Assignment 1 – Vacuum Cleaner

Utility Based Agent

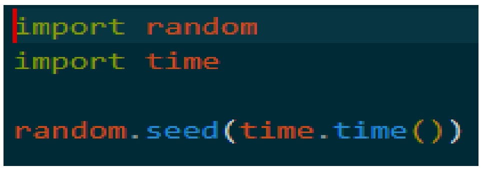
We proposed a utiliy based agent whose utility function is to earn as many points as possible. To do this, we develope an algorithm for our robot’s decision mechanism as follows:

- If the room is dirty, clean the room. (This option will give +1 point)
- If, the room is already clean, perform the noOp, goLeft or goRight. (If the agent is B, goLeft or goRight operations will decrease the total points by -0.5)

To perform noOp, goLeft or goRight operations, there is a decision algorithm : we calculate the estimated probabilities (for each room, for each 100 iterations), in each step we increase or decrease some priorities for each room with It’s estimated probabilities of getting dirty. At the end of some comparison between the priorities, the next step will be determined.

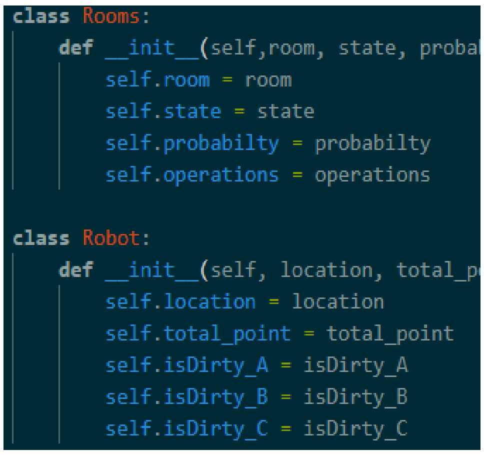
Source Code For Agent- A

We used random and time modules.



```
import random
import time

random.seed(time.time())
```

At the beginning of the code, we are setting the seed to the current time to make the generation of random numbers appear more unpredictable.

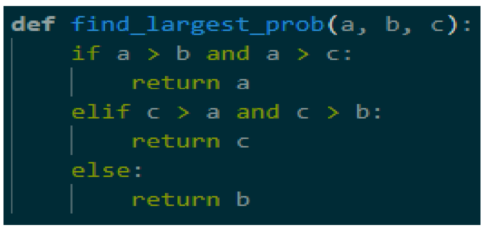


```
class Rooms:
    def __init__(self,room, state, proba
        self.room = room
        self.state = state
        self.probabilty = probabilty
        self.operations = operations

class Robot:
    def __init__(self, location, total_p
        self.location = location
        self.total_point = total_point
        self.isDirty_A = isDirty_A
        self.isDirty_B = isDirty_B
        self.isDirty_C = isDirty_C
```

We considered robot and rooms as an object, so we create classes for them.

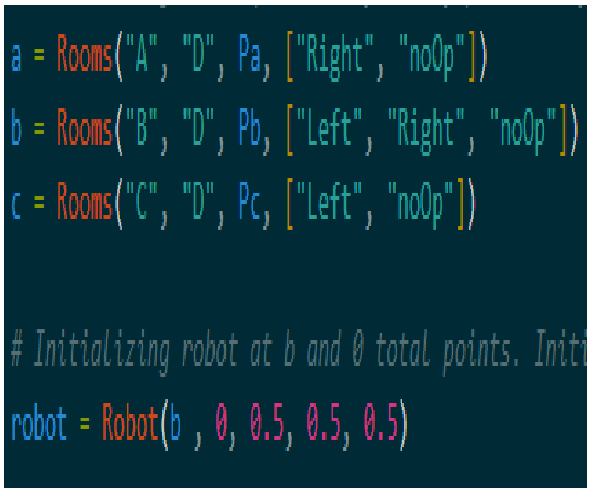
In the Rooms class, the room name (A , B or C), states (Dirty as D, Clean as C), probability (to get dirty), operations (noOp, goLeft, goRight). (suck operation performed if the room is dirty, so we don’t store it as an operation).

