

```
language=Python, basicstyle=, stringstyle=, otherkeywords=0,1,2,3,4,5,6,7,8,9,  
morekeywords=TRUE, FALSE, deletekeywords=data,frame,length,as,character,  
keywordstyle=, commentstyle=,
```



Erasmus University Rotterdam

TINBERGEN INSTITUTE / BUSINESS DATA SCIENCE

Teachers: Bernd Heidergott, Ad Ridder

Aircraft Boarding Strategies Using Discrete Event Simulation

Abstract

As the need of optimal boarding strategies which can substantially reduce the amount of delays is becoming more important, more research studies have aimed to investigate the most optimal boarding strategy by means of discrete event simulation. The objective of this report is to simulate an aircraft boarding strategy based on single door boarding which minimises the boarding time based on discrete simulation where the delay for each passenger is drawn from a uniform distribution whereas the arrival of each passenger is modeled through an exponential distribution

Keywords: JEL C15, C63, discrete event simulation (DES)

Author:

David Gyaraki
Lavin Pallas
Julius de Clercq

Student ID:

582340
581619
582360

April 2, 2023

Contents

1	Introduction	1
2	Conceptual model	2
2.1	Events and states	3
2.2	Boarding strategies	5
3	Simulation program	7
4	Validation and Verification	9
5	Results	9
6	Discussion	10
7	Conclusion	10
8	Appendix	11
9	Code	18

1 Introduction

The purpose of this report is the simulation of an aircraft boarding system and the evaluation of the system using different boarding strategies based on the paper of [Jafer & Mi \(2017\)](#) with the use of discrete event simulation. It involves events occurring at specific points in time, and the system state changing as a result of each event. In such simulations, time is divided into discrete intervals and the system state is updated only at the occurrence of an event. These events include the arrival of passengers, the start or end of a process in general, or any other event that can be modeled as discrete (either by nature or by simplifying the system into a discrete event using assumptions) and occurring at a specific time.

Literature suggests that the most efficient boarding strategies consist of the Window-Middle-Aisle and its variant, the Reverse Pyramid boarding strategy whereas some of the less efficient strategies have been found to be the Back-to-Front strategy ([Jafer & Mi, 2017](#)). It has been found that as congestion increases, the order among boarding groups becomes less relevant and the various sizes of the groups are the only parameters that counts. The total boarding time for the random boarding strategy is computed using the Euler-Lagrange equation for different time step values k as the joint distribution of the queue is uniform ([Bachmat et al., 2009a](#)). Nevertheless, according to [Bachmat et al. \(2009b\)](#), the dynamically optimized boarding strategy has been found to be more efficient compared to the window middle aisle seat strategy, which involves both allowing passengers to board in cliques, such as families with young children for instance, as well as depending on their seat. The most essential parameters considered for simulation consist of the passenger's walking speed, the luggage storage time, the time to clear a seat interference, and the passenger inter-arrival time. The seat interference potentially involves 0, 1 or 2 seats depending on the number of passengers (and the specific aircraft models), and this applies for this simulation study as well.

According to [Ferrari & Nagel \(2005\)](#), the half block boarding strategy consisting of only one group was the most efficient, as well as alternating rows which eventually was found to be too complicated however. The most inefficient strategy was proven to be the back to front boarding strategy as well as block boarding with many sub passenger groups. A deterministic process whereby each passenger can move one cell or row forward per time step has been implemented. In addition, a different number of disturbances was introduced consisting of early and late passengers, the aircraft occupancy as well as the aircraft dimensions, which play a role in the quality of the boarding strategy. The seating model which determines the boarding time is determined by the sum of the total time for seating plus two times the time to get from seat into aisle and back multiplied by the number of obstructing passengers who are already seated.

Another research study by [Van Landeghem & Beuselinck \(2002\)](#) who were among the first scholars to study aircraft boarding strategies, found that calling all passengers individually by their row and seat number is the fastest approach, yet the riskiest and most complicated method if anything went to go wrong, which can lead to confusion and a tremendous amount of delays. A more realistic approach is random boarding which leads to the overall reduction of total boarding time, nevertheless it results individual boarding time delays. The boarding time was calculated by taking the sum of the seating time of all passengers including the delays of each individual, the amount of time necessary to store luggage, delays within the same class caused by individuals attaining their seat and delay among different classes caused by interference between consecutive classes. In this

study a triangular distribution for passenger movement was assumed.

The rest of this paper follows as: Section 2 describes the conceptual model of our system used in the simulations as well as the events and different boarding strategies used; Section 3 briefly describes the programs, functions and random distributions used for the simulation; Section 4 touches upon the use of parameters and their practical viability compared to other papers; Section 5 summarizes the simulation results using the evaluation metrics; Section 6 elaborates on the results and discusses the optimal boarding strategies in light of our result and finally, Section 7 concludes.

2 Conceptual model

This section will first briefly describe the conceptual model we use for Cellular DES of an aircraft boarding system, and then elaborate a bit more on the system states, events and transformations that can occur within our system, and then describe the different boarding strategies used for testing this system. To develop the conceptual model of our system, we made a number of different assumptions, sometimes following [Jafer & Mi \(2017\)](#) and in other cases making our own assumptions of the system. These assumptions help us construct a concept of an aircraft and the boarding of this aircraft which we can then test.

For aircraft type, we picked an Airbus 320 with the arrangement of seats applied by WizzAir¹. Therefore we assume an aircraft with symmetric sides, 30 rows of equal length seats (no business class), 3 seats on each side of the aisle. The draft of the model we follow can be found also in the Appendix in the boarding strategy visualization graphs (such as Figure 2). The equal seat length is an assumption which does not hold in reality since emergency exit rows have to be longer, but we can simplify this to avoid confusion over row and aisle cell length heterogeneity. The general cell measure in our system is one seat, which is 28 inches long and 18 inches wide¹, which corresponds to 71.12cm cell length and 45.72cm cell width. Similarly to [Jafer & Mi \(2017\)](#), we assume that the aisle space in a row corresponds to a single seat cell equivalent, i.e. the aisle is 71.12cm long between two seats and 45.72cm wide. In addition to the aisle among the rows, we assume that the entrance to the rows of seat consists of one more aisle cell and 4 aisle cells which the passengers have to traverse on a cross-way direction. This means that the passenger (from entering the aircraft) needs to cover $4 \times 45.72\text{cm}$ and once 71.12cm to reach the first row.

Furthermore, we assume that the airline can only conduct front row boarding in order to reduce complexity of front-rear boarding and questions arising from this, such as the problem of non-complying passengers, who board in the front even though they are assigned to rear and vice-versa. This means that the aircraft can only be boarded from the front, for example the airplane is directly connected to the airport terminal. Furthermore, as Figure 2 also shows, apart from the aisle cells and the seats, all other space is assumed to be inaccessible for the passengers, i.e. they cannot walk towards the cockpit, they cannot walk past the last row, etc. One cell may only contain one passenger at any given time with one exception which will be mentioned later. The aisle cell is said to contain the passenger from the time the passenger passes the boundary of the aisle and the cell is freed up again when the passenger crosses the other boundary. Therefore we assume that passengers may not pass each other on the aisle, so if a passenger stops

¹The aircraft seat layout and parameters can be found at SeatGuru.

on a certain aisle cell, another second passenger behind needs to stop on the previous aisle cell in case they catch up and wait until the first passenger progresses or sits down.

Furthermore, we assume that from the moment a passenger enters the aircraft, they progress in a single direction on the aisle (no back-tracking) and goes to their row (walking continuously or stopping to wait for another passenger). Passengers are all assumed to know their seat exactly and cannot choose a different seat. Once the passenger reaches their row, they stop and first stash their luggage (in the overhead compartment or under the seat) if they have any. Since a passenger can stash the luggage both under the seat and overhead, we assume that the airline made sure before the boarding that there is enough space for all passengers' luggage where they are sitting (or within reach) and there is no need for the passengers to move back and forth on the aisle to find space for their luggage. After the luggage (if the passenger has any) is stowed away, then begins the seating procedure for that passenger (i.e. the passenger has to stow their luggage before sitting down). However, it is possible that a passenger arrives to their row to find that someone is already sitting there, blocking their way to the seat.

In this case, make the following assumptions for the system. In order for any passenger to reach their seat, first any potential passenger blocking the access needs to stand up (the arriving passenger cannot squeeze past the already sitting passengers). For example, if a passenger has the window seat, but the row they are sitting on already has the middle and aisle seat occupied, then first the aisle seat passenger needs to stand up, then the middle seat passenger needs to stand up, then the passenger who arrived can take their seat, then the middle seat passenger can sit back and finally the aisle seat passenger can take their place again. In the case of standing up however, we assume that the aisle cell on the same row can host multiple people, by the one or two people standing up also stand in this cell, or leaning towards the other side of the row, etc. so the stand-up does not take up the preceding or succeeding aisle cell. While this may not be completely realistic (to fit two or potentially even three passengers in the same row and aisle space standing), we can potentially make use of this simplifying assumption to avoid unsolvable blockage and back-tracking situations, for example if two people need to stand up on the last row with no runoff area to go to and the aisle space preceding the last row is blocked as well. Furthermore, the aisle space is assumed to be blocked until either the arriving passenger has taken their place (if nobody had to stand up) or until the last passenger who had to stand up takes their place once again. After having made these simplifying assumptions about our system, we can move on to describing (some of) the events, states and transitions our system makes use of when simulating the boarding process.

