Lifecare Hospital ER Simulation

A comprehensive **Discrete Event Simulation (DES)** model for analyzing Emergency Room performance and optimizing staffing decisions. This project implements a queue-based simulation system that models patient flow through triage and consultation stages.

📋 Project Overview

This simulation helps hospital administrators understand:

- **Patient flow** through the ER system
- **Bottleneck identification** in triage vs. consultation
- **Resource utilization** of nurses and doctors
- **Wait time optimization** strategies
- **Staffing recommendations** based on data analysis

🏥 System Model

The simulation models a two-stage ER process:

```
Patient Arrival → Triage Queue → Triage Nurse → Consultation Queue → Doctor → Discharge
```

System Parameters

- **Patient Arrivals**: Exponential distribution (mean: 4 minutes)
- **Triage Service**: Uniform distribution (3-6 minutes)
- **Consultation Service**: Uniform distribution (5-15 minutes)
- **Shift Duration**: 8 hours (480 minutes)
- **Resources**: 2 triage nurses, 3 doctors (configurable)

📁 Project Structure

```
project-2/
├── er_simulation.py          # Core simulation engine
├── scenario_analysis.py      # Comprehensive scenario testing
├── demo.py                   # Quick demo script
├── test_simulation.py        # Verification and testing
├── requirements.txt          # Python dependencies
├── README.md                 # This documentation
└── PROJECT_SUMMARY.md        # Detailed project summary
```

🚀 Installation

1. **Clone or download** the project files

2. **Install dependencies**:

```
pip install -r requirements.txt
```

💻 Usage

Quick Demo

Run a single simulation with baseline parameters:

```
python demo.py
```

Comprehensive Analysis

Run all scenarios and generate recommendations:

```
python scenario_analysis.py
```

Custom Simulation

```
from er_simulation import ERSimulation

# Create simulation with custom parameters
sim = ERSimulation(
    num_triage_nurses=3,
    num_doctors=4,
    shift_duration=600,  # 10 hours
    arrival_mean=3.5     # 3.5 minutes between arrivals
)

# Run simulation
results = sim.run()
sim.print_results()
```

🔧 Code Architecture

Core Classes

1. Event System

```
class EventType(Enum):
    PATIENT_ARRIVAL = "patient_arrival"
    TRIAGE_COMPLETE = "triage_complete"
```

```
    CONSULTATION_COMPLETE = "consultation_complete"

@dataclass
class Event:
    time: float
    event_type: EventType
    patient_id: int

    def __lt__(self, other):
        return self.time < other.time
```

Purpose: Manages simulation events using a priority queue for chronological processing.

Simple Explanation:

- **EventType**: This is like a list of different things that can happen in the hospital. We have three main events:
    - `PATIENT_ARRIVAL`: When a new patient shows up at the ER
    - `TRIAGE_COMPLETE`: When a nurse finishes checking a patient
    - `CONSULTATION_COMPLETE`: When a doctor finishes seeing a patient
- **Event**: This is like a note that says "something happened at a specific time". Each event has:
    - `time`: When this event happened (like 10:30 AM)
    - `event_type`: What kind of event it was (patient arrived, triage finished, etc.)
    - `patient_id`: Which patient this event is about (like patient #5)
- `__lt__` **method**: This tells the computer how to sort events by time. It's like putting events in order from earliest to latest.

2. Patient Tracking

```
@dataclass
class Patient:
    id: int
    arrival_time: float
    triage_start_time: Optional[float] = None
    triage_end_time: Optional[float] = None
    consultation_start_time: Optional[float] = None
    consultation_end_time: Optional[float] = None

    @property
    def triage_wait_time(self) -> float:
        if self.triage_start_time is None:
            return 0
        return self.triage_start_time - self.arrival_time

    @property
    def total_wait_time(self) -> float:
        return self.triage_wait_time + self.consultation_wait_time
```

Purpose: Tracks individual patient journey and calculates wait/service times.

