



**DEPARTMENT OF COMPUTER &  
SOFTWARE ENGINEERING  
COLLEGE OF E&ME, NUST, RAWALPINDI**



**Project Report: Final Project Report**

**SUBMITTED TO:**

Lec. Anum Abdul Salam

**SUBMITTED BY:**

**Student Names: Noor-al-ain Kharal, Rijaa Khalid, Ammar bin Shahid**

**Reg # 414421, 403705, 420220**

**DE- 44 Dept C&SE Syn B**

**Submission Date: 17/01/2024**

<b>Introduction:</b>	<b>3</b>
<b>List of Functionalities:</b>	<b>3</b>
<b>Functionalities in Depth:</b>	<b>4</b>
• Input/Output	4
• Clock:	4
• Add Departure:	5
• Add Arrival:	7
• GROUND_NETWORK Tasks:	8
• AIRPLANE tasks	8
<b>Project Execution Flow:</b>	<b>10</b>
<b>UML Class Diagram</b>	<b>12</b>
Airplane	12
AirplaneList	12
ControlLogic	13
Details	14
Flight	14
FlightSchedule	15
Global_Clock	15
Task	15
TaskQueue	16
Time	16
Traffic_Network	16
<b>Results and output</b>	<b>19</b>
For Plane Departure:	20
<b>Mathematical Modeling</b>	<b>27</b>
<b>Limitations and Conclusion:</b>	<b>29</b>

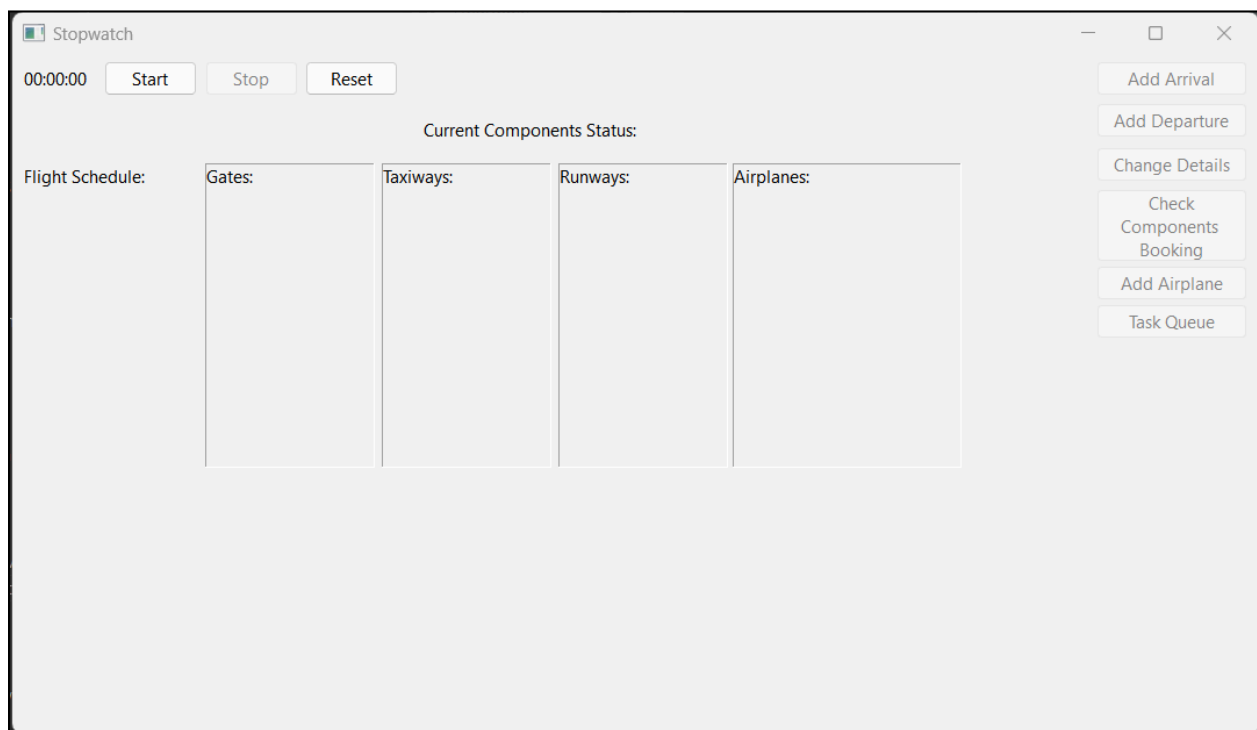
## Introduction:

Our objective was to create an air traffic control system simulation, where there is control over the ground network objects and airplanes in order to either send or receive a plane.

## List of Functionalities:

Our project has the following functionalities

- Clock (stop, start, restart)
- Plane Arrival
- Plane Departure
- See real time updates of ground network and airplane list
- See bookings of ground network and airplane list
- Add airplanes
- See flight schedule
- Change details of plane arrival/departure schedule



## Functionalities in Depth:

- Input/Output

- We are displaying the data through a graphical user interface. The first main page of the interface displays a variety of options to the user. However they are disabled until the user presses the start button.

The main page has the following input output procedures:

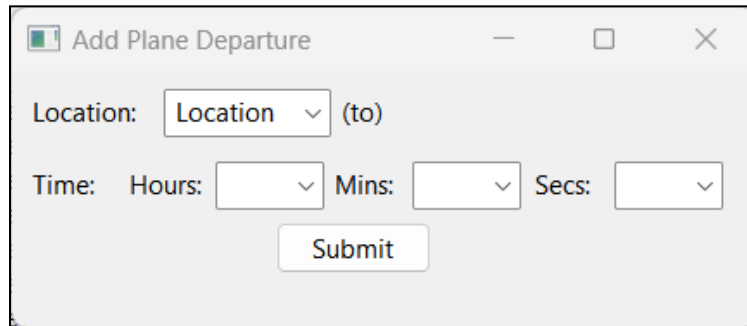
- Start Clock: starts simulation
- Stop Clock: pauses simulation
- Components Display: shows current real time status of all components (i.e Airplanes, Gates, Taxiways, and Runways)
- Task Queue: displays a new form which lists the tasks currently in task queue.
- Flight Schedule: displays the list of flights incoming or outgoing
- Add Airplane: allows user to add a new airplane
- Check Components Booking: allows user to see the future bookings of the components
- Add Arrival: opens a new form that allows user to choose location from where flight is coming from, as well as the time of arrival
- Add Departure: opens a new form that allows user to choose location to where the flight should go to, as well as the time of departure
- Change Details: allows user to add

- Clock:

- Start clock: When the button is clicked, a thread is created of the class Global\_Clock, and the thread is started. This allows for the whole operation to commence as the thread not only starts updating the clock, it also creates another thread of the taskQueue class.
- Stop clock: This pauses the clock, thus pausing the simulation as a whole as all the operations are time dependent

