

## Summary Sheet

If you have ever been on an aircraft, it is plane to see how slow boarding and disembarking is. For many this is insignificant, but for an airline company saving even a couple of minutes for each flight's boarding and disembarking will result in huge savings when considering the tens of thousands of airports and flights that occur each time. For an industry still struggling from the collapse of the tourism industry due to COVID-19, optimal and robust boarding and disembarking methods must be found.

To achieve this we developed two models, one for each of boarding and disembarking. As boarding and disembarking planes is an inherently stochastic process, we created a computational simulation over a pure mathematical model. Thus, we could better account for variable human behaviours and scenarios, giving a much more accurate distribution of data. Whilst many models already exist for this purpose, a key point of difference of our model is a greater consideration to several aspects of human behaviour. Namely, disobedience of boarding instructions, and travelling in groups.

We first modelled the Narrow Body Aircraft, simulating different boarding and disembarking methods using a Monte Carlo method. To create different boarding methods, we generated a randomized queue of passengers in the order that the boarding method prescribes (accounting for disobedient people) which could then be simulated boarding. Over many simulations, we could obtain an accurate average for the total time taken, allowing us to determine the most optimal method (least time taken). We also proposed two additional methods and ran them through the same simulations.

To simulate disembarking, we gave all seated passengers a priority value. Disembarking was carried out by moving passengers towards the exit at different rates dependent on their priority level. By altering the priority values we could carry out different disembarking methods and account for disobedience.

Both models implemented real-world data for factors such as moving speeds. This was to ensure the highest accuracy of our resulting times. We comprehensively analysed the results of these simulations, determining the effect of altering variables such as the number of people who disobey instructions, and varying numbers of carry-on baggage.

We adapted our models to two other passenger aircraft, the Flying Wing and the Two-Entrance Two-Aisle, and applied the most optimal boarding and disembarking methods used on the Narrow Body plane. Furthermore, we considered the effect of a reduced capacity of the passenger aircraft, a relevant deliberation in the age of COVID-19.

Overall, it was found that for boarding, one of our own proposed methods – boarding in the order of window, middle and aisle seats with the allowance of groups to board together – was on the whole the most optimal over the three aircraft. The optimal disembarking method was one in which the plane was unloaded from the back of the craft to the front.

# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>   | <b>3</b>  |
| 1.1      | Background . . . . .  | 3         |
| 1.2      | Problem Restatement . . . . .                               | 3         |
| 1.3      | Basic Assumptions . . . . .                                 | 3         |
| 1.4      | Variables and Factors . . . . .                             | 4         |
| <b>2</b> | <b>Narrow-Body Boarding</b>                                 | <b>4</b>  |
| 2.1      | Boarding Model Situation . . . . .                          | 4         |
| 2.1.1    | Carry-on Baggage Delay . . . . .                            | 5         |
| 2.1.2    | Shifting Seats Delay . . . . .                              | 7         |
| 2.2      | Boarding Queue Generation . . . . .                         | 7         |
| 2.2.1    | Disobedience Coefficient . . . . .                          | 8         |
| 2.2.2    | Groups of People . . . . .                                  | 8         |
| 2.2.3    | Bag Coefficient . . . . .                                   | 8         |
| 2.3      | Modelled Results for Provided Boarding Methods . . . . .    | 8         |
| 2.3.1    | Random Boarding . . . . .                                   | 9         |
| 2.3.2    | Boarding by Section . . . . .                               | 9         |
| 2.3.3    | Boarding by Seat (WMA/WilMA) . . . . .                      | 10        |
| 2.3.4    | Sensitivity Analysis of Provided Boarding Methods . . . . . | 10        |
| 2.4      | Modelled Results for Other Boarding Methods . . . . .       | 12        |
| 2.4.1    | Modified Steffen Method . . . . .                           | 12        |
| 2.4.2    | Prioritised Groups . . . . .                                | 12        |
| 2.4.3    | Modified Boarding by Seats (WMA) . . . . .                  | 13        |
| 2.5      | Optimal Boarding Method . . . . .                           | 13        |
| <b>3</b> | <b>Narrow-Body Disembarking</b>                             | <b>14</b> |
| 3.1      | Generation of Priority Map . . . . .                        | 14        |
| 3.2      | Logic of Disembarking . . . . .                             | 16        |
| 3.3      | Time to Unstow Bags . . . . .                               | 16        |
| 3.4      | Optimal Disembarking Method . . . . .                       | 16        |
| <b>4</b> | <b>Extension of Model to Other Aircraft</b>                 | <b>18</b> |
| 4.1      | Flying Wing Aircraft . . . . .                              | 18        |
| 4.1.1    | Flying Wing Boarding . . . . .                              | 18        |
| 4.1.2    | Flying Wing Disembarking . . . . .                          | 19        |
| 4.2      | Two-Entrance Two-Aisle Aircraft . . . . .                   | 20        |
| 4.2.1    | Two-Entrance Two-Aisle Boarding . . . . .                   | 20        |
| 4.2.2    | Two-Entrance Two-Aisle Disembarking . . . . .               | 21        |
| <b>5</b> | <b>Pandemic Capacity Decrease</b>                           | <b>21</b> |
| 5.1      | Boarding . . . . .  | 21        |
| 5.2      | Disembarking . . . . .                                      | 22        |
| <b>6</b> | <b>Evaluation of Models</b>                                 | <b>22</b> |
| 6.1      | Strengths . . . . .   | 22        |
| 6.2      | Limitations . . . . .                                       | 22        |
| <b>7</b> | <b>Appendices</b>   | <b>24</b> |

# 1 Introduction

## 1.1 Background

As society becomes increasingly globalised, the importance of air travel grows. Flight numbers before the COVID-19 pandemic were at an all-time high, and they have doubled in the past 20 years. Following a temporary disruption due to COVID-19, this trend appears ready to continue it's steady upwards climb[1]. This has the consequence that small optimisation changes can result in enormous savings for both airline companies, passengers, and airports in terms of usable time wasted. Some of the biggest bottlenecks for plane turnaround are boarding and disembarking efficiency - that is, the way that passengers are loaded to and unloaded from planes[2]. There exist a variety of methods for these processes, each with varying theoretical and practical efficacies. As such, this report presents our developed model and simulates different onboarding and embarking methods for various aircraft models.

## 1.2 Problem Restatement

To ascertain the efficiency of different systems, we will develop two models with allowances for practical considerations that can be adapted to a variety of conditions.

1. Develop a plane boarding model and disembarking model which allows us to test the efficiency of different boarding/disembarking methods on a narrow-body plane
2. Adapt the models to test on different aircraft types (i.e. Flying Wing and Two-Entrance Two-Aisle) and also the effects of limited capacity flights due to COVID-19
3. Write a one-page letter to an airline executive that explains our results and its benefits to their airline

## 1.3 Basic Assumptions

Our initial model uses a few basic assumptions. The aircraft is to be divided into cells which one person can occupy at a time. The aisle space between rows and each seat is represented by one cell.

- Only one person can comfortably walk in an aisle cell

**Justification:** Although aisle width varies by aircraft, a reasonable estimate is 0.50m wide[3]. On average, men have longer shoulder width than women, at 0.41m wide[4] and passengers are often carrying luggage which increases their width requirement. Thus, it is reasonable to assume that only one person can walk down the aisle at a time, with passengers both being laden with bags, respecting personal space, and potentially being weary of close contact due to infection risks. As such, when a passenger is loading their carry-on luggage into an overhead bin, the aisle is also blocked.

- Seated passengers block passengers who wish to sit further down in the same row

**Justification:** The passenger cannot leap over the seated passenger. Not only is this valid from a social etiquette perspective, but in the provided aircraft designs, legroom looks to be minimal so it is physically unfeasible too.

- When a seat passenger leaves a row to make room for an incoming passenger, they are momentarily able to inhabit the same aisle cell

**Justification:** As the passenger will want to reach their seat, they will not mind temporarily having reduced room as they move into their seat cell.

- Time to walk one aisle cell is constant

**Justification:** This time was obtained by analysing a sample of  $n = 10$  YouTube videos of people walking down aisles on flights, by counting the number of frames elapsed when each individual walks one aisle cell, and the playback details of the YouTube videos (typically either 60 or 30 frames per second - these are listed in the references). Using this, we can determine that the time to move one cell down the aisle is given by 1.05s.

## 1.4 Variables and Factors

Several variables were used in our model to account for real-life phenomena. Some of these will be expanded on in later sections.

A **bag coefficient** was used to give a weighted probability of each passenger having carry-on luggage that they would want to stow in an overhead locker.

Another variable was the **number of groups**. Passenger populations are not homogenous; often they contain inseparable groups such as families of varying sizes. Members of these groups were seated adjacently in the same row and entered the plane in adjacent cells too. Upon entering the plane, it was assumed that groups would be in an order that would minimize blockage when getting into seats (i.e. in the order window, middle, aisle). This is reasonable as groups would want to minimize their own inconvenience and could communicate with each other to align themselves in this order. This factor has an appreciable effect on different boarding methods and was rarely investigated with any depth in any of the papers found in our literature review.

A **disobedience coefficient** was introduced to model the common scenario of passengers not following instructions. In these cases, a passenger (or group) would enter the plane in a different boarding category than ordered, which could be caused by ignorance, impatience or lateness. This, much like the **number of groups**, was rarely considered in an in-depth manner in the existing literature but would still significantly affect boarding times.

## 2 Narrow-Body Boarding

For both our models, we simulated the entire boarding/disembarking process. Keeping track of time during this simulation, we could calculate total boarding/disembarking time. Python 3.9 was used for this simulation.

### 2.1 Boarding Model Situation

To model boarding, we designed an algorithm that would see all passengers make their way to their assigned seat. Once on the plane after waiting in the boarding queue, passengers would follow a rigid set of rules, and variation would naturally occur due to variation in input: passengers had randomly generated differing numbers of baggage, and orders in which they entered the plane. Different boarding methods would be accounted for in the order of which passengers in prioritized seats entered the plane. A simplified process of the model as experienced by a passenger is best represented in the flow chart in Fig 2.1. This logic is easily followed and provides a robust algorithm that passengers can follow.

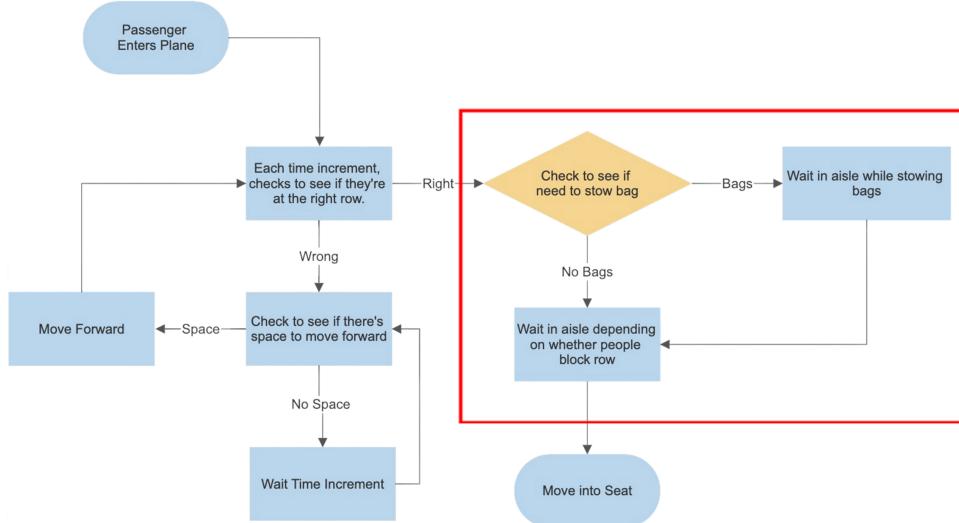


Figure 2.1: The logic behind passenger movement in the narrow-body aircraft

In the model this is simulated for all passengers simultaneously, as any passenger in the aisle could be at any step at any time. This is done by repetitively iterating down the aisle, starting from the passenger furthest from the entrance. Their state is determined, and an action done accordingly. Since it is assumed that there is a steady flow rate into the plane, if the first position in the aisle is ever empty, then the next passenger in the boarding queue occupies this space – ‘passenger enters plane’ in the flow chart. A key part of this simulation is the concept of an **internal clock**. Each passenger has this attribute, which counts down the real time (e.g., 1 sec) until they can complete an action. For example, the time to progress one cell forward is constant. The section of the flow chart enclosed in red is implemented in the simulation by calculating the total time that these actions would take and increasing the passenger’s internal clock until this time is achieved, whereupon they can undertake their action. A visualisation tool was used on the code, allowing us to generate real-time visualisations of the simulations (see Fig 2.2, and the code in Appendix NUMBERHERE).

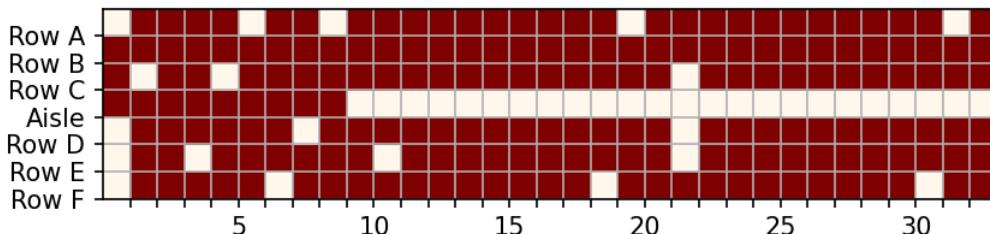


Figure 2.2: Visualisation tool in use on the narrow aircraft. Note that the aisle is currently blocked by a passenger in row 9.

In the following sections, we derive how these times are calculated.

### 2.1.1 Carry-on Baggage Delay

In airplanes it is commonplace that passengers load their carry-on luggage into the overhead bins. The aisle is blocked for the duration of this process. To account for this, the following piecewise function was developed to model the time that each passenger blocks the aisle while loading carry-on bags (which impedes the flow of passengers down the aisle).

| Variable        | Description   |
|-----------------|---|
| $T_{bags}$      | Time that the main aisle is blocked due to carry-on luggage loading |
| $n_{bags}$      | Number of carry-on bags to be stored in the overhead bins           |
| $n_{bins}$      | Number of carry-on bags present in overhead bins before storing     |
| $n_{max}$       | Maximum number of bags overhead bins can hold                       |
| $C_0, C_1, C_2$ | Scaling constants depending on the value of $n_{max}$               |

$$T_{bags}(n_{bags}, n_{bins}, n_{max}) = \begin{cases} 0 & \text{if } n_{bags} = 0 \\ \frac{C_0}{1 - C_1 n_{bins}/n_{max}} & \text{if } n_{bags} = 1 \\ \frac{C_0}{1 - C_1 n_{bins}/n_{max}} + \frac{C_2}{1 - (n_{bins} + 1)/n_{max}} & \text{if } n_{bags} = 2 \end{cases} \quad (1)$$

The model only considers  $n_{bags} \in \{0, 1, 2\}$  since it is assumed that the maximum number of carry-on items that each passenger is permitted to have is  $n_{bags} = 2$ . Many airlines, including Air New Zealand[5], impose this maximum (even for business class passengers). The benefit of this equation is in its generality; its many parameters allow for precise calibration to produce more accurate results, especially for different aircraft models. For the purposes of modelling the narrow plane, we assumed each row of three had an overhead bin with capacity  $n_{max} = 6$  since each passenger in the row could carry at most  $n_{bags} = 2$ . This is assuming not all the stowed items are full size suitcases: some carry-ons are likely to be smaller items such as handbags/tote bags. The passengers will be able to fit more of these into an overhead bin, thus the larger capacity. Then, taking  $n_{max} = 6$ , the values of  $C_0, C_1, C_2$  were calibrated to be 4, 0.8, and 2.25 respectively. This yields the following equation, which was implemented into our model.

$$T_{bags}(n_{bags}, n_{bins}, 6) = \begin{cases} 0 & \text{if } n_{bags} = 0 \\ \frac{4}{1 - 0.8n_{bins}/6} & \text{if } n_{bags} = 1 \\ \frac{4}{1 - 0.8n_{bins}/6} + \frac{2.25}{1 - (n_{bins} + 1)/6} & \text{if } n_{bags} = 2 \end{cases} \quad (2)$$

The function is piecewise to easily account for the varying number of bags that each passenger carries. Passengers carrying no bags do not take time to stow, while those stowing two bags take longer than those stowing one bag (thus the added term). Another consideration is that the function is designed to increase when there is less space in the overhead bin (i.e., when  $n_{bins}/n_{max}$  is large) as passengers will have to find space and squeeze their bags in, increasing aisle blockage time. For instance, if a passenger has one bag and there are already 1/6 bags in the overhead bin, then  $T_{bags}(1, 1, 6) = 4.6$ . However, if the compartment is almost full with 5/6 bags, then  $T_{bags}(1, 5, 6) = 12$  as the passenger will have to locate a space and squeeze their carry-on in.

### 2.1.2 Shifting Seats Delay

Another large source of aisle blockage arises from the common situation where a passenger tries to reach their seat in a row but is blocked by a seated passenger. Before the passenger can reach their seat, the seated individual must stand up and move out to the aisle to allow the passenger to reach their seat, before sliding back. This process is lengthy and will impede the flow of passengers down the main aisle. This is furthermore complicated by the fact that there are many variations on this scenario, with different seated passenger positions and passenger seat goals, which will have appreciably different delay times. To model the additional time needed for these different shuffles, Eq. 3 was derived.

| Variable      | Description   |
|---------------|---|
| $T_{shuffle}$ | Time that the main aisle is blocked                             |
| $t_{up}$      | Time taken for a seated passenger to stand up                   |
| $t_s$         | Time taken for a passenger to travel the width of a seat        |
| $f$           | The index of the furthest seat that blocks the passenger's seat |
| $n_s$         | The number of seated passengers that block the passenger's seat |

Let the seats be indexed such that the aisle seat has index 1 and the index of each consecutive seat increases until the window seat. Since we are only concerned with total time that the aisle is blocked, only the time that passengers are occupying the aisle needs to be kept track of. First, the person seated furthest from the aisle stands and moves into the aisle ( $t_{up} + ft_s$ ). Then the passenger moves into the row ( $t_s$ ), and finally the previously seated passengers move back into the row ( $n_s t_s$ ).

$$\begin{aligned} T_{shuffle}(f, n_s) &= t_{up} + ft_s + t_s + n_s t_s \\ &= t_{up} + t_s(f + 1 + n_s) \end{aligned} \quad (3)$$

Following this derivation, we state that the equation makes the following assumptions:

- The seated passengers notice the passenger once they are standing next to the row
- All the required seated passengers stand up at the same time and begin to exit the row
- That two people can inhabit the aisle cell adjacent to the row (assumed earlier)
- Once the passenger has entered the row, the previously seated passengers begin moving back into the aisle, following right behind the passenger in the correct order

These assumptions are sufficiently realistic to generate results which closely model reality.

## 2.2 Boarding Queue Generation

A queue of passengers with assigned seats was generated to move into the aisle. By altering the order of the passengers in this queue, we could simulate different boarding methods. For example, we could place everyone in the queue in order of aft, middle, front. Within these sub-sections of the queue, the order was randomized each trial to further increase realism. At this point, we also assigned each passenger a discrete number of baggage, either 0, 1, or 2. This was done by utilizing a weighted probability. Overall, we implemented algorithms to create boarding queues for all the required boarding methods, as well as several others. However, to increase realism of the model, we added additional variation within these.

### 2.2.1 Disobedience Coefficient

Undoubtedly, there will be passengers who do not follow the rules of whichever boarding method is in place. This is due to two main reasons: impatience (boarding before they are called), and lateness (being late to their boarding time). These passengers are rarely accounted for in the literature, yet they have an appreciable effect on boarding times. To include this in our model, we introduced the **disobedience coefficient**,  $\psi$ , the probability of any passenger in the queue to not follow the desired boarding method. For instance, in a sectional boarding method, a passenger sitting in the aft section of the plane would have a  $\psi$  chance of boarding with a different group (and given that they do, a 50% chance for either group). Initially this was fixed at  $\psi = 0.3$ ; online studies found that 30% of passengers are late for their flights, and we thought that this was a reasonable number that would be impatient as well.

### 2.2.2 Groups of People

Another important consideration in the model is the existence of groups of people that board together. Families, couples, and the like are present in high concentration on flights and are often seated together. Importantly and as discussed previously, they board together and enter the queue in the way in which they would enter seat rows, decreasing total boarding time. To account for this in our model, when a passenger in queue is generated, there is a weighted probability that they will be in a group of 1, 2 or 3. Groups of 1 are simply regular passengers. Groups of 2 or 3 are adjacent in the boarding queue and are seated in adjacent cells. Groups of 4 or larger were excluded since the aircraft only allowed a maximum of 3 to sit together in a row, effectively meaning a group above 3 can be split into two groups. Initially, the weighted probabilities of a passenger being in a group of 1, 2 or 3 was set at (20,80,10).

We also considered the effect of the disobedience coefficient on groups. We initially considered a group to be disobedient if any members of the group of size  $n$  were disobedient. However, as  $(1 - \psi)$  is the probability that a passenger is obedient, then  $(1 - \psi)^n$  is the probability that the entire group is obedient. Hence,  $1 - (1 - \psi)^n$  is the probability that the group would be disobedient. For a  $\psi$  value of 0.3, this would create a disobedience probability of 0.51 for groups of 2 and 0.657 for groups of 3. We thought that this was unrealistically high, and instead determined that the disobedience probability would be  $\psi$  for the entire group.

### 2.2.3 Bag Coefficient

A key stochastic variable in this model is the number of carry-on bags that any given passenger will stow in the overhead lockers. Just as in real plane boarding, this is clearly prone to variation. To account for this, we introduced another 3-tuple in the code to give a weighted probability of a passenger stowing either 0, 1 or 2 bags. Unfortunately, there was a lack of available data on average passenger bag count online. As such, further analysis of the previous YouTube videos allowed us to tentatively obtain an estimate of (20,80,10). However, in the sensitivity analyses later this value was changed appropriately, allowing us to determine the validity of this initial assumption.

## 2.3 Modelled Results for Provided Boarding Methods

The three provided methods for boarding were random boarding, boarding by section, and boarding by seat. It was assumed that boarding by seat would make no allowances for groups of people. However, the other methods were modelled using groups.

### 2.3.1 Random Boarding

At first glance, the method of random boarding seems crude and inefficient. However, simulations run on our model reveal that the random method is reasonably effective. It took on average 689.4 seconds to finish boarding the plane, with a 5th percentile of 626.7s and a 95th percentile of 755.7s. This means that 90% of the values fall in this range of 129 seconds.