Simple Explanation:

- **Patient**: This is like a patient's medical record that follows them through the ER. It keeps track of:
  - `id`: Patient's number (like patient #1, #2, etc.)
  - `arrival_time`: When they first walked into the ER
  - `triage_start_time`: When a nurse started checking them
  - `triage_end_time`: When the nurse finished checking them
  - `consultation_start_time`: When a doctor started seeing them
  - `consultation_end_time`: When the doctor finished seeing them
- `triage_wait_time`: This calculates how long the patient had to wait before a nurse could see them. It's like: "When did the nurse start seeing them?" minus "When did they arrive?"
- `total_wait_time`: This adds up all the waiting time - both waiting for the nurse and waiting for the doctor.

Think of it like a stopwatch that starts when the patient arrives and keeps track of every step of their journey through the ER.

3. Main Simulation Engine

```
class ERSimulation:
    def __init__(self,
                 num_triage_nurses: int = 2,
                 num_doctors: int = 3,
                 shift_duration: float = 480.0,
                 arrival_mean: float = 4.0,
                 triage_min: float = 3.0,
                 triage_max: float = 6.0,
                 consultation_min: float = 5.0,
                 consultation_max: float = 15.0):

        # System parameters
        self.num_triage_nurses = num_triage_nurses
        self.num_doctors = num_doctors
        self.shift_duration = shift_duration

        # Simulation state
        self.current_time = 0.0
        self.events = []  # Priority queue
        self.patients = {}  # Dictionary of all patients

        # Resource states
        self.available_triage_nurses = num_triage_nurses
        self.available_doctors = num_doctors

        # Queues
```

```
        self.triage_queue = []
        self.consultation_queue = []

        # Statistics tracking
        self.stats = {
            'total_patients': 0,
            'completed_patients': 0,
            'triage_wait_times': [],
            'consultation_wait_times': [],
            'total_wait_times': [],
            'triage_service_times': [],
            'consultation_service_times': [],
            'triage_queue_lengths': [],
            'consultation_queue_lengths': [],
            'triage_utilization': [],
            'doctor_utilization': [],
            'time_points': []
        }
```

Simple Explanation: This is like setting up a virtual hospital ER. Think of it as creating a digital version of the emergency room with:

System Setup (like hospital planning):

- `num_triage_nurses`: How many nurses are working (default: 2)
- `num_doctors`: How many doctors are working (default: 3)
- `shift_duration`: How long the shift lasts in minutes (default: 480 = 8 hours)
- `arrival_mean`: How often patients arrive (default: every 4 minutes on average)
- `triage_min/max`: How long triage takes (3-6 minutes)
- `consultation_min/max`: How long doctor consultation takes (5-15 minutes)

Hospital State (like the current situation):

- `current_time`: What time it is in the simulation (starts at 0)
- `events`: A list of things that will happen (like a schedule)
- `patients`: A list of all patients in the system

Available Staff (like who's free to work):

- `available_triage_nurses`: How many nurses are not busy right now
- `available_doctors`: How many doctors are not busy right now

Waiting Lines (like the waiting room):

- `triage_queue`: Patients waiting to see a nurse
- `consultation_queue`: Patients waiting to see a doctor

Statistics (like keeping track of everything):

- `total_patients`: How many patients have come in total
- `completed_patients`: How many patients have finished their treatment
- `triage_wait_times`: How long each patient waited for a nurse

- `consultation_wait_times`: How long each patient waited for a doctor
- And many more tracking items...

Key Methods

Event Scheduling

```
def _schedule_next_arrival(self):
    """Schedule the next patient arrival using exponential distribution"""
    if self.current_time < self.shift_duration:
        inter_arrival_time = random.expovariate(1.0 / self.arrival_mean)
        arrival_time = self.current_time + inter_arrival_time

        if arrival_time <= self.shift_duration:
            event = Event(arrival_time, EventType.PATIENT_ARRIVAL,
self.patient_id_counter)
            heapq.heappush(self.events, event)
```

Simple Explanation: This function is like a receptionist who decides when the next patient will arrive. Here's what it does:

1. **Check if shift is still going**: `if self.current_time < self.shift_duration`
    - Only schedule new patients if the hospital is still open
2. **Calculate arrival time**: `inter_arrival_time = random.expovariate(1.0 / self.arrival_mean)`
    - This uses a special math formula (exponential distribution) to randomly decide when the next patient will arrive
    - If `arrival_mean = 4`, patients arrive on average every 4 minutes, but sometimes it might be 2 minutes, sometimes 6 minutes
3. **Create the arrival event**: `event = Event(arrival_time, EventType.PATIENT_ARRIVAL, self.patient_id_counter)`
    - This creates a note saying "Patient #X will arrive at time Y"
4. **Add to schedule**: `heapq.heappush(self.events, event)`
    - This puts the event in a special list that keeps everything in time order

Real-world example: If it's 10:00 AM and patients arrive every 4 minutes on average, this might schedule the next patient to arrive at 10:03 AM, then the next at 10:07 AM, etc.