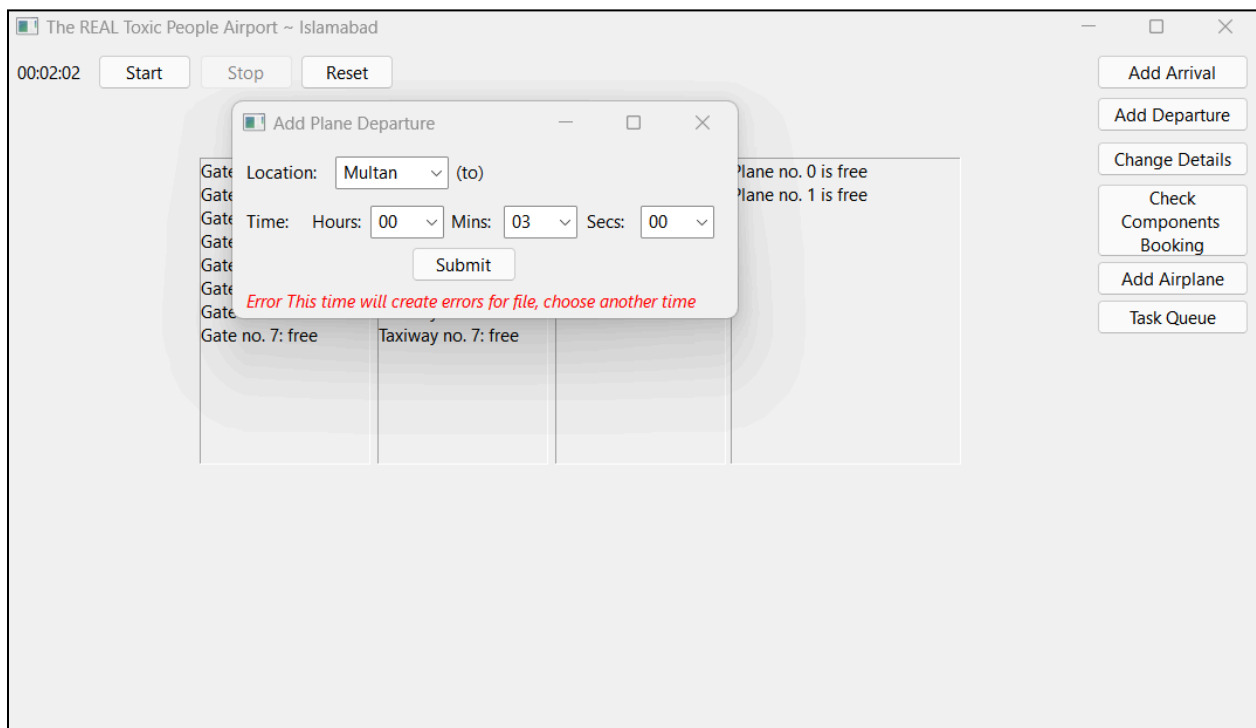
- Add Departure:

- Opens a form to allow the user to enter details for an outgoing flight. The inputs



are all in combos to not allow user to enter invalid data

- The user can choose a location of their choice as well as set time
- Since the operation for departure does not include only departure itself, but boarding, taxiing, runway checks etc, there is a check on time. The time must be more than the current clock time + boarding begins time. If not then an error message is displayed



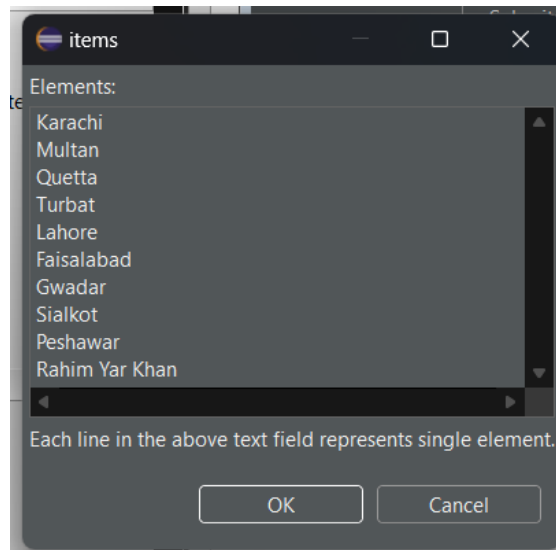
- If the data is valid, a task of the category CONTROL\_TASK is created for the departure flow, which is immediately executed

- The goal of this task is to generate the tasks required to allow a plane to depart
- The tasks created are:
  - At departure time - mins boarding
    - AIRPLANE\_TASK: move airplane to a free gate
    - GATE\_TASK: declare gate as busy
  - At departure time - mins boarding ended
    - AIRPLANE\_TASK: move airplane to free taxiway
    - TAXIWAY\_TASK: declare that taxiway as busy
    - GATE\_TASK: declare gate as free
  - At departure time - mins runway
    - AIRPLANE\_TASK: move airplane to free runway
    - RUNWAY\_TASK: declare runway as busy
    - TAXIWAY\_TASK: declare taxiway as free
  - At departure time
    - AIRPLANE\_TASK: lift off plane
    - RUNWAY\_TASK: declare runway as free
  - At departure time + deletion time
    - AIRPLANE\_TASK: delete airplane
- These tasks are created, with the time they should be executed on, and are added to the task queue. (where they are immediately sorted according to time)



- Add Arrival:
  - Similarly to add departure, a new form is opened, and it allows the user to enter the inputs through combo (dropdown). The locations are the same, as well as the time input. User submits data and there is check (similar to departure)
  - If the data is valid, a task of the category CONTROL\_TASK is created for the departure flow, which is immediately executed, and since this is plane arrival, we create a new plane instead of choosing from existing ones
  - The goal of this task is to generate the tasks required to allow a plane to arrive
  - The tasks created are:
    - At arrival time:
      - AIRPLANE\_TASK: Airplane is moved to runway
      - RUNWAY\_TASK: Runway is declared busy
    - At Arrival time + time to taxiway:
      - AIRPLANE\_TASK: Airplane is moved to taxiway
      - TAXIWAY\_TASK: declare that taxiway as busy
      - RUNWAY\_TASK: declare runway as free
    - At arrival time + time start unboarding
      - AIRPLANE\_TASK: move airplane to free gate

- TAXIWAY\_TASK: declare that taxiway as free
- GATE\_TASK: declare gate as busy
- At arrival time + time stop boarding
  - AIRPLANE\_TASK: declare plane as free
  - GATE\_TASK: declare gate as free
- Plane can now be reused for departure



- GROUND\_NETWORK Tasks:
    - Ground network component booked: the specified ground network component is booked, and its time of booking is also noted
    - Ground network component unbooked: after the specified ground network component is user, it is declared free.
  - AIRPLANE tasks
    - Airplane has the following states
      0. creation (meaning ke its doing nothing rn, the default orientation
      1. plane is landing
      2. plane is lifting off
      3. Touchdown/Takeoff (at runway)
      4. Taxiing plane is coming in or out of taxiway
      5. Loading (at gates to load/unload)
- So whenever it moves, the state is changed.



- Airplane can be deleted
- Airplane can be booked

## **Project Execution Flow:**

When user starts the simulation, the following things happen:

1. Global clock thread gets started, and `isRunning` is true
2. TaskQueue thread also gets started

These two threads are working simultaneously. If TaskQueue has any tasks, it will sort it according to execution time in ascending order, and check if the topmost task and globalClock's time match up. If they do, the following happens:

1. The taskqueue checks how many more tasks in its queue has the same execution time
2. According to the number of tasks, controlLogic objects are made, and tasks are dispatched to the control logic objects to be executed
3. The control logic objects share the objects of `airplaneList`, `flightSchedule`, and `groundNetwork`. (they are static), thus a common value can be maintained amongst all instances. So any changes made during object execution is saved
4. After task execution, the control logic objects are destroyed(automatically), and the tasks executed are removed by taskqueue, allowing next task to take its place

This loop continues on.

