FLIGHT BOARDING SIMULATION

Behind the Scenes

AGENDA

▶ *BASIC MODEL ARCHITECTURE*

▶ DECISION RULES

▶ DEEP DIVE INTO MODELS

INITIALIZE

Time Delays

```
// TIME TAKEN TO GET TO SEAT
// if middle seater, aisle seat occupied
var midmovetime = 3;
// window seater, aisle seat occupied
var windowmovetime1 = 4;
// window seater, middle seat occupied
var windowmovetime2 = 5;
// window seater, aisle + middle occupied
var windowmovetime3 = 6;

// TIME TO STOW BAGGAGE
var bagtime = 8;
```

Key Variables & Links to Images

```
var WINDOWBORDERSIZE = 10; // green border
var animationDelay = 1; //controls simulation and transition speed
var isRunning = false; // for simStep() and toggleSimStep()
var surface; // Used in D3 selection of svg drawing surface
var simTimer; // initialize timer
var seatFull = 0; // increment everytime a seat is occupied
var seatFullCheck = [0,0]; // decide when to terminate simulation
var n = 500; // number of simulation runs
var seatsOccupied = {}; // switch states in updatePassenger()

// IMAGES
const urlChair = "img/Chair-icon.png";
const urlPassenger = "img/passenger-icon.png";
const windowpic = "img/window-new.png"
```

Passenger States

```
// STATES
const UNSEATED=0; // walk to desired aisle
const WAITING=1; // in desired aisle
const SEATED=2; // seated
```

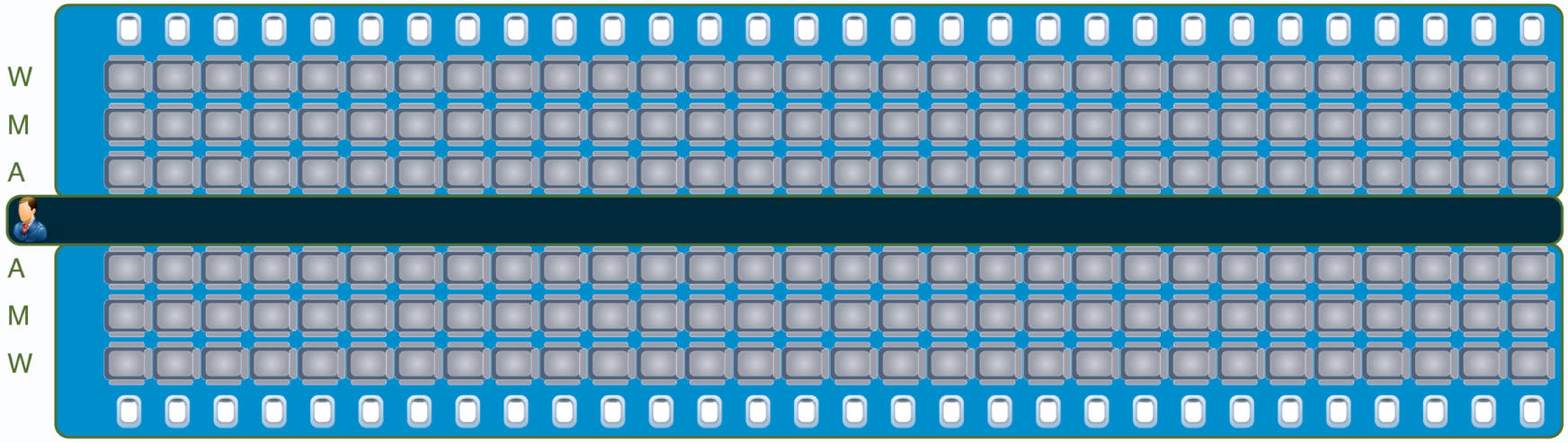
Cell Size

```
//CELL SIZE
var maxCols = 40;
var cellWidth; // initialize for Reload()
var cellHeight; // initialize for Reload()
```

D3.JS CANVAS

1 *Surface*

2 *Plane Body*



3 *Windows & Seats*

Current Run Time: 0.0
Grand Total Time: 0.0
Average Run Time: 0.0
Number of Runs: 0.0

4 *Statistics*

Random Boarding

Animation Speed:

passenger icon from icons-land.com

1 DRAW SURFACE

```
<body class="official bgColor6">

<svg id="surface" style="width:100%; height:100%" onclick="toggleSimStep();">
</svg>
<div id="title">Random Boarding</div>

<div id="controls">Animation Speed:
<input id="slider1" type="range" min="0" value:"275" max="500"
step = "10" onchange="adjust();" />
</div>

<a id="credits" href="http://www.icons-land.com">
passenger icon from icons-land.com</a>

<script type="text/javascript" src="lib/FlightModel.js"></script>

</body>
```

- *Define surface as SVG element in HTML*
- *Animation Slider*
- *Link to JavaScript File*

1 DRAW SURFACE

```
var drawsurface = document.getElementById("surface");
var creditselement = document.getElementById("credits");
var w = window.innerWidth; //width of entire window
var h = window.innerHeight; //height of entire window
var surfaceWidth =(w - WINDOWBORDERSIZE);
var surfaceHeight= (h-creditselement.offsetHeight - 3*WINDOWBORDERSIZE);

drawsurface.style.width = surfaceWidth+"px";
drawsurface.style.height = surfaceHeight+"px";
drawsurface.style.left = WINDOWBORDERSIZE/2+'px';
drawsurface.style.top = WINDOWBORDERSIZE/2+'px';
drawsurface.style.border = "thick solid rgba(80,109,47,0.5)";
drawsurface.innerHTML = ''; // Empties contents of drawing surface

numCols = maxCols;
cellWidth = surfaceWidth/numCols;
numRows = Math.ceil(surfaceHeight/cellWidth);
cellHeight = surfaceHeight/numRows;

surface = d3.select('#surface');
```

Surface DOM selection

- *Defining width, height*
- *Green Border around surface*

Defining cell size (crucial for d3 element placement!)