Service Time Generation

```
def _generate_triage_time(self) -> float:
    """Generate triage service time using uniform distribution"""
    return random.uniform(self.triage_min, self.triage_max)
```

```
    def _generate_consultation_time(self) -> float:
        """Generate consultation service time using uniform distribution"""
        return random.uniform(self.consultation_min, self.consultation_max)
```

Simple Explanation: These functions are like timers that decide how long each medical procedure will take. In real life, some patients take longer to treat than others.

`_generate_triage_time()`:

- This decides how long a nurse will spend checking a patient
- Uses `random.uniform(self.triage_min, self.triage_max)` which means:
  - If `triage_min = 3` and `triage_max = 6`
  - The nurse might spend 3 minutes, 4 minutes, 5 minutes, or 6 minutes
  - Each time is equally likely (uniform distribution)

`_generate_consultation_time()`:

- This decides how long a doctor will spend with a patient
- Uses `random.uniform(self.consultation_min, self.consultation_max)` which means:
  - If `consultation_min = 5` and `consultation_max = 15`
  - The doctor might spend 5 minutes, 8 minutes, 12 minutes, or 15 minutes
  - Each time is equally likely

Real-world example:

- A simple case might take 3 minutes for triage and 5 minutes for consultation
- A complex case might take 6 minutes for triage and 15 minutes for consultation
- The simulation randomly picks times within these ranges to make it realistic

Patient Flow Management

```
    def _handle_patient_arrival(self, patient_id: int):
        """Handle a new patient arrival"""
        patient = Patient(id=patient_id, arrival_time=self.current_time)
        self.patients[patient_id] = patient
        self.stats['total_patients'] += 1

        # Try to start triage immediately
        if self.available_triage_nurses > 0:
            self._start_triage(patient)
        else:
            # Add to triage queue
            self.triage_queue.append(patient)

        # Schedule next arrival
        self._schedule_next_arrival()
```

Simple Explanation: This function is like a receptionist who greets each new patient and decides what happens next. Here's the step-by-step process:

1. **Create patient record**: `patient = Patient(id=patient_id, arrival_time=self.current_time)`
    - This creates a new patient file with their ID and arrival time
    - Like filling out a new patient form
2. **Add to patient list**: `self.patients[patient_id] = patient`
    - This puts the patient in the hospital's master list
    - Like adding them to the hospital database
3. **Update statistics**: `self.stats['total_patients'] += 1`
    - This increases the count of total patients who have arrived
    - Like updating the daily patient count
4. **Check if nurse is available**: `if self.available_triage_nurses > 0:`
    - This checks if there's a free nurse to see the patient immediately
    - Like looking around to see if any nurses are free
5. **If nurse is free**: `self._start_triage(patient)`
    - Start the triage process right away
    - Like immediately calling a nurse to see the patient
6. **If no nurse is free**: `self.triage_queue.append(patient)`
    - Put the patient in the waiting line
    - Like telling the patient "Please take a seat, a nurse will be with you shortly"
7. **Schedule next patient**: `self._schedule_next_arrival()`
    - Plan when the next patient will arrive
    - Like the receptionist looking at their schedule

Real-world example:

- Patient #5 arrives at 10:30 AM
- Receptionist checks: "Is there a free nurse?"
- If yes: "Nurse Sarah, please see patient #5"
- If no: "Patient #5, please wait in the triage waiting area"

Resource Allocation

```
def _start_triage(self, patient: Patient):
    """Start triage for a patient"""
    self.available_triage_nurses -= 1
    patient.triage_start_time = self.current_time
```

```
        # Schedule triage completion
        triage_time = self._generate_triage_time()
        triage_end_time = self.current_time + triage_time

        event = Event(triage_end_time, EventType.TRIAGE_COMPLETE, patient.id)
        heapq.heappush(self.events, event)
```

