

IA Section C

Haipeng Chang

Directory

Haipeng Chang 1

1. Genetic Algorithm for Seating Space Optimization..... 2

2. Java- Python interactions through jpye library 7

3. Cellular Automata for simulation 8

4. Boarding order and Boarding order implementation by generation pre-simulation 11

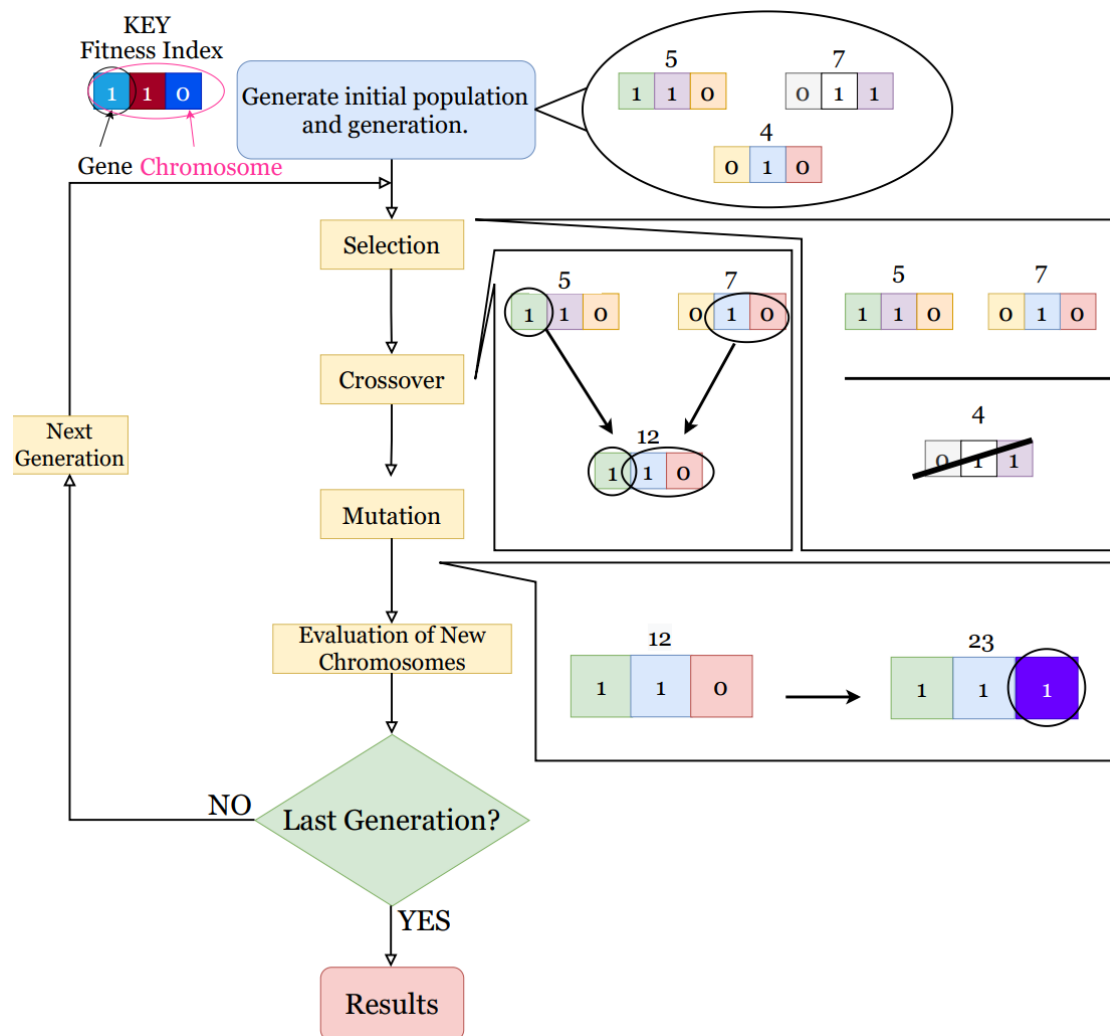
5. Parameters for Cellular Automata 16

6. Visualization of Simulation using MoviePy 18

1. Genetic Algorithm for Seating Space Optimization

Summary	The product presents an adapted algorithm based off the theoretical structure of genetic algorithm. It is an effective and accurate way to meet the airline companies' and passengers' need of a safe flight.
Ingenuity	The program consists of a completely original algorithm, based on algorithmic theory introduced by John Holland in 1975. It is split into a theoretical structure, based on the theory of evolution, which is adapted and applied in my program adapted to the specific purpose to reduce covid transmittance probability. The airplane was creatively and conveniently modeled as a simple matrix, with the algorithm composing of editing and creating matrices is representing seating. With initially randomized seating plans acting as individuals, and having the most optimal individual combining advantageous traits to produce the next generation, the individuals in the new generations grows fitter by interactions. This serves as a more effective and efficient method compared to permutating all possible arrangements, which would make to time complexity unacceptable for user interactions.
Client needs	The programs solve the client's need for in-flight safety. It improves the consumer's willingness to take flight from maximized safety (for airline companies), and improves mental insurance for consumers for a presentation of the safest possible flight. The algorithm also saves time for the client compared to other possible methods. The matrix representation of the airplane passenger distribution also makes it easier to make effective visualizations for the client to easily understand.
Source	Murray-Smith, David J. '6 - Experimental modelling: system identification, parameter estimation and model optimisation techniques'. <i>Modelling and Simulation of Integrated Systems in Engineering</i> . Εμπέδ. David J. Murray-Smith. Woodhead Publishing, 2012. 165–214. Web.

