Project proposal for Queueing Theory class project

Requirements Analysis for a Queue Simulator

Introduction:

This project aims to develop a versatile queue simulator in Python to explore various queue types and their characteristics. The simulator will allow users to experiment with different queuing disciplines and analyze their impact on system performance.

General ideas
Add functionalities to find the best number of servers, or some other utilities
Model a simple real-world scenario through queuing theory and relative utilities

Calendar
1/12 - 6/12 build the simulations
6-12 - 9/12 build a GUI
write a report
9/12 - 11/12 Functionality idea: min # servers for obtaining a Ws < x

Simple simulator

Type of Queue	Simulation implementation	Formula-based implementation	Author
Single server M/M/1			ChatGPT
Multi-server M/M/C/K	NO	NO	Rachel
Unlimited servers M/M/C/C	NO	NO	Raff
Unlimited servers M/M/∞	NO	NO	Torstein

Single server M/M/1
IN arrival rate, service rate
OUT Ls, Lq, Ws, Wq, λe

Multi-server M/M/C/K
IN arrival rate, service rate, C, K
OUT Ls, Lq, Ws, Wq, rejection rate, λe

Unlimited servers M/M/C/C
IN arrival rate, service rate, C
OUT Ls, Lq, Ws, Wq, rejection rate, λe

Unlimited servers M/M/∞
IN arrival rate, service rate
OUT Ls, Lq, Ws, Wq, λe

ChatGPT simulation for a single server queue:

```
import random

class SingleServerQueueSimulator:
    def __init__(self, arrival_rate, service_rate, queue_capacity, simulation_time):
        self.arrival_rate = arrival_rate
        self.service_rate = service_rate
        self.queue_capacity = queue_capacity
        self.simulation_time = simulation_time

        self.clock = 0
        self.queue = []
        self.departure_time = float('inf')
        self.num_served_customers = 0
        self.total_wait_time = 0

    def generate_interarrival_time(self):
        return random.expovariate(self.arrival_rate)

    def generate_service_time(self):
        return random.expovariate(self.service_rate)

    def run_simulation(self):
        while self.clock < self.simulation_time:
            if self.clock < self.departure_time:
                self.clock = self.generate_interarrival_time() + self.clock
                if len(self.queue) < self.queue_capacity:
                    self.queue.append(self.clock)
                else:
                    print("Queue full. Customer lost at time:", self.clock)
            else:
                if self.queue:
                    arrival_time = self.queue.pop(0)
                    self.total_wait_time += self.clock - arrival_time
                    service_time = self.generate_service_time()
                    self.departure_time = self.clock + service_time
                    self.num_served_customers += 1
                else:
                    self.departure_time = float('inf')
            # Move the clock to the next event time (arrival or departure)
            self.clock = min(self.clock, self.departure_time)

    def get_simulation_results(self):
        average_wait_time = self.total_wait_time / self.num_served_customers if
self.num_served_customers > 0 else 0
        return {
            "Total served customers": self.num_served_customers,
            "Average wait time": average_wait_time
        }

# Example usage:
```

```
arrival_rate = 0.2  # Arrival rate
service_rate = 0.3  # Service rate
queue_capacity = 5   # Queue capacity
simulation_time = 100  # Simulation time

simulator = SingleServerQueueSimulator(arrival_rate, service_rate, queue_capacity,
simulation_time)
simulator.run_simulation()
results = simulator.get_simulation_results()

print("Simulation Results:")
for key, value in results.items():
    print(f"{key}: {value}")
```