Simple Explanation: This function is like a nurse manager who assigns a nurse to a patient and schedules when they'll finish. Here's what happens:

1. **Assign a nurse**: `self.available_triage_nurses -= 1`
    - This marks one nurse as "busy" (reduces the count of available nurses)
    - Like saying "Nurse Sarah is now busy with a patient"
2. **Record start time**: `patient.triage_start_time = self.current_time`
    - This writes down exactly when the nurse started seeing the patient
    - Like clocking in when the nurse starts the triage
3. **Calculate how long it will take**: `triage_time = self._generate_triage_time()`
    - This randomly decides how long this specific triage will take (3-6 minutes)
    - Like estimating "This patient will take about 4 minutes to check"
4. **Calculate finish time**: `triage_end_time = self.current_time + triage_time`
    - This calculates when the nurse will finish
    - Like saying "If it's 10:30 AM now and it takes 4 minutes, they'll finish at 10:34 AM"
5. **Schedule the completion**: `event = Event(triage_end_time, EventType.TRIAGE_COMPLETE, patient.id)`
    - This creates a reminder that says "At 10:34 AM, nurse will finish with patient #5"
    - Like putting an appointment in the calendar
6. **Add to schedule**: `heapq.heappush(self.events, event)`
    - This puts the completion event in the hospital's master schedule
    - Like adding it to the daily planner

Real-world example:

- It's 10:30 AM
- Nurse Sarah starts seeing patient #5
- The system randomly decides this triage will take 4 minutes
- It schedules a reminder for 10:34 AM: "Nurse Sarah will finish with patient #5"

Main Simulation Loop

```
def run(self):
    """Run the simulation"""
    print(f"Starting ER simulation...")
    print(f"Parameters: {self.num_triage_nurses} nurses, {self.num_doctors}
doctors")
    print(f"Shift duration: {self.shift_duration} minutes")

    while self.events and self.current_time < self.shift_duration:
        # Get next event
        event = heapq.heappop(self.events)
        self.current_time = event.time

        # Update statistics
        self._update_statistics()

        # Handle event
        if event.event_type == EventType.PATIENT_ARRIVAL:
            self._handle_patient_arrival(event.patient_id)
        elif event.event_type == EventType.TRIAGE_COMPLETE:
            self._handle_triage_complete(event.patient_id)
        elif event.event_type == EventType.CONSULTATION_COMPLETE:
            self._handle_consultation_complete(event.patient_id)

    print(f"Simulation completed at time {self.current_time:.2f} minutes")
    return self.get_results()
```

Simple Explanation: This is the main "brain" of the simulation - like a hospital manager who runs the entire ER shift. It's like a clock that ticks forward and handles everything that happens. Here's how it works:

1. **Start the shift**:
    - Print the setup information (how many nurses, doctors, shift length)
    - Like a manager starting their shift and reviewing the day's plan
2. **Main loop**: `while self.events and self.current_time < self.shift_duration:`
    - Keep going as long as there are events to handle AND the shift isn't over
    - Like saying "Keep working until there's nothing left to do or the shift ends"
3. **Get the next event**: `event = heapq.heappop(self.events)`
    - Take the next scheduled event from the list (the earliest one)
    - Like looking at your calendar and seeing "What's the next thing I need to do?"
4. **Update the clock**: `self.current_time = event.time`
    - Move the simulation time forward to when this event happens
    - Like saying "It's now 10:34 AM" (when the event is scheduled)
5. **Update statistics**: `self._update_statistics()`
    - Record what's happening right now (queue lengths, staff busy/free)

    - Like taking a snapshot of the current situation

6. **Handle the event**: The big decision tree
    - **If it's a patient arrival**: `self._handle_patient_arrival(event.patient_id)`
        - Like a receptionist greeting a new patient
    - **If it's triage completion**: `self._handle_triage_complete(event.patient_id)`
        - Like a nurse finishing with a patient and sending them to the doctor
    - **If it's consultation completion**: `self._handle_consultation_complete(event.patient_id)`
        - Like a doctor finishing with a patient and discharging them
7. **End the shift**: When the loop ends, print completion message and return results

Real-world example:

- It's 10:30 AM: "Next event: Patient #5 arrives"
- Handle patient arrival
- It's 10:34 AM: "Next event: Nurse finishes with patient #5"
- Handle triage completion
- It's 10:40 AM: "Next event: Patient #6 arrives"
- And so on until the shift ends...