2.1 Events and states

Table 1: List of events and states

events	states
α - passenger entering the door of aircraft	A_k - aisle space cell k is occupied [yes(1) or no(0)]
β - passenger walks one aisle cell	P_{ij} - passenger of row i and seat j in the plane, unseated [yes(1) or no(0)]
γ - passenger reaches their row	S_{ij} - seat j in row i occupied by passenger [yes(1) or no(0)]
δ - passenger stops without reaching their row	
ω - passenger takes their own place	

Our state space can be specified by the following arrays of individual state values as described in Table 1:

$$A = \begin{bmatrix} \{0,1\} & \{0,1\} & \{0,1\} & \{0,1\} \\ \{0,1\} & \{0,1\} \end{bmatrix} \quad P, S = \begin{bmatrix} \{0,1\} & \{0,1\} & \{0,1\} & \{0,1\} & \{0,1\} & \{0,1\} \\ \{0,1\} & \{0,1\} & \{0,1\} & \{0,1\} & \{0,1\} & \{0,1\} \end{bmatrix}$$

(1)

where we have a restriction on the combined values of S and P in such a way that $P_{ij} + S_{ij} \leq 1 \forall i \in \{1, \dots, 30\}, j \in \{1, \dots, 6\}$, which takes care of the restriction that a passenger cannot be in the plane walking (or waiting) and seated at the same time. Due to the large number of cells, each with their own state, it would be a massively complex list to describe all possible events in each state, but we can write up the transitions on the state space with events happening for one passenger. For passenger i, j , the events in certain states (Table 1) can occur in the following ways:

$$\begin{aligned}
L(A_1 = 1, P_{ij} = 0, S_{ij} = 0) &= \emptyset \\
L(A_1 = 0, P_{ij} = 0, S_{ij} = 0) &= \{\alpha\} \\
\text{If } P_{ij} = 1, \text{ then let the aisle cell taken up by } P_{ij} \text{ be denoted by } A_k, \text{ where } k - 5 \leq i \\
L(A_{k+1} = 0, 1, 0) &= \{\beta\} \text{ if } k - 5 < i \\
L(A_{k+1} = 1, 1, 0) &= \{\delta\} \\
L(A, 1, 0) &= \{\gamma\} \text{ if } k - 5 = i \\
L(A, 1, 0) &= \{\omega\} \\
L(A, 0, 1) &= \emptyset
\end{aligned} \tag{2}$$

Alternatively, we can line up the transitions on the state space each event occurrence causes in the following ways:

$$\begin{aligned}
\phi\left(([0, A_2, \dots, A_{35}], 0, 0), \alpha\right) &= ([1, A_2, \dots, A_{35}], 1, 0) \\
\phi\left([\dots, A_{k-1}, 1, 0, \dots], 1, 0), \beta\right) &= ([\dots, A_{k-1}, 0, 1, \dots], 1, 0) \\
\phi\left([\dots, A_{k-1}, 1, 1, \dots], 1, 0), \delta\right) &= \text{no change on state space} \\
\phi\left([\dots, A_{k-1}, 1, \dots], 1, 0), \gamma\right) &= \text{no change on state space} \\
\phi\left([\dots, A_{k-1}, 1, \dots], 1, 0), \omega\right) &= ([\dots, A_{k-1}, 0, \dots], 0, 1)
\end{aligned} \tag{3}$$

2.2 Boarding strategies

Our conceptual boarding model provides the system we use for simulating an aircraft boarding procedure, but it also raises the question of evaluation and optimisation of the system, i.e. how do we measure efficiency and how to make the functioning of the system more efficient. As [Jafer & Mi \(2017\)](#) describes a number of different boarding strategies, most of which with foundations in real life cases, our paper will utilize these boarding strategies as tools to evaluate the system. The list of these boarding strategies are: Back-to-Front (BF), Window-Middle-Aisle (WMA), Rotating Zone (RZ), Optimal (O), Practical-Optimal (PO), Reverse Pyramid (RP) and Random (R). The most obvious evaluation metric of these strategies applied on our boarding system is the total boarding time. From an airline's perspective, perhaps the most important measure of different boarding strategies is the time it takes to receive and host all passengers, from the time boarding is started until all passengers stow their luggage and take their seats. Another important metric could be the individual time a passenger spends boarding the aircraft with specific attention to not only the average but the values in the tails. The assumption here is that passengers generally do not enjoy standing, queuing and changing seats while within the narrow and crowded space of an aircraft. In the following, we will describe the boarding strategies also presented by [Jafer & Mi \(2017\)](#) in detail and in particular where we deviate from their methodology. One of the major deviations is that our aircraft model assumes no business seats (seats in a particular area) and no priority boarding,

i.e. all seats are treated as equal priority for the boarding strategies.

The *Back-to-Front* (BF) strategy means that the 30 row aircraft is divided into 6 boarding groups and their corresponding zones, where each zone is called upon to board one after the other. Like in all the other cases where we have some systematic strategy, we make a simplifying assumption that passengers are compliant with the boarding policies, that is they all board when they are called upon (such as having a boarding group designation on their boarding pass). The BF strategy designates zone 1 as rows 1-5, zone 2 as rows 6-10, zone 3 as rows 11-15, zone 4 as rows 16-20, zone 5 as rows 21-25 and zone 6 as rows 26-30. Then we begin the boarding from the back, i.e. passengers arrive randomly from zone 6 and once all of them arrived, we begin calling passengers from zone 5 and so on until everyone boards from zone 1. The order of arrival within the zone groups is sampled randomly without replacement. Intuitively, this strategy might be very helpful to avoid congestion along the aisle (people seated in the front will not keep back people in the back for long), but may be costly in terms of having to resolve passengers hopping in and out of their seats to let each other pass to their seats.

Similarly, the *Window-Middle-Aisle* (WMA) strategy creates 3 zones and calls upon the passengers there, who arrive randomly within the zones. Zone 1 contains all the window seats on both sides of the airplane, zone 2 contains all middle seats and zone 3 contains all aisle seats in all rows on both sides. This strategy may be very helpful in resolving issues presented by the BF strategy, namely eliminating the need for people to stand up and sit down again to let each other pass to their seats, but complementarily may cause long queues and blockage in the aisle.

Then the *Rotating Zone* (RZ) strategy addresses the issue of this blockage by changing the order of the boarding zones in the following fashion: zone 6 (rows 26-30) boards first so that the back zone can be eliminated. Then zone 1 (rows 1-5) is boarding to eliminate potential interference with the ones finishing the boarding in the back. Then zone 5, zone 2, zone 4 and zone 3 in this order. The rotation of the zones is meant to attempt to eliminate the interference among the groups.

The *Reverse Pyramid* (RP) strategy attempts to combine the BF and WMA strategies and harvest the benefits of both at the same time by building a 6 symmetric zone (both sides of the aircraft are used at the same time) boarding order. In our methodology, the groups are defined in a slightly different logic than the one described by [Jafer & Mi \(2017\)](#). Zone 1 consists of the window seats on rows 16-30. Zone 2 contains the window seats of rows 8-15 and middle seats of rows 24-30. Then zone 3 contains the window seats of rows 1-7 and middle seats of 16-23. Zone 4 contains the middle seats of rows 8-15 and aisle seats of 24-30. Zone 5 contains the middle seats of rows 1-7 and aisle seats of 16-23. And finally, zone 6 entails the aisle seats of rows 1-15. The boarding order is also the same, i.e. zone 1 starts and zone 6 finishes the boarding.

Then the *Optimal* (O) strategy details a very detailed zoning technique similar to [Jafer & Mi \(2017\)](#) where we define 12 zones based on the row number being odd or even, the position of the seat (WMA) and the side which they are on. Zone 1 consists of all window seats on even rows on one side (left) of the plane, zone 2 includes all window seats on even rows on the opposite (right) side, zone 3 entails the window seats on the odd rows on the left side of the plane, zone 4 contains all window seats on odd rows on the other (right) side of the plane, and then same is repeated for middle seats in zones 5-8 and for aisle seats in zones 9-12. Boarding order among zone groups is sequential from 1 to 12. The advantage of this strategy is that it combines the WMA strategy with the BF in a way that it tries to avoid congestion resulting from passengers trying to board the

same zone in the BF strategy. However, questions clearly arise in terms of the viability of practical implementation of such a strategy due to the underlying complexity.

In addition to the optimal, the *Practical-Optimal* (PO) strategy attempts to simplify the strategy by only considering the odd and even rows. In this strategy, zone 1 consists of all even row seats (with no WMA distinction) on the left side of the aircraft, zone 2 contains all even rows on the right side, zone 3 entails all odd row seats on the left and zone 4 the odd row seats on the right side of the plane. Similarly to the previous strategy, the boarding order of the zone groups is sequential, so it progresses from 1 to 4.

Finally, the *Random* (R) strategy simply randomizes the order of the arriving passengers regardless of their seating positions. This boarding strategy may serve as the default case of the boarding if the airline does not want to pay attention or put effort into defining boarding groups. Alternatively, it may also serve as a counterfactual to the strategies in case of a high rate of non-compliance, i.e. if passengers disregard (intentionally or accidentally) the calls for their own boarding groups and decide to enter randomly. Albeit the assignment of passengers to certain groups (always takers, compliers, etc.) may not be random to their characteristics, we may assume that their assigned seats are random to both their individual characteristics and their compliance groups.

3 Simulation program

Our simulation procedure consists of a number of functions managing the aisle space of the aircraft, the waiting delays and the seating manager taking care of luggage and shuffling of passengers on the same row. In this section, we will present the main logic of these functions along with the parameters and distributions used and the flowchart of the logic and events. First of all, we can discuss the attributes of the passengers boarding the plane and the random variables pertaining to the passengers. The passengers are assumed to have different walking speeds, which we estimate by a uniformly distributed random variable s . This variable describes the passengers' speed in meters per second, which they walk the aisle (we assume continuous and even walking speed, no acceleration or deceleration). Then we estimate the entrance time of each arriving passenger by an exponentially distributed random variable e , which is described in Table 2.

