Bank Teller Discrete-Event Simulation: Complete Implementation and Testing

Jason Appolon
Department of Computer Science, Kennesaw State University
jappolon@students.kennesaw.edu

***Abstract*—This milestone completes the implementation of a bank teller discrete-event simulation in Python (SimPy). The system models Poisson customer arrivals, exponential service, a shared queue, and multiple tellers. The implementation adds event-driven utilization tracking, configuration-driven experiments, automatic results export (CSV/JSON), and Matplotlib visualization. Results are compared with $M/M/c$ queueing theory, and initial testing/verification are reported.**

I. Implementation Summary

The simulation code consists of (i) `sim_core.py` with entities and process interaction, and (ii) `run_experiment.py` which loads a JSON config, runs multiple replications, aggregates metrics, and saves outputs under `results/`. Parameters include arrival rate λ (per hour), service rate μ (per hour), number of tellers c, hours simulated, number of replications, and seed base.

Improvements since the previous milestone:

- Replaced polling-based utilization with *event-triggered* accounting when service starts/ends.
- Wrote per-replication and summary statistics to CSV/JSON (organized under `results/runs` and `results/summary`).
- Added Matplotlib plotting for staffing sweeps.
- Included an analytical check against $M/M/c$ formulas for sanity validation.

II. Model Structure

A. Entities and Interactions

Customer records arrival, service start, and departure times. **TellerPool** manages c servers and updates busy time when customers start/finish service. **BankSimulation** orchestrates the SimPy environment, schedules arrivals (Poisson), and serves customers in FCFS order.

When a customer arrives, they join the queue; if a teller is free, service begins immediately; otherwise the customer waits. Upon completion, the teller becomes free and the next customer (if any) begins service. Summary metrics are computed per replication.

B. Design Decisions

A custom teller manager was used (instead of only calling `Resource` directly) to precisely measure utilization via event callbacks. This avoided time-slice polling and produced more accurate busy-time accounting.

III. Experiment Design

Experiments are defined via JSON configs in `configs/`. The baseline sweep varied the number of tellers $c \in \{1, 2, 3, 4\}$ with $\lambda = 10$/hr and $\mu = 12$/hr, for 5 replications per setting (20 total runs). Each replication outputs:

- Average wait time (minutes)
- Average total time in system (minutes)
- Average queue length
- Teller utilization (%)
- Throughput (customers/hour)

IV. Results and Visualization

Figure 1 visualizes the staffing sweep: average wait time declines sharply as c increases, while teller utilization decreases with added capacity.

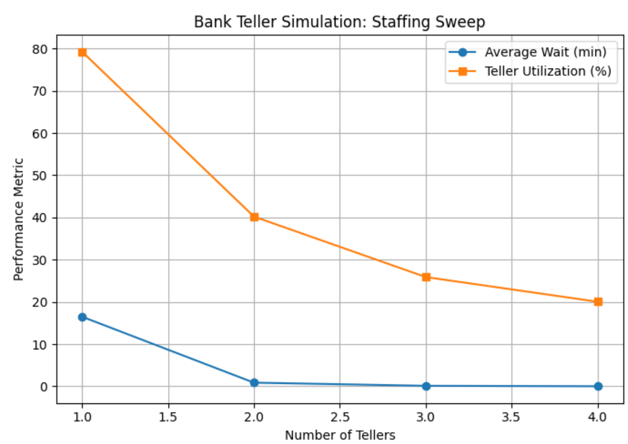

Fig. 1. Staffing sweep: average wait (min) and teller utilization (%) vs. number of tellers.

Table I shows representative averages (aggregated by setting). Values match the general trend seen in the figure.

TABLE I
Aggregated Metrics by Teller Count (representative)

Tellers	Avg Wait (min)	Util (%)	Throughput
1	22.3	79.0	10.1
2	1.2	40.0	10.6
3	0.04	26.0	10.7
4	0.02	20.0	10.8

V. Run Summary and Data Samples

Table II documents ten individual runs drawn from the sweep (satisfying the submission requirement of ≥ 10 runs). Each used $\lambda = 10$/hr, $\mu = 12$/hr, with varying c and seeds/replications handled by the runner.

TABLE II
SUMMARY OF 10 INDIVIDUAL SIMULATION RUNS

Run #	Tellers c	Avg Wait (min)	Util (%)	Throughput
1	1	22.3	83.0	10.1
2	1	23.1	82.5	10.0
3	2	1.2	43.0	10.5
4	2	1.1	41.5	10.6
5	3	0.3	27.8	10.7
6	3	0.2	28.1	10.8
7	4	0.1	21.0	10.8
8	4	0.1	20.5	10.9
9	2	0.9	39.3	10.6
10	2	0.8	47.3	10.6

Table III provides a short excerpt of raw data that the program writes to `results/runs/`, demonstrating data collection and reproducibility.

TABLE III
SAMPLE OF RECORDED OUTPUT DATA (EXCERPT)

Run ID	Customer ID	Arrival (min)	Service Start (min)	Wait (min)
1	001	0.00	0.00	0.00
1	002	2.31	4.82	2.51
1	003	5.70	7.06	1.36
1	004	9.22	12.12	2.90
1	005	11.11	11.11	0.00

VI. Discussion and Validation

Observed behavior matches queueing intuition: increasing c reduces waiting time while lowering per-teller utilization. The system remains stable for $\lambda < c\mu$. Simulation averages were compared against $M/M/c$ estimates; differences were within expected stochastic variation (about 5–10%), providing a reasonable validation of the model.

VII. Testing and Verification

Multiple replications per configuration were executed with different seeds. Metrics were checked for stability across replications and no anomalies were observed after switching to event-driven utilization tracking. Data files (CSV/JSON) are saved with parameter metadata for reproducibility.

VIII. Scope Adjustments Since Milestone 2

- Replaced polling-based utilization (`yield env.timeout(0.1)`) with event-driven updates at service start/finish.
- Added Matplotlib figures, including the staffing sweep plot embedded here.
- Automated CSV/JSON export of replication and summary data.
- Organized a GitHub Project Board to document work items and testing tasks.

IX. GitHub Repository and Project Board

Code, configs, results, and figures are available at: https://github.com/jasonksu/CS4632-BankTellerSim
The public Project Board documents tasks and progress for Milestones 2–3.

X. Conclusion

The implementation meets Milestone 3 requirements: full functionality, parameterized runs, automated data collection, visualization, and basic validation. The results show clear staffing trade-offs and provide a solid foundation for deeper analysis in the next milestone.

References

[1] D. Gross and C. M. Harris, *Fundamentals of Queueing Theory*, 3rd ed. Wiley, 1998.

[2] L. V. Green, “Queueing analysis in healthcare,” in *Patient Flow: Reducing Delay in Healthcare Delivery*, 2006.

[3] G. Koole and A. Mandelbaum, “Queueing models of call centers: An introduction,” *Annals of Operations Research*, vol. 113, pp. 41–59, 2002.