Extended explanation:



This is a diagram explaining the function of the algorithm. The diagram is referenced in the design of the code, and referenced an underlined in the explanation of code.

Matrix of airplane

- Class called by python
- Stores the optimized seating matrix in txt file
- Prints minimized probability in file

```

public static ArrayList<set> Genetics(int [][] a, int count) {
    HighScore=new ArrayList<Double> ();
    int generations=500;
    int num=count(a);
    //initialize
    ArrayList<ArrayList<set>> thing=new ArrayList<ArrayList<set>> ();
    ArrayList<Double> score=new ArrayList<Double> ();
    top=new ArrayList<Double>();
    for(int i=0;i<10;i++) {
        thing.add(randomize(count,a));
        score.add(score(thing.get(i)));
    }
    Main loop for algorithm
    for(int gen=0;gen<generations;gen++) {
        int first=0;
        int second=0;
        for(int i=0;i<score.size();i++) {
            if(score.get(i)<score.get(first)) {
                first=i;
            }
        }
        for(int i=0;i<score.size();i++) {
            if(score.get(i)<score.get(second) && i!=first) {
                second=i;
            }
        }
        for(int i=0;i<thing.size();i++) {
            if (i!=first && i!=second) {
                thing.set(i, offspring(thing.get(first),thing.get(second)));
                score.set(i,score(thing.get(i)));
            }
        }
        double max=score.get(0);
        for(double d:score) {
            if(d<max) {
                max=d;
            }
        }
        HighScore.add(max);
    }
    int max=0;
    for(int i=0;i<score.size();i++) {
        if(score.get(i)<score.get(max)) {
            max=i;
        }
    }
    return thing.get(max);
}

```

→ 50 generations to be iterated

Storing

Initialization

Iteration → Evaluate-Select -breed cycle

Select/Evaluate

the two fittest indiv. for breeding
Minimizes the result of the fitness function.
The less the value is, the less probability for transmittance

Two breed and produces a new generation.

Finds fittest individual in last generation

return fittest.

```

public static ArrayList<set> randomize(int a,int [][] ba) {
    ArrayList<set> array=new ArrayList<set> ();
    array=new ArrayList<set> ();
    ArrayList<set> exists=new ArrayList<set> ();
    while (array.size()<a) {
        int x=(int) (Math.random()*ba.length);
        int y=(int) (Math.random()*ba[0].length);
        boolean b=true;
        for(set s:exists) {
            if(s.x==x && s.y==y) {
                b=false;
            }
        }
        if(ba[x][y]==0) {
            b=false;
        }
        if(b==true) {
            set s=new set (x,y);
            array.add(s);
            exists.add(s);
        }
    }
    return array;
}

```

→ generate until filling airplane with unique coordinate pairs.

ruling out repeated coordinates

passenger only sits on seats

→ return passenger seating coordinates.

```

public static ArrayList<set> offspring(ArrayList<set> one, ArrayList<set> two) {
    int rand1=(int) (Math.random() * (one.size() / 2));
    int rand2=(int) (Math.random() * (one.size() / 2) + one.size() / 2);
    int num=(int) (Math.random() * 2);
    ArrayList<ArrayList<set>> union=new ArrayList<ArrayList<set>>();
    union.add(one);
    union.add(two);
    ArrayList<set> New=new ArrayList<set>();
    if(num==1) {
        for(int i=0;i<one.size();i++) {
            if(i>rand1&&i<rand2) {
                New.add(union.get(1).get(i));
            }
            else {
                New.add(union.get(0).get(i));
            }
        }
    }
    else{
        for(int i=0;i<one.size();i++) {
            if(i>rand1&&i<rand2) {
                New.add(union.get(0).get(i));
            }
            else {
                New.add(union.get(1).get(i));
            }
        }
    }
    double mutate=Math.random();
    if(mutate<=0.5) {
        while(true) {
            int x=(int) (Math.random() * narrow.length);
            int y=(int) (Math.random() * narrow[0].length);
            if (narrow[x][y]==0) {
                continue;
            }
            else{
                int rand3=(int) Math.random() * one.size();
                New.set(rand3, new set (x,y));
                break;
            }
        }
        while(true) {
            int x=(int) (Math.random() * narrow.length);
            int y=(int) (Math.random() * narrow[0].length);
            if (narrow[x][y]==0) {
                continue;
            }
            else{
                int rand3=(int) Math.random() * one.size();
                New.set(rand3, new set (x,y));
                break;
            }
        }
        while(true) {
            int x=(int) (Math.random() * narrow.length);
            int y=(int) (Math.random() * narrow[0].length);
            if (narrow[x][y]==0) {
                continue;
            }
            else{
                int rand3=(int) Math.random() * one.size();
                New.set(rand3, new set (x,y));
                break;
            }
        }
    }
    return New;
}

public static int count(int [][]a) {
    int count=0;
    for(int i=0;i<a.length;i++) {
        for(int j=0;j<a[i].length;j++) {
            if (a[i][j]==1) {
                count++;
            }
        }
    }
    return count;
}

public static double score(ArrayList<set> a) {
    double score=0;
    for(int i=0;i<a.size();i++) {
        for(int j=i+1;j<a.size();j++) {
            if (Math.sqrt(Math.pow((double) (a.get(i).x-a.get(j).x),2)+Math.pow((double) (a.get(i).y-a.get(j).y),2))>0) {
                if (Math.sqrt(0.75*Math.pow((double) (a.get(i).x-a.get(j).x),2)+0.75*Math.pow((double) (a.get(i).y-a.get(j).y),2))<4) {
                    score+=18.19*Math.log(Math.sqrt(0.75*Math.pow((double) (a.get(i).x-a.get(j).x),2)+0.75*Math.pow((double) (a.get(i).y-a.get(j).y),2)))+43);
                }
            }
        }
    }
    return score;
}