IMPORTANT : d3 selection to 'paint' on canvas

1 DRAW SURFACE

Result: an empty canvas for all the action to take place

② INSERT PLANE BODY

```
var planeArea = [
 {"label":"Right Wing","startRow":7,"numRows":4,
 "startCol":5,"numCols":31,"color":'#008DCB'},
 {"label":"Left Wing","startRow":12,"numRows":4,
 "startCol":5,"numCols":31,"color":"#008DCB"},
 {"label":"Aisle","startRow":11,"numRows":1,
 "startCol":4,"numCols":32,"color":"#00293c"}];

var selectSeats = surface.selectAll(".planeArea").data(planeArea);
var seatArea = selectSeats.enter().append("g").attr("class","planeArea");
// For each new area, append a rectangle to the group
seatArea.append("rect")
.attr("x", function(d){return (d.startCol-1)*cellWidth;})
.attr("y", function(d){return (d.startRow-1)*cellHeight;})
.attr("width", function(d){return d.numCols*cellWidth+5;})
.attr("height", function(d){return d.numRows*cellWidth;})
.style("fill", function(d) {return d.color;})
.style("stroke","#506D2F")
.style("stroke-width",2)
.attr("rx", 10)
.attr("ry", 10);
```

Define plane body co-ordinates

- *Select all svg elements of class "planeArea" & map to planeArea list data*
- *Create svg group "g" of class "planeArea" for each new entry*
- *Append elements to surface*

② INSERT PLANE BODY

Result: Aisle, Left Wing, Right Wing

3 SEATS & WINDOWS

Define Seats & Aisle

```
var left_wing = [];
var right_wing = [];
var aisle = [];
var x = 0;
var y = 0;
var z = 0;

for (var j = 8; j < 11; j++) {
 for (var i = 6; i < 36; i++) {
  left_wing[x] = {"location": {"row":j,"col":i}, "id":x+1, "occupied": 0};
  x += 1
 };
};

for (var j = 12; j < 15; j++) {
 for (var i = 6; i < 36; i++) {
  right_wing[y] = {"location": {"row":j,"col":i}, "id":y+91, "occupied": 0};
  y += 1
 };
};

for (var i = 4; i < 36; i++){
 aisle[z] = {"location": {"row":11,"col":i}, "occupied": 0}
 z+=1
};

var allseats = left_wing.concat(right_wing);
```

Define Windows

```
var a = 0;
var b = 0;
left_window = [];
right_window = [];

for (var i = 6; i < 36; i++) {
  left_window[a] = {"location": {"row":7,"col":i}};
  a += 1
};

for (var i = 6; i < 36; i++) {
  right_window[b] = {"location": {"row":15,"col":i}};
  b += 1
};

var allwindow = left_window.concat(right_window);
```

Define Seat Labels

```
var seatLabel = [
  {"name":"W","location":{"row":8,"col":3}},
  {"name":"M","location":{"row":9,"col":3}},
  {"name":"A","location":{"row":10,"col":3}},
  {"name":"A","location":{"row":12,"col":3}},
  {"name":"M","location":{"row":13,"col":3}},
  {"name":"W","location":{"row":14,"col":3}}
];
```

3 SEATS & WINDOWS

```
var chair = surface.selectAll(".sittingarea").data(allseats);
var chairplace = chair.enter().append("g").attr("class", "sittingarea");

chairplace.append("svg:image")
.attr("x", function(d){var cell= getLocationCell(d.location); return cell.x+"px";})
.attr("y", function(d){var cell= getLocationCell(d.location); return cell.y+"px";})
.attr("width", Math.min(cellWidth,cellHeight)+"px")
.attr("height", Math.min(cellWidth,cellHeight)+"px")
.attr("xlink:href", function(d){return urlChair;});

var allwindowimg = surface.selectAll(".allwindow").data(allwindow);
var allwindowimg1 = allwindowimg.enter().append("g").attr("class", "allwindow");

allwindowimg1.append("svg:image")
.attr("x", function(d){var cell= getLocationCell(d.location); return cell.x+"px";})
.attr("y", function(d){var cell= getLocationCell(d.location); return cell.y+"px";})
.attr("width", Math.min(cellWidth,cellHeight)+"px")
.attr("height", Math.min(cellWidth,cellHeight)+"px")
.attr("xlink:href", function(d){return windowpic;});

var seatLabelpos = surface.selectAll(".seatLabel").data(seatLabel);
var seatLabelpos1 = seatLabelpos.enter().append("g").attr("class","seatLabel");

seatLabelpos1.append("text")
  .attr("x", function(d) { var cell= getLocationCell(d.location);
   return (cell.x+cellWidth)+"px"; })
  .attr("y", function(d) { var cell= getLocationCell(d.location);
   return (cell.y+cellHeight/2)+"px"; })
  .attr("dy", ".35em").attr("font-size", "20px")
  .attr("font-family", "Helvetica").attr("fill", "#506D2F")
  .style("text-align", "center")
  .text(function(d) {return d.name;});
```

Append SEATS

Append WINDOWS

Append SEAT LABELS

3 SEATS & WINDOWS

Let's add passengers while we're at it

```
var passengers = [];

var allpassengers = surface.selectAll(".passenger").data(passengers);
var newpassengers = allpassengers.enter().append("g").attr("class","passenger");

newpassengers.append("svg:image")
 .attr("x",function(d){var cell= getLocationCell(d.location); return cell.x+"px";})
 .attr("y",function(d){var cell= getLocationCell(d.location); return cell.y+"px";})
 .attr("width", Math.min(cellWidth,cellHeight)+"px")
 .attr("height", Math.min(cellWidth,cellHeight)+"px")
 .attr("xlink:href",function(d){return urlPassenger});

 //Select the image elements in the 'passengers' list
 var images = allpassengers.selectAll("image");

 images.transition()
  .attr("x",function(d){var cell= getLocationCell(d.location); return cell.x+"px";})
  .attr("y",function(d){var cell= getLocationCell(d.location); return cell.y+"px";})
  .duration(animationDelay).ease('linear'); // speed and type of transition
```