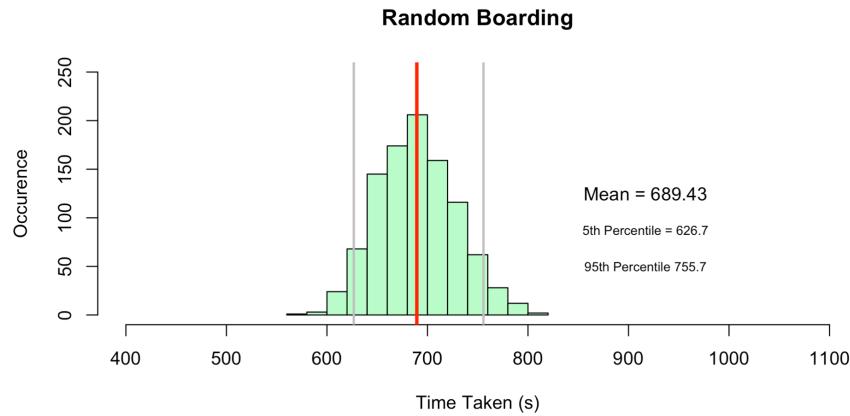


Figure 2.3: Monte Carlo simulation graph of the random boarding method

### 2.3.2 Boarding by Section

The second supplied method was to board the plane in sections. Boarding by aft (rows 23-33), middle (12-22) and front (rows 1-11) sections in varying order produced different results in our model. A set of results for all possible variations can be seen in the bar chart in Fig 6.1, but we discuss only the most and least optimal methods.

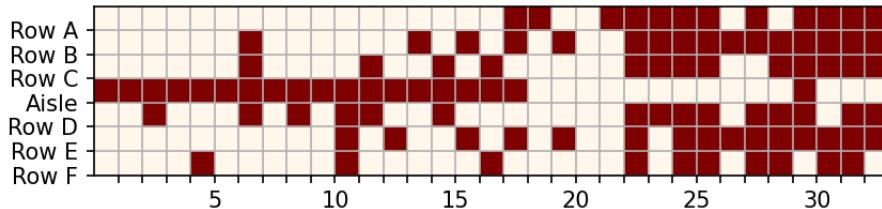


Figure 2.4: Visualised boarding by section starting with the aft. Note the disobedient passengers who have already seated themselves in the front and middle sections.

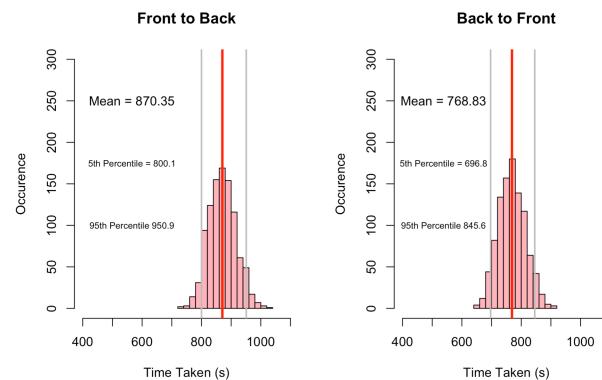


Figure 2.5: Monte Carlo simulation graphs of boarding front, middle, aft and aft, middle front

After running 10,000 trials, we found that the most optimal order of boarding was aft, middle, front (see breakdown in Appendix A). The mean time taken to fill up the narrow body airplane was **768.8 seconds**, with 90% of the times falling between 696.8s and 845.6s (spread of 148.8s). In comparison to this, it took on average **870.4 seconds** to board using the front, middle, aft method, with 90% of the times between 800.1s and 950.9s (spread of 150.8 seconds). This difference can be explained by considering Fig 2.6. On the left visualisation, the back fills first and so there is room to queue in the aisle, while on the right when the front is boarded first, the queue extends outside of the plane. Interestingly, this common method for boarding the plane is actually significantly slower than a random boarding order. However, the ability to simplistically split boarding into groups of people is valuable for airline companies, as it provides structure as to who should line up when. In the random boarding method, everyone is called to line up at once. This may potentially cause large queues and waste passengers time queuing in a long line.

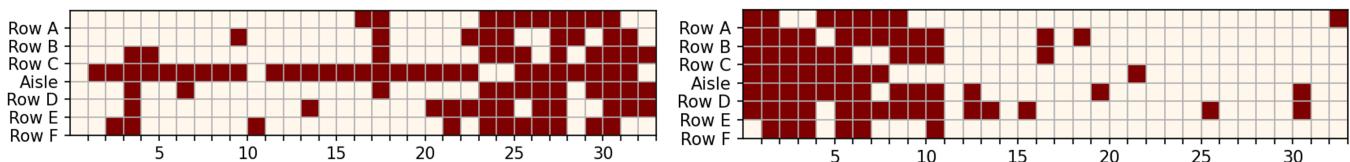


Figure 2.6: Visualisation of boarding by section, with AMF on the left and FMA on the right. Note the disobedience passengers sitting in the incorrect sections.

### 2.3.3 Boarding by Seat (WMA/WilMA)

The plane can also be boarded by seat type. This method allows all passengers with a window seat to board first, then middle, and finally aisle seats. Initially it seems like an ideal boarding method as it is relatively fast, with a mean boarding time 519.1 seconds. It is consistent too, with 90% of the values within 85 seconds of each other (5th percentile 479.1s, 95th percentile 564.1s). Not only this, but it is also straightforward to implement, with 3 easily definable groups of passengers. However, it splits groups. This is effectively unworkable in practice due to the separation of groups, particularly in the case of children and elderly.

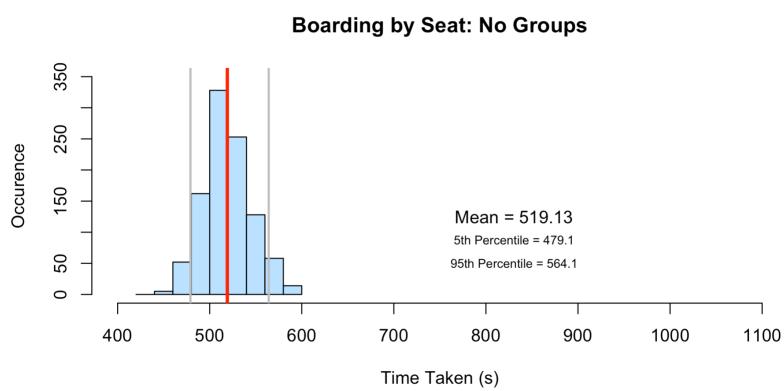


Figure 2.7: Monte Carlo simulation graph of boarding by seat without groups

### 2.3.4 Sensitivity Analysis of Provided Boarding Methods

We now perform a sensitivity analysis on the provided boarding methods.

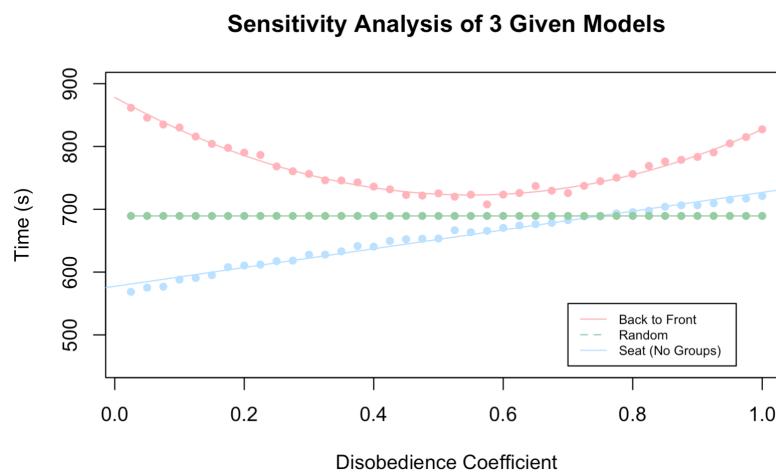


Figure 2.8: Sensitivity analysis of the disobedience coefficient on the interval  $0 \leq \psi \leq 1$

Fig 2.8 shows the impact of changing the disobedience coefficient on the time taken to board, for the three given models. The effect of changing the disobedience coefficient for the section boarding was most interesting. As the number of people not following the prescribed method increased, the boarding method trended towards random. This meant the time taken decreased as random boarding is faster than section boarding. At a disobedience coefficient of  $\psi = 0.5$ , the boarding method is effectively random, thus the times are equivalent. However, as more people decide not to board with their prescribed group, the time starts to increase again. This is due to the boarding becoming ‘ordered’ again by section, which is slower than a random boarding method. This behaviour from the boarding by section method is ideal for airline companies, as a realistic extent of disobedience will help their boarding times. The random boarding method is completely insensitive to changes in disobedience, as there are no rules to disobey. The boarding by seat method without groups is the fastest boarding method provided, but it is also the method most impacted by changes in disobedience. This is potentially undesirable behaviour in a boarding method for airline companies, however under all reasonable values of the disobedience coefficient, boarding by seat is the fastest boarding method.

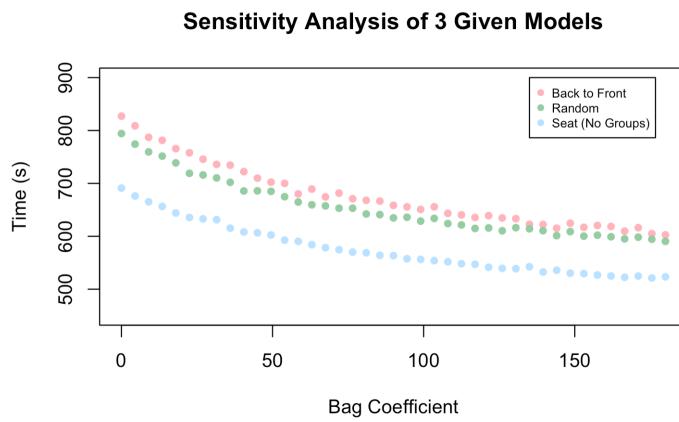


Figure 2.9: Sensitivity analysis of provided boarding methods by scaling a part of the bag coefficient.

Changing the bag coefficient changes the number of people without bags. The higher the coefficient, the higher the number of people without bags. The relevant time relating to bag numbers is the time spent in the aisle stowing. As such, only the number of bags stowed is pertinent to this model. Therefore, our variation of the bag coefficient (integers from 0 to 180) is effective at describing the impact of all plausible variations in bag numbers and bag stowage on the time take to board an

aircraft. From this analysis, we found that the three recommended methods are of equal sensitivity to variations in the bag coefficient. This is shown by the identical shape of the curves.

## 2.4 Modelled Results for Other Boarding Methods

### 2.4.1 Modified Steffen Method

The Steffen method is a plane boarding method proposed by Jason Steffen in 2008 which is suggested to be the method that produces the optimal plane boarding time[6]. However, this method is highly theoretical. It relies on the unrealistic assumption that passengers are efficient and highly organised. Instead, we present the modified version of the Steffen method which has a slightly larger grounding in reality. This method boards even numbered rows on the right hand side, then even rows on the left, then odd rows on the right, to odd rows on the left. This was almost the fastest boarding method we tested, with a mean time to board of 647.05 seconds. The 5th percentile was 595.3 seconds, and the 95th percentile was 696.5 seconds (a spread of 101.2 seconds).

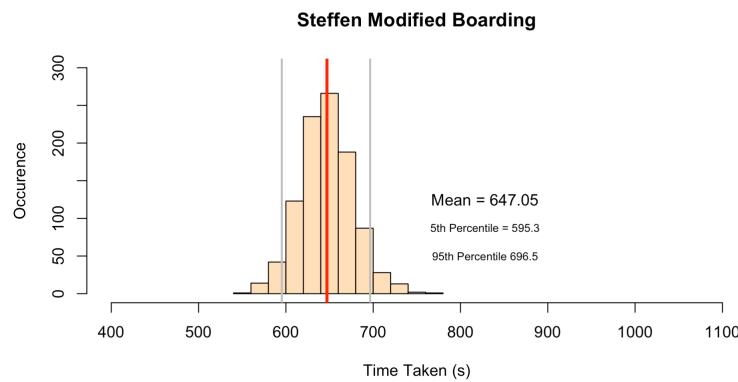


Figure 2.10: Monte Carlo simulation graph of the modified Steffen method

### 2.4.2 Prioritised Groups

In this method, passengers are classified as having one of two classes of walking speeds: normal and slow. This removes the need for our initial assumption that walking speed is relatively constant and allows us to test the validity of this assumption. Many airlines allow prioritised groups such as families with young children, disabled and elderly people to board first. The passengers in these prioritised groups are classified as having slow walking speed. We run this method through our model to determine its efficacy.

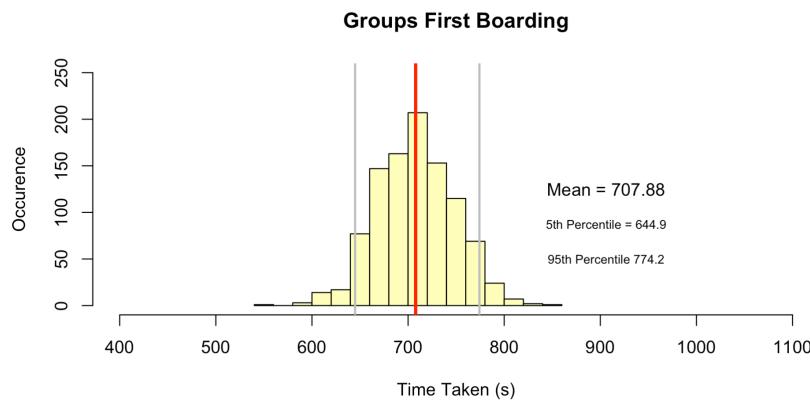


Figure 2.11: Monte Carlo simulation graph of the prioritised group boarding method

### 2.4.3 Modified Boarding by Seats (WMA)

As mentioned, the WMA has some serious drawbacks, particularly in regard to the splitting of groups. To overcome this, we devised a modified WMA method, which is one of our additional boarding methods. In this seating method, window seats are boarded first. However, if someone with a window seat is also part of a group, that whole group will board. The same thing occurs for middle seats and aisle seats. This avoids the problem of splitting groups while maintaining some of the efficiency of the WMA method.

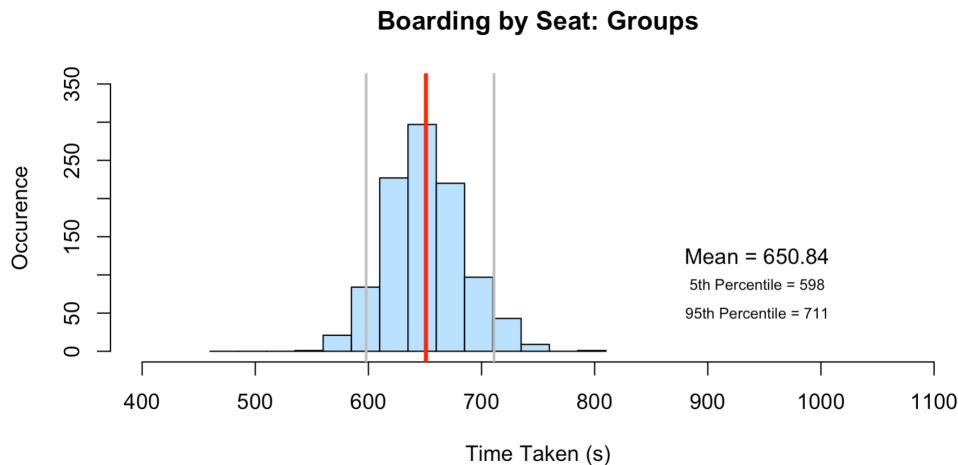


Figure 2.12: Monte Carlo simulation graph of the modified boarding by seats (WMA) method

The mean boarding time we obtained from this method was 650.84 seconds, with a 5th percentile of 598s and a 95th percentile of 711s (spread of 113 seconds). This adjusted method is relatively novel and hasn't seen much discussion in literature. However, its unique combination of practical and theoretical efficiency makes it an attractive proposition.

## 2.5 Optimal Boarding Method

After analysis of the previous five methods, we conclude that the modified WMA method is the best. The mean time to board the narrow body plane after 1,000 trials is 650.84 seconds. It should be noted that this isn't the optimal time that was achieved; the modified Steffen took only 647.05 seconds, and WMA without groups took 519.13s. This data is summarised in Fig 2.13. However, the modified WMA is significantly more practical to implement than both. The modified Steffen requires an unrealistic degree of coordination from random passengers and WMA without groups has the unrealistic assumption of splitting families and other groups apart. The modified WMA method allows for groups and can be easily implemented by airlines (by just calling seat letters to board, including family groups). It is also less sensitive to changes in the disobedience coefficient than alternative methods, such the Steffen modified. Although the time to board is initially slightly faster in the Steffen modified, as the disobedience coefficient increases, the time to board from the Steffen method increases faster than the time to board from the modified WMA. This is advantageous, as it means there is likely less variation in this modified WMA model in comparison to similarly fast boarding methods, allowing airline companies to better predict the boarding times.

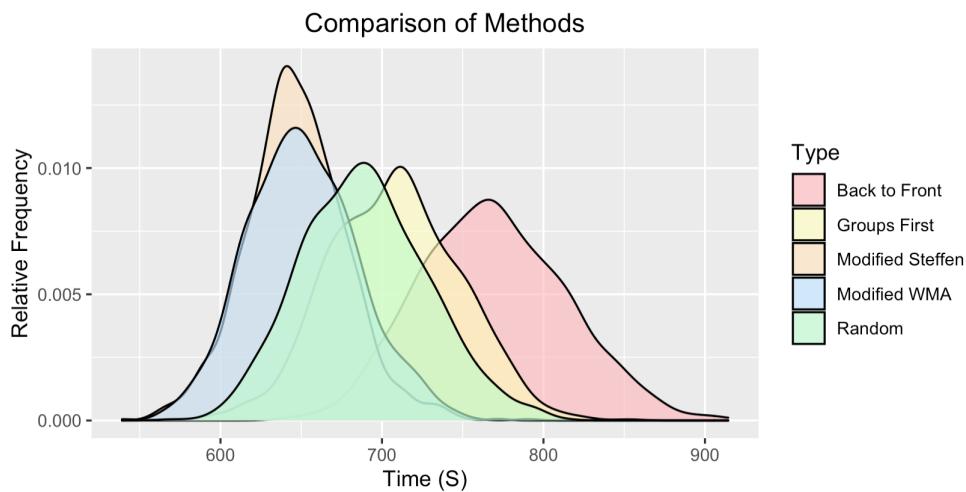


Figure 2.13: Comparison of Monte Carlo simulation graphs of different boarding models featured in previous sections

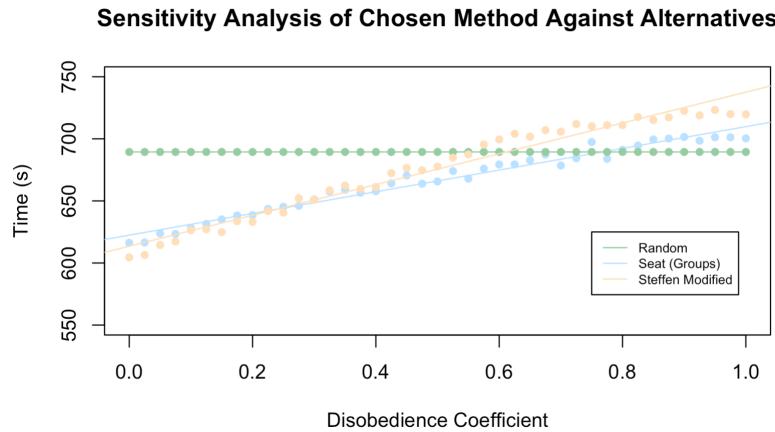


Figure 2.14: Sensitivity analysis of chosen methods for disobedience coefficient

### 3 Narrow-Body Disembarking

Having run simulations on our model under different boarding methods, we now turn our attention to the problem of disembarking. When exiting a plane, people typically move towards the nearest exit whenever space becomes available. A simulation of this is the basis of our disembarking model. By modelling individual interactions, such as what happens when two people come into the same space, we were able to ensure that our model was true as possible to a real disembarking.

#### 3.1 Generation of Priority Map

The disembarking model runs through the generation of a priority map. Each person/group is assigned a priority value, representing how much they want to leave the plane. This is realistic since some people are desperate to leave and others being happy to sit on the plane until the rush dies down. This value is used when there is a passenger interaction. The priority values of each passenger that can move into the square are compared, and the passenger with highest priority is given the right of way. This map can also be manipulated to get different disembarking methods. By giving the highest priority to passengers we want to leave first, we can manipulate the order of who leaves first to find an optimal disembarking method. As such, different methods call for different priority

maps. The creation of the priority maps begins with the creation of an ideal priority map. In this map everyone would be assigned values such that they'd disembark in the desired fashion. Fig 3.1 shows an ideal priority map for disembarking by row, back to front, in the narrow body aircraft.

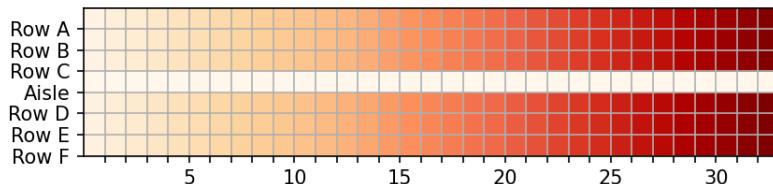


Figure 3.1: Ideal priority heatmap of back to front disembarking

In practice it is highly unlikely that everyone would follow a perfect disembarking model and therefore a disobedience coefficient was implemented, similar to the boarding model. The value of the disobedience coefficient was increased from 0.3 (in the boarding model) to 0.4 in this disembarking model. This choice was based on the fact that people are more likely to be tired, and may just want to leave the plane as soon as possible following a long flight. There is no feasible way to obtain data for this particular coefficient, and to investigate the effect this coefficient has on boarding times we performed a sensitivity analysis, varying the disobedience coefficient. Like in the boarding model, the disobedience coefficient describes the chance that a particular person won't follow their prescribed disembarking method. These disobedient people are then randomly assigned a new priority value ranging from 1 to the maximum possible priority value which varies depending on method. An implementation of this on the previously given priority map can be seen in Fig 3.2.

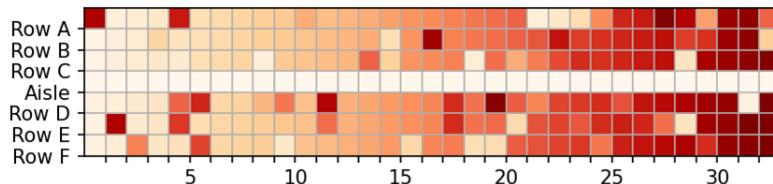


Figure 3.2: Introduction of disobedience coefficient to the ideal heatmap in Fig 3.1

As in the boarding method, we accounted for the fact that many people travel in groups that cannot be split. To implement this in the model, the priority of a group of size  $n$  is set to the mean of each member's priority in that group like so:  $P_{group} = \frac{1}{n} \sum_{i=1}^n P_i$ . The effect of this can be seen in Fig 3.3. Note the group in row 32 (seats ABC).

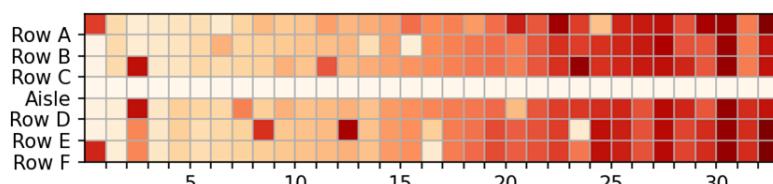


Figure 3.3: Introduction of groups to the heatmap in Fig 3.3

### 3.2 Logic of Disembarking

The diagram to the right shows the logic of the disembarking. By looping through the unoccupied aisle spots, and moving individuals into them, we can simulate the whole moving out. We considered the movement in and out of aisles as well.

