IA Section B

B.1 Design

Directory

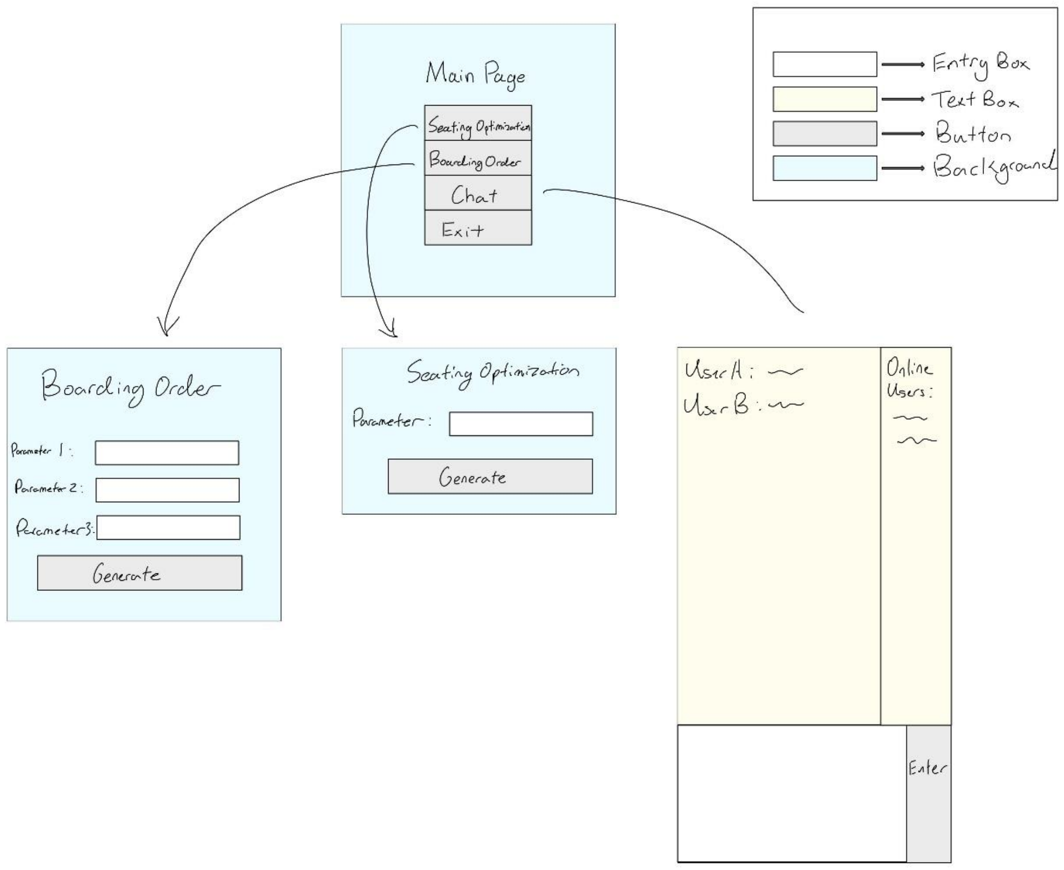
Design Methodology:

Process	Explanation
Research	1. Search for existing methods 2. Compare Methodologies
Design	1. Plan the general Structure of the Product a) Plan the Windows for the Product b) Create a Hierarchy for the windows leading from the Main Menu 2. General Design of Algorithms a) List boarding order for Cellular Automata b) Rules for Cellular Automata, c) Flowchart for Genetic Algorithm 3. UML Diagram for entire program 4. Test Plan: Highlighted specific functions for the program to be considered successful

