ID: int

  - WindDirection: str

  - WindSpeed: double

- Functions:

  - get_WindDirection(): Returns wind direction as a compass direction

  - set_WindDirection(str): Sets which direction the wind is travelling in

  - get_WindSpeed(): Returns the wind speed in km/h

  - set_WindSpeed(double): Sets the wind's speed

- Hanger:

- Attributes:

  - PlanesInside: Vehicle**

  - RefuelingLeft: double*

  - EstRefuelingTimeLeft: int*

- Functions:

  - ContainedPlanes(): returns all planes in hanger

  - DisplayRefuelingLeft(): returns list of each plane's remaining fuel time left

  - AddRunwayHanger(Vehicle*): adds new vehicle to the hanger

  - RemoveRunwayHanger(Vehicle*): removes a vehicle from the hanger

Relationships between Classes

- Vehicle is the parent to both Plane and Helicopter.

- Taxiways connect between Runways, TerminalGates, and Hanger.

- TerminalGates require Planes / Helicopters to stay stationary in them to offload and onload cargo.

- Planes land and take off from runways

- Wind determines which runways are unusable due to cross winds (based on wind speed and direction)

- Hanger is used by Planes for refuelling

- Airport contains all Runways, Taxiways, and Terminal Gates

- InAir contains all the Planes and Helicopters that are yet to land but are waiting to be allocated a landing area (i.e., allocated a runway).

- Simulation runs most of the knowledge behind where planes and helicopters currently are and where they will be moving to and from

Project Task List and Timeline

1. Design the class structures and relationships (1 day) (deadline 4th Oct)
2. Implement Vehicle, Plane, Helicopter classes (1-2 days) (deadline: 6th Oct)
3. Implement TerminalGate, Taxiway, Runway classes (1-2 days (deadline: 8th Oct)
4. Implement InAir, Airport, Wind classes (2-3 days) (deadline: 11th Oct)
5. Implement "Simulation" class (2-3 days) (deadline 16th Oct)
6. Debug (1-4 days) (deadline: 17th Oct)
7. Write User Documentation

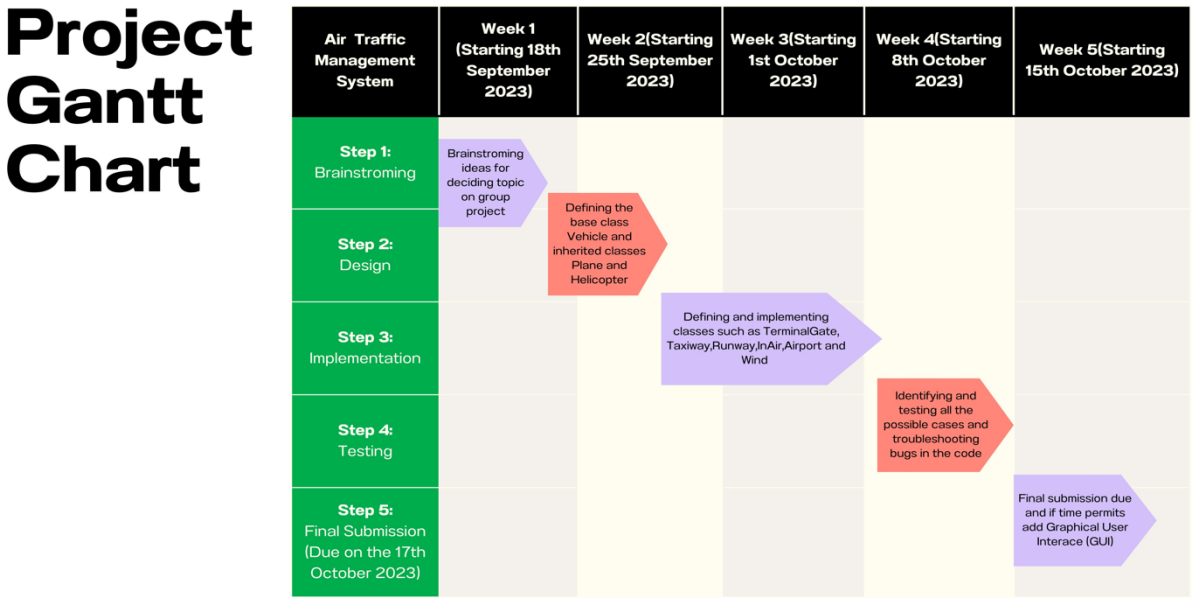


Figure 1 *Gantt Chart showing the project timeline*

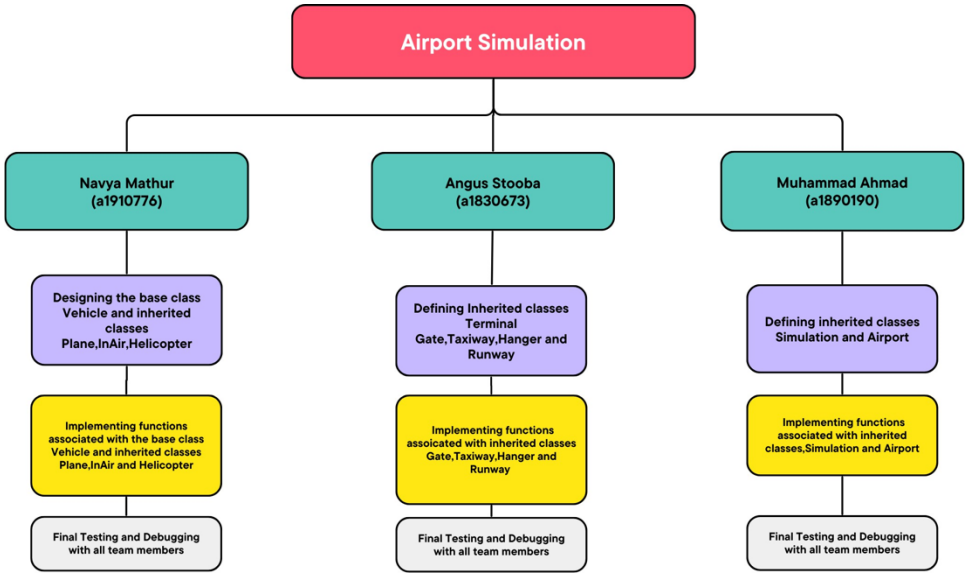


Figure 2 *Workload breakdown structure*

User Interaction Description

User interacts with planes by determining which runways, taxiways and terminals will be used by the plane before the plane lands.