Passengers will be added to this list in addPassengers() function

Define passenger transition

4 STATISTICS

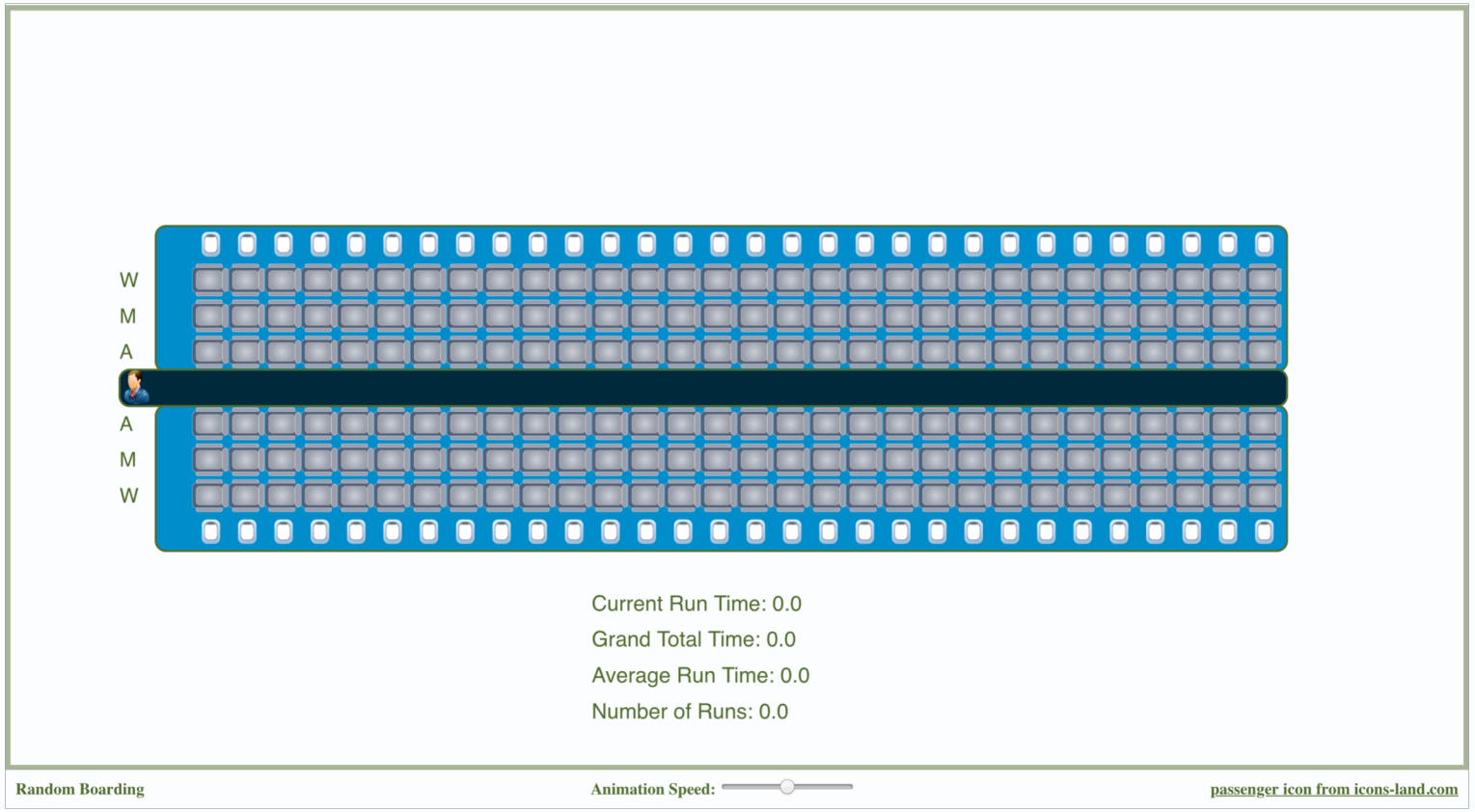
```
var statistics = [
{"name":"Current Run Time: ","location":{"row":17,"col":16},
"cumulativeValue":0,"count":0},

{"name":"Grand Total Time: ","location":{"row":18,"col":16},
"cumulativeValue":0,"count":0},

{"name":"Average Run Time: ","location":{"row":19,"col":16},
"cumulativeValue":0,"count":0},

{"name":"Number of Runs: ","location":{"row":20,"col":16},
"cumulativeValue":0,"count":0}];

var allstatistics = surface.selectAll(".statistics").data(statistics);
var newstatistics = allstatistics.enter().append("g").attr("class","statistics");
// For each new statistic group created we append a text label
newstatistics.append("text")
  .attr("x", function(d) { var cell= getLocationCell(d.location);
    return (cell.x+cellWidth)+"px"; })
  .attr("y", function(d) { var cell= getLocationCell(d.location);
    return (cell.y+cellHeight/2)+"px"; })
  .attr("dy", ".35em")
  .attr("font-size", "21px").attr("font-family", "Helvetica")
  .attr("fill", "#506D2F").text("");

allstatistics.selectAll("text").text(function(d) {
  return d.name+d.cumulativeValue.toFixed(1); });
};
```

Define Statistics

Append to surface

3 SEATS & WINDOWS
4 STATISTICS

Final Result



All d3.js code resides in updateSurface() !

BASIC SIMULATION STRUCTURE

simStep executes one simulation step

```
function simStep(){
  if (isRunning){ // toggled to pause Simulation when user clicks on surface
 1 addPassengers(); // Add passengers to surface
 2 updateDynamicAgents(); // Update passengers' location on surface
 3 isCompleted(); // Controls N simulation runs and updates statistics
    seatFullCheck.push(seatFull) // List of occupied seats
    statistics[0].cumulativeValue+=2.27; // Increment current time
    statistics[1].cumulativeValue+=2.27; // Increment grand total time
  }};
```

Initialization function sets a timer to call simStep

```
(function() {
 // call the function simStep every animationDelay milliseconds
 simTimer = window.setInterval(simStep, animationDelay);
 Reload();
})();
```


AGENDA

▶ BASIC MODEL ARCHITECTURE

▶ *DECISION RULES*

▶ DEEP DIVE INTO MODELS

HOW ARE PASSENGERS ADDED?

1 ADDPASSENGERS()

```
function addPassengers(){

 var luggage = 0;
 if (Math.random()> 0.5) {luggage = 1;}

 if (allseats.length>0 && aisle[0].occupied == 0) {
  var passengerseat = randomSeat(allseats);
  var passengerRow = Number(passengerseat.location.row);
  var passengerCol = Number(passengerseat.location.col);
  var newpassenger = {"location":{"row":11,"col":4},
  "target":{"row":passengerRow,"col":passengerCol},
  "aisle":{"row":11,"col":passengerCol},
  "state":UNSEATED, "luggage":luggage,
  "bagwaitcount":1, "seatmovecount": 1};
  passengers.push(newpassenger);
  var seatindex = allseats.indexOf(passengerseat);
  allseats.splice(seatindex, 1);
  aisle[0].occupied = 1;
 };};
```

50% chance of a passenger carrying a luggage

If aircraft is not full and if entry cell is unoccupied, send one in with a randomly allocated seat (randomSeat)

Define Passenger Attributes

Add passenger to 'passengers' list (recall slide 11)
Remove allocated seat from 'allseats' list so that no two passengers will be assigned the same seat

1 ADDPASSENGERS()

randomSeat()

A custom function we wrote to allocate a random seat to a passenger before entering the aircraft

```
function randomSeat(obj){
    var keys = Object.keys(obj)
    return obj[keys[ keys.length * Math.random() << 0]];
};
```

Note: The seat list contains all 180 seats for Random method, but for the Back-to-Front & Outside-In models, the list is split into 3 groups

