Finite State Machine Documentation

U Skanda Aithal