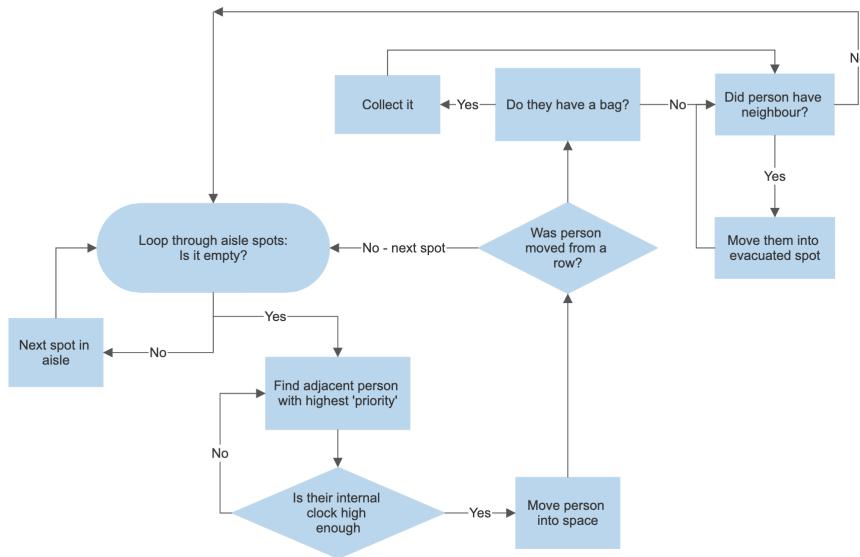


Figure 3.4: A flow diagram of the disembarking algorithm from a passenger's perspective

### 3.3 Time to Unstow Bags

Just as stowing bags during boarding blocks the aisle, the act of unstowing bags during disembarking blocks the aisle too. The following formula is a variant of Eq 1 that simply changes  $n'_{bins} = n_{bins} - 2$ . This is done to avoid division by zero, since many overhead bins will have  $n_{bags} = 6$  as they are full. Eq 1 accounts for the number of bags already in bins – it takes longer difficult to remove a bag out of a packed luggage bin than an empty one. Note this  $n'_{bins}$  is simply labelled  $n_{bins}$  in Eq 1.

$$T_{bags}(n_{bags}, n_{bins}, 6) = \begin{cases} 0 & \text{if } n_{bags} = 0 \\ \frac{4}{1 - 0.8(n_{bins} - 1)/6} & \text{if } n_{bags} = 1 \\ \frac{4}{1 - 0.8(n_{bins} - 2)/6} + \frac{2.25}{1 - (n_{bins} - 1)/6} & \text{if } n_{bags} = 2 \end{cases} \quad (4)$$

### 3.4 Optimal Disembarking Method

The optimal disembarking method for the narrow body aircraft was found to be disembarking from back to front by row. This was initially surprising. However further analysis suggested it to be the quickest due to it having the greatest aisle flow out of all methods. The rate of free aisle flow hindered by retrieving baggage determines the rate people can enter the aisles and hence leave the plane. Back to front results in the greatest aisle flow due to people feeding into the aisles from the back of the plane. Should they need to retrieve a bag, they **a)** hold very few people up as they are at near the end of the queue, and therefore hold very few people up and **b)** by them stopping, they allow people in front of them flow into the queue meaning no gaps are left open.

This is opposite to the ‘front to back’ boarding method which is employed by most airlines and is the slowest disembarking method. This is because when someone enters the aisle from the front

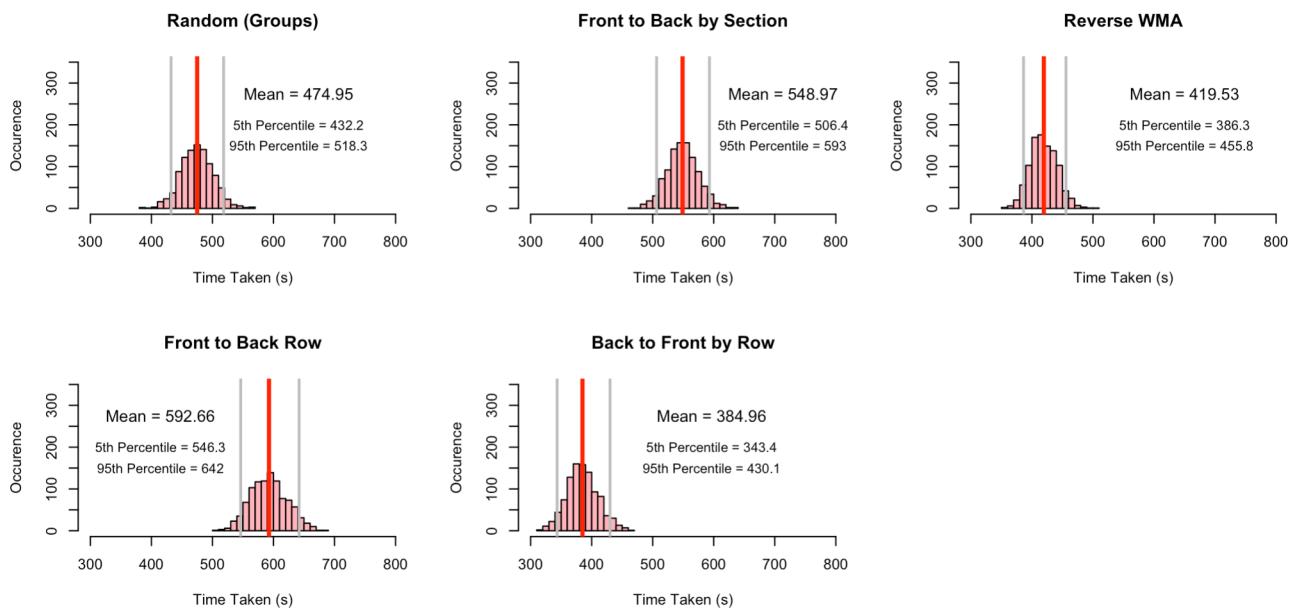


Figure 3.5: Monte Carlo simulation graphs of various disembarking methods (1k trials)

of the aircraft and retrieve their bag they hold the whole queue up whilst not allowing anyone in front of them to enter the queue as there is no one else in front. This back to front system would rely on a ‘right of way’ approach to disembarking the plane, where people at the back have the highest priority. When there is a space that two people could move into, the passenger at the front would have to give way to the passenger coming from behind them.

Other notable disembarking methods were the Reversed WMA which grants priority to aisle-seat passengers followed by middle then window seat. The Reversed WMA produces slower time than Back to Front as more people enter the queue near the front of the aircraft and thus block the queue as they retrieve bags. Reversed WMA is also impractical to implement as it requires a large degree of coordination in comparison to the relatively simple Back to Front method which is a reverse of the commonly used Front to Back Disembarking.

| Disobedience Coefficient | Reverse WMA | Back to Front | Random |
|--------------------------|-------------|---------------|--------|
| 0                        | 393         | 231           | 474.3  |
| 0.2                      | 421.7       | 355           | 474.3  |
| 0.4                      | 438.2       | 404.2         | 476.1  |
| 0.6                      | 459.4       | 435.7         | 474.9  |
| 0.8                      | 470.2       | 451           | 474.2  |
| 1.0                      | 474.9       | 474.1         | 474.7  |

Table 1: Sensitivity analysis of disembarking methods by scaling the disobedience coefficient

This table shows that as disobedience increases, the time taken to disembark decreases. At no disobedience, we get fast disembarking times for reverse Wilma and back to front and at the maximum disobedience we see the times being similar to a random boarding time. Importantly this table also reveals that these models are very sensitive to disobedience especially back to front between 0 – 0.4. It is important to note that despite back to front disobedience sensitive nature it still remains the quickest at the assumed disobedience coefficient of 0.4. This also suggests the importance of airlines employing methods to increase obedience when disembarking as a 20% reduction in disobedience could cause up to two minutes in extreme cases.

| People Not Retrieving Bags | Reverse WMA | Back to Front | Random |
|----------------------------|-------------|---------------|--------|
| 0.2                        | 411.9       | 374.7         | 468.2  |
| 0.4                        | 352.7       | 325.3         | 352.7  |
| 0.6                        | 289.9       | 261.0         | 309.8  |
| 0.8                        | 234.1       | 215.7         | 235.6  |
| 1.0                        | 200.1       | 200.9         | 200.0  |

Table 2: Sensitivity analysis of disembarking methods by changing people not retrieving bags

Table 2 shows a steady trend where the boarding time decreases and tends towards a constant time of 200s as the people not retrieving bags increases. This trend is important for two reasons. Firstly, it shows that if airlines could reduce the amount of bags carried it would result in much faster disembarking times, to the point it would no longer matter which disembarking method was employed. This is because less bags mean the aisle is blocked for a reduced amount of time. Even a minor increase in people not taking bags, for example from 40% to 60%, would result in a drastic reduction in disembarking time of 30s. This could be achieved by encouraging passengers to retrieve their bag in the period between when the plane lands and the disembarking process begins thus increasing the amount of people not retrieving during the disembarking.

## 4 Extension of Model to Other Aircraft

### 4.1 Flying Wing Aircraft

#### 4.1.1 Flying Wing Boarding

The Flying Wing Aircraft has a revolutionary seating plan with an additional 3 aisles and 18 seats across, but only 14 rows. To account for this, we built upon the core algorithm of the narrow body in which passengers walk down the aisle, by simulating all 4 aisles at once, with an additional aisle connecting all of these at the top from the entrance. We initially considered simulating only one aisle and simply quartering the flow rate into the aisle. However, this is not realistic as the top aisle can still be blocked – for example, consider the case where a passenger is stowing their luggage in row 1 of the first aisle, whilst a passenger behind them waits to get into this aisle. Keeping with the assumption that only one passenger can fit into an aisle, such a scenario would block passengers from accessing all other aisles, increasing total boarding time. Thus, we must simulate all aisles boarding at once. Furthermore, although the number of aisles in this plane may cause confusion about where to go, we assumed that this would already be accounted for by the presence of flight attendants, causing no passengers to walk down the wrong aisle. The extended algorithm as experienced by a passenger is represented in the flow chart. A visualization of this model nearing completion is also displayed. Note: the top aisle is not included in this visualisation.

Different boarding models can be applied to the flying wing aircraft to different effect. Random and sectional boarding are relatively easily implemented, and both would be theoretically and practically effective. However, our optimal boarding method for the narrow body aircraft, the WMA method, is now rendered impractical to implement. When considering a seat block between two aisles, where ‘A’, ‘M’, and ‘W’ represent aisle, middle, and window seats respectively. Translating into rows six seats wide, you get the pattern A—M—W—W—M—A. It would be impractical for passengers to judge whether their seat is designated as A, M or W, even without incorporating groups.

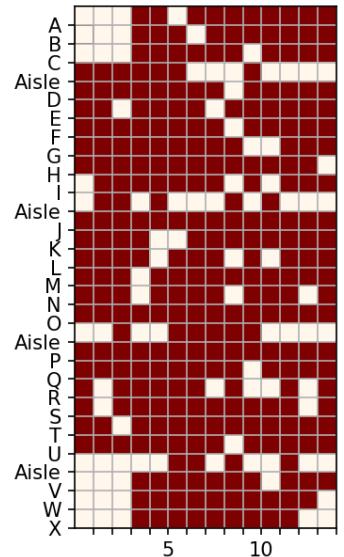


Figure 4.1: Flying Wing model

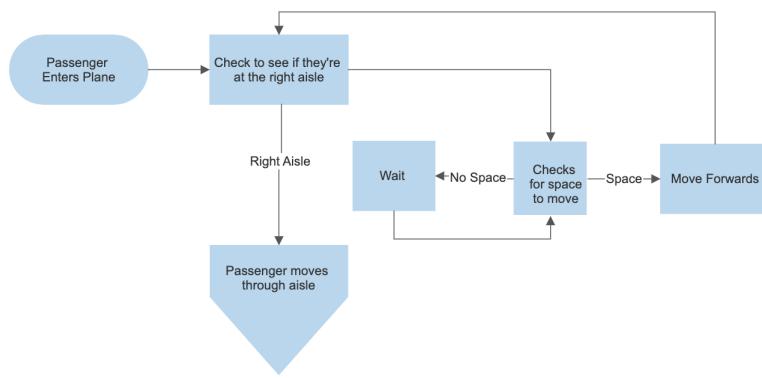


Figure 4.2: Flow diagram of passenger movement logic for the flying wing aircraft

Another potential boarding method we could adapt to a wide wing aircraft would be the modified Steffen method. However, given the established impracticality of the modified Steffen method on the narrow body, this would be even less realistic to expect passengers to follow it when the aircraft is boasting multiple aisles. Thus, we disregarded this method for this plane type. This left us with two viable boarding methods for the wing plane. These are shown in Fig 4.3, along with adjusted WMA times. The mean result times from these boarding methods were 593.2 seconds for the back to front time, 558.9 seconds for the random boarding, and 546.1 seconds for the adjusted WMA boarding time. Despite this, the optimal boarding method is the section boarding, from back to front. Despite having the lowest times, the impracticality of other solutions makes it the most attractive. The closest in terms of overall effectiveness would be random boarding. However, the organisational issues of trying to queue all the passengers in a random order with resulting in excessively large queues would more than outweigh the mere 34.3 second boarding time advantage.

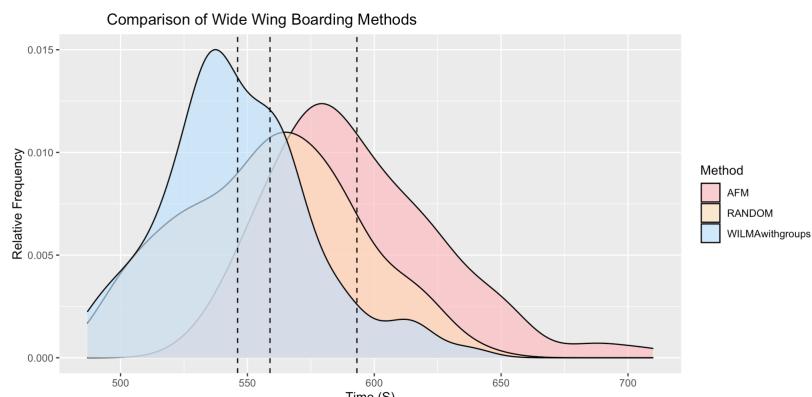
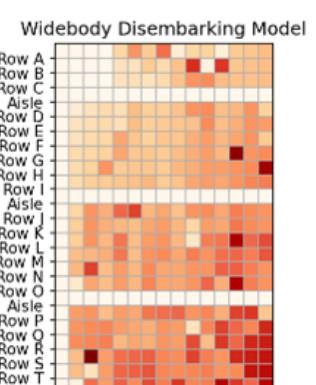


Figure 4.3: Comparison of distribution of times for different methods on the Flying Wing aircraft.

Note that the mean lines for AFM, random and WMA are located at 546.1, 558.9 and 593.2 respectively.

#### 4.1.2 Flying Wing Disembarking

The core logic of the priority disembarking model remains the same when adapted to the Flying Wing Aircraft. The plane is broken into four sub-aisles that passengers are able to move into, dependent on the priority logic. Additionally a leaving aisle has been added that runs by all of the sub aisles. Passengers are moved into and along this leaving aisle to the exit through the use of the priority logic.



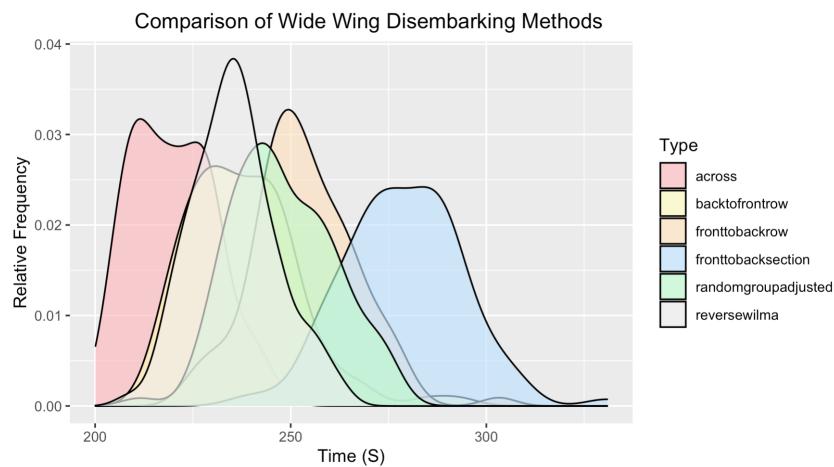


Figure 4.5: Comparison of disembarking methods on the flying wing aircraft

Similarly to the narrow body disembarking, models that prioritise aisle flow will be the fastest in the flying wing aircraft. The most successful model was the across method in Fig 4.4. The across method increased the priority assigned to passengers the further they are from the door, with respect to row and seat. This is in comparison to the ‘front to back’ disembarking method that only increases priority across the row. The across method of distributing priority resulted in the quickest disembarking times as it systematically emptied the aircraft from the bottom right to top left of the diagram (with the exit in the top left). This allowed most people to enter the aisle when they are few people behind them (as the people behind have already left). This means there are minimal aisle blockages when passengers retrieve bags, and passengers in front of the blockage will be able to move into the aisle. The across method is quite practical as it just requires for people to wait for the person behind to leave, or the person behind to retrieve a bag hence allowing them to leave. Similarly to other disembarking models, this is a form of ‘right of way system’, where people furthest from the door have right of way.

## 4.2 Two-Entrance Two-Aisle Aircraft

### 4.2.1 Two-Entrance Two-Aisle Boarding

The two-entrance two-aisle aircraft adds the complexity of multiple entrances, as well as a first-class section. However, we made the following assumptions:

- The first-class section would board first, as is standard across airlines. Any late passengers would interfere negligibly with the rest of the boarding process, as they do not walk down the same aisles as the rest of the passengers
- Rows 12 to 26 (and first-class) would board from the front entrance, whereas rows 27 to 47 would board from the back
- All passengers would board from the correct entrance. In a similar reasoning to everybody walking down the right aisle, we assumed that a plane of this size – and especially one with first class – would have sufficient flight attendants to ensure that this did not happen.

Given these assumptions, a valid simplification can be made to the model: the total boarding time would simply be the time taken to board first class, added to the greatest boarding time of the two sections (seats accessed from entrance 1 and seats accessed from entrance 2) of the plane. Boarding

methods in first class are unreasonable to implement in real life given the smaller number of seats, but also the easier access to seats due to greater space. Thus, this time was calculated using the random boarding method with increased speeds for walking and stowing luggage. The average was found to be We again tested this plane with unstructured (random) and sectional methods and our proposed method of Wilma with groups (in this case, the middle seat does not exist). Once again, Wilma with groups was found to be the most effective method.

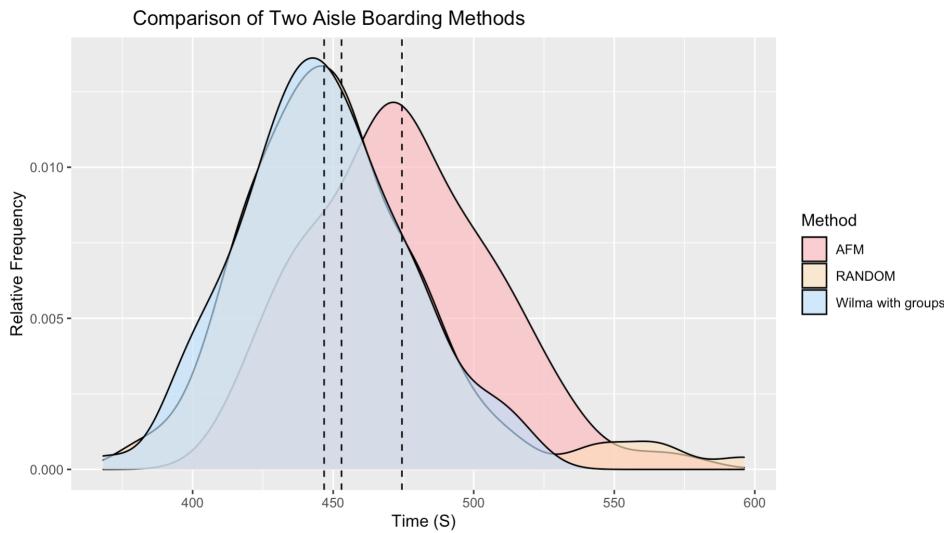


Figure 4.6: Comparison of boarding methods on the two-entrance two-aisle aircraft

#### 4.2.2 Two-Entrance Two-Aisle Disembarking

The assumptions made in the boarding model of Two Entrance Two Aisle can be carried across to the disembarking model. Consequently, simulation were run on both halves of the plane for disembarking and the times were added to the time taken to board first class. Although disembarking occurs on two aisles in this model, aisle flow is still valued and therefore models that increase aisle flow such as back to front is still fastest with a mean time of 180s.

## 5 Pandemic Capacity Decrease

The COVID-19 pandemic has introduced many additional barriers to air travel. Notably, the passenger capacity of aircrafts is forced to decrease to help combat the spread of the disease. We test and present the effect on embarking and disembarking the three aircraft types when passenger capacity is limited to 70%, 50% and 30%. We ran 1000 test cases for all aircraft at all capacity levels with the three most optimal methods, the averages for which can be found in the appendix.

### 5.1 Boarding

An important consideration is that it is not simply random what tickets are not for sale on an aircraft with reduced capacity. Instead, they are chosen to maximise social distancing. For capacity  $c$  each row with numbers of seats  $s$ , we allowed a maximum of  $\lceil c \times s \rceil$  seats to be filled in that row (where  $\lceil x \rceil$  is the ceiling function), and reduced passengers randomly from there on until the number of passengers reached  $c$ . However, groups are still allowed to sit with each other. An analysis of the data would suggest that for both the Narrow Body and Flying Wing aircraft, the Wilma with groups method remains preferable up until a capacity of 50%. At this value its efficiency is only marginal. However, beyond this it becomes optimal to board by section (aft then middle then front). This

can be explained in practice, as at lower numbers of passengers, there is lesser chance of someone blocking the way to a seat – the main problem with sectional boarding. Therefore, without this problem, filling up from the back allows the most passengers to enter the queue at once, resulting in being more optimal. For the Two Entrance Two Aisle aircraft, sectional boarding also quickly becomes the most This method has the added benefit of splitting passengers into boarding groups that also sit together, meaning that although contact cannot be completely avoided, it is minimized to be with the same people. If the pandemic is at the point that capacities of 50% or below need to be enforced, then this is a valuable aspect. Therefore, we would recommend this method for capacities of 50% and below on all aircraft. For capacities of 70% and 100%, the original recommended method remains the most optimal (this is still sectional for the Two Entrance Two Aisle aircraft).

## 5.2 Disembarking

To model reduced capacity of disembarking, we assumed a similar dispersal of passengers as in boarding. For all three aircraft, back to front remained the quickest method of disembarking no matter the capacity. Unfortunately, this does not preserve the social distancing between differing sections of the plane as achieved by boarding. However, at low capacities, disembarking times between different methods became exceedingly quick (under 2 minutes) and closer together. It would be no huge cost to the aircraft to favour a slower method at these capacities. Thus, at low capacities which aim to contain Covid, we would recommend a front to back method. Although this has not been modelled, by extrapolating data for modelled methods, it is clear that this would still be done in a tight timeframe.