1 ADDPASSENGERS()

Passenger Attributes

```
var newpassenger = {"location":{"row":11,"col":4},
"target":{"row":passengerRow,"col":passengerCol},
"aisle":{"row":11,"col":passengerCol},
"state":UNSEATED, "luggage":luggage,
"bagwaitcount":1, "seatmovecount": 1};
```

1. ***Location:*** *co-ordinate of starting point*
2. ***Target:*** *co-ordinate of seat*
3. ***Aisle:*** *co-ordinate of any point along the aisle*
   - *Used to move passenger along aisle until target.col matches aisle.col*
4. ***Luggage:*** *1 if has luggage*
5. ***Bagwaitcount:*** *used to make passenger wait in case luggage = 1*
6. ***Seatmovecount:*** *used to make passenger at target aisle wait in case a seated passenger has to give way*

② UPDATE.DYNAMIC.AGENTS()

Recall slide 14 all d3 code resides here

```
function updateDynamicAgents(){
  updateSurface();
   seatFull = 0;
 for (var passengerIndex in passengers){
      updatePassenger(passengerIndex);
 };};
```

- *Loop over each passenger on surface and update his/her location*
- *Call updateSurface() to animate the display*
- *Passenger movement behaviour is controlled in updatePassenger()*

HOW IS THE SIMULATION AUTOMATED?

3 ISCOMPLETED()

```
var record = [];
var time = 0;

function isCompleted() {

 if (seatFullCheck[seatFullCheck.length-1] == 180) {
     isRunning = false;
     record.push(statistics[0].cumulativeValue);

    statistics[3].cumulativeValue=record.length;
    statistics[2].cumulativeValue = 0;
    time = 0;
    for (var i = 0; i < record.length; i++) {
      time += record[i];}
    statistics[2].cumulativeValue = time / record.length;

    if (record.length == n) {
      updateSurface();
    };

    if (record.length < n) {
     4 Reload();
        isRunning = true;
     };}};
```

Used to calculate 'Average Time' & 'Number of Runs'

When all 180 passengers are seated, stop simulation and calculate:

1. *Average Time = time / record.length*
2. *Number of Runs = record.length*

Finally, do either:

1. *Reset and Run new simulation if # runs < n (record.length < n)*
2. *Terminate if n runs done (record == n)*

Where variable n is initialized in slide 3

④ RELOAD()

⑤

```
function Reload(){ //initialize page
adjust();

statistics[0].cumulativeValue=0;
statistics[0].count=0;
seatFull = 0;
seatFullCheck = [0,0];
passengers = [];
allseats = left_wing.concat(right_wing);

var drawsurface = document.getElementById("surface");
var creditselement = document.getElementById("credits");
var w = window.innerWidth; //width of entire window
var h = window.innerHeight; //height of entire window
var surfaceWidth =(w - WINDOWBORDERSIZE);
var surfaceHeight= (h-creditselement.offsetHeight - 3*WINDOWBORDERSIZE);

drawsurface.style.width = surfaceWidth+"px";
drawsurface.style.height = surfaceHeight+"px";
drawsurface.style.left = WINDOWBORDERSIZE/2+'px';
drawsurface.style.top = WINDOWBORDERSIZE/2+'px';
drawsurface.style.border = "thick solid rgba(80,109,47,0.5)";
drawsurface.innerHTML = ''; // Empties contents of drawing surface

numCols = maxCols;
cellWidth = surfaceWidth/numCols;
numRows = Math.ceil(surfaceHeight/cellWidth);
cellHeight = surfaceHeight/numRows;

surface = d3.select('#surface');
surface.selectAll('*').remove();
surface.style("font-size","100%");
// rebuild contents of the drawing surface
updateSurface();
};
```

Reload() is called whenever simulation is refreshed, or reset in the isCompleted() method

- *Re-initialize variables*
- *Empty contents on surface*
- *Redraw contents via updateSurface()*

5 ADJUST() & OTHERS

```
var animationDelay = 50; //controls simulation and transition speed
var simTimer;

function adjust() {
  window.clearInterval(simTimer); // clear the Timer
  animationDelay = 510 - document.getElementById("slider1").value;
  simTimer = window.setInterval(simStep, animationDelay);
};
```

Whenever Reload() is called, adjust is called as well

Called in isCompleted() & on-click event

```
function toggleSimStep(){
    isRunning = !isRunning;};
```

Convert (row, column) into (x,y) for screen

```
function getLocationCell(location){
    var row = location.row;
    var col = location.col;
    var x = (col-1)*cellWidth;
    var y = (row-1)*cellHeight;
    return {"x":x,"y":y};};
```

HOW DO PASSENGERS KNOW WHAT TO DO?

UPDATEPASSENGER()

1. Variables to control movement along AISLE

Passenger's current state & cell

e.g. passenger is at cell (11, 5)

```
{"location": {"row":11,"col":6}, "occupied": 1}
```

e.g. cell (11, 6) is empty

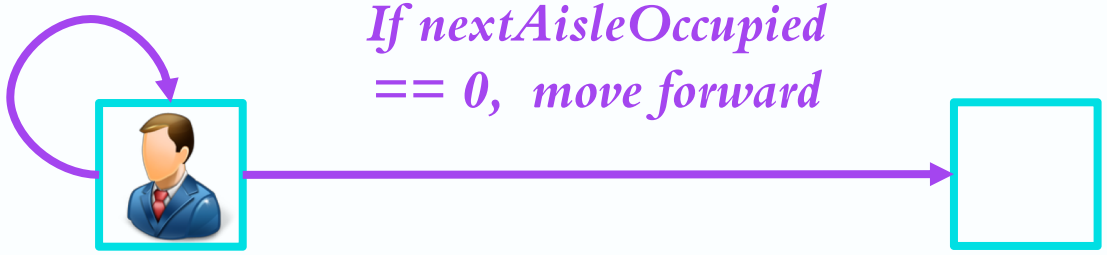
```
{"location": {"row":11,"col":7}, "occupied": 0}
```

- *hasArrivedAisle: switch state from UNSEATED to WAITING*
- *hasArrivedSeat: switch state from WAITING to SEATED*
- *seatFull: Keep track of occupied seats*

```
function updatePassenger(passengerIndex){

 //passengerIndex is an index into the passengers data array
 passengerIndex = Number(passengerIndex);
 var passenger = passengers[passengerIndex];

 // CURRENT location of passenger
 var row = passenger.location.row;
 var col = passenger.location.col;
 var state = passenger.state;

 // CURRENT aisle position
 var currentAislePos = aisle.filter(function(a){ return a.location.col === col });
 var currentAisleIndex = Number(aisle.indexOf(currentAislePos[0]));
 var currentAisleOccupied = aisle[currentAisleIndex];

 // NEXT aisle position
 var nextAislePos = aisle.filter(function(a){ return a.location.col === col + 1 });
 var nextAisleIndex = Number(aisle.indexOf(nextAislePos[0]));
 var nextAisleOccupied = aisle[nextAisleIndex];

 // determine if passenger has arrived at destination column in aisle
 var hasArrivedAisle = (Math.abs(passenger.target.col-col))==0;
 // determine if passenger is seated
 var hasArrivedSeat = (Math.abs(passenger.target.row-row)+Math.abs(passenger.target.col-col))==0;
 if (hasArrivedSeat) {
   seatFull += 1;
 };
```

