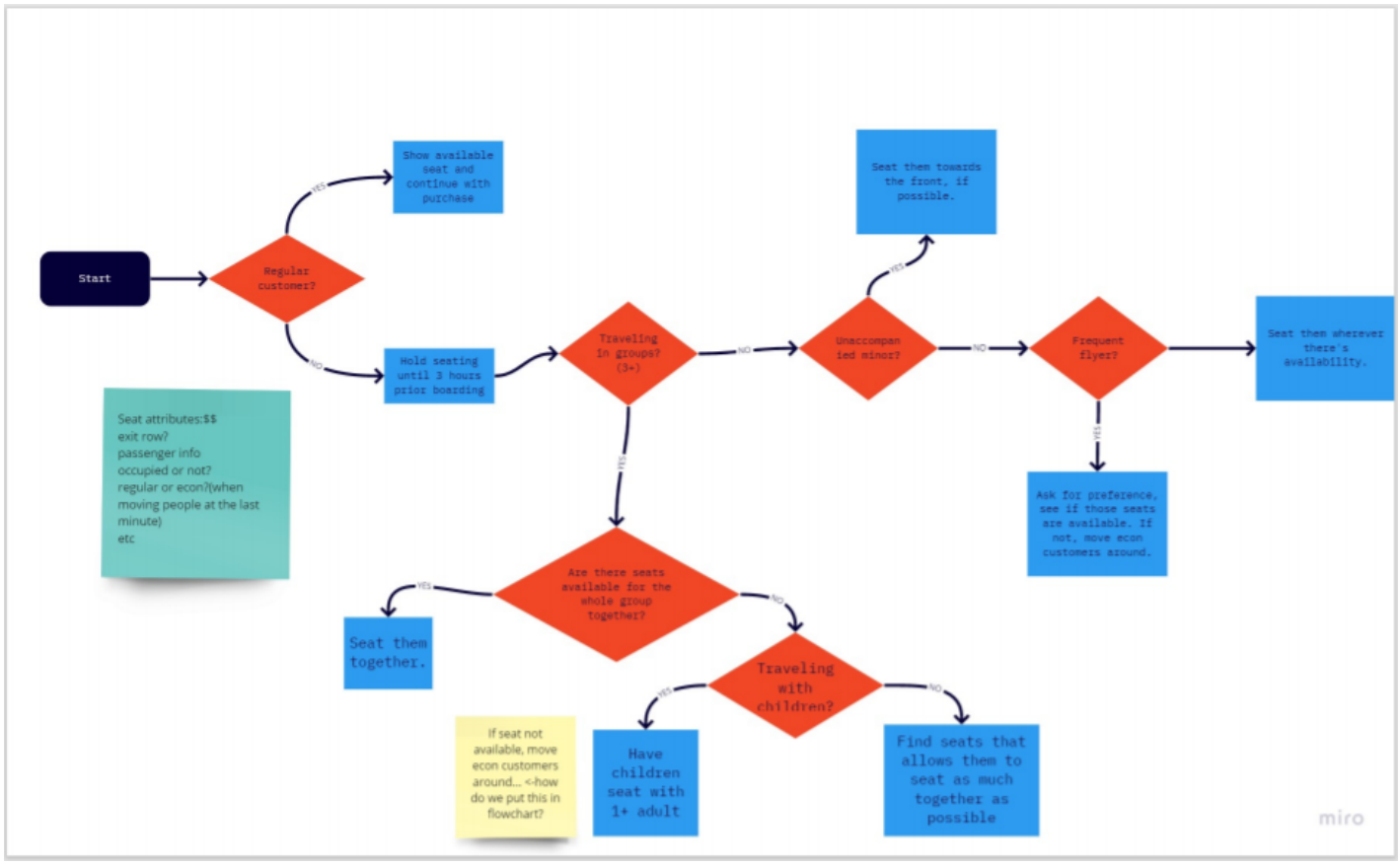
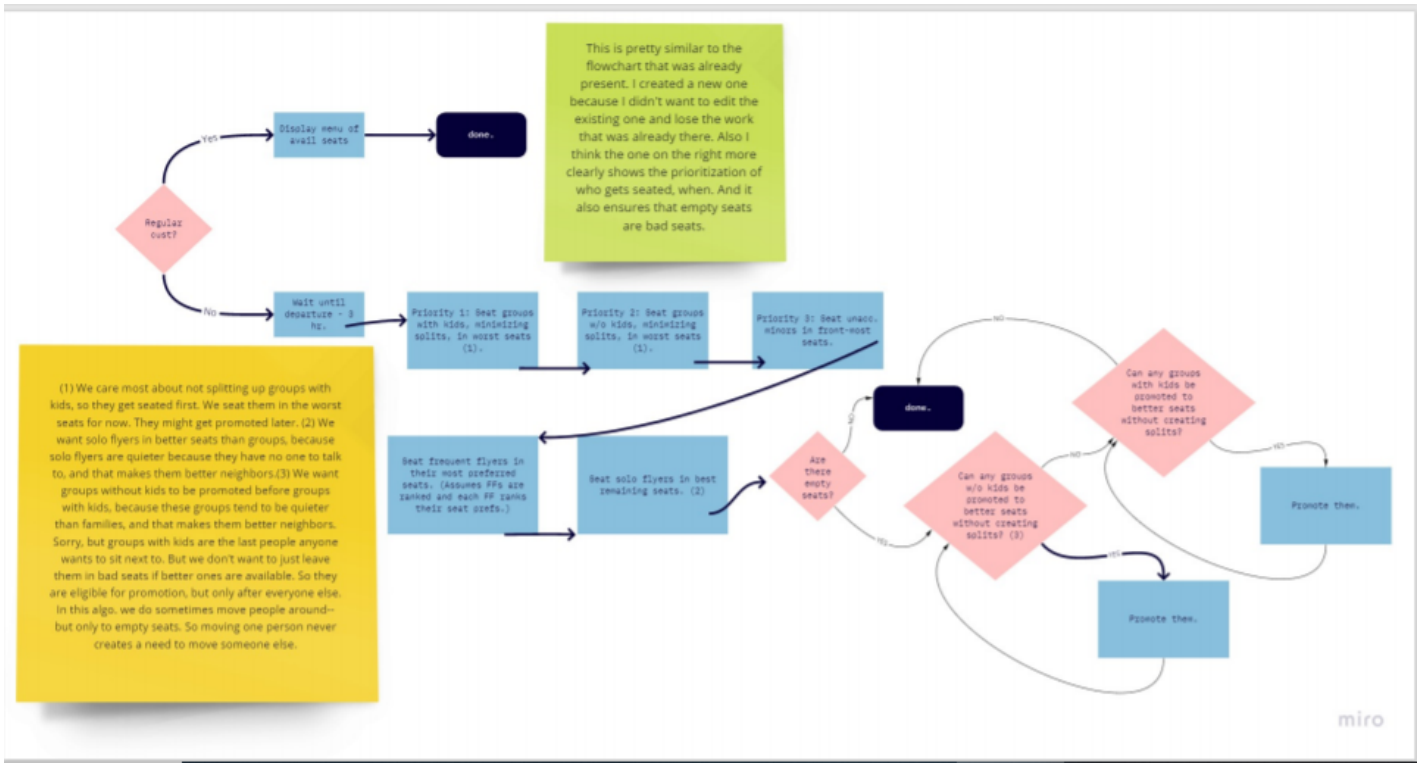
MIRO First Board (Sagmin)