```

Breed

Creates an offspring of two parents

Crossover

Combining traits of parents

Mutation

Chance for every coordinate to mutate to a random set of values.

Evaluate

Fitness function based on fluid dynamics.

In the above code, methods and sub-methods presented by the diagram is shown by the blue hand written text. The “genetic” method acts as the main method that controls all the subprocess. It first calls on the “randomize” method to initialize the population in the first generation. The main method contains a loop that iterates, simulating generations passing. It calls on other methods like offspring and score to evaluate the “fitness” of each individual. The fittest individuals are selected to breed new offspring for the new generation. The offspring represents the seating arrangement, and the fitness represents the infection probability.

2. Java- Python interactions through jpype library

Summary:	The python gui application calls on a Java implementation of genetic algorithm. This benefits the app and the client for efficiency and memory usage.
Ingenuity	<p>Python applications are interpreted, which causes runtime lag, whereas Java programs are primitively compiled, which would be faster. The workload of the interpreter is increased by having to determine the variable type at runtime in python. Memory utilization is also impacted by remembering the object type of items that are fetched from container objects.</p> <p>For these reasons, in contrast to the common thought of using python to code the entire app, using a separate java program and calling a JVM serves as an innovative method to improve the speed and efficiency of the algorithm and the app.</p>
Source	<p><i>Python vs. Java Performance</i>. SnapLogic. (2020, September 30). Retrieved October 11, 2022</p> <p>JPype documentation — JPype 1.4.1_dev0 documentation</p>
Client Need	One of the most important aspects of an app is for it to be efficient. While the genetic algorithm is designed to solve the main concern of the Clients, it is very reliant on system resources. Running the algorithm on python instead of java benefits the user's experience with the app, with it taking up less memory and time.

Jpype is a library that allows python script to call a Java Virtual environment and uses it to run java app and create java object within python. The program, through the jpype library, starts a Java Virtual environment for the jar of the genetic algorithm to run in. It creates a java object within the variable "thing", whose initializer runs a program that creates the optimized matrix of seating distribution generated from the genetic algorithm. It interacts with the python program through an output txt file located in the python app directory, where python sets the parameter for a java method through jpype, and receives the output written into the file. The python program interprets the output and stores it as an instance of a "Genetic" python object for use of other python classes by aggregation.

```

import matplotlib.pyplot as plt
from py4j.java_gateway import JavaGateway, GatewayParameters
import jpype → Calling the jpype library
import os
import numpy as np
import json
class Genetic():
    data=0
    passengers=0
    aircraft=0
    def __init__(self, num, type):
        self.passengers = num
        self.aircraft = type
        jarpath = os.path.join(os.path.abspath('.'), r'C:\Users\Eric Chang\PycharmProjects\IA\IA.jar')
        dependency = os.path.join(os.path.abspath('.'), r'C:\Users\Eric Chang\PycharmProjects\IA\dependencies')

        jvmPath = 'C:\Program Files\Java\jdk-18.0.2.1\bin\server\jvm.dll'
        jpype.startJVM(jvmPath, "-ea", "-Djava.class.path=%s" % jarpath)
        JClass = jpype.JClass('geneticRef.narrow')
        thing = JClass(num, type)
        jpype.shutdownJVM()
        data=np.zeros((32,7))
        f = open("C:\\Users\\Eric Chang\\PycharmProjects\\IA\\problemname.txt", "r")
        for i in range(0,32):
            arr=f.readline().split(',')
            for j in range(0,7):
                data[i][j]=int(arr[j])
        self.data=data
        f.readline()
        a=f.readline()
        a=float(a.replace("\n",""))
        #print(a)
        f.close()