User Interaction Description

In the Airport Simulation program, user interaction plays a crucial role in managing aircraft movements and airport operations. Users have the responsibility of determining runway assignments, taxiway routes, and terminal gates for incoming planes before they land. This interactive process allows users to control and optimize airport activities effectively. Here's a detailed description of how users will interact with the program:

User Input Methods:

(Possible) Command Line Interface (CLI): Users interact with the program through a command line interface where they can input commands and make decisions.

(Future) Graphical User Interface (GUI): For a more user-friendly experience, the program may include a graphical interface with buttons, menus, and visual representations of the airport layout. Users can make selections using the mouse or keyboard.

User Actions and Interactions:

Plane Assignment: When a new plane approaches the airport, the user receives a notification. They must decide which runway the plane should use for landing. This decision is based on factors like the plane's size, current runway availability, and weather conditions.

Taxiway Routing: After landing, the user determines the taxiway route the plane should follow to reach its designated terminal gate. This decision considers the airport layout and ensures efficient movement of planes on the ground.

Terminal Assignment: Once the plane reaches the terminal area, the user assigns a gate for the plane to park. This step involves managing available gates, considering the plane's airline or destination, and optimizing gate allocation.

Emergency Handling: In the event of emergencies, such as a plane requesting an emergency landing, the user must make critical decisions quickly. This may involve clearing a runway or coordinating with emergency services.

Resource Management: Users manage airport resources, such as fuel, ground crew, and baggage handling, by allocating them to planes as needed. Efficient resource allocation contributes to smooth operations.

User Interfaces:

Notification Panel: The program provides a notification panel displaying incoming plane details, including type, origin, and request for landing. Users receive alerts when decisions are required.

Airport Map: In GUI mode, an airport map is displayed, showing the runway layout, taxiways, terminals, and parked planes. Users can drag and drop planes to assign gates visually.

Unit Testing and Debugging Plan

A plan testing each combination of different inputs and testing if the class performs accurately based on the different combinations.

The Airport Simulation requires thorough unit testing to ensure the accuracy and reliability of each class's functionality. The following plan outlines a systematic approach to testing each class with different input combinations and provides a strategy for debugging in case of issues:

Unit Testing and Debugging Plan

The Airport Simulation should go undergo rigorous unit testing to ensure its accuracy and reliability. Unit Testing will be used to ensure that the various classes and functions, perform as they were intended. This plan outlines the testing approach, specific methods, success criteria, and debugging strategies:

Testing Methods:

1. Runway Assignment Testing:

- **Inputs:** Various types and sizes of planes, weather conditions, and runway availability.
- **Testing Criteria:** Verify that planes are correctly assigned to runways based on their characteristics and operational constraints.
- **Success Criteria:** All planes are assigned to appropriate runways, considering safety and operational efficiency.

2. Taxiway Routing Testing:

- **Inputs:** Planes with different destinations, airport layout, and traffic congestion.
- **Testing Criteria:** Evaluate the program's ability to calculate optimal taxiway routes for planes to reach their designated gates.
- **Success Criteria:** Planes follow efficient routes without collisions or delays.

3. Terminal Assignment Testing:

- **Inputs:** Planes of different airlines, gate availability, and airport terminals.
- **Testing Criteria:** Ensure that planes are allocated to appropriate gates, considering airline preferences and gate capacity.
- **Success Criteria:** All planes are assigned to gates without exceeding gate capacity, optimizing terminal usage.

4. Emergency Handling Testing:

- **Inputs:** Simulate emergency scenarios like plane or engine malfunctions, rerouting, medical emergencies, and unscheduled landings.
- **Testing Criteria:** Validate the program's ability to prioritize and respond to emergency situations promptly.
- **Success Criteria:** Emergencies are handled effectively, ensuring the safety of all aircraft and passengers.

5. (Future) Resource Management Testing:

- **Inputs:** Vary resource availability (fuel, ground crew, baggage handling) and demand.
- **Testing Criteria:** Verify that resources are allocated efficiently to meet the operational requirements of planes.
- **Success Criteria:** Resources are allocated optimally, preventing resource shortages or over-allocations.

By sticking to this unit testing and debugging plan, the Airport Simulation program will be thoroughly evaluated and refined to ensure its accuracy, stability, and robustness under various scenarios.

Debugging Strategy:

- Error Logging
    - Implementing a logging system to save timestamps and program states, to easily revert to when problems or errors occur.
- Code Inspection
    - Carefully examine and review the code to identify any logical, syntax or runtime errors.
- Input Validation
    - Revaluate input data and validation procedures to ensure proper handling of inputs.
- Unit Testing
    - Debug the specific unit or class responsible for issues by executing forced unit tests
- Isolation
    - Isolate the problem to specific class to minimize the debugging scope.
- Peer Review
    - Regularly get code reviewed by peers to gain fresh insight and perspectives.
- Documentation Update
    - Keeping comprehensive records of problems with the actions taken during debugging for future reference