# 6 Evaluation of Models

## 6.1 Strengths

- Adaptable. Not only can our models be used for many different boarding and disembarking methods, but our models can be adapted to wide range of plane shapes and sizes, that could consist of multiple aisles or entrances, with relative ease.
- Comprehensive. Our models take into account a wide range of factors affecting boarding and disembarking times, such as people moving past each other within rows, or time taken to stow and retrieve luggage. These times are calculated using real life data, ensuring the highest accuracy.
- Realistic. Many online models may have the strengths above, but fail to account for many common behaviours, such as people disobeying boarding instructions and passengers travelling together in groups.

## 6.2 Limitations

- Memory intensive. Due to boarding/disembarking being a stochastic process, a large number of test cases are needed to obtain an accurate average for any given scenario. Our model is bulky.
- Large number of assumptions made. For ease of simulation, we assumed many things, ranging from a constant walking speed down the aisle, or that passengers would always sit in the correct seat or walk down the correct aisle to their seat. In reality, this will not always be the case. To improve our model, we could include these factors in our simulation.

## Letter to Executive

Dear Airline Executive,

Through analysis of boarding and disembarking methods, our team has developed a model that has allowed us to determine the optimal boarding and disembarking methods for a variety of different aircraft, with a number of different restrictions.

One consideration when deciding boarding methods is the impact that it will have on family groups. It is vital that we avoid splitting these groups when boarding to ensure all passengers have a positive travelling experience with you. Another factor that affects loading times is the number of passengers that don't follow boarding methods.

We found that three main factors that contribute to the time take to board an aircraft. The first of these is the walking speed of the passengers. However, there is not much that can realistically be done about this. Secondly, there is the time taken to stow overhead luggage. While passengers are doing this, they are blocking the aisle. Another aisle blockage comes passengers try to get to seats that are blocked by other passengers in same row.

These impact of these aisle blockages can be minimised by the chosen boarding method, and also by several different techniques. These include ensuring that people follow the boarding method (potentially through regulation from air stewards), and also by making more easily accessible overhead storage, to minimise time spent retrieving stowed luggage.

That being said, method choice is an easy was to immediately speed up passenger boarding/disembarking. In a standard narrow body aircraft, passengers should be boarded with the adjusted WMA method (window seats board first, followed by middle seats and then aisle seats, but groups board together), and should disembark giving the right of way to passengers coming from the back. Both methods minimise aisle blockages, and allow optimal aisle flow.

In a wide wing aircraft, the optimal boarding method is by section (back to front). This fastest method that can be practically implemented. To disembark, we recommend the 'across' method, similar the method for a narrow aircraft, where passengers furthest from the door get right of way. For the 2 aisle, 2 entrance aircraft, the recommended methods are the same as the narrow body: adjust WMA for boarding and back to front for disembarking. We hope these recommendations and explanations will aid you in running your airline, and look forward to your feedback.

## 7 Appendices

### References

- [1] E. Mazareanu. *Number of flights performed by the global airline industry from 2004 to 2022*. 2022. URL: <https://www.statista.com/statistics/564769/airline-industry-number-of-flights/> (visited on 03/28/2022).
- [2] Suzanne Hiemstra-Van Mastrigt, Richard Ottens, and Peter Vink. “Identifying bottlenecks and designing ideas and solutions for improving aircraft passengers’ experience during boarding and disembarking”. In: *Applied Ergonomics* 77 (2019), pp. 16–21.
- [3] Highskyflying. *How Wide Are Airplane Aisles?* 2022. URL: <https://www.highskyflying.com/how-wide-are-airplane-aisles/> (visited on 03/29/2022).
- [4] Kathryn Watson. *What’s an Average Shoulder Width?* 2018. URL: <https://www.healthline.com/health/average-shoulder-width> (visited on 03/30/2022).
- [5] Air New Zealand. *Carry-on Bags*. URL: <https://www.airnewzealand.co.nz/carry-on-baggage> (visited on 03/29/2022).
- [6] Jason H. Steffen. “Optimal boarding method for airline passengers”. In: *Journal of Air Transport Management* 14.3 (May 2008), pp. 146–150. DOI: 10.1016/j.jairtraman.2008.03.003. URL: <https://doi.org/10.1016%2Fj.jairtraman.2008.03.003>.

YouTube videos used for analysis throughout the report

- <https://www.youtube.com/watch?v=SnAYtU3nc0g>
- <https://www.youtube.com/watch?v=F4Dt2-kc5Mw>
- <https://www.youtube.com/watch?v=OCCUGcb4LvE>
- <https://www.youtube.com/watch?v=V3RXT0D0CnM>
- <https://www.youtube.com/watch?v=jv4y4ir07Nc>
- <https://www.youtube.com/watch?v=AjSFC3yLA9s>
- <https://www.youtube.com/watch?v=a6oBGKEdYxc>
- <https://www.youtube.com/watch?v=perToEokKR0>
- <https://www.youtube.com/watch?v=godZCdvG8CI>
- <https://www.youtube.com/watch?v=mrcQL5Od-Fg>

## Appendix A

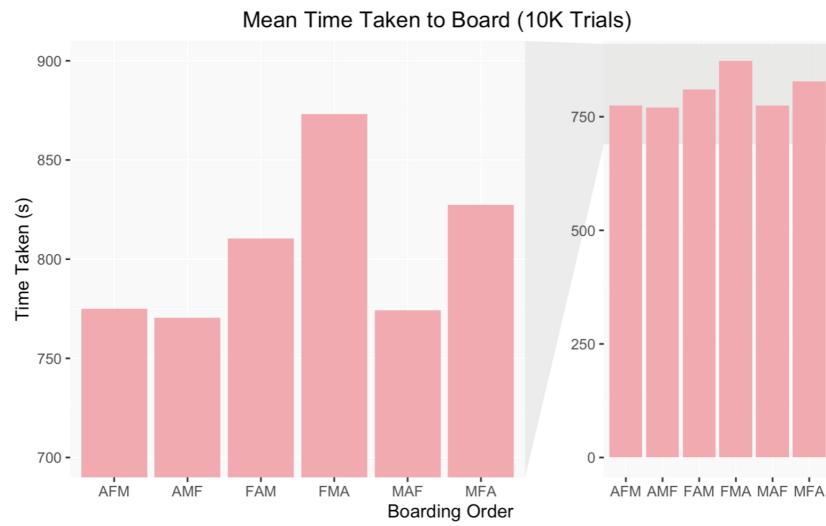


Figure 7.1: Bar chart of mean boarding times for different boarding by section combinations. The labels are in order of section boarding (e.g. AFM means aft first, front second, middle last). Note too that the chart on the left is an enlarged section of the chart on the right.

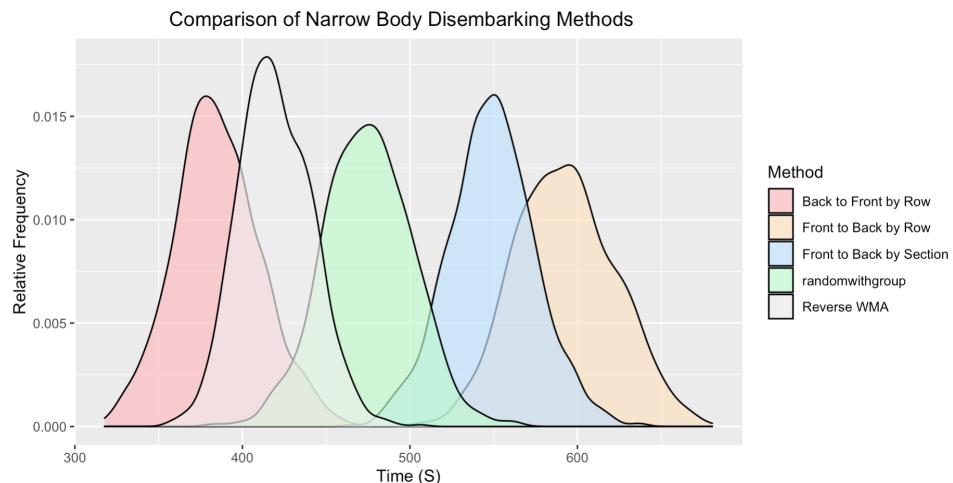


Figure 7.2: Comparison of disembarking methods for narrow body aircraft

## Appendix B

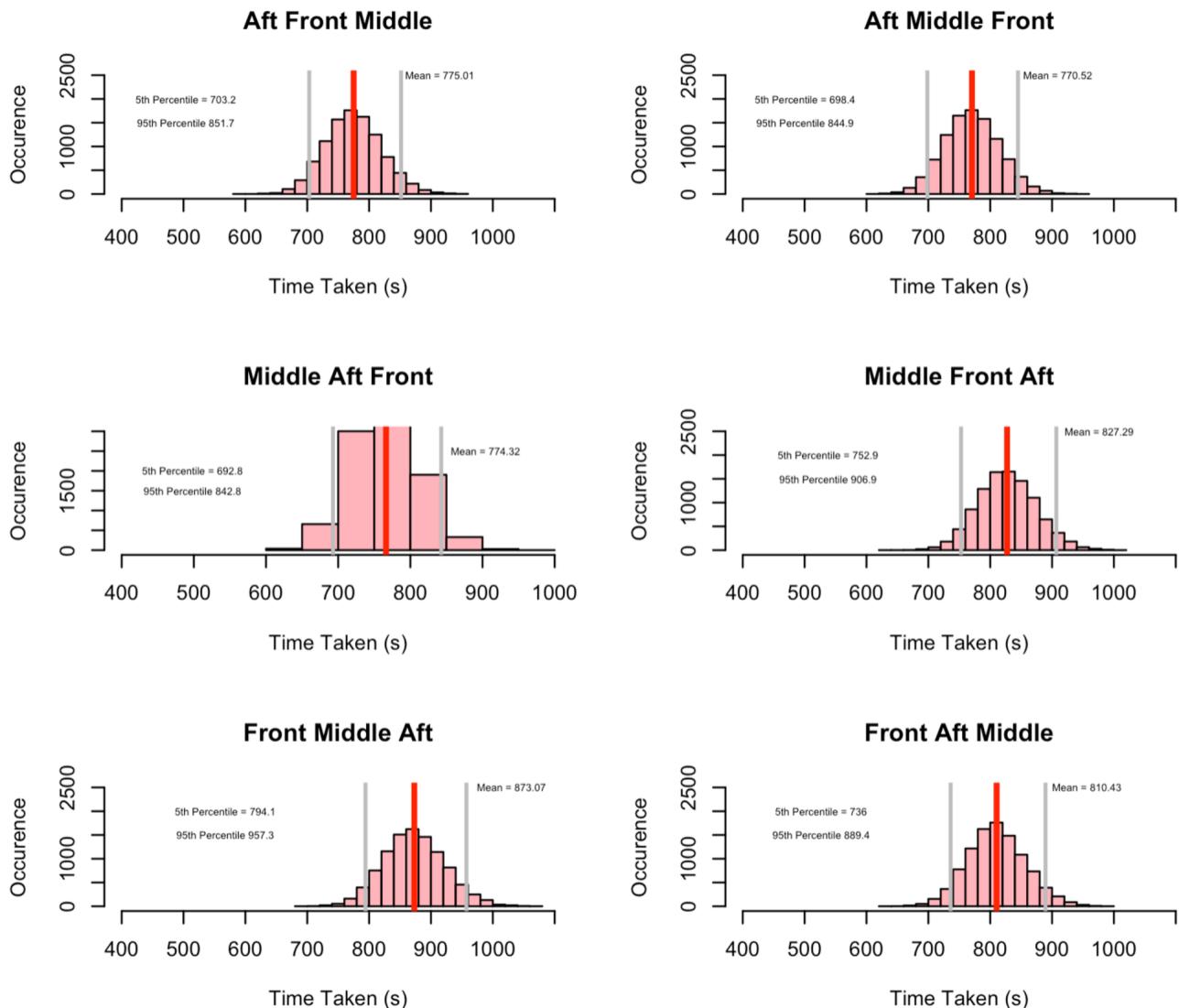


Figure 7.3: Monte Carlo simulation graphs of boarding by sections method in different orders. For instance, Aft Front Middle means first the aft is boarded, then front, and finally the middle.

## Appendix C

|                   |                   |                 | Narrow Body      | Time taken depending on different boarding methods (s) |
|-------------------|-------------------|-----------------|------------------|--|
| Capacity of plane | Wilma with groups | Random Boarding | Section boarding |  |
| 100%              | 644               | 700             | 764              |  |
| 70%               | 545               | 550             | 600              |  |
| 50%               | 301               | 311             | 302              |  |
| 30%               | 194               | 202             | 180              |  |

| Flying Wing       | Time taken depending on different boarding methods (s) |                 |                  |
|-------------------|--|-----------------|------------------|
| Capacity of plane | Wilma with groups                                      | Random Boarding | Section boarding |
| 100%              | 493  | 552             | 575              |
| 70%               | 332  | 364             | 383              |
| 50%               | 237  | 263             | 254              |
| 30%               | 189  | 187             | 177              |

| 2E2A              |                   |                 |                  |
|-------------------|-------------------|-----------------|------------------|
| Capacity of plane | Wilma with groups | Random Boarding | Section boarding |
| 100%              | 453               | 456             | 454              |
| 70%               | 319               | 302             | 292              |
| 50%               | 240               | 238             | 222              |
| 30%               | 179               | 177             | 152              |

**COVID TALES DISEMBARKING**

| Narrow Body       | Time taken depending on different boarding methods (s) |               |        |
|-------------------|--|---------------|--------|
| Capacity of plane | Back to front  | Reverse Wilma | Random |
| 100%              | 384  | 419           | 474    |
| 70%               | 275  | 302           | 308    |
| 50%               | 128  | 224           | 220    |
| 30%               | 106  | 139           | 132    |

| Flying Wing       | Time taken depending on different boarding methods (s) |               |        |
|-------------------|--|---------------|--------|
| Capacity of plane | Across (modified Back to Front)                        | Reverse Wilma | Random |
| 100%              | 221  | 239           | 248    |
| 70%               | 167  | 160           | 178    |
| 50%               | 111  | 125           | 124    |
| 30%               | 95   | 97            | 106    |

| 2E2A              | Time taken depending on different boarding methods (s) |               |        |
|-------------------|--|---------------|--------|
| Capacity of plane | Back to front  | Reverse Wilma | Random |
| 100%              | 180  | 198           | 209    |
| 70%               | 142  | 168           | 192    |
| 50%               | 104  | 154           | 142    |
| 30%               | 87   | 92            | 104    |

**Appendix D**

Code used in R for statistical analysis

```

1 # INSTALL AND LOAD PACKAGES #####
2
3 library(datasets) # Load/unload base packages manually
4
5 ##### Graph Function #####
6 make_histogram <- function(x, colour, title, labelpos, meanpos, breaks1) {
7   quantile5 <- c(quantile(x, probs = 0.05))
8   quantile95 <- c(quantile(x, probs = 0.95))
9   mean <- mean(x)
10  hist(x,
```

```
11     xlim=c(400,1100),
12     ylim=c(0,350),
13     col=colour,
14     breaks = breaks1,
15     xlab="Time Taken (s)",
16     ylab="Occurrence",
17     main=title)
18 # Add lines for mean, 5%, and 95%
19 abline(v = mean,
20         col = "red",
21         lwd = 3)
22 abline(v = c(quantile5, quantile95),
23         col = "grey",
24         lwd = 2)
25 # Add labels
26 text(x = mean*meanpos,
27       y = 125,
28       paste("Mean =", round(mean,2)),
29       col = "black",
30       cex = 1)
31 text(x = mean*labelpos,
32       y = 87.5,
33       paste("5th Percentile =", round(quantile5,1)),
34       col = "black",
35       cex = 0.7)
36 text(x = mean*labelpos,
37       y = 50,
38       paste("95th Percentile =", round(quantile95, 1)),
39       col = "black",
40       cex = 0.7)
41 }
42 dev.off()
43 #function(x,colour, title,labelpos,meanpos)
44 #####Finding best and worst boarding by sections #####
45 par(mfrow = c(3,2))
46 make_histogram(sectionsdata$afm, "#ffb3ba",
47                 "Aft Front Middle",
48                 0.65, 1.18 )
49 make_histogram(sectionsdata$amf, "#ffb3ba",
50                 "Aft Middle Front",
51                 0.65, 1.18 )
52 make_histogram(sectionsdata$maf, "#ffb3ba",
53                 "Middle Aft Front",
54                 0.65, 1.18 )
55 make_histogram(sectionsdata$mfa, "#ffb3ba",
56                 "Middle Front Aft",
57                 0.65, 1.18 )
58 make_histogram(sectionsdata$fma, "#ffb3ba",
59                 "Front Middle Aft",
60                 0.65, 1.18 )
61 make_histogram(sectionsdata$fam, "#ffb3ba",
62                 "Front Aft Middle",
63                 0.65, 1.18 )
64 fma<-mean(sectionsdata$fma)
65 #Compare with groups to without gorups
66 dev.off()
67 #function(x,colour, title,labelpos,meanpos)
68 #Compare four boarding methods: Random, Seats, Best Section, Worst Section
69 par(mfrow = c(1,2))
70 breaks2<-rep(10,1)
```

```

71 make_histogram(dataogmethod$Random , "#bafffc9",
72                 "Random Boarding",
73                 1.33, 1.33 )
74 make_histogram(dataogmethod$Seat , "#bae1ff",
75                 "Boarding by Seat: Groups",
76                 1.45, 1.45, c
77                 (460,485,510,535,560,585,610,635,660,685,710,735,760,785,810) )
78 make_histogram(dataogmethod$fma , "#ffb3ba",
79                 "Front to Back",
80                 0.65, 0.65 )
81 make_histogram(dataogmethod$amf , "#ffb3ba",
82                 "Back to Front",
83                 0.7, 0.7 )
84 make_histogram(dataogmethod$Seats.No.Groups , "#bae1ff",
85                 "Boarding by Seat: No Groups",
86                 1.6, 1.6, c(420,440,460,480,500,520,540,560,580,600))
87 dev.off()
88 make_histogram(groupsfirst$prioritize_groups_boarding , "#fffffa",
89                 "Groups First Boarding",
90                 1.28, 1.28 )
91 make_histogram(modifiedsteffen$modsteffen , "#ffdffa",
92                 "Steffen Modified Boarding",
93                 1.3, 1.3 )
94 #####Overlay histograms#####
95 library(ggplot2)
96 library(plyr)
97 #mean
98 mu <- ddply(dataogmethod.transp, "Type", summarise, grp.mean=mean(Time))
99 # Basic density
100 #ggplot(dataogmethod.transp, aes(x=Time, fill=Type)) +
101   #geom_density(color="darkblue", fill="lightblue")
102 # Use semi-transparent fill
103 p <- ggplot(dataogmethod.transp, aes(x = Time, fill = Type)) +
104   geom_density(alpha=0.6) +
105   #theme_bw() +
106   #geom_vline(data=mu, aes(xintercept=grp.mean, color="black"),
107   #linetype="dashed") +
108   labs(title="Comparison of Methods",
109         x="Time (S)", y = "Relative Frequency")
110 # Add mean lines
111 p+scale_fill_manual(values=c("#ffb3ba", "#fffffa","#ffdffa","#bae1ff","#bafffc9"))
112 p
113 #####Provided Methods Sensitivity Analysis#####
114 dev.off()
115 attach(bagsensitivity)
116 linearModel <- lm(SectionBTF ~ NBC, data=sensitivityanalysis)
117 summary(linearModel)
118 lm(formula = Seats.No.Group ~ NBC)
119 sensitivityanalysis$NBC2 <- sensitivityanalysis$NBC^2
120 quadraticModel <- lm(SectionBTF ~ NBC + NBC2, data=sensitivityanalysis)
121 summary(quadraticModel)
122 #create sequence of hour values
123 NBCValues <- seq(0, 1, 0.01)
124 #create list of predicted time values using quadratic model
125 timePredict <- predict(quadraticModel, list(NBC=NBCValues, NBC2=NBCValues^2))
126 plot(bagsensitivity$bagcoef, bagsensitivity$section, pch = 19, cex = 0.75,
127       col = "#ffb3ba", xlab = "Bag Coefficient",
128       ylab = "Time (s)",
129       ylim=c(450,900),

```

```

130     main = "Sensitivity Analysis of 3 Given Models")
131 #lines(NBCValues, timePredict, col='#fffb3ba')
132 abline(lm(bagsensitivity$section ~ bagsensitivity$bagcoef), col = "#fffb3ba")
133 points(bagsensitivity$bagcoef, bagsensitivity$random, pch = 19, cex = 0.75, col =
    "#95ccaa")
134 abline(lm(bagsensitivity$random ~ bagsensitivity$bagcoef), col = "#95ccaa")
135 #lines(bagsensitivity$bagcoef, bagsensitivity$wma, col="#bae1ff")
136 points(bagsensitivity$bagcoef, bagsensitivity$wma, pch = 19, cex = 0.75, col = "#"
    bae1ff)
137 abline(lm(bagsensitivity$wma ~ bagsensitivity$bagcoef), col = "#bae1ff")
138 legend(0.7, 550, legend=c("Back to Front", "Random", "Seat (No Groups)", col=c("
    #fffb3ba", "#95ccaa", "#bae1ff")), lty=1:2, cex=0.65)
139 ##### Suggested Methods Sensitivity#####
140 dev.off()
141 attach(seat_with_groups1)
142 plot(NBC, SeatwithGroups, pch = 19, cex = 0.75,
       col = "#bae1ff", xlab = "Disobedience Coefficient",
       ylab = "Time (s)",
       ylim=c(550,750),
       main = "Sensitivity Analysis of Chosen Method Against Random")
143 abline(lm(SeatwithGroups ~ NBC), col = "#bae1ff")
144 lines(NBC, Random, col="#95ccaa")
145 points(NBC, Random, pch = 19, cex = 0.75, col = "#95ccaa")
146
147 legend(0.75, 625, legend=c("Random", "Seat (Groups)"), col=c("#95ccaa", "#bae1ff"
    ), lty=1, cex=0.65)
148 #####
149 barplot((Barplot_section_order$Mean),
           main="Mean Time Taken to Board (10k Trials)",
           names.arg=c("AFM", "AMF", "MAF","MFA","FMA","FAM"),
           ylim=c(700,900),
           xpd=FALSE,
           col="#fffb3ba",
           ylab = "Time Taken to Board (s)",
           xlab = "Boarding Method")
150 #####
151 install.packages("ggforce")
152 library("ggforce")
153 attach(Barplot_section_order)
154 ggplot(Barplot_section_order, aes(Order, Mean)) +                      # ggplot2 facet_
    zoom_plot
155     geom_bar(stat = "identity", fill = "#f2aab1") +
156     ggtitle("Mean Time Taken to Board (10K Trials)")+
157     labs(y= "Time Taken (s)", x = "Boarding Order")+
158     theme(panel.background = element_rect("#f9f9f9"))+
159     facet_zoom(ylim = c(700, 900), show.area=TRUE)
160 #####
161 #####