MIRO Second Board (Daniel)



Double-click (or enter) to edit

Pat's original comment:

The passenger ranking procedure will essentially rank the passengers into one large priority queue with the following ranking strata: groups(2+) with children (highest rank) groups(2+) unaccompanied minors solo travelers (lowest rank) Two questions: Would you like to see any change in the "layering"? - (It's really easy to do so no trouble whatsoever.) For your purposes, would it be easier if I then put the groups into their own individual queues/arrays/list/etc according to their ranking so that you could work with one group at a time or will one long queue suffice?

Double-click (or enter) to edit

Pat's Summary:

Things to do:

1. Seat regular customers who pay extra to select their seat.
2. Create Passenger class with the following as possible attributes:

- priority number
- seat
- traveling with child indicator (Boolean)
- age (to identify unaccompanied minors)
- ticket number (first passenger to buy a ticket is 1, second is number 2…or something similar)
- pod number(used to identify the passengers travelling together as a group. (4 people traveling together would all have the same pod number. Each individual passenger would also have a pod number. Perhaps these can be related to the ticket number?)

3. Have the pods of passengers stored someway that allows them to be pulled one pod at a time. (The PODS DO NOT HAVE TO SORTED, just stored in a list/array/etc. The ranking function just calls each “pod” and assigns the same ranking/priority to each passenger in the pod)

Alonso's Original Pseudocode

Algorithm for seat assignment on an airplane:

- Regular customers choose their seat
- Economy customers are assigned seats Decisions:
- Economy customers will receive assignments 3 hrs. prior to departure

- Larger groups will receive priority
- Frequent Flyers will receive priority within the same size groupings

Data Structure(would best be a class) :

- 2D array of objects, columns will be translated to letters for display o Minimum information:

1. Occupied/Available (Boolean)
2. Priority (extra space, first class, exit row…) (Boolean)
3. Price (Float) – combined with #3 for full “value” of seat
4. Customer’s name (String)
5. Frequent Flyer? (Boolean)
6. Miles (Integer)
7. Group size (Integer)
8. Seats of other group members (1D array of seats).
   • Seats could be stored in a single 4 digit integer RRCC (row,row,column,column), easily decoded to find each seat.
   • Should be dynamic (linked list, ArrayList, etc.).
   • It will not change much, and the size is determined by #7.

- 1D Array of objects for customers without a seat assignment (Economy) o Minimum information:

1. Customer’s name (String)
2. Frequent Flyer? (Boolean)
3. Miles (Integer)
4. Group size (Integer)

- NumberOfSeats - Global (or attribute) – Integer. o Starts as the number of available seats on the airplane.
  o Decreased by the group size every time a ticket is sold Algorithm:

1. If NumberOfSeats <= GroupSize then “Sorry, plane is full!” or, “Only x number of seats are currently available.”
2. Else If Regular Customer: a) Display availability b) Record their choice (toggle the seat to occupied) c) Collect the rest of the information
3. Else If Economy Customer: a) Collect the required information
4. 3 hrs before departure: a) Start assigning Economy customers

```
### Pats code - Please dont erase

import math
from queue import PriorityQueue
```

```
podsize = 6       # the number of passengers traveling in a group
childcount = 0
ffmiles = 0       # frequent flyer miles of the group or individual traveling together
ticketnumber = 0  # a way to identify order passengers purchased tickets.
Customerdata = 1  ### value is to just simulate customer data found in the passengers profile
Childcheck = True ### value is to just simulate customer data found in the passengers profile
Age = 6           ### value is to just simulate customer data found in the passengers profile
Group = 8         ### value is to just simulate customer data found in the passengers profile


class Passenger(object):

    def __init__(self, name, priority):
        self.name = name
        self.priority = priority


passenger1 = Passenger("Maude", 5)
passenger2 = Passenger("Barleen", 9)
passenger3 = Passenger("David", 1)
passenger4 = Passenger("Halen" , 3)
passenger5 = Passenger("Van" , 2)
passenger0 = Passenger("Eddy" , 1)
passengers = [passenger4, passenger1, passenger5, passenger3, passenger2, passenger0]


### These lists will hold the passengers that have been sorted to them
GroupwithChildren = []
Group = []
Solo = []
Unaccompanied = []


### It takes in a 'pod' of passengers travelling together (solo or group size).
### and generates a ranking number based on fflyer miles, early ticket purchase,
### and category the passengers fall into. The same number is used for each
### passenger in the group as a means of keeping them together.

### It then does two things:
### 1. Stores the ranking in each passengers profile
### 2. Creates a list of the passengers in each category

for x in range(0, podsize):
    ffmiles = ffmiles + Customerdata
    if Childcheck:   ### counts number of children in group
        if podsize >1:                    ### Group with children
            childcount = childcount + 1
            Childcheck = True
            Group = 3000000
        else:                             ### Unacompanied Minors
            Group = 1000000
    elif podsize >1:                      ### Group
```

```
        elif podsize >1:                    ### Group
            Group = 2000000
        else:
            Group = 0                       ### Solo Travelers

    ### Generates the Ranking number

    Ranking = (Group + ffmiles + podsize + ticketnumber)


    ### Assigns the Ranknig to each member in the the 'pod'
    print ("Pod check")
    print ("Passenger and Ranking")
    for x in range(0, podsize):
        passengers[x].priority = Ranking
        print (passengers[x].name , passengers[x].priority)



    ### This section of code places every passenger into the appropriate list



    for x in range(len(passengers)):

        if passengers[x].priority < 1000000:
            Solo.append(passengers[x])

        elif passengers[x].priority < 2000000:
            Unaccompanied.append(passengers[x])

        elif passengers[x].priority < 3000000:
            Group.append(passengers[x])

        else:
            GroupwithChildren.append(passengers[x])



    ### A print to check that each member has the appropriate priority number.
    print ("Check to see if grouping worked")
    for x in range(0,len(GroupwithChildren)):
        print (GroupwithChildren[x].name,GroupwithChildren[x].priority)
```

```
Pod check
Passenger and Ranking
Halen 3000012
Maude 3000012
Van 3000012
David 3000012
Barleen 3000012
Eddv 3000012
```

```
# Izagma ALonso's code
# Modification of Seat class
class Seat(object):
    def __init__(self, r, c, value):
        letter = ["a","b","c","d","e","f"]
        self.occupied = False;
        # automatically creates the seat's location from the row & column pair