ALONG AISLE

If nextAisleOccupied == 1



If nextAisleOccupied == 0, move forward

CurrentAisle Position

NextAisle Position

```
{"location": {"row":11,"col":7}, "occupied": 0}
```

```
{"location": {"row":11,"col":6}, "occupied": 1}
```

UPDATEPASSENGER()

3 CASES

1. *UNSEATED*
   - *Passenger is moving along aisle, till he/she stops at the target aisle*
   - *Then, if applicable, stow baggage and/or wait for seated passengers to give way*

2. *WAITING*
   - *Window/Middle seat passenger is moving to his/her seat*
     - *2 iterations for Window, 1 iteration for Middle*

3. *SEATED*
   - *Passenger arrives at his/her seat*

UPDATEPASSENGER()

Case: UNSEATED

- *If passenger has arrived at target aisle, check seatsOccupied dictionary*

- *Else if the passenger hasn't arrived at target aisle & the cell in front is vacant, move 1 step forward*

Case: WAITING

- *Window Seat (2 steps)*
- *Middle Seat (1 steps)*

Note: passenger moves in 1 step at the same time when case is switched from UNSEATED to WAITING

CASES

```
// Behavior of passenger depends on his or her state
switch (state) {
  case UNSEATED:
    if (hasArrivedAisle){
      if (seatsOccupied[col].includes(passenger.target.row)==false) {
        seatsOccupied[col].push(passenger.target.row);
      } // update dictionary
      if (passenger.luggage==0) { // NO LUGGAGE…
      }// end of NO LUGGAGE loop

      if (passenger.luggage==1) { // LUGGAGE…
      }// end of LUGGAGE loop
  }
    else if (nextAisleOccupied.occupied==0) { // hasn't reached AISLE
      currentAisleOccupied.occupied = 0;
      nextAisleOccupied.occupied = 1;
      passenger.location.col = col + 1;
      passenger.location.row = row;
  }
    break;

  case WAITING:

    if (Math.abs(passenger.target.row - row) == 1) { // MIDDLE seat
      passenger.location.row = passenger.target.row;
      passenger.location.col = col;
      passenger.state = SEATED;
    }

    else if (passenger.target.row - row == 2) { // WINDOW seat
      passenger.location.row = row + 1;
      passenger.location.col = col;
      passenger.state = WAITING;
    }

    else if (passenger.target.row - row == -2) { // WINDOW seat
      passenger.location.row = row - 1;
      passenger.location.col = col;
      passenger.state = WAITING;
    }
    break;
```

UPDATEPASSENGER()

Inside Luggage Loops

7 scenarios for movement from aisle to seat

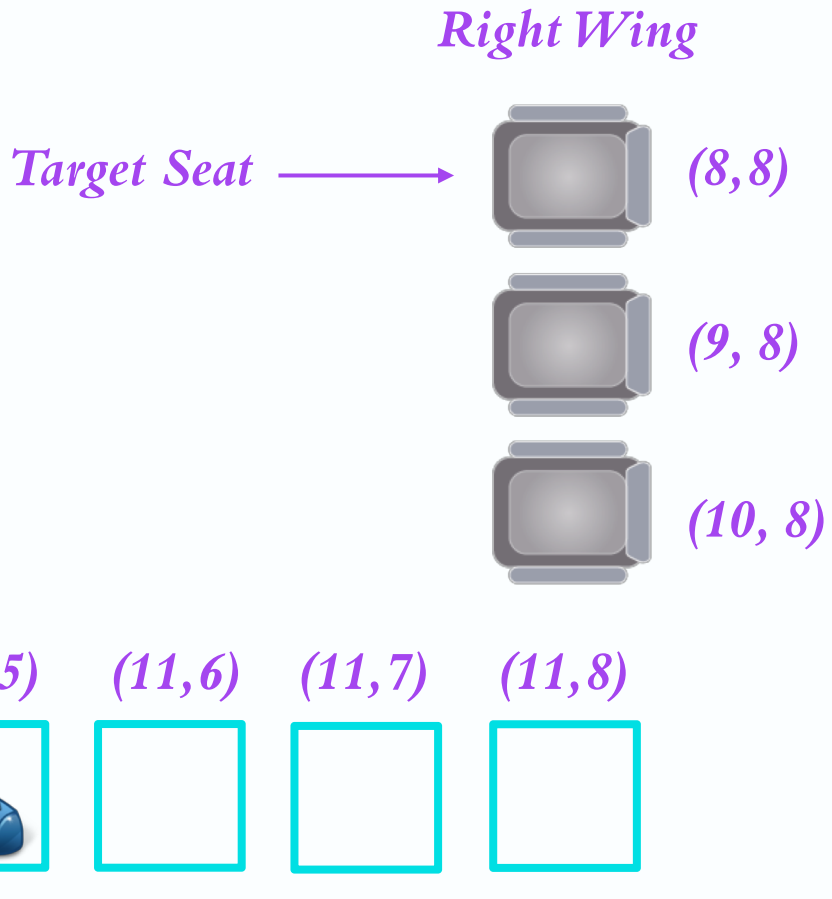
1. *Aisle Seater*
   - *Unblocked (1)*

2. *Middle Seater*
   - *Unblocked (2)*
   - *Blocked by aisle seater (3)*

3. *Window Seater*
   - *Unblocked (4)*
   - *Blocked by aisle seater only (5)*
   - *Blocked by middle seater only (6)*
   - *Blocked by both aisle and middle seater (7)*