📊 Performance Metrics

Wait Times

- **Triage Wait**: Time from arrival to triage start
- **Consultation Wait**: Time from triage completion to consultation start
- **Total Wait**: Sum of triage and consultation wait times

Simple Explanation: Think of these like waiting in line at different places:

Triage Wait: How long you wait before a nurse sees you

- Like waiting in line at the reception desk
- "I arrived at 10:00 AM, but the nurse didn't start seeing me until 10:05 AM"
- Wait time = 5 minutes

Consultation Wait: How long you wait before a doctor sees you

- Like waiting in the doctor's waiting room after the nurse is done
- "The nurse finished at 10:08 AM, but the doctor didn't start seeing me until 10:15 AM"
- Wait time = 7 minutes

Total Wait: All the waiting time combined

- "I waited 5 minutes for the nurse + 7 minutes for the doctor = 12 minutes total"

Service Times

- **Triage Service**: Time from triage start to completion

- **Consultation Service**: Time from consultation start to completion
- **Total Service**: Sum of triage and consultation service times

Simple Explanation: These are the actual treatment times - how long the medical staff spend with each patient:

Triage Service: How long the nurse spends checking the patient

- Like the actual time the nurse is examining you
- "The nurse started checking me at 10:05 AM and finished at 10:08 AM"
- Service time = 3 minutes

Consultation Service: How long the doctor spends with the patient

- Like the actual time the doctor is treating you
- "The doctor started seeing me at 10:15 AM and finished at 10:25 AM"
- Service time = 10 minutes

Total Service: All the treatment time combined

- "Nurse spent 3 minutes + doctor spent 10 minutes = 13 minutes total treatment time"

Queue Analysis

- **Triage Queue Length**: Number of patients waiting for triage
- **Consultation Queue Length**: Number of patients waiting for consultation
- **Maximum Queue Lengths**: Peak queue sizes during simulation

Simple Explanation: These tell you how crowded the waiting areas are:

Triage Queue Length: How many people are waiting to see a nurse

- Like counting people in the reception area
- "There are 3 people waiting to see a nurse right now"

Consultation Queue Length: How many people are waiting to see a doctor

- Like counting people in the doctor's waiting room
- "There are 5 people waiting to see a doctor right now"

Maximum Queue Lengths: The busiest the waiting areas ever got

- Like remembering "The reception area was most crowded with 8 people waiting"
- This helps understand peak demand periods

Resource Utilization

- **Triage Nurse Utilization**: Percentage of time nurses are busy
- **Doctor Utilization**: Percentage of time doctors are busy

Simple Explanation: These tell you how busy the medical staff are:

Triage Nurse Utilization: How much of their time nurses spend working vs. waiting

- Like saying "Nurses are busy 80% of the time, free 20% of the time"
- 80% utilization means nurses are working most of the time
- 20% utilization means nurses are often free and waiting for patients

Doctor Utilization: How much of their time doctors spend working vs. waiting

- Like saying "Doctors are busy 90% of the time, free 10% of the time"
- High utilization (90%) means doctors are very busy
- Low utilization (30%) means doctors often have free time

What this means:

- **High utilization** (80-90%): Staff are working hard, but patients might wait longer
- **Low utilization** (20-30%): Staff have free time, but might be overstaffed
- **Balanced utilization** (50-70%): Good balance between efficiency and patient care

Throughput

- **Patients per Hour**: Average patients completed per hour
- **Total Patients Served**: Number of patients who completed the entire process

Simple Explanation: These tell you how many patients the ER can handle:

Patients per Hour: How many patients complete their entire treatment per hour

- Like saying "The ER can treat 15 patients per hour on average"
- This measures the ER's capacity and efficiency
- Higher numbers mean the ER is more efficient

Total Patients Served: How many patients completed their full treatment during the shift