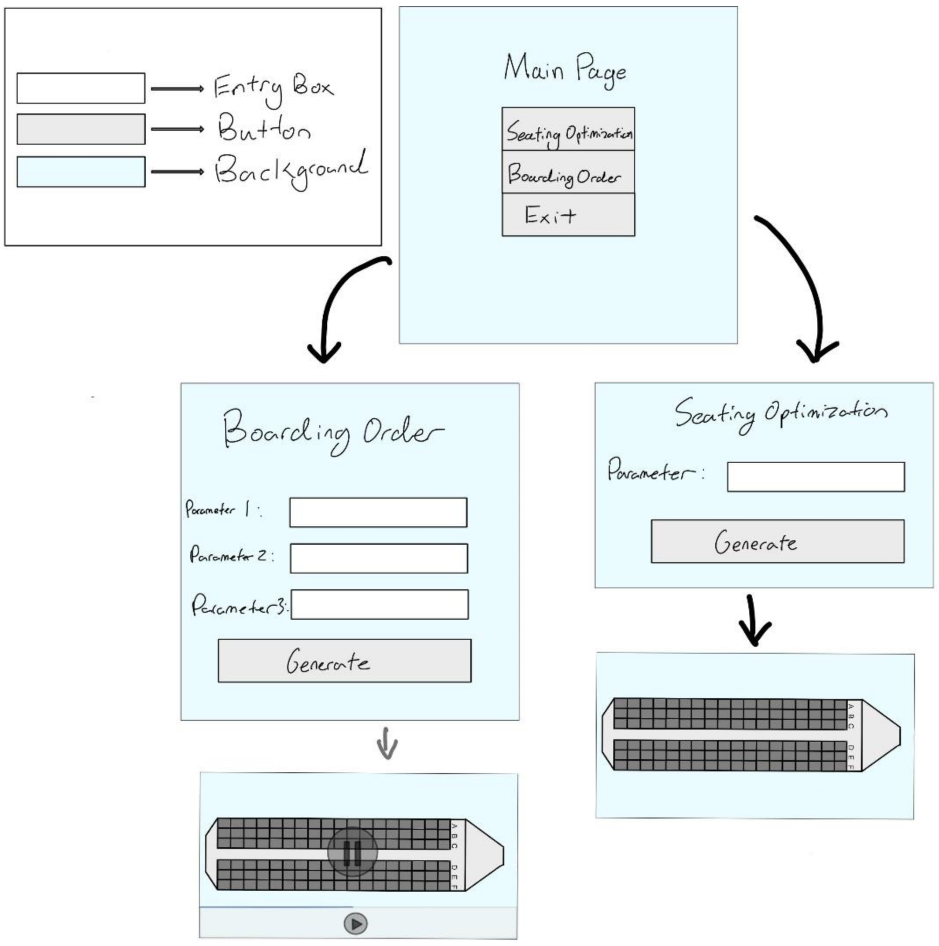
This is a good design process for the program because it goes from the general hierarchy in the program and narrows down gradually to specific algorithms and to classes within the general program and algorithms. Doing so provides insight to the overall program when designing specifics, reducing the chance of making major problems the in the program, and help find potential loopholes in the general design. The research part also comes before the design so that we are certain that we have the best solution to the problem, and sufficient insight into the problem.

The test plan comes last because it needs to be based on the entire design of the program as it tests specific functions.

Interface Visualization (Old):



Interface Visualization (New):

