Call Center Simulation: Discrete Event Simulation
Ryan Fallon

Abstract

This project dives into a call center simulation using discrete event simulation to model customer calls, agent availability, and call completions. The goal was to better understand how various factors—like the number of agents, call arrival rates, and average call durations—impact performance metrics, including total calls, abandoned calls, and agent utilization. I experimented with multiple scenarios to uncover insights into managing call center efficiency while balancing resource use and customer satisfaction. All results, including CSV outputs and visualizations, are available in this GitHub repository.

Introduction

Call centers are at the heart of many organizations' customer service operations, and they present an ongoing challenge: how do you make sure customer wait times are reasonable while keeping your staff utilized effectively? This project uses a simulation to explore that balancing act by modeling a hypothetical call center. Using Go, I built a program to test different staffing and operational configurations, aiming to find strategies that minimize abandoned calls and keep agents busy without overloading them.

Problem Definition

Picture this: customers are calling in, but you've only got so many agents available. Some customers get help right away, others wait in line, and a few might hang up if they've been on hold too long. That's the essence of this problem.

The key questions I tackled were:

1. How many agents should be working to minimize abandoned calls without overstaffing?
2. What happens to customer wait times when call volumes spike unexpectedly?
3. How can I balance agent utilization with keeping customers happy?

By running simulations of these scenarios, I aimed to find patterns that would help answer these questions.

Simulation Design

The simulation models key components of a call center:

- **Agents**: Staff available to answer calls.
- **Calls**: Events that either get answered, wait in line, or are abandoned.
- **Event Queue**: A system that tracks when calls arrive, complete, or are abandoned.

Inputs

The program allows flexibility in testing various scenarios:

1. **Number of Agents**: Between 1 and 10 agents.
2. **Simulation Time**: Fixed at 1,440 minutes (a 24-hour day).
3. **Maximum Wait Time**: 5, 10, or 15 minutes before a call is considered abandoned.
4. **Call Arrival Rate (Lambda)**: Between 0.5 and 2 calls per minute.
5. **Average Call Duration**: 3, 5, 7, or 9 minutes per call.

Metrics Captured

1. **Total Calls**: The total number of calls during the simulation.

2. **Abandoned Calls**: Calls that weren't answered within the maximum wait time.
3. **Agent Utilization**: The percentage of time agents spent handling calls during the day.

Results and Interpretation

After running the simulation across a variety of configurations, here's what stood out:

- **Adding Agents Helps**: More agents reduced abandoned calls but also lowered utilization. At some point, you hit diminishing returns.
- **Call Arrival Rates Matter**: Higher arrival rates increased abandoned calls, even with additional agents.
- **Longer Wait Times Are a Band-Aid**: Increasing the maximum wait time helped slightly, but it didn't fully solve the issue when arrival rates were too high.
- **Call Durations Add Complexity**: Longer calls mean fewer customers get through, even if you have more agents.

Full metrics are available in the CSV output files in the GitHub repository.

Management Recommendations

Based on these findings:

1. **Be Strategic with Staffing**: You don't always need the maximum number of agents. Adjusting staff during peak hours is more effective than keeping everyone on all the time.

2. **Consider a Callback System**: This could reduce customer frustration while keeping abandonment rates low.
3. **Monitor Agent Utilization**: Keeping agents around 80-85% utilization seems like the sweet spot for balancing efficiency and workload.

Conclusion

This simulation highlighted how discrete event modeling can give practical insights into real-world problems. By running these scenarios, I identified patterns that could help manage call center operations more effectively. If I were to extend this project, I'd look at integrating priority systems for certain calls (like VIP customers) or testing alternative programming environments (i.e. Python's SimPy) to compare approaches.

All code, output files, and instructions for running the simulation are available in the linked GitHub repository. You'll also find a README with tips on sorting the results in Excel to quickly identify the best-performing configurations.