Assigment 1

M Ahsan Al Mahir

November 13, 2020

1 Token Generator

Problem 1. Program a simulation system which the tokengenerator will use to get the waiting time estimation and booth assignment for custoomers and new boot opening.

a. Define the system state variables and units of the model?

Solution [a]**.** The system state is the overall state of the booths. The variables of this state include:

```
# booth variables
num_booth       = 3                         # initially, might increase or decrease
booth           = {i:[]}                    # list of customers assigned to booth i
customers       = []                        # sorted list of customers queue, the i'th
                                            # customer has id i
customer_object = Object(id = i,            # customer id
                         booth_no = b,      # assigned booth no
                         time_0 = time s,   # time of arrival
                         wait = time s)     # waiting time
```

b. What are the global variables of this system?

Solution [b]**.** The global variables of this system are the booth numbers.

c. Whate are the events of this system?

Solution [c]**.** There are several events of this system:

1. A new customer has arrived: this envokes the token generator, which will then do the following:
   a) find a token for that customer
   b) creater a customer object and add it to the customer array
   c) run a check if new booth needs to be opened, and opens one and adds to the booth array.
2. A customer has finished taking their service: then token generator will do the following:
   a) destroy that customer's object
   b) send the next customer to their assigned booth
   c) run a check if some booth is no longer required, i.e. it has been a while since that booth had any customers, and delete it.

3. A customer has waited for more than 54 minutes, or the total number of waiting customers has exceeded 20: then the token generator will do the following:

    a) immediately open a booth and reassign the newly opened booth to every customer who has waited for more than 54m.

d. Design the organization of your program.

Solution [d]. 1. At first, we will initialize by creating the necessary variables, and adding rules to their capacity.

2. Then we'll create a token generator program that will have several call functions.

    a) a customer has finished servicing: call `del(customerID)`.

    b) a new customer has entered: call `add(timeOfArrival)`

    It will also have a cronjob, that will periodically check for any overflow in the booth's capacity, and will open new booths according to the rules

3. Whenever a new customer arrives, the system will call the `add` function of the token generator. It will reassign the tokens

4. Whenever a customer finishes their service, the system will call `del` function to remove that customer from the array, and re-sort it.

e. Define the initial state of the system.

Solution [e]. The initial state will contain

1. Three booths, all empty

2. An empty array for customer objects

f. From the conceptual model, name the functions/methods that will be needed to model the system. Define the input and output parameters of these functions/methods.

Solution [f]. The variables of the system will be

```
# booth array holding the list of customer ids assigned to them
booth = {1:[], 2:[], 3:[]}
customer = []
time_now = 0 # will increment
```

A customer token is defined as

```
class TokenObject():
    def __init__(self, customerID, timeOfArrival, booth_no):
        self.id = customerID
        self.start_time = timeOfArrival
        self.booth_no = booth_no
```

The token generator object is the central part of the system. It will have the following functions:

```
def assignBooth(customerID, timeOfArrival):
    # search for an appropriate booth
    return booth_no

def createToken(customerID, timeOfArrival):
    booth_no = assignBooth(customerID, timeOfArrival)
    token = TokenObject(id=customerID,
                        start_time=timeOfArrival,
                        booth_no=booth_no)
    return token

def add():
    customerID = len(customer) + 1
    booth_no = assignBooth(customerID, time_now)
    customer.append(reateToken(customerID, time_now, booth_no))
    time_now += 1
    booth[no].append(customerID)
    reAssing()

def del(customerID):
    token = customer.pop(customerID)
    reAssign()
    return token

def reAssign():
    # This is where all the magic happens, where the generator reassigns
    # the booth numbers to each customer based on the given state
    # no input output, only changes the overall state

    # this methon also needs to update the customer IDs and rearrange the booth arrays

def openBooth():
    booth[len(booth)+2] = [] # create new empty booth
```

g. Which entities can have multiple instances? How to deal with these multiple instances?

Solution [g]. The `TokenObject` can have multiple instances. They will be stored in a sorted array, and whenever the customer array will change, the `reAssing()` function will be called and it will update all the customer IDs. It will also update the booth arrays.

h. Name the numerical methods you may have to use for different computations of this system. Mention, for which computations you will need these methods.

Solution [h]. The `reAssign()` will predict when to open a new booth by cheching the `scipy.stats.norm` function. It should be able to detect the trend in the arrival of customers to decide when to open a new booth.