The tasks are created in two instances:

1. Add arrival
2. Add departure

For these, the following is the execution flow:

1. Open add arrival/departure form
2. Submit location plus arrival/departure times
3. Control logic task generated, and controlLogic object created to immediately execute task. This would generate tasks required for plane to departure/arrive (list of tasks generated explained in above sections)
4. Tasks are added to taskqueue, and the execution flow returns to normal loop

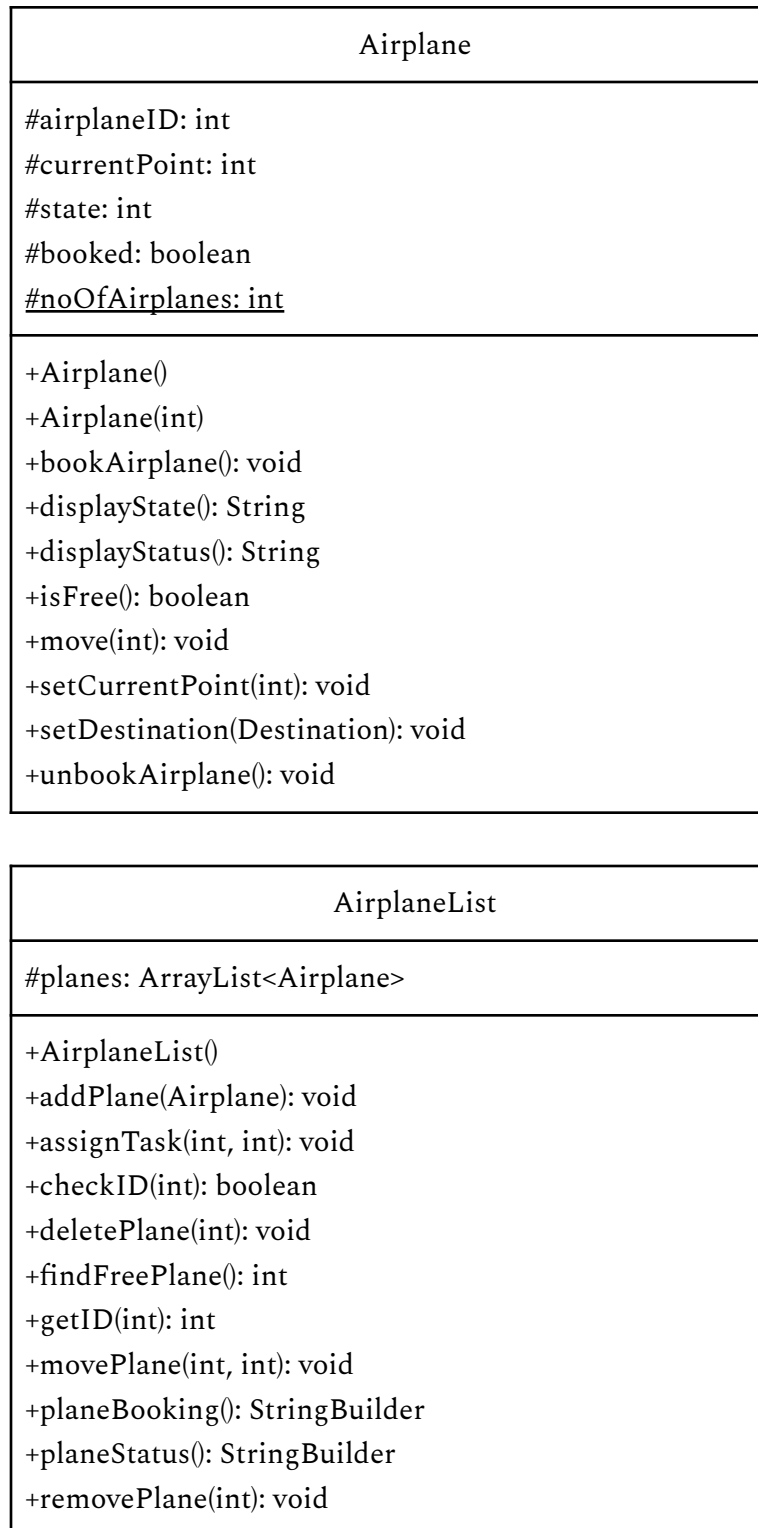
Beyond this, there are displays to show status of components to user, and user can add airplanes to the `airplaneList`.

Both threads are stopped at stop button, and simulation ends when the user exit the window



## UML Class Diagram

Class List:



```
+setCurrentPoint(int, int): void  
+setDestination(int, Destination): void
```

### ControlLogic

```
#task: Task  
#protected planesList: AirplanesList  
#protected groundNetwork: Traffic Network  
#protected flights: FlightSchedule  
#protected flightDetails: Details
```

```
+ControlLogic()
```

```
<<task related methods>>
```

```
+setTask(Task): void
```

```
+runTask(): void
```

```
+deleteTask():void
```

```
<<control logic tasks>>
```

```
+startArrival():void
```

```
+addArrival(int): void
```

```
+startDeparture():void
```

```
+addDeparture(int): void
```

```
+holdRunway():void
```

```
<<plane tasks>>
```

```
+addPlane(Airplane): void
```

```
+planeMoveGate(): void
```

```
+planeMoveTaxiway(): void
```

```
+planeMoveRunway(): void
```

```
+planeLiftoff(): void
```

```
+planeLand(): void
```

```
+planeFree(): void
```

```
+planeDelete(): void
```

<<gate tasks>> +gateFree():void +gateBook():void
<<taxiway tasks>> +taxiwayFree(): void +taxiwayBook(): void
<<runway tasks>> +runwayFree(): void +runwayBook(): void

Details
#minsBoarding: int #minsTaxiway: int #minsRunway: int #minsTakeoff: int
+Details() +changeDetails(int,int,int,int):void

Flight
#departure: boolean #location: String #planeTime: Time #planeID: int
+Flight(boolean, int, Time, int) +flightDetails(): StringBuilder

FlightSchedule
#flights:ArrayList<Flight> flights = new ArrayList<Flight>
+FlightSchedule() +addFlight(Flight):void +sortFlight():void +deleteFlight():void +flightDisplay():StringBuilder

Global_Clock
#globalTime:Time #isRunning: boolean
+Global_Clock(int,int,int) +setTime(int,int,int):void +startClock2():void +stopClock():void +advanceTime():void +getCurrentTime():String +getHour():int +getMinute():int +getSeconds():int +run():void

Task
#primaryLabel:int #secondaryLabel:int[] #supplementalData:int #taskExecution: Time #holdTime: Time

+Task(int,int,int,Time) +Task(int,int,int,int,Time) +Task(int,int.int,Time,Time) +display(): void +taskTranslation():StringBuilder
--

TaskQueue
#tasksList: ArrayList<Task>
+TaskQueue() +addTask(Task):void +sortTask():void +displayTasks():void +checkTaskRun(int):boolean +runTask():void +taskDisplayGUI():StringBuilder +run(): void

Time
#hour: int #min: int #sec: int
+Time(int,int,int) +addTime(int):Time +compareTime(Time):boolean +getTime():String +greaterThan(Time):boolean +lessThan(Time):boolean +subtractTime(int):Time

