X - Boarding an Airplane

Note – The implementation language used for this example is called XOJO™, and is supported by XOJO Inc. (www.xojo.com). XOJO is a proprietary object-oriented language with a syntax derived from BASIC. The primary reason for choosing this language is greater ease of use than many alternatives, especially for novice or casual programmers. This allows more focus to be placed on the mathematics and mechanics of simulation rather than that of the language. Also, XOJO is a cross-platform development environment that supports Windows, Linux and Mac with minimal changes to either code or user interface, allowing flexibility in the choice of environments in which to work.

X.1 Background

If you have traveled on commercial airliners over the past decade, you may have noticed some changes in how the planes are boarded. In the past, most airlines would call people to board on the basis of their seating row, with the first class passengers first, and then those passengers in the rear of the plane, filling it from back to front. Now it is more common for the airline to print a boarding zone number on your boarding pass, and then call you up by that number. As is no surprise, the first class passengers invariably are still the first to be called, followed by various 'premier travelers', travelers with small children, and those travelers needing extra time to board. After that the boarding typically proceeds

by the printed zone number.

Why have airlines moved to this alternate boarding system? One reason is that it can make the boarding process a little more orderly than just going strictly from back-to-front. While the de-boarding process is still a chaotic free-for-all, getting the passengers on the plane is something the airlines can still influence through planning, and if they plan correctly they can execute the boarding process a a few minutes faster. While a few minutes may not sound like a lot, when you multiply them by the number of flights even a medium-sized airline operates a year, the total time savings can result in the ability to run significantly more flights with the same set of airplanes and the same airport gate allocations. Increasing the number of passengers you can carry, and the corresponding revenue is a highly desirable goal, especially in a business like air travel where the profit margins can be razor thin.

To implement an effective boarding zone strategy requires assigning zone numbers to each seat on the plane. While loading up an airplane with passengers may seem like a straightforward task, determining how to load it up in the shortest time isn't. Simulation can be an effective tool in comparing how different boarding zone strategies compare.

X.2 System Description

The system we will be modeling should be familiar to most readers: the passenger compartment of a commercial airliner. To simplify the modeling task, we will assume a single class of seating, with 30 rows of 6 seats, with 3 seats on each side of the aisle. The airplane starts out initially empty (all seats unoccupied) and passengers enter the aisle from the front of the plane and move to their seats as quickly as they can.

X.3 Modeling Paradigm

> **Discrete Variable/Discrete State** – Because there are a finite number in the cabin (despite the airlines' best and continual efforts to squeeze in more), this collection of seats is a discrete

variable model. Since they are either empty or occupied, they assume discrete states. While it might possible to model the presence of passengers in the aisle as continuous positions, in this model, the aisle is divided up into a series of discrete 'occupancy cells' which can contain at most one passenger. This also results in a set of discrete variables holding a discrete state.

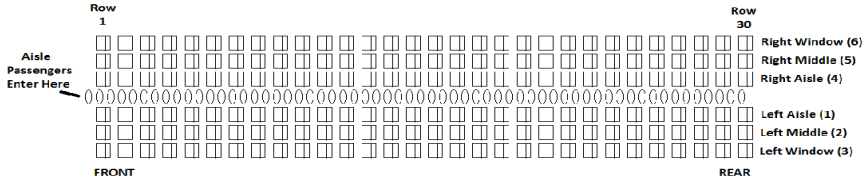
Discrete Fixed dT – Time in this model will be advanced in fixed intervals of dT. While this model could potentially be realized in the discrete-event paradigm, the additional organizational complexity may not be justified by either the increase in model fidelity or savings in compute time.

Deterministic (mostly)– The simulation model of how passengers behave in the aisle and during seating is be completely deterministic; if the model is run twice with the same passenger boarding queue, it will exhibit the same behavior and total boarding time. The order in which passengers arrive within each boarding zone, however, (the input) will be randomized, and is stochastic. The probabilistic nature of the simulation resulting from randomness in the boarding queues will therefore require a statistically significant number of runs to be able to meaningfully compare the effectiveness of the strategies.

X.4 Model Description

In this model there are two main sources of state to consider; the seats and the aisle, which both may be either occupied or unoccupied. Figure X.1 shows the cabin layout that will be modeled. In this cabin there are 30 rows of seats, with 6 seats in each row, divided by an aisle down the middle. The seats are relatively straightforward to model as they are either empty or occupied and can be represented by an array of binary variables.

