Summary

Nowadays, technologies are evolving. People connect with each other from all over the world. Therefore, travelling by airplane is becoming more and more popular. People are now familiar with the terms "boarding" and "disembarking". Because the demand for travelling by airplane is rapidly increasing, the turnaround time needs to be reduced as much as possible. And to do so, they have to find out how long does it takes to board and disembark the passengers. Today, we are bringing you the solution to this problem: a model for calculating boarding and disembarking time.

For boarding, the average amount of time needed in order to load all passengers on board is about twenty minutes. Going on an airplane can be troublesome. Passengers has to wait in the queue to their seat, and when they get there, not only they have to pack their belongings, but also they need to adjust the other passengers who are sitting in the same row, in order to advance to their correct seat. A lot of factors are involved in this process, so instead of calculating them all, our model uses a virtual timer, which checks the status of each passenger after each second, until all the passengers' status are "seated", assuming that all passengers have the same priority. Our model gives out very close results, only deviate from the average time about 1 minute.

After the plane lands, the disembarking process is also really tricky. After a tiring flight, passengers would like to exit the airplane as soon as possible, but in this case, the aisles for airplanes has the width of one person, so that people can't barge in unexpectedly in the aisle. Again we assumed all passengers have the same priorities. For this model, not only we use a virtual timer, we also use a scaling system that divided equally the chances of getting into the aisle of the nearby passengers, given that passengers have the same priorities. Because of this assumption, we have an easy and quick model for disembarking time, and the result is also very close from the average time, which means our model is very helpful for people to understand more about those process.

There are a lot of airplane types, which means different layout for each airplane and also, different boarding and disembarking time. But with the correct modification, our models can deal with lots of airplane types, another benefit of our model. By comparing our results with the average time, our model can be modified in order to get the closest result, hence increasing the correctness time to time.

Contents

1 Introduction

The invention of airplanes is a significant milestone in human history, as it revolutionised transport in general, enabling people to travel great distances in a much shorter amount of time (e.g. a 9-hour car trip can be replaced by a 1-hour plane trip). However, planes only generate money and carry passengers when they are in the air. This means that the more time a plane spends on land, the less efficient and lucrative it becomes. This idling period usually include smaller processes like refueling, loading and offloading baggage, etc. Reducing this period, even for a few minutes, will greatly increase the efficiency and profitability of air travel. Airlines can achieve this by shortening the time required to complete **some** of such processes. The most feasible process whose completion time can be shortened is arguably the boarding/disembarking of passengers, since it accounts for a sizeable chunk of the average turnaround time and unlike procedures concerning technical problems like fuel, it can be optimized without endangering the safety of the airplane itself and its passengers.

To solve this problem, we have developed a mathematical model that computes the approximate amount of time it takes for planes to load and unload all of their passengers, taking various factors into account such as the amount of luggage each passenger carries and their seats.

2 Requirement 1

2.1 General Assumptions and Justifications

Assumption	**Justification**
The aisle is only wide enough for one person to pass at a time	This is in the problem statement. We will not consider the case when passengers try to sneak through the hindrance
Each passenger travels as an individual	For an easier model
There are no types of priority queue (business class, special needs,...) in place	For an easier model and not stated in the problem
All passengers can only carry up to two suitcases	In real life, most airlines will allow you to have one carry-on bag. We will allow up to two bags only
All passengers walk at a constant speed	For an easier model and since passengers walk at similar speed on airplanes
All passengers in the aisle start moving as soon as the person in front of them starts moving	This is true in real life

All suitcases are considered the same	Although there are many different types of suitcases, we will consider all suitcases the same for an easier model
All passengers are expected to sit in the same seat they are assigned to	We will not consider the case where some passengers do not sit in their assigned spots
The airplane's structure is like the airplane structure in Figure 1	For a simpler model and adaptation

2.2 Definitions of terms and variables used in the model

2.2.1 Definitions of terms

Term	**Definition**
'stowing bags' phase	The phase when the passenger is stowing the bags in to the overhead cabin
'getting into seats' phase	The phase when the passenger is trying to get into his/her seat if there is hindrance

2.2.2 Definitions of variables

Term	**Definition**
P[i]	The position of the i-th passenger in the aisle
Q	The array for the queue of passengers boarding the airplane
R[i]	The row of the i-th passenger
$b[i]$	The number of bags of the i-th passenger in Q
l_p	The length of the passenger in the aisle
v_{pw}	The velocity of passengers in the airplane (cm/s)
t_{mbs}	The time to move between seats in a row (seconds)
t_{sb}	The time in 'stowing bags' phase (seconds)
t_{gts}	The time in 'getting into seats' phase (seconds)
l_{btr}	The length between two successive rows (cm)

2.3 Model Development For Boarding

2.3.1 Concept

In terms of finding how long the boarding process is, measuring how much time each passenger takes to get to their seat is *more difficult than it looks*. This is due to the many factors that need accounting for when calculating, including how many times our passenger will be stopped, how long each stop actually is, etc. Constituting all of these factors will be *nearly impossible without high-level mathematics* due to the sheer amount of cases that has to be taken into

consideration.

Therefore, we have opted for a simpler method, which is writing a program that *takes account of change* after a certain amount of time has passed. In other words, from the moment the first passenger starts moving, our model will start a virtual timer which will *stop at certain intervals* to *check and update the 'properties'* of each passenger. After each interval of *1 second* (there will be explanation for why we chose this number later in this paper), we will stop the timer and check every passenger *in the aisle's* 'properties'. The moment the last passenger in the aisle manages to sit down, the timer will stop, and whatever the number the timer is displaying is the output - how much time it took for all passengers to board the plane. A key thing that makes our entire model work is the importance of *the properties of the passenger in front of one* when it comes to examining any passenger's state. To explain this better, the passenger will move or wait according to the next passenger's position and state. If the next passenger is *in the 'stowing bags' or 'getting into seats' phase*, the passenger behind will have to wait and cannot move. On the other hand, if the next passenger is *moving to his/her row*, the behind passenger can move as well. This simplifies the problem a lot because we do not need to consider the whole queue when examining each passenger's state, only the person directly in front of him, thus speeding up the calculation process.

Row 1:	01	02	03	- -	04	05	06
Row 2:	07	08	09	- -	10	11	12
Row 3:	13	14	15	- -	16	17	18
Row 4:	19	20	21	- -	22	23	24
Row 5:	25	26	27	- -	28	29	30

...