```

Defining path to the program in Java

Defining path to Java Virtual machine

Starting JVM through Jpype library

Create Java object in the file through JVM and Jpype method, outputs optimized matrix into text file

Retrieves the matrix from the output file and stores it as instance of this python object

3. Cellular Automata for simulation

Function:	A simulation is performed on different possible boarding methods on to a plane. The program, after the simulation, computes the boarding method with the fastest boarding speed. With method with the least boarding time computed, passenger can board accordingly with the least risk of exposure to covid-19.
-----------	--

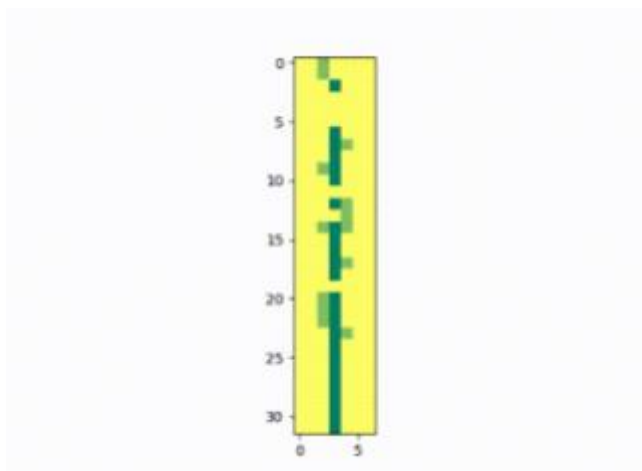
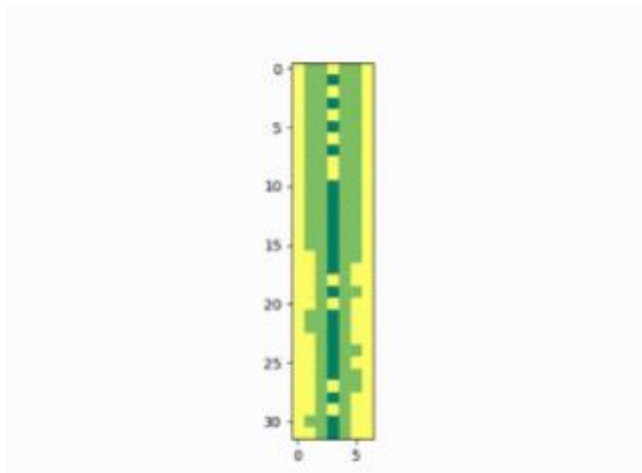
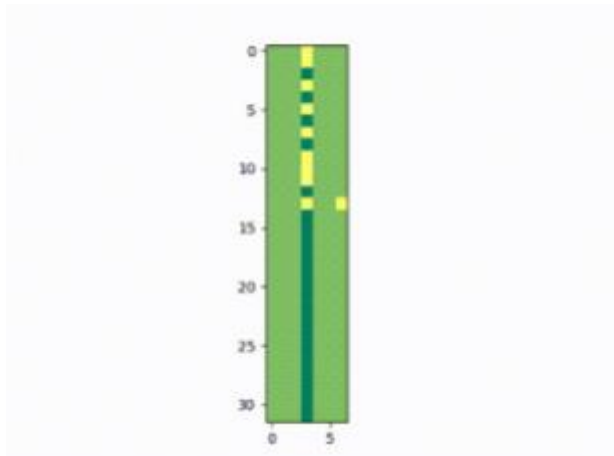
Ingenuity	Cellular automata serve as the best possible method to simulate passenger boarding. Compared to other simulation methods, it accounts for the factors of random chance and time which other algorithms rarely accounts for. Random chance is statistically significant in making a simulation realistic. Measurement of time is required to compare the effectiveness of boarding methods. Cellular automation, which accounts for both serves as a perfect, ingenious way to produce both at once.
Source	Cellular Automata (Stanford Encyclopedia of Philosophy)
Client Need	The time taken for a passenger to board a plane reflects the length of close interactions with other people in the boarding process. Thus, minimizing the time of boarding can guarantee the passenger's safety, and minimize the risk of producing a fully infected flight for the airline.

(Full code is too long... too complicated to list)

General Process of the Cellular Automata:

- A matrix which models the plane is created, containing entrance, hallways, and seats, with each unit of the matrix representing a cell.
- A passengers list is created with randomly assigned seats
- With frames number as the unit for time, passengers are put on the plane in the order assigned by the boarding method
- The passengers, through predefined mathematical rules, travels through the hallway of the plane, searching for their seats frame by frame, moving to adjacent cells until finding the right row, the taking two frames to seat
- Once all passengers found their seats, the number of frames taken for the process to finish is outputted for comparison.

The process looks as follows.



Dark Green- Hallway

Light Green- Seat

Yellow- Passengers

With reference to the next complexity, this is a demonstration of the reverse pyramid methods, which can be seen through the shape of the second image.