2 Space Agency

a. Define the system state variables and units

Solution [a]. The system state variables are as following:

```
# Earth object
class earth:
    # contains:
    earth.R          = 6.378e6 * m               # radius
    earth.r          = 1.5e9 * m                 # gravitational pull range
    earth.g          = 9.8 * m / s**2            # gravitational acceleration
    earth.position   = [pos.x, pos.y, pos.z]     # earth's coordinate
    earth.velocity   = [vel.x, vel.y, vel.z]     # earth's velocity
    earth.w          = rad * s                   # rotational velocity
    earth.sun_face   = [x, y, z]                 # position of the point on earth
                                                 # that faces the sun,
                                                 # to find where the asteroid hits

# asteroid object
class asteroid:
    # contains:
    asteroid.mass    = kg                        # mass
    asteroid.pos     = [pos.x, pos.y, pos.z]     # position
    asteroid.vel     = [vel.x, vel.y, vel.z]     # velocity

# other system objects
class system:
    # contains
    sys.detect_R     =                           # detect radius
    sys.time_0       = 0 * s                     # time 0
    sys.time_now     = s                         # time now
    sys.earth        = earth                     # earth object instance
    sys.asteroid     = asteroid                  # asteroid object instance
```

b. State the organization of your program

Solution. The simulation will increment the time by `dt` and make changes to the system state variables. The rough structure should be like the following:

```
# the main simulate function, runs a while loop
def simulate(dt):
    while true:
        sys.time += dt
        sys.update(dt) # this function will update the system
        point = self.hit()
        if point:
            # calculate the distance from self.earth.sun_face to point
            return point

# methods under system object
def update(self, dt):
    self.earth.update(dt)
    self.asteroid.update(dt)

def hit(self):
    distance =  # check distance between earth's center and asteroid center
    if distance <= self.earth.R:
        return self.asteroid.pos
    return false

# method under earth object
def update(self, dt):
    self.position = []  # new position, cosidering the previous velocity constant
    self.sun_face = []  # change according to self.w

# method under asteroid object
def update(self, dt):
    self.vel = []       # update the velocity wrt earth's gravitational pull
    self.pos = []       # update the position for dt time
```

c. What are the assumptions you are making to model the system?

Solution [c]. There are several variables that we are getting rid of to make the computation simpler. Such as:

1. We aren't taking note of the earth's atmosphere. Due to the drag, the velocity of the asteroid is significantly different from what we are calculating. Also air resistance creates a lot of temperature, which might burn the asteroid away before it reaches the earth.

2. We aren't considering other astrological figures, such as the Moon, Jupiter or Mars, which might change the gravitational field around the astoroid, and significantly affect its trajectory.

d. Which numerical method you will use to find the velocity, $v_a(t)$ and the acceleration, $a_a(t)$ of the asteroid? Mention the SciPy function for this numerical method, what are the input and output parameters?

Solution [d]. We need to use the `derivative` module from `SciPy` package. We also need to compute 3D velocity and acceleration.

e. Is it possible to check how the distance between the asteroid and Earth are changing?

Solution. Sure, we just need to update the `stimulate` function:

```
x_axis = []
y_axis = []

def simulate(dt):
    while true:
        sys.time += dt
        sys.update(dt) # this function will update the system

        x_axis.append(sys.time_now)
        distance = # distance between sys.earth.pos and sys.asteroid.pos
        y_axis.append(distance - sys.earth.R)

        point = self.hit()
        if point:
            # calculate the distance from self.earth.sun_face to point
            return point
```

```python

plt.plot(x_axis, y_axis)
plt.show()
```

f. If we know the velocity and acceleration of the asteroid, is it possible to predict if the asteroid will attack Earth or if it will pass by Earth? For now, assume the Earth's position is fixed, describe how you can predict this?

Solution [f]. Yes it is possible. We run the simulation, and check if the distance between the asteroid and the earth's center is $\sim$ earth's radius or not. Also, since earth is not moving, we just need to update the asteroid's positions and velocity.

g. Write the functions/methods of your program to predict an attacking or passing by

Solution. The fucntion `hits()` in [b] works fine for predicting whether a asteroid has hit the earth or has passed by. The input variables are the earth's position, asteroid's position and earth's radius.

3 Zedi

3.1 a

i. Identify the system state variables of Madiha's model?

Solution [a.i]. The system state variables of her model should be:

```python
total_profit = 0         # total profit till now
current_sale = 0         # total sale till now
sale_rate    = q         # current sales rate unit per month, initially q
unit_cost    = c         # production cost per unit
unit_price   = R         # sale price per unit
```

ii. Identify the functions/methods Madiha has to implement to show this comparison. Mention the input and output parameters.

Solution [a.ii]. Matilda needs to simulate two functions, one for normal sales, and one for discounted sales. These are given below:

```python
system = # current state with the variables given before

def no_discount(months):
    while months > 0:
        profit = system.unit_price - system.unit_cost
        system.total_profit += profit * system.sale_rate
        months -= 1
    return system.total_profit

def discount(months):
    system.unit_price *= .75
    while months > 0:
        profit = system.unit_price - system.unit_cost
        system.total_profit += profit * system.sale_rate
        months -= 1
        system.sale_rate *= 3        # sales increases three times
    return system.total_profit
```

Madiha then can just compare the output of these two functions.

iii. For each functions identify necessary numerical methods Madiha might need.

Solution [a.iii]**.** Madiha only needs to do basic arithmatic in both functions.

3.2 b