In Robot class, the location of the robot, total points and estimated probabilities of each room stored.



```
def find_largest_prob(a, b, c):
    if a > b and a > c:
        return a
    elif c > a and c > b:
        return c
    else:
        return b
```

We define a function that returns the max value. We used this function to determine which room has the biggest probability of get dirty.



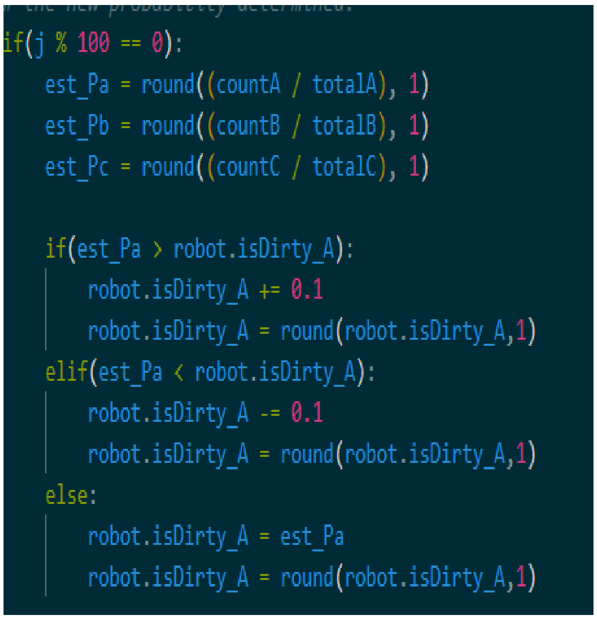
```
a = Rooms("A", "D", Pa, ["Right", "noOp"])
b = Rooms("B", "D", Pb, ["Left", "Right", "noOp"])
c = Rooms("C", "D", Pc, ["Left", "noOp"])


# Initializing robot at b and 0 total points. Initi
robot = Robot(b , 0, 0.5, 0.5, 0.5)
```

We initalize the rooms as dirty.

The given probabilities of rooms by user stored on P variable.

For each room, It has It's own operation. (There is no room on A's left, so only right or noOp operation performed, same as C's right.)

We initalize the robot in b and the initial estimated probabilities as 0.5 . According to learning process, the estimated probabilities will be calculated again.

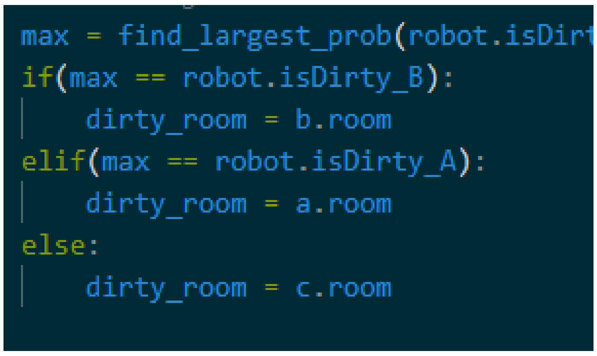


```
# the new probability determined.
if(j % 100 == 0):
    est_Pa = round((countA / totalA), 1)
    est_Pb = round((countB / totalB), 1)
    est_Pc = round((countC / totalC), 1)

    if(est_Pa > robot.isDirty_A):
        robot.isDirty_A += 0.1
        robot.isDirty_A = round(robot.isDirty_A,1)
    elif(est_Pa < robot.isDirty_A):
        robot.isDirty_A -= 0.1
        robot.isDirty_A = round(robot.isDirty_A,1)
    else:
        robot.isDirty_A = est_Pa
        robot.isDirty_A = round(robot.isDirty_A,1)
```

In each 100 iteration, the estimated probabilites recalculated. According to countX (counts the number of suck operation) and totalX (counts the number of visited that room), the division operation gives an approximation about probabilities.