Figure X.1 – Cabin Layout, 30 Rows of 6 Seats



Most of the interesting action in this model happens in the aisle. While there are numerous ways in which aisle and the people in it could be modeled, we will be taking the approach of dividing the aisle into a number of occupancy cells, each of which can contain a single person, or be empty. For each row of seating, there will be two aisle occupancy cells, representing a maximum aisle capacity of two people for each row of seating. Passenger movement in the aisle will be represented by passengers transitioning from one cell to the next empty cell. A movement time will be specified to pace how fast passengers may move down the aisle. When a cell contains a passenger it will be assumed that passengers wishing to advance down the aisle through that cell must wait for it to become unoccupied. A passenger stopped in the aisle therefore blocks passengers immediately behind him from passing.

When a passenger reaches his destination row, he will move out of the aisle and into his seat. The amount of time required to do so will be a function of the people already seated in the row. In an actual airplane (ignoring issues of stowing baggage), if a row is empty, a passenger can quickly move into his seat. On the other hand, if a passenger has a window seat, and the aisle and middle seats are already occupied, getting to his seat typically requires the seated passengers to get up, let him in, and then re-seat themselves, taking additional time. The seating time can be represented in tabular form as shown in Table X.1.

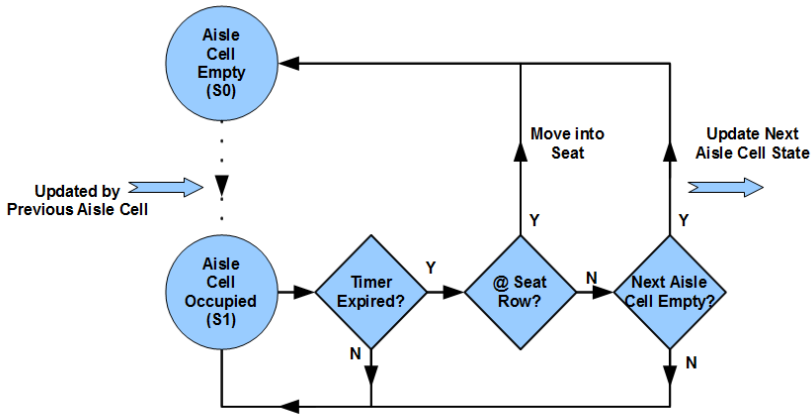
Table X.1 – Seating Time From Aisle with Various Seats Filled

Destination Seat	Aisle Seat	Middle Seat	Time to Seat	Default Value (sec)
Aisle	-	-	T_A	7
Middle	Empty	-	T_{ME}	7
Middle	Filled	-	T_{MF}	15
Window	Empty	Empty	T_{WEE}	7
Window	Empty	Filled	T_{WEF}	15
Window	Filled	Empty	T_{WFE}	15
Window	Filled	Filled	T_{WFF}	25

For the purposes of this model, we will make the simplifying assumption that the time for a passenger to get into his seat is a function of the number of people who must get up, so that ($T_{WEF} = T_{WFE}$).

The aisle cells are the heart of this simulation, and are modeled as an array of state machines, each aisle cell's operation described by the flowchart of Figure X.2. The aisle cell is initially in an 'empty' state (S0), and remains there until it is set into the occupied state (S1) by having a passenger seating code written into it by either the previous aisle cell, or if it the first aisle cell, by the passenger queue. When this update occurs, the aisle cell's associated timer is also set to '0'. On each call of the ADVANCE method, all timers are incremented by timestep dT.

Figure X.2 – Aisle Cell State Machine



When in the occupied state the aisle cell makes a series of decisions at each time step. The first test is whether the timer has expired. If the seating code corresponds to the adjacent row, however, the expiration time will be given by the seating time from aisle to the assigned seat, and depends on the people already seated in that row as shown in Table X.1. The indicated seat is then filled, and the aisle state is set back to empty.

If the seating code for that aisle cell is not equal to the adjacent row, then the expiration time is the passenger advance time, the interval which a passenger needs to move to the next aisle cell. When this condition occurs, and the next aisle cell is empty, then the next aisle cell is set to occupied, and the current aisle cell is set to empty – the passenger advances one aisle cell. If the next aisle cell is occupied, however, the present aisle cell remains in the occupied state, and on each future time step continues to check if it can advance the passenger along.

Finally, one must generate the input sequences of passengers. In this model, passengers have no unique identifying characteristics other than their seating assignment. For a given boarding strategy, each passenger must be assigned a boarding zone number, and the passengers order of entry must be sorted by zone. To be able to evaluate a given strategy, it must be statistically tested against various exact boarding orders, so the passenger order must also be randomized within each boarding zone to provide meaningful results. In addition to managing the case of a

completely full airplane, the model should also be able to evaluate the effects of partial boardings.

The facilities for building the boarding queue support the following boarding strategies (illustrated in Figure X.3):

1. **Random** – Passengers enter the plane in random order without regard to their seating.