Traffic_Network
-----------------



```
#runways: boolean[]  
#taxiways: boolean[]  
#gates: boolean[]
```

```
#futureRunways: boolean[]  
#futureTaxiways: boolean[]  
#futureGates: boolean[]
```

```
#futureRunwaysTime: Time[]  
#futureTaxiwaysTime: Time[]  
#futureGatesTime: Time[]
```

```
+Traffic_Network()
```

```
+findFreeRunway():int  
+findFreeTaxiway(int):int  
+findFreeGate():int
```

```
+bookRunway(Time, int): void  
+bookTaxiway(Time, int): void  
+bookGate(Time, int): void
```

```
+unbookRunway(int): void  
+unbookTaxiway(int): void  
+unbookGate(int): void
```

```
+changeRunwayState(int,boolean):void  
+changeTaxiwayState(int,boolean):void  
+changeGateState(int,boolean):void
```

```
+currentStatusGate():StringBuilder  
+currentStatusTaxiway():StringBuilder  
+currentStatusRunway():StringBuilder
```

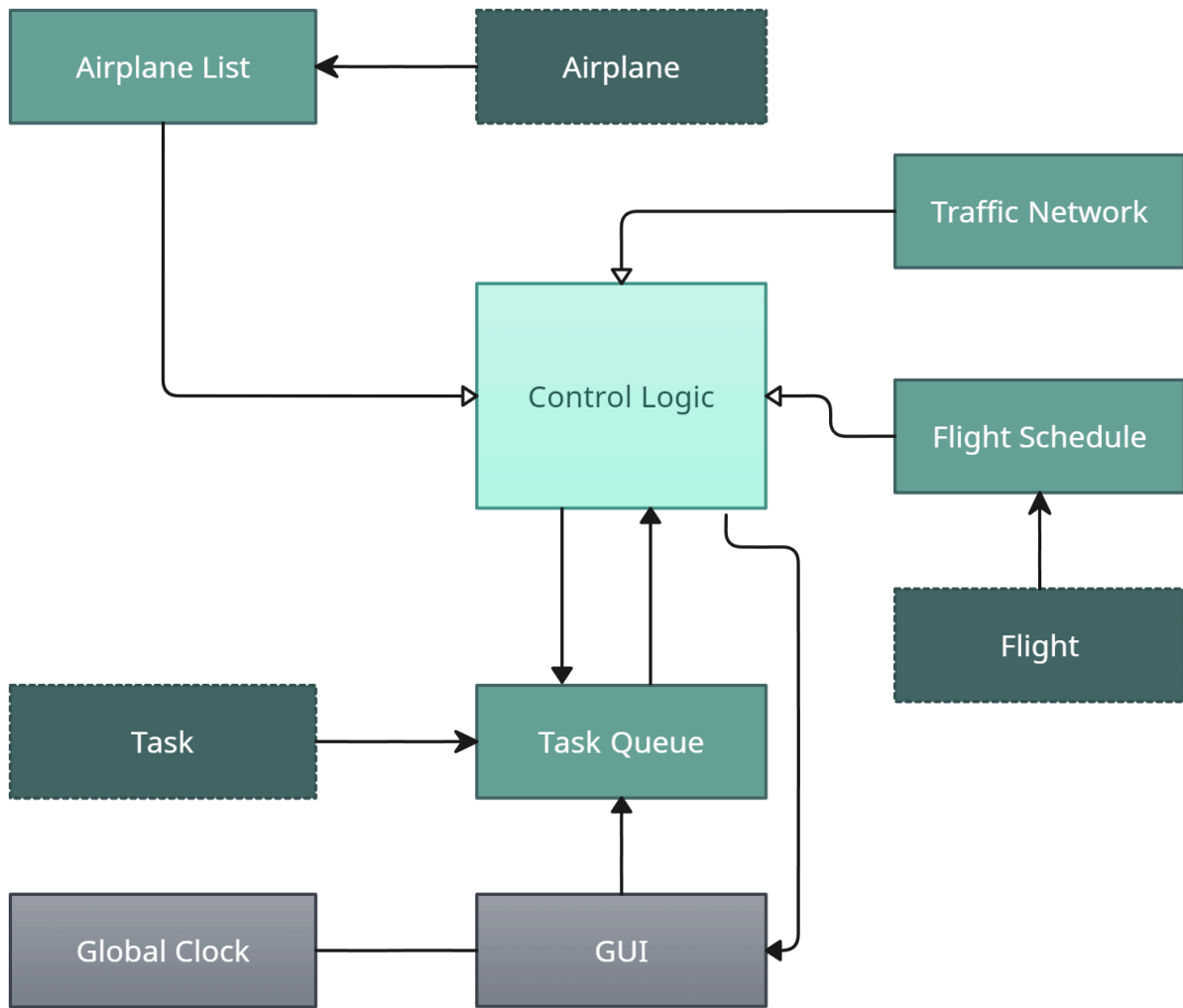
```
+futureStatusGate():StringBuilder  
+futureStatusTaxiway():StringBuilder  
+futureStatusRunway():StringBuilder
```



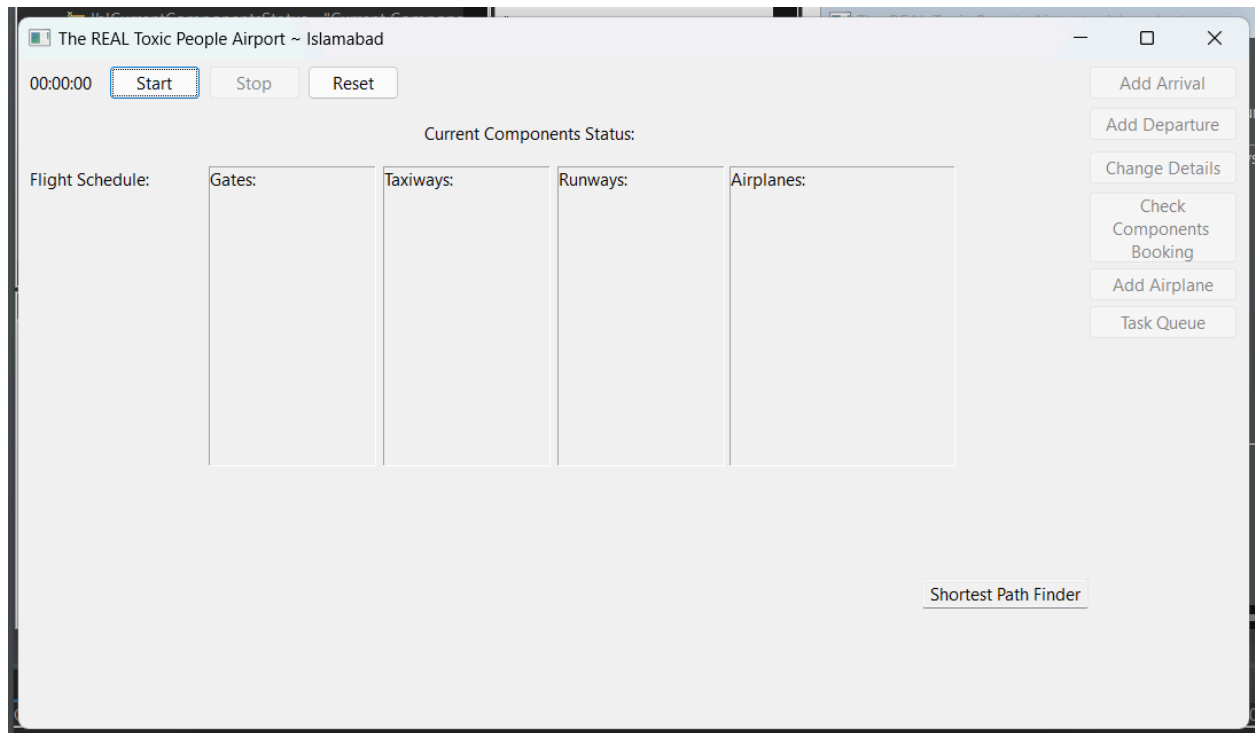
Node
-name:String i shortestPath:LinkedList<Node> -distance:int -adjacentNodes:HashMap<Node,Int>
+Node() +addDestination(Node, int):void +getName():String +setName(String):void +getAdjacentNodes():Map +setAdjacentNodes(Map, Int, adjacentNodes):void +getDistance():Int +setDistance(Int):void +getShortestPath():List +setShortestPath(LinkedList):void

