NATIONAL INSTITUTE OF TECHONOLOGY
ROURKELA

PROJECT REPORT
ON
Airport Runway Management Simulation

Under the guidance of

Prof.Bibhudatta Sahoo,

HOD (Computer Science and Engineering)

Introduction:

Managing airport runways is a critical aspect of air traffic control. This simulation models the operation of an airport with three runways. It incorporates realistic challenges such as emergency landings, limited runway availability, and varying plane arrival/departure rates. The simulation demonstrates how an airport might handle these scenarios efficiently, with a focus on reducing crashes, minimizing waiting times, and ensuring smooth operations.

This report explains the simulation's objectives, functionality, workflow, key metrics, results, and future improvements.

Objectives:

The simulation was designed to:

1. Simulate real-world runway operations at an airport with **three runways**.
2. Prioritize safety by:
    - Managing **emergency landings** for planes running critically low on fuel.
    - Preventing crashes due to fuel depletion.
3. Optimize airport efficiency by:
    - Allocating planes to runways for both landings and take-offs based on availability and priority.
    - Reducing average waiting times for landing and take-off planes.
4. Provide clear performance metrics to evaluate the system, such as:
    - Average waiting times.
    - Total crashes and successful operations.

Simulation Design:

The Plane structure represents each plane in the simulation. It includes:

- **ID**: A unique identifier distinguishing each plane.
- **Fuel Level**: Indicates the remaining fuel for landing planes. Planes with low fuel have higher landing priority.
- **Waiting Time**: Tracks the time the plane has been waiting to land or take off.
- **Landing/Takeoff Status**: Determines whether the plane is waiting to land or take off.

Landing Queue

- Landing planes are managed using a **priority queue**.
- Planes with lower fuel levels are prioritized for landing, simulating real-world air traffic control practices.

Takeoff Queues

- Each runway has a dedicated **takeoff queue** implemented using the queue data structure.
- Planes in the takeoff queues are assigned runways based on congestion and availability.

Emergency Queue

- Planes with critically low fuel (fuel level = 0) are moved to an **emergency queue**.
- Emergency landings are prioritized on a designated emergency runway (EMERGENCY_LANDING_RUNWAY).

Strengths of the Simulation

1. **Realistic Priority Management**: The use of a priority queue for landing planes ensures critical scenarios like low fuel are handled appropriately.
2. **Dynamic Take-off Allocation**: Balancing between regular and priority take-offs helps manage runway congestion.
3. **Modular Design**: The simulation can be easily adapted to different scenarios by modifying constants.
4. **Efficient Landing Management**: A **priority queue** prioritizes planes with lower fuel levels, ensuring critical landings happen first. This avoids crashes and reduces computational overhead compared to manual sorting.
5. **Balanced Take-off Allocation**: Separate **FIFO queues** for each runway distribute take off tasks evenly, minimizing delays and ensuring no single runway becomes a bottleneck.
6. **Minimal Resource Usage**: Lightweight data structures like queue and priority queue are used, with updates performed in-place, reducing memory consumption and processing time.
7. **Emergency Handling**: Critical planes are immediately assigned to an emergency queue and prioritized for landing, preventing unnecessary crashes.
8. **Dynamic Adaptability**: The code dynamically reallocates runways based on congestion and traffic thresholds, ensuring resources are optimally utilized.
9. **Modular Design**: The clear separation of logic into functions simplifies maintenance and future scalability.
10. **Real-Time Readings**: At each simulation step (time unit), the program displays key metrics:

- Status of landing and take-off queues.
- Average waiting times for landing and take-off.
- Number of crashes and successfully handled planes. This provides a time- to-time update on the airport's performance.