UNSEATED

HAS NOT ARRIVED AT TARGET AISLE

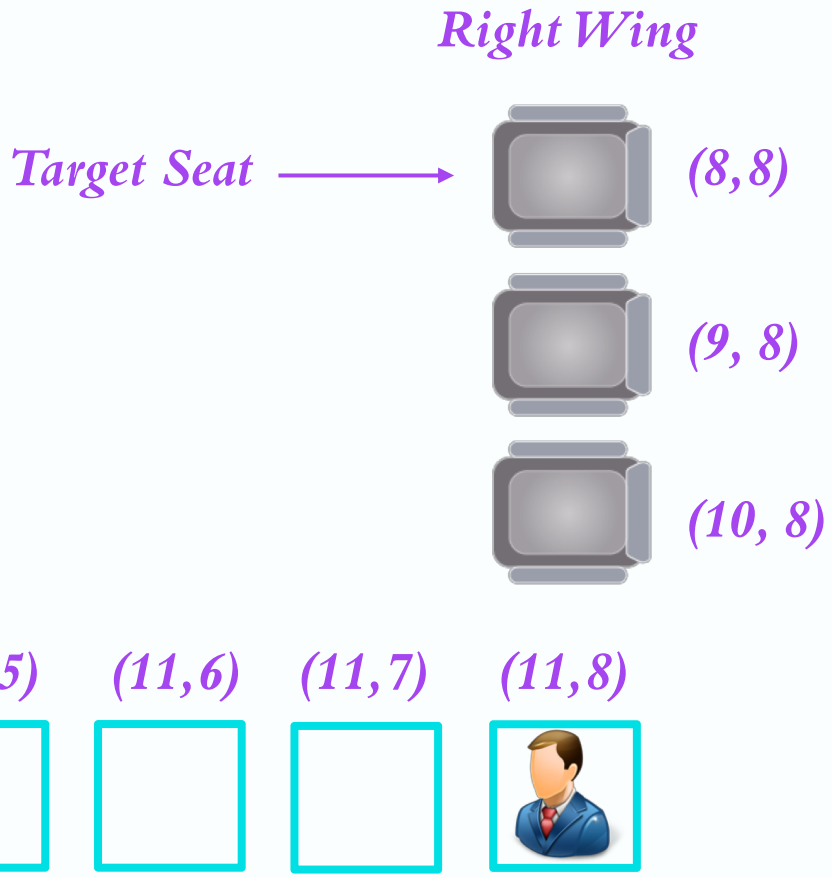


1. *Check if (11,6) is empty*
   - *Yes, move forward*
   - *No, stay*

2. *Repeat till reached (11,8)*

UNSEATED

ARRIVED AT TARGET AISLE



1. *hasArrivedAisle (11,8) Hurray!*

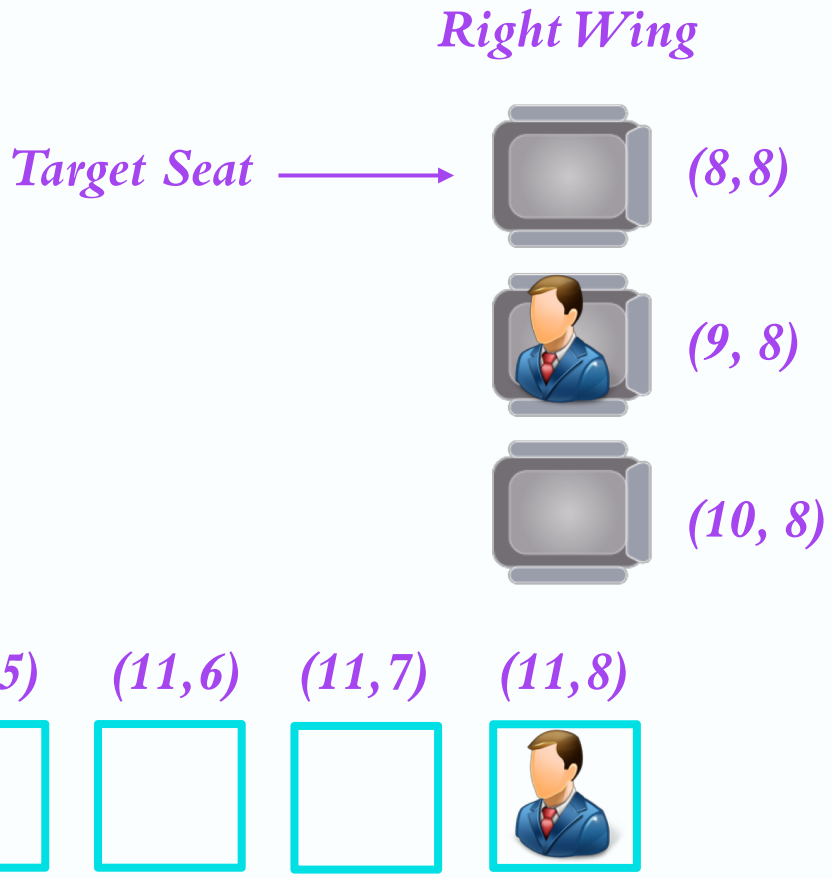
2. *Check if he has luggage*
   - *Yes, wait 8 iterations then go to step 3*
   - *No, then go straight to step 3*

```
if (passenger.luggage==1) { // LUGGAGE
      if (passenger.bagwaitcount<bagtime) {
        passenger.bagwaitcount+=1;
      }
      if (passenger.bagwaitcount==bagtime) {…}
    }// end of LUGGAGE loop
```

