Model Documentations
Airport Arrivals in a Post-Pandemic World

Vincent Leonardo
Sarah Wong I-Mae
Sarah Ramjoo

December 12, 2021

Overview

Dependencies

- Internet connection for the different libraries imported by the model.
- A modern browser updated within this year (2021) is preferred due to some newer CSS dependencies. This model is also compatible with mobile browsers.
- This model can also be accessed at `https://github.vinleonardo.com/sma-airport-model/`.
- The webpage version of this documentation can be accessed at `https://github.vinleonardo.com/sma-airport-model/documentations`

Objectives of the model

- The objective of this simulation is to bring a passenger from the the plane (left border) and out of the system.
- There are two ways of exiting: one is through the normal way of clearing the defined stations, the other is when, if there is a testing station, the passenger tested positive for COVID-19.
- The behaviour of the passengers is that they will all walk at the same speed without social distancing, always moving forward, and selection of queues will be done either randomly or in choosing the shortest queue (user-set probability). Moreover, for COVID-19 testings, everything is assumed to be 100% accurate.

Immigration station

Figure: Immigration station

- The immigration station is comprised of pair-wise queue lanes and booths.
- The behaviour is a simple one in that passengers will queue and wait for their turn to go next into the booth.

Testing station

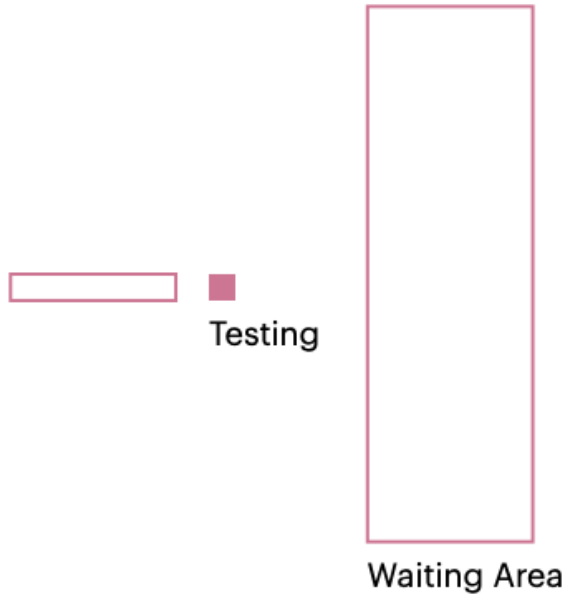


Figure: Testing station

- The testing station is similar to the immigration station.
- But, after the booth, the passengers go into a waiting area where they randomly choose their sitting positions to wait for a set amount of time.

Baggage station

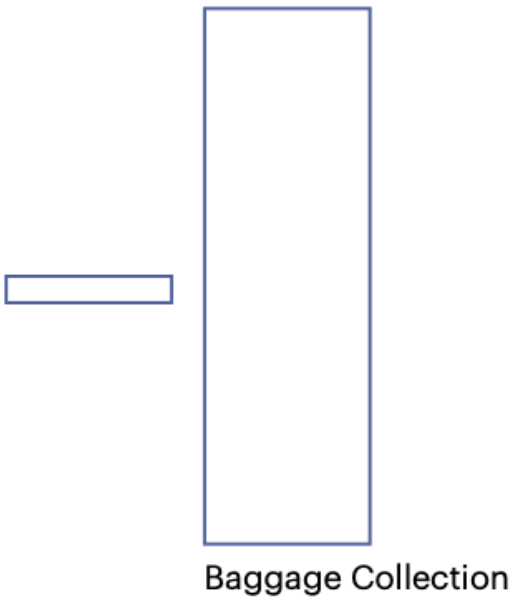


Figure: Baggage station

- The baggage area is simple. It contains a lane to enter in (which is not a queue) and a baggage area afterwards.
- This is, of course, assuming that the baggages are not on carousels as per normal, but left down on the ground due to the special circumstances.
- The passenger will try to look for their baggages within this area and then leave.