CODE:

```
#include <iostream>
#include <queue>
#include <cstdlib>
#include <ctime>
using namespace std;

const int MAX_DEPARTURES = 3;
const int NO_OF_RUNWAYS = 3;
const int EMERGENCY_FUEL = 0;
const int MAX_ARRIVALS = 2;
const int Limit_for_takeoff_queue=5;
const int EMERGENCY_LANDING_RUNWAY=2;

struct Plane
{
   int id;
   int fuel_level;
   int waitingtime;
   bool isLanding;

   // Comparator for priority queue (low fuel has higher priority)
   bool operator<(const Plane &other) const
   {
      return fuel_level > other.fuel_level; // Lower fuel comes first
   }

   Plane(int id, int fuel_level, int waitingtime = 0)
      : id(id), fuel_level(fuel_level), waitingtime(waitingtime), isLanding(!(id % 2)) {}
```

```
};

priority_queue<Plane> landingPriorityQueue;
queue<Plane> takeoffQueue[MAX_DEPARTURES];
queue<Plane> EMERGENCY;

int total_TakeoffWaitingTime = 0, total_LandingWaitingTime = 0;
int Departure_Count = 0, Arrival_Count = 0;
int crashCount = 0;
// here we are going check the simulation in each time unit
int timeUnit = 0;

void Generate_Random_Planes()
{

  // We will generate Random planes and add them into simulation
  //  Add up to MAX_ARRIVALS planes to the landing queue
  int No_landing_planes = rand() % MAX_ARRIVALS + 1;
  for (int i = 0; i < No_landing_planes; i++)
  {
    Plane newPlane = {2 * (timeUnit * MAX_ARRIVALS + i), rand() % 5 + 1, 0}; // Random fuel (1-5)
    landingPriorityQueue.push(newPlane);
  }
  // Add up to MAX_ARRIVALS planes to the takeoff queues
  int No_of_takeoff_planes = rand() % MAX_DEPARTURES + 1;
  for (int i = 0; i < No_of_takeoff_planes; i++)
  {
    Plane newPlane = {2 * (timeUnit * MAX_ARRIVALS + i) + 1, 0, 0};

    // Assign to the least busy queue
    int minQueue = 0;
```

```
        for (int j = 1; j < NO_OF_RUNWAYS; j++)
        {
            if (takeoffQueue[j].size() < takeoffQueue[minQueue].size())
            {
                minQueue = j;
            }
        }
        takeoffQueue[minQueue].push(newPlane);
    }
}

void Simulation_Per_One_timeunit()
{
    // Here we are going to check the simulation in each time unit
    // Track free runways
    vector<bool> isRunwayFree(NO_OF_RUNWAYS, true);
    vector<Plane> assignedPlanes(NO_OF_RUNWAYS, {0, 0, 0}); // Planes assigned to runways

    // Prioritize emergency landings

    if (!EMERGENCY.empty() && isRunwayFree[EMERGENCY_LANDING_RUNWAY])
    {
        Plane emergencyPlane = EMERGENCY.front();
        EMERGENCY.pop();
        assignedPlanes[EMERGENCY_LANDING_RUNWAY] = emergencyPlane; // Assign to Runway
EMERGENCY_LANDING_RUNWAY
        isRunwayFree[EMERGENCY_LANDING_RUNWAY] = false;          // Mark Runway
EMERGENCY_LANDING_RUNWAY as occupied
        Arrival_Count++;
        total_LandingWaitingTime += emergencyPlane.waitingtime;

        cout << "Emergency Landing: Plane " << emergencyPlane.id
        << " on Runway EMERGENCY_LANDING_RUNWAY (Fuel: " << emergencyPlane.fuel_level <<
")" << endl;
    }

    // Regular landings (exclude Runway EMERGENCY_LANDING_RUNWAY if still free)
    for (int i = 0; i < NO_OF_RUNWAYS; i++)
    {
        if (i == EMERGENCY_LANDING_RUNWAY && isRunwayFree[i])
            continue; // Keep Runway EMERGENCY_LANDING_RUNWAY free unless necessary

        if (isRunwayFree[i] && !landingPriorityQueue.empty())
        {
            Plane landingPlane = landingPriorityQueue.top();
            landingPriorityQueue.pop();

            if (landingPlane.fuel_level > 0)
            {
                assignedPlanes[i] = landingPlane;
                isRunwayFree[i] = false;
                Arrival_Count++;
                total_LandingWaitingTime += landingPlane.waitingtime;

                cout << "Landing: Plane " << landingPlane.id << " on Runway " << i << " (Fuel: " <<
landingPlane.fuel_level << ")" << endl;