Table 2: List of random variables used for passenger attributes and seating management

Parameter description	Distribution
Walking speed of the passenger (m/s)	$s \sim U(0.27, 0.44)$
Seconds until passenger entrance/arrival	$e \sim Exp(\lambda = 1/2)$
Seconds it takes to stow away luggage	$l \sim U(6, 30)$
Seconds it takes to sit down/stand up aisle seat	$a \sim U(2, 5)$
Seconds it takes to sit down/stand up middle seat	$m \sim U(3, 6)$
Seconds it takes to sit down window seat	$w \sim U(4, 7)$

WRITE ABOUT AISLE

Then once the passenger reaches their own row, they stop on the aisle cell and the seating management process begins. Figure 1 presents the flow of logic once the passenger arrives to their row. First, the passenger takes care of stowing away their luggage if they have any, which takes a random amount of time estimated by a random parameter l

which follows a uniform distribution (Table 2). Then after the potential luggage stash, we need to establish if the arriving passenger has a window, middle or aisle seat. If the passenger has the aisle seat ($j = \{3,4\}$), they can take their seat regardless of whether other passengers are sitting on the same row already. The time to take the seat and vacate the aisle cell takes a random parameter a from a uniform distribution. If the arriving passenger has a middle seat ($j = \{2,5\}$), we need to investigate whether the aisle seat is occupied already. If not, then the arriving passenger takes the middle seat using up time estimated by a parameter m from a uniform distribution. If the aisle seat is taken, then the passenger there needs to stand up (a), then the arriving passenger takes their seat (m) and then the aisle passenger sits back, vacating the space (time spent on this is again the same a drawn from the distribution earlier).

Finally if the arriving passenger has a window seat ($j = \{1,6\}$), we have 4 cases. First is if both the aisle and middle seats are free, in which case the passenger takes their seat with a time parameter w from another uniform distribution (Table 2). The second and third cases entail when either the aisle or the middle seat is occupied but the other one is not. In this case, the occupying passenger stands up (a or m depending on seat), the arriving passenger takes their place (w) and then the other passenger sits back, vacating the aisle cell. Fourth and last is if both aisle and middle seats are occupied. In this case, first the aisle seat passenger stands up (a), then the middle passenger stands up (m), then the arriving passenger takes their seat (w), after which the occupying passengers sit back in the same time variables.

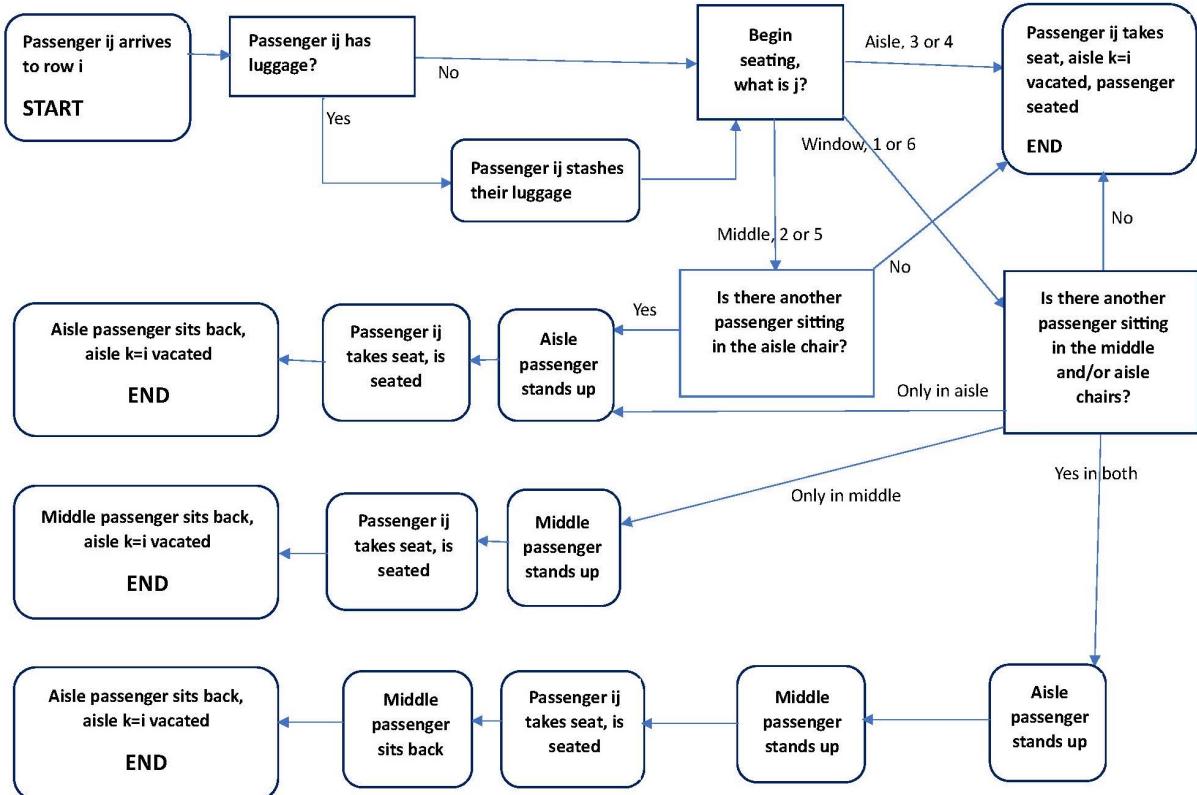


Figure 1: Flow chart of the seating procedure

4 Validation and Verification

Research by [Chun & Wai Tak Mak \(1999\)](#) suggests the use of an exponential distribution for passenger arrival and check in processing time, with a time varying mean parameter. The average waiting time resulting from the simulation is found to be 10 minutes when the counter profile is equipped with sufficient staff. The maximum waiting time is found to be 55 minutes, the maximum queue length is found to be 8 minutes whereas the average queue length is 1.4 minutes. Another research study [Schultz \(2008\)](#) used the triangular distribution to model the storage process which was found to be 13.9 seconds as well as the Weilbull distribution with the scale parameter $\alpha = 1.7$ and shape parameter $\beta = 16$ seconds. The study by [Van Landeghem & Beuselinck \(2002\)](#) suggested γ is the number of capacity of luggage space which was found to be 6, α which is defined as the time to stow luggage was found to be 2 and β is the correction coefficient which was found to be 11. According [Qiang et al. \(2018\)](#) who used the logit model to model seat choice behavior the parameter V which refers to the attraction value of seat was estimated to take a value of 9.1. In addition , the suggested optimal value for the scaled parameter θ which measures preference difference toward the two successive takes the value of 0.15 and leads to a delay of the boarding time when the value increases. The parameter ϵ which measures the exclusion effect between individual passengers in the same half row is found to be 0.5, as the value of the parameter increases, it will lead to passengers select seats in different rows. Finally the simulation results by [Jafer & Mi \(2017\)](#) who found that the window middle aisle boarding strategy which was the fastest took 26 minutes suggest the parameters walking speed is between 0.27 and 0.44 m/s, the clearing time is between 6 and 30 seconds, the getting out of a seat is between 3 and 4.2 seconds and the passenger flow rate is between 0.2 and 1 pax/s. According to the random variables for seating management which have been generated from a uniform distribution, the time to stow luggage takes 6 seconds on average with a variance of 30 seconds which is lower than the results of [Schultz \(2008\)](#) who found the time to stow luggage to be approximately 13.9 seconds. The number of seconds to sit down and stand up from an aisle seat has been estimated to be on average 2 seconds with a variance of 5 seconds. Similarly, The number of seconds to sit down and stand up from a middle seat has been estimated to be on average 3 seconds with a variance of 6 seconds and from a window seat 4 seconds on average with a variance of 7 seconds. The findings of [Jafer & Mi \(2017\)](#) indicate that getting in and of a seat takes approximately between 3 and 4.2 seconds depending on the seat, which indicate that the results from this study are quite in line with their results. The walking speed was estimated to be approximately 0.2 seconds on average with a variance of 0.44 whereas the findings of [Jafer & Mi \(2017\)](#) also indicate that the walking speed was estimated as approximately between 0.27 and 0.44 miles per second which is quite in line with the results of this study.

5 Results

In our evaluation of the different boarding strategies, we make a simulated list of passengers corresponding to a particular boarding strategy, repeat this 100 times for each strategy and run the simulation. Then we evaluate each strategy by the average individual boarding time, the 95th percentile c-var measure of the same individual boarding time and the total boarding time from start to finish.

Table 3: Results of a 100 simulations for each boarding strategy, with the average time in minutes across the 100 simulations for each strategy

Boarding strategy times	Mean individual boarding	CVAR95 individual boarding	Total boarding
Back-to-front (BF)	4.025	6.604	38.821
Window-Middle-Aisle (WMA)	1.968	3.141	24.451
Rotating-Zone (RZ)	3.483	5.929	35.941
Optimal (O)	1.901	3.024	23.835
Practical-Optimal (PO)	2.208	3.566	27.617
Reverse-Pyramid (RP)	2.256	3.862	26.325
Random (R)	2.259	3.695	28.071

6 Discussion

7 Conclusion

Based on our results after taking the average of all 100 simulations we find that the window middle aisle is most effective boarding strategy according to the total boarding tie,