UI elements - before simulation box

The lines before the simulation box denote the changing variables during the simulation. This includes the time within the current simulation, the number of exited passengers within the current simulation, and the average time taken per passenger within the current simulation along with the average time taken to finish one simulation and its standard deviation over the set amount of simulations.

Current time is 0 seconds or 0.00 minutes or 0.00 hours.

Number of exited passengers: 0.

Average time taken by a passenger: 0. Average time taken for 150 passenger(s): 0.00 over 0 simulation(s) with a standard deviation of 0.00.

Figure: What is seen before the simulation box

Before setting any stations, the simulation box will be an empty box with a black border.

UI elements - after simulation box

Below the simulation box, there are two graphs: one to track the average time taken per passenger within one simulation run, and the other one to track the average time taken to finish one simulation run.

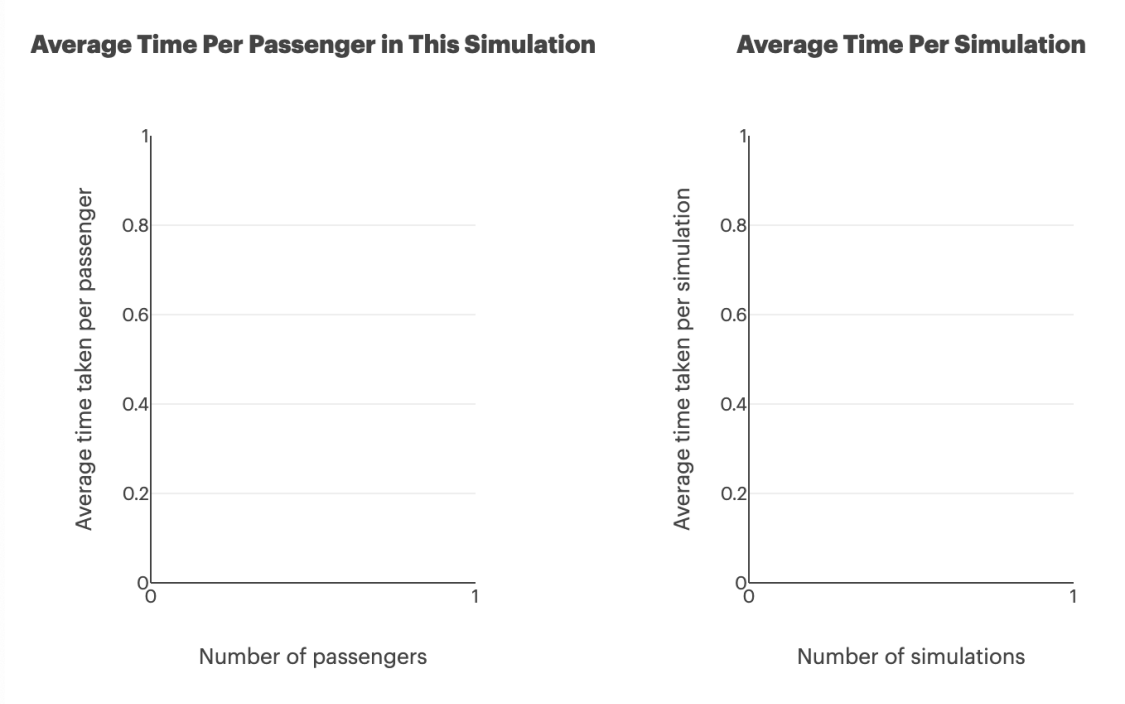


Figure: What is seen after the simulation box

UI elements - after simulation box

Right below that, there are control buttons to start, pause, and reset the simulations. It is also possible to download the data collected for the time taken per simulation over one or multiple simulations as a CSV file.

Figure: What is seen after the simulation box

Inputs

There are three different sections covering different aspects of the simulation.