            }
            else
            {
                crashCount++;
                cout << "Crash: Plane " << landingPlane.id << " ran out of fuel!" << endl;
            }
        }
    }
    // Threshold-based takeoff allocation
    int maxQueueSize = 0;
    int maxQueueIndex = -1;
    for (int i = 0; i < NO_OF_RUNWAYS; i++)
    {
        if ((int)takeoffQueue[i].size() > maxQueueSize)
        {
            maxQueueSize = takeoffQueue[i].size();
            maxQueueIndex = i;
        }
    }

    if (maxQueueSize > Limit_for_takeoff_queue)
    { // Handle congestion in takeoff queues
        for (int i = 0; i < NO_OF_RUNWAYS; i++)
        {
            if (isRunwayFree[i])
            {
                Plane takeoffPlane = takeoffQueue[maxQueueIndex].front();
                takeoffQueue[maxQueueIndex].pop();
                assignedPlanes[i] = takeoffPlane;
                isRunwayFree[i] = false;
                Departure_Count++;
                total_LandingWaitingTime += takeoffPlane.waitingtime;

                cout << "Priority Takeoff: Plane " << takeoffPlane.id << " from Runway " << i << " (Threshold
Logic)" << endl;
                break;
            }
        }
```

```
}

// Regular takeoffs
for (int i = 0; i < NO_OF_RUNWAYS; i++)
{
    if (isRunwayFree[i])
    {
        for (int j = 0; j < NO_OF_RUNWAYS; j++)
        {
            if (!takeoffQueue[j].empty())
            {
                Plane takeoffPlane = takeoffQueue[j].front();
                takeoffQueue[j].pop();

                assignedPlanes[i] = takeoffPlane;
                isRunwayFree[i] = false;
                Departure_Count++;
                total_TakeoffWaitingTime += takeoffPlane.waitingtime;

                cout << "Takeoff: Plane " << takeoffPlane.id << " from Runway " << i << endl;
                break;
            }
        }
    }
}

// Update waiting times and fuel levels
vector<Plane> tempLandingQueue;
while (!landingPriorityQueue.empty())
{
    Plane p = landingPriorityQueue.top();
    landingPriorityQueue.pop();

    p.fuel_level--; // Decrease fuel
    p.waitingtime++;

    if (p.fuel_level == EMERGENCY_FUEL)
    {
        EMERGENCY.push(p);
    }
    else
    {
        tempLandingQueue.push_back(p);
    }
}

// Rebuild landing priority queue with updated planes
for (Plane &p : tempLandingQueue)
{
    landingPriorityQueue.push(p);
}

// Update waiting times for takeoff planes
for (int i = 0; i < NO_OF_RUNWAYS; i++)
{
    int size = takeoffQueue[i].size();
    for (int j = 0; j < size; j++)
    {
        Plane p = takeoffQueue[i].front();
        takeoffQueue[i].pop();
        p.waitingtime++;
        takeoffQueue[i].push(p);
        }
    }
}

void printStatistics()
{
    cout << "Time Unit: " << timeUnit << endl;

    cout << "Takeoff Queues:" << endl;
    for (int i = 0; i < NO_OF_RUNWAYS; i++)
    {
        cout << "Queue " << i << ": ";
        queue<Plane> temp = takeoffQueue[i];
        while (!temp.empty())
        {
            cout << temp.front().id << " ";
            temp.pop();
        }
        cout << endl;
    }

    cout << "Average Landing Waiting Time: "
         << (Arrival_Count > 0 ? (double)total_LandingWaitingTime / Arrival_Count : 0) << endl;
    cout << "Average Takeoff Waiting Time: "
         << (Departure_Count > 0 ? (double)total_TakeoffWaitingTime / Departure_Count : 0) << endl;
    cout << "Planes Crashed: " << crashCount << endl;
    cout << "----------------------------------" << endl;
}

int main()
{
    srand(time(0));         // Seed for random number generation
    int totalTimeUnits = 10; // Simulation length

    for (timeUnit = 1; timeUnit <= totalTimeUnits; timeUnit++)
    {
        Generate_Random_Planes();
        Simulation_Per_One_timeunit();
        printStatistics();
    }

    return 0;
}
```

