Discrete Event Simulation (DES): Airplane Unloading

Henry Roeth

February 16, 2026

Jupyter Notebook Link

1 Overview

This report documents a discrete event simulation (DES) model for unloading passengers from an airplane using `simpy`. The goal is to estimate the average time to fully unload a full flight as a function of the number of rows. The basic configuration uses:

- Seats per row: 4
- Rows tested: $25, 30, 35, 40, 45$
- All seats filled; passengers exit through the front exit only

A full implementation of the model class structures is provided in the Code Appendix for reference.

2 Part 1: General Unloading of Flight

2.1 Model (Basic)

Entities:

- **Passenger**: Has row index, seat label, and a walking speed
- **Plane**: Contains a full list of passengers and processes them leaving the aircraft

Stochastic components: Passenger walking speed is treated as random, sampled independently for each passenger:

$$v \sim \text{Uniform}(v_{\min}, v_{\max})$$

Service-time logic (basic): A passenger's time to reach the exit is modeled as proportional to row distance and inversely proportional to speed. In the implementation, time is computed using a form like:

$$t_{\text{walk}}(r, v) = \frac{(3r)}{v}$$

where r is the passenger's row number and v is speed. The simulation records the total time until the final passenger exits.

Queueing / blocking assumptions: In the basic model, unloading occurs one passenger at a time through a single process; passengers are processed sequentially (single-server abstraction for the front exit).

2.2 Simulation (Basic)

The basic DES is implemented in Python using `simpy`. For each plane size (rows = 25 to 45 in steps of 5), the simulation is repeated N times to estimate:

- Mean unloading time
- Standard deviation of unloading time

Implementation Notes:

- The environment is reset each run
- A new full plane is created each run with randomized passenger speeds
- The simulation ends when all passengers have been processed and the final completion time is recorded

Implementation (Basic Model):

Listing 1: Basic Simulation Runner

```
class UnloadSimRunner:
        def __init__(self, num_rows_options, num_seats_per_row: int, num_runs: int):
                self.num_rows_options = num_rows_options
                self.num_seats_per_row = num_seats_per_row
                self.num_runs = num_runs

                self.unload_results = []
                self.num_rows_options_step = 0

        def _run_unload_sim(self, special: bool):
                run_times = []
                num_rows = self.num_rows_options[self.num_rows_options_step]
                num_passengers = num_rows * self.num_seats_per_row
                if self.num_rows_options_step < len(self.num_rows_options):
                        self.num_rows_options_step += 1
                if special:
                        print(f"Running {num_passengers}-seat ({num_rows} rows, {self.num_seats_per_row} seats
                        per row) special airplane unloading simulation...")
                else:
                        print(f"Running {num_passengers}-seat ({num_rows} rows, {self.num_seats_per_row} seats
                        per row) airplane unloading simulation...")
                for i in range(self.num_runs):
                        env = simpy.Environment()
                        plane = Plane(env, i, num_rows, self.num_seats_per_row, 0, special)
                        # plane.print_passenger_data()
                        if special:
                        env.process(plane.unload_airplane_special())
                        else:
                        env.process(plane.unload_airplane())
                        env.run()
                        run_times.append(plane.get_process_time())

                import statistics
                avg_unload_time = statistics.mean(run_times)
                std_dev = statistics.stdev(run_times)

                self.unload_results.append((avg_unload_time, std_dev))
                print(f"Average unload time after {self.num_runs} runs "
                        f"for a {num_passengers}-seat airplane was "
                        f"{avg_unload_time:.3f} minutes "
                        f"(std dev = {std_dev:.3f})\n")

        def run_unload_sim(self, special: bool):
                for i in range(len(self.num_rows_options)):
                        self._run_unload_sim(special)

        def plot_results(self, plt, sepcial):
                means = []
                stds = []
                for result in self.unload_results:
                        means.append(result[0])
                        stds.append(result[1])
                        plt.errorbar(self.num_rows_options, means, yerr=stds, fmt='ro')
                        plt.xlabel(f"Number of Rows ({self.num_seats_per_row} Seats/Row)")
                        plt.ylabel("Time (Minutes)")
                if special:
                        title_string = "Special"
                else:
                        title_string = "Basic"
                plt.title(f"Average Airplane Unloading Times After {self.num_runs} Runs ({title_string})")
                plt.show()

import matplotlib.pyplot as plt

num_rows_options = [25, 30, 35, 40, 45]
num_seats_per_row: int = 4
num_runs: int = 30
special = False

unload_sim_runner = UnloadSimRunner(num_rows_options, num_seats_per_row, num_runs)
unload_sim_runner.run_unload_sim(special)
unload_sim_runner.plot_results(plt, special)
```