- One is Stations, which covers both the layout of the stations within the simulation box and the global simulation settings.
- The next one is Station Parameters, which is used to customise how many booths or lanes each station has.
- The final one is Agent Parameters, which control the agents' behaviours and time taken to go through each station.

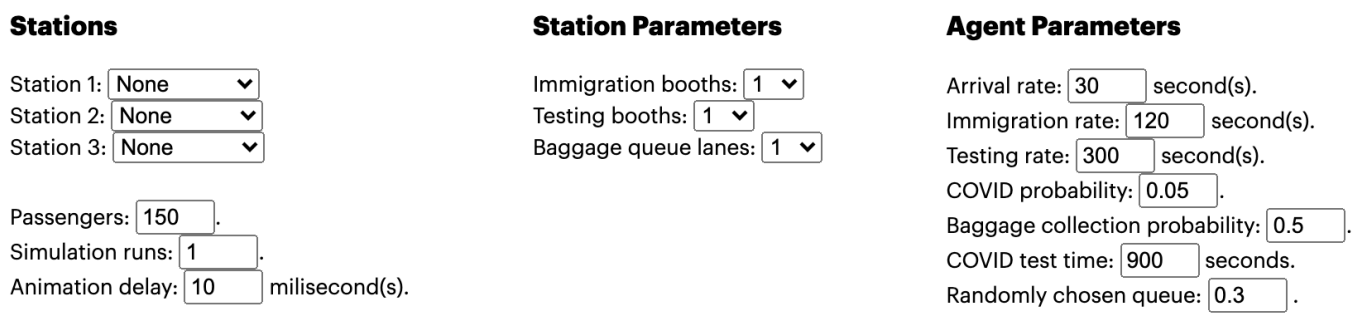


Figure: Inputs

Inputs - Stations

There must be at least one choice of station for the simulation to run. The station numbers correspond to their position on the simulation box from left to right. For example, the picture below shows when Station 1 is Immigration, Station 2 is Testing, and Station 3 is Baggage.

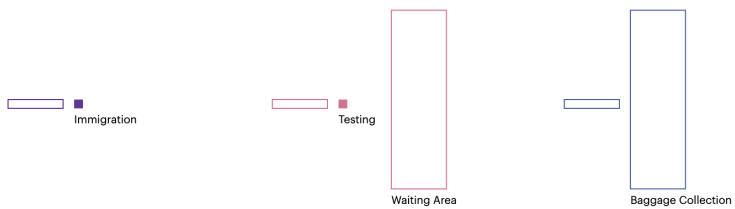
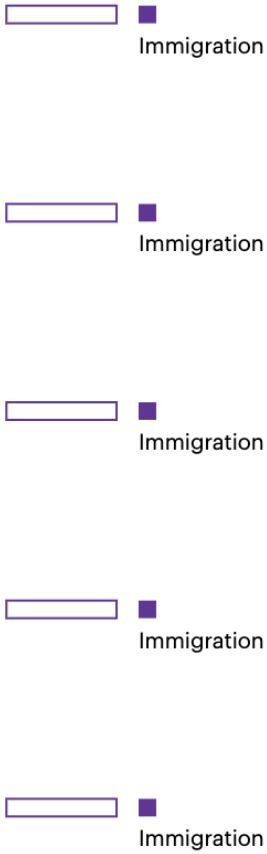


Figure: Example of model set

Inputs - Station Parameters



The dropdowns denote the amount (from one to ten) of booths, queues, or lanes provided in the simulation for each station. An example for when the immigration booth is set to have five booths is as follows.

Figure: Multiple immigration booths

Inputs - Agent Parameters

The rates that are denoted in seconds will be converted into their per-second probability (its inverse), excepting the COVID test time, which is a definite time delay. This meant that the probability, for example, of an arrival rate of 30 seconds will be 1/30 per second.