Graph
-nodes:HashSet<Nodes>
+addNode(Node):void +getNodes():Set<Node> +setNodes(Set):void +getNodeByName(String):Node

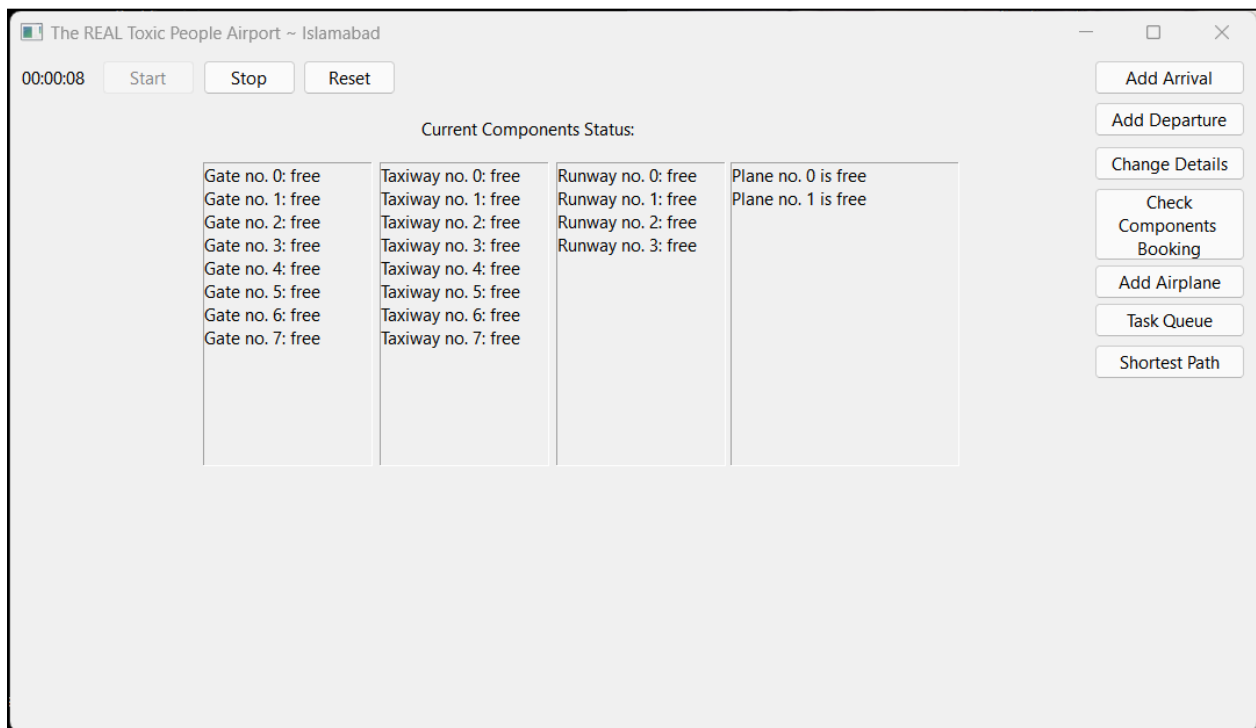
Dijkstra
<gets implemented in the ShortestPath class with a GUI and predefined map data for our model>
<u>+calculateShortestPathFromSource(Graph, Node):Graph</u> <u>+CalculateMinimumDistance(Node):void</u> <u>+getLowestDistanceNode(Set):Node</u>



## Results and output



*Initial state*



*Starting simulation*

SWT Application

Components booking (with time)

Gates:	Taxiways:	Runways:	Airplanes:
Gate no. 0: Not booked	Taxiway no. 0: Not Booked	Runway no. 0: Not Booked	Plane ID: 0: Not Booked
Gate no. 1: Not booked	Taxiway no. 1: Not Booked	Runway no. 1: Not Booked	Plane ID: 1: Not Booked
Gate no. 2: Not booked	Taxiway no. 2: Not Booked	Runway no. 2: Not Booked	
Gate no. 3: Not booked	Taxiway no. 3: Not Booked	Runway no. 3: Not Booked	
Gate no. 4: Not booked	Taxiway no. 4: Not Booked		
Gate no. 5: Not booked	Taxiway no. 5: Not Booked		
Gate no. 6: Not booked	Taxiway no. 6: Not Booked		
Gate no. 7: Not booked	Taxiway no. 7: Not Booked		

*Initial components booking*

**For Plane Departure:**

Add Plane Departure

Location:  (to)

Time: Hours:  Mins:  Secs:

*Adding departure form*

Add Plane Departure

Location:  (to)

Time: Hours:  Mins:  Secs:

*Error This time will create errors for file, choose another time*

*Adding departure form checks*

The REAL Toxic People Airport ~ Islamabad

00:00:48

Start

Stop

Reset

Add Arrival

Add Departure

Change Details

Check Components Booking

Add Airplane

Task Queue

Shortest Path

Current Components Status:

<div>Departure, Plane no.1</div> <div>Location: Gwadar</div> <div>Time:00:05:00</div>	<div>Gate no. 0: free</div> <div>Gate no. 1: free</div> <div>Gate no. 2: free</div> <div>Gate no. 3: free</div> <div>Gate no. 4: free</div> <div>Gate no. 5: free</div> <div>Gate no. 6: free</div> <div>Gate no. 7: free</div>	<div>Taxiway no. 0: free</div> <div>Taxiway no. 1: free</div> <div>Taxiway no. 2: free</div> <div>Taxiway no. 3: free</div> <div>Taxiway no. 4: free</div> <div>Taxiway no. 5: free</div> <div>Taxiway no. 6: free</div> <div>Taxiway no. 7: free</div>	<div>Runway no. 0: free</div> <div>Runway no. 1: free</div> <div>Runway no. 2: free</div> <div>Runway no. 3: free</div>	<div>Plane no. 0 is free</div> <div>Plane no. 1 is free</div>
---	---	---	---	---

Adding departure

SWT Application

Components booking (with time)