        self.loc = str(r+1) + letter[c]
        self.value = value

# creates the plane
def make_plane(rows, columns):
  # value defaults to $105
  plane = [[Seat(r,c,105) for c in range(columns)] for r in range(rows)]
  # add correct values by row and column - NOT DONE
  return plane

# prints the plane indicating available seats
def print_plane(plane):
  print("-----------------PLANE----------------------")
  # print the plane seats
  for r in range(len(plane)):
    for c in range(len(plane[0])):
      # add two spaces for the aisle
      if c == 3:
        print("      ",end=" ")
      # adjust for smaller row numbers
      if r < 9:
        print("",end=" ")
      print(plane[r][c].loc,end=" ")
      # -- indicates available, XX indicates occupied
      if plane[r][c].occupied:
        print("XX",end=" ")
      else:
        print("--",end=" ")
    print()


# Occupies the specified seat
def occupy_seat(plane,r,c):
  # Only occupy it if it is free,
  # Otherwise return false
  if plane[r][c].occupied:
    return False
```

```
    else:
      plane[r][c].occupied = True
      return True

  # occupy a row
  def occupy_row(plane,row):
    for c in range(len(plane[row])):
      occupy_seat(my_plane,row,c)

  # If all children traveling alone have been seated
  # the first row can be freed
  def unoccupy_seat(plane,r,c):
    plane[r][c].occupied = False

  # main
  ROWS = 20
  COLUMNS = 6
  #          0   1   2   3   4   5
  letter = ["a","b","c","d","e","f"]
  my_plane = make_plane(ROWS,COLUMNS)       # make a plane
  occupy_row(my_plane,0)                    # reserve the first row for children alone
  success = occupy_seat(my_plane,1,3)       # Occupy 2d
  success = occupy_seat(my_plane,10,5)      # Occupy 11f
  success = occupy_seat(my_plane,10,5)      # try to Occupy 11f again!
  if not success:
    s = str(11) + letter[5]
    print("Seat "+ s +" is already occupied.")
  print_plane(my_plane)                     # Print the plane's current Status
```

```
Seat 11f is already occupied.
-----------------PLANE---------------------
 1a XX  1b XX  1c XX         1d XX  1e XX  1f XX
 2a --  2b --  2c --         2d XX  2e --  2f --
 3a --  3b --  3c --         3d --  3e --  3f --
 4a --  4b --  4c --         4d --  4e --  4f --
 5a --  5b --  5c --         5d --  5e --  5f --
 6a --  6b --  6c --         6d --  6e --  6f --
 7a --  7b --  7c --         7d --  7e --  7f --
 8a --  8b --  8c --         8d --  8e --  8f --
 9a --  9b --  9c --         9d --  9e --  9f --
10a -- 10b -- 10c --        10d -- 10e -- 10f --
11a -- 11b -- 11c --        11d -- 11e -- 11f XX
12a -- 12b -- 12c --        12d -- 12e -- 12f --
13a -- 13b -- 13c --        13d -- 13e -- 13f --
14a -- 14b -- 14c --        14d -- 14e -- 14f --
15a -- 15b -- 15c --        15d -- 15e -- 15f --
16a -- 16b -- 16c --        16d -- 16e -- 16f --
17a -- 17b -- 17c --        17d -- 17e -- 17f --
18a -- 18b -- 18c --        18d -- 18e -- 18f --
19a -- 19b -- 19c --        19d -- 19e -- 19f --
20a -- 20b -- 20c --        20d -- 20e -- 20f --
```

```
  # Alonso
```

```
# Simple Main Logic - lots of pieces to write

# This is a simple routine, no priority or complex seating
# If it works we can ellaborate from here
# that is the best way to design, like Tofr said

# the plane must be created once and stored from use to use or stay in memory all the time
mu_plane=create_plane(20,6)
regular = print("Do you want to choose your seat? (yes or no)")
# add code to account for caps and y or n instead
if regular == "yes":
  print_plane(my_plane)
  seat = print("which seat do you want?")
  # check for valid seat
  # decipher into row and column
  occupy_seat(my_plane,r,c)
else:
  #collect passanger(s) information

if DEPARTURE - 3 == NOW:
  assign_economy()
  # sort first by children yes followed by no
  # special case child alone - seat on first row (should be reserved)
  # then sort by group size within the children
  # seat all passangers with children
  # make sure none is alone
  # sort the rest by group size
  # within same group size, sort by frequent flyer miles
  # now seat in order
```