- Arrival rate: the mean time taken for each passenger to enter the simulation.
- Immigration rate: the mean time to clear immigration for each passenger
- Testing rate: the mean time to clear the testing booth.
- COVID probability: the probability of each passenger having COVID.
- Baggage collection probability: the probability that when a passenger reaches the baggage area, their baggage is found.
- COVID test time: the time taken to clear the COVID test.
- Randomly chosen queue: the probability that a passenger will deviate from the rational queue choice (shortest queue).

Behaviours and code

This is going to be a high level description of the model, written in the perception of an individual agent. The animations are done using the D3.js library.

```
function simStep() {
  if (isRunning) {
   // Chunk of code was here, deleted for clarity.

    if (enteredPassengers < passengerCount) {
      removeDynamicAgents();
      addDynamicAgents();
      updateDynamicAgents();
    } else if (exitedPassengers < passengerCount) {
      removeDynamicAgents();
      updateDynamicAgents();
    } else {
      /*
      Chunk of code was here, deleted for clarity.
      Mainly to stop the simulation and to re-run if multiple simulations
      were configured.
      */
    }
}
```

@vincentleoooo at thiscodeWorks.com

Figure: High level overview

Behaviours and code

The code shown in the last slide shows the process for each second passing. Firstly, it is the garbage collection. This means that it will try to clear out whichever agent is at the going out state. Afterwards, it will create new passengers based on the rates set by the user as long as the number of passengers does not exceed the specified amount of passengers. It will then update the positions of all the agents in the simulation based on their target locations.

Removing passengers

In removing the passengers, it deletes both the passengers data and the data associated with the animation.

```
function removeDynamicAgents() {
  let allPassengers = surface.selectAll(".passenger").data(objects.passengers);
  let exitingPassengers = allPassengers.filter(function (d) {
    return d.state == "out";
  });
  exitingPassengers.remove();
  objects.passengers = objects.passengers.filter(function (d) {
    return d.state != "out";
  });
}
```

@vincentleoooo at thiscodeWorks.com

Figure: Removing passengers

Adding passengers

- In this case, it will follow a process of finding the next station.
- It will either choose the queue lane for the next station rationally based on queue lengths or based on random choice. This parameter is set by the user.
- Afterwards, other attributes are assigned to keep track of its state. The user's choice for COVID infected probability will affect how many passengers have COVID.
- Code snippet can be seen at `https://www.thiscodeworks.com/61b4b094fe9bbd0015d5e7f3` since it is too long to be seen here.

Updating passengers

- In updating the passenger, there will be roughly two different states. One is when the passenger is still on the way to a target, and one where the passenger has reached a target. Depending on the station, the passenger will do different things.
- Code snippet can be seen at `https://www.thiscodeworks.com/61b4b185fe9bbd0015d5e7f4` since it is too long to be seen here.

Updating passengers

- At the immigration booth, the passenger will queue up until it is their turn. After coming out of the booth, it will select the next station. At the testing booth, the passenger will also queue up to get tested. Afterwards, it is a free seating environment within the area used to wait for test results. At the baggage collection area, the passenger will go through the lanes into a space where the baggages are placed. They will then try to find their baggages. This probability is set by the user. If the next station is to go out, they will then proceed to go to the exit node.
- For COVID patients who are screened during testing, they will go out via a different path, thus reducing their time taken to go through the simulation.

Area creation and modularity

- JavaScript objects
- Area creation
- Tracking states and substates

JavaScript objects

The example in the next slide shows the different ways to get keys or values within an object. This is how the model was able to find the positions of the stations based on what is set by the user and to connect the pieces together such that the passenger can find the next station and hence the queue. The object keys associated with queues are always called `{station} + Queue`. This leads to the stations behaving like Lego blocks, being able to connect with any other station with a standard input/output.