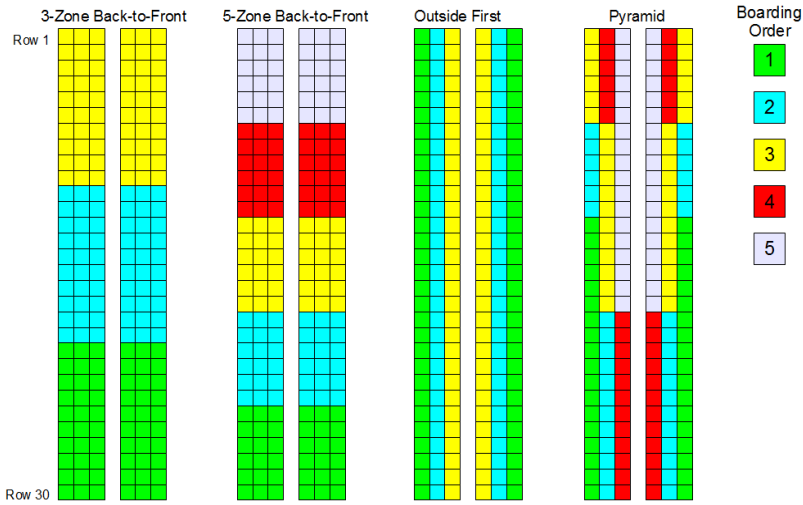
2. **3-Zone Back-to-front (3ZBF)**– The cabin is divided into three zones which are boarded from the back of the cabin to the front. The entry order of passengers within each zone is random.

3. **5-Zone Back-to-front (5ZBF)**– The cabin is divided into five zones which are boarded from the back of the cabin to the front. The entry order of passengers within each zone is random.

4. **Outside-First (OF)**– The cabin is first boarded with window seats, then middle seats, then aisle seats. Again, within each boarding zone, passenger order is random.

5. **Pyramid** – This is a hybrid of the back-to-front and outside-first boarding strategies.

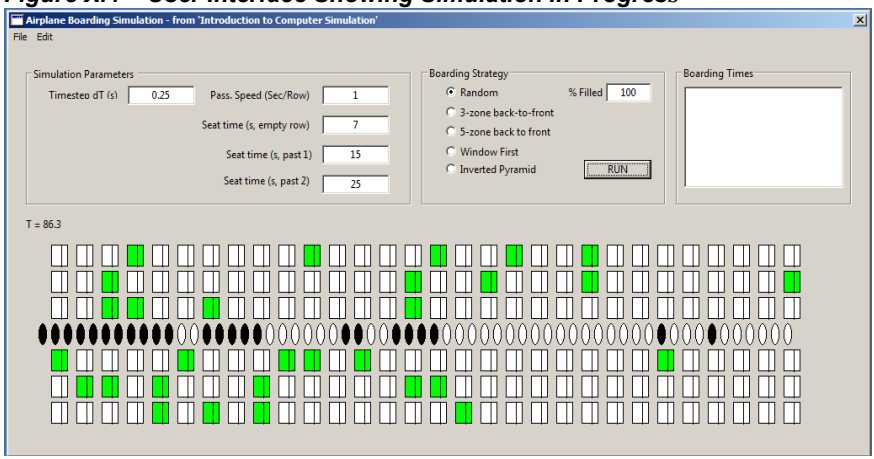
Figure X.3 – Seating Zone Assignments for Various Boarding Strategies



X.5 Implementation

While the simulation ultimately generates a single value as an output – time to board the plane and get everyone into their seats – animating the process makes for a much more instructive and entertaining model. The simulation is based on a single window, where the seating and aisle statuses are graphically depicted. Also, controls are provided for specifying how fast the passengers can move, the boarding strategy to be modeled, and the percentage of the plane that is to be filled. When the simulation completes, the total boarding time is reported in a text box. Figure X.4 shows the layout of the simulation window.

Figure X.4 – User Interface Showing Simulation in Progress



The User Interface (UI) controls for this simulation are:

- **Timestep** (textfield *txtDT*)– This is a real value that allows the user to specify the amount of time by which the simulation advances (dT). Normally this number should be set to be smaller that any of the other values specified in the 'Simulation Parameters' group.

- **Pass. Speed** (textfield *txtPassSpeed*)– This real value specifies how many seconds of simulated time are required for a passenger to get down the aisle one seating row (two aisle cells) if unimpeded.

- **Seat Time(past 0)** (textfield *txtPass0*)– This real value specifies how many seconds are required for a passenger to get from the aisle into his seat if there is nobody has to get up to let him in.

- **Seat Time(past 1)** (textfield *txtPass1*)– This real value specifies how many seconds are required for a passenger to get from the aisle into his seat if one seated passenger has to get up to let him in.