4. Boarding order and Boarding order implementation by generation pre-simulation

Function:	In the Cellular Automata, real life boarding methods are organized before the simulation to provide the most efficient program, providing speed and efficiency.
Ingenuity	Real-life, scientific methods that were commonly used by airline companies are incorporated in the Cellular Automata may sometime be hard to implement real-time. This method acts as an ingenious method as it provides an effective approach to boarding airplanes, through simple code before the simulation. By simply generating the lists of passengers before boarding by the correct designated order, 4 variations of boarding methods can be successfully implemented. In contrast to other methods to changing the boarding order of the simulation like by complicated Realtime decision making of the passengers during the simulation, this proves to be the best way of producing the order: by computing the order into a passenger list before the simulation.
Source	Bidanda, Rahul & Geng, Zhaohui & Winakor, J. & Vidic, Natasa. (2017). A REVIEW OF OPTIMIZATION MODELS FOR BOARDING A COMMERCIAL AIRPLANE.
Client Need	This method, being more efficient than other potential methods, saves the users time and computing resource during the simulation. With the cellular automata being a complicated algorithm that takes time, optimizations to it, even minor ones, make huge differences in user experience.

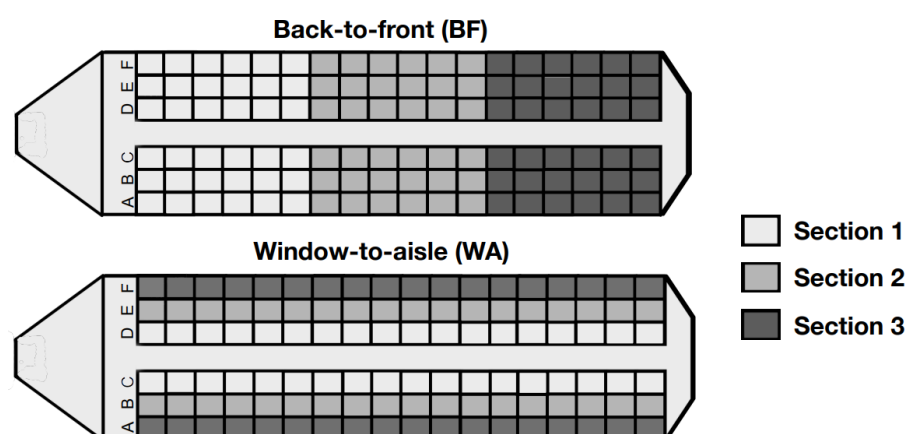
```

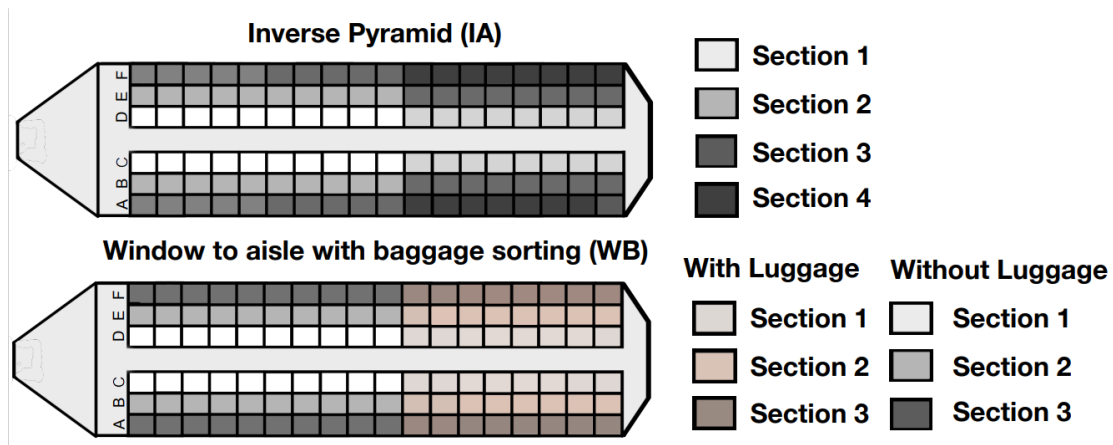
class outsidein():
    def __init__(self, disabled, purse, luggage):
        time = []
        board_avg = []
        wait_avg = []
        bag_avg = []
        stop_avg = []
        for i in range(1):
            p_list = self.create_passenger_list(disabled, purse, luggage)
            p_list = initialize_timer(p_list)
            narrow = [[0] * 8 for _ in range(32)]
            narrow = initialize_array(narrow)
            a, b, c, d, e = run_narrow(p_list, narrow, 'outsidein')
            time.append(a)
            board_avg.append(b)
            wait_avg.append(c)
            bag_avg.append(d)
            stop_avg.append(e)

```

The part highlighted in yellow is the initialization of the passenger list before-hand of the iterative simulation by the function `create_passenger_list()`, while the part highlighted in blue is the actual simulation, which references the `p_list` which is the generated, ordered passenger list. For each boarding method, there is a unique sorting process to represent the features of the boarding methods listed below.

