Finite State Machines

Jyotishko Banerjee

***Abstract*—**

"What is a Machine?" I am sure anyone hearing this question will remember a film from 2009. But this is not the question we ask today. Today we ask questions on a specific type of machine "Finite State Machines". So, What are Finite State Machines?

Finite State Machines are very familiar to us in our day to day lives. Bulbs, Television, Toasters, Air Conditioners, Traffic Lights, etc. Even characters in video games we play are Finite State Machines like Mario and David Jones.

Lets' take example of Mario. Initially, it stands signifying "Standing State", then we click Space and we bring in a "Transition" and the player goes into "Jumping State". We can make it move forward by pressing "-¿" that Transitions its state into "Moving Forward State". In this way, by our commands, we bring in the Transitions in Mario and he goes from one State to another. This gives an idea of how virtual Finite State Machines work.

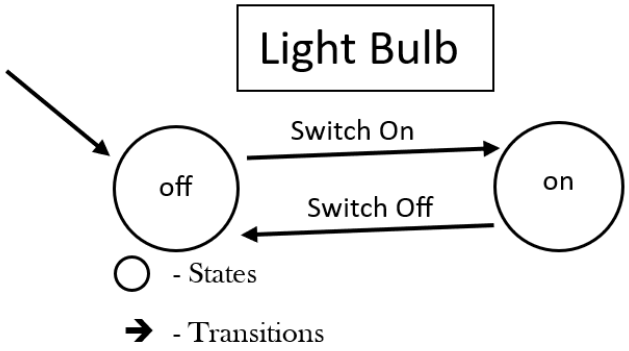
Let's take the practical example of a Light Bulb. A light bulb can have 2 states – off and on. Initially, bulb is in off state, then we switch it on, which brings in a transition that makes the Light Bulb switch on. Most Finite State Machines work more or less like this. Just as the machine gets complex, the number states and transitions keep increasing.

Overall, Finite State Machines are very useful for developing codes for complex virtual machines as we will see further.

I. INTRODUCTION

Finite State Machines are machines which have a fixed number of states and transitions occur where the machine goes from one state to another under specific conditions. Transitions may occur manually, like when we switch on or switch off a light Bulb, of may be automated, like Traffic Lights.

FSMs can be represented in state diagrams



Finite State Machine is used in programming to make virtual machines in proper organised manner. Usually, codes involve a lot of clumsy if-else statements that makes it difficult to analyse the code. FSM provides an alternative to this situation to manage it properly and make the code organised. Breaking down the code into various States and Transitions make it easier to work on them individually, edit them and update them easily and makes the machine's algorithm and flow understandable and executable.

Here, a Vending Machine is made using the Finite State Machine concept in Python. Making FSMs in Python requires strong knowledge of OOPs (Object Oriented Programming) – classes, subclasses, objects, attributes, etc.

II. PROBLEM STATEMENT

We are supposed to make a Vending Machine using Finite State Machine logic with a few criterias.

- Displaying names, codes and costs of each drink
- Users will enter code of drink they want
- Then user will enter the amount they will feed into the vending machine. If amount is exact as required, no change us returned, but if more is given, then change is given and drink is vended
- Every drink quantity is initialised to 50. Once all of 50 of a drink are exhausted, if user asks for it, display warning message to choose another message
- If all cans are exhausted, the user needs to type in REFILL to replenish the stocks and get juices again

Basically, we have to make a vending machine with various states and transitions. We are not supposed to use if-else statements, which is the purpose of using FSM. We are supposed to define the specific States and Transitions and bring them under FSM and then finally make changes between the States as and when necessary using Transitions. Below is the display as wanted.

Sl. no.	Drink	Code	Cost
1	Pepsi	PEPS	30
2	Mountain Dew	MDEW	30
3	Dr. Pepper	DPEP	50
4	Coke	COKE	20
5	Gatorade	GATO	20
6	Diet Coke	DCOK	30
7	Minute Maid	MINM	25
8	Tropicana	TROP	30

III. Related Work

I first had to learn OOPs about classes, subclasses, objects, attributes and others from the sources available.

Then I learnt about Finite State Machines from the resources given in the PDF, also from papers and videos about which I have mentioned in References. I mainly read and tried to understand examples how the States and Transitions were framed and used through classes, the function used, the operations made. Studying various examples like Light Bulb and Elevator had helped me understood a lot about making FSMs using OOPs in Python.