2.3 Results (Basic)

Table 1 reports the mean and standard deviation of unloading time for each row count. Figure 1 visualizes the trend.

Table 1: Basic unloading results (4 seats/row).

Rows	Seats	Mean time (min)	Std dev (min)
25	100	*15.087*	*0.388*
30	120	*21.473*	*0.416*
35	140	*29.183*	*0.554*
40	160	*37.862*	*0.671*
45	180	*47.726*	*0.970*

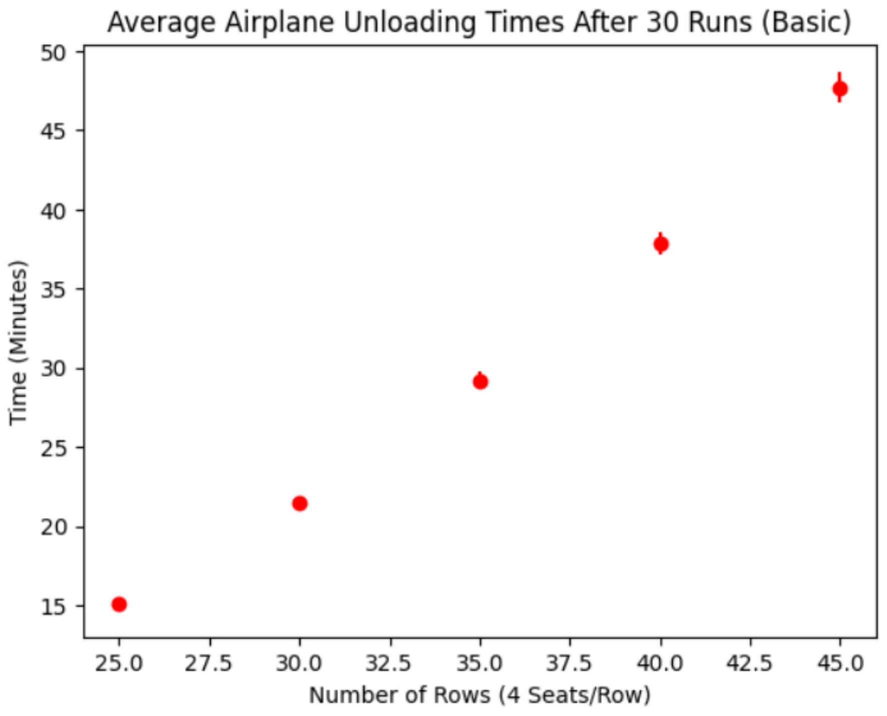

Figure 1: Basic mean unloading time vs. number of rows with error bars

Analysis: As the number of rows increases, the time it takes to unload increases in a linear pattern. There is very little variability for each run of each simulation, indicating a very consistent trend.

3 Part 2: Special Unloading of Flight

3.1 Model (Special Feature)

The special unloading model modifies the basic by adding realistic features that can increase unloading time. In the implementation, these include:

- **Carry-on baggage delay**: Passengers may incur an additional delay to retrieve a bag
- **Seat-interference delay**: Passengers in window/middle seats may be blocked by passengers closer to the aisle, creating additional random interference time

One possible conceptual decomposition is:

$$t_{\text{total}} = t_{\text{walk}} + t_{\text{bag}} + t_{\text{interference}}$$

where t_{bag} and $t_{\text{interference}}$ are stochastic and may depend on seat position and a probability of interference events.