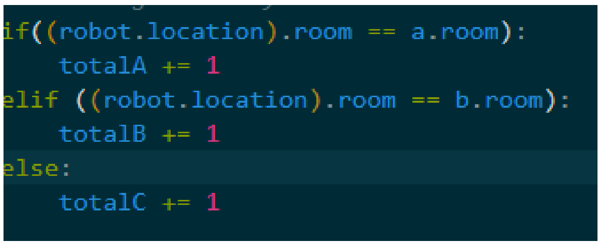
The round() operation performed for clean results to get 0.x results.

At the bottom if blocks, If the estimated probabilities results different than the stored in robot's probabilities, the increment or decrement operations performed.
This is the learning part of our code.



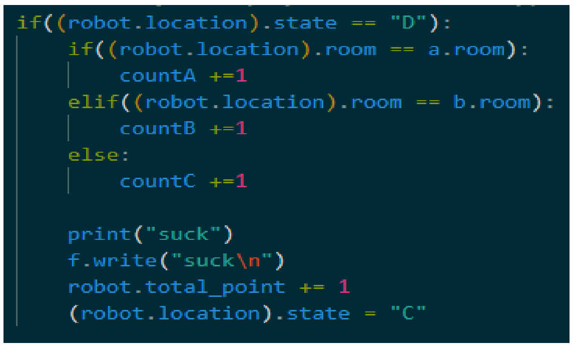
```
max = find_largest_prob(robot.isDirt
if(max == robot.isDirty_B):
    dirty_room = b.room
elif(max == robot.isDirty_A):
    dirty_room = a.room
else:
    dirty_room = c.room
```

The "main room" selected here. According to the main room (the probabilistic dirtiest room) we give some prioritisation.
The reason of why we do this, If the move operation performed according to only probabilities, the robot stays the same room for a long time.



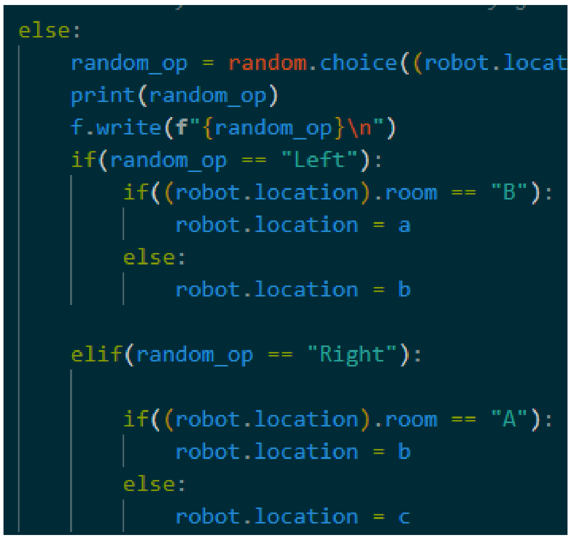
```
if((robot.location).room == a.room):
    totalA += 1
elif ((robot.location).room == b.room):
    totalB += 1
else:
    totalC += 1
```

The calculation of totalX (number of visited that specific room) performed here.



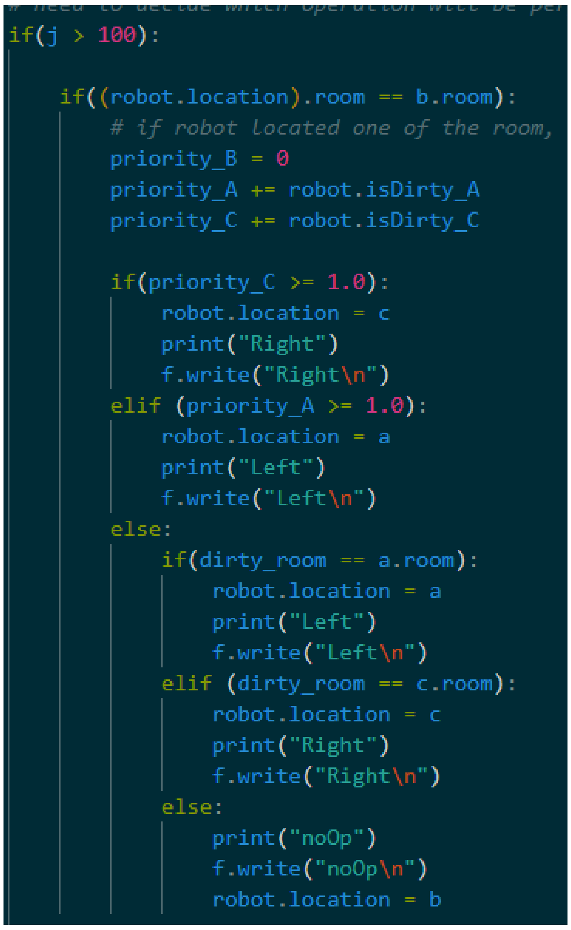
```
if((robot.location).state == "D"):
    if((robot.location).room == a.room):
        countA +=1
    elif((robot.location).room == b.room):
        countB +=1
    else:
        countC +=1

    print("suck")
    f.write("suck\n")
    robot.total_point += 1
    (robot.location).state = "C"
```