<div>Gates:</div> <div>Gate no. 0: Not booked</div> <div>Gate no. 1: Not booked</div> <div>Gate no. 2: Not booked</div> <div>Gate no. 3: Not booked</div> <div>Gate no. 4: Not booked</div> <div>Gate no. 5: Not booked</div> <div>Gate no. 6: Not booked</div> <div>Gate no. 7: Booked</div> <div>Time: 00:02:00</div>	<div>Taxiways:</div> <div>Taxiway no. 0: Not Booked</div> <div>Taxiway no. 1: Not Booked</div> <div>Taxiway no. 2: Not Booked</div> <div>Taxiway no. 3: Not Booked</div> <div>Taxiway no. 4: Not Booked</div> <div>Taxiway no. 5: Not Booked</div> <div>Taxiway no. 6: Not Booked</div> <div>Taxiway no. 7: Booked</div> <div>Time: 00:03:00</div>	<div>Runways:</div> <div>Runway no. 0: Not Booked</div> <div>Runway no. 1: Not Booked</div> <div>Runway no. 2: Not Booked</div> <div>Runway no. 3: Booked</div> <div>Time: 00:04:00</div>	<div>Airplanes:</div> <div>Plane ID: 0: Not Booked</div> <div>Plane ID: 1: Booked</div>
---	--	---	---

Components booking after adding departure

00:02:04

Start

Stop

Reset

Add Arrival

Add Departure

Change Details

Check Components Booking

Add Airplane

Task Queue

Shortest Path

Current Components Status:

Departure, Plane no.1 Location: Gwadar Time:00:05:00	Gate no. 0: free Gate no. 1: free Gate no. 2: free Gate no. 3: free Gate no. 4: free Gate no. 5: free Gate no. 6: free Gate no. 7: busy	Taxiway no. 0: free Taxiway no. 1: free Taxiway no. 2: free Taxiway no. 3: free Taxiway no. 4: free Taxiway no. 5: free Taxiway no. 6: free Taxiway no. 7: free	Runway no. 0: free Runway no. 1: free Runway no. 2: free Runway no. 3: free	Plane no. 0 is free Plane no. 1 is on Gate no. 7
--	--	--	--	--

Plane at gate

00:03:03

Start

Stop

Reset

Add Arrival

Add Departure

Change Details

Check Components Booking

Add Airplane

Task Queue

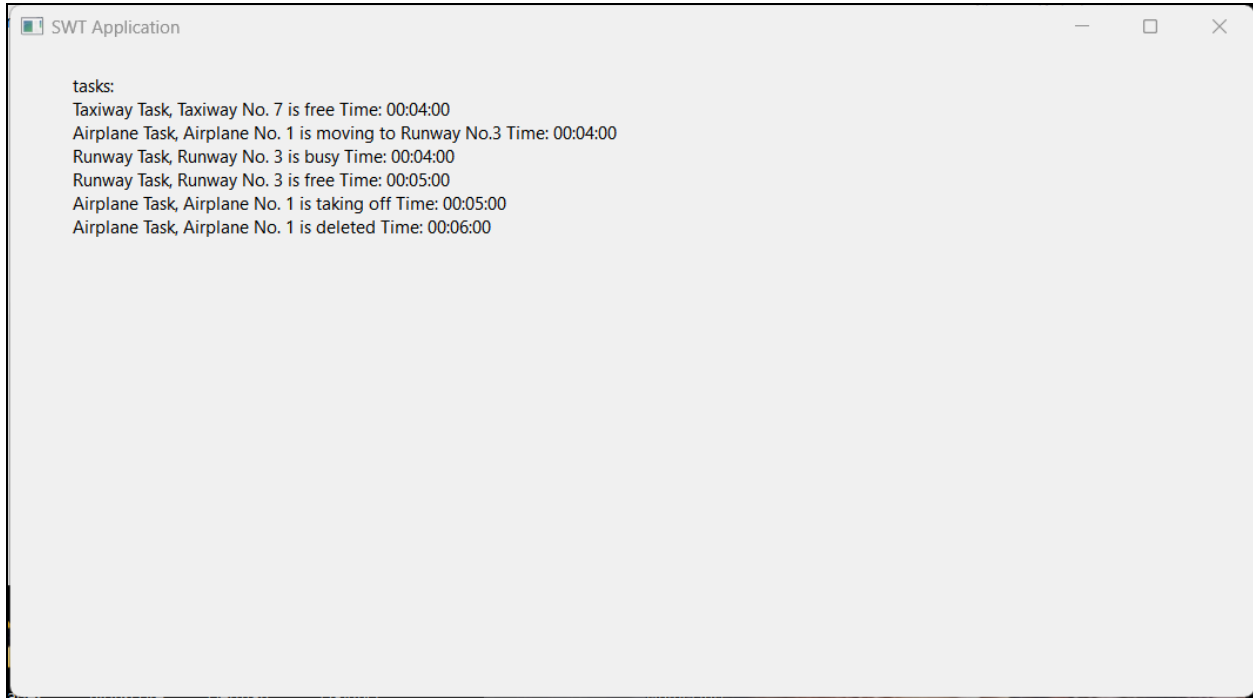
Shortest Path

Current Components Status:

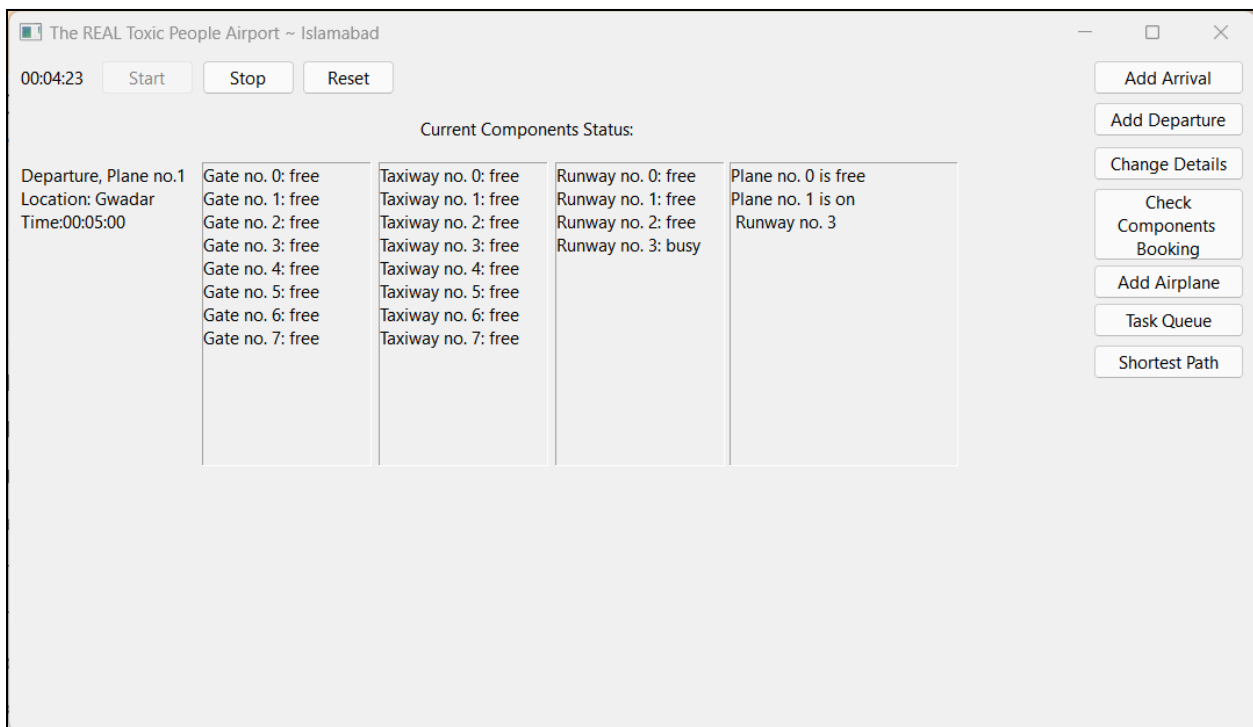
Departure, Plane no.1 Location: Gwadar Time:00:05:00	Gate no. 0: free Gate no. 1: free Gate no. 2: free Gate no. 3: free Gate no. 4: free Gate no. 5: free Gate no. 6: free Gate no. 7: free	Taxiway no. 0: free Taxiway no. 1: free Taxiway no. 2: free Taxiway no. 3: free Taxiway no. 4: free Taxiway no. 5: free Taxiway no. 6: free Taxiway no. 7: busy	Runway no. 0: free Runway no. 1: free Runway no. 2: free Runway no. 3: free	Plane no. 0 is free Plane no. 1 is on Taxiway no. 7
--	--	--	--	---