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





Boarding by Section (back-to-front), BF. The aircraft is split into 3 categories. Take 18 rows of 4 for example: Every 6 rows will be categorized to section 1, section 2, and section 3 respectively. This method will line the passengers up prior to boarding according to their section, being section 3 goes first, then section 2 so on (note that the arrangements within the sections are random). The line will then proceed to board the cabin.

```
def create_passenger_list(self, disabled, purse, luggage):
    p_list = []
    for i in range(32 * 6):
        p_list.append(passenger(disabled, purse, luggage))
        p_list[i].aisle = i // 6
        p_list[i].seat = i % 6
    # p_list.reverse()
    # random.shuffle(p_list)
    temp = p_list[:64]
    random.shuffle(temp)
    p_list[:64] = temp
    temp = p_list[64:128]
    random.shuffle(temp)
    p_list[64:128] = temp
    temp = p_list[128:]
    random.shuffle(temp)
    p_list[128:] = temp
    # random.shuffle(p_list[64:128])
    # random.shuffle(p_list[128:])
    return p_list
    return p_list
```

Boarding by seat (Window to aisle), WA. The aircraft is split into 3 categories. Differing from the back to front boarding method, the sections will be split by their

respective column: window seats (Seat A and Seat F) will be categorized as section 3, the middle seats (Seat B and Seat E) will be categorized as section 2, and the aisle seats (Seat C and Seat D) will be categorized as section 1. Prior to boarding, the passengers will line up according to their section, with section 3 at the front of the queue and section 1 at the back. The line can then proceed to the cabin.

```
def create_passenger_list(self, disabled, purse, luggage):
    p_list = []
    for i in range(6):
        for j in range(32):
            p_list.append(passenger(disabled, purse, luggage))
            p_list[i * 32 + j].aisle = j
            p_list[i * 32 + j].seat = i
    p_list_2 = []
    p_list_2.extend(p_list[:32])
    p_list_2.extend(p_list[-32:])
    p_list_3 = []
    p_list_3.extend(p_list[32:64])
    p_list_3.extend(p_list[-64:-32])
    p_list_4 = []
    p_list_4.extend(p_list[64:96])
    p_list_4.extend(p_list[-96:-64])
    random.shuffle(p_list_2)
    random.shuffle(p_list_3)
    random.shuffle(p_list_4)
    p_list[:64] = p_list_2
    p_list[64:128] = p_list_3
    p_list[128:] = p_list_4
    p_list.reverse()
    return p_list
```

Inverse pyramid, IP. This boarding method is a hybrid between the Back-to-Front method and the Window-to-Aisle method.

- The back and middle seats by the window (rows 7-18 seats A and seats F) are the first group.
- The front seats by the window (rows 1-6 seats A and seats F) and the middle to back row seats in the middle column (row 7-18 seats B and seats E) the second group.
- Then, the third group is the front row seats in the middle column (rows 1-6 seats B and seats E) and the back rows in the aisle column (rows 13-18 seats C and seats D).

– The last group is the front and middle rows in the aisle column (rows 1-12 seats C and seats D). The passengers will line up accordingly and board according to their group number, noting that the order within the group is random.

```
def create_passenger_list(self, disabled, purse, luggage):
    p_list = []
    for i in range(4):
        for j in range(48):
            p_list.append(passenger(disabled, purse, luggage))
            if i == 0:
                if j % 2 == 0:
                    p_list[48 * i + j].seat = 0
                else:
                    p_list[48 * i + j].seat = 5
                p_list[48 * i + j].aisle = 31 - j // 2
            if i == 1:
                if j < 16:
                    if j % 2 == 0:
                        p_list[48 * i + j].seat = 0
                    else:
                        p_list[48 * i + j].seat = 5
                    p_list[48 * i + j].aisle = 31 - (j // 2 + 24)
                else:
                    if j % 2 == 0:
                        p_list[48 * i + j].seat = 1
                    else:
                        p_list[48 * i + j].seat = 4
                    p_list[48 * i + j].aisle = 31 - (j // 2 - 8)
            if i == 2:
                if j < 32:
                    if j % 2 == 0:
                        p_list[48 * i + j].seat = 1
                    else:
                        p_list[48 * i + j].seat = 4
                    p_list[48 * i + j].aisle = 31 - (j // 2 + 16)
                else:
                    if j % 2 == 0:
                        p_list[48 * i + j].seat = 2
                    else:
                        p_list[48 * i + j].seat = 3
                    p_list[48 * i + j].aisle = 31 - (j // 2 - 16)
            if i == 3:
                if j % 2 == 0:
                    p_list[48 * i + j].seat = 2
                else:
                    p_list[48 * i + j].seat = 3
                p_list[48 * i + j].aisle = 31 - (j // 2 + 8)
    p_list_1 = p_list[:48]
    p_list_2 = p_list[48:96]
    p_list_3 = p_list[96:144]
    p_list_4 = p_list[144:]
    random.shuffle(p_list_1)
    random.shuffle(p_list_2)
    random.shuffle(p_list_3)
    random.shuffle(p_list_4)
    p_list[:48] = p_list_1
    p_list[48:96] = p_list_2
    p_list[96:144] = p_list_3
    p_list[144:] = p_list_4
    p_list.reverse()
```

Window-to-Aisle with Baggage Sorting, WB. This method is an original method. As the results in Section 4.2 show that the Window-to-Aisle boarding method is the

most efficient, we build upon this method by adding a sorting method considering the luggage.