JavaScript objects

```
var positions = {
  arrivals: 0,
  immigration: 0,
  testing: 0,
  baggage: 0,
  exiting: 5,
  undefined: 0,
};

var objects = {
  immigration: [],
  immigrationQueue: [],
  passengers: [],
  testing: [],
  testingBox: [],
  testingQueue: [],
  baggage: [],
  baggageQueue: [],
};

// Get the key of a certain value by going through
// the whole object.
function getKeyByValue(object, value) {
  return Object.keys(object).find((key) => object[key] === value);
}

// How to access values through keys
objects.immigration == []; // true
let newString = immigration;
objects[newString] == []; // true
```


Figure: JavaScript hacks

Area creation

The creation of the areas will depend on whether the positions of the stations are defined (>0 in this case). Depending on the number of queues or booths set, the `objects` object will be modified where the stations' booths or queues will be added in such that it can be generated later on. On the next slide is an example for the immigration booth.

Area creation

```
function createImmigrationOfficers() {
  if (positions.immigration > 0) {
    for (let i = 1; i <= numImmigrationOfficers; i++) {
      let newRow = (height * i) / (numImmigrationOfficers + 1);
      newRow = newRow.toFixed(0);

      let newCol = (width * (positions.immigration - 1)) / (numStations + 1);
      newCol = newCol.toFixed(0);

      let newImmigrationOfficer = {
        id: i,
        type: ICAOFFICER,
        label: "ICAOfficer",
        row: newRow,
        col: newCol,
        state: "IDLE",
      };

      objects.immigration.push(newImmigrationOfficer);
    }
  } else {
    objects.immigration = [];
  }
}

function createImmigrationQueues() {
  if (positions.immigration > 0) {
    for (let i = 1; i <= numImmigrationOfficers; i++) {
      let newRow = (height * i) / (numImmigrationOfficers + 1);
      newRow = newRow.toFixed(0);

      let newCol = (width * (positions.immigration - 1)) / (numStations + 1);
      newCol -= 60;
      newCol = newCol.toFixed(0);

      let newImmigrationQueue = {
        id: i,
        row: newRow,
        col: newCol,
        state: IDLE,
        stack: 0,
      };

      objects.immigrationQueue.push(newImmigrationQueue);
    }
  } else {
    objects.immigrationQueue = [];
  }
}
```

@vincentleoooo at thiscodeWorks.com

Figure: Area creation

Tracking states and substates

The passengers have a `state` to denote which station to go, and a `queueState` to denote which part of each station to go to. They also have the `station` integer key to denote which part of the simulation box they are in, and as such they can track their progress and go find which station is in the next part using the tricks mentioned above.