In this block, If the room is dirty, Suck operation performed.
Also the countX (the number of suck operation performed that specific room) calculated here.
At the end of this block, the total point incremented and the state changed to Clear.



```
else:
    random_op = random.choice((robot.locat
    print(random_op)
    f.write(f"{random_op}\n")
    if(random_op == "Left"):
        if((robot.location).room == "B"):
            robot.location = a
        else:
            robot.location = b

    elif(random_op == "Right"):

        if((robot.location).room == "A"):
            robot.location = b
        else:
            robot.location = c
```

In first 100 iteration, the robot will perform randomly operations. According to that random opearations, robot will have an estimation of probability of getting dirty for each room.
According to that estimations, It will update it's initial suggestions (0.5, 0.5 , 0.5)

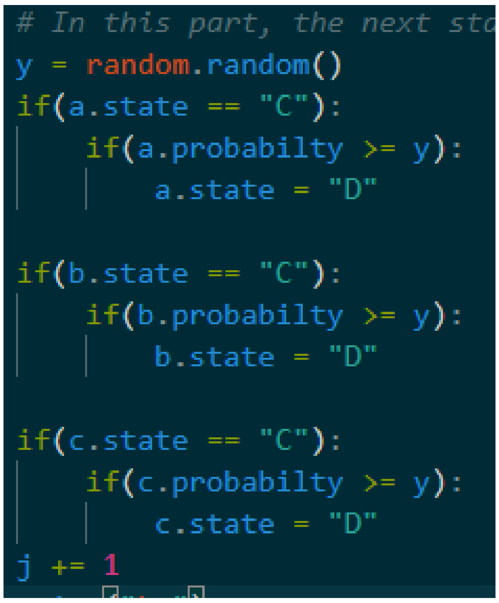


```
# need to decide which operation will be per
if(j > 100):

    if((robot.location).room == b.room):
        # if robot located one of the room,
        priority_B = 0
        priority_A += robot.isDirty_A
        priority_C += robot.isDirty_C

        if(priority_C >= 1.0):
            robot.location = c
            print("Right")
            f.write("Right\n")
        elif (priority_A >= 1.0):
            robot.location = a
            print("Left")
            f.write("Left\n")
        else:
            if(dirty_room == a.room):
                robot.location = a
                print("Left")
                f.write("Left\n")
            elif (dirty_room == c.room):
                robot.location = c
                print("Right")
                f.write("Right\n")
            else:
                print("noOp")
                f.write("noOp\n")
                robot.location = b
```

After 100 iteration, If the room is dirty, next operation will be selected in here.
If the room is in X, the priority of other rooms will be incremented by its estimated probabilities and priority of that X room decremented to zero because It's already cleaned.

According to priority comparisons, the next operation performed.