The seat's numbers of passengers

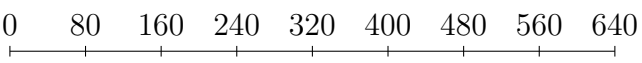
2.3.2 The creation of the queue

- We will denote each passenger to their seat's number by counting each row starting from row 1, left to right. After that, we will set Q as an array - the queue of passengers boarding the airplane (E.g: $Q = [11, 3, 6, 4, 8, 1, 5, 9, 2, 12, 10, 7]$, the i-th element means that the person with seat number $Q[i]$ will be the i-th person to board the plane). With this array, we can adapt different prescribed methods and stimulate passengers who do not follow the instructions by changing the order of array.
- Then we can find the row of passenger i $R[i]$ based on the $Q[i]$:

$R[i] = floor(Q[i] - 1)/6 + 1)$ ($floor(x)$ is largest integer not greater than x)

2.3.3 The position of passengers in the aisle

- We will consider the aisle as a x-axis:



- The entrance is at 0 (cm); row j will be at $j \cdot 80$ (cm)

- The length of passenger i can be calculated as $l_p[i] = 20 + 40 \cdot b[i]$(cm) since the length of human standing is 20cm and the length of each suitcase added is approximately 40cm.

aisle →	bag	$P[i]$	→

Example of a passenger with his bag in the aisle

- From this we can calculate each passenger position P[i] by their middle length.
+ For example: If a passenger has a length of 60cm and takes the position from 20-80, his/her position will be (80+20)/2 = 50(cm)
- And we can then calculate each passenger's position in the queue from the beginning. Let's denote $P[1] = 0$ in the beginning, which means that the first person in the queue will have the position 0cm in the beginning. And $P[2] = -(l_p[1] + l_p[2])/2$, because $P[i]$ is the middle location of l_{p_1}. Continuing from that, we can see that $P[i+1] = P[i] - (l_p[1] + l_p[2])/2$, and from that we can calculate all $P[i]$ for every passenger.

2.3.4 Values of parameters

Because we could not get the data from airlines or have access to real airplanes, we had to stimulate the airplane environment in our own house with seats, rows and aisle. After some experiments that we have conducted ourselves, we have found the value of these parameters:

- The average speed a passenger moves in the aisle during boarding is $v_{pw} = 40$ cm/s
- The time for a human to move between 2 consecutive seats in a row is $t_{mbs} = 2$s
- The length between 2 successive rows is $l_{btr} = 80$ cm
- The average time to stow a bag into an overhead cabin is 10s

2.3.5 The 'stowing bags' phase

The time that passenger i needs to stow the bags is $10 \cdot b[i]$

2.3.6 The 'getting into seat' phase

Here, only *the time that affects the aisle itself is accounted for*, *NOT* the time it takes for the passenger in question to get into his seat.
Let's denote that [x] means the seat is occupied and [o] means that the seat is not occupied.
Consider 3 cases:
Case 1: Our passenger has to move to the aisle seat
+ Whether the other two seats are occupied does not matter here, the aisle will only be affected for t_{mbs} seconds.
- **Case 2**: Our passenger has to move to the middle seat
We split this case into 4 mini-cases:
+ Minicase 1: [o|o|o]- : the aisle will be affected for t_{mbs} seconds.
+ Minicase 2: [x|o|o]- : the aisle will be affected for t_{mbs} seconds.

+ Minicase 3: |o|o|x| : the aisle will be affected for $4 \cdot t_{mbs}$ seconds.
+ Minicase 4: |x|o|x| : the aisle will be affected for $4 \cdot t_{mbs}$
- **Case 3:** Our passenger has to move to the window seat
+ Minicase 1: |o|o|o| : the aisle will be affected for t_{mbs} seconds.
+ Minicase 2: |o|o|x| : the aisle will be affected for $4 \cdot t_{mbs}$ seconds.
+ Minicase 3: |o|x|o| : the aisle will be affected for $5 \cdot t_{mbs}$ seconds.
+ Minicase 4: |o|x|x| : the aisle will be affected for $6 \cdot t_{mbs}$ seconds.

2.3.7 Process in boarding

- After each second, each passenger's position will increase by $v_{pw} = 40$cm/s. If he/she reach their row or $P[i] = R[i]$, they will not move anymore and start standing still in the 'stowing bags' phase then the 'getting into seats' phase. If he/she finish the 'getting into seats' phase, we will remove them from the queue list. The timer will end when there is no one left in the queue.
- And they can only move if their next passenger's distance is enough to move. In other words, the P[i] will increase if $P[i-1] - P[i] > (l_p[1] + l_p[2])/2$. And the distance they move will be equal to $\min(a, 80, R[i] \cdot 80 - P[i])$ with $a = P[i-1] - P[i] - (l_p[1] + l_p[2])/2$

2.4 Model Development for Disembarking

2.4.1 Concept

Along with boarding time, disembarking time is also an important problem that has to be taken into consideration. However, there is not as much randomness involved in the calculation of the latter. In other words, unlike the boarding process where there is a seemingly infinite number of possible seat combinations at the beginning, we assume that only the passengers that are currently sitting in the aisle seat are able to get in the queue. The moment the debarking process starts, one ofof the two aisle-seated passengers in each row will get in the queue and start collecting their luggage in overhead bins. Just like the boarding process, we concluded that using the aforementioned 'timer' method for our program will work better for the same reasons. As the virtual timer stops at certain intervals, this program we wrote will check each passenger's properties. The moment the last passenger exits the airplane, the timer stops and whatever time it shows is the output we need - the time for all passengers to disembark the airplane.

2.4.2 Additional Assumptions

Assumption	Justification
1. Each passenger travels as an individual	To simplify the problem by considering everyone as an individual
2. Only the aisle-seated passengers are allowed to get in the queue	To simplify the problem and avoid niche situations
3. The moment the plane allows passengers to disembark, each row will have one passenger from one of the aisle seats	To simplify the problem by considering everyone as an individual
4. A passenger in the aisle is considered *out of the queue* the moment he starts moving *only if* he is standing in the first row	When this happens, the only option for our passenger is to exit the plane, therefore he will be omitted from the queue
5. Passengers *always* move to the seat next to them that is closer to the aisle when that seat is empty	Humans are impatient, so the moment the aisle-seated passenger starts to get in the queue, the middle-seated one will replace him at the aisle seat, and same goes with the window-seated passenger
6. A passenger in the aisle seat can interfere in the line only when the movement of a random passenger creates sufficient space	To take into account the possibility that a sitting passenger may interrupt the line

2.4.3 Value of parameters

The dimensions of the airplane environment will stay the same the boarding method, as well as the speed of each passenger. Here are some additional constants that are unique to the disembarking process that we have found using hands-on experiments that we carried out ourselves:

- The time it takes for each passenger to collect their luggage on overhead bins is $10 \cdot b[i]$ seconds.
- The time it takes for each passenger to advance one row in the queue is still 2 seconds.