3.2 Simulation (Special)

The simulation procedure mirrors Part 1, but uses the special unloading process. The basic and special simulations are kept as separate paths (do not overwrite the basic logic).

Implementation (Special Model).

Listing 2: Special Unloading Logic:

```
special = True

unload_sim_runner_special = UnloadSimRunner(num_rows_options, num_seats_per_row, num_runs)
unload_sim_runner_special.run_unload_sim(special)
unload_sim_runner_special.plot_results(plt, special)
```

3.3 Results (Special Feature + Comparison)

Table 2: Special unloading results (4 seats/row).

Rows	Seats	Mean time (min)	Std dev (min)
25	100	*23.052*	*5.567*
30	120	*35.793*	*4.546*
35	140	*40.828*	*8.334*
40	160	*56.201*	*7.369*
45	180	*65.281*	*9.775*

Table 3: Basic vs. Special comparison (mean times).

Rows	Basic mean (min)	Special mean (min)	Difference (min)
25	*15.087*	*23.052*	*7.965*
30	*21.473*	*35.793*	*14.320*
35	*29.183*	*40.828*	*11.645*
40	*37.862*	*56.201*	*18.339*
45	*47.726*	*65.281*	*17.555*

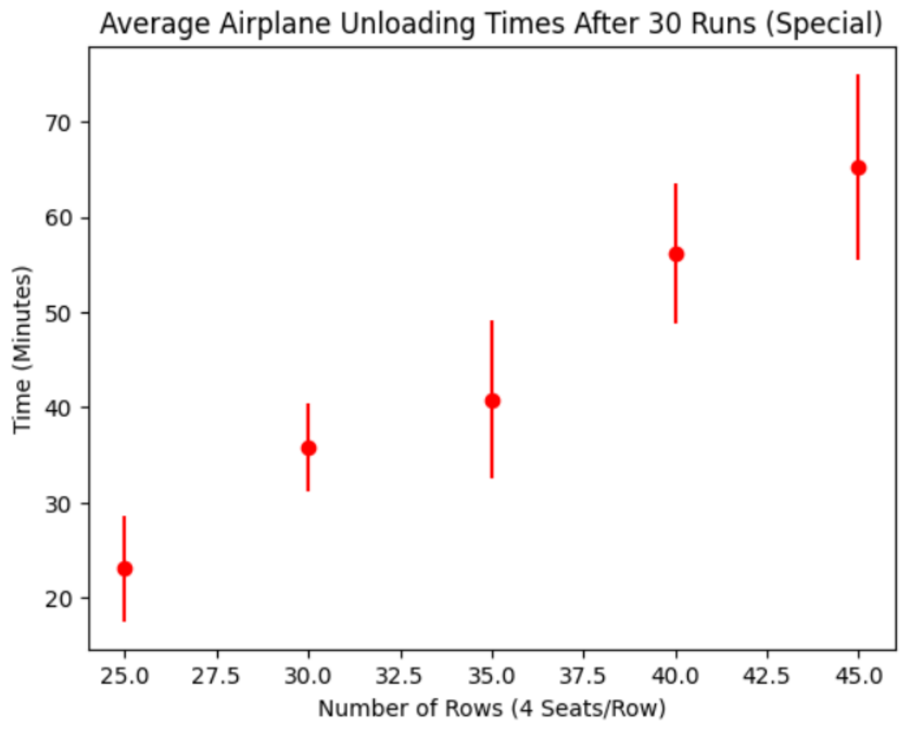

Figure 2: Special mean unloading time vs. number of rows with error bars

Analysis: The special model takes longer because passengers do more than just walk to the exit. Some must retrieve bags and others are blocked by people closer to the aisle, which creates extra delays. As the plane gets larger there are more passengers, so these delays happen more often and the difference between the special and basic models becomes bigger. The special model also has much higher variability because the delays are random. In the basic model the time mainly depends on distance, so results are consistent, but in the special model some runs have many delays while others have few, leading to a larger spread in times.