IV. INITIAL ATTEMPTS

Seeing the number lines an FSM code takes to be made, at first blew my mind. A Light Bulb FSM code took 70 lines of code. An elevator working in just 2 floors took 90+ lines of code from the examples I saw. I realised, the Vending Machine with so many states would be a challenging amount of task which might take a few hundred lines.

It was important to understand how exactly was the example codes framed and the logic. So, I understood them for days and developed a few small scale FSMs like Vending Machine without payments or Vending Machines with just 1 drink. Experimenting in various ways finally helped me to develop the final Vending Machine.

V. Final Approach

First, I define 2 classes of 'State' and 'System' which will be later used to define sub classes.

• I define class Drink under which I define instance attributes name, quantity, price and code

• I define class Drink System, a sub-class of System under which I define instance attribute 'drinks' which is a dictionary which contains codes as keys and Drink instances as items which are 8 in number in this problem.

• Also, a function get_code() with parameters self and curr_code to get the current code of the drink ordered and assign it to attribute self.curr_drink

• The next segment contains definitions of various states that the Vending Machine comes into

• Interface State fetches the code given by the user by using input function and analyses its correctness and proceeds further if and only if the code is valid. Otherwise, it puts the program in a loop until one of 8 correct code it entered

• Quantity_Checker() State checks for availability of the current drink by checking the quantity of the given Drink_System instance

• Q_Check State checks if all drinks have been expired of the given Drink_System instance

• Deploy_State() is responsible for printing "Deploying the Drink" or in practical manner, vending the drink after the clearance

• Next is Display_State() where an instance attribute takes an Drink System object and prints the table as given at the end of PROBLEM STATEMENT, along with an added feature of also displaying the current drink quantities, cause, why not?

• Next is state Fill with 2 functions fill() and refill(). fill() is used for filling for the first time once the machine starts and refill() is used for refilling if all the drinks get exhausted while the machine runs

• Now, both the above functions do the same thing, except 2 things are done extra by refill(). refill() checks for emptiness and then fills, also prints a lot statements that are necessary to be displayed to let the user know that the drinks are being refilled and when the refilling is completed and machine is ready to be used again

• In the next Segment, we take care of 'finances of the Vending Machine'

• I define the Payment_System(), which stores the total amount present in the Vending Machine

• I define Accept_Pay() State and a function under it to accept the payment from user. Also, if the user enters less amount, it demands the remaining payment until the total payment given is more or equal to the required payment. It also additionally displays how much more amount is required to be entered to enable vending the drink

• Next State is Give_Change, to give change if any which is checked using the given Payment_System instance, or if user has previously given exact amount, then just display No change.

• Then, comes the main class FSM which defines 2 instance attributes of payment system and drink system calling the previous classes with same name

• In the following parts it defines functions to call various States and Transition States

• Start_Machine() - starts the machine, or can be treated as Transitioning from off to Start mode

• Transition_to_Fill() - transitions to the State of filling up drinks

• Initial_Fill() - for filling up drink after starting the machine

• Transition_to_Ready_State() - to transition to Ready State for ordering drinks by the user

• Set_to_State_Display() - defines Display object to display the drink table

• Set_to_State_accept_code() - to call Interface and display the table

• Transition_quality_checker() - Transition to State of Checking drink quantity

• Set_to_State_quantity_checker() – defines Quality_Checker instance

• Transition_transfer() – Transitioning to Payment System

• Set_to_State_accept_payment() – to call Accept_Pay instance and accept the payment

• Transition_payment_change – transitioning to state to give change

• Set_to_State_give_change() - to define Give Change instance and give change

• The above 2 States also makes significant changes to the payment system

• Set_to_State_deploy_drink() – to vend the drink to user

• Transition_to_Initial() - to reset the vending machine to get orders again

• Transition_to_Stop() – To shut down the Vending Machine

After all this under FSM, we define a function operation_VM() which mainly defines the FSM object and calls the functions from FSM. From Starting the Vending Machine, Transitioning to Fill and Filling up Drinks then putting the Machine to a while loop that runs until the user wants and shuts it down when the user doesn't want anymore drink. The choice to get another drink or shut down the machine comes after vending drink each and every time. But if all drinks get over, then the user is prompted to type REFILL to refill all drinks. And of course, if a drink isn't available, and is ordered, the user is prompted to order another drink. The main task of operation_VM() is controlling the flow of calling States and Transitions in proper sequence.
Finally, in the main scripts, I call the operation_VM() function.