8 Appendix

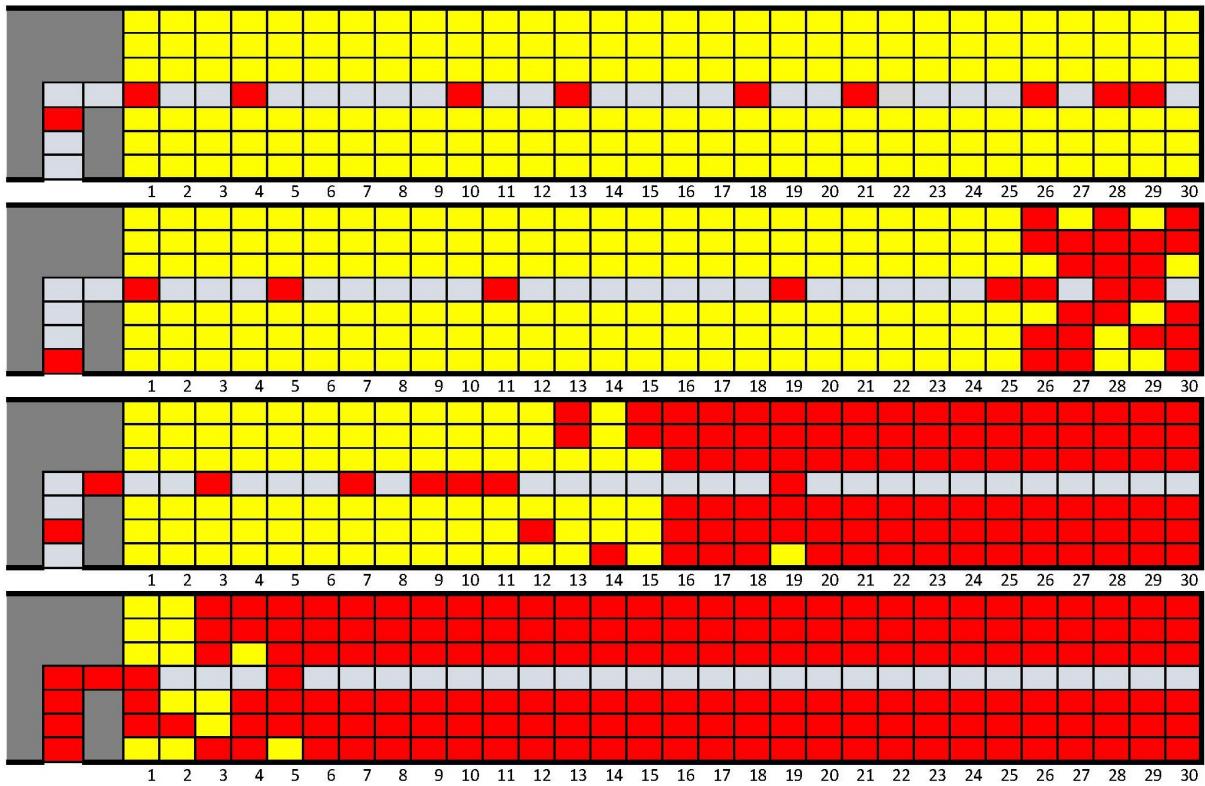


Figure 2: Back-to-front strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

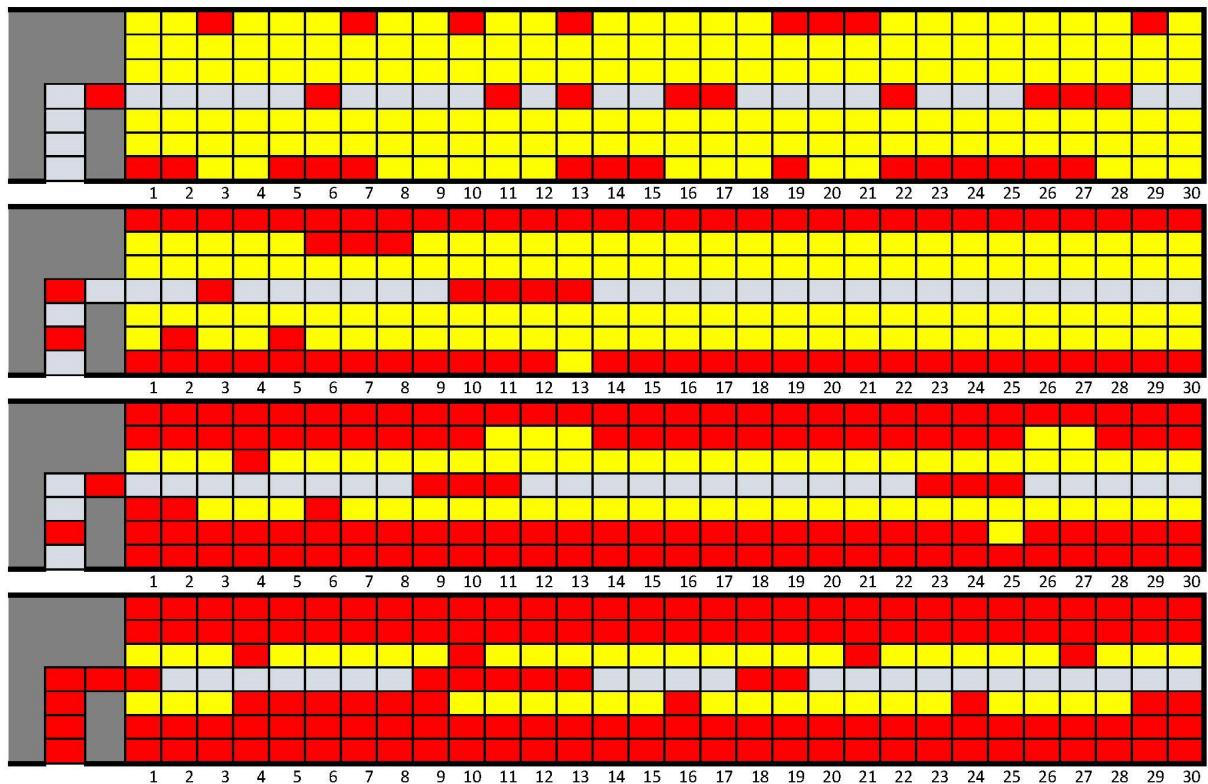


Figure 3: Window-Middle-Aisle strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

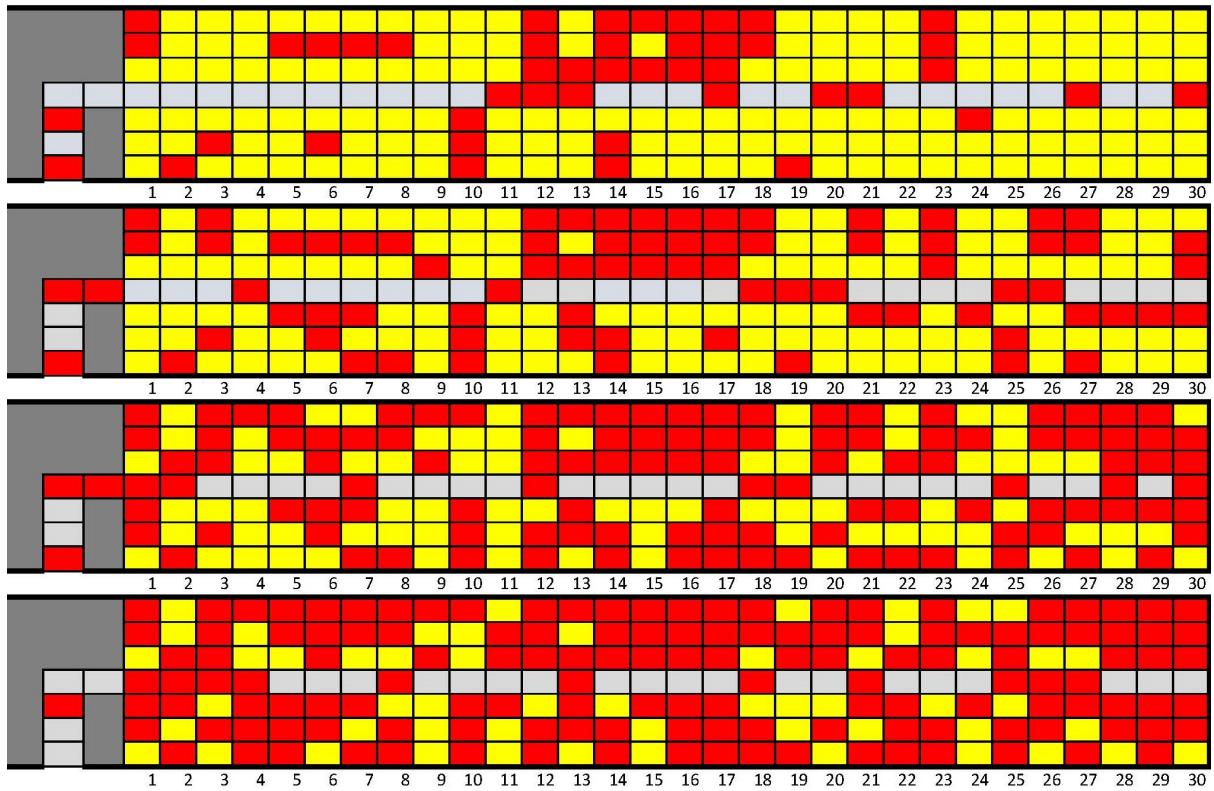


Figure 4: Random strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.