2.4.4 How it works

At the beginning of the disembarking process, *one of the two* aisle-seated passenger in each row will get in the queue and start taking out luggage in the overhead bins. From this moment, the timer starts ticking and stops at the first interval when the passenger in the very front of the queue finishes collecting his luggage and starts moving. During this interval, the program detects whether the second person can move: if he can, then the program continues to do the

same with the passenger behind, and so on. If the program manages to find a passenger that is still collecting his stuff, it declares the space between him and the person in front sufficient for either of the two aisle-seated passengers to get in the queue (note that due to assumption #5, the aisle seat will *never* be empty unless the 3-seat block is empty). If it does not manage to, however, it simply waits for each passenger in the queue to advance one row.

From here, the program continually stops at certain intervals to check each passenger's properties from the nearest to the furthest from the exit, primarily whether he is moving, standing still or collecting baggage, after which it uses the collected data to determine the next passenger's properties in line. This loops until it has checked every passenger in line. When a passenger manages to debark the plane, his corresponding variable is omitted from the original input array. The process goes on until the input array is empty, meaning that the unloading procedure is completed, and the timer will stop, with the time recorded on the program's timer being the time it took for this procedure to finish - the output.

3 Requirement 2

3.1 Task a

With our model, to perform the time for random boarding, we'll randomize the queue of passengers getting in the airplane. We will assume that each passenger will have a random number of carry-on bags from 0 to 2. We try this for 10000 cases and get the following results:

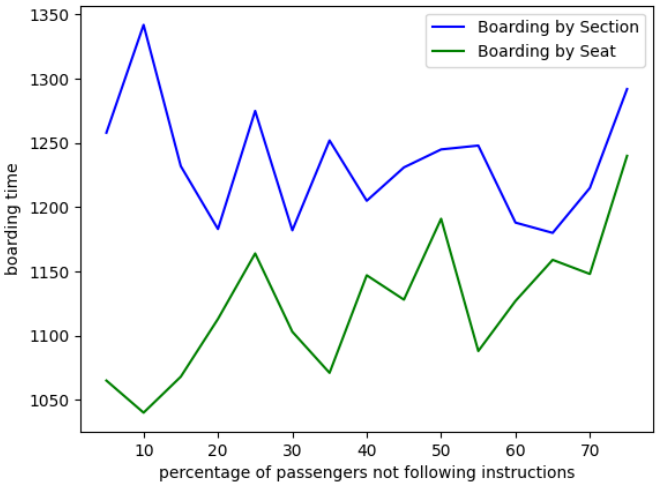
	Average	**95th percentile**	**5th percentile**
Random Boarding	1277s	1466s	1111s
Boarding by Section	1463s	1731s	1231s
Boarding by Seat	1152s	1308s	1013s

This is quite accurate, since there have been many studies showing that *'boarding by section'* is not efficient at all. In fact, from our model, it shows that the average boarding time using the *'boarding by section'* is much slower than random boarding, by 219 seconds. Moreover, the '*boarding by seat'* shows the most efficient time among all three methods. This is in good agreements with many studies conducted about boarding time before.

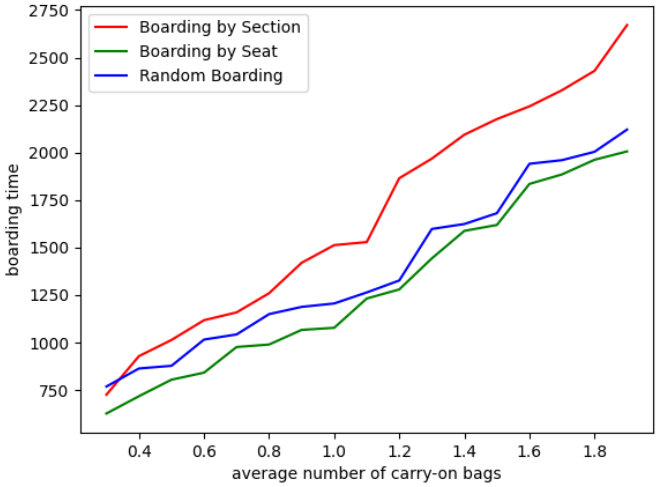
3.2 Task b

Analysis of the time based on the percentage of passengers not following the prescribed methods:

- Random boarding will not be considered in this case since this method is still completely random.



Analysis of the time based on the average number of carry-on bags per flight:



Based on the two graphs above, we can obviously see that the green line is lowest, meaning that the boarding by seat is the most efficient method among all three. This is quite an accurate result according to many other studies.

3.3 Task c

From task b, we can obviously see that if the number of bags increases, the boarding time also increases. And because of the limitation of space for carry-on bags on airplanes, we will only consider the average number of carry-on bags from the range 0,3 - 1,9. In conclusion, the result will increase if the number of carry-on bags increase, and it will increase on a linear model, according to the plot above.

3.4 Task d

After some research online, and with our own stimulations and analysis of results, we have found two realistic and optimal methods that might be used in real life one day with further

research.

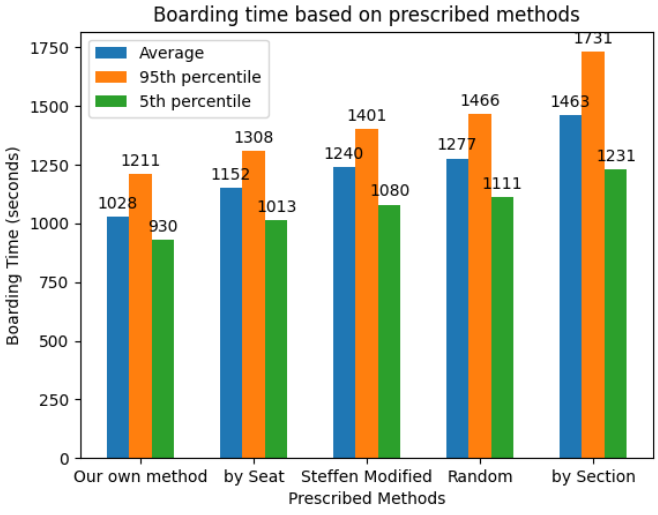
- **The Steffen Modified Method:**
  This method is quite popular and it is considered the most potential boarding system to use in real life. The queue will be divided into 4 groups, and can be described as follows:

rows	1	2	3	···	16	17	···	31	32	33
column 6	2	4	2	···	4	2	···	2	4	2
column 5	2	4	2	···	4	2	···	2	4	2
column 4	2	4	2	···	4	2	···	2	4	2
aisle				···			···			
column 3	1	3	1	···	3	1	···	1	3	1
column 2	1	3	1	···	3	1	···	1	3	1
column 1	1	3	1	···	3	1	···	1	3	1