- Like saying "During the 8-hour shift, we treated 120 patients"
- This shows the total workload handled

What this means:

- **High throughput** (15+ patients/hour): ER is efficient and can handle many patients
- **Low throughput** (5-8 patients/hour): ER might be understaffed or inefficient
- **Balanced throughput** (10-15 patients/hour): Good balance of quality care and efficiency

🔍 Scenario Analysis

Staffing Configurations

The system tests various combinations:

```
staffing_configs = [
    (1, 2, "1 nurse, 2 doctors"),
    (1, 3, "1 nurse, 3 doctors"),
    (2, 2, "2 nurses, 2 doctors"),
    (2, 3, "2 nurses, 3 doctors"),
    (2, 4, "2 nurses, 4 doctors"),
```

```
    (3, 3, "3 nurses, 3 doctors"),
    (3, 4, "3 nurses, 4 doctors"),
    (3, 5, "3 nurses, 5 doctors")
]
```

Shift Duration Testing

```
durations = [240, 360, 480, 600, 720]  # 4, 6, 8, 10, 12 hours
```

Arrival Rate Scenarios

```
arrival_means = [2, 3, 4, 5, 6]  # minutes between arrivals
```

Service Time Variations

```
service_configs = [
    ((2, 4), (4, 12), "Faster service times"),
    ((3, 6), (5, 15), "Baseline service times"),
    ((4, 8), (6, 18), "Slower service times"),
    ((5, 10), (8, 20), "Much slower service times")
]
```

📈 Output Files

The comprehensive analysis generates:

Data Files

- `scenario_comparison.csv`: Detailed results table with all metrics

Visualization Charts

- `wait_times_comparison.png`: Wait time comparison across scenarios
- `utilization_comparison.png`: Resource utilization comparison
- `queue_lengths.png`: Queue length comparison

Example Results Table

Scenario	Total Patients	Completed Patients	Patients/Hour	Avg Total Wait (min)	Triage Utilization	Doctor Utilization

Scenario	Total Patients	Completed Patients	Patients/Hour	Avg Total Wait (min)	Triage Utilization	Doctor Utilization
Baseline (2 nurses, 3 doctors)	121	119	14.88	9.19	67.1%	87.2%
1 nurse, 2 doctors	98	85	10.62	10.56	86.6%	87.2%
2 nurses, 4 doctors	120	111	13.88	1.36	64.0%	65.7%

🎯 Key Findings

Baseline Performance (2 nurses, 3 doctors, 8-hour shift):

- **Average Total Wait Time**: 9.19 minutes
- **Patients per Hour**: 14.88 patients
- **Triage Nurse Utilization**: 67.1%
- **Doctor Utilization**: 87.2%
- **Maximum Queue Lengths**: 5 (triage), 10 (consultation)

Optimal Configurations:

- **Lowest Wait Time**: 2 nurses, 4 doctors (1.36 min)
- **Highest Throughput**: 2 nurses, 3 doctors (14.88 patients/hour)
- **Best Resource Balance**: 3 nurses, 3 doctors (42.4% nurse, 76.9% doctor utilization)

Bottleneck Analysis:

- **Primary Bottleneck**: Doctors (consultation wait time > triage wait time)
- **Recommendation**: Add more doctors to reduce consultation queues

🛠️ Technical Implementation Details

Discrete Event Simulation

- **Event-driven architecture** using priority queues
- **Realistic probability distributions** for arrivals and service times
- **Accurate tracking** of patient flow through the system

Data Structures

- **Patient**: Tracks individual patient journey and timing
- **Event**: Represents simulation events with chronological ordering
- **ERSimulation**: Main simulation engine with comprehensive statistics

Statistics Collection

- **Real-time tracking** of queue lengths and resource utilization
- **Comprehensive performance metrics** calculation
- **Scenario comparison** and analysis capabilities

📋 Requirements Fulfilled

☑ Queue-Based ER Process Simulation (30 marks)

- Complete two-stage ER workflow simulation
- Realistic patient arrival and service time distributions
- Proper queue management and resource allocation

☑ Performance Metrics Tracking

- Average patient wait times at each stage
- Resource utilization for nurses and doctors
- Queue lengths and throughput analysis
- Patients served per shift tracking