Figure 5: Reverse-Pyramid strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

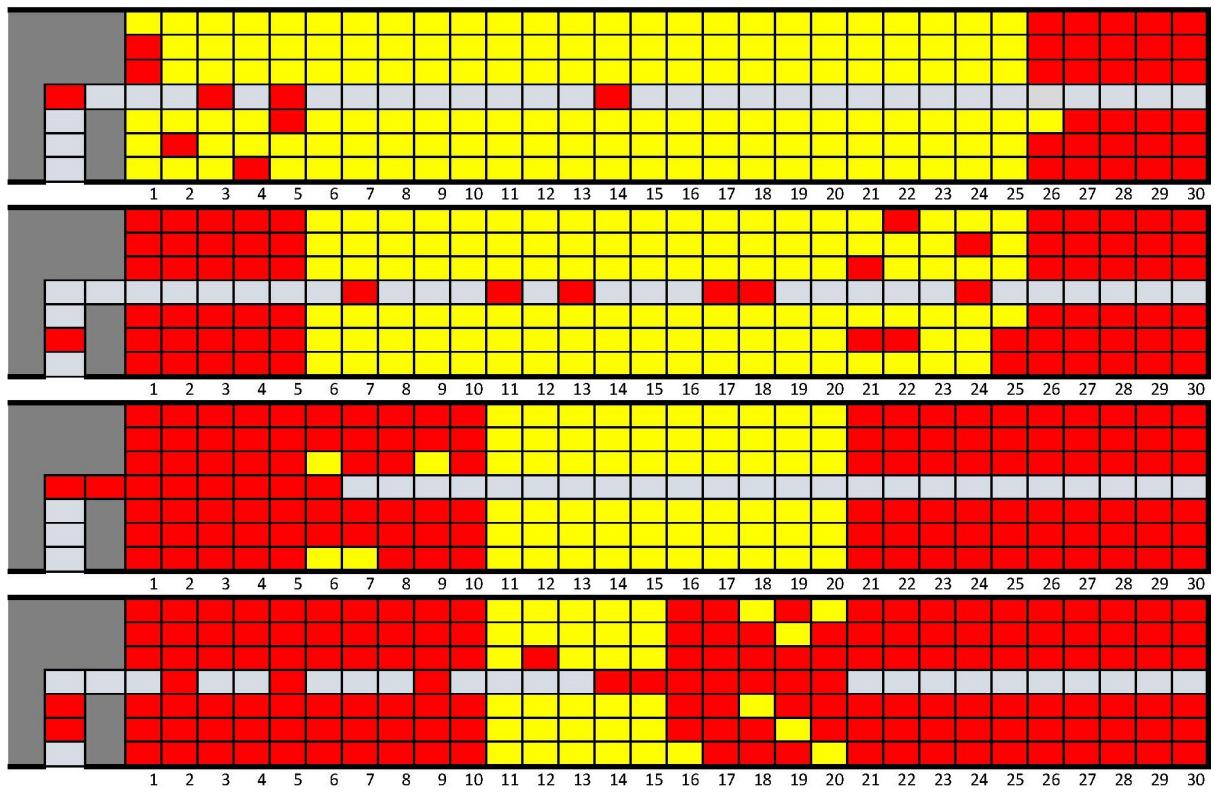


Figure 6: Rotating-zone strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

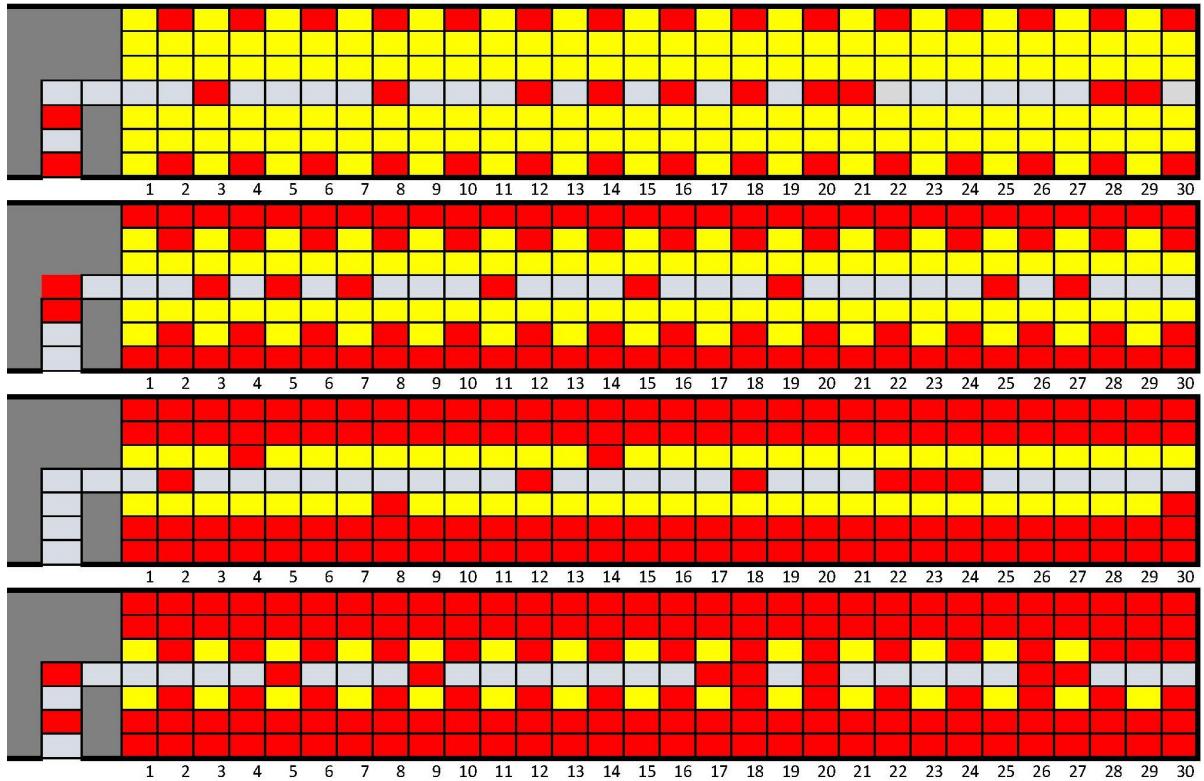


Figure 7: Optimal strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

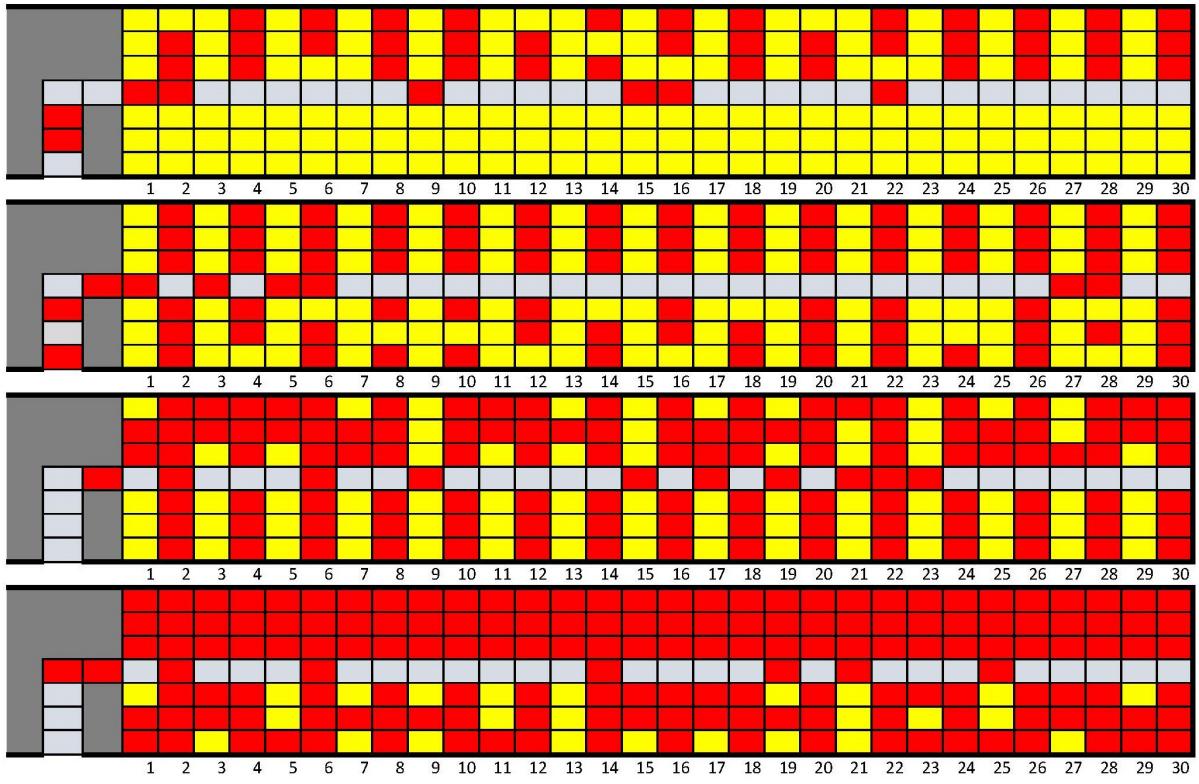


Figure 8: Practical-optimal strategy snapshots of the boarding procedure (conceptual snapshots, not taken from our simulations). The dark grey area is inaccessible for passengers, light grey marks a free aisle cell, yellow marks an unoccupied seat and red marks cells occupied by a passenger.

9 Code

Listing 1: Package imports, magic numbers and function calls

```
1 import numpy as np
2 import numpy.random as rnd
3 import random
4 import time
5 from statistics import mean
6 import csv
7
8 random.seed(10)
9
10 #####
11 ### Magic numbers
12 # Number of simulations per boarding strategy
13 nsims = 100
14
15 # All analyzed boarding strategies
16 strategies = ["backtofront", "outsidein", "rotatingzone", "optimal",
17   ↪ "pracoptimal", "revpyramid"]
18
19 # Plane attributes
20 # Number of seat rows on the plane
21 nrows = 30
22 # Seats per row
23 nseats = 6
24 # Total length of the aisle
25 l_aisle = 35
26
27 # The space occupied on the aisle is larger next to the seats than on
28   ↪ the first part.
29 size_aisle_seats = 0.7112
30 size_aisle_begin = 0.4572
31
32 # Mean arrival delay (in seconds)
33 mu_arrival = 2
34
35 #####
36 ### Simulation      (initialization is done in the SimulationExecutor
37   ↪ function)
38
39 # Run the simulation!
40 time_measures = Simulation_analysis(strategies, nrows, nseats, l_aisle,
41   ↪ size_aisle_seats, size_aisle_begin, mu_arrival)
```