- **Our own method:**
  This method is the combination of the 'boarding by section' and 'boarding by seat' method. The queue will be divided into 12 groups, and can be described as follows:

rows	1	2	3	···	16	17	···	31	32	33
column 6	2	2	2	···	2	7	···	7	7	7
column 5	4	4	4	···	4	9	···	9	9	9
column 4	6	6	6	···	6	11	···	11	11	11
aisle				···			···			
column 3	12	12	12	···	12	5	···	5	5	5
column 2	10	10	10	···	10	3	···	3	3	3
column 1	8	8	8	···	8	1	···	1	1	1

From the methods above, we applied the methods into our model and get the following results about boarding time in different methods.



From the bar chart above, we can see that the method that we created is the most efficient when applied in our model. The 'Steffen Modified' method on the other hand, is quite good but not as good as the 'Boarding by Seat' method. We haven't figured out why the 'Steffen

Modified' is not as efficient as we expected. Maybe there're factors that we haven't considered yet. But overall, we can see that each method has its own strengths and weaknesses and it will take more time to find the most practical boarding method.

3.5 Task e

From the disembarking model, we can obviously see that if the passenger queue wasn't cut off in any parts then the amount of disembarking time will be minimum. Passengers will move continuously, and there would be no free spaces between passengers in the aisle. Therefore, the minimum time to disembark the passengers is 2n, as n denote the number of passengers on the airplane at first.

4 Requirement 3

4.1 Model Modification

4.1.1 "Flying Wing" passenger aircraft

1. Definitions

Term	Definition
aisle	In this subsection only, we will define the term 'aisle' as the path that is only *between the seat columns*
walkway	Part of the walkable path that is horizontal and lies north of the seats, passing through each aisle
safety line	This line forces passengers to go in the aisles

2. Boarding

a. Concept

Regarding the 'Flying Wing' airplane model, it still only has one entrance like the 'Narrow Body' model. Therefore, we can utilize the coordinate method described in 2.3c, setting the entrance at point (0,0). However, there are more than one seat aisle in the airplane plus a horizontal walkway just North of the seats, so our original model needs some additional tweaks. Therefore, we have set some assumptions:

- Passengers will choose the aisle *closest* to their respective seats. This is possible since there are no 'middle seats' that are equidistant to any two aisles.
- Same assumptions as with the 'Narrow-Body'.

This airplane model is similar to the **narrow-body airplane** that each passenger knows exactly which aisle they should be getting in. Therefore we can *assign each passenger to the*

aisle they are going to take. Moreover, passengers in different aisles do *not* affect one another, so we can safely assess the properties of passengers in each discrete aisle, determine the time each aisle takes to fill its respective passengers. Since activity in each aisle happens simultaneously, there will be four copies of the original model to assess it. Other than that, the program should stay relatively unchanged, as we can consider each aisle and their assigned seats a mini version of the **"Narrowed-Body"**.

b. Method

As for the fastest time to board in each aisle, regarding all of the passengers who are waiting in the queue, the closest person to the entrance is called a_1, similarly, the n-th closest person to the entrance is called a_n (if the queue has n people).
All of the passengers in the queue will be divided into many groups which consist of 4 people: (a_1, a_2, a_3, a_4),(a_5, a_6, a_7, a_8),...,$(a_{4k-3}, a_{4k-2}, a_{4k-1}, a_{4k})$,...
Boarding time will be minimized as possible as it can be in case these two factors happen in the same time:
Passenger a_m (with m=4t+j: j can be 1,2,3 or 4) will be the passenger who is going to be at the j-th aisle. (1)
Another factor is that if the row of the a_k person is called r_{ak}, in order to minimize the boarding time, a thing that has to happen is that $r_{a1} > r_{a5} > r_{a9}$¿..., the similar thing with the other r_{ak} has to occur. (2)
Explanation for this method:
For (1), this requirement for all the passengers is to let 4 consecutive passengers in the queue can stow their package and access their seats at the same time. This happens because they do not have to wait since they are in 4 distinct aisles.
For (2), this requirement is to make the passengers in the same aisle not be delayed from waiting for anyone because in case the in front person has to stow their package or try to access their seat, those actions will happen in the row which is further from the entrance than the row of the passenger who is walking behind the delayed passenger.

3. Disembarking

a. Concept

Contrary to the boarding process, assessing how long it takes for a 'Flying Wing' type airplane to debark its passengers is much more difficult than that of a 'Narrow Body' type airplane. This is due to the fact that the small triangle area near the exit makes it possible for there to be multiple ways to deplane instead of a single line like the 'Narrow Body'. Therefore, again, some assumptions must be made to simplify this problem.