VI. Results and Observation

Screenshots of the typical outputs have been given below.

```
Starting Vending Machine....
PLease wait....
Vending Machine is On
Just a moment
Filling drinks...
Vending Machine is ready for use
```

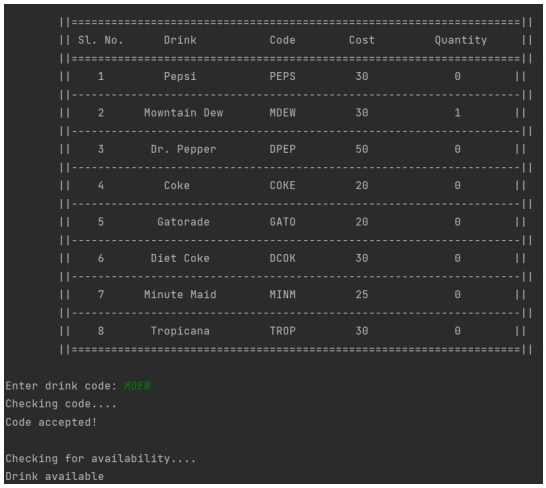
Sl. No.	Drink	Code	Cost	Quantity
1	Pepsi	PEPS	30	50
2	Mowntain Dew	MDEW	30	50
3	Dr. Pepper	DPEP	50	50
4	Coke	COKE	20	50
5	Gatorade	GATO	20	50
6	Diet Coke	DCOK	30	50
7	Minute Maid	MINM	25	50
8	Tropicana	TROP	30	50

```
Enter drink code: dcok
Checking code....
Wrong code. PLease re-enter code
Enter drink code: DCOK
Checking code....
Code accepted!

Checking for availability....
Drink available

Referring to Payment System.....

Enter amount Rs. 30 or more: 20
Enter Rs. 10  or more: 20
Payment accepted
Transacting change......
Change given: Rs. 10

Redirecting to Drink System......
Please wait...
Your drink is being fetched.....
Diet Coke has been deployed
Enjoy your drink!

Do you want to buy another drink? (y/n)n
Shutting down Vending Machine.....
```

An important point to be noted that in a lot of places, I have placed time lags to give the real feel of using a vending machine in real life.
Below screenshots give idea of a few explicit cases.
When a drink gets over –

Sl. No.	Drink	Code	Cost	Quantity
1	Pepsi	PEPS	30	0
2	Mowntain Dew	MDEW	30	1
3	Dr. Pepper	DPEP	50	0
4	Coke	COKE	20	0
5	Gatorade	GATO	20	0
6	Diet Coke	DCOK	30	0
7	Minute Maid	MINM	25	0
8	Tropicana	TROP	30	0

```
Enter drink code: PEPS
Checking code....
Code accepted!

Checking for availability....
Pepsi is not available. Please try another drink
```



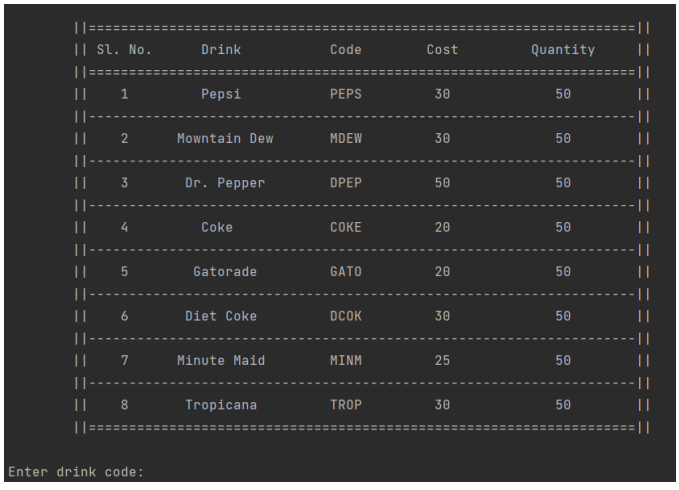
Sl. No.	Drink	Code	Cost	Quantity
1	Pepsi	PEPS	30	0
2	Mowntain Dew	MDEW	30	1
3	Dr. Pepper	DPEP	50	0
4	Coke	COKE	20	0
5	Gatorade	GATO	20	0
6	Diet Coke	DCOK	30	0
7	Minute Maid	MINM	25	0
8	Tropicana	TROP	30	0

```
Enter drink code: MDEW
Checking code....
Code accepted!

Checking for availability....
Drink available
```