Listing 2: Boarding strategy function

```

1 def BoardingStrat(type):
2     list_order = []
3     if type == "backtofront":
4         zone1_seats = []
5         zone2_seats = []
6         zone3_seats = []
7         zone4_seats = []
8         zone5_seats = []
9         zone6_seats = []
10    for row in [1, 2, 3, 4, 5]:
11        for col in range(6):
12            zone1_seats.append((row, col + 1))
13    for row in [6, 7, 8, 9, 10]:
14        for col in range(6):
15            zone2_seats.append((row, col + 1))
16    for row in [11, 12, 13, 14, 15]:
17        for col in range(6):
18            zone3_seats.append((row, col + 1))
19    for row in [16, 17, 18, 19, 20]:
20        for col in range(6):
21            zone4_seats.append((row, col + 1))
22    for row in [21, 22, 23, 24, 25]:
23        for col in range(6):
24            zone5_seats.append((row, col + 1))
25    for row in [26, 27, 28, 29, 30]:
26        for col in range(6):
27            zone6_seats.append((row, col + 1))
28    while bool(zone6_seats) is True:
29        choice = random.sample(zone6_seats, 1)[0]
30        remove = zone6_seats.index(choice)
31        list_order.append(choice)
32        zone6_seats.pop(remove)
33    while bool(zone5_seats) is True:
34        choice = random.sample(zone5_seats, 1)[0]
35        remove = zone5_seats.index(choice)
36        list_order.append(choice)
37        zone5_seats.pop(remove)
38    while bool(zone4_seats) is True:
39        choice = random.sample(zone4_seats, 1)[0]
40        remove = zone4_seats.index(choice)
41        list_order.append(choice)
42        zone4_seats.pop(remove)
43    while bool(zone3_seats) is True:
44        choice = random.sample(zone3_seats, 1)[0]
45        remove = zone3_seats.index(choice)
46        list_order.append(choice)
47        zone3_seats.pop(remove)

```

```

48     while bool(zone2_seats) is True:
49         choice = random.sample(zone2_seats, 1)[0]
50         remove = zone2_seats.index(choice)
51         list_order.append(choice)
52         zone2_seats.pop(remove)
53     while bool(zone1_seats) is True:
54         choice = random.sample(zone1_seats, 1)[0]
55         remove = zone1_seats.index(choice)
56         list_order.append(choice)
57         zone1_seats.pop(remove)
58 elif type == "outsidein":
59     window_seats = []
60     middle_seats = []
61     aisle_seats = []
62     for row in range(30):
63         for col in [1,6]:
64             window_seats.append((row + 1, col))
65     for row in range(30):
66         for col in [2,5]:
67             middle_seats.append((row + 1, col))
68     for row in range(30):
69         for col in [3,4]:
70             aisle_seats.append((row + 1, col))
71     while bool(window_seats) is True:
72         choice = random.sample(window_seats, 1)[0]
73         remove = window_seats.index(choice)
74         list_order.append(choice)
75         window_seats.pop(remove)
76     while bool(middle_seats) is True:
77         choice = random.sample(middle_seats, 1)[0]
78         remove = middle_seats.index(choice)
79         list_order.append(choice)
80         middle_seats.pop(remove)
81     while bool(aisle_seats) is True:
82         choice = random.sample(aisle_seats, 1)[0]
83         remove = aisle_seats.index(choice)
84         list_order.append(choice)
85         aisle_seats.pop(remove)
86 elif type == "rotatingzone":
87     zone1_seats = []
88     zone2_seats = []
89     zone3_seats = []
90     zone4_seats = []
91     zone5_seats = []
92     zone6_seats = []
93     for row in [1,2,3,4,5]:
94         for col in range(6):

```

```

95             zone1_seats.append((row, col+1))
96     for row in [6,7,8,9,10]:
97         for col in range(6):
98             zone2_seats.append((row, col+1))
99     for row in [11,12,13,14,15]:
100        for col in range(6):
101            zone3_seats.append((row, col+1))
102    for row in [16,17,18,19,20]:
103        for col in range(6):
104            zone4_seats.append((row, col+1))
105    for row in [21,22,23,24,25]:
106        for col in range(6):
107            zone5_seats.append((row, col+1))
108    for row in [26,27,28,29,30]:
109        for col in range(6):
110            zone6_seats.append((row, col+1))
111    while bool(zone6_seats) is True:
112        choice = random.sample(zone6_seats, 1)[0]
113        remove = zone6_seats.index(choice)
114        list_order.append(choice)
115        zone6_seats.pop(remove)
116    while bool(zone1_seats) is True:
117        choice = random.sample(zone1_seats, 1)[0]
118        remove = zone1_seats.index(choice)
119        list_order.append(choice)
120        zone1_seats.pop(remove)
121    while bool(zone5_seats) is True:
122        choice = random.sample(zone5_seats, 1)[0]
123        remove = zone5_seats.index(choice)
124        list_order.append(choice)
125        zone5_seats.pop(remove)
126    while bool(zone2_seats) is True:
127        choice = random.sample(zone2_seats, 1)[0]
128        remove = zone2_seats.index(choice)
129        list_order.append(choice)
130        zone2_seats.pop(remove)
131    while bool(zone4_seats) is True:
132        choice = random.sample(zone4_seats, 1)[0]
133        remove = zone4_seats.index(choice)
134        list_order.append(choice)
135        zone4_seats.pop(remove)
136    while bool(zone3_seats) is True:
137        choice = random.sample(zone3_seats, 1)[0]
138        remove = zone3_seats.index(choice)
139        list_order.append(choice)
140        zone3_seats.pop(remove)
141 elif type == "optimal":

```

```

142     zone1_seats = []
143     zone2_seats = []
144     zone3_seats = []
145     zone4_seats = []
146     zone5_seats = []
147     zone6_seats = []
148     zone7_seats = []
149     zone8_seats = []
150     zone9_seats = []
151     zone10_seats = []
152     zone11_seats = []
153     zone12_seats = []
154     for row in range(30, 0, -2):
155         zone1_seats.append((row, 1))
156     for row in range(30, 0, -2):
157         zone2_seats.append((row, 6))
158     for row in range(29, 0, -2):
159         zone3_seats.append((row, 1))
160     for row in range(29, 0, -2):
161         zone4_seats.append((row, 6))
162     for row in range(30, 0, -2):
163         zone5_seats.append((row, 2))
164     for row in range(30, 0, -2):
165         zone6_seats.append((row, 5))
166     for row in range(29, 0, -2):
167         zone7_seats.append((row, 2))
168     for row in range(29, 0, -2):
169         zone8_seats.append((row, 5))
170     for row in range(30, 0, -2):
171         zone9_seats.append((row, 3))
172     for row in range(30, 0, -2):
173         zone10_seats.append((row, 4))
174     for row in range(29, 0, -2):
175         zone11_seats.append((row, 3))
176     for row in range(29, 0, -2):
177         zone12_seats.append((row, 4))
178     while bool(zone1_seats) is True:
179         choice = random.sample(zone1_seats, 1)[0]
180         remove = zone1_seats.index(choice)
181         list_order.append(choice)
182         zone1_seats.pop(remove)
183     while bool(zone2_seats) is True:
184         choice = random.sample(zone2_seats, 1)[0]
185         remove = zone2_seats.index(choice)
186         list_order.append(choice)
187         zone2_seats.pop(remove)
188     while bool(zone3_seats) is True:

```

```

189     choice = random.sample(zone3_seats, 1)[0]
190     remove = zone3_seats.index(choice)
191     list_order.append(choice)
192     zone3_seats.pop(remove)
193     while bool(zone4_seats) is True:
194         choice = random.sample(zone4_seats, 1)[0]
195         remove = zone4_seats.index(choice)
196         list_order.append(choice)
197         zone4_seats.pop(remove)
198     while bool(zone5_seats) is True:
199         choice = random.sample(zone5_seats, 1)[0]
200         remove = zone5_seats.index(choice)
201         list_order.append(choice)
202         zone5_seats.pop(remove)
203     while bool(zone6_seats) is True:
204         choice = random.sample(zone6_seats, 1)[0]
205         remove = zone6_seats.index(choice)
206         list_order.append(choice)
207         zone6_seats.pop(remove)
208     while bool(zone7_seats) is True:
209         choice = random.sample(zone7_seats, 1)[0]
210         remove = zone7_seats.index(choice)
211         list_order.append(choice)
212         zone7_seats.pop(remove)
213     while bool(zone8_seats) is True:
214         choice = random.sample(zone8_seats, 1)[0]
215         remove = zone8_seats.index(choice)
216         list_order.append(choice)
217         zone8_seats.pop(remove)
218     while bool(zone9_seats) is True:
219         choice = random.sample(zone9_seats, 1)[0]
220         remove = zone9_seats.index(choice)
221         list_order.append(choice)
222         zone9_seats.pop(remove)
223     while bool(zone10_seats) is True:
224         choice = random.sample(zone10_seats, 1)[0]
225         remove = zone10_seats.index(choice)
226         list_order.append(choice)
227         zone10_seats.pop(remove)
228     while bool(zone11_seats) is True:
229         choice = random.sample(zone11_seats, 1)[0]
230         remove = zone11_seats.index(choice)
231         list_order.append(choice)
232         zone11_seats.pop(remove)
233     while bool(zone12_seats) is True:
234         choice = random.sample(zone12_seats, 1)[0]
235         remove = zone12_seats.index(choice)

```

```

236         list_order.append(choice)
237         zone12_seats.pop(remove)
238     elif type == "pracoptimal":
239         zone1_seats = []
240         zone2_seats = []
241         zone3_seats = []
242         zone4_seats = []
243         for row in range(30, 0, -2):
244             for col in [1, 2, 3]:
245                 zone1_seats.append((row, col))
246         for row in range(30, 0, -2):
247             for col in [4, 5, 6]:
248                 zone2_seats.append((row, col))
249         for row in range(29, 0, -2):
250             for col in [1, 2, 3]:
251                 zone3_seats.append((row, col))
252         for row in range(29, 0, -2):
253             for col in [4, 5, 6]:
254                 zone4_seats.append((row, col))
255     while bool(zone1_seats) is True:
256         choice = random.sample(zone1_seats, 1)[0]
257         remove = zone1_seats.index(choice)
258         list_order.append(choice)
259         zone1_seats.pop(remove)
260     while bool(zone2_seats) is True:
261         choice = random.sample(zone2_seats, 1)[0]
262         remove = zone2_seats.index(choice)
263         list_order.append(choice)
264         zone2_seats.pop(remove)
265     while bool(zone3_seats) is True:
266         choice = random.sample(zone3_seats, 1)[0]
267         remove = zone3_seats.index(choice)
268         list_order.append(choice)
269         zone3_seats.pop(remove)
270     while bool(zone4_seats) is True:
271         choice = random.sample(zone4_seats, 1)[0]
272         remove = zone4_seats.index(choice)
273         list_order.append(choice)
274         zone4_seats.pop(remove)
275     elif type == "revpyramid":
276         zone1_seats = []
277         zone2_seats = []
278         zone3_seats = []
279         zone4_seats = []
280         zone5_seats = []
281         zone6_seats = []
282         # Zone 1