b. Assumptions

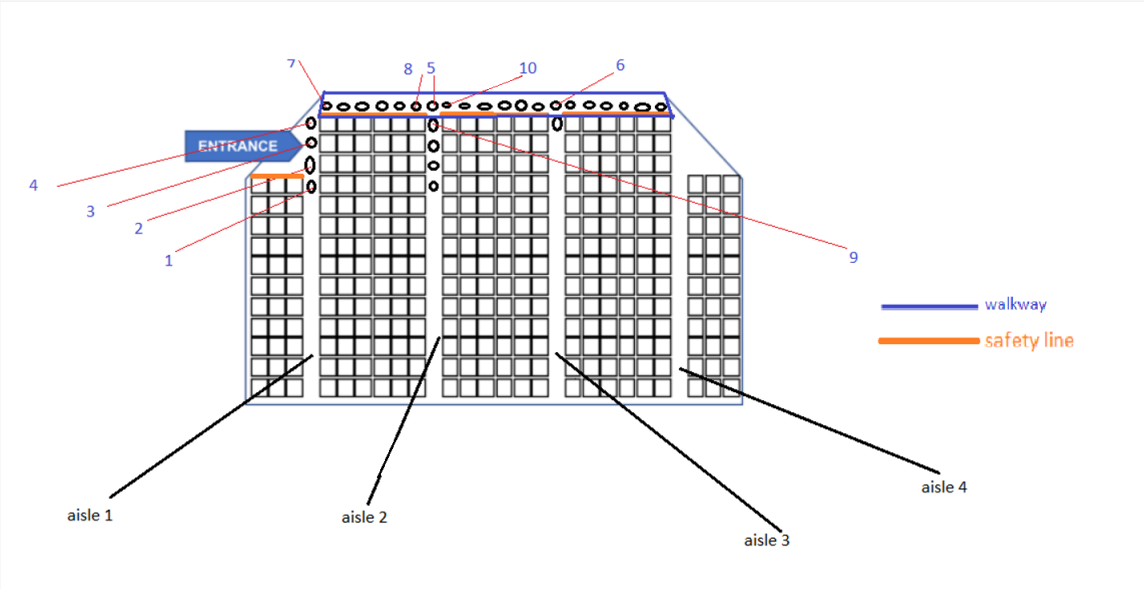


Figure. 1: 'Flying Wing' layout with added notations

Assumption	**Justification**
1. Same assumptions as the **'Narrow-Body'**	This keeps the value of certain parameter consistent
2. Passengers always choose the nearest aisle to their seat	To simplify the problem and keep it practical
3. Passengers can only follow three lines, starting from position 2, 3 and 4, going straight towards the exit (let it be defined as 2-, 3- and 4-miniqueue) demonstrated in the **narrow-type airplane**	to simplify the problem
4. Passengers in the top row of each section are only allowed to move horizontally, demonstrated by the safety lines in **narrow-type airplane** 1	Enforces assumption #3

c. Explanation

Position 2, 3 and 4, as demonstrated in **narrow-type airplane** 1, are the three pivotal points where *all* passengers need to pass to get to the exit. However, the exit only allows one person to exit at a time. Therefore, there's a 33% for each row to have the front person pass the exit. Let's consider each 'mini-queue' in front of the exit (consider that each mini-queue has a maximum capacity of two people):

- If someone from the 2-miniqueue manages to debark, position 2 will open up, and the left-sided aisle-seated passenger will take his place. However, only the three leftmost-seated passengers in the 3rd row of this section are eligible. After all three has debarked, the person in the front of aisle 1 will get this spot.

- If someone from the 3-miniqueue manages to debark, position 3 will open up, but only three leftmost-seated passengers in the 2nd row of the 2nd section are eligible, so after all three has exited the plane, this queue will be removed.
- If someone from the 4-miniqueue manages to debark, position 4 will open up, and the left-sided aisle-seated passenger in the first row will take this spot. Similar to the 2-miniqueue, only three passengers in the 1st row are eligible, and once all those three has debarked, the person in the front of the walkway will take this spot. A special property of the walkway is that *everyone* there has already completed the process of collecting baggage, thus, how the walkway clears is that after the initial walkway has been created, *only* passengers from aisle 4 can move in walkway, and after aisle 4 has emptied, passengers from aisle 3 can now move in the line. The same goes for aisle 2.

These are the only real differences compared to the 'Narrow-Body'. The model we developed for it can process the actions happening in each aisle. The modified version needs to do the above as well as assess change in the three pivotal positions.

Optimal methods

Since determining which of the three miniqueues at each interval is chosen randomly, there's no real way we can speed up the activity near the entrance. However, we can shorten activity in the aisles, as each aisle and the seats of its assigned passengers can be considered a mini-version of the 'Narrow-Body' airplane (for aisle 1, in the 2nd section, only seats from the 4th row onwards are assigned, the first three rows can automatically enter the three pivotal points). We have examined various different methods, but ultimately decided that the optimal method for disembarking the 'Flying Wing' is to let each miniqueue empty itself before allowing another. In other words, whoever at the top of a miniqueue arrives at a pivotal point first, the program will only allow passengers in that miniqueue to deboard the airplane. After this miniqueue has fully disembarked, the next miniqueue will be chosen in the same way. This is done to minimise the potential congestion in each aisle and 'intersections' in the walkway. As for activity in each aisle itself, the program for disembarking the 'Narrow-Body' is sufficient.

4.1.2 2 Entrances - 2 Aisles (TETA)

As this airplane type has 2 entrances, we will assume that the first entrance is for the first section, and the second entrance is for the second section of the airplane.

a. Explanation

Boarding

For this airplane type, we will split the airplane into 4 parts:

- The first part is the VIP lounge, starting from row 1 to 3.
- The second part is the first section, starting from row 12 to 25.
- The third part is the emergency exit in the middle, contains only row 26.
- The last part is the second section, starting from row 27 to row 47.

For this airplane type, we are assuming that the VIP lounge is the only type of priority queue in place.

1. The VIP lounge

For this part, we have to reconsider the cases, since there are 2 aisles and the width of each column is 2 seats.

Considering 3 cases:

Case 1: Our passenger have to move to the window seat

- Minicase 1: |o|o|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 2: |o|x|- : the aisle will be affected for $4 \cdot t_{mbs}$ seconds.

Case 2: Our passenger have to move to the aisle seat of the left and right column

- Minicase 1: |o|o|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 2: |x|o|- : the aisle will be affected for t_{mbs} seconds.

Case 3: Our passenger have to move to the middle column

- Minicase 1: -|o|o|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 2: -|x|o|- : the aisle will be affected for t_{mbs} seconds when going in the right aisle, $4 \cdot t_{mbs}$ seconds when going in the left aisle.
- Minicase 3: -|o|x|- : the aisle will be affected for $4 \cdot t_{mbs}$ seconds when going in the right aisle, t_{mbs} seconds when going in the left aisle.

2. The first section

For the first and second section, we have to reconsider the cases again, since the width of the middle column is now 3 seats, while the other columns are still 2 seats. Also the middle column has one less row than the 2 other columns, but it won't affect our model.

Considering 5 cases:

Case 1: Our passenger have to move to the window seat

- Minicase 1: |o|o|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 2: |o|x|- : the aisle will be affected for $4 \cdot t_{mbs}$ seconds.

Case 2: Our passenger have to move to the aisle seat of the left and right column

- Minicase 1: |o|o|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 2: |x|o|- : the aisle will be affected for t_{mbs} seconds.

Case 3: Our passenger have to move to the left aisle seat of the middle column, when moving in the left column

- Minicase 1: -|o|o|o|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 2: -|o|x|o|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 3: -|o|x|x|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 4: -|o|o|x|- : the aisle will be affected for t_{mbs} seconds.

Case 4: Our passenger have to move to the left aisle seat of the middle column, when moving in the right column

- Minicase 1: -|o|o|o|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 2: -|o|x|o|- : the aisle will be affected for $5 \cdot t_{mbs}$ seconds.
- Minicase 3: -|o|x|x|- : the aisle will be affected for $6 \cdot t_{mbs}$ seconds.
- Minicase 4: -|o|o|x|- : the aisle will be affected for $4 \cdot t_{mbs}$ seconds.