☑ Scenario Testing Capabilities

- Different staffing configurations (1-3 nurses, 2-5 doctors)
- Various shift durations (4-12 hours)
- Different arrival rates and service time distributions
- Comprehensive comparison and analysis

☑ Recommendations Generated

- Bottleneck identification and analysis
- Optimal staffing recommendations
- Data-driven improvement suggestions
- Resource allocation strategies

🎓 Educational Value

This project demonstrates:

- **Discrete Event Simulation** principles and implementation
- **Queue theory** applications in healthcare
- **Resource allocation** optimization techniques
- **Performance analysis** and metrics calculation
- **Data-driven decision making** in operational management
- **Python programming** with object-oriented design
- **Statistical analysis** and visualization techniques

🔮 Future Enhancements

The simulation can be extended with:

- **Patient priority levels** (urgent vs. non-urgent)

- **Different triage categories** and routing logic
- **Real-time monitoring** capabilities
- **Cost analysis** features
- **Multiple shift scheduling** and dynamic staffing
- **Advanced queue management** strategies

Example Usage

Running a Quick Test

```
python demo.py
```

Expected Output:

```
LIFECARE HOSPITAL ER SIMULATION DEMO
==================================================
Starting ER simulation...
Parameters: 2 nurses, 3 doctors
Shift duration: 480.0 minutes
Simulation completed at time 480.84 minutes

============================================================
ER SIMULATION RESULTS
============================================================
Total Patients Arrived: 121
Patients Completed: 119
Patients per Hour: 14.88

WAIT TIMES (minutes):
  Average Triage Wait: 1.41
  Average Consultation Wait: 7.80
  Average Total Wait: 9.19

SERVICE TIMES (minutes):
  Average Triage Service: 4.59
  Average Consultation Service: 10.12
  Average Total Service: 14.72

QUEUE LENGTHS:
  Maximum Triage Queue: 5
  Maximum Consultation Queue: 10

RESOURCE UTILIZATION:
  Average Triage Nurse Utilization: 67.08%
  Average Doctor Utilization: 87.22%
============================================================
```

How to Read These Results:

Patient Flow:

- **121 patients arrived** during the 8-hour shift
- **119 patients completed** their full treatment (2 left without finishing)
- **14.88 patients per hour** - the ER treated about 15 patients every hour

Wait Times (Patient Experience):

- **1.41 minutes** average wait for a nurse - patients usually get seen quickly by nurses
- **7.80 minutes** average wait for a doctor - patients wait longer for doctors
- **9.19 minutes** total wait time - from arrival to starting treatment

Service Times (Treatment Duration):

- **4.59 minutes** average nurse triage time - nurses spend about 4-5 minutes checking patients
- **10.12 minutes** average doctor consultation time - doctors spend about 10 minutes with patients
- **14.72 minutes** total treatment time - from start to finish of medical care

Queue Analysis (Crowding):

- **Maximum 5 patients** waiting for nurses at any time
- **Maximum 10 patients** waiting for doctors at any time
- The doctor waiting area gets more crowded than the nurse area

Staff Utilization (Efficiency):

- **67.08% nurse utilization** - nurses are busy about 2/3 of the time
- **87.22% doctor utilization** - doctors are busy almost 90% of the time
- Doctors are working harder than nurses in this setup

What This Means:

- The ER is running efficiently (treating 15 patients/hour)
- Patients don't wait too long (under 10 minutes total wait)
- Doctors are very busy (87% utilization) - might need more doctors
- Nurses have some free time (67% utilization) - might have enough nurses

Running Comprehensive Analysis

```
python scenario_analysis.py
```

This will generate:

- Detailed comparison tables
- Visualization charts
- CSV export of results
- Data-driven recommendations

License

This project is created for educational purposes as part of CS401 coursework.

🤝 Contributing

This simulation provides a solid foundation for healthcare operations research and can be extended with additional features like:

- Patient prioritization systems
- Multi-shift scheduling
- Cost-benefit analysis
- Real-time monitoring dashboards
- Integration with hospital information systems

Note: This simulation provides valuable insights for hospital administrators to optimize ER operations and improve patient satisfaction through reduced wait times and better resource utilization.