Assigment 1

M Ahsan Al Mahir
ID: 20216007
ahsan.al.mahir@g.bracu.ac.bd

November 14, 2020

1 Token Generator

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

class customer_object:      # contains:
    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 id for that customer
    b) run a check if new booth needs to be opened, and opens one and adds to the booth array.

c) creater a customer object and add it to the customer 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.
   b) reassign booths to the customers

d. Design the organization of your program.

Solution [d]**.** Th organization of the simulation is as followed:

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.
5. After assigning a token, deleting a customer or opening a new booth, the token generator will call for a `reAssign()` function, that will loop over the customers and assign them to booths properly, and update their serial no.

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]. A customer token is defined as

```
class TokenObject():
    def __init__(self, customerID, timeOfArrival, booth_no):
        self.id = customerID                    # unique id
        self.start_time = timeOfArrival         # arrival time
        self.booth_no = booth_no                # assigned booth
```

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

```
# it is called when adding a customer, reassigns the queue
def assignBooth(customerID, timeOfArrival):
    reAssign()
    booth_no = # 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

# adds a fucntion to the end of the queue and reassigns the queue
def add():
    id = len(customer) + 1
    new_customer = createToken(id, time_now)
    customer.append(new_customer)
    booth[new_customer.booth_no].append(id)
    time_now += 1

# deletes a customer who has finished servicing
def del(customerID):
    token = customer.pop(customerID)
    reAssign()
    return token

# this method updates the customer IDs and rearranges the booth arrays
def reAssign():
    while checkBooth():
        openBooth()
        for people in customer:
            people.booth_no = findBooth(people)
            # remove people.id from its previous booth's array,
            # and add it to the new one
        if checkIfAlright():
            break

def checkBooth():
    # runs a check to see whether new booth need to be opened.
```

```
    # returns true if yes, and false otherwise

def findBooth(people):
    # returns the booth number this person can be assigned to

def checkIfAlright():
    # runs a check to see if the current system is stable
    # as in no person is waiting for more than 54 mins
    # and less than 20 people are waiting
    # returns true if stable, false otherwise

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 Will the space brick hit the earth

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   = [x, y, z]        # earth's coordinate
    earth.velocity   = [x, y, z]        # earth's velocity
    earth.w          = rad * s          # rotational velocity
    earth.mass       = 6e24 kg          # mass of earth

    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     = [x, y, z]        # position
    asteroid.vel     = [x, y, z]        # velocity

# other system objects
class system:                           # contains:
    sys.detect_R     = m                # asteroid detection radius
    sys.G            = 6.974e-11 N*m**2/kg**2
    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:
            # find position of the asteroid on earth w.r.t.
            # the self.earth.sun_face
            pos_on_earth = # calculate position
            return pos_on_earth

```

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


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

def system.vectorify(direction, magnitude)
    # returns a vector

# method under earth object
def earth.update(self, dt):
    self.position = []  # new position, += dt * self.vel
    self.sun_face = []  # change according to rotational velocity

# method under asteroid object
def update(self, dt):
    if self.distance > earth.r:
        self.pos += self.vel * dt
    else:
        self.vel += self.gravity * dt
        self.pos =      # update using physics formula

def distance(self):
    # returns the distance between self and earth

def gravity(self):
    direction = earth.pos - self.pos
    # make the unit direction
    magnitute = earth.mass * G / self.distance()**2
    return sys.vectorify(direction, magnitude)
```

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 will be 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, Jupitar 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. All other computations are basic arithmetic.

e. Is it possible to check how the distance between the asteroid and the 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

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 Organic Food, do we give discounts?

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:

```
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:

```
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

i. What should be the steps to validate this model? What will be updated after the validation?

Solution [b.i]**.** Madiha assumed that after the discount, the sales rate will be multiplied by three times. Which might be an over or under estimation. Before making an assumption, she should test it with actual data. She should study the data of other companies in the market and see how giving discount on the products affected their sales rate.

After her analysis, she will have a better estimation of the sales rate, which might better validate the simulation model.

Also, she needs to address the fluctuations of sales rate, seasonal demands and the availability of other non-GMO crops in the market in her model.

ii. What should be the steps of verification? What is the next step after verification?

Solution [b.ii]**.** To verify her testing she can do several things:

1. Test the simulation with different input data, and different crops.
2. Check of any bugs in her program by subprogramming.
3. Verify with other previously done simulations to see if her model is producing reasonable results

After verifying her model, she needs to do actual testing in the market with the discount. And she should modify her model based on those data.

3.3 c

i. What changes will you suggest to be made to your System state variables to update the model?

Solution [c.i]**.** Madiha should make a product object, that will hold product specific data, and then run the simulation on each product to find a discount rate that will increase the profit.

```
class product:                  # contains
    cost        = $             # production cost
    price       = $             # base price
    discount    = %             # discount rate

    popularity  =               # popularity modifier, 1 < x < 2
    demand      =               # variable based on factors, 1 < x < 2
    sales_rate  = unit/mounth   # base sales rate
    sales_mod   =               # sales rate modifier

    profit_tot  = $             # total profit in 12 months
```

She should make multiple instances of this class for each crop, and consider the various factors including public interest and seasonal demand to determine the sales modifier. And she should produce individual profit for each of these crops.

ii. What will be the new equations to simulate the profit? Are there any new functions/methods to implement? If yes, write the functions, mentioning the input/output parameters. Don't worry about being perfect, give a suggestion.

Solution [c.ii]**.** If Madiha takes all different modifiers into consideration, the equation for sales rate will change. A possible formula for the sales rate modifier would be

$$\text{sales\_mod} = \text{popularity} * \text{demand} * \text{discount}$$

And then the sales rate function would look something like

$$\text{sales\_rate}(t) = \text{sales\_mod}^t * \text{sales\_rate}$$

A function that takes in a product object and discount rate, and returns the sales modifier:

```
1 def sales_mod(product, discount):
2     pop = product.popularity
3     dem = product.demand
4     dis = discount
5     return pop * dem * dis
```

This function takes public interest, seasonal demand into consider, and then multiplies by the discount, the higher the discount, the higher the modifier.

Madiha needs to implement this equation in her `discount` function to find the total profit output.