- **Seat Time(past 2)** (textfield *txtPass2*)– This real value specifies

how many seconds are required for a passenger to get from the aisle into his seat if two seated passengers need to get up to let him in. This value only applies when someone is trying to get into a window seat.

- **Boarding Strategy Select–** This set of radio buttons lets the user select the boarding strategy to simulate.

- **% Filled** (textfield *txtPctFull*)**–** This real value specifies the percentage to fill the cabin to. For example, specifying a 75 percent filled cabin will result in a boarding queue with 135 passengers (75% x 180 available seats).

- **Boarding Times** (textarea *txtOUT*)**–** This textbox shows the total simulation time that elapses between the start of the simulation and the time at which all passengers have been seated.

- **RUN button** (Pushbutton *btnRUN*)**–** Calls RUN_SIMULATION method and starts a simulation run

- **Simulation Time** (label *lblSimTime*) – used to display simulation time during run.

- **Animation Area** (canvas *cvsOUT*) – used as drawing surface for animation display. The status of the seats and aisle is updated visually every time tick as the simulation proceeds.

The key variables and data structures in this simulation (all properties of the main Window) are:

- **dT –** The timestep at which the simulation advances.

- **SimTime** – The current simulation time.

- **PassengerQueue() -** The list of passengers boarding, each represented by a passenger seat code.

- **Aisle_AdvanceTime** – the amount of time required for a passenger to advance from one aisle cell to the next if unimpeded.

- **SeatingTimeTable(,,)** - a table containing seating times under various conditions of row occupancy. Referenced through the method SeatingTime(Seatcode).

- **Aisle_Exit()** - An array containing the seating rows into which the indexed aisle cells exit. Aisle cells that do not exit to seats contain '0'. In the current model, only even-numbered aisle cells can exit into a seat – this allows two people to be in the aisle for every row of seats. This array does not change during the course of the simulation, but is used to associate aisle cells with seating rows. By storing this information in an array versus hard-coding it into the program logic makes it simple to change the association. For example, in the length of an aisle where it passes through a bulkhead there may be no seats, but this section of aisle may be able to hold considerably more than two or three people.

- **Aisle_Status()** - An array containing the status of each cell in the aisle, represented by a passenger seat code. Empty cells contain '0'. Occupied cells contain a non-zero seating code.

- **Aisle_Timer()** - An array containing the elapsed time since that aisle cell became 'occupied' by its current passenger. Each entry in this array is incremented by dT on every simulation clock tick. 'Expiration' of these timers is determined by program logic, and is a different value depending on where the passenger is headed to (next aisle cell or into a seat).

- **Seats(,)** - A binary matrix containing the seat status (1=filled/0=empty).

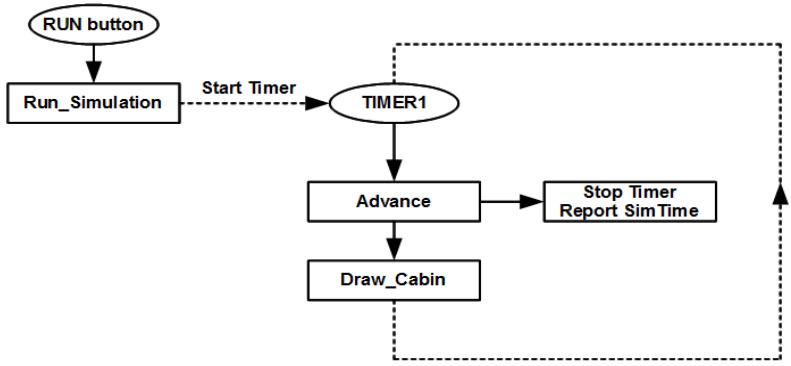
Passenger seat codes are represented by integers of the form *RRS*, where *RR* is the row number from 1-30, and the seat in that row *S* is represented by 1-6, defined in Table X.2.

Table X.2 – Seating Positions Within Row

Seat	Position
1	Left Aisle (C)
2	Left Middle (B)
3	Left Window (A)
4	Right Aisle (D)
5	Right Middle (E)
6	Right Window (F)

Figure X.5 shows the top-level structure of the simulator. When the RUN button is clicked, it calls the Run_Simulation method, which reads parameters from the UI, builds the passenger queue and performs other initialization functions before enabling the Timer1. On each timer tick, the Advance and Draw_Cabin methods are called. When all passengers are seated, Timer1 is disabled and the stop time reported.

Figure X.5 – Top-level Simulator Structure



At a more detailed level, the simulator uses the following methods:

- **Run_Simulation** – Performs initialization functions including the construction of the boarding queue.

- **Build_Boarding_Queue(Strategy, PctFull)** – builds the passenger boarding queue given the boarding strategy and the percentage to fill the plane (0-100%).