– People with more luggage will be arranged to sit at the back of the plane, and people

without luggage will be arranged to sit at the front.

– Then, passengers aboard the plane with window seats first, then middle seats, and finally aisle seats

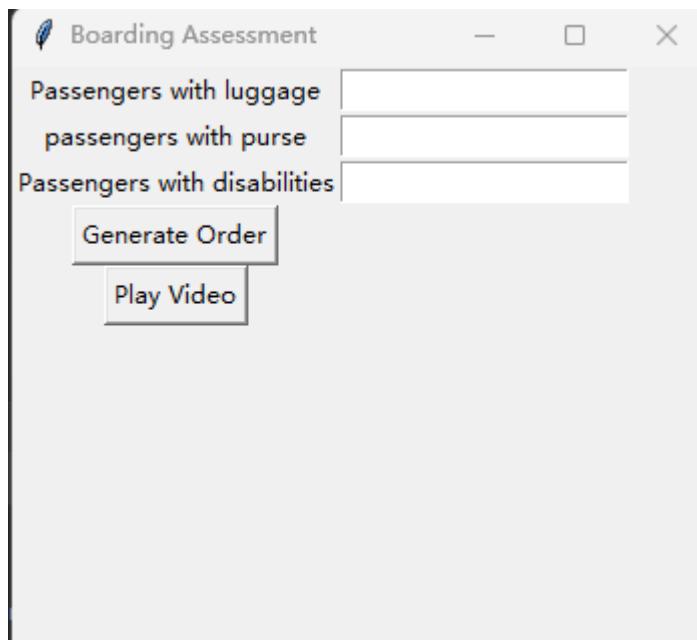
```
def create_passenger_list(self, disabled, purse, luggage):
    p_list = []
    for i in range(32 * 6):
        p_list.append(passenger(disabled, purse, luggage))
        p_list[i].aisle = i // 6
        p_list[i].seat = i % 6
    # p_list.reverse()
    # random.shuffle(p_list)
    temp = p_list[:64]
    random.shuffle(temp)
    p_list[:64] = temp
    temp = p_list[64:128]
    random.shuffle(temp)
    p_list[64:128] = temp
    temp = p_list[128:]
    random.shuffle(temp)
    p_list[128:] = temp
    # random.shuffle(p_list[64:128])
    # random.shuffle(p_list[128:])
    return p_list
```

5. Parameters for Cellular Automata

Function:	The Program allows for the input of parameters for the Cellular Automata simulation to account for different flights and passengers, which serves crucial to an accurate simulation.
Ingenuity	An important part of a simulation is to account for real life factors. The program ingeniously provides a place for the airline company, who has access to their consumer information, to enter parameters regarding areas like the number of passengers that are disabled and the number that holds a luggage or purse, and takes that into consideration. The simulation correspondingly slows the movement and the seating time for people like that, reflecting the reality in a boarding situation. Without it the simulation wouldn't be realistic.

Client Need	In a real-life scenario, passengers with luggage will need to stop to put their luggage in its corresponding storage, which takes a considerable time. People with disability need longer to travel down the hall way and get seated. The airline company's results will be biased and incorrect if these are not accounted, and results in higher chance of covid exposure if the wrong boarding methods is proposed.
-------------	--

Examples of Implementation:



The screenshot shows a software window titled "Boarding Assessment". It contains three text input fields for data entry: "Passengers with luggage", "passengers with purse", and "Passengers with disabilities". Below the input fields are two buttons: "Generate Order" and "Play Video".

-interface for entering parameters of Passengers with luggage, purse, and disabilities.

```

class reversepy():
    def __init__(self, disabled, purse, luggage):
        time = []
        board_avg = []
        wait_avg = []
        bag_avg = []
        stop_avg = []
        for i in range(1):
            p_list = self.create_passenger_list(disabled, purse, luggage)
            p_list = initialize_timer(p_list)
            narrow = [[0] * 8 for _ in range(32)]
            narrow = initialize_array(narrow)
            a, b, c, d, e = run_narrow(p_list, narrow, 'reversepy')
            time.append(a)
            board_avg.append(b)
            wait_avg.append(c)
            bag_avg.append(d)
            stop_avg.append(e)

```

Input from interface to simulation object initializer

From initializer parameters create passenger list based on the parameters

-The parameter values from the interface are passed to the simulation object, which then corresponds to variable values in mathematical calculations that models time taken to board in equation: $T_i = (2 + L_i + B_i + S_i) \cdot (1 + D_i)$.

```

class passenger(object):
    def __init__(self, disabled, purse, luggage):
        self.aisle = 0
        self.seat = 0
        self.speed = 99 - 50*pr.Prob(disabled)
        self.purse = pr.Prob(purse)
        self.carry = pr.Prob(luggage)