Front-end niceties

- Inputs and value-safety
- Plots
- Downloads

Inputs and value-safety

These are some of the functions that help to key in inputs from the front-end and to ensure that they are tenable.

```
// One of the functions that react to the dropdown menus being changed and
// to prevent duplicate stations.
function onChange1() {
  if (isRunning || isPaused) {
    alert(
      "Please reset the model first before changing this parameter. Changing the parameter will reset
the simulation."
    );
  } else {
    var choice1 = document.getElementsByName("choices1")[0].value;
    if (choice1 == "none") {
      let position2 = getKeyByValue(positions, 2);
      positions[position2] = 0;
    } else {
      let position2 = getKeyByValue(positions, 2);
      let position3 = getKeyByValue(positions, 3);
      let position4 = getKeyByValue(positions, 4);
      if (choice1 == position3 || choice1 == position4) {
        alert("Duplicates found. Please choose a different station.");
        let select = document.querySelector("#select1");
        select.value = "none";
      } else {
        positions[choice1] = 2;
      }
      positions[position2] = 0;
    }
    redrawWindow();
  }
}

// Function to track changes in the input value boxes.
function changeProb() {
  if (isRunning || isPaused) {
    alert(
      "Please reset the model first before changing this parameter. Changing the parameter will reset
the simulation."
    );
  } else {
    probArrival = 1 / Number(document.getElementById("probArrival").value);
    probImmigration =
      1 / Number(document.getElementById("probImmigration").value);
    probTesting = 1 / Number(document.getElementById("probTesting").value);
    probCovid = Number(document.getElementById("probCOVID").value);
    testingTime = Number(document.getElementById("timeTest").value);
    probFindBaggage = Number(document.getElementById("probBaggage").value);
    animationDelay = Number(document.getElementById("animationDelay").value);
    passengerCount = Number(document.getElementById("passengerCount").value);
    simulationRuns = Number(document.getElementById("simulationRuns").value);
    randomChosenQueue = Number(
      document.getElementById("randomChosenQueue").value
    );
  }
}

// Value safety for the input boxes
function runSim() {
  let choice1 = document.getElementsByName("choices1")[0].value;
  let choice2 = document.getElementsByName("choices2")[0].value;
  let choice3 = document.getElementsByName("choices3")[0].value;
  isPaused = false;
  changeProb();

  if (choice1 == "none" && choice2 == "none" && choice3 == "none") {
    alert("Please make at least one station choice.");
  } else if (isRunning) {
    alert("Already running.");
  } else if (
    isNaN(probArrival) ||
    isNaN(probCovid) ||
    isNaN(probImmigration) ||
    isNaN(probTesting) ||
    isNaN(probFindBaggage) ||
    isNaN(testingTime) ||
    isNaN(animationDelay) ||
    isNaN(passengerCount) ||
    isNaN(simulationRuns) ||
    isNaN(randomChosenQueue)
  ) {
    alert("At least one of the inputs is not a number.");
  } else {
    isRunning = true;
    simTimer = window.setInterval(simStep, animationDelay);
  }
}
```

@vincentleoooo at thiscodeWorks.com

Figure: Front-end inputs framework

Plots

The plots are made using Plotly.

```
// set global variables
const limit = 10000; // How many points can be on the graph before sliding occurs
const refreshInterval = 100; // Time between refresh intervals

// set functions to retrieve
function getData1() {
  if (listMeanTimeToClear.length > 0) {
    return listMeanTimeToClear[listMeanTimeToClear.length - 1];
  } else {
    return 0;
  }
}

// set chart layout
const layout1 = {
  title: {
    text: "<b>Average Time Per Passenger in This Simulation</b>",
  },
  paper_bgcolor: "rgba(0,0,0,0)",
  plot_bgcolor: "rgba(0,0,0,0)",
  xaxis: { title: "Number of passengers", rangemode: "tozero" },
  yaxis: { title: "Average time taken per passenger (s)", rangemode: "tozero" },
  font: { family: "Graphik", size: 11 },
};

// plot all charts
Plotly.newPlot(
  "tester",
  [
    {
      y: [getData1()],
      mode: "lines",
      line: {
        color: "rebeccapurple",
        width: 1,
      },
    },
  ],
  layout1,
  { responsive: true }
);

// set refresh interval and graph limit
var cnt = 0;

// to extend plot
if (isRunning == true) {
  Plotly.extendTraces("tester", { y: [[getData1()]] }, [0]);
  cnt++;
  if (cnt > limit) {
    Plotly.relayout("tester", {
      xaxis: {
        range: [cnt - limit, cnt],
      },
    });
  }
}
```


Figure: Plots code

Downloads

The downloading function takes in the CSV string and converts it to be downloaded as a proper CSV file.

```
function download(content, filename, contentType) {
  if (!contentType) contentType = "application/octet-stream";
  var a = document.createElement("a");
  var blob = new Blob([content], { type: contentType });
  a.href = window.URL.createObjectURL(blob);
  a.download = filename;
  a.click();
}

download(csvContent, 'listSimulationRunTimes.csv', 'text/csv');
```

@vincentleoooo at thiscodeWorks.com

Figure: Downloads code