OUTPUT:

```
Landing: Plane 4 on Runway 0 (Fuel: 3)
Landing: Plane 6 on Runway 1 (Fuel: 5)
Takeoff: Plane 5 from Runway 2
Time Unit: 1
Takeoff Queues:
Queue 0:
Queue 1:
Queue 2:
Average Landing Waiting Time: 0
Average Takeoff Waiting Time: 0
Planes Crashed: 0
----------------------------------
Landing: Plane 8 on Runway 0 (Fuel: 2)
Takeoff: Plane 9 from Runway 1
Takeoff: Plane 11 from Runway 2
Time Unit: 2
```

```
Takeoff Queues:
Queue 0:
Queue 1:
Queue 2: 13
Average Landing Waiting Time: 0
Average Takeoff Waiting Time: 0
Planes Crashed: 0
----------------------------------
Landing: Plane 12 on Runway 0 (Fuel: 1)
Landing: Plane 14 on Runway 1 (Fuel: 4)
Takeoff: Plane 13 from Runway 2
Time Unit: 3
Takeoff Queues:
Queue 0: 17
Queue 1: 15
Queue 2: 13
Average Landing Waiting Time: 0
Average Takeoff Waiting Time: 0
Planes Crashed: 0
----------------------------------
Landing: Plane 16 on Runway 0 (Fuel: 5)
Takeoff: Plane 17 from Runway 1
Takeoff: Plane 17 from Runway 2
Time Unit: 4
Takeoff Queues:
Queue 0:
Queue 1: 15 19
Queue 2: 13 21
Average Landing Waiting Time: 0
Average Takeoff Waiting Time: 0.166667
Planes Crashed: 0
```

```
----------------------------------
Landing: Plane 20 on Runway 0 (Fuel: 1)
Landing: Plane 22 on Runway 1 (Fuel: 5)
Takeoff: Plane 21 from Runway 2
Time Unit: 5
Takeoff Queues:
Queue 0: 23
Queue 1: 15 19
Queue 2: 13 21
Average Landing Waiting Time: 0
Average Takeoff Waiting Time: 0.142857
Planes Crashed: 0
----------------------------------
Landing: Plane 24 on Runway 0 (Fuel: 1)
Takeoff: Plane 23 from Runway 1
Takeoff: Plane 25 from Runway 2
Time Unit: 6
Takeoff Queues:
Queue 0: 27
Queue 1: 15 19 29
Queue 2: 13 21
Average Landing Waiting Time: 0
Average Takeoff Waiting Time: 0.222222
Planes Crashed: 0
----------------------------------
Landing: Plane 28 on Runway 0 (Fuel: 3)
Takeoff: Plane 27 from Runway 1
Takeoff: Plane 29 from Runway 2
Time Unit: 7
Takeoff Queues:
Queue 0:
```

```
Queue 1: 15 19 29
Queue 2: 13 21
Average Landing Waiting Time: 0
Average Takeoff Waiting Time: 0.272727
Planes Crashed: 0
----------------------------------
Landing: Plane 32 on Runway 0 (Fuel: 2)
Landing: Plane 34 on Runway 1 (Fuel: 2)
Takeoff: Plane 33 from Runway 2
Time Unit: 8
Takeoff Queues:
Queue 0: 35
Queue 1: 15 19 29
Queue 2: 13 21
Average Landing Waiting Time: 0
Average Takeoff Waiting Time: 0.25
Planes Crashed: 0
----------------------------------
Landing: Plane 36 on Runway 0 (Fuel: 3)
Takeoff: Plane 35 from Runway 1
Takeoff: Plane 37 from Runway 2
Time Unit: 9
Takeoff Queues:
Queue 0:
Queue 1: 15 19 29
Queue 2: 13 21
Average Landing Waiting Time: 0
Average Takeoff Waiting Time: 0.285714
Planes Crashed: 0
----------------------------------
Landing: Plane 40 on Runway 0 (Fuel: 3)
```

```
Takeoff: Plane 41 from Runway 1

Takeoff: Plane 43 from Runway 2

Time Unit: 10

Takeoff Queues:

Queue 0:

Queue 1: 15 19 29

Queue 2: 13 21

Average Landing Waiting Time: 0

Average Takeoff Waiting Time: 0.25

Planes Crashed: 0
```

Conclusion

The Airport Runway Management Simulation successfully models the complexities of air traffic control, demonstrating how priority management, efficient resource allocation, and emergency handling can be implemented. While the simulation performs well under predefined conditions, there is potential for further enhancement to better replicate real-world scenarios and handle larger scales.

References:

- **C++ Documentation: https://cplusplus.com**
- **GeeksforGeeks: QUEUE**

SUBMITTED BY:

SAI CHARAN GOKA,

123CS0168