After further payment and drink vending, all drinks get empty. Then. . .
Case when refill is needed –

Sl. No.	Drink	Code	Cost	Quantity
1	Pepsi	PEPS	30	0
2	Mowntain Dew	MDEW	30	0
3	Dr. Pepper	DPEP	50	0
4	Coke	COKE	20	0
5	Gatorade	GATO	20	0
6	Diet Coke	DCOK	30	0
7	Minute Maid	MINM	25	0
8	Tropicana	TROP	30	0

```
All empty. Enter 'REFILL' to refillrefil
Please enter 'REFILL' before proceedingREFIL
Please enter 'REFILL' before proceedingREFILL
Please wait while Drinks are being refilled.....
Drinks are refilled
Vending Machine is ready for use
```



Sl. No.	Drink	Code	Cost	Quantity
1	Pepsi	PEPS	30	50
2	Mowntain Dew	MDEW	30	50
3	Dr. Pepper	DPEP	50	50
4	Coke	COKE	20	50
5	Gatorade	GATO	20	50
6	Diet Coke	DCOK	30	50
7	Minute Maid	MINM	25	50
8	Tropicana	TROP	30	50

```
Enter drink code:
```

As we can see, after REFILL is typed, the Vending Machine is ready for use.
Please note that I have tried to account for the possible errors that the user might make while typing the required choices and commands, like putting in wrong code or putting less amount than required or not exactly writing 'REFILL'. These cases have been dealt with properly in the code. Also, I have displayed in the screenshots how the code acts when errors as explained above are done by the user.

VII. Future Work

Important Precaution - It is to be noted that I have deliberately provided **time.sleep()** at a lot of places to make delays to get the real experience of Vending Machine as far as possible. However if you choose to speed up the process, then please **# comment** out the time.sleep() lines.

Moreover, if you want to test out the Vending machine under **unconventional conditions**, then you need to do 2 things.

- In the class Drink_System(System) (line 24 to 34), change the quantities from 0 to any positive integer while expanding the dictionary 'drinks', or just keep them 0 to directly take the machine to 'ASK for a REFILL' state after the following step
- Then comment out machine.Initial_Fill() (line 301) under the function operation_VM(), this basically will not fill the machine with drinks after the machine starts but will carry the initial quantities assigned under the instance attributes of Drink_System(System)

As we have even seen before in the Abstract and INTRODUCTION sections, Finite State Machines can be used in wide range of practical applications. But while using FSM in programming, a few points are to be concerned about. Although FSMs are helpful in designing complex machines, but when applied for simple machines, it makes them complicated. A simple Light Bulb takes about 70 lines of code using FSM, whereas, a normal if-else statement would have taken around 10 lines of code. So as we see, for small machines, it might not be very useful but unnecessarily complicate things for us.

But definitely have a much greater application for complex machines. If we talk about ARK, drones can be treated as a complex FSM system. Stable, moving up, moving sideways, etc are its different states of motion. Working of its camera and sensors are also based on principles of FSM.

CONCLUSION

Overall, working on this project has taught me how to implement Finite State Machines to make complex systems like Vending Machine in Python language using OOPs.
FSMs have huge applications in creating complex machines in an organised and structured manner in general. For ARK's work, it has huge applicability, as explained in the previous section.

REFERENCES

[1] Haris Khan, Video #52 to #57, "Python Tutorials for Absolute Beginner in Hindi",CodeWithHarry, YouTube, 2019

[2] Trevor Payne, "Let's Learn Python #19 - Finite-State Machines (FSM)", Trevor Payne, YouTube, 2014

[3] Adam Sawicky, "Rise of the finite-state machines", Medium, 2020

[4] Mark Shead, "Understanding State Machines", Medium, 2018

[5] Mordechai Ben-Ari and Francesco Mondada, "Finite State Machines" chap 4 in book "Elements of Robotics", 2018