```

```

283     for row in range(15, 30):
284         for col in [1, 6]:
285             zone1_seats.append((row + 1, col))
286     # Zone 2
287     for row in range(7, 15):
288         for col in [1, 6]:
289             zone2_seats.append((row + 1, col))
290     for row in range(23, 30):
291         for col in [2, 5]:
292             zone2_seats.append((row + 1, col))
293     # Zone 3
294     for row in range(0, 7):
295         for col in [1, 6]:
296             zone3_seats.append((row + 1, col))
297     for row in range(15, 23):
298         for col in [2, 5]:
299             zone3_seats.append((row + 1, col))
300     # Zone 4
301     for row in range(23, 30):
302         for col in [3, 4]:
303             zone4_seats.append((row + 1, col))
304     for row in range(7, 15):
305         for col in [2, 5]:
306             zone4_seats.append((row + 1, col))
307     # Zone 5
308     for row in range(0, 7):
309         for col in [2, 5]:
310             zone5_seats.append((row + 1, col))
311     for row in range(15, 23):
312         for col in [3, 4]:
313             zone5_seats.append((row + 1, col))
314     # Zone 6
315     for row in range(0, 15):
316         for col in [3, 4]:
317             zone6_seats.append((row + 1, col))
318     while bool(zone1_seats) is True:
319         choice = random.sample(zone1_seats, 1)[0]
320         remove = zone1_seats.index(choice)
321         list_order.append(choice)
322         zone1_seats.pop(remove)
323     while bool(zone2_seats) is True:
324         choice = random.sample(zone2_seats, 1)[0]
325         remove = zone2_seats.index(choice)
326         list_order.append(choice)
327         zone2_seats.pop(remove)
328     while bool(zone3_seats) is True:
329         choice = random.sample(zone3_seats, 1)[0]

```

```

330         remove = zone3_seats.index(choice)
331         list_order.append(choice)
332         zone3_seats.pop(remove)
333     while bool(zone4_seats) is True:
334         choice = random.sample(zone4_seats, 1)[0]
335         remove = zone4_seats.index(choice)
336         list_order.append(choice)
337         zone4_seats.pop(remove)
338     while bool(zone5_seats) is True:
339         choice = random.sample(zone5_seats, 1)[0]
340         remove = zone5_seats.index(choice)
341         list_order.append(choice)
342         zone5_seats.pop(remove)
343     while bool(zone6_seats) is True:
344         choice = random.sample(zone6_seats, 1)[0]
345         remove = zone6_seats.index(choice)
346         list_order.append(choice)
347         zone6_seats.pop(remove)
348 else: # random
349     seats = []
350     for row in range(30):
351         for col in range(6):
352             seats.append((row+1, col+1))
353     while bool(seats) is True:
354         choice = random.sample(seats, 1)[0]
355         remove = seats.index(choice)
356         list_order.append(choice)
357         seats.pop(remove)
358 return list_order

```

Listing 3: Functions developed for the simulation

```

1 def Queue_former(strategy):
2     arrival_queue = BoardingStrat(strategy)
3     passengers = []
4     ID = 0
5     for i in enumerate(arrival_queue):
6         ID += 1
7         passengers.append(PassengerGenerator(ID, i[1][0], i[1][1]))
8     return passengers
9
10
11
12 def SeatingManager(arriving_passenger, seating_matrix, current_time):
13     seat = arriving_passenger["seat"]
14     row = arriving_passenger["row"]
15     if seat==3 or seat==4:

```

```

16     # Arriving has aisle seat
17     aisle_seating = random.uniform(2, 5) # Aisle seating time
18     ↪ (aisle can be taken regardless of people already sitting in
19     ↪ the row
20     arriving_passenger["timespent"]["seating_time"] += aisle_seating
21     seating_time = aisle_seating
22 elif seat==2 or seat==5:
23     # Arriving has middle seat
24     if seat==2:
25         target_seat = 3
26     else:
27         target_seat = 4
28     if bool(seating_matrix[row-1][target_seat-1]) is True:
29         # Aisle seat is taken
30         aisle_seating = random.uniform(2, 5) # Standup time of the
31         ↪ person in the aisle seat
32         middle_seating = random.uniform(3, 6) # middle seating time
33         seating_matrix[row-1][target_seat-1]["timespent"]["standup"]
34         ↪ += aisle_seating*2 + middle_seating
35         arriving_passenger["timespent"]["seating_time"] +=
36         ↪ middle_seating+aisle_seating
37         seating_time = aisle_seating*2 + middle_seating
38     else:
39         middle_seating = random.uniform(3, 6) # middle seating time
40         arriving_passenger["timespent"]["seating_time"] +=
41         ↪ middle_seating
42         seating_time = middle_seating
43 else:
44     # Arriving has window seat
45     if seat==1:
46         target_middle = 2
47         target_aisle = 3
48     else:
49         target_middle = 5
50         target_aisle = 4
51     if bool(seating_matrix[row - 1][target_aisle - 1]) is True:
52         if bool(seating_matrix[row - 1][target_middle - 1]) is True:
53             # Both aisle and middle are taken
54             aisle_seating = random.uniform(2, 5) # Standup time of
55             ↪ the person in the aisle seat
56             middle_seating = random.uniform(3, 6) # Standup time of
57             ↪ the person in the middle seat
58             window_seating = random.uniform(4, 7) # window seating
59             ↪ time
60             arriving_passenger["timespent"]["seating_time"] +=
61             ↪ window_seating + aisle_seating + middle_seating
62             seating_matrix[row - 1][target_aisle -
63             ↪ 1]["timespent"]["standup"] += aisle_seating*2 +
64             ↪ middle_seating*2 + window_seating

```

```

53         seating_matrix[row - 1][target_middle -
54             ↵ 1]["timespent"]["standup"] += middle_seating * 2 +
55             ↵ window_seating
56         seating_time = aisle_seating*2 + middle_seating*2 +
57             ↵ window_seating
58     else:
59         # Only aisle seat is taken
60         aisle_seating = random.uniform(2, 5) # Standup time of
61             ↵ the person in the aisle seat
62         window_seating = random.uniform(4, 7) # window seating
63             ↵ time
64         seating_matrix[row - 1][target_aisle -
65             ↵ 1]["timespent"]["standup"] += aisle_seating * 2 +
66             ↵ window_seating
67         arriving_passenger["timespent"]["seating_time"] +=
68             ↵ window_seating + aisle_seating
69         seating_time = aisle_seating * 2 + window_seating
70     else:
71         if bool(seating_matrix[row - 1][target_middle - 1]):
72             # Only middle seat is taken
73             middle_seating = random.uniform(3, 6) # Standup time of
74                 ↵ the person in the middle seat
75             window_seating = random.uniform(4, 7) # window seating
76                 ↵ time
77             seating_matrix[row - 1][target_middle -
78                 ↵ 1]["timespent"]["standup"] += middle_seating * 2 +
79                 ↵ window_seating
80             arriving_passenger["timespent"]["seating_time"] +=
81                 ↵ window_seating + middle_seating
82             seating_time = middle_seating * 2 + window_seating
83     else:
84         window_seating = random.uniform(4, 7) # window seating
85             ↵ time
86         arriving_passenger["timespent"]["seating_time"] +=
87             ↵ window_seating
88         seating_time = window_seating
89
90     # Setting time measures.
91     aisle_occupancy_time =
92         ↵ arriving_passenger["timespent"]["luggage_stash"] + seating_time
93     arriving_passenger["timespent"]["time_seated"] = current_time[0] +
94         ↵ arriving_passenger["timespent"]["luggage_stash"] +
95         ↵ arriving_passenger["timespent"]["seating_time"]
96     arriving_passenger["timespent"]["board_time_cum"] =
97         ↵ arriving_passenger["timespent"]["time_seated"] -
98         ↵ arriving_passenger["timespent"]["time_entrance"]

```

```

80     # Seat the arriving passenger in the seating matrix
81     seating_matrix[row - 1][seat - 1] = arriving_passenger
82
83     # Return the seating matrix, the time it takes for the aisle to
84     # clear.
85     res_seating = [seating_matrix, aisle_occupancy_time]
86     return res_seating
87
88
89 def PassengerGenerator(ID, row, seat):
90     # Simulated parameters
91     has_luggage = rnd.choice((1, 0), p=[0.8, 0.2]) # logical if person
92     # has a luggage (1-yes, 0-no)
93     if has_luggage==1:
94         luggage_stash = random.uniform(6, 30) # seconds spent with
95         # luggage stash
96     else:
97         luggage_stash = 0
98     walkspeed = random.uniform(0.27, 0.44) # meter per second
99
100    # Calculated attributes and containing these attributes
101    walking_time = (0.7112/walkspeed) * row + 4*(0.4572/walkspeed)
102    dict_res = {
103        "row": row,
104        "seat": seat,
105        "has_luggage": has_luggage,
106        "walkspeed": {'aisle_begin' : size_aisle_begin/walkspeed,
107                      'aisle_seats' : size_aisle_seats/walkspeed},
108        "aisle_pos": None,
109        "seated": False,
110        "ID": ID,
111        "timespent": {
112            "board_time_cum": None,
113            "time_entrance": None,
114            "time_seated": None,
115            "walking": walking_time,
116            "waiting": 0,
117            "luggage_stash": luggage_stash,
118            "seating_time": 0,
119            "standup": 0
120        }
121    }
122
123    return dict_res