```

Passenger Objects are created with regard to the parameters, in how the proportion of disable people on the plane affects the probability one a passenger object to be stabled, which directly affects the speed of the person, while proportions with pursers and luggage affects the time one need to take a seat, like below:

```

def initialize_timer(p_list):
    for i in range(len(p_list)):
        p_list[i].timer += int(np.random.weibull(2)*(3*p_list[i].purse + 5*p_list[i].carry))
    return p_list

```

6. Visualization of Simulation using MoviePy

Function:	Using the python library MoviePy, the cellular automata is visualized for the client to see, effectively and quickly.
Ingenuity	During the Cellular automata algorithm, the frames for the simulation are captured and combined into an array of images, which is then combined into a playable video for the user to see the best boarding order. It is ingenious because it is the only way for the user to be accurately informed of the optimal boarding order. Simple feedback of words like Reverse pyramid o back to front may be confusing for inexperienced users. A video provides a visual approach for all kinds of users to understand.
Source	<p><i>Example Scripts — MoviePy 1.0.2 documentation.</i> (n.d.). Zulko.github.io.</p> <p>Retrieved August 5, 2022, from</p> <p>https://zulko.github.io/moviepy/examples/examples.html</p> <p>J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007.</p>
Client Need	It provides a simple to use feedback method for the Cellular Automata that visualize to the audience by click of a button. The property of it being simple to use and understand provide the user a great experience with the visualization function of the app, as well as being accurately informed of the optimal boarding method.

```

if time % 6 == 0 or sit > 191:
    plt.show()
    a += 1
    plt.close()
    fig, ax = plt.subplots()
    plt.imshow(viewer, cmap='summer', vmin=0, vmax=10)
    plt.savefig('C:\\Users\\Eric Chang\\PycharmProjects\\IA\\'+s+'\\' + str(a/100) + '.png')

```

The program captures every 6 frame of the simulation and use *matplotlib* to turn the 2D array used in simulation to a .png image, which for every boarding methods is saved in their respective folders, signified by the variable *s*.

```

def function():
    a=int(d)/192 #computed to probability
    b=int(e)/192
    c=int(f)/192
    data = np.array([our(a,b,c).returnTime(), backfront(a,b,c).returnTime(), reversepy(a,b,c).returnTime(), outsidein(a,b,c).returnTime()]) #calls all boarding methods
    index = 0
    print (data)
    for i in range(1, 4): #comparison of speed
        if data[i] < data[index]:
            index = i

```

After the calling of the simulation, the results for the 4 boarding methods are computed into an array, which is compared by a linear search for the least time spent on simulation.

```
if i==1: #compute the fastest method
    arr = os.listdir('C:\\Users\\Eric Chang\\PycharmProjects\\IA\\our')
    print (arr)
    a=[]
    for i in arr:
        a.append('C:\\Users\\Eric Chang\\PycharmProjects\\IA\\our\\'+i)
if i==2:
    arr = os.listdir('C:\\Users\\Eric Chang\\PycharmProjects\\IA\\outsidein')
    print (arr)
    a=[]
    for i in arr:
        a.append('C:\\Users\\Eric Chang\\PycharmProjects\\IA\\outsidein\\'+i)
if i==3:
    arr = os.listdir('C:\\Users\\Eric Chang\\PycharmProjects\\IA\\reversepy')
    print (arr)
    a=[]
    for i in arr:
        a.append('C:\\Users\\Eric Chang\\PycharmProjects\\IA\\reversepy\\'+i)
if i==0:
    arr = os.listdir('C:\\Users\\Eric Chang\\PycharmProjects\\IA\\backfront')
    print (arr)
    a=[]
    for i in arr:
        a.append('C:\\Users\\Eric Chang\\PycharmProjects\\IA\\backfront\\'+i)
a=sorted(a)
print (a)
clip = ImageSequenceClip(a, fps=3)
clip.write_videofile("C:\\Users\\Eric Chang\\PycharmProjects\\IA\\video.mp4", fps=5)
```

The most optimal boarding method's folder is selected, and its component .png images are made into a list of directories, which is sorted by ascending order and combined in the ImageSequenceClip class object from MoviePy. They using a static function of the ImageSequenceClip class, it is exported as a video file, which can be played by the click of a button (highlighted in blue text).

```

@staticmethod
def video():
    os.startfile("C:\\Users\\Eric Chang\\PycharmProjects\\IA\\video.mp4")

@staticmethod
def cell():
    window = tk.Tk()
    window.title('Boarding Assessment')
    window.geometry('350x300')
    window.grid()
    font1 = tkFont.Font(family="Helvetica", weight="bold")
    title = tk.Label(window, text='Boarding Method \nGenerator', font=font1)
    label1 = tk.Label(window, text='Passengers with luggage')
    label2 = tk.Label(window, text='passengers with purse')
    label3 = tk.Label(window, text='Passengers with disabilities')

    entry1 = tk.Entry(window)
    entry2 = tk.Entry(window)
    entry3 = tk.Entry(window)
    d, e, f = entry1.get(), entry2.get(), entry3.get()
    button = tk.Button(window, command=CA.function, text="Generate Order")
    button1 = tk.Button(window, command=CA.video, text="Play Video")

```

Word Count: 1150