Case 5: Our passenger have to move to the middle seat of the middle column.

- Minicase 1: -|o|o|o|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 2: -|x|o|o|- : the aisle will be affected for t_{mbs} seconds when going in the right aisle, $4 \cdot t_{mbs}$ seconds when going in the left aisle.
- Minicase 3: -|o|o|x|- : the aisle will be affected for $4 \cdot t_{mbs}$ seconds when going in the right aisle, t_{mbs} seconds when going in the left aisle.
- Minicase 4: -|x|o|x|- : the aisle will be affected for $4 \cdot t_{mbs}$ seconds.

Case 6: Our passenger have to move to the right aisle seat of the middle column, when moving in the right column (same as **Case 3**)
Case 7: Our passenger have to move to the right aisle seat of the middle column, when moving in the left column (same as **Case 4**)
3. The emergency exit seat
Note that the passenger who seated in this part can enter by both of the entrance. Because there are only 2 columns and 1 row here, so there are only 2 cases:
Case 1: Our passenger have to move to the window seat

- Minicase 1: |o|o|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 2: |o|x|- : the aisle will be affected for $4 \cdot t_{mbs}$ seconds.

Case 2: Our passenger have to move to the aisle seat

- Minicase 1: |o|o|- : the aisle will be affected for t_{mbs} seconds.
- Minicase 2: |x|o|- : the aisle will be affected for t_{mbs} seconds.

4. The second section (same as the first section)

b. Method

To speed up the boarding process, I think we should increase the fee for carry-on luggage, in order to reduce the number of carry-on luggage of each passenger. Then, the delay when passengers have to stow their luggage would be minimum. Another delay we have to reduce is the delay when people have to get out of their seat in order for one to enter his or her seat. This can be reduced by using the "outside-in" method, at which we set the priority for the window seated passenger to the highest. Then the passengers seated in the left and right column aisles. After that, we let the passengers who seated in the left aisle of the middle column to enter. The passengers who seated in the middle of the middle column will enter next, altogether. Note that we are reducing the delay as much as possible, so the fly attendant would have to guide the middle seated passengers to go by the right aisle instead, where the right aisle seat isn't occupied yet. After that, we let the remaining passenger in. By doing this, seated passengers won't have to move for one to enter his or her seat.

Disembarking

a. Explanation

Note that the VIP lounge always has the priority to go out first, so the whole plane have to wait for the VIP passengers to exit the plane.

This picture shows the only difference in calculating the disembarking time from the **narrow-body airplane**. The circles are the spaces which are occupied by the passengers sitting in the corresponding seats. The star represents the unoccupied space when the airplane just started to disembark. The orange circles represent the passengers i_1 and i_2 who are about to occupy the star. If $b_{i1} > b_{i2}$ then passenger i_1 occupies the space and vice-versa. If $b_{i1} = b_{i2}$ then it is a fifty-fifty situation.

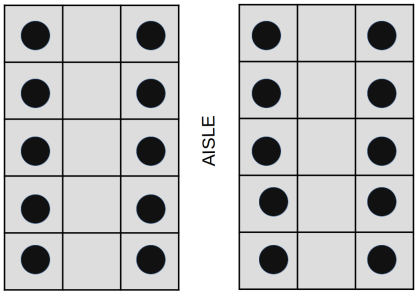
b. Method As the VIP passengers will be the one to exit first, we are assuming that they won't mind the extra carry-on luggage fee, which means we cannot reduce their disembarking time. For passengers seated near the emergency exit, we are assuming that the space for them to move around is wide enough for all the passengers there to get out of their seat and takes their luggage, so the delay in this section is irrelevant. We then apply the solution in Requirement 2e to section 1 and 2, since passengers here are economy class passengers. The section with higher amount of disembarking time is total disembarking time of the two section. Add that to the VIP passengers disembarking time to determine the lowest amount of disembarking time for this airplane type.

5 Requirement 4

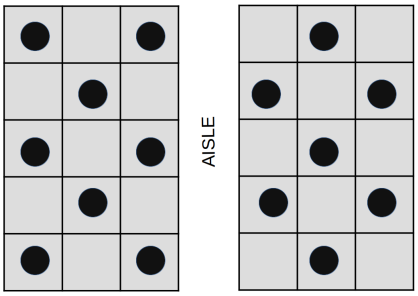
5.1 Airplane Seats Diagram In Covid-19

Based on the Covid-19 pandemic and the social distancing rules, we need to reduce the amount of passengers. And after doing some research about airplanes during Covid-19, we have analyze and found out the new seating locations as follows for Figure 1. Denote that • is the representation of a passenger

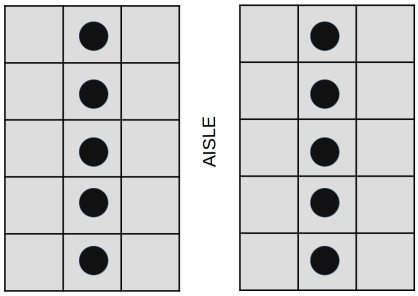
- **Analysis for the passenger's arrangement in case of 70%**



- **Analysis for the passenger's arrangement in case of 50%**



- **Analysis for the passenger's arrangement in case of 30%**



For the two other figures 'Flying Wing' and 'TETA', we can also arrange the seats similarly. With this, in the boarding time, the time in 'getting into seats' phase will reduce. And because of that, the time will be the most effective if the queue is more 'back-to-front'.

Moreover, because of the 2m social distancing rule, the distance between successive passengers will increase by 2m, making the boarding and disembarking time slower.

5.2 Method modification

5.2.1 Narrow Body

In the narrow body airplane, the 'Steffen Modified' and 'Our own method' will be the most effective. Since less seats in a row means that there will be less time to do the 'getting into seats' phase. This as a result will be more efficient to create a method that is more of a back-to-front method, in order to reduce the time in the 'stowing bags' method.

5.2.2 Flying Wing

With regards to the 'Flying Wing' airplane, our group's prescribed methods for both disembarking and boarding will stay unchanged, because the method's target is to minimize these two amounts of time by avoiding all of the factors which are capable of causing delay among people. Particularly, our team's method is to ensure the continuity of the boarding and disembarking process, which means that the number of passengers is not the main concern, but the continuous operation of the process is the main thing that the plane's managers have to worry about. And that continuous operation is not affected by the number of passengers, but by the delaying factors that our team pointed out before.

Thus, it is our group's contention that the method does not have to be modified in case the plane's capacity has to change due to many external factors.