- **Assign_Boarding_Zones(Strategy, Zones())** - assigns a boarding zone to each passenger in the boarding queue based on his seat assignment and the chosen strategy.

- **Advance()** – Steps the simulation a single time step dT and updates all the state variables. This function is the heart of the simulation.

- **Seating_Time(Seatcode)** – returns the time to get a passenger out of the aisl and into his seat based on his seatcode and the status of the passengers already seated in his aisle.

- **Draw_Cabin(g)** – called by the cvsOUT Paint event to draw the cabin in that contr

- **Passenger_Row(pc), Passenger_Seat(pc), Passenger_Seatcode(row,seat)** – Utility functions to encode and decode the seatcode into and from rows and seats. Centralizing these operations as functions makes the code a little easier to understand.

X.6 Results & Analysis

Although the model itself is deterministic, and will yield the same results for the same passenger boarding order, differences in boarding order can result in noticeable boarding time differences. For this reason, one must run the simulation several times for each case and analyze the results statistically. For the case of a 100% full aircraft, 10 runs of the simulation yielded the results shown in Table X.3.

Table X.3 – Results for 10 Runs of Each Strategy with Full Plane

Run	Random	3ZBF	5ZBF	OF	Pyramid
1	592	557	624	344	368
2	558	621	706	355	347
3	569	582	690	356	344
4	574	586	661	366	353
5	558	617	701	342	368
6	561	606	673	311	350
7	615	633	667	345	336
8	550	594	695	320	348
9	531	610	682	365	378
10	573	617	700	345	337
mean	568	602	680	345	353
stdev	23	23	25	18	14

One counter-intuitive result of these simulations is that the Random boarding strategy beats both of the back-to-front strategies (3ZBF, 5ZBF), and that boarding from the outside-in (OF) beats both. Although the mean times for the Random and 3ZBF strategies are close, performing a T-test on the data yields a P-value of 0.0018, which suggests that they are unlikely to be the same. The performance of the more complex pyramid strategy, however, did not seem to significantly differ from that of the simpler outside-in strategy.

One question that arises is how general the above results are. If one changes the relative values of passenger speed and seating times, one could reasonably expect different strategies to come out on top. Let us consider the case, however, where the airplane is only filled to 75% of capacity, as opposed to the 100% assumption made in the previous experiment. Table X.4 shows the results of running the model with the plane only 75% full. Note that in this case the 3-zone back-to-front (3ZBF) strategy handily beats the random boarding strategy, but also that outside-in still looks significantly better.

Table X.4 – Results for 10 Runs with 75% Full Plane

Run	Random	3ZBF	5ZBF	OF	Pyramid
1	424	376	432	263	292
2	434	325	443	266	267
3	395	378	370	282	257
4	388	394	445	262	262
5	429	375	442	271	278
6	382	381	452	263	252
7	398	438	423	273	268
8	400	356	454	277	270
9	412	409	378	275	266
10	407	399	399	274	270
mean	407	383	424	271	268
stdev	18	30	31	7	11

Perhaps the best way to validate this model would be to rent a plane, and hire several hundred people for a day or two and stage some boarding exercises. An alternate observational method might be to go on some trips and video people boarding, but this is not recommended, as that kind of behavior may be frowned upon by the U.S. Transportation Safety Administration, and could conceivably win you an all-expense paid vacation to Guantanamo Bay - with a one-way ticket.

One observation of these results is that they seem excessively optimistic (boarding a plane in 10 minutes or less) versus the experiences I have had in boarding similar-sized planes. Several things could contribute to this discrepancy:

1) The model is excessively simplified. In the modeling process it is often good to start with a simple model and add complexity when it is needed, as opposed to starting out at maximum complexity and discovering what can be pared away. Some simplifying assumptions made are that passengers all move uniformly, they go directly to their seat, and there is no time specifically associated with stowing baggage. None of these assumptions are valid in reality, but without supporting data on which to model them, adding them in is just adding gratuitous complexity.

2) There are fundamental entities or processes missing from the model. For example, we do not model the storage capacity of the baggage compartments, although on a full flight this seems to have a lot to do with how fast people can get on board and seated.

3) The parameters used as input are not sufficiently accurate. Even though this model relies on extreme simplification, the rate at which a passenger can move down the aisle and time required to get out of the aisle can have a dramatic effect on the results. For the purposes of this example, these times were all assumed, and were not based on observation or measurement. While scaling all of these times together should not affect the relative ranking of the various boarding strategies, having their relative ratios change might be reasonably expected to have a significant effect. Without validating the input parameters, even the relative ranking results of the model should be taken with an appropriately-sized grain of salt.