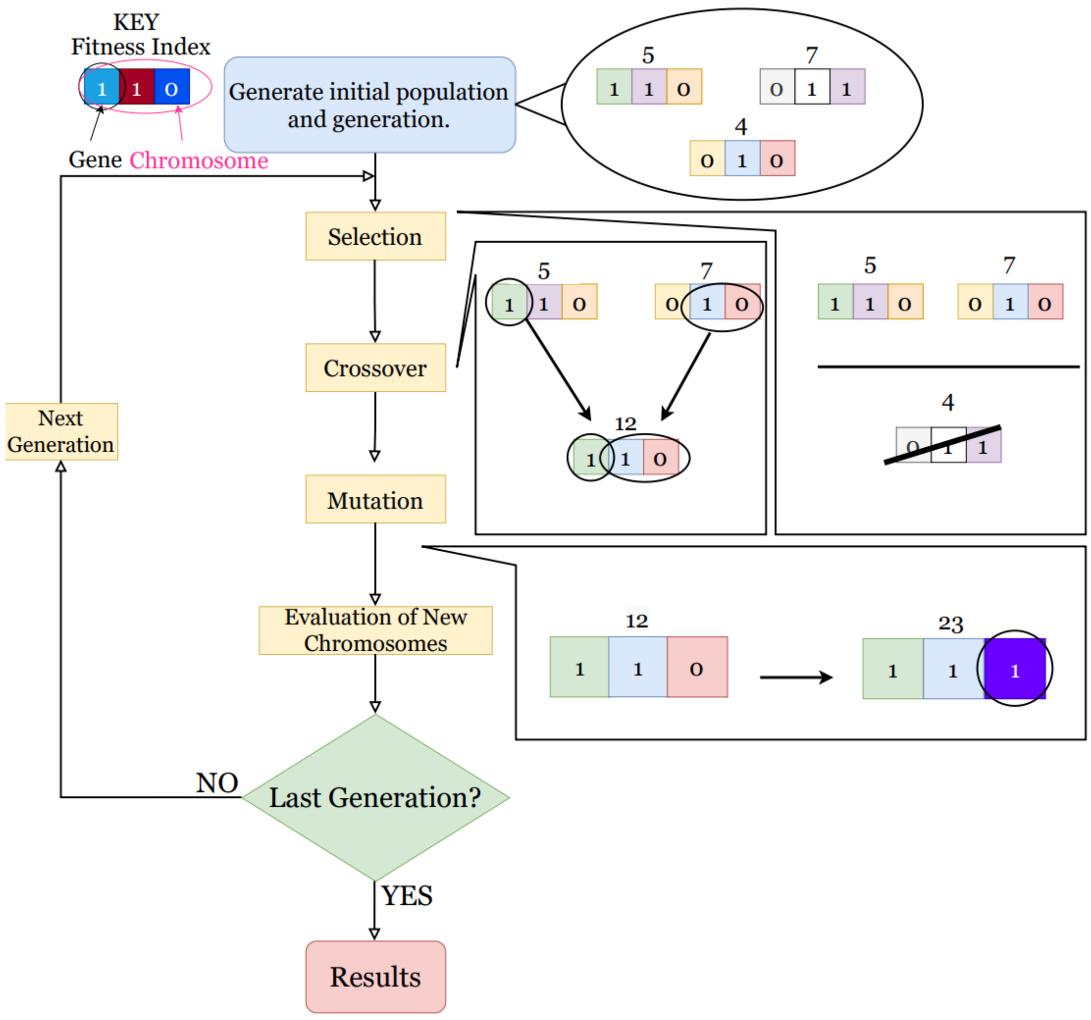


Genetic Algorithm Flowchart:

To obtain the optimal seating arrangement, we use the genetic algorithm. A flowchart illustrating the process of the genetic algorithm is shown below. To clarify, a higher fitness index indicates a higher rate of survival. The process of Genetic Algorithm is as follows:

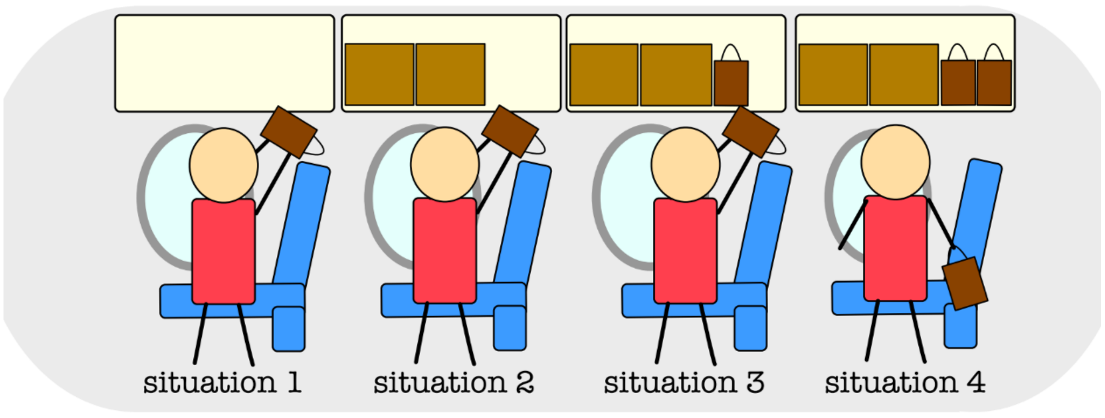
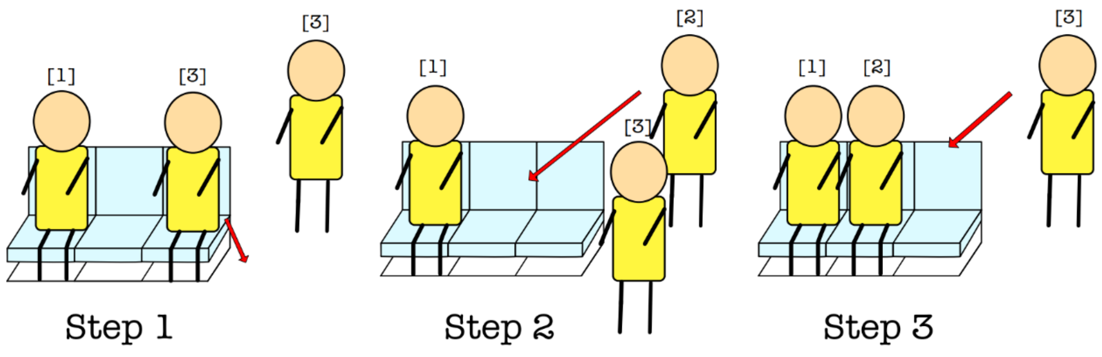
1	We set the initial population in a matrix environment that is organized analogously to a plane seating chart, and use the genetic algorithm inside the environment of the cellular automaton.
2	We set the initial population in a matrix environment that is organized analogously to a plane seating chart, and use the genetic algorithm inside the environment of the cellular automaton.
3	Then, we select "good" iterations, or seating arrangements where the distances between nearby passengers are relatively maximized.