3. *Check seatsOccupied dictionary for (8,8), (9,8), (10,8)*

UNSEATED

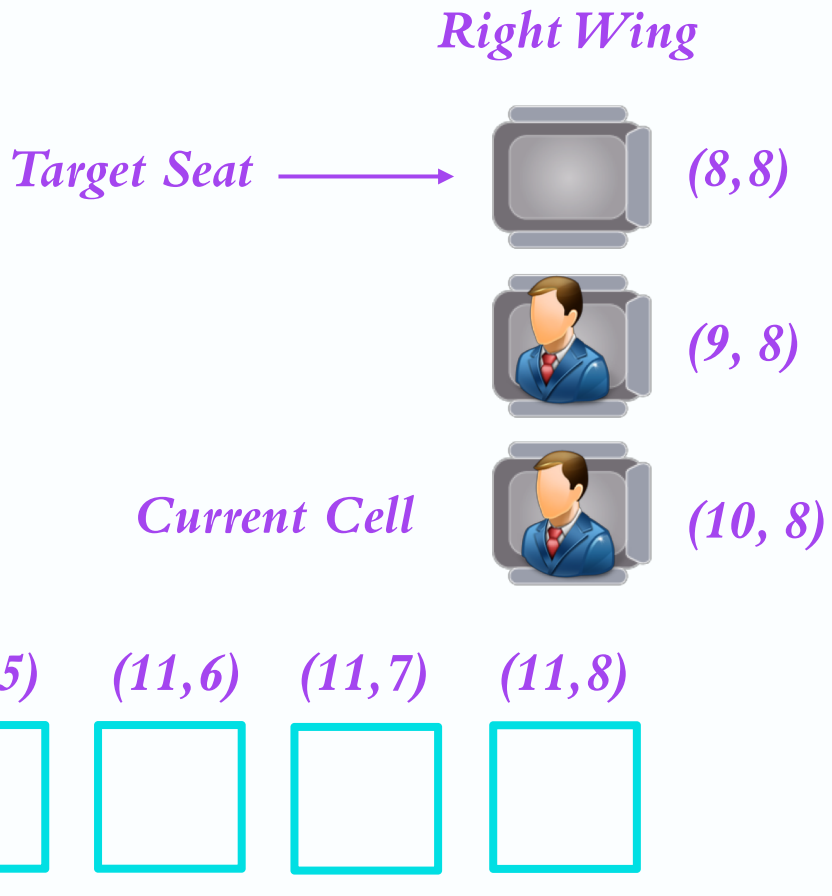
WAITING TO BE GIVEN PASSAGE



1. *seatsOccupied dictionary contains (9,8)! Meaning, there is someone seated at (9,8)*
2. *In this case, wait 5 iterations, and them move in to (10,8) and switch case to WAITING*
3. *Wait iteration loop varies depending on 7 scenarios outlined earlier*

WAITING

MOVING IN TO TARGET SEAT



1. *During WAITING state, passenger moves one cell at a time to (8,8)*

SEATED

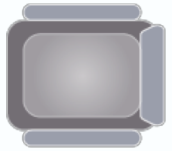
ARRIVED AT TARGET SEAT

Right Wing

Current Cell = Target Seat

(8,8)

(9, 8)



(10, 8)

(11,5) *(11,6)* *(11,7)* *(11,8)*

1. *hasArrivedSeat*
   - *Switch case to SEATED*
2. *Add (8,8) to seatsOccupied dictionary*

UPDATEPASSENGER()

1. Aisle seater is always unblocked

```
if (Math.abs(passenger.target.row-row)==1) {
  passenger.location.col = col;
  passenger.location.row = passenger.target.row;
  currentAisleOccupied.occupied = 0;
  passenger.state = SEATED;
}
```

UPDATEPASSENGER()

```
else if (passenger.target.row-row==2) {
  if (seatsOccupied[col].includes(12)) {
    passenger.seatmovecount+=1;

    if (passenger.seatmovecount==midmovetime) {
      passenger.location.row = row+1;
      passenger.location.col = col;
      currentAisleOccupied.occupied = 0;
      passenger.state = WAITING;
    }
  }
  else {
    passenger.location.row = row+1;
    passenger.location.col = col;
    currentAisleOccupied.occupied = 0;
    passenger.state = WAITING;
  }
}
```

2. Middle seater blocked by aisle seater

- *seatsOccupied dictionary is used to check for blockage (recall slide 24) at row 12.*
- *Once aisle seater has given way (seatmovecount == midmovetime), move in (row+1)*

3. Middle seater unblocked

Move in, State = WAITING (recall 'Note' in slide 24)

Note: Repeat this loop for seats in Right Wing, where

- *passenger.target.row – row == -2*
- *.includes(10)*
- *passenger.location.row = row - 1*

FROM AISLE TO SEAT

Example of Middle seater blocked by aisle seater

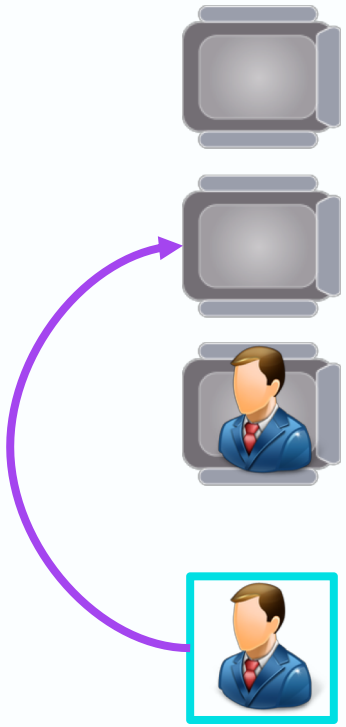
SeatsOccupied Dictionary contains (12,5)

Make passenger at (11,5) wait till 'midmovetime' is complete

Then, passenger moves in

- *(passenger.location.row = row – 1) x 2*

Right Wing



(8,5)

(9, 5)

(10, 5)

(11,5)

Basic Principle applies across 6 scenarios (barring aisle seater)

UPDATEPASSENGER()

```
else if (passenger.target.row-row==3) {
 // blocked by aisle and middle
  if (seatsOccupied[col].includes(12)) {
    if (seatsOccupied[col].includes(13)) {
      passenger.seatmovecount+=1;
      if (passenger.seatmovecount==windowmovetime3) {
        passenger.location.row = row+1;
        passenger.location.col = col;
        currentAisleOccupied.occupied = 0;
        passenger.state = WAITING;
      }}
    else { // blocked only by aisle
      passenger.seatmovecount+=1;
      if (passenger.seatmovecount==windowmovetime1) {
        passenger.location.row = row+1;
        passenger.location.col = col;
        currentAisleOccupied.occupied = 0;
        passenger.state = WAITING;
      }}}
  else { // blocked only by middle
    if (seatsOccupied[col].includes(13)) {
      passenger.seatmovecount+=1;

      if (passenger.seatmovecount==windowmovetime2) {
        passenger.location.row = row+1;
        passenger.location.col = col;
        currentAisleOccupied.occupied = 0;
        passenger.state = WAITING;
      }}
    else { // unblocked
      passenger.location.row = row+1;
      passenger.location.col = col;
      currentAisleOccupied.occupied = 0;
      passenger.state = WAITING;
    }}}
```

4. *Window seater blocked by both aisle seater & middle seater*

5. *Window seater blocked by aisle seater only*

6. *Window seater blocked by middle seater only*

7. *Window seater unblocked*

AGENDA

▶ BASIC MODEL ARCHITECTURE

▶ DECISION RULES

▶ ***DEEP DIVE INTO MODELS***

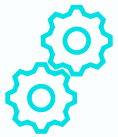
MODEL VARIATIONS

How are the Outside-In & Back-to-Front models different?