X.7 Conclusion

In this chapter we examined a highly simplified simulation of an airplane boarding process, in which passengers were modeled as seat destinations, and were moved along the aisle through an array of deterministic state machines. Through an animated user interface, behavior such as aisle blockage and congestion could readily be observed. Although the model comprised low-complexity components, their interaction provided some counter-intuitive system-level results, for example random boarding beating a more orderly back-to-front strategy.

X.8 Questions/Problems

1. Variation in Passenger Characteristics

Some passengers are faster or slower than others. For example, families

with small children, and the elderly often take more time to get down the aisle and into their seats than go-getting executive types. What changes to the model would be needed to accommodate variations in passenger speed? *Hint – replacing the integer seating code with a passenger 'object' will make it easier to define a given complex characteristics for given passengers.*

2. Variation in Cabin Layouts

Most large commercial aircraft do not have seating arrangements that are as uniform as were considered in this model; for example, the first-class seating section invariably has fewer seats per row than coach. Larger planes can have even more complex seating arrangements including multiple aisles, exemplified by the cabin layout of the Boeing 747. How might this model be generalized to handle arbitrary cabin layouts?

3. Data collection

The output of the model was used to drive animation, and the only numerical result produced was total boarding time. How might the model be augmented to collect additional data on the boarding process? Such additional data might include how long people stood blocked in the aisle and how often people had to get up to let other people get into their seats. Also, for purposes of collecting statistically significant numbers of runs, automating the run process (and possibly eliminating the animation) would greatly aid data collection.

3. Southwest Airline's Boarding Strategy

Some airlines, most notably Southwest, have dispensed with the whole idea of assigned seating, and have their passengers select their seats on a first-come-first-served basis as they enter the plane. At the gate, the passengers are typically lined up into queues based on a boarding order number before entering the plane, where the passenger's boarding number is printed on his or her boarding pass. In addition to maintain an orderly boarding process, this queuing also allows Southwest to offer 'premium seating' by allowing select passengers to be closer to the front of the queue.

In practice, Southwest's boarding technique seems to operate smoothly and efficiently despite its apparent simplicity. From a simulation standpoint, however, Southwest's boarding process is actually a much more complex system to model than the more conventional assigned seating process, as passengers do not necessarily have a fixed seating target to head to, but a set of general preferences (e.g. aisle row in back) which must be resolved on-the-fly. What kinds of changes to the model might be needed to simulate Southwest's boarding process?

4. Optimal Seating Strategies

The model compared a couple of simple predetermined seating strategies. What kinds of approaches might be used to discover optimal or near-optimal seating strategies for a given number of boarding zones?

X.9 Further Reading

AIRCRAFT BOARDING FINE-TUNING, Capelo, E., de Castro Silva, J.L., van den Briel, M.H.L., Villalobos, J.R.. XIV INTERNATIONAL CONFERENCE ON INDUSTRIAL ENGINEERING AND OPERATIONS MANAGEMENT, Rio de Janeiro, Brazil, Oct 13-16, 2008

X.10 XOJO Code

Method RUN_SIMULATION

```
  ' set up aisle_exits array. Non-zero values indicate the row where
  ' the passenger can exit at each point in the aisle.
  for i = 1 to 30
    Aisle_Exit(i*2) = i
  Next

  '***** get simulation parameters from UI *****

  ' timestep
  dT = txtdT.text.Val

  ' passenger speed in aisle
  Aisle_AdvanceTime = txtPassSpeed.text.Val/2

  ' initialize the Seating time table
  ' indices are [target_seat, aisle_occupancy, window_occupancy]
  '
  SeatingTimeTable(1,0,0) = txtSeat0.text.Val  ' Aisle Seat - middle & window
occupancy don't matter
  SeatingTimeTable(1,0,1) = txtSeat0.text.Val
  SeatingTimeTable(1,1,0) = 0
  SeatingTimeTable(1,1,1) = 0
  SeatingTimeTable(2,0,0) = txtSeat0.text.Val  ' Middle seat - only aisle
seat occupancy matter
  SeatingTimeTable(2,0,1) = 0
  SeatingTimeTable(2,1,0) = txtSeat1.text.Val
  SeatingTimeTable(2,1,1) = 0
  SeatingTimeTable(3,0,0) = txtSeat0.text.Val  ' Window seat - aisle & middle
occupancy both count
  SeatingTimeTable(3,0,1) = txtSeat1.text.Val
  SeatingTimeTable(3,1,0) = txtSeat1.text.Val
  SeatingTimeTable(3,1,1) = txtSeat2.text.Val



  ' get percent full from UI
  pctfull = txtPctFull.text.val

  ' get stratgy from UI
  if rbt3Zone.value then
    strategy = S_3ZBF
  elseif rbt5Zone.value then
    strategy = S_5ZBF
  elseif rbtWindowFirst.value then
    strategy = S_OUTIN
  elseif rbtPyramid.value then
    strategy = S_PYRAMID
  else
    strategy = S_RANDOM
  end if

  ' Initialize Boarding Queue

  Build_Boarding_Queue(Strategy, PctFull)
```



```
  ' initialize seating & aisle states

  for i = 1 to ROWS
    for j = 1 to ROW_WIDTH
      Seats(i,j) = 0
    next j
  next i

  for i = 1 to ubound(Aisle_Status)
    aisle_status(i) = 0
  next i


  ' set up & start timer1 to repeticitively call 'Advance' method
  SimTime = 0
  timer1.Mode = timer.ModeMultiple
  timer1.Period = 50
  Timer1.Enabled=True
```