```

## Appendix E

```

1
2 from audioop import reverse
3 import graphlib
4 from lib2to3.pgen2.token import NUMBER
5 import random
6 from string import ascii_letters
7 import math
8

```

```
9 #Comment out if not using
10 import matplotlib.pyplot as plt
11 import numpy
12 import matplotlib.colors as colors
13 import matplotlib as mpl
14 from matplotlib.colors import Colormap, LinearSegmentedColormap, ListedColormap
15 #Constants for references
16 PRIORITY = 0
17 INTERNAL_COCK = 1
18 HAS_LUGGAGE = 2
19
20 # all measured in standard units (m,s,m/s etc)
21 AVERAGE_WALKING_SPEED = 0.8
22 AVERAGE_SEAT_PITCH = 0.78
23 TIME_TO_MOVE = AVERAGE_SEAT_PITCH / AVERAGE_WALKING_SPEED
24 TIME_TO_MOVE_PAST_SEAT = 2
25
26 #Priority system
27 priority_weightings = []
28 highest_priority_assigned = 0
29 #Things to change
30 BAG_COEFFICIENT = (20,80,10)
31 NAUGHTY_BOY_COEFFICIENT = 0.3
32 THANOS_SNAP_COEFFICIENT = 0.5
33 # proportions of group sizes
34 SINGLE_PRINGLE_COEFFICIENT = 70
35 COUPLES_COEFFICIENT = 20
36 THREESOME_COEFFICIENT = 10
37
38
39
40 # General setup shotput all seats in
41 NUM_ROWS = 33
42 NUM_SEATS = 6
43 AISLE_INDEX = 3
44 #Wide body shot
45 WIDE_WING_SEATS = 28
46 WIDE_WING_ROWS = 15
47 TWO_SEATS = 9
48 TWO_ROWS = 42
49 TWO_A_ROWS = 18
50 TWO_B_ROWS = 21
51 GAP_SIZE = 3
52
53
54
55
56 #Normal render
57 def initialize_render():
58
59     global highest_priority_assigned
60     #Absolute mess of code
61     image = []
62     for i in range(NUM_SEATS+1):
63         subimage = []
64         for k in range(NUM_ROWS):
65             if k % 2 == 0:
66                 subimage.append(0.5)
67             else:
68                 subimage.append(1.5)
```

```
69
70     image.append(subimage)
71
72
73 fig,ax = plt.subplots(1,1)
74
75 cmap = mpl.cm.OrRd
76 norm = mpl.colors.Normalize(vmin=-1, vmax=highest_priority_assigned)
77
78
79
80 image = numpy.array(image)
81 im = ax.imshow(image, cmap=cmap, norm = norm)
82
83 ax.set_yticks(numpy.arange(0.5, NUM_SEATS+1.5, 1).tolist(), minor=False)
84 ax.yaxis.grid(True, which='major')
85 ax.set_yticklabels(['Row A','Row B','Row C','Aisle','Row D','Row E','Row F'])
86 ax.set_ylim(top=-0.5)
87
88 ax.set_xticks(numpy.arange(0.5, NUM_ROWS+.5, 1).tolist(), minor=False)
89 ax.xaxis.grid(True, which='major')
90 xticklist = []
91 #Create list of numbers between
92 for i in range(NUM_ROWS):
93     if ((i+1) % 5 == 0) and (i != 0):
94         xticklist.append(str(i+1))
95     else:
96         xticklist.append(',')
97
98 ax.set_xticklabels(xticklist)
99 ax.set_xlim(left=-0.5)
100
101 return im,fig
102 def update_render(seat_plan):
103
104     visualizer = []
105
106     for i,column in enumerate(seat_plan):
107         visualizer.append([])
108         for seats in column:
109
110             visualizer[i].append(seats[PRIORITY])
111
112
113
114
115
116     im.set_data(visualizer)
117     fig.canvas.draw_idle()
118     plt.pause(1)
119 def intalize_render_two_thing():
120
121     global highest_priority_assigned
122
123     #Absolute mess of code
124     image = []
125     for i in range(TWO_SEATS):
126         subimage = []
127         for k in range(TWO_ROWS):
128             if k % 2 == 0:
```

```
129         subimage.append(0.5)
130     else:
131         subimage.append(1.5)
132
133     image.append(subimage)
134
135
136 fig,ax = plt.subplots(1,1)
137
138 cmap = mpl.cm.OrRd
139 norm = mpl.colors.Normalize(vmin=-1, vmax=highest_priority_assigned)
140
141
142
143 image = numpy.array(image)
144 im = ax.imshow(image, cmap=cmap, norm = norm)
145
146 ax.set_yticks(numpy.arange(0.5, TWO_SEATS+0.5, 1).tolist(), minor=False)
147 ax.yaxis.grid(True, which='major')
148 ax.set_yticklabels(['Row A','Row B','Aisle','Row C','Row D','Row E','Aisle','Row F','Row G'])
149 ax.set_ylim(top=-0.5)
150 ax.set_title('Two Doors Two Aisles Disembarking Model')
151 ax.set_xticks(numpy.arange(0.5, TWO_ROWS+.5, 1).tolist(), minor=False)
152 ax.xaxis.grid(True, which='major')
153 xticklist = []
154 #Create list of numbers between
155 for i in range(TWO_ROWS):
156     if ((i+1) % 5 == 0) and (i != 0):
157         xticklist.append(str(i+1))
158     else:
159         xticklist.append(',')
160
161 ax.set_xticklabels(xticklist)
162 ax.set_xlim(left=-0.5)
163
164 return im,fig
165 def intalize_render_widebody():
166
167     global highest_priority_assigned
168
169     #Absolute mess of code
170     image = []
171     for i in range(WIDE_WING_SEATS):
172         subimage = []
173         for k in range(WIDE_WING_ROWS):
174             if k % 2 == 0:
175                 subimage.append(0.5)
176             else:
177                 subimage.append(1.5)
178
179         image.append(subimage)
180
181
182 fig,ax = plt.subplots(1,1)
183
184 cmap = mpl.cm.OrRd
185 norm = mpl.colors.Normalize(vmin=-1, vmax=highest_priority_assigned)
186
187
```

```
188     image = numpy.array(image)
189     im = ax.imshow(image, cmap=cmap, norm = norm)
190
191     ax.set_yticks(numpy.arange(0.5, WIDE_WING_SEATS+0.5, 1).tolist(), minor=False)
192
193     ax.yaxis.grid(True, which='major')
194     ax.set_yticklabels(['Row A','Row B','Row C','Aisle','Row D','Row E','Row F',
195     'Row G','Row H','Row I','Aisle','Row J','Row K','Row L','Row M','Row N','Row O',
196     'Aisle','Row P','Row Q','Row R','Row S','Row T','Row U','Aisle','Row V','Row
197     W','Row X'])
198     ax.set_ylim(top=-0.5)
199     ax.set_title('Widebody Disembarking Model')
200     ax.set_xticks(numpy.arange(0.5, WIDE_WING_ROWS+.5, 1).tolist(), minor=False)
201     ax.xaxis.grid(True, which='major')
202     xticklist = []
203     #Create list of numbers between
204     for i in range(WIDE_WING_ROWS):
205         if ((i+1) % 5 == 0) and (i != 0):
206             xticklist.append(str(i+1))
207         else:
208             xticklist.append('')
209
210     ax.set_xticklabels(xticklist)
211     ax.set_xlim(left=-0.5)
212
213     return im,fig
214
215 #General shot to setup
216 def generate_priorities(highest_priority_assigned):
217
218     weights = list(range(1, highest_priority_assigned+1))
219
220     return(weights)
221 def group_size():
222     return random.choices([1,2,3], weights=(SINGLE_PRINGLE_COEFFICIENT,
223     COUPLES_COEFFICIENT, THREESOME_COEFFICIENT), k=1)[0]
224 def assign_luggage():
225     return random.choices([0,1,2], weights=BAG_COEFFICIENT, k=1)[0]
226 def bag_shit():
227     global seating_plan
228     #Intalize bag amounts
229     lockers = [[0,0] for i in range(NUM_ROWS)]
230     for row in range(NUM_ROWS):
231         for seat in range(NUM_SEATS+1):
232
233             if seat < 3:
234                 lockers[row][0] += seating_plan[seat][row][HAS_LUGGAGE]
235
236             elif seat > 3:
237                 lockers[row][1] += seating_plan[seat][row][HAS_LUGGAGE]
238
239     return lockers
240 #Widebody
241 def bag_shit_wide():
242     global seating_plan
243     #Intalize bag amounts
244
245     lockers = [[ [0,0] for _ in range(WIDE_WING_ROWS-1)] for _ in range(4)]
```

```
243     for row in range(1,WIDE_WING_ROWS):
244
245         for seat in range(1,WIDE_WING_SEATS):
246             sublocker = math.floor((seat)/7)
247             if seating_plan[seat][row][PRIORITY] != -1:
248                 if seat % 7 < 3:
249                     lockers[sublocker][row-1][0] += seating_plan[seat][row][
250 HAS_LUGGAGE]
251
252                 elif seat % 7 > 3:
253                     lockers[sublocker][row-1][1] += seating_plan[seat][row][
254 HAS_LUGGAGE]
255
256
257
258
259
260     return lockers
261 #Modifitying shot
262 def group_shit():
263     for row in range(NUM_ROWS):
264         #Resets var
265         current_group_size = 0
266         current_group_priority = []
267         current_group_people_added = 0
268         for seat in range(NUM_SEATS+1):
269             #make sure we not in aisles
270             if seat != AISLE_INDEX:
271                 #If not currently generating create a new group
272                 if current_group_people_added == 0:
273                     current_group_size = current_group_people_added =
274 group_size()
275
276                 if current_group_people_added == 1:
277                     current_group_people_added = 0
278                 else:
279                     current_group_priority.append(seating_plan[seat][row][
280 PRIORITY])
281
282             current_group_people_added -=1
283
284
285             else: # Currently generating a group
286                 current_group_priority.append(seating_plan[seat][row][
287 PRIORITY])
288
289             current_group_people_added -=1
290
291             #If all people added to group
292             if current_group_people_added == 0:
293                 #Loop back through people and send priority to
294 average
295
296
297             gone_through_aisles = 0
298             for i in range(current_group_size):
299                 #Go back through and adjust priority
300
301                 #If gone through aisles add another
302                 if (seat - i) == AISLE_INDEX:
303                     gone_through_aisles = 1
```

```
297
298                     seating_plan[seat-(i+gone_through_aisles)][row][
299             PRIORITY] = round(sum(current_group_prioty) / len(current_group_prioty))
300
301     def naughty_people():
302
303         #Generate total amount of naughty boys
304         naughty_bois = math.ceil(NUM_ROWS*NUM_SEATS*NAUGHTY_BOY_COEFFICIENT)
305
306         for i in range(naughty_bois):
307             numbers = list(range(0, NUM_SEATS+1))
308
309             numbers.remove(3)
310
311             seat = random.choice(numbers)
312             row = random.randrange(NUM_ROWS)
313
314             seating_plan[seat][row][PRIORITY] = random.randrange(1,
315             highest_priority_assigned+1)
316
317     def thanos_snap():
318         for seat in range(NUM_SEATS+1):
319             for row in range(NUM_ROWS):
320                 if THANOS_SNAP_COEFFICIENT > random.random():
321                     seating_plan[seat][row] = [-1,0]
322
323 #Narrow body boarding
324     def reverse_wilma():
325         global seating_plan
326         global highest_priority_assigned
327
328         highest_priority_assigned = 3
329
330         seating_plan = [[ [3,0,assign_luggage()] for _ in range(NUM_ROWS)] for _ in
331             range(NUM_SEATS + 1)]
332         seating_plan[0]= [[1,0,assign_luggage()] for _ in range(NUM_ROWS)]
333         seating_plan[6]= [[1,0,assign_luggage()] for _ in range(NUM_ROWS)]
334         seating_plan[1]= [[2,0,assign_luggage()] for _ in range(NUM_ROWS)]
335         seating_plan[5]= [[2,0,assign_luggage()] for _ in range(NUM_ROWS)]
336         seating_plan[AISLE_INDEX]= [[-1,0] for _ in range(NUM_ROWS)]
337
338         naughty_people()
339         group_shit()
340     def random_deboard():
341         global seating_plan
342         global highest_priority_assigned
343
344         highest_priority_assigned = 10
345
346         seating_plan = [[ [random.randrange(1,10),0,assign_luggage()] for _ in range(
347             NUM_ROWS)] for _ in range(NUM_SEATS + 1)]
348         seating_plan[AISLE_INDEX]= [[-1,0] for _ in range(NUM_ROWS)]
349
350         group_shit()
351     def sections():
352
353         global seating_plan
354
355         global highest_priority_assigned
356
357         highest_priority_assigned = 3
358         fjuk = []
359         for i in range(NUM_SEATS+1):
```

```
353     aisles = []
354     for k in range(0,11):
355         aisles.append([3,0,assign_luggage()])
356     for k in range(11,22):
357         aisles.append([2,0,assign_luggage()])
358     for k in range(22,NUM_ROWS):
359         aisles.append([1,0,assign_luggage()])
360     fjuk.append(aisles)
361 seating_plan = fjuk
362 seating_plan[AISLE_INDEX] = [[-1,0] for _ in range(NUM_ROWS)]
363 naughty_people()
364 group_shit()
365 def back_to_front():
366     global seating_plan
367     global highest_priority_assigned
368
369
370 #Create empty seating plan
371 seating_plan = [[[-1,0,assign_luggage()] for _ in range(NUM_ROWS)] for _ in
range(NUM_SEATS + 1)]
372
373 highest_priority_assigned = 0
374
375 for row in range(NUM_ROWS):
376     for seat in range(NUM_SEATS+1):
377         #Increment priority
378
379
380         if seat == 0 or seat == 3:
381             highest_priority_assigned += 1
382             seating_plan[seat][row] = ([highest_priority_assigned,0,
assign_luggage()])
383
384 seating_plan[AISLE_INDEX] = [[-1,0] for _ in range(NUM_ROWS)]
385 #naughty_people()
386 #group_shit()
387 def generate_front_to_back():
388     global seating_plan
389     global highest_priority_assigned
390
391
392 #Create empty seating plan
393 seating_plan = [[[-1,0,assign_luggage()] for _ in range(NUM_ROWS)] for _ in
range(NUM_SEATS + 1)]
394
395 highest_priority_assigned = 0
396
397 for row in reversed(range(NUM_ROWS)):
398     for seat in range(NUM_SEATS+1):
399         #Increment priority
400
401
402         if seat == 0 or seat == 3:
403             highest_priority_assigned += 1
404             seating_plan[seat][row] = ([highest_priority_assigned,0,
assign_luggage()])
405
406 seating_plan[AISLE_INDEX] = [[-1,0] for _ in range(NUM_ROWS)]
```

```
409     naughty_people()
410     group_shit()
411
412 def group_shit_two():
413     for row in range(4, TWO_ROWS):
414         #Resets var
415         current_group_size = 0
416         current_group_priority = []
417         current_group_people_added = 0
418
419
420
421     for seat in range(0,9):
422
423         #make sure we not in aisles
424         if (seat != 2 or seat !=4) and seating_plan[seat][row][PRIORITY] != -1:
425             #If not currently generating create a new group
426             if current_group_people_added == 0:
427                 current_group_size = current_group_people_added = group_size
428             ()
429
430             if current_group_people_added == 1:
431                 current_group_people_added = 0
432             else:
433                 current_group_priority.append(seating_plan[seat][row][
434                     PRIORITY])
435
436             current_group_people_added -=1
437
438
439             else: # Currently generating a group
440                 current_group_priority.append(seating_plan[seat][row][PRIORITY]
441                     ))
442
443             current_group_people_added -=1
444
445             #If all people added to group
446             if current_group_people_added == 0:
447                 #Loop back through people and send priority to average
448
449
450             gone_through_aisles = 0
451             for i in range(current_group_size):
452                 #Go back through and adjust priority
453
454                 #If gone through aisles add another
455                 if (seat - i) == 2 or (seat - i) == 6:
456                     gone_through_aisles = 1
457
458             seating_plan[(seat-(i+gone_through_aisles))][row][
459                 PRIORITY] = round(sum(current_group_priority) / len(current_group_priority))
460
461 #Wide body boarding
462 def clear_aisles_widebody():
463     global seating_plan
464
465
466     #Clear aisles
467     for i in range(WIDE_WING_SEATS):
468         if i % 7 == 3 and i != WIDE_WING_SEATS:
469             seating_plan[i]= [[-1,0]] for _ in range(WIDE_WING_ROWS)]
470
471     #Clear front aisles
472     for k in range(WIDE_WING_SEATS):
```

```

464 seating_plan[k][0] = [-1,0]
465
466 #Clear extra 9 on both sides
467 for k in range(1, 4):
468     for j in range(0,3):
469         seating_plan[j][k] = [-1,0]
470
471 #Clear extra 9 on both sides
472 for k in range(1, 4):
473     for j in range(WIDE_WING_SEATS-3,WIDE_WING_SEATS):
474         seating_plan[j][k] = [-1,0]
475 def naughty_people_wide():
476
477     #Generate total amount of naughty boys
478     naughty_bois = math.ceil(((WIDE_WING_ROWS-1)*(WIDE_WING_SEATS-4)-18)*
479 NAUGHTY_BOY_COEFFICIENT)
480
481     for i in range(naughty_bois):
482
483         while True:
484             seat = random.randrange(WIDE_WING_SEATS)
485             row = random.randrange(WIDE_WING_ROWS)
486
487             if seating_plan[seat][row][PRIORITY] != -1:
488                 seating_plan[seat][row][PRIORITY] = random.randrange(1,
489 highest_priority_assigned+1)
490                 break
491 def group_shit_wide():
492     for row in range(WIDE_WING_ROWS):
493
494         #Resets var
495         current_group_size = 0
496         current_group_prioty = []
497         current_group_people_added = 0
498
499
500         for current_aisle in range(0, WIDE_WING_SEATS,7):
501             for seat in range(0,7):
502
503                 #make sure we not in aisles
504                 if seat != 3 and seating_plan[current_aisle+seat][row][
505 PRIORITY] != -1:
506                     #If not currently generating create a new group
507                     if current_group_people_added == 0:
508                         current_group_size = current_group_people_added =
509                         group_size()
510                         if current_group_people_added == 1:
511                             current_group_people_added = 0
512                         else:
513                             current_group_prioty.append(seating_plan[
514                         current_aisle+seat][row][PRIORITY])
515                         current_group_people_added -=1
516
517                         else: # Currently generating a group
518                             current_group_prioty.append(seating_plan[
519                         current_aisle+seat][row][PRIORITY])
520                             current_group_people_added -=1
521
522                         #If all people added to group
523                         if current_group_people_added == 0:
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
999