```

```

124 def New_arrival(passengers, aisle, current_time, events):
125     # We find the first passenger on the passengers list that has not
126     # yet been assigned a spot in the aisle.
127     for i in range(len(passengers)):
128         if passengers[i]['aisle_pos'] == None:
129             while aisle[0] == 0:
130                 passengers[i]['aisle_pos'] = 0
131                 passengers[i]['timespent']['time_entrance'] =
132                     current_time[0]
133                 aisle[0] = 1
134                 # Make a new entry for the passenger in the event list
135                 pass_event = Passenger_event(passengers[i]['ID'],
136                     passengers[i]['walkspeed']['aisle_begin'])
137                 events.append(pass_event)
138
139
140
141
142
143 def BoardingSimulator9000(current_time, passengers, aisle, events,
144     seating_matrix, mu_arrival):
145     # Step 1: Identify the next event
146     min_timer = min(events, key = lambda x: x['timer'])['timer']      #
147     # Find the time to the next event
148     # Retrieve the event list index of the next event (by matching the
149     # smallest timer)
150     action_index = next((index for (index, e) in enumerate(events) if
151         e["timer"] == min_timer), None)
152
153     current_time[0] += min_timer # Update current time
154     for i in range(len(events)):
155         events[i]['timer'] -= min_timer      # Update the event timers.
156
157     # Step 2: Execute the next event
158     # First we define what happens if the event is a new arrival
159     if events[action_index]['ID'] == 'Arrival':
160         # In case the first spot on the aisle is occupied, the arrival
161         # is delayed.
162         if bool(aisle[0]):
163             events[action_index]['timer'] = 1    # Setting the delay
164             # arbitrarily to 1 second
165             # Note that we are only counting waiting time on the plane.
166             # Waiting time in the tube is
167         else:

```

```

161     events[action_index]['timer'] =
162         random.expovariate(1/mu_arrival) # Otherwise draw from
163         # an exp. distribution with mean 2
164         New_arrival(passengers, aisle, current_time, events)      #
165         # And let the new passenger arrive.
166
167     # If all passengers are in the plane, remove the arrival event
168     # from the event list.
169     if all(p['aisle_pos'] is not None for p in passengers):
170         del events[action_index]
171
172
173     else: # If we are not dealing with a new arrival, we must be dealing
174         # with a passenger.
175         # Designate the passenger that is going to move.
176         pass_index = next((index for (index, d) in enumerate(passengers)
177                         if d["ID"] == events[action_index]['ID']), None)
178         # Retrieve the passenger's position on the aisle.
179         aisle_position = passengers[pass_index]['aisle_pos']
180
181         # First we check if the passenger has already been seated. If
182         # so, we remove the passenger's actions from the event list.
183         if passengers[pass_index]['seated'] == True:
184             del events[action_index]
185             # Uncomment line below for continuously keeping track of the
186             # simulation.
187             # print(f" Passenger {passengers[pass_index]['ID']} is
188             # seated at time {current_time[0]}")
189             # And his space on the aisle is cleared.
190             aisle[aisle_position] = 0
191
192         # If not yet seated, the passenger will try to move up a space
193         # on the aisle.
194         else:
195             # If the next space on the aisle is occupied, the passenger
196             # needs to wait.
197             if bool(aisle[aisle_position + 1]):
198                 # Taking into account smaller tiles at the beginning of
199                 # the aisle.
200                 if aisle_position < l_aisle - nrows:
201                     # Schedule the passenger's next movement.
202                     events[action_index]['timer'] =
203                         passengers[pass_index]['walkspeed']['aisle_begin']
204                         # Update waiting time.
205                         passengers[pass_index]['timespent']['waiting'] +=
206                             passengers[pass_index]['walkspeed']['aisle_begin']
207                         # Now for the larger aisle tiles at the seats.
208                         else:
209                             events[action_index]['timer'] =
210                                 passengers[pass_index]['walkspeed']['aisle_seats']

```

```

194         passengers[pass_index]['timespent']['waiting'] +=  

195             → passengers[pass_index]['walkspeed']['aisle_seats']  

196  

197     # If the next space on the aisle is vacant, the passenger  

198     → moves.  

199     else:  

200         aisle[aisle_position] = 0  

201         aisle[aisle_position + 1] = 1  

202         passengers[pass_index]['aisle_pos'] = aisle_position + 1  

203  

204     # If he has not arrived at his row number yet, we simply  

205     → schedule his next movement.  

206     if not passengers[pass_index]['aisle_pos'] ==  

207         → passengers[pass_index]['row'] + l_aisle - nrows - 1:  

208         # Again accounting for different tile sizes. Some  

209         → duplicate code cannot be avoided here.  

210         if aisle_position < l_aisle - nrows:  

211             events[action_index]['timer'] =  

212                 → passengers[pass_index]['walkspeed']['aisle_begin']  

213         else:  

214             events[action_index]['timer'] =  

215                 → passengers[pass_index]['walkspeed']['aisle_seats']  

216  

217     # Now for something very important: if the passenger  

218     → arrives at the row of his seat he needs to sit!  

219     else:  

220         # Next time this passenger's event occurs, the  

221         → passenger is seated and removed from the event  

222         → list.  

223         # Here we already mark this passenger as seated.  

224         passengers[pass_index]['seated'] = True  

225         res_seating = SeatingManager(passengers[pass_index],  

226             → seating_matrix, current_time)  

227         seating_matrix = res_seating[0]  

228         events[action_index]['timer'] = res_seating[1]  

229  

230  

231 def SimulationExecutor(strategy, nrows, nseats, l_aisle,  

232     → size_aisle_seats, size_aisle_begin, mu_arrival):  

233     # Initialization  

234     current_time = [0]                                # Initializing  

235     → global time. For some reason this cannot simply be an integer,  

236     → otherwise it does not update (took me forever to find this).  

237     aisle = np.zeros(l_aisle)                          # The aisle is a  

238     → binary vector, where '1' designates occupancy and '0' vacancy.  

239     seating_matrix = np.full((nrows, nseats), {})    # Initializing  

240     → seating_matrix

```

```

226 passengers = Queue_former(strategy)                      # Initializing the
227   → passengers according to the boarding strategy.
228 events = [ {'ID': 'Arrival', 'timer' :
229   → random.expoariate(1/mu_arrival)}]      # Initializing the event
230   → list.
231 # At first the only active event is that of the next arrival.
232 # As passengers enter the plane, their next action gets its separate
233   → event.
234
235 # Execution
236 while len(events)>0:                                     # Once everybody is
237   → seated there are no more scheduled events and the simulation is
238   → done.
239   → BoardingSimulator9000(current_time, passengers, aisle, events,
240     → seating_matrix, mu_arrival)
241
242 return seating_matrix
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260

```

```

261     for i in range(nsims):
262         seating_matrix = SimulationExecutor(strat[1], nrows, nseats,
263             ↪ l_aisle, size_aisle_seats, size_aisle_begin, mu_arrival)
264         passenger_list = seating_matrix.reshape(-1)
265         for key in keys:
266             if key not in keys0:
267                 key_list = [p['timespent'][key] for p in
268                             ↪ passenger_list]
269
270                 ↪ time_measures[strat[0]][strat[1]][key].extend(key_list)
271
272                 key_mean = mean(key_list)
273                 key_95percentile = np.percentile(key_list, 95)
274
275                 ↪ time_measures[strat[0]][strat[1]]['mean'][key].extend([key_mean])
276
277                 ↪ time_measures[strat[0]][strat[1]]['95thpercentile'][key].extend(
278                     [key_95percentile])
279
280                 ↪ time_measures[strat[0]][strat[1]]['Total_boarding_time'].append(max(
281                     key = lambda x:
282                         x['timespent']['time_seated'])['timespent']['time_seated'])
283
284         finish = time.time() - start
285         print(f" Running {nsims} simulations of boarding strategy
286             ↪ {strat[1]} was completed in {finish} seconds.")
287
288     return time_measures

```

References

- Bachmat, E., Berend, D., & Sapir, L. (2009a). Analysis of airplane boarding times. *Aerospace*, 2, 34. doi: 10.1287/opre.1080.0630
- Bachmat, E., Berend, D., & Sapir, S., L.and Skiena. (2009b). A dynamically optimized aircraft boarding strategy. *Aerospace*, 58, 7. doi: 10.1016/j.jairtraman.2016.10.010
- Chun, H. W., & Wai Tak Mak, R. (1999). Reducing passenger boarding time in airplanes:a simulation based approach. *Aerospace*, 23, 11. doi: 1094â€¢6977/99
- Ferrari, P., & Nagel, K. (2005). Robustness of efficient passenger boarding strategies for airplanes. *Aerospace*, 1915, 10. doi: 10.3141/1915-06
- Jafer, S., & Mi, W. (2017, 11). Comparative study of aircraft boarding strategies using cellular discrete event simulation. *Aerospace*, 4, 57. doi: 10.3390/aerospace4040057
- Qiang, S., Jia, B., Qingxia, H., & Jiang.R. (2018). Simulation of free boarding process using a cellular automaton model for passenger dynamics. *Aerospace*, 15, 11. doi: 10.1007/s11071-017-3867-5
- Schultz, M. (2008). Implementation and application of a stochastic aircraft boarding model. *Aerospace*, 15, 11. doi: 10.1016/2018.03.016
- Van Landeghem, H., & Beuselinck, A. (2002). Reducing passenger boarding time in airplanes:a simulation based approach. *Aerospace*, 1915, 14. doi: 10.1016/S0377-2217(01)00294-6