4 Code Appendix: Model Classes

Listing 3: Passenger and Plane Classes

```
import simpy
import string
import random

class Passenger:
        def __init__(self, id, row: int, seat: str, speed: float, has_bag: bool = False, bag_time: float = 0):
                self.id = id
                self.row = row
                self.seat = seat
                self.speed = speed
                self.bag_time = bag_time
                self.unloaded = False
                self.process_time: float = 0.0

        def get_has_bag(self):
                return self.has_bag
        def get_bag_time(self):
                return self.bag_time

        def get_data(self):
        """Method to get the passenger data.

        Returns:
        List: Tuple of the passenger data.
        """
                return [self.id, self.row, self.seat, self.speed, self.unloaded]

        def get_id(self):
        """Method to get the passenger id.

        Returns:
        int: The passenger id.
        """
                return self.id

        def get_row(self):
        """Method to get the passenger row.

        Returns:
        int: The passenger row.
        """
                return self.row

        def get_seat(self):
        """Method to get the passenger seat.

        Returns:
        str: The passenger seat.
        """
                return self.seat

        def get_speed(self):
        """Method to get the passenger speed.

        Returns:
        float: The passenger speed.
        """
                return self.speed

        def set_process_time(self, process_time: float):
        """Method to set the passenger process time.

        Args:
        process_time (float): The process time to set for the passenger.
        """
                self.process_time = process_time

        def get_process_time(self):
        """Method to get the passenger process time.

        Returns:
        float: The passenger process time.
        """
                return self.process_time

class Plane:
        def __init__(self, env: simpy.Environment, id, num_rows: int, num_seats_per_row: int, sim_type: int,
        special: bool, min_passenger_speed: float = 3, max_passenger_speed: float = 6):
                self.env = env
                self.id = id
                self.num_rows = num_rows
                self.num_seats_per_row = num_seats_per_row
                self.min_passenger_speed = min_passenger_speed
                self.max_passenger_speed = max_passenger_speed
```

```
            self.sim_type = sim_type
            self.special = special
            self.num_passengers = self.num_rows * self.num_seats_per_row
            self.process_time: float = 0.0

            self.loaded_passengers = []
            self.unloaded_passengers = []

            self._setup_plane(self.loaded_passengers, self.unloaded_passengers, self.sim_type,
            self.special)

    def _setup_plane(self, loaded_passengers, unloaded_passengers, sim_type, special):
    """Internal method to set up the airplane for either loading or unloading.

    Args:
    loaded_passengers (list[Passenger]): List to be populated with loaded passengers.
    unloaded_passengers (list[Passenger]): List to be populated with unloaded passengers.
    sim_type (int): 0 for unloading, 1 for loading.

    Returns:
    None: Modifies the passenger lists in-place.

    Raises:
    ValueError: If `sim_type` is not 0 or 1.
    """
            if self.special != False and self.special != True:
                    raise ValueError("'special' must be either True (special) or False (not special)")
            if self.special:
                    has_bag = random.random() < 0.75
            if has_bag:
                    bag_time = random.triangular(4, 10, 6)
            else:
                    bag_time = 0.0

            uppercase_alphabet = list(string.ascii_uppercase)
            seat_labels = []
            for i in range(self.num_seats_per_row):
                    seat_labels.append(uppercase_alphabet[i])

            row = 1
            seat = seat_labels[0]
            seat_step = 0
            for i in range(self.num_passengers):
                    seat = seat_labels[seat_step]

                    # Create new passenger (speed is a random float between min_passenger_speed and
max_passenger_speed)
                    if self.special:
                            new_passenger = Passenger(i, row, seat, random.uniform(self.
min_passenger_speed, self.max_passenger_speed), has_bag, bag_time)
                    else:
                            new_passenger = Passenger(i, row, seat, random.uniform(self.
min_passenger_speed, self.max_passenger_speed))
                    if sim_type == 0:
                            loaded_passengers.append(new_passenger)
                    elif sim_type == 1:
                            unloaded_passengers.append(new_passenger)
                    else:
                            raise ValueError("'sim_type' must be either 0 (unload) or 1 (load)")

                    if seat == seat_labels[-1]:
                            row += 1
                            seat_step = 0
                    else:
                            seat_step += 1

    def unload_passenger(self, passenger):
    """Method to unload a passenger from the airplane.
```

```
Args:
passenger (Passenger): Passenger to be unloaded.

Returns:
Passenger: The passenger that was processed.
"""
        self.idle = False
        time = passenger.get_row() * 3 / passenger.get_speed()
        yield self.env.timeout(delay=time)
        passenger.set_process_time(self.env.now / 60)
        # print(f"Unloading passenger {passenger.id} at {self.env.now/60} minutes.")
        passenger.unloaded = True
        self.idle = True
        return passenger

def unload_airplane(self):
"""Method to unload all loaded passengers from the airplane.

Returns:
None: Modifies the passenger lists and updates unloading results.
"""
        while len(self.loaded_passengers) > 0:
                p = self.loaded_passengers[0]
                yield self.env.process(self.unload_passenger(p))
                self.loaded_passengers.remove(p)
                self.unloaded_passengers.append(p)
                if len(self.loaded_passengers) == 0:
                self.process_time = p.get_process_time()
        # print(f"Unloaded airplane {self.id} in {self.process_time} minutes")

def unload_passenger_special(self, passenger):
        self.idle = False
        time = passenger.get_row() * 3 / passenger.get_speed()
        bag_time = passenger.get_bag_time()
        blockers = 0
        if passenger.get_seat() in ("A", "F"):
                blockers = 2
        elif passenger.get_seat() in ("B", "E"):
                blockers = 1
        interference = 0.0
        if blockers > 0 and random.random() < 0.6:
                interference = random.triangular(1, 5, 2) * blockers
        total_time = time + bag_time + interference
        yield self.env.timeout(total_time)
        passenger.set_process_time(self.env.now / 60)
        passenger.unloaded = True
        self.idle = True
        return passenger

def unload_airplane_special(self):
        while len(self.loaded_passengers) > 0:
                p = self.loaded_passengers[0]
                yield self.env.process(self.unload_passenger_special(p))
                self.loaded_passengers.remove(p)
                self.unloaded_passengers.append(p)
                if len(self.loaded_passengers) == 0:
                        self.process_time = p.get_process_time()

def load_passenger(self, passenger):
        pass # future logic

def load_airplane(self):
        pass # future logic

def get_process_time(self):
"""Method to get the time it took to process the plane.
```

```
        Returns:
        float: The total process time.
        """
                return self.process_time

        def print_passenger_data(self):
        """Method to print all of the passenger data.

        Returns:
        None: Iteratively prints the passenger data.
        """
                print("Number of passengers on plane: " + str(len(self.loaded_passengers)))
                for i in range(len(self.loaded_passengers)):
                        passenger = self.loaded_passengers[i]
                        print(passenger.get_data())
```

5 Summary

This project used a discrete event simulation to estimate how long it takes to unload an airplane. The basic model assumed passengers simply walk to the exit, while the special model added bag retrieval and seat interference delays. The results showed that unloading time increases with plane size in both models, but the special model takes much longer and is less consistent because random delays occur more often. This demonstrates how small passenger interactions can significantly affect total system time.

In the future, I would like to implement a loading simulation to compare boarding strategies. The loading model would reverse the process so passengers enter the plane instead of leaving it. I would assign each passenger a seat and simulate them walking down the aisle, storing luggage, and waiting if someone ahead blocks the aisle. I would then test different boarding methods such as random boarding, back to front boarding, and window middle aisle boarding. The goal would be to measure total boarding time and determine which strategy minimizes delays and congestion.