```

```
518                         #Loop back through people and send priority to
519     average
520
521             gone_through_aisles = 0
522             for i in range(current_group_size):
523                 #Go back through and adjust priority
524
525                     #If gone through aisles add another
526                     if (seat - i) == 3:
527                         gone_through_aisles = 1
528
529                     seating_plan[(seat-(i+gone_through_aisles))+
530 current_aisle][row][PRIORITY] = round(sum(current_group_priorty) / len(
531 current_group_priorty))
530 def reverse_wilma_widebody():
531     global seating_plan
532     global highest_priority_assigned
533
534     highest_priority_assigned = 3
535
536     #Make an empty
537     seating_plan = [[ [-1,0,0] for _ in range(WIDE_WING_ROWS)] for _ in range(
538 WIDE_WING_SEATS)]
539
540         for i in range(WIDE_WING_SEATS):
541             if i in [2,4,9,11,16,18,23,25]:
542                 seating_plan[i]= [[3,0,assign_luggage()] for _ in range(
543 WIDE_WING_ROWS)]
544                 elif i in [1,5,8,12,15,19,22,26]:
545                     seating_plan[i]= [[2,0,assign_luggage()] for _ in range(
546 WIDE_WING_ROWS)]
547                     elif i in [0,6,7,13,14,20,21,27]:
548                         seating_plan[i]= [[1,0,assign_luggage()] for _ in range(
549 WIDE_WING_ROWS)]
550
551     clear_aisles_widebody()
552     naughty_people_wide()
553     group_shit_wide()
554 def random_deboard_widebody():
555     global seating_plan
556     global highest_priority_assigned
557
558     highest_priority_assigned = 10
559
560     seating_plan = [[ [ random.randrange(1,10),0,assign_luggage()] for _ in range(
561 WIDE_WING_ROWS)] for _ in range(WIDE_WING_SEATS)]
562
563     clear_aisles_widebody()
564     group_shit_wide()
565 def sections_widebody():
566
567     global seating_plan
568
569     global highest_priority_assigned
570
571     highest_priority_assigned = 3
572     fjuk = []
573     for i in range(WIDE_WING_SEATS):
```

```
570     aisles = []
571     for k in range(1,8):
572         aisles.append([3,0,assign_luggage()])
573     for k in range(8,12):
574         aisles.append([2,0,assign_luggage()])
575     for k in range(12,WIDE_WING_ROWS+1):
576         aisles.append([1,0,assign_luggage()])
577     fjuk.append(aisles)
578 seating_plan = fjuk
579
580
581
582 clear_aisles_widebody()
583
584 naughty_people_wide()
585 group_shit_wide()
586 def back_to_front_widebody():
587     global seating_plan
588     global highest_priority_assigned
589
590
591 #Create empty seating plan
592 seating_plan = [[ [k+j,0,assign_luggage()] for k in range(WIDE_WING_ROWS)] for j in range(WIDE_WING_SEATS)]
593
594 highest_priority_assigned = WIDE_WING_SEATS+WIDE_WING_ROWS
595
596 clear_aisles_widebody()
597
598 naughty_people_wide()
599 group_shit_wide()
600 def across_widebody():
601     global seating_plan
602     global highest_priority_assigned
603
604
605 #Create empty seating plan
606 seating_plan = [[ [k,0,assign_luggage()] for k in range(WIDE_WING_ROWS)] for j in range(WIDE_WING_SEATS)]
607
608 highest_priority_assigned = WIDE_WING_ROWS
609
610 clear_aisles_widebody()
611
612 naughty_people_wide()
613 group_shit_wide()
614 def naughty_people_two():
615
616     global seating_plan
617     global highest_priority_assigned
618
619     #Generate total amount of naughty boys
620     naughty_bois = math.ceil(((TWO_SEATS-2)*(TWO_A_ROWS+TWO_B_ROWS)+18)*NAUGHTY_BOY_COEFFICIENT)
621
622     for i in range(naughty_bois):
623
624         while True:
625             seat = random.randrange(TWO_SEATS)
626             row = random.randrange(TWO_ROWS)
```

```
627             if seating_plan[seat][row][PRIORITY] != -1:
628                 seating_plan[seat][row][PRIORITY] = random.randrange(1,
629 highest_priority_assigned)
630                 break
631 def generate_front_to_back_widebody():
632     global seating_plan
633     global highest_priority_assigned
634
635     highest_priority_assigned = WIDE_WING_SEATS+WIDE_WING_ROWS
636     #Create empty seating plan
637     seating_plan = [[ [highest_priority_assigned-(k+j),0,assign_luggage()] for k
638 in range(WIDE_WING_ROWS)] for j in range(WIDE_WING_SEATS)]
639
640
641     clear_aisles_widebody()
642
643     naughty_people_wide()
644     group_shit_wide()
645
646
647 def bag_shit_Two():
648     global seating_plan
649     #Intalize bag amounts
650     lockers = [[[0,0] for _ in range(TWO_A_ROWS)],[[0,0] for _ in range(
651 TWO_B_ROWS)]]
652
653     #A first
654     for row in range(TWO_A_ROWS):
655
656         for seat in range(TWO_SEATS):
657
658             if seating_plan[seat][row][PRIORITY] != -1:
659                 if seat <=3:
660                     lockers[0][row][0] += seating_plan[seat][row][HAS_LUGGAGE]
661
662                 elif seat > 3:
663                     lockers[0][row][1] += seating_plan[seat][row][HAS_LUGGAGE]
664
665     #B second
666     for row in range(TWO_B_ROWS):
667
668         for seat in range(TWO_SEATS):
669
670             if seating_plan[seat][row+GAP_SIZE+TWO_A_ROWS][PRIORITY] != -1:
671                 if seat <=3:
672                     lockers[1][row][0] += seating_plan[seat][row+GAP_SIZE+
673 TWO_A_ROWS][HAS_LUGGAGE]
674
675                 elif seat > 3:
676                     lockers[1][row][1] += seating_plan[seat][row+GAP_SIZE+
677 TWO_A_ROWS][HAS_LUGGAGE]
678
679
680     return lockers
681 def two_first_class():
```

```
682     global seating_plan
683     global highest_priority_assigned
684
685     #Generate first class
686     seating_plan[0][0][0] = highest_priority_assigned
687     seating_plan[0][1][0] = highest_priority_assigned
688     seating_plan[0][2][0] = highest_priority_assigned
689     seating_plan[1][0][0] = highest_priority_assigned
690     seating_plan[1][1][0] = highest_priority_assigned
691     seating_plan[1][2][0] = highest_priority_assigned
692     seating_plan[3][0][0] = highest_priority_assigned
693     seating_plan[3][1][0] = highest_priority_assigned
694     seating_plan[3][2][0] = highest_priority_assigned
695     seating_plan[5][0][0] = highest_priority_assigned
696     seating_plan[5][1][0] = highest_priority_assigned
697     seating_plan[5][2][0] = highest_priority_assigned
698     seating_plan[7][0][0] = highest_priority_assigned
699     seating_plan[7][1][0] = highest_priority_assigned
700     seating_plan[7][2][0] = highest_priority_assigned
701     seating_plan[8][0][0] = highest_priority_assigned
702     seating_plan[8][1][0] = highest_priority_assigned
703     seating_plan[8][2][0] = highest_priority_assigned
704
705     #Two
706     def two_cleanup():
707         global seating_plan
708         global highest_priority_assigned
709
710
711     #Clear aisles
712     seating_plan[2] = [[-1,0] for _ in range(TWO_ROWS)]
713     seating_plan[6] = [[-1,0] for _ in range(TWO_ROWS)]
714     #Tidy up first class
715     seating_plan[4][0] = [-1,0]
716     seating_plan[4][1] = [-1,0]
717     seating_plan[4][2] = [-1,0]
718
719
720     #Clear queues out
721     for k in range(TWO_SEATS):
722         for j in range(TWO_ROWS):
723             if j in [3, 18, 19, 20, 42]:
724                 seating_plan[k][j] = [-1,0]
725
726     def two_random():
727         global seating_plan
728         global highest_priority_assigned
729
730         seating_plan = [[ [random.randrange(1,10),0,assign_luggage()] for _ in range(TWO_ROWS)] for _ in range(TWO_SEATS)]
731
732         highest_priority_assigned = 10
733         two_first_class()
734         two_cleanup()
735         naughty_people_two()
736         group_shit_two()
737
738     def two_back_to_front():
739         global seating_plan
740         global highest_priority_assigned
```

```
741     highest_priority_assigned = 0
742     #Generate an empty plane
743     seating_plan = [[ [0,0,0] for _ in range(TWO_ROWS)] for _ in range(TWO_SEATS)]
744 ]
745     for _ in range(TWO_SEATS):
746         for k in range(4,TWO_A_ROWS):
747
748             seating_plan[_][k] = [k,0,assign_luggage()]
749
750     for _ in range(TWO_SEATS):
751         for k in (range(TWO_B_ROWS)):
752
753             seating_plan[_][(TWO_B_ROWS-k)+(TWO_A_ROWS+GAP_SIZE-1)] = [k,0,
754 assign_luggage()]
755
756     highest_priority_assigned = TWO_B_ROWS
757
758     two_first_class()
759     two_cleanup()
760     naughty_people_two()
761     group_shit_two()
762
763 def two_front_to_back():
764     global seating_plan
765     global highest_priority_assigned
766     highest_priority_assigned = 0
767     #Generate an empty plane
768     seating_plan = [[ [0,0,0] for _ in range(TWO_ROWS)] for _ in range(TWO_SEATS)]
769 ]
770     for _ in range(TWO_SEATS):
771         for k in range(4,TWO_A_ROWS):
772
773             seating_plan[_][TWO_A_ROWS-k] = [k,0,assign_luggage()]
774
775     for _ in range(TWO_SEATS):
776         for k in (range(TWO_B_ROWS)):
777
778             seating_plan[_][(k)+(TWO_A_ROWS+GAP_SIZE)] = [k,0,assign_luggage()]
779
780     highest_priority_assigned = TWO_B_ROWS
781
782     two_first_class()
783     two_cleanup()
784     naughty_people_two()
785     group_shit_two()
786
787 def two_reverse_wilma_widebody():
788     global seating_plan
789     global highest_priority_assigned
790
791     highest_priority_assigned = 2
792
793     #Make an empty
794     seating_plan = [[ [-1,0,0] for _ in range(TWO_ROWS)] for _ in range(TWO_SEATS)]
795 ]
796     for i in range(TWO_SEATS):
797         if i in [1,3,5,7]:
798             seating_plan[i]= [[2,0,assign_luggage()] for _ in range(TWO_ROWS)]
799         elif i in [0,4,8]:
```

```
797         seating_plan[i] = [[1, 0, assign_luggage()]] for _ in range(TWO_ROWS)]
798
799     two_first_class()
800     two_cleanup()
801     naughty_people_two()
802     group_shit_two()
803
804
805 def two_reverse_sections_360():
806     global seating_plan
807
808     global highest_priority_assigned
809
810     highest_priority_assigned = 3
811     fjuk = []
812     for i in range(TWO_SEATS):
813         aisles = []
814         for k in range(0, 8):
815             aisles.append([3, 0, assign_luggage()])
816         for k in range(8, 13):
817             aisles.append([2, 0, assign_luggage()])
818         for k in range(13, 21):
819             aisles.append([1, 0, assign_luggage()])
820         for k in range(21, 28):
821             aisles.append([1, 0, assign_luggage()])
822         for k in range(28, 36):
823             aisles.append([2, 0, assign_luggage()])
824         for k in range(36, TWO_ROWS):
825             aisles.append([3, 0, assign_luggage()])
826         fjuk.append(aisles)
827     seating_plan = fjuk
828
829     two_first_class()
830     two_cleanup()
831     naughty_people_two()
832     group_shit_two()
833
834
835
836
837
838
839
840 #Logic
841 def check_locker_space_wide(luggage_number, current_row, seat, lockers):
842     # if passenger has no baggage
843     if luggage_number == 0:
844         return 0
845
846     sublocker = math.floor((seat)/7)
847
848     if seat % 7 < 3:
849         nbins = lockers[sublocker][current_row-1][0]
850         lockers[sublocker][current_row-1][0] -= luggage_number
851
852     elif seat % 7 > 3:
853         nbins = lockers[sublocker][current_row-1][1]
854         lockers[sublocker][current_row-1][1] -= luggage_number
855
856     # derivations in writeup
```

```
857     if luggage_number == 1:
858         t = (4)/(1-(0.8*((nbins-2))/6)
859     if luggage_number == 2:
860         t = (4)/(1-(0.8*((nbins-2))/6) + (2.25)/(1-((nbins-2))/6)
861
862     return t
863 def check_locker_space(luggage_number, current_row, down, lockers):
864     # if passenger has no baggage
865     if luggage_number == 0:
866         return 0
867
868     # if on right side of aisle
869     if down==True:
870
871         nbins = lockers[NUM_ROWS-current_row-1][1]
872         lockers[NUM_ROWS-current_row-1][1] -= luggage_number
873     else:
874
875         nbins = lockers[NUM_ROWS-current_row-1][0]
876         lockers[NUM_ROWS-current_row-1][0] -= luggage_number
877     if luggage_number == 1:
878         t = (4)/(1-(0.8*((nbins-2))/6)
879     if luggage_number == 2:
880         t = (4)/(1-(0.8*((nbins-2))/6) + (2.25)/(1-((nbins-2))/6)
881
882     return t
883 def check_locker_space_Two(luggage_number, current_row, seat, lockers,sectionA):
884
885     if sectionA:
886         thingy = 0
887     else:
888         thingy = 1
889
890     # if passenger has no baggage
891     if luggage_number == 0:
892         return 0
893
894     # if on right side of aisle
895     if seat > 3:
896
897         nbins = lockers[thingy][current_row][1]
898         lockers[thingy][current_row][1] -= luggage_number
899     else:
900
901         nbins = lockers[thingy][current_row][0]
902         lockers[thingy][current_row][0] -= luggage_number
903     if luggage_number == 1:
904         t = (4)/(1-(0.8*((nbins-2))/6)
905     if luggage_number == 2:
906         t = (4)/(1-(0.8*((nbins-2))/6) + (2.25)/(1-((nbins-2))/6)
907
908     return t
909 def move_up(seat, row):
910
911     if seating_plan[seat-1][row][0] == -1:
912         seating_plan[seat-1][row] = seating_plan[seat][row]
913         seating_plan[seat-1][row][1] = 0
914         seating_plan[seat][row] = [-1,0]
915 def move_down(seat, row):
```

```
917
918     if seating_plan[seat+1][row][0] == -1:
919         seating_plan[seat+1][row] = seating_plan[seat][row]
920         seating_plan[seat+1][row][1] = 0
921         seating_plan[seat][row] = [-1,0]
922 def aisle_take_above(row,hello,world):
923     #Move person from above into aisle
924     seating_plan[AISLE_INDEX][row] = seating_plan[AISLE_INDEX-1][row]
925     seating_plan[AISLE_INDEX-1][row] = [-1,0]
926
927     #Luggage
928     seating_plan[AISLE_INDEX][row][INTERNAL_COCK] = -locker_shit_type[
929         boarding_type](seating_plan[AISLE_INDEX][row][HAS_LUGGAGE], row, True, lockers)
930
931     def aisle_take_below(row,frick,me):
932         #Move person from below into aisle
933         seating_plan[AISLE_INDEX][row] = seating_plan[AISLE_INDEX+1][row]
934         seating_plan[AISLE_INDEX+1][row] = [-1,0]
935
936         seating_plan[AISLE_INDEX][row][INTERNAL_COCK] = -locker_shit_type[
937             boarding_type](seating_plan[AISLE_INDEX][row][HAS_LUGGAGE], row, False, lockers)
938
939     def aisle_take_above_wide(row,current_aisles,uguil):
940         #Move person from above into aisle
941         #Luggage
942         seating_plan[current_aisles-1][row][INTERNAL_COCK] = -locker_shit_type[
943             boarding_type](seating_plan[current_aisles-1][row][HAS_LUGGAGE], row,
944             current_aisles-1, lockers)
945
946         seating_plan[current_aisles][row] = seating_plan[current_aisles-1][row]
947         seating_plan[current_aisles-1][row] = [-1,0]
948
949     def aisle_take_above_Two(row,current_aisles,SectionA):
950         #Move person from above into aisle
951         #Luggage
952         if row != 3:
953             seating_plan[current_aisles-1][row][INTERNAL_COCK] = -locker_shit_type[
954                 boarding_type](seating_plan[current_aisles-1][row][HAS_LUGGAGE], row-(GAP_SIZE
955                 +TWO_A_ROWS), current_aisles-1, lockers,SectionA)
956
957         seating_plan[current_aisles][row] = seating_plan[current_aisles-1][row]
958         seating_plan[current_aisles-1][row] = [-1,0]
959
960     def aisle_take_below_Two(row,current_aisles,sectionA):
961         #Move person from below into aisle
962         if row != 3:
963             seating_plan[current_aisles+1][row][INTERNAL_COCK] = -locker_shit_type[
964                 boarding_type](seating_plan[current_aisles+1][row][HAS_LUGGAGE], row-(GAP_SIZE
965                 +TWO_A_ROWS), current_aisles+1, lockers,sectionA)
966
967         seating_plan[current_aisles][row] = seating_plan[current_aisles+1][row]
968         seating_plan[current_aisles+1][row] = [-1,0]
969
970     def aisle_take_below_wide(row,current_aisles,aszgasdhasdh):
971         #Move person from below into aisle
972         if row != 0:
973             seating_plan[current_aisles+1][row][INTERNAL_COCK] = -locker_shit_type[
974                 boarding_type](seating_plan[current_aisles+1][row][HAS_LUGGAGE], row,
975                 current_aisles+1, lockers)
```

```
965 seating_plan[current_aisles][row] = seating_plan[current_aisles+1][row]
966 seating_plan[current_aisles+1][row] = [-1,0]
967
968
969
970
971
972 def aisle_take_left(row, current_aisles, whythefucknot):
973
974     #Move person from right into aisle
975     seating_plan[current_aisles][row] = seating_plan[current_aisles][row-1]
976     seating_plan[current_aisles][row-1] = [-1,0]
977     seating_plan[current_aisles][row][INTERNAL_COCK] = 0
978
979 def aisle_take_right(row, idonot, careanymore):
980
981     #Move person from right into aisle
982     seating_plan[AISLE_INDEX][row] = seating_plan[AISLE_INDEX][row+1]
983     seating_plan[AISLE_INDEX][row+1] = [-1,0]
984     seating_plan[AISLE_INDEX][row][INTERNAL_COCK] = 0
985
986 def aisle_take_right_wide(row, current_aisles, whythefucknot):
987
988     #Move person from right into aisle
989     seating_plan[current_aisles][row] = seating_plan[current_aisles][row+1]
990     seating_plan[current_aisles][row+1] = [-1,0]
991     seating_plan[current_aisles][row][INTERNAL_COCK] = 0
992
993 #While loop
994 def off_the_plane(generation_method, text):
995     global im, fig
996     #isual shot
997
998     global seating_plan
999     global priority_weightings
1000    global lockers
1001    test_cases = []
1002
1003
1004
1005    for i in range(N_TEST_CASES):
1006        generation_method()
1007        total_time=0
1008        left_plane = 0
1009        priority_weightings = generate_priorities(highest_priority_assigned)
1010        lockers = bag_shit_type[boarding_type]()
1011        if VISUALIZER:
1012            im, fig = render_type[boarding_type]()
1013
1014        while True:
1015            if boarding_type == TWO:
1016                #Exit square
1017                if seating_plan[2][41][PRIORITY] != -1 and seating_plan[2][41][
1018 INTERNAL_COCK] >= TIME_TO_MOVE:
1019                    #Empty square
1020                    seating_plan[2][41] = [-1,0]
1021                    left_plane += 1
1022
1023                    if seating_plan[6][41][PRIORITY] != -1 and seating_plan[6][41][
1024 INTERNAL_COCK] >= TIME_TO_MOVE:
```

```

1023             #Empty square
1024             seating_plan[6][41] = [-1,0]
1025             left_plane += 1
1026             if seating_plan[TWO_SEATS-1][3][PRIORITY] != -1 and seating_plan[TWO_SEATS-1][3][INTERNAL_COCK] >= TIME_TO_MOVE:
1027                 #Empty square
1028                 seating_plan[TWO_SEATS-1][3] = [-1,0]
1029                 left_plane += 1
1030
1031
1032             #End code shot
1033             if left_plane == 249:
1034                 test_cases.append(total_time)
1035                 break
1036
1037             for current_aisle in (range(TWO_SEATS)):
1038                 if seating_plan[current_aisle][3][PRIORITY] == -1:
1039
1040                     priorities = [0,0,0,0]
1041                     possible_moves = [aisle_take_above_Two,
1042 aisle_take_below_Two, aisle_take_right_wide,aisle_take_left ]
1043                     total_move_possibilites = 0
1044
1045                     #Get things to check
1046                     is_person_above_moving = seating_plan[current_aisle-1][3][PRIORITY] != -1 and seating_plan[current_aisle-1][3][INTERNAL_COCK] >=
TIME_TO_MOVE_PAST_SEAT
1047
1048                     if is_person_above_moving:
1049                         priorities[0] = priority_weightings[seating_plan[current_aisle-1][3][PRIORITY]-1]
1050                         total_move_possibilites +=1
1051                     if current_aisle != 8:
1052                         is_person_below_moving = False #seating_plan[current_aisle+1][3][PRIORITY] != -1 and seating_plan[current_aisle+1][3][INTERNAL_COCK] >= TIME_TO_MOVE_PAST_SEAT
1053                         else: is_person_below_moving = 0
1054                     if is_person_below_moving:
1055
1056                         priorities[1] = 0 #priority_weightings[
1057 seating_plan[current_aisle+1][3][PRIORITY]-1]
1058                         total_move_possibilites +=1
1059                         #Prevent indexing error
1060                         if True: #row+1+ec != TWO_ROWS:
1061                             is_person_right_moving = seating_plan[current_aisle][3+1][PRIORITY] != -1 and seating_plan[current_aisle][3+1][INTERNAL_COCK] >= TIME_TO_MOVE
1062                         else:
1063                             is_person_right_moving = 0
1064
1065                         if is_person_right_moving and (current_aisle == 2 or
1066 current_aisle == 6):
1067
1068                             priorities[2] = priority_weightings[seating_plan[current_aisle][3+1][PRIORITY]-1]
1069                             total_move_possibilites +=1
1070                             #ewginsdaogvnadsklbvasj nwklfnwdsf
1071                             if True: #row+1+ec != TWO_ROWS:
1072                                 is_person_left_moving = seating_plan[current_aisle][3-1][PRIORITY] != -1 and seating_plan[current_aisle][3-1][

```

```

INTERNAL_COCK] >= TIME_TO_MOVE
1070
1071     else:
1072         is_person_right_moving = 0
1073
1074     if is_person_left_moving and (current_aisle == 2 or
1075         current_aisle == 6):
1076         priorities[3] = priority_weightings[seating_plan[
1077             current_aisle][3-1][PRIORITY]-1]
1078         total_move_possibilities +=1
1079
1080     #Decide who moves above and below
1081     if total_move_possibilities > 0:
1082         #Reset time
1083         seating_plan[current_aisle][3][INTERNAL_COCK] = 0
1084
1085         #frick knows what is happening here but it works
1086         so_it_stays
1087
1088         move = numpy.argwhere(priorities == numpy.amax(
1089             priorities))
1090
1091         possible_moves[(random.choice(move))[0]](3,
1092             current_aisle, False)
1093
1094
1095         for current_aisle in range(2, TWO_SEATS, 4):
1096
1097             for row in reversed(range(0, TWO_B_ROWS)):
1098
1099                 #extra constant
1100                 ec = GAP_SIZE+TWO_A_ROWS
1101                 #Check if aisles place is empty
1102                 if seating_plan[current_aisle][row+ec][PRIORITY] == -1:
1103
1104                     priorities = [0,0,0]
1105                     possible_moves = [aisle_take_above_Two,
1106                         aisle_take_below_Two, aisle_take_left]
1107                     total_move_possibilities = 0
1108
1109                     #Get things to check
1110                     is_person_above_moving = seating_plan[current_aisle
1111                         -1][row+ec][PRIORITY] != -1 and seating_plan[current_aisle-1][row+ec][
1112                         INTERNAL_COCK] >= TIME_TO_MOVE_PAST_SEAT
1113
1114                     if is_person_above_moving:
1115                         priorities[0] = priority_weightings[seating_plan[
1116                             current_aisle-1][row+ec][PRIORITY]-1]
1117                         total_move_possibilities +=1
1118
1119                     is_person_below_moving = seating_plan[current_aisle
1120                         +1][row+ec][PRIORITY] != -1 and seating_plan[current_aisle+1][row+ec][
1121                         INTERNAL_COCK] >= TIME_TO_MOVE_PAST_SEAT
1122
1123                     if is_person_below_moving:
1124
1125                         priorities[1] = priority_weightings[seating_plan[
1126                             current_aisle+1][row+ec][PRIORITY]-1]
1127                         total_move_possibilities +=1
1128                         #Prevent indexing error
1129                         if True: #row+1+ec != TWO_ROWS:
1130                             is_person_right_moving = seating_plan[
1131                                 current_aisle][row-1+ec][PRIORITY] != -1 and seating_plan[current_aisle][row
1132                                     -1+ec][INTERNAL_COCK] >= TIME_TO_MOVE
1133
1134                         else:

```



```

1161                     if is_person_right_moving:
1162
1163                         priorities[2] = priority_weightings[
seating_plan[current_aisle][row+1][PRIORITY]-1]
1164                             total_move_possibilites +=1
1165                             #ewginsdaognadsklbvasj nwklhsfnwdsf
1166                             if True: #row+1+ec != TWO_ROWS:
1167                                 is_person_left_moving = seating_plan[
current_aisle][row-1][PRIORITY] != -1 and seating_plan[current_aisle][row-1][
INTERNAL_COCK] >= TIME_TO_MOVE
1168                             else:
1169                                 is_person_right_moving = 0
1170
1171                         if is_person_left_moving and row == 3:
1172                             priorities[3] = priority_weightings[
seating_plan[current_aisle][row-1][PRIORITY]-1]
1173                             total_move_possibilites +=1
1174
1175                             #Decide who moves above and below
1176                             if total_move_possibilites > 0:
1177                                 #Reset time
1178                                 seating_plan[current_aisle][row][
INTERNAL_COCK] = 0
1179
1180                         #frick knows what is happening here but it
1181                         works so it stays
1182                         move = numpy.argwhere(priorities == numpy.
amax(priorities))
1183                         possible_moves[(random.choice(move))[0]](row,
current_aisle, False)
1184
1185                         #Move towards aisle
1186                         for row in range(TWO_ROWS):
1187                             if row!= 3:
1188                                 if seating_plan[0][row][INTERNAL_COCK] >=
TIME_TO_MOVE_PAST_SEAT:
1189                                     move_down(0,row)
1190                                     if seating_plan[TWO_SEATS-1][row][INTERNAL_COCK] >=
TIME_TO_MOVE_PAST_SEAT and seating_plan[TWO_SEATS-1][row][PRIORITY] != -1:
1191                                         move_up(TWO_SEATS-1, row)
1192                                         if seating_plan[4][row][INTERNAL_COCK] >=
TIME_TO_MOVE_PAST_SEAT:
1193                                             if random.randint(0,1) and seating_plan[3][row][
PRIORITY] == -1:
1194                                                 move_up(4, row)
1195                                             elif seating_plan[5][row][PRIORITY] == -1:
1196                                                 move_down(4, row)
1197
1198
1199
1200                         for i in range(TWO_SEATS):
1201                             for k in range(TWO_ROWS):
1202                                 seating_plan[i][k][INTERNAL_COCK] += TIME_STEP
1203
1204
1205
1206
1207
1208