4	We then crossover elements of good iterations and combine them into a hypothetically better iteration.
5	Then, we allow these newly created seating arrangements to adjust itself to the better.



Seating Situations:

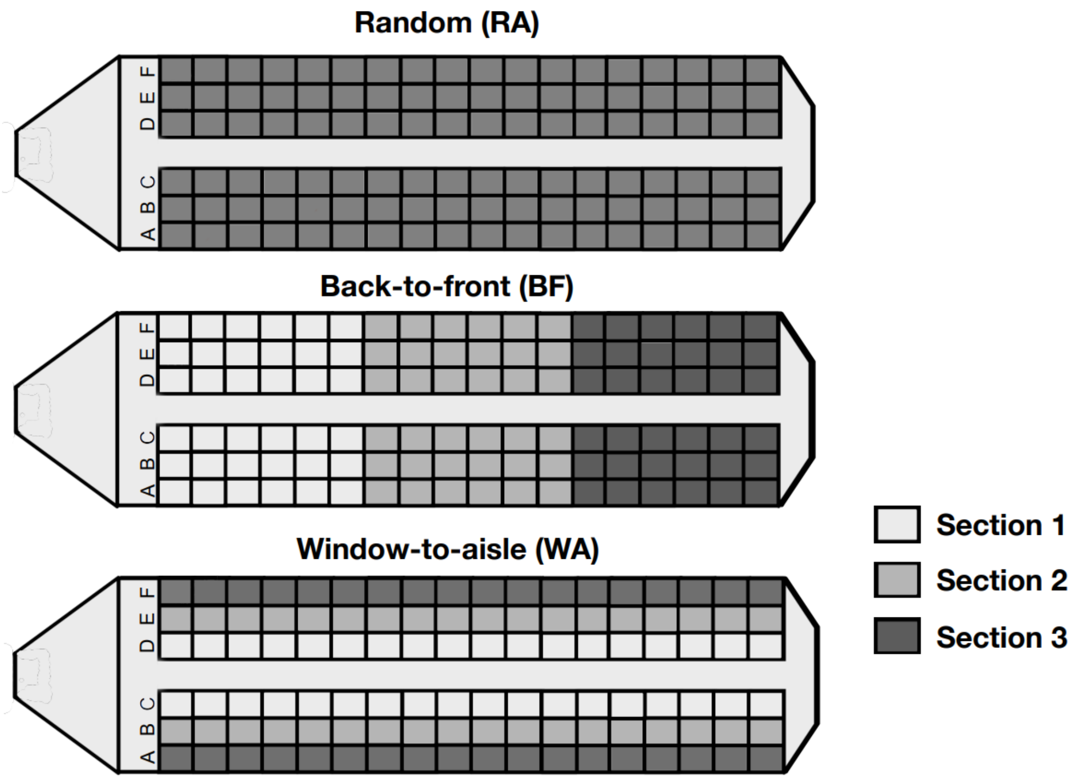
Each passenger has a corresponding time taken to move out of the aisle and sit down in their correct seat upon reaching their aisle. This is calculated based on their luggage situation, the luggage storage capacity of their respective aisle.