1. *AddPassengers()*
2. *Seats List*

Decision rules are ultimately the same across all models

SEATS LIST



OUTSIDE-IN

Remember the allseats array?
It's split into 6 now

- *Left Wing*
  1. *Window Seats*
  2. *Middle Seats*
  3. *Aisle Seats*

Repeat for Right Wing

Re-Initalized in Reload() everytime

```
var left_window_seat = [];
var left_middle_seat = [];
var left_aisle_seat = [];
var right_window_seat = [];
var right_middle_seat = [];
var right_aisle_seat = [];

 for (var i = 6; i < 36; i++) {
  left_window_seat[x] = {"location": {"row":8,"col":i}, "occupied": 0};
  x += 1};

 for (var i = 6; i < 36; i++) {
  left_middle_seat[x] = {"location": {"row":9,"col":i}, "occupied": 0};
  x += 1};

 for (var i = 6; i < 36; i++) {
  left_aisle_seat[x] = {"location": {"row":10,"col":i}, "occupied": 0};
  x += 1};

 for (var i = 6; i < 36; i++) {
  right_window_seat[y] = {"location": {"row":14,"col":i}, "occupied": 0};
  y += 1};

 for (var i = 6; i < 36; i++) {
  right_middle_seat[y] = {"location": {"row":13,"col":i}, "occupied": 0};
  y += 1};

 for (var i = 6; i < 36; i++) {
  right_aisle_seat[y] = {"location": {"row":12,"col":i}, "occupied": 0};
  y += 1};
```

BACK-TO-FRONT

Seats are split into 3 groups
Back, Middle, Front

Each group contains 20 rows of seats

Re-Initalized in
Reload() everytime

```
for (var j = 8; j < 11; j++) {
 for (var i = 6; i < 16; i++) {
  left_front_wing[x] = {"location": {"row":j,"col":i}, "occupied": 0};
  x += 1};};

for (var j = 8; j < 11; j++) {
 for (var i = 16; i < 26; i++) {
  left_middle_wing[x] = {"location": {"row":j,"col":i}, "occupied": 0};
  x += 1};};

for (var j = 8; j < 11; j++) {
 for (var i = 26; i < 36; i++) {
  left_back_wing[x] = {"location": {"row":j,"col":i}, "occupied": 0};
  x += 1};};

for (var j = 12; j < 15; j++) {
 for (var i = 6; i < 16; i++) {
  right_front_wing[y] = {"location": {"row":j,"col":i}, "occupied": 0};
  y += 1};};

for (var j = 12; j < 15; j++) {
 for (var i = 16; i < 26; i++) {
  right_middle_wing[y] = {"location": {"row":j,"col":i}, "occupied": 0};
  y += 1};};

for (var j = 12; j < 15; j++) {
 for (var i = 26; i < 36; i++) {
  right_back_wing[y] = {"location": {"row":j,"col":i}, "occupied": 0};
  y += 1

for (var i = 4; i < 36; i++){
 aisle[z] = {"location": {"row":11,"col":i}, "occupied": 0}
 z+=1};

var allfrontseats = left_front_wing.concat(right_front_wing);
var allmiddleseats = left_middle_wing.concat(right_middle_wing);
var allbackseats = left_back_wing.concat(right_back_wing);
```

ADDING PASSENGERS

OUTSIDE-IN

Initialize variables

PassengerCount: *increments everytime a passenger enters the plane. Used to trigger the waiting time between groups entering the aircraft (10 iterations)*

Waitingtime: *also used to trigger start and end of the waiting time between groups. Increased by 10 after the last passenger of every group enters, as dictated by the constants on the right.*

```
var PassengerCount = 0;
var waitingtime = 0;
const targetwaitingtime = 10;
const targetwaitingtime2 = 20;
const targetwaitingtime3 = 30;
const targetwaitingtime4 = 40;
const targetwaitingtime5 = 50;
```

Variables Re-Initalized in Reload() everytime

OUTSIDE-IN

1st group: Left Window seat passengers to enter first

PassengerCount increases to 30, then breaks loop (all 30 passengers of the group have entered the plane)

Waitingtime increased to 10, before the next group is sent in

```
if (PassengerCount < 30 & aisle[0].occupied == 0) {
  var passengerseat = randomSeat(left_window_seat);
  var passengerRow = Number(passengerseat.location.row);
  var passengerCol = Number(passengerseat.location.col);

 var newpassenger = {"location":{"row":11,"col":4},
   "target":{"row":passengerRow,"col":passengerCol},
   "aisle":{"row":11,"col":passengerCol},
   "state":UNSEATED, "luggage":luggage,
   "bagwaitcount":1, "seatmovecount": 1};
 passengers.push(newpassenger);
 var seatindex = left_window_seat.indexOf(passengerseat);
 left_window_seat.splice(seatindex, 1);
 aisle[0].occupied = 1;
 PassengerCount += 1;};

if (PassengerCount == 30 & waitingtime != targetwaitingtime ) {
  waitingtime += 1;};
```

OUTSIDE-IN

2nd group: Right Window seat passengers to enter next

```
if (PassengerCount >= 30 & PassengerCount < 60
  & waitingtime == targetwaitingtime) {
  var passengerseat = randomSeat(right_window_seat);
   var passengerRow = Number(passengerseat.location.row);
   var passengerCol = Number(passengerseat.location.col);
   var newpassenger = {"location":{"row":11,"col":4},
  "target":{"row":passengerRow,"col":passengerCol},
  "aisle":{"row":11,"col":passengerCol},
  "state":UNSEATED, "luggage":luggage,
  "bagwaitcount":1, "seatmovecount": 1};
  passengers.push(newpassenger);
  var seatindex = right_window_seat.indexOf(passengerseat);
  right_window_seat.splice(seatindex, 1);
  aisle[0].occupied = 1;
  PassengerCount += 1;};

 if (PassengerCount == 60 & waitingtime != targetwaitingtime2) {
   waitingtime += 1;};
```

PassengerCount increases to 60, then breaks loop (all 30 passengers of the 2nd group have entered the plane)

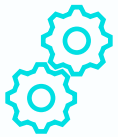
Waitingtime increased to 20, before the next group is sent in

OUTSIDE-IN

Process repeats till the last group (Right Aisle seat passengers) are sent in

```
if (PassengerCount >= 150 & PassengerCount < 180
  & waitingtime == targetwaitingtime5) {
 var passengerseat = randomSeat(right_aisle_seat);
 var passengerRow = Number(passengerseat.location.row);
 var passengerCol = Number(passengerseat.location.col);
 var newpassenger = {"location":{"row":11,"col":4},
 "target":{"row":passengerRow,"col":passengerCol},
 "aisle":{"row":11,"col":passengerCol},
 "state":UNSEATED, "luggage":luggage,
 "bagwaitcount":1, "seatmovecount": 1};
 passengers.push(newpassenger);
 var seatindex = right_aisle_seat.indexOf(passengerseat);
 right_aisle_seat.splice(seatindex, 1);
 aisle[0].occupied = 1;
 PassengerCount += 1;};
```

PassengerCount increases to 180, then no more passengers are added



BACK-TO-FRONT

Same steps apply! Only fewer, since there are only 3 groups now

```
var PassengerCount = 0;
var waitingtime = 0;
const targetwaitingtime = 10;
const targetwaitingtime2 = 20;
```

THANKS

for your attention