5.2.3 TETA

The same goes to "Flying Wing" passenger aircraft.

6 Requirement 5

Letter to the director

Dear Sir/Madam, As we all know, a great amount of turnaround time of airplanes is spent on the process of loading and unloading passengers. In order to shorten these delays, we need a tool to calculate the time this process takes. Our team has developed a mathematical model which can do this exactly for three most popular airplane types, therefore being able to suggest optimal solutions for this problem.

Calculating the amount of boarding and disembarking time is very complicated, due to factors such as when passengers need to stow the carry-on luggage, hence stopping the entire passengers queue. Moreover, each passenger has different amounts of luggage, so the time it takes to stow will vary. This means that to calculate every single one of the passengers is nearly impossible without high-level mathematics. However, we have developed a simpler solution to this problem.

Instead of calculating all the factors above, our team uses a program that takes account of change after a certain amount of time has passed. In other words, from the moment the first passengers start moving, our model will start a virtual timer which will stop at certain intervals to check what each passenger is doing and what they can do after the stop. The disembarking method also relies on a virtual timer, but also using a scaling system which decides which passenger to move or stay still. Our program proved to be able to produce fast and accurate results for several random passenger lists, and it also proved to be effective when applied to the standard 'Narrow-Body' airplane. We also applied this model to two more airplane types, the 'Flying Wing' and the 'Two Entrances, Two Aisles'. Results collected are not far from the average time in real life. COVID-19 restrictions have also been accounted for, and we used data for when the plane only has 70%, 50% and 30% capacity to adjust our recommendations.

After collecting data from our programs, we have found that the current Back to Front method being widely used is actually less efficient than randomly boarding everyone. Instead, the lesser-used Outside-In method is actually, in our opinion, the fastest method of boarding while still being practical, unlike the theoretically-perfect Steffen method which requires extreme cooperation from all passengers. As for disembarking, we have found that the best method minimizes the delays in the aisles, therefore it is inherently better for passengers to debark by columns instead of by rows, which most airlines are utilizing.

We have tried to make our model work out the total time for each scenario as fast as possible. However due to shortage of time, this model is not as perfect as it could be, and may be subject to further research. Despite this, our model still displays favorable results, and we hope that you will put serious considerations into our model. If you need any further information, please do not hesitate to contact us back. We hope to receive your reply soon.

Yours sincerely,

Team IMMC2022071

7 Conclusion

In conclusion, we have devised a new model for computing the time of the loading and unloading process of three types of commercial airplanes that takes into account various factors of all passengers on board, including the speed and amount of luggage of each person.

For the standard 'Narrow-Body', it is quite nearly impossible to determine the time each passenger takes to get in their seat without complex mathematics due to how many factors that need considering. Hence, we had to improvise and develop a program in Python that assess changes in the assigned 'property' values of each passenger. After running no less than 2000 instances, we have arrived at the conclusion that the currently in use 'Back to Front' method is one of the least efficient method, and the 'Outside-In' method is the most practical method for airlines to apply. As for disembarking, apparently exiting by columns is faster than exiting by rows, which airlines are also using widely.

Regarding the two other models of airplane, the premise of the program is the same: a virtual timer that stops at certain intervals to assess change. However, there are a few additions that need consideration, so our program received some modifications, taking account of the extra aisles and extra queues included in the plane types. The most optimal methods for these two figures are also the same as the 'Narrow-Body' plane: each aisle needs to follow the 'Outside-In' method for embarking and passengers exit by columns instead of by rows.
As far as the COVID-19 pandemic is concerned, our team assumed the best possible pattern of seats on all plane types, and modified our program so that it fits the less capacity during the quarantine. The collected data indicates that the best methods for 100% capacity are also considered the fastest methods for cases 70%, 50%, and 30% capacity respectively.

8 References

References

Eitan Bachmat, Daniel Berend, Luba Sapir, Steven Skiena, and Natan Stolyarov. Analysis of airplane boarding times. *Operations Research*, 57(2):499–513, 2009.

Andrew Best, Sean Curtis, David Kasik, Christopher Senesac, Tim Sikora, and Dinesh Manocha. Ped-air: A simulator for loading, unloading, and evacuating aircraft. *Transportation Research Procedia*, 2:273–281, 2014. ISSN 2352-1465. doi: https://doi.org/10.1016/j.trpro.2014.09.052. URL https://www.sciencedirect.com/science/article/pii/S235214651400088X. The Conference on Pedestrian and Evacuation Dynamics 2014 (PED 2014), 22-24 October 2014, Delft, The Netherlands.

Aarian Marshall. The art and science of boarding an airplane in a pandemic, Jan 2021. URL https://www.wired.com/story/art-science-boarding-airplane-pandemic/.

R John Milne, Camelia Delcea, and Liviu-Adrian Cotfas. Airplane boarding methods that reduce risk from covid-19. *Safety Science*, 134:105061, 2021.

Hendrik Van Landeghem and Annelies Beuselinck. Reducing passenger boarding time in airplanes: A simulation based approach. *European Journal of Operational Research*, 142(2):294–308, 2002.

Bachmat et al. (2009) Best et al. (2014) eve int Marshall (2021) Milne et al. (2021) uni Van Landeghem and Beuselinck (2002)

9 Appendix

9.1 Code Model For Boarding

```
TIME_TO_MOVE_BETWEEN_SEATS = 2
import matplotlib.pyplot as plt
import time
import numpy as np
import random

def bruh():
    queue1 = [[i*6+1,i*6+2,i*6+3] for i in range(0,33,2)]
    queue1 = sum(queue1,[])
    random.shuffle(queue1)
    queue2 = [[i*6+4,i*6+5,i*6+6] for i in range(0,33,2)]
    queue2 = sum(queue2,[])
    random.shuffle(queue2)
    queue3 = [[i*6+1,i*6+2,i*6+3] for i in range(1,33,2)]
    queue3 = sum(queue3,[])
    random.shuffle(queue3)
    queue4 = [[i*6+4,i*6+5,i*6+6] for i in range(1,33,2)]
    queue4 = sum(queue4,[])
    random.shuffle(queue4)


    queue = queue1+queue2+queue3+queue4

    number_of_bags = [random.randint(0,2) for i in queue]

    mytime = 0
    #queue = [i for i in range(60,0,-1)]
    #queue = [9, 2, 7,10,11, 3,12, 1, 6, 4, 8, 5]
    def get_your_seat_row(index):
        i = (queue[index]-1)%6
        s1 = [0,1,2]
        s2 = [3,4,5]
        if i in s1:
            return [j + queue[index]-i for j in s1]
        elif i in s2:
            return [j + queue[index]-i for j in s2]