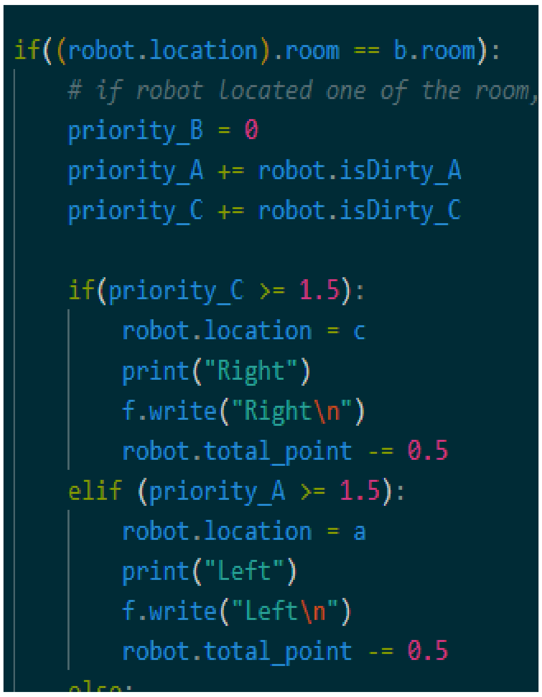
As a result, we use the estimated probabilities to give some priorities to the rooms, and according to that priorization, the next operation is selected.



```
# In this part, the next sta
y = random.random()
if(a.state == "C"):
    if(a.probabilty >= y):
        a.state = "D"

if(b.state == "C"):
    if(b.probabilty >= y):
        b.state = "D"

if(c.state == "C"):
    if(c.probabilty >= y):
        c.state = "D"
j += 1
```

This block determines the next operation randomly for first 100 iterations.

Source Code For Agent-B



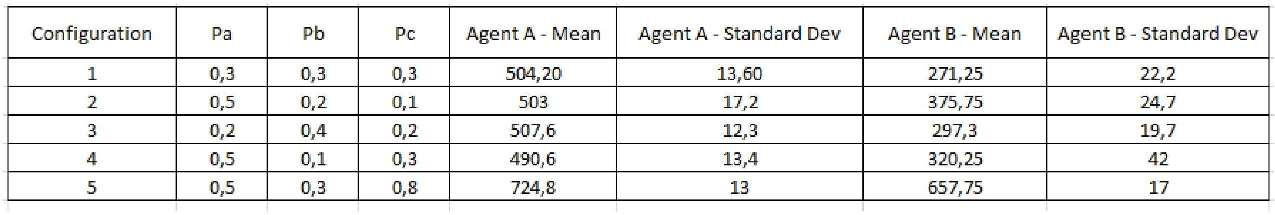
```
if((robot.location).room == b.room):
    # if robot located one of the room,
    priority_B = 0
    priority_A += robot.isDirty_A
    priority_C += robot.isDirty_C

    if(priority_C >= 1.5):
        robot.location = c
        print("Right")
        f.write("Right\n")
        robot.total_point -= 0.5
    elif (priority_A >= 1.5):
        robot.location = a
        print("Left")
        f.write("Left\n")
        robot.total_point -= 0.5
    else:
```

We use the same code with a few additions;

0.5 penalty points for move operations.

Higher priority criteria for move operations. The reason for this is that when we perform a move operation, at the end of this operation we have to gain a 0.5 points. So, we carry out our utility function (get as many points as)

Results of The Simulation



Configuration	Pa	Pb	Pc	Agent A - Mean	Agent A - Standard Dev	Agent B - Mean	Agent B - Standard Dev
1	0,3	0,3	0,3	504,20	13,60	271,25	22,2
2	0,5	0,2	0,1	503	17,2	375,75	24,7
3	0,2	0,4	0,2	507,6	12,3	297,3	19,7
4	0,5	0,1	0,3	490,6	13,4	320,25	42
5	0,5	0,3	0,8	724,8	13	657,75	17

We can see that from configure 5 that Agent A and Agent B perform better when given high probabilities.

Also, If we look at configure 3 and configure 4, the Agent-A performs slightly worse but Agent-B performs slightly better. The reason may be that Agent-B stays more in room A than Agent-B.

If we check configure 1 and configure 2, again Agent-B performs slightly less, but Agent-B performs better. Agent-B probably waits for room A to be sure that other rooms will get dirty.

If we look at configure 2 and configure 4, the decrease in Pb by 0.1 effect more than increase of Pc by 0.2. Both agents performed less. The reason could be that the room B is in the center. If the robot performs to go C, it needs to be sure that B is also dirty.