Plane at taxiway

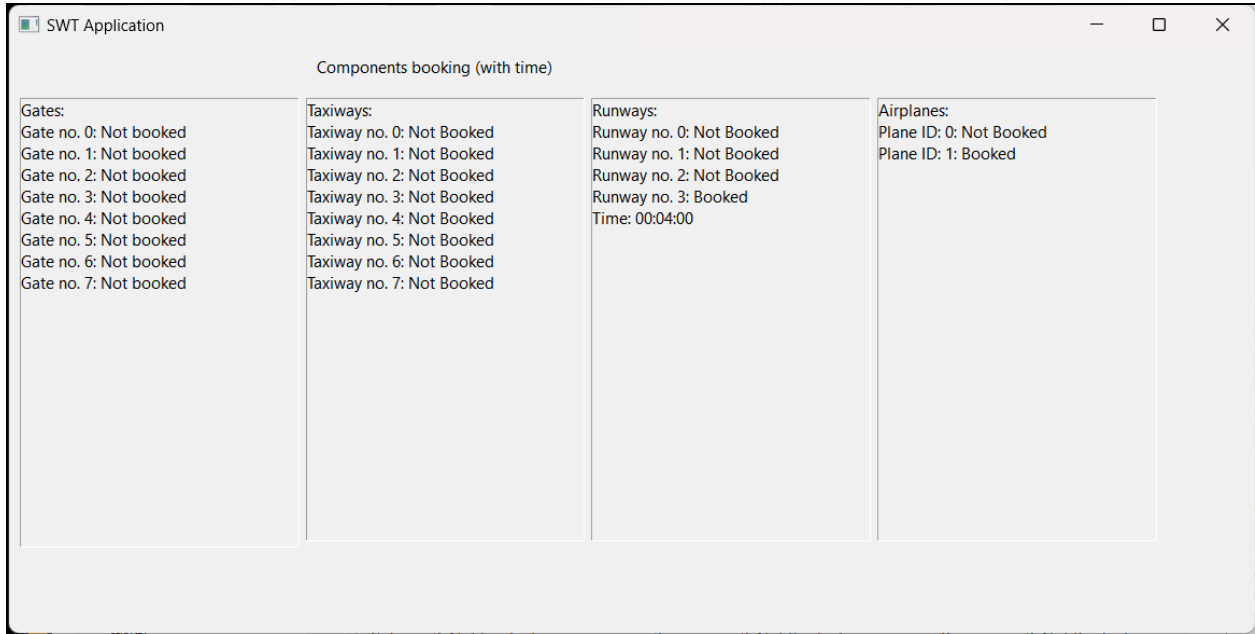




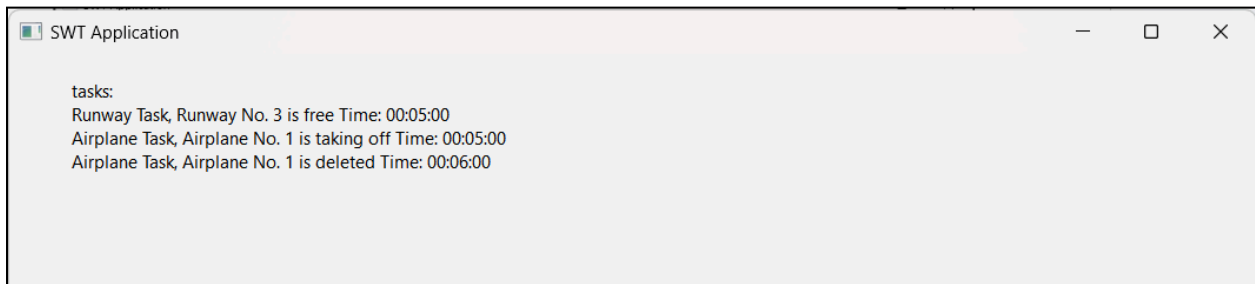
*Task Queue after plane is at taxiway*



*Plane at runway*



*Components booking when plane is at runway*



*Task queue display when plane is in runway*

00:05:05

Start

Stop

Reset

Departure, Plane no.1

Location: Gwadar

Time:00:05:00

Gate no. 0: free	Taxiway no. 0: free	Runway no. 0: free	Plane no. 0 is free
Gate no. 1: free	Taxiway no. 1: free	Runway no. 1: free	Plane no. 1 is in air
Gate no. 2: free	Taxiway no. 2: free	Runway no. 2: free	
Gate no. 3: free	Taxiway no. 3: free	Runway no. 3: free	
Gate no. 4: free	Taxiway no. 4: free		
Gate no. 5: free	Taxiway no. 5: free		
Gate no. 6: free	Taxiway no. 6: free		
Gate no. 7: free	Taxiway no. 7: free		

Current Components Status:

Add Arrival

Add Departure

Change Details

Check Components Booking

Add Airplane

Task Queue

Shortest Path

*Plane is in the air*

00:06:05

Start

Stop

Reset

Gate no. 0: free

Gate no. 1: free

Gate no. 2: free

Gate no. 3: free

Gate no. 4: free

Gate no. 5: free

Gate no. 6: free

Gate no. 7: free

Taxiway no. 0: free	Taxiway no. 1: free	Runway no. 0: free	Plane no. 0 is free
Taxiway no. 2: free	Taxiway no. 3: free	Runway no. 1: free	
Taxiway no. 4: free	Taxiway no. 5: free	Runway no. 2: free	
Taxiway no. 6: free	Taxiway no. 7: free	Runway no. 3: free	

Current Components Status:

Add Arrival

Add Departure

Change Details

Check Components Booking

Add Airplane

Task Queue

Shortest Path

*Plane deleted*

For plane arrival, the outputs are similar, but opposite in time

**Add Plane Arrival**

Location:  (to)

Time: Hours:  Mins:  Secs:

*Add arrival form*

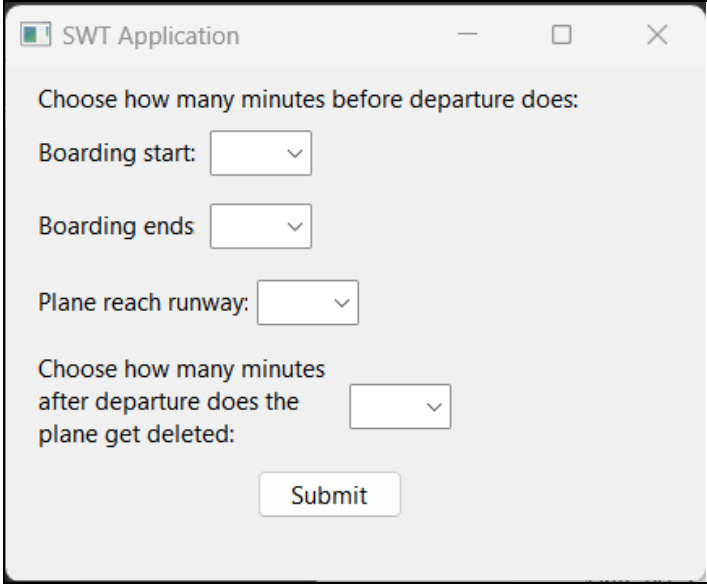
**The REAL Toxic People Airport ~ Islamabad**

00:01:20

Current Components Status:

Gate no. 0: free	Taxiway no. 0: free	Runway no. 0: free	Plane no. 0 is free
Gate no. 1: free	Taxiway no. 1: free	Runway no. 1: free	Plane no. 1 is free
Gate no. 2: free	Taxiway no. 2: free	Runway no. 2: free	Plane no. 2 is free
Gate no. 3: free	Taxiway no. 3: free	Runway no. 3: free	
Gate no. 4: free	Taxiway no. 4: free		
Gate no. 5: free	Taxiway no. 5: free		
Gate no. 6: free	Taxiway no. 6: free		
Gate no. 7: free	Taxiway no. 7: free		

*Adding airplane*



SWT Application

Choose how many minutes before departure does:

Boarding start:

Boarding ends:

Plane reach runway:

Choose how many minutes after departure does the plane get deleted:

Submit

*Change details form*

## Mathematical Modeling

Dijkstra's algorithm is a pathfinding algorithm used to find the shortest possible path from one point to another in a weighted graph. The algorithm works by initializing the source node with 0 and the rest of the nodes with infinity. A node is said to be settled when its minimum distance from the source node is found and it is said to be unsettled if this condition is not fulfilled.

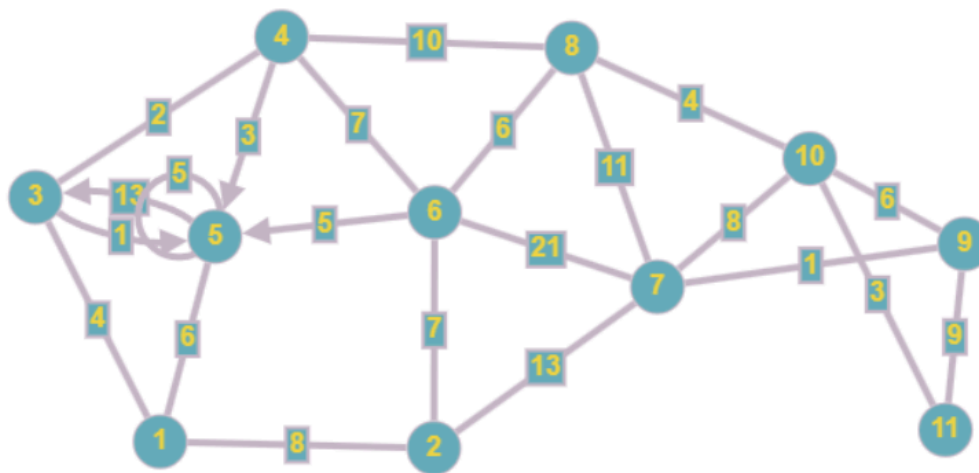
The algorithm starts at the source node. We get the neighboring nodes using an adjacency matrix or an adjacency list. In the code, we have used a 'Map.Entry' object to denote adjacent nodes with their weights. The closest adjacent node is used. This closest node is identified by choosing the adjacent node with the smallest edge weight. We calculate the distance from the source towards that node by adding its edge weight to the current path's weight and if it is smaller than the previous distance value for that node, the previous value is replaced with the newly calculated value. This process is then repeated for every node that is unsettled until all nodes have been settled. This allows us to iterate through all the nodes of the graph and check if any node has a shorter path from one node than another.

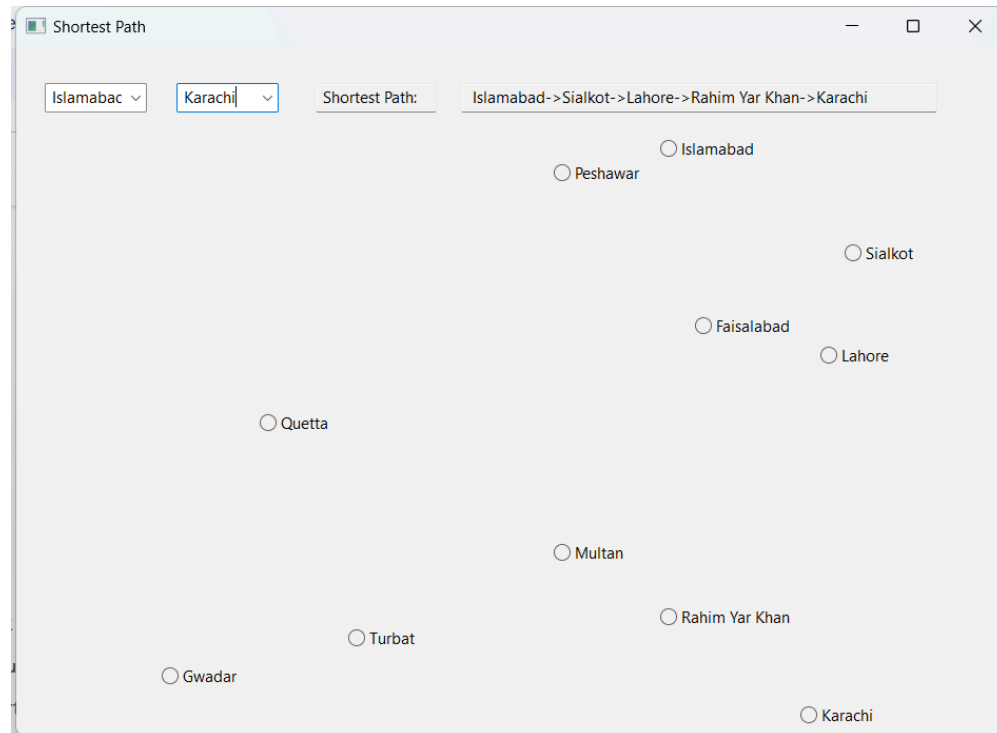
Implementation in the project:

We have modeled an airport map of Pakistan where only certain cities have direct flights between them. For the rest of the cities, planes must arrive at their destination through

connecting flights. Different transit locations have different distances between them. A flight must choose the shortest possible path through these connecting flights for maximum efficiency. This problem has been solved using Dijkstra's algorithm which takes our airport simulation as the source node. The user specifies the destination for the plane and then the algorithm is used to display the shortest possible path for that flight to take.

The map modeled as a graph:





*Shortest path window*

### **Limitations and Conclusion:**

The simulation, like a real airport, has the physical constraint of the number of runways, taxiways, and gates. We are hardcoding the limit, and are not allowing the user to have unlimited links.

While we have not limited the number of departures and arrivals that the user can add, our simulation cannot accommodate more than 8 queued at a time. While we have the solution for this, we were unable to implement it in the given time constraints.