End RUN_SIMULATION

Method BUILD_BOARDING_QUEUE

```
   ' Fill Queue with all seats - assumes 180 total seats
  ' Passengers are represented by integer seat codes RRS where row varies 1-
99 and seat in row varies 1-9

  dim r as integer, s as integer, i as Integer, zones(-1) as integer, n as
integer

  ' fill queue with seats

  redim PassengerQueue(-1)

  for r = 1 to ROWS
    for s = 1 to ROW_WIDTH
      PassengerQueue.Append Passenger_SeatCode(r,s)
    next s
  next r


  n = ((UBound(PassengerQueue)+1) * (pctfull/100)) - 1

  ' randomly shuffle & cut to percentage
  PassengerQueue.Shuffle
  redim PassengerQueue(n)

  ' Assign Zones
  Assign_Boarding_Zones (Strategy, Zones)

  ' sort by zone
  zones.SortWith PassengerQueue
```

End BUILD_BOARDING_QUEUE

Method ASSIGN_BOARDING_ZONES(Strategy as integer, Zones() as Integer)

```
' Assigns boarding zones based on strategy selection in UI
' Zones are assigned as priority (higher number boards first)
' Assumes cabin configuration of 30 rows of 6 seats, 180 total seats

dim r as integer, s as integer, z as integer, i as integer, n as integer

n=ubound(PassengerQueue)

redim Zones(n)

select case strategy
case S_3ZBF
  ' back to front in 3 waves
  for i = 0 to n
    select case Passenger_Row(PassengerQueue(i))
    case 1 to 10
      Zones(i) = 1
    case 11 to 20
      Zones(i) = 2
    case 21 to 30
      Zones(i) = 3
    end Select
  next i

case S_5ZBF
  ' back to front in 5 waves
  for i = 0 to n
    select case Passenger_Row(PassengerQueue(i))
    case 1 to 6
      Zones(i) = 1
    case 7 to 12
      Zones(i) = 2
    case 13 to 18
      Zones(i) = 3
    case 19 to 24
      Zones(i) = 4
    case 25 to 30
      Zones(i) = 5
    end Select
  next i

case S_OUTIN
  ' Board Windows 1st
  for i = 0 to n
    select case Passenger_Seat(PassengerQueue(i))
    case 1,4  ' aisle
      Zones(i) = 1
    case 2,5  ' middle
      Zones(i) = 2
    case 3,6  ' window
      Zones(i) = 3
    end Select
  next i

case S_PYRAMID
  '  pyramid boarding
```



```
    for i = 0 to n

      select case Passenger_Seat(PassengerQueue(i))

      case 1,4  ' aisle
        select case Passenger_Row(PassengerQueue(i))
        case 1 to 18
          Zones(i) = 1
        case 19 to 30
          Zones(i) = 2
        end select

      case 2,5 ' middle
        select case Passenger_Row(PassengerQueue(i))
        case 1 to 6
          Zones(i) = 2
        case 7 to 18
          Zones(i) = 3
        case 19 to 30
          Zones(i) = 4
        end select

      case 3,6 ' window
        select case Passenger_Row(PassengerQueue(i))
        case 1 to 6
          Zones(i) = 3
        case 7 to 12
          Zones(i) = 4
        case 13 to 30
          Zones(i) = 5
        end select
      end Select

    next i

  case else  ' assume random
    ' random - just assign a sequence as passenger queue has already
    ' been shuffled
    for i = 0 to n
      Zones(i) = i
    next i

  end select
```