```

```
1209
1210
1211
1212     elif boarding_type == WIDEBODY:
1213         #Exit square
1214         if seating_plan[0][0][PRIORITY] != -1 and seating_plan[0][0][
1215 INTERNAL_COCK] >= TIME_TO_MOVE:
1216             #Empty square
1217             seating_plan[0][0] = [-1,0]
1218
1219             #End code shot
1220             left_plane += 1
1221             if left_plane == (WIDE_WING_ROWS -1)*(WIDE_WING_SEATS -4) -18:
1222                 test_cases.append(total_time)
1223
1224             break
1225
1226             #Down queue do not touch 2am code
1227             for seat in range(WIDE_WING_SEATS):
1228
1229                 #if can move something in
1230                 if seating_plan[seat][0][PRIORITY] == -1:
1231
1232                     priorities = [0,0]
1233                     possible_moves = [ aisle_take_below_wide ,
aisle_take_right_wide ]
1234                     total_move_possibilities = 0
1235
1236                     #Check shot
1237                     is_person_below_moving = seating_plan[seat][1][PRIORITY]
1238 ! = -1 and seating_plan[seat][1][INTERNAL_COCK] >= TIME_TO_MOVE_PAST_SEAT and
seat in [3,10,17,24]
1239                     if is_person_below_moving:
1240
1241                         priorities[1] = priority_weightings[seating_plan[seat]
1242 ] [1][PRIORITY]-1]
1243                         total_move_possibilities +=1
1244                         #Prevent indexing error
1245                         if seat != WIDE_WING_SEATS-1:
1246                             is_person_right_moving = seating_plan[seat+1][0][
1247 PRIORITY] != -1 and seating_plan[seat+1][0][INTERNAL_COCK] >= TIME_TO_MOVE
1248                             else:
1249                                 is_person_right_moving = 0
1250
1251                         if is_person_right_moving:
1252
1253                             priorities[0] = priority_weightings[seating_plan[seat
1254 ] +1][0][PRIORITY]-1]
1255                             total_move_possibilities +=1
1256
1257                             #Decide who moves above and below
1258                             if total_move_possibilities > 0:
1259                                 #Reset time
1260                                 seating_plan[seat][0][INTERNAL_COCK] = 0
1261
1262                                 #frick knows what is happening here but it works so
1263                                 it stays
1264                                 move = numpy.argwhere(priorities == numpy.amax(
1265 priorities))
1266                                 possible_moves[(random.choice(move))[0]](0,seat,False)
```

```
)  
1260  
1261  
1262  
1263     for current_aisle in range(3, WIDE_WING_SEATS,7):  
1264  
1265  
1266     for row in range(1,WIDE_WING_ROWS):  
1267  
1268  
1269  
1270  
1271         #Check if aisles place is empty  
1272         if seating_plan[current_aisle][row][PRIORITY] == -1:  
1273  
1274             priorities = [0,0,0]  
1275             possible_moves = [aisle_take_above_wide,  
aisle_take_below_wide, aisle_take_right_wide ]  
1276             total_move_possibilities = 0  
1277  
1278             #Get things to check  
1279             is_person_above_moving = seating_plan[current_aisle  
-1][row][PRIORITY] != -1 and seating_plan[current_aisle-1][row][INTERNAL_COCK]  
1280             >= TIME_TO_MOVE_PAST_SEAT  
1281  
1282                 if is_person_above_moving:  
1283                     priorities[0] = priority_weightings[seating_plan[  
current_aisle-1][row][PRIORITY]-1]  
1284                     total_move_possibilities +=1  
1285  
1286                 is_person_below_moving = seating_plan[current_aisle  
+1][row][PRIORITY] != -1 and seating_plan[current_aisle+1][row][INTERNAL_COCK]  
1287             >= TIME_TO_MOVE_PAST_SEAT  
1288                 if is_person_below_moving:  
1289  
1290                     priorities[1] = priority_weightings[seating_plan[  
current_aisle+1][row][PRIORITY]-1]  
1291                     total_move_possibilities +=1  
1292                     #Prevent indexing error  
1293                     if row != WIDE_WING_ROWS-1:  
1294                         is_person_right_moving = seating_plan[  
current_aisle][row+1][PRIORITY] != -1 and seating_plan[current_aisle][row+1][  
INTERNAL_COCK] >= TIME_TO_MOVE  
1295                     else:  
1296                         is_person_right_moving = 0  
1297  
1298                     if is_person_right_moving:  
1299  
1300                         priorities[2] = priority_weightings[seating_plan[  
current_aisle][row+1][PRIORITY]-1]  
1301                         total_move_possibilities +=1  
1302  
1303                         #Decide who moves above and below  
1304                         if total_move_possibilities > 0:  
1305                             #Reset time  
1306                             seating_plan[current_aisle][row][INTERNAL_COCK] =  
1307                             0  
1308  
1309                             #frick knows what is happening here but it works  
1310                             so it stays
```

```
1307                         move = numpy.argwhere(priorities == numpy.amax(
1308     priorities))
1309                         possible_moves[(random.choice(move))[0]](row,
1310     current_aisle, False)
1311
1312             # I fricking HATE INDENDATION
1313
1314
1315             #Get total amount of move possibilites
1316             #total_move_possibilites = is_person_above_moving +
1317             is_person_below_moving + is_person_right_moving
1318
1319
1320
1321             for current_aisle in range(0, WIDE_WING_SEATS, 7):
1322
1323                 # Move down
1324                 for seat in reversed(range(0, 2)):
1325                     # Loops through all 37 rows
1326
1327                     #Move towards aisle
1328                     for row in range(0, WIDE_WING_ROWS):
1329                         if seating_plan[seat+current_aisle][row][
1330 INTERNAL_COCK] >= TIME_TO_MOVE_PAST_SEAT:
1331                             move_down(seat+current_aisle, row)
1332
1333                         #Increases internal clock
1334
1335
1336
1337             # Move up
1338             for seat in range(5, 7):
1339                 # Loops through all 37 rows
1340
1341                 #Move towards aisle
1342                 for row in range(0, WIDE_WING_ROWS):
1343                     if seating_plan[seat+current_aisle][row][
1344 INTERNAL_COCK] >= TIME_TO_MOVE_PAST_SEAT:
1345                         move_up(seat+current_aisle, row)
1346
1347                         #Increases internal clock
1348                         for i in range(WIDE_WING_SEATS):
1349                             for k in range(WIDE_WING_ROWS):
1350                                 seating_plan[i][k][INTERNAL_COCK] += TIME_STEP
1351
1352
1353
1354             elif boarding_type == NORMAL:
1355
1356                 #Exit square
1357                 if seating_plan[AISLE_INDEX][0][PRIORITY] != -1 and seating_plan
1358 [3][0][INTERNAL_COCK] >= TIME_TO_MOVE:
1359                     #Empty square
1360                     seating_plan[AISLE_INDEX][0] = [-1, 0]
1361
1362
1363
1364             #End code shot
1365             left_plane += 1
1366             if left_plane == NUM_ROWS*NUM_SEATS:
1367                 test_cases.append(total_time)
1368
1369             break
```

```
1361
1362
1363     #Aisle handling code
1364     for row in range(0,NUM_ROWS):
1365         #Check if aisles place is empty
1366         if seating_plan[AISLE_INDEX][row][PRIORITY] == -1:
1367
1368             priorities = [0,0,0]
1369             possible_moves = [aisle_take_above, aisle_take_below,
1370 aisle_take_right ]
1371             total_move_possibilities = 0
1372
1373             #Get things to check
1374             is_person_above_moving = seating_plan[AISLE_INDEX-1][row]
1375             [PRIORITY] != -1 and seating_plan[AISLE_INDEX-1][row][INTERNAL_COCK] >=
1376             TIME_TO_MOVE_PAST_SEAT
1377
1378                 if is_person_above_moving:
1379                     priorities[0] = priority_weightings[seating_plan[
1380 AISLE_INDEX-1][row][PRIORITY]-1]
1381                     total_move_possibilities +=1
1382
1383             is_person_below_moving = seating_plan[AISLE_INDEX+1][row]
1384             [PRIORITY] != -1 and seating_plan[AISLE_INDEX+1][row][INTERNAL_COCK] >=
1385             TIME_TO_MOVE_PAST_SEAT
1386
1387                 if is_person_below_moving:
1388
1389                     priorities[1] = priority_weightings[seating_plan[
1390 AISLE_INDEX+1][row][PRIORITY]-1]
1391                     total_move_possibilities +=1
1392
1393             #Prevent indexing error
1394             if row != NUM_ROWS-1:
1395                 is_person_right_moving = seating_plan[AISLE_INDEX][
1396             row+1][PRIORITY] != -1 and seating_plan[AISLE_INDEX][row+1][INTERNAL_COCK] >=
1397             TIME_TO_MOVE
1398
1399                 else:
1400                     is_person_right_moving = 0
1401
1402                 if is_person_right_moving:
1403
1404                     priorities[2] = priority_weightings[seating_plan[
1405 AISLE_INDEX][row+1][PRIORITY]-1]
1406                     total_move_possibilities +=1
1407
1408             #Get total amount of move possibilites
1409             #total_move_possibilities = is_person_above_moving +
1410             is_person_below_moving + is_person_right_moving
1411
1412             #Decide who moves above and below
1413             if total_move_possibilities > 0:
1414                 #Reset time
1415                 seating_plan[AISLE_INDEX][row][INTERNAL_COCK] = 0
1416
1417                 #frick           knows what is happening here but it
1418                 works so it stays
1419                 move = numpy.argwhere(priorities == numpy.amax(
1420 priorities))
```

```
1408                     possible_moves[(random.choice(move))[0]](row, False,
1409                         False)
1410
1411
1412             # Move down
1413             for seat in reversed(range(0,2)):
1414                 # Loops through all 37 rows
1415
1416                 #Move towards aisle
1417                 for row in range(0,NUM_ROWS):
1418                     if seating_plan[seat][row][INTERNAL_COCK] >=
1419                         TIME_TO_MOVE_PAST_SEAT:
1420                             move_down(seat, row)
1421                             #Increases internal clock
1422
1423             # Move up
1424             for seat in range(5,7):
1425                 # Loops through all 37 rows
1426
1427                 #Move towards aisle
1428                 for row in range(0,NUM_ROWS):
1429                     if seating_plan[seat][row][INTERNAL_COCK] >=
1430                         TIME_TO_MOVE_PAST_SEAT:
1431                             move_up(seat, row)
1432                             #Increases internal clock
1433
1434             for i in range(NUM_SEATS+1):
1435                 for k in range(NUM_ROWS):
1436                     seating_plan[i][k][INTERNAL_COCK] += TIME_STEP
1437
1438
1439
1440
1441             total_time += TIME_STEP
1442
1443
1444             #Update render comment out if not using
1445             if VISUALIZER: update_render(seating_plan)
1446             print(text + str(sum(test_cases)/len(test_cases)))
1447             rows.append(test_cases)
1448             return test_cases
1449
#Types
1450 render_type = [intalize_render,intalize_render_widebody,intalize_render_two_thing
1451 ]
1452 bag_shit_type = [bag_shit, bag_shit_wide,bag_shit_Two]
1453 locker_shit_type = [check_locker_space,check_locker_space_wide,
1454     check_locker_space_Two]
1455
#What thing to do
1456 NORMAL = 0
1457 WIDEBODY = 1
1458 TWO = 2
1459
#Test stuff
1460 N_TEST_CASES = 50
1461 VISUALIZER = True
1462 TIME_STEP = 0
1463 boarding_type = TWO
1464
#Data csv
1465 import csv
```

```
1463 fields = []
1464 rows = []
1465 index = []
1466 #Add the indexing
1467 for i in range(N_TEST_CASES):
1468     index.append(i)
1469
1470 rows.append(index)
1471 #Vroom
1472 seating_plan = []
1473 ''
1474 ''
1475 #Flat body
1476 off_the_plane(random_deboard, 'Random: ')
1477 off_the_plane(sections, 'Sections: ')
1478 off_the_plane(reverse_wilma, 'Reverse Wilma: ')
1479 off_the_plane(generate_front_to_back, 'Front to back Row: ')
1480 off_the_plane(back_to_front, 'Back to Front Row: ')
1481 # field names add whatever field names that you are creating data for
1482 fields = ['Index', 'Random Group Adjusted', 'Front to back - sections', 'Reverse
    Wilma', 'Front to Back Row', 'Back to Front Row']
1483 file_name = 'narrow.csv'
1484
1485
1486 #Wide body
1487 off_the_plane(random_deboard_widebody, 'Random: ')
1488 off_the_plane(sections_widebody, 'Sections: ')
1489 off_the_plane(reverse_wilma_widebody, 'Reverse Wilma: ')
1490 off_the_plane(generate_front_to_back_widebody, 'Front seat to back seat: ')
1491 off_the_plane(back_to_front_widebody, 'Back seat to Front seat: ')
1492 off_the_plane(across_widebody, 'Across: ')
1493 # field names add whatever field names that you are creating data for
1494 fields = ['Index', 'Random Group Adjusted', 'Front to back - sections', 'Reverse
    Wilma', 'Front to Back Row', 'Back to Front Row', 'Across']
1495 file_name='widebody.csv'
1496 ''
1497 # field names add whatever field names that you are creating data for
1498 fields = ['Index', 'Back to front', 'Sections', 'Random', 'Reverse Wilma', 'Front
    to back']
1499 off_the_plane(two_reverse_sections_360, 'back to front')
1500 #off_the_plane(two_reverse_sections_360, 'Sections')
1501 #
1502 #off_the_plane(two_front_to_back, 'front to back')
1503 #off_the_plane(two_reverse_wilma_widebody, 'reverse wilma')
1504 #off_the_plane(two_random, 'random')
1505 file_name='twoaisles.csv'
1506
1507 ''
1508 nbsensitivity = []
1509
1510 for i in range(0,41):
1511     NAUGHTY_BOY_COEFFICIENT = (i*2.5)/100
1512
1513     # put method wanted in here
1514     nbsensitivity.append(off_the_plane(back_to_front, 'back to front: '))
1515
1516     print('for test with NB coefficient {}'.format((i*2.5)/100))
1517 ''
1518 ''
1519 #Makes the shotinto colums honestly magic
```

```

1520 rows = zip(*rows)
1521 #Create the rows
1522 with open(file_name, 'w', newline='') as f:
1523
1524     # using csv.writer method from CSV package
1525     write = csv.writer(f)
1526
1527     write.writerow(fields)
1528
1529     write.writerows(rows)
1530 ...

```

## Appendix F

```

1 import random
2 import matplotlib.pyplot as plt
3 import numpy
4 import math
5
6 # visualizer things
7
8 # render stuff that I don't understand
9 def intalize_render():
10
11     global plane
12
13     #Absolute mess of code
14     image = []
15     for i in range(NUM_SEATS+len(AISLES)):
16         subimage = []
17         for k in range(NUM_ROWS):
18             if k % 2 == 0:
19                 subimage.append(-1)
20             else:
21                 subimage.append(0)
22
23         image.append(subimage)
24
25
26     fig,ax = plt.subplots(1,1)
27     plt.set_cmap('OrRd')
28     print(image)
29     image = numpy.array(image)
30
31     im = ax.imshow(image)
32     number_of_runs = range(1,NUM_ROWS)      # use your actual number_of_runs
33     ax.set_xticks(number_of_runs, minor=False)
34     ax.xaxis.grid(True, which='major')
35
36
37
38
39     ax.set_yticks(numpy.arange(0.5, NUM_SEATS+len(AISLES)+.5, 1).tolist(), minor=False)
40     ax.yaxis.grid(True, which='major')
41
42     if plane == 'wide wing':
43         ax.set_yticklabels(['A','B','C','Aisle','D','E','F','G','H','I','Aisle','J','K','L','M','N','O','Aisle','P','Q','R','S','T','U','Aisle','V','W','X'])