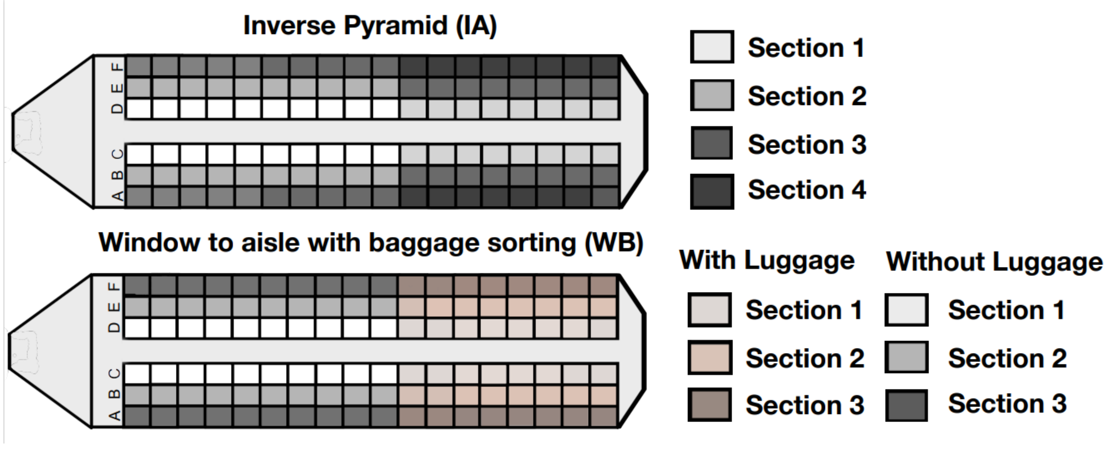
Factor	Description
Baggage	Each passenger has a chance to carry a purse bag and a carry-on luggage. $$L_i = (5 * c_i + 3 * p_i) \cdot F(x, \kappa, \lambda)$$ Li expresses the i-th passenger's increase in seating time due to luggage, ci expresses whether the i-th passenger holds a carry-on luggage, pi expresses whether the i-th passenger holds a purse, and F (x, κ, λ) is the Weibull cumulative distribution function $F(x, \kappa, \lambda) = 1 - e^{-\left(\frac{x}{\lambda}\right)^{\kappa}}$ $of\ probabilisic\ seating\ time\ distribution.$
Storage	 In situation 1, the passenger would not experience any increase in luggage stowage time. In situations 2 and 3, the passenger would experience some increase in stowage time. In situation 4, the passenger would experience the highest luggage stowage time of 5 timesteps.
Seating	The seating situation at the passenger's aisle also affects the time to seat of the passenger since passengers already seated may have to move out of the way to allow other passengers to sit.  [1] [3] [3] Step 1 [1] [2] [3] Step 2 [1] [2] [3] Step 3

Boarding Orders:

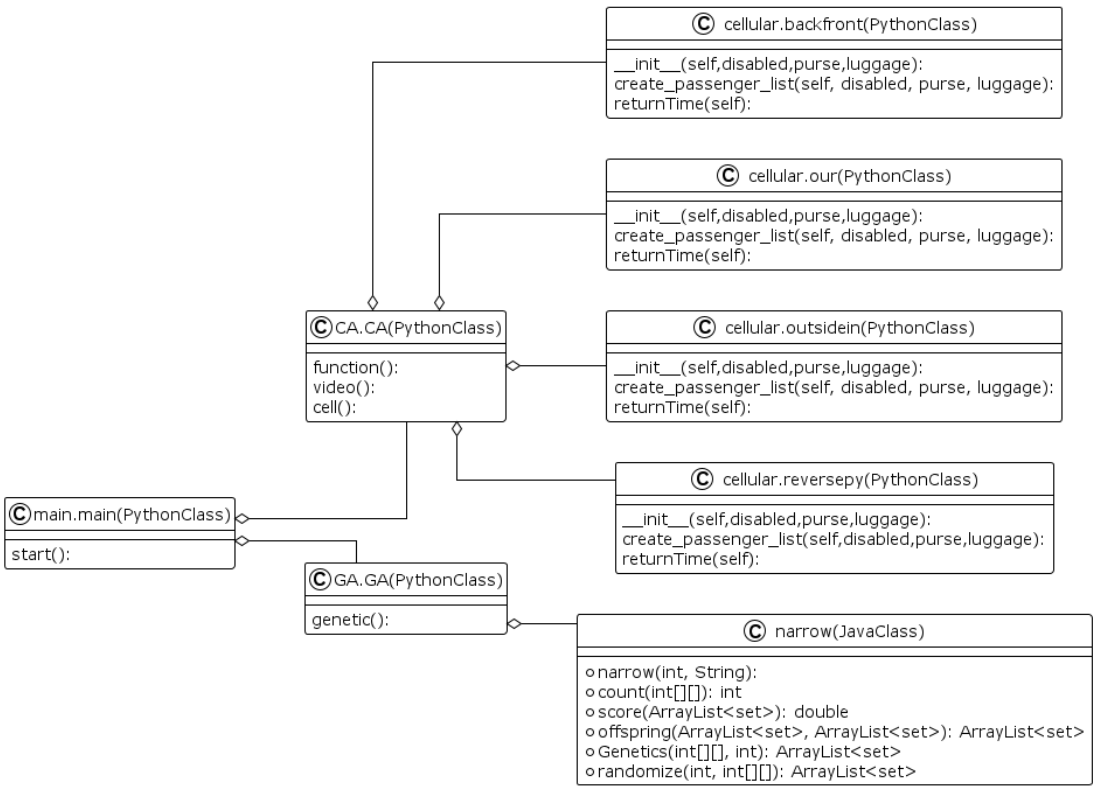
List of Potential Boarding orders

In the order of Section 3/4 to Section 1:





UML Demonstration of Design:


cellular.backfront(PythonClass)
__init__(self,disabled,purse,luggage):
create_passenger_list(self, disabled, purse, luggage):
returnTime(self):
cellular.our(PythonClass)
__init__(self,disabled,purse,luggage):
create_passenger_list(self, disabled, purse, luggage):
returnTime(self):
CA.CA(PythonClass)
function():
video():
cell():
cellular.outsidein(PythonClass)
__init__(self,disabled,purse,luggage):
create_passenger_list(self, disabled, purse, luggage):
returnTime(self):
cellular.reversepy(PythonClass)
__init__(self,disabled,purse,luggage):
create_passenger_list(self,disabled,purse,luggage):
returnTime(self):
main.main(PythonClass)
start():
GA.GA(PythonClass)
genetic():
narrow(JavaClass)
narrow(int, String):
count(int[][]): int
score(ArrayList<set>): double
offspring(ArrayList<set>, ArrayList<set>): ArrayList<set>
Genetics(int[][], int): ArrayList<set>
randomize(int, int[][]): ArrayList<set>

Test Plan

Criterions	Idea	Explanation	Example
1.1	Main Menu	Main Page, like designed in Interface Visualization, has to link to the windows for the individual functions of the program.	By clicking on the button indicating seating optimization, it must lead to a page with the corresponding function”
1.2	Reusability of Windows and Menus	The interface may be reused to for repeated used of the program without needing to restart the program.	After the entry of parameters (12,23,14) in the seating plan menu and successful visualization, it can be directly used again with another set of parameters.
2.1/1.3	Window for Seating Optimization	There must be a working, appropriate window for the Seating optimization function of the program including the necessary interactions with the user for preliminary information of the algorithm.	“The window may contain a textbox for the entry of parameter “number of passengers” and a button to generate the seating plan according to that.

2.2	Parameters and Implementation of Seating Optimization	The program must accept 1 parameter (number of passengers) for each use. Based on the parameter, the algorithm must compute the optimal seating plan for the flight based on the procedures outlined in genetic algorithm flowchart.	With the input of 50 passengers, the program produces a seating distribution of 50 passengers with the least risk for virus transmission.
3.1/1.3	Window for Boarding Method	There must be a working, appropriate window for the Boarding Method function of the program including the necessary interactions with the user for preliminary information of the algorithm.	The window may contain 3 textboxes for the entry of parameters and a button to produce the best boarding methods according to that.
3.2	Parameters and Implementation of Boarding Method Comparison	The program must accept 3 parameters (Passengers with Luggage, Passengers with Purse, Passengers, with disabilities) for each use. Based on the parameter, the algorithm must compute the optimal boarding method for the flight out of the four described in boarding orders.	With the input of (20,10,50), the program will compute the optimal boarding method.
4.1	Visualization for Seating Optimization	There must be a visualization method for the audience for the results of the Seating Optimization Algorithm. It must be simple to understand and apply.	A picture presented to the user of the seating distribution, with different colors representing empty, taken seats and the hallway.
4.2	Visualization for Boarding Methods	There must be a visualization method for the audience for the results of the Boarding Method Comparison. It must be simple to understand and apply.	Video outlining the process of the order in which the passenger boards, with color scheme separating empty seats, taken seats, persons in motion, and the hallway.

5	Error handling	In the case of Wrong inputs, the program must be able to handle them properly.	Suppose the airplane capacity is 191 individuals, the program must stop the user from entering number greater than 191 and less than 0

Word Count: 215