```

```
    def get_priority_seats(queue, index):
        seat_row = get_your_seat_row(index)
        previous_seats_occupied = []
        passenger_index = (queue[index]-1)%6
        for i in range(len(queue)):
            if i == index:
                break
            if queue[i] in seat_row:
                previous_seats_occupied.append((queue[i]-1)%6)
        if passenger_index in [3,4,5]:
            passenger_index = 5 - passenger_index
            previous_seats_occupied = [5 - j for j in
            ↪ previous_seats_occupied]
        previous_seats_occupied.sort()
        return [passenger_index, previous_seats_occupied]

    def get_priority_time(queue,index):
        if get_priority_seats(queue,index)[0] == 2:
            return TIME_TO_MOVE_BETWEEN_SEATS
        elif get_priority_seats(queue,index)[0] == 1:
            if get_priority_seats(queue,index)[1] == [] or
            ↪ get_priority_seats(queue,index)[1] == [0]:
                return TIME_TO_MOVE_BETWEEN_SEATS
            elif get_priority_seats(queue,index)[1] == [2] or
            ↪ get_priority_seats(queue,index)[1] == [0,2]:
                return 3*TIME_TO_MOVE_BETWEEN_SEATS
        elif get_priority_seats(queue,index)[0] == 0:
            if get_priority_seats(queue,index)[1] == []:
                return TIME_TO_MOVE_BETWEEN_SEATS
            elif get_priority_seats(queue,index)[1] == [2]:
                return 3*TIME_TO_MOVE_BETWEEN_SEATS
            elif get_priority_seats(queue,index)[1] == [1]:
                return 4*TIME_TO_MOVE_BETWEEN_SEATS
            elif get_priority_seats(queue,index)[1] == [1,2]:
                return 6*TIME_TO_MOVE_BETWEEN_SEATS



    time_to_get_into_seats = [get_priority_time(queue, i) for i in
    ↪ range(len(queue))]
    length_of_passenger = [20 + 40*number_of_bags[i] for i in
    ↪ range(len(queue))] #(cm)
```

```
    row_queue = [(int((i-1)/6)+1) for i in queue] #(row of queue)
    time_stow_bags = [10*number_of_bags[i] for i in range(len(queue))]
    ↪ #(seconds)
    total_time_to_wait = [time_stow_bags[i] + time_to_get_into_seats[i] for i
    ↪ in range(len(queue))]
    pos =[0]
    s = 0
    for i in range(0,len(queue)-1):
        s += -int((length_of_passenger[i] + length_of_passenger[i+1])/2)
        pos.append(s) #(cm)
    while True:
        #print(f'Time is {mytime} and the queue_pos is {pos}')
        mytime += 1
        for i in range(0, len(queue)):
            if pos[i] >= row_queue[i]*80 and total_time_to_wait[i] >= 1:
                total_time_to_wait[i] -= 1
                pass
            elif pos[i] >= row_queue[i]*80 and total_time_to_wait[i] == 0:
                pos.pop(i)
                queue.pop(i)
                row_queue.pop(i)
                total_time_to_wait.pop(i)
                break
            else:
                if i == 0:
                    t = min([40,row_queue[0]*80-pos[0]])
                    pos[0] += t
                elif pos[i-1] >= pos[i] + int((length_of_passenger[i-1] +
                ↪ length_of_passenger[i])/2):
                    a = pos[i-1] - pos[i] - int((length_of_passenger[i-1] +
                    ↪ length_of_passenger[i])/2)
                    t = min([a, 40,row_queue[i]*80-pos[i]])
                    pos[i] += t
        if pos == []:
            break
    return mytime

t = time.time()

#queue1 = [i for i in range(1,194,6)] + [i for i in range(6,199,6)]
#random.shuffle(queue1)
#queue2 = [i for i in range(2,195,6)] + [i for i in range(5,198,6)]
```

```
#random.shuffle(queue2)
#queue3 = [i for i in range(3,196,6)] + [i for i in range(4,197,6)]
#random.shuffle(queue3)

#queue1 = [i for i in range(133, 199)]
#random.shuffle(queue1)
#queue2 = [i for i in range(67, 133)]
#random.shuffle(queue2)
#queue3 = [i for i in range(1,67)]
#random.shuffle(queue3)

#queue1 = [i for i in range(1,13)]
#random.shuffle(queue1)
#queue2 = []
#queue3 = []









def get_new_queue(queue, number_of_people_who_dont_listen):
    queue = [i for i in queue]
    list_of_unlisten_people = random.sample(queue,
    ↪ number_of_people_who_dont_listen)
    for person in list_of_unlisten_people:
        for i in range(len(queue)):
            if person == queue[i]:
                queue.pop(i)
                break

    for person in list_of_unlisten_people:
        queue.insert(random.randint(0, len(queue)), person)

    return queue

def get_number_of_bags(queue, average_number_of_bags):
    sum = int(average_number_of_bags * len(queue))
    number_of_bags = np.random.multinomial(sum,
    ↪ np.ones(len(queue))/len(queue))

    return number_of_bags

```

```
154 def mybruh(queue, average_number_of_bags):
155     array = []
156     for i in range(200):
157         array.append(bruh(queue, get_number_of_bags(queue,
            ↪  average_number_of_bags)))
158     return array[100]
159 
160 def mymybruh(queue, number_of_people_who_dont_listen):
161     array = []
162     for i in range(200):
163         array.append(bruh(get_new_queue(queue,
            ↪  number_of_people_who_dont_listen),[1 for i in
            ↪  range(len(queue))]))
164     return array[100]
165 
166 
167 array = []
168 
169 for i in range(2000):
170     array.append(bruh())
171 
172 array.sort()
173 print((array[999]+array[1000])/2)
174 print(array[99])
175 print(array[1949])
176 
177 
178 
179 
180 #print(mymybruh(queue, 10))
181 #print(mymybruh(queue, 20))
182 #print(mymybruh(queue, 30))
183 #print(mymybruh(queue, 40))
184 #print(mymybruh(queue, 50))
185 #print(mymybruh(queue, 60))
186 #print(mymybruh(queue, 70))
187 #print(mymybruh(queue, 80))
188 #print(mymybruh(queue, 90))
189 #print(mymybruh(queue, 100))
190 #print(mymybruh(queue, 110))
191 #print(mymybruh(queue, 120))
192 #print(mymybruh(queue, 130))
```

```
#print(mymybruh(queue, 140))
#print(mymybruh(queue, 150))









#with open ("bruh.txt","w",encoding="utf-8") as f:
#    f.write(str(array))
```