```

```
44     elif plane == 'narrow body':
45         ax.set_yticklabels(['Row A', 'Row B', 'Row C', 'Aisle', 'Row D', 'Row E', 'Row
46 F'])
47     elif plane == 'two entrance two aisle':
48         ax.set_yticklabels(['Row A', 'Row B', 'Aisle', 'Row C', 'Row D', 'Row E', 'Row
49 F', 'Row G'])
50     ax.set_ylim(top=-0.5)
51
52
53
54     ax.set_xticks(numpy.arange(0.5, NUM_ROWS+.5, 1).tolist(), minor=False)
55     ax.xaxis.grid(True, which='major')
56
57     xticklist = []
58     #Create list of numbers between
59     for i in range(NUM_ROWS):
60         if ((i+1) % 5 == 0) and (i != 0):
61             xticklist.append(str(i+1))
62         else:
63             xticklist.append(',')
64
65     ax.set_xticklabels(xticklist)
66     ax.set_xlim(left=-0.5)
67
68
69     return im,fig
70 def update_render(seat_plan):
71
72     visualizer = []
73     for i,column in enumerate(seat_plan):
74         visualizer.append([])
75         for seat in column:
76             if i not in AISLES:
77                 if seat != -1:
78                     visualizer[i].append(0)
79                 else: visualizer[i].append(-1)
80             else:
81                 if seat != ',':
82                     visualizer[i].append(0)
83                 else: visualizer[i].append(-1)
84
85     im.set_data(visualizer)
86     fig.canvas.draw_idle()
87     plt.pause(0.01)
88
89
90
91
92
93
94
95 # -----
96 # stuff to board the plane with (given a boarding queue)
97 # -----
98
99 # calculate time taken to get to seat if someone in the way
100 def get_past_people(seating_plan, passenger, current_row):
```

```
102     # number of people blocking seats
103     N=0
104     time_to_stop_blocking_aisle = 0
105
106
107     # aisle seat
108     for aisle in AISLES:
109         if abs(passenger[1] - aisle) == 1:
110             time_to_stop_blocking_aisle += TIME_TO_MOVE_PAST_SEAT
111     # middle or window seat: people are in the way
112     else:
113
114         for aisle in AISLES:
115
116             if passenger[1]-aisle == -3:
117
118                 # if aisle seat taken IMPORTANT to check aisle seat first so f is
119                 # maximised
120                 if seating_plan[passenger[1]+1][NUM_ROWS-current_row-1] != -1:
121                     N+=1
122                     f=1
123                     # if middle seat taken
124                     if seating_plan[passenger[1]+2][NUM_ROWS-current_row-1] != -1:
125                         N+=1
126                         f=2
127
128                     break
129
130             elif passenger[1]-aisle == -2:
131
132                 # if aisle seat taken
133                 if seating_plan[passenger[1]+1][NUM_ROWS-current_row-1] != -1:
134                     N+=1
135                     f=1
136
137                 # window seat F
138                 elif passenger[1]-aisle == 3:
139                     # if aisle seat taken IMPORTANT to check aisle seat first so f is
140                     # maximised
141                     if seating_plan[passenger[1]-2][NUM_ROWS-current_row-1] != -1:
142                         N+=1
143                         f=1
144                         # if middle seat taken
145                         if seating_plan[passenger[1]-1][NUM_ROWS-current_row-1] != -1:
146                             N+=1
147                             f=2
148
149                 # middle seat B
150                 elif passenger[1] == 2:
151                     # if aisle seat taken
152                     if seating_plan[passenger[1]-1][NUM_ROWS-current_row-1] != -1:
153                         N+=1
154                         f=1
155
156             if N==0:
157                 time_to_stop_blocking_aisle = TIME_TO_MOVE_PAST_SEAT
158             else:
159                 time_to_stop_blocking_aisle += TIME_TO_SIT_OR_STAND +
```

```
TIME_TO_MOVE_PAST_SEAT*(N+f+1)

160
161     return time_to_stop_blocking_aisle, N
162 # stow in overhead lockers
163 def check_locker_space(passenger, current_row, lockers, passengers_loaded_bags,
164     aisle=0):
165
166     # if passenger has no baggage
167     if passenger[2] == 0:
168         return 0
169
170     # if on right side of aisle
171
172     for aisle in AISLES:
173
174         if abs(passenger[1]-aisle) <= 3:
175
176             correct_aisle = aisle
177             break
178
179
180         if passenger[1] > correct_aisle:
181             if [passenger[0],passenger[1]] not in passengers_loaded_bags:
182                 nbins = lockers[AISLES.index(correct_aisle)][NUM_ROWS-current_row
183 -1][1]
184                 lockers[AISLES.index(correct_aisle)][NUM_ROWS-current_row-1][1] +=
185 passengers[2]
186                 passengers_loaded_bags.append([passenger[0],passenger[1]])
187             else:
188                 nbins = lockers[AISLES.index(correct_aisle)][NUM_ROWS-current_row
189 -1][0]-passenger[2]
190
191         if [passenger[0],passenger[1]] not in passengers_loaded_bags:
192             nbins = lockers[AISLES.index(correct_aisle)][NUM_ROWS-current_row
193 -1][0]
194                 lockers[AISLES.index(correct_aisle)][NUM_ROWS-current_row-1][0] +=
195 passengers[2]
196                 passengers_loaded_bags.append([passenger[0],passenger[1]])
197             else:
198                 nbins = lockers[AISLES.index(correct_aisle)][NUM_ROWS-current_row
199 -1][0]-passenger[2]
200
201     # derivations in writeup
202
203
204     if passenger[2] == 1:
205         t = (4)/(1-(0.8*nbins)/6)
206     if passenger[2] == 2:
207         t = (4)/(1-(0.8*nbins)/6) + (2.25)/(1-(nbins+1)/6)
208
209     return t
210
211 # board the plane
212 def board_the_plane(boardingQueue, family=False):
213     # initialize seating plan, top queue (if multiple aisles) and overhead
214     # lockers
215     seating_plan = [[-1 for _ in range(NUM_ROWS)] for _ in range(NUM_SEATS + len
216     (AISLES))]
217     for aisle in AISLES:
218         seating_plan[aisle]=['' for _ in range(NUM_ROWS)]
```

```
210 top_queue = ['' for _ in range(NUM_SEATS+len(AISLES))]  
211 lockers = [[[0,0] for i in range(NUM_ROWS)] for j in range(len(AISLES))]  
212 seated = []  
213 passengers_loaded_bags = []  
214  
215 n_passengers = len(boardingQueue)  
216  
217 total_time=0  
218  
219 # this is false for all scenarios except where families are prioritized  
220 if family == False:  
221     time_to_move = TIME_TO_MOVE  
222 else:  
223     time_to_move = FAMILY_TIME_TO_MOVE  
224  
225 while True:  
226  
227     #print(seating_plan)  
228  
229     # loop through top queue  
230     for current_column, passenger in enumerate(reversed(top_queue)):  
231  
232         if passenger != '':  
233  
234             # increase internal clock  
235             passenger[3] += TIME_STEP  
236  
237             # check if passenger in right aisle and thus they can seat  
238             if (NUM_SEATS+len(AISLES)-current_column-1) in AISLES and abs(  
passenger[1]-(NUM_SEATS+len(AISLES)-current_column-1))<=AISLES[0]+1:  
239  
240                 # move into aisle  
241                 if seating_plan[NUM_SEATS+len(AISLES)-current_column-1][0] ==  
'' and passenger[3] >= time_to_move:  
242                     #reset internal clock  
243                     passenger[3]=0  
244                     seating_plan[NUM_SEATS+len(AISLES)-current_column-1][0] =  
passenger  
245                     top_queue[NUM_SEATS+len(AISLES)-current_column-1]=''  
246  
247             else:  
248                 # if passenger in front has moved  
249                 if top_queue[NUM_SEATS+len(AISLES)-current_column] == '' and  
passenger[3] >= time_to_move:  
250                     # move people along  
251                     top_queue[NUM_SEATS+len(AISLES)-current_column] =  
passenger  
252                     top_queue[NUM_SEATS+len(AISLES)-current_column-1] = ''  
253  
254                     # reset internal clock  
255                     passenger[3] = 0  
256  
257             for aisle in AISLES:  
258  
259                 # loop through aisle from back to front  
260                 for current_row,passenger in enumerate(reversed(seating_plan[aisle])):  
261  
262                     if passenger != '':
```

```
264         # increase internal clock
265         #print(seating_plan[aisle])
266         passenger[3] += TIME_STEP
267
268         # check if passenger in right row and thus they can seat
269         if passenger[0] == NUM_ROWS - current_row:
270
271             # if passenger has baggage
272             time_to_stow = check_locker_space(passenger, current_row,
273             lockers, passengers_loaded_bags)
274
275             # time it takes to stop blocking aisle and number of
276             # people in the way
277             try:
278                 time_to_stop_blocking_aisle=passenger[5]
279             except:
280                 time_to_stop_blocking_aisle, N = get_past_people(
281                 seating_plan, passenger, current_row)
282                 passenger.append(time_to_stop_blocking_aisle)
283
284             # make sure there is an empty space
285             if N==2 and current_row != 0 and seating_plan[aisle][
286             NUM_ROWS-current_row] != '' and current_row != 0:
287                 time_to_wait_for_spot_in_aisle += time_to_move -
288                 passenger[3]
289             else:
290                 time_to_wait_for_spot_in_aisle=0
291
292             # if time to wait has finished i.e. SIT DOWN BE HUMBLE
293             if passenger[3] >= time_to_stop_blocking_aisle +
294             time_to_stow + time_to_wait_for_spot_in_aisle:
295
296                 seating_plan[passenger[1]][passenger[0]-1] =
297                 passenger
298                 seated.append(passenger)
299                 #print(seated)
300
301                 # set queue place to empty
302                 seating_plan[aisle][NUM_ROWS-current_row-1]=''
303
304             else:
305                 # if passenger in front has moved
306                 if seating_plan[aisle][NUM_ROWS-current_row] == '' and
307                 passenger[3] >= time_to_move:
308
309                     # move people along
310                     seating_plan[aisle][NUM_ROWS-current_row] = passenger
311                     seating_plan[aisle][NUM_ROWS-current_row-1] = ''
312
313                     # reset internal clock
314                     seating_plan[aisle][NUM_ROWS - current_row][3] = 0
315
316             if VISUALIZER: update_render(seating_plan)
317
318             total_time += TIME_STEP
```

```
316     if len(seated) == n_passengers:
317         return total_time
318
319
320
321     if top_queue[0] == '' and len(boardingQueue)!=0:
322
323         # only considered in method where families board first.
324         if family == True and boardingQueue[0] == 'b':
325             time_to_move = NON_FAMILY_TIME_TO_MOVE
326             boardingQueue.pop(0)
327
328             #Set first place in isle to the first passenger in the seat data
329             seating_plan[3] then remove it from seat data
330             top_queue[0] = boardingQueue[0]
331             boardingQueue.pop(0)
332
333
334 # luggage
335 def assign_luggage():
336     return random.choices([0,1,2], weights=BAG_COEFFICIENT, k=1)[0]
337
338 # naughty boy
339 def is_not_disobedient():
340     return random.randrange(100) > NAUGHTY_BOY_COEFFICIENT*100
341 # create a group size
342 def group_size(group_weights):
343
344     return random.choices([1,2,3], weights=group_weights, k=1)[0]
345
346 # return average of list
347 def average(x):
348     return sum(x)/len(x)
349
350
351
352
353
354
355
356 # create order of boarding
357 def create_boarding_order_for_section_but_with_groups(boarding_section,
358     other_section1, other_section2, start_row, end_row):
359     current_group_member = 0
360     current_group_section = 1
361     current_group_size = 1
362     boarding_section.append([])
363
364     for row in range(start_row,end_row+1):
365         for seat in range(0,NUM_SEATS+len(AISLES)):
366
367             if seat not in AISLES:
368
369                 current_group_member += 1
370
371                 if current_group_section == 1:
372                     boarding_section[-1].append([row,seat,assign_luggage(),0])
373                 elif current_group_section == 2:
374                     other_section1[-1].append([row,seat,assign_luggage(),0])
```

```
374         elif current_group_section == 3:
375             other_section2[-1].append([row, seat, assign_luggage(), 0])
376
377     if current_group_member == current_group_size:
378
380         for aisle in AISLES:
381
383             if seat - aisle in [2, 3]:
384                 if current_group_section == 1:
385                     boarding_section[-1].reverse()
386                 elif current_group_section == 2:
387                     other_section1[-1].reverse()
388                 elif current_group_section == 3:
389                     other_section2[-1].reverse()
390
392             if seat - aisle in [-3, -2, 1]:
393                 current_group_size = group_size((
394 SINGLE_PRINGLE_COEFFICIENT, COUPLES_COEFFICIENT, THREESOME_COEFFICIENT))
395                 elif seat - aisle in [-1, 2]:
396                     current_group_size = group_size((
397 SINGLE_PRINGLE_COEFFICIENT, COUPLES_COEFFICIENT, 0))
398                     elif seat == 6:
399                         current_group_size = 1
400
401             current_group_member = 0
402
403         if is_not_disobedient():
404             current_group_section = 1
405             boarding_section.append([])
406
407             # else they try board during different sections
408             else:
409                 if random.randrange(100) < 50:
410                     current_group_section = 2
411                     other_section1.append([])
412                 else:
413                     current_group_section = 3
414                     other_section2.append([])
415
416
417
418
419
420 # create order of boarding for doing windows first
421 def create_boarding_order_for_aisle(boarding_section, other_section1,
422     other_section2, seats):
423     for seat in seats:
424
425         for row in range(1, NUM_ROWS+1):
426
427             # if passenger is not useless
428             if is_not_disobedient():
429                 boarding_section.append([row, seat, assign_luggage(), 0])
430             # else they try board during different sections
431             else:
```

```
431         if random.randrange(100) < 50:
432             other_section1.append([row, seat, assign_luggage(), 0])
433         else:
434             other_section2.append([row, seat, assign_luggage(), 0])
435
436 # create order of boarding for doing windows first using groups
437 def create_boarding_order_for_aisle_but_with_groups(boarding_section,
438     other_section1, other_section2, seats):
439
440     for seat in seats:
441         # window seats
442         for row in range(1,NUM_ROWS+1):
443
444             # check if item in group already appended
445             if (not any([row,seat] in x for x in boarding_section)
446                 and not any([row,seat] in x for x in other_section1)
447                 and not any([row,seat] in x for x in other_section2)):
448
449                 # if passenger is not useless
450                 if is_not_disobedient():
451
452                     for aisle in AISLES:
453
454                         if aisle-seat == 3:
455
456                             current_group_size = group_size((70,50,20))
457
458                             if current_group_size == 3:
459                                 boarding_section.append([[row, seat],[row, seat
460 +1],[row, seat+2]])
461
462                             elif current_group_size == 2:
463                                 boarding_section.append([[row, seat],[row, seat
464 +1]])
465
466                             else:
467                                 boarding_section.append([[row, seat]])
468
469                             elif aisle-seat==3:
470                                 current_group_size = group_size((70,50,20))
471
472                                 if current_group_size == 3:
473                                     boarding_section.append([[row, seat],[row, seat
474 -1],[row, seat-2]])
475
476                                 elif current_group_size == 2:
477                                     boarding_section.append([[row, seat],[row, seat
478 -1]])
479
480                             else:
481                                 boarding_section.append([[row, seat]])
482
483                             elif aisle-seat==2:
484                                 current_group_size = group_size((80,40,0))
485
486                                 if current_group_size == 2:
487                                     boarding_section.append([[row, seat],[row, seat
488 +1]])
489
490                             else:
491                                 boarding_section.append([[row, seat]])
492
493                             elif aisle-seat==2:
494                                 current_group_size = group_size((80,40,0))
495
496                                 if current_group_size == 2:
497                                     boarding_section.append([[row, seat],[row, seat
498 +1]])
```

```
-1]])
485
486         else:
487             boarding_section.append([[row, seat]])
488
489     else: boarding_section.append([[row, seat]])
490
491         break
492
493     # else they try board during different sections
494     else:
495         if random.randrange(100) < 50:
496             other_section1.append([[row, seat]])
497         else:
498             other_section2.append([[row, seat]])
499
500
501 # reduce boarding queue capacity due to Covid
502 def cull_boarding_queue(boarding_queue):
503     #this function has two aims: reduce capacity due to COVID, and remove any
504     seats not included in planes
505
506     # first see if need to cull the seats that would be in grid of planes, but
507     # not there
508     # remove them here as easier than having to not add them in the first place
509     # in every method
510     global plane
511     if plane == 'wide wing':
512         for index,passenger in enumerate(boarding_queue):
513             # Seats A B C V W X in rows 1-3
514             if passenger[0]-1 in [0,1,2] and passenger[1] in [0,1,2,25,26,27]:
515                 del boarding_queue[index]
516     elif plane == 'narrow body':
517         for index,passenger in enumerate(boarding_queue):
518             # Row 1 seats D E F
519             if passenger[0]-1 in [0] and passenger[1] in [4,5,6]:
520                 del boarding_queue[index]
521
522     if COVID_CAPACITY==0:
523         return boarding_queue
524     target_to_kill = math.floor((COVID_CAPACITY)*NUM_SEATS)
525     for row in range(NUM_ROWS):
526
527         killed = 0
528         for index,passenger in enumerate(boarding_queue):
529             if passenger[0] == row:
530                 killed += 1
531                 del boarding_queue[index]
532             if killed==target_to_kill:
533                 break
534
535     return boarding_queue
536
537
538 # -----
539 # BOARDING METHODS
540 # -----
541
542
543 # boarding in random order
544 def random_boarding():
545
```

```
541     test_cases = []
542     for _ in range(N_TEST_CASES):
543         boardingQueue = []
544         for row in range(1,NUM_ROWS+1):
545             for seat in range(NUM_SEATS+len(AISLES)):
546
547                 # assign bag based on probability that passenger has bag
548                 if seat not in AISLES: boardingQueue.append([row,seat,
549 assign_luggage(),0])
550
551             random.shuffle(boardingQueue)
552
553             test_cases.append(board_the_plane(boardingQueue, AISLES))
554
555
556
557
558
559 def random_boarding_with_groups():
560
561     test_cases = []
562     for _ in range(N_TEST_CASES):
563         boardingQueue = [[]]
564
565         current_group_member=0
566         current_group_size = group_size((SINGLE_PRINGLE_COEFFICIENT,
567 COUPLES_COEFFIENCT,THREESOME_COEFFICIENT))
568
569         for row in range(1,NUM_ROWS+1):
570
571             for seat in range(0,NUM_SEATS+len(AISLES)):
572
573                 if seat not in AISLES: boardingQueue[-1].append([row,seat,
574 assign_luggage(),0])
575
576                 current_group_member += 1
577
578                 if current_group_member == current_group_size:
579
580                     for aisle in AISLES:
581
582                         if seat-aisle in [2,3]:
583                             boardingQueue[-1].reverse()
584
585                         if seat-aisle in [-3,-2,1]:
586                             current_group_size = group_size((
587 SINGLE_PRINGLE_COEFFICIENT,COUPLES_COEFFIENCT,THREESOME_COEFFICIENT))
588                         elif seat-aisle in [-1,2]:
589                             current_group_size = group_size((
590 SINGLE_PRINGLE_COEFFICIENT,COUPLES_COEFFIENCT,0))
591                         elif seat == 6:
592                             current_group_size = 1
593
594                         current_group_member = 0
595                         boardingQueue.append([])
596
597                         break
```

```
596
597
598     random.shuffle(boardingQueue)
599
600     # flatten groups
601
602     boardingQueue = [j for sub in boardingQueue for j in sub]
603
604     #print(boardingQueue)
605
606     boardingQueue = cull_boarding_queue(boardingQueue)
607
608     test_cases.append(board_the_plane(boardingQueue))
609
610
611 print('Random with groups: ', sum(test_cases)/len(test_cases))
612 print(test_cases)
613 return average(test_cases)
614
615
616 # sectional boarding but with groups
617 def section_boarding_with_groups():
618
619     test_cases = []
620     amf, fma = [], []
621     for _ in range(N_TEST_CASES):
622
623         aft, middle, front = [], [], []
624
625         # aft section
626         create_boarding_order_for_section_but_with_groups(aft, middle, front,
627 A_SEC_START, A_SEC_END)
628         # middle section
629         create_boarding_order_for_section_but_with_groups(middle, aft, front,
630 M_SEC_START, M_SEC_END)
631         # front section
632         create_boarding_order_for_section_but_with_groups(front, middle, aft,
633 F_SEC_START, F_SEC_END)
634
635         random.shuffle(aft)
636         random.shuffle(middle)
637         random.shuffle(front)
638
639         #print(boardingQueue)
640         boardingQueue = aft+middle+front
641         boardingQueue = [j for sub in boardingQueue for j in sub]
642         boardingQueue = cull_boarding_queue(boardingQueue)
643         amf.append(board_the_plane(boardingQueue))
644         #boardingQueue = front+middle+aft
645         #boardingQueue = [j for sub in boardingQueue for j in sub]
646         #fma.append(board_the_plane(boardingQueue))
647
648 print('Sectional amf: ', average(amf))
649 #print('Sectional fma: ', average(fma))
650
651 return(average(amf))
```

```
653
654
655
656 # boarding by seat but allowing groups to board together
657 def seat_boarding_with_groups():
658
659     test_cases = []
660     boardingQueue=[]
661     for _ in range(N_TEST_CASES):
662
663         window,middle,aisle = [],[],[]
664
665         # window seats
666         #window_seats = [aisle-3 for aisle in AISLES] + [aisle+3 for aisle in
AISLES]
667         #create_boarding_order_for_aisle_but_with_groups(window,middle,aisle,
window_seats)
668         # middle seats
669         middle_seats = [aisle-2 for aisle in AISLES] + [aisle+2 for aisle in
AISLES]
670         create_boarding_order_for_aisle_but_with_groups(middle,window,aisle,
middle_seats)
671         # aisle seats
672         aisle_seats = [aisle-1 for aisle in AISLES] + [aisle+1 for aisle in
AISLES]
673         create_boarding_order_for_aisle_but_with_groups(aisle,window,middle,
aisle_seats)
674
675         random.shuffle(window)
676         random.shuffle(middle)
677         random.shuffle(aisle)
678
679
680         window = [j for sub in window for j in sub]
681         middle = [j for sub in middle for j in sub]
682         aisle = [j for sub in aisle for j in sub]
683
684         boardingQueue1 = window+middle+aisle
685         for x in boardingQueue1:
686             if x not in boardingQueue:
687                 boardingQueue.append(x)
688
689         for passenger in boardingQueue:
690             passenger.append(assign_luggage())
691             passenger.append(0)
692
693         boardingQueue = cull_boarding_queue(boardingQueue)
694         test_cases.append(board_the_plane(boardingQueue))
695
696         print('By seat with groups: ', sum(test_cases)/len(test_cases))
697
698     return average(test_cases)
699
700 def prioritize_groups_boarding():
701
702     test_cases = []
703     for _ in range(N_TEST_CASES):
704         mainBoardingQueue = [[]]
705         priorityQueue=[]
706         boardingQueue=[]
```

```
707     current_group_member=0
708     current_group_size = group_size((SINGLE_PRINGLE_COEFFICIENT ,
709 COUPLES_COEFFICIENT , THREESOME_COEFFICIENT ))
710     current_boarding_section = 2
711     for row in range(1,NUM_ROWS+1):
712
713         for seat in range(0,NUM_SEATS+len(AISLES)):
714
715             if seat not in AISLES:
716
717                 if current_boarding_section == 1:
718                     priorityQueue[-1].append([row,seat,assign_luggage(),0])
719                 else:
720                     mainBoardingQueue[-1].append([row,seat,assign_luggage()
721 ,0])
722
723             current_group_member += 1
724
725             if current_group_member == current_group_size:
726
727                 for aisle in AISLES:
728
729                     if seat-aisle in [2,3]:
730                         if current_boarding_section == 1:
731                             priorityQueue[-1].reverse()
732                         else:
733                             mainBoardingQueue[-1].reverse()
734
735                     if seat-aisle in [-3,-2,1]:
736                         current_group_size = group_size((
737 SINGLE_PRINGLE_COEFFICIENT ,COUPLES_COEFFICIENT ,THREESOME_COEFFICIENT ))
738                         elif seat-aisle in [-1,2]:
739                             current_group_size = group_size((
740 SINGLE_PRINGLE_COEFFICIENT ,COUPLES_COEFFICIENT ,0))
741                             elif seat-aisle == 3:
742                                 current_group_size = 1
743
744                         break
745
746             current_group_member = 0
747
748             if current_group_size == 3:
749                 if random.randrange(100) > 80:
750                     mainBoardingQueue.append([])
751                     current_boarding_section = 2
752                 else:
753                     priorityQueue.append([])
754                     current_boarding_section = 1
755             elif current_group_size == 2:
756                 if random.randrange(100) > 20:
757                     mainBoardingQueue.append([])
758                     current_boarding_section = 2
759                 else:
760                     priorityQueue.append([])
761                     current_boarding_section = 1
762             elif current_group_size == 1:
763                 if random.randrange(100) > 5:
```

```
763                         mainBoardingQueue.append([])
764                         current_boarding_section = 2
765                 else:
766                     priorityQueue.append([])
767                     current_boarding_section = 1
768
769
770
771         random.shuffle(mainBoardingQueue)
772         random.shuffle(priorityQueue)
773         # flatten groups
774         boardingQueue = priorityQueue+[‘b’]+mainBoardingQueue
775         boardingQueue = [j for sub in boardingQueue for j in sub]
776
777         #print(boardingQueue)
778
779         test_cases.append(board_the_plane(boardingQueue, True))
780
781     print(‘Prioritizing groups: ’, average(test_cases))
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802 # modified steffen method
803 def steffen_modified():
804
805     test_cases = []
806     for _ in range(N_TEST_CASES):
807
808
809         rightOdd, leftOdd, rightEven, leftEven = [],[],[],[]
810         steffenPerfected = [rightOdd, leftOdd, rightEven, leftEven]
811         # window seats
812         for row in range(1,NUM_ROWS+1):
813             for seat in range(-3,4):
814                 #naughty boy
815                 if not is_not_disobedient() and seat != 0:
816                     steffenPerfected[random.randrange(0,2)].append([row,seat+3,
817 assign_luggage(),0])
818
819                     elif (seat > 0): #right side
820                         steffenPerfected[(row%2)*2].append([row,seat+3,assign_luggage()
821 () ,0])
822                     elif (seat < 0): #left side
```

```
821     steffenPerfected[(row%2)*2+1].append([row, seat+3,  
822         assign_luggage(), 0])  
823  
824  
825     random.shuffle(steffenPerfected[0])  
826     random.shuffle(steffenPerfected[1])  
827     random.shuffle(steffenPerfected[2])  
828     random.shuffle(steffenPerfected[3])  
829  
830  
831  
832     steffenPerfected = [j for sub in steffenPerfected for j in sub]  
833     test_cases.append(board_the_plane(steffenPerfected))  
834  
835     print('By steffen perefected: ', sum(test_cases)/len(test_cases))  
836  
837     return(average(test_cases))  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847 plane = 'narrow body'  
848  
849 if plane == 'narrow body':  
850     NUM_ROWS = 33  
851     NUM_SEATS = 6  
852     AISLES = [3]  
853     F_SEC_START = 1  
854     F_SEC_END = 11  
855     M_SEC_START = 12  
856     M_SEC_END = 22  
857     A_SEC_START = 23  
858     A_SEC_END = 33  
859 elif plane == 'wide wing':  
860     NUM_ROWS = 14  
861     NUM_SEATS = 24  
862     AISLES = [3, 10, 17, 24]  
863     F_SEC_START = 1  
864     F_SEC_END = 5  
865     M_SEC_START = 6  
866     M_SEC_END = 9  
867     A_SEC_START = 10  
868     A_SEC_END = 14  
869 elif plane == 'two entrance two aisle':  
870     # simulating only the back half of the plane  
871     NUM_ROWS = 20  
872     NUM_SEATS = 7  
873     AISLES = [2, 6]  
874     F_SEC_START = 1  
875     F_SEC_END = 7  
876     M_SEC_START = 8  
877     M_SEC_END = 14  
878     A_SEC_START = 15  
879     A_SEC_END = 20
```

```
880 elif plane == 'two entrance two aisle first class':
881     # total loading time for 2E2A plane will be 2E2A + 2E2A first class
882     NUM_ROWS = 3
883     NUM_SEATS = 6
884     AISLES = [2,5]
885
886 # CHANGE THESE FOR SENSITIVITY
887 BAG_COEFFICIENT = (20,80,10)
888 NAUGHTY_BOY_COEFFICIENT = 0.18
889 COVID_CAPACITY = 0.5 #0, 0.3 0.5 or 0.7
890
891 N_TEST_CASES = 100
892 VISUALIZER = True
893 TIME_STEP = 0.1
894
895 # all measured in standard units (m,s,m/s etc)
896 AVERAGE_WALKING_SPEED = 0.8
897 AVERAGE_SEAT_PITCH = 0.78
898 TIME_TO_MOVE = AVERAGE_SEAT_PITCH / AVERAGE_WALKING_SPEED
899 FAMILY_TIME_TO_MOVE = 1.3 * TIME_TO_MOVE
900 NON_FAMILY_TIME_TO_MOVE = TIME_TO_MOVE
901 TIME_TO_SIT_OR_STAND = 2.5
902 TIME_TO_MOVE_PAST_SEAT = 2
903 # proportions of group sizes
904 SINGLE_PRINGLE_COEFFICIENT = 70
905 COUPLES_COEFFICIENT = 20
906 THREESOME_COEFFICIENT = 0
907
908 if VISUALIZER: im,fig = initialize_render()
909
910 #Data csv
911 import csv
912 fields = []
913 rows = []
914 index = []
915 #Add the indexing
916 for i in range(N_TEST_CASES):
917     index.append(i)
918
919 rows.append(index)
920
921
922
923 # BOARDING METHODS: comment out if not using
924 #random_boarding()
925 #section_boarding()
926 #seat_boarding()
927 #random_boarding_with_groups()
928 #section_boarding_with_groups()
929 #seat_boarding_with_groups()
930 #prioritize_groups_boarding()
931
932 # steffen methods can only be used with narrow body
933 #steffen_deeznuts()
934 #steffen_modified_method()
935
936 #naughty_boy_sensitivity()
937 #bag_sensitivity()
938
939 # field names add whatever field names that you are creating data for
```

```
940 fields = [ 'Index' , 'Section' ]
```