End ASSIGN_BOARDING_ZONES

Method ADVANCE

```
  ' Advances the state of the simulation by one clock tick each time called


  dim i as integer, r as integer, PC as Integer, Aisle_Empty as Boolean

  ' Advance simulation time
  SimTime = SimTime + dT

  ' Check if any passengers in boarding queue or aisle. If not, simulation is
DONE,
  ' turn off TIMER1, Report Stoptime in txtOUT text field, and return

  Aisle_Empty = True
  for i = 1 to AISLESLOTS
    if Aisle_Status(i) > 0 then Aisle_Empty =False
  next i

  if Aisle_Empty  and ubound(PassengerQueue) <  0 then
    timer1.Enabled = False
    txtOUT.text = txtOUT.text + str(SimTime)+EndOfLine
    return
  end if

  ' if first aisle slot empty and passengers in boarding queue, put next
passenger into slot #1 and reset timer
  if ubound(PassengerQueue) > -1  and Aisle_Status(1) = 0 then
    Aisle_Status(1) = PassengerQueue.Pop
    Aisle_Timer(1) =0
  end if

  ' increment all aisle timers
  for i = 1 to AISLESLOTS
    Aisle_timer(i) = Aisle_Timer(i) + dT
  next i

  ' update the aisle slot states from back to front to correctly update the
next state
  for i = AISLESLOTS-1 downto 1

    PC = aisle_status(i)  ' get passenger code

    if passenger_row(PC) = Aisle_Exit(i) then ' passenger is waiting to move
into seat

      if Aisle_Timer(i) >= Seating_Time(PC) then ' timer expired - move into
seat, clear Aisle

        Seats( passenger_row(PC), Passenger_Seat(PC) ) = 1
        Aisle_Status(i) = 0

      end if

    else  ' passenger is waiting to advance to next aisle slot

      if Aisle_Timer(i) >= Aisle_AdvanceTime and Aisle_Status(i+1)= 0 then

        ' if next aisle slot empty and timer expired, move into next slot

        Aisle_Status(i+1) = PC
```



```
        Aisle_Timer(i+1) = 0
        Aisle_Status(i) = 0

      end if

    end if

  next i
```

End ADVANCE

Method DRAW_CABIN(g as graphics)

```
    ' This method draws cabin status into passed graphics object (cvsOUT)
  ' This method assumes 30 rows of 6 seats and 60 aisle slots

  dim r as integer, i as integer, x as integer, y as integer

  const cBLACK = &c000000
  const cGREEN = &c00FF00
  const cWHITE = &cFFFFFF

  ' update display of current simulation time

  lblSimTime.text = "T = "+format(SimTime,"#0.0")

  ' draw aisle This is hard-coded for 60 aisle slots

  y=100
  for i= 1 to 60
    x = 15 * i
    if Aisle_Status(i) > 0 then g.ForeColor = &c000000 else g.ForeColor
=cWHITE
    g.FillOval x,y,10,25
    g.ForeColor = &c000000
    g.DrawOval x,y,10,25
  next i

  ' draw seats - this is hard-coded for 6 setas per row (3 left, 3 right)
  for r = 1 to 30
    x = 30 *r
    for i = 1 to 3
      ' Seats A-C (1-3) (bottom of screen)
      y = 100 + 30* i
      if Seats(r,i)>0  then g.ForeColor = &c00FF00 else g.ForeColor = cWHITE
      g.FillRect x,y, 20,25
      g.ForeColor = &c000000
      g.DrawRect x,y,20,25
      g.DrawLine x+12, y,x+12,y+25

      ' Seats D-F (4-6) (top of screen)
      y = 100 - 30* i
      if Seats(r,i+3)>0  then g.ForeColor = &c00FF00 else g.ForeColor =
cWHITE
      g.FillRect x,y, 20,25
      g.ForeColor = &c000000
      g.DrawRect x,y,20,25
      g.DrawLine x+12, y,x+12,y+25
```



```
    next i
  next r
```

End DRAW_CABIN

Method PASSENGER_ROW(PC as integer) as integer

```
  ' Returns 'row' part of passenegr seat code
  return PC \10
```

End PASSENGER_ROW

Method PASSENGER_SEAT(PC as integer) as integer

```
  ' Returns Passenger seat (1-9) part of passenger seat code
  return PC mod 10
```

End PASSENGER_SEAT

Method PASSENGER_SEATCODE(Row as integer, Seat as integer) as integer

```
  ' returns a passenger seatcode based on Row and Seat
  return row*10 + seat
```

End PASSENGER_SEAT

Method SEATING_TIME(Seatcode as integer) as single

```
    ' takes passenger seating code and determines the amount of time to seat
depending on who is already seated
  ' This routine takes the approriate values from the SeatingTimeTable array

  dim r as integer, s as integer, target_index as integer, aisle_index as
integer, middle_index as integer

  r = Passenger_Row(Seatcode)
  s = Passenger_Seat(Seatcode)

  select case s
  case 1 to 3  ' left aisle to window
```



```
    target_index = s
    return SeatingTimeTable(target_index , seats(r,1), seats(r,2) )

  case 4 to 6 ' right aisle to Window
    target_index = s-3
    return SeatingTimeTable(target_index , seats(r,4), seats(r,5) )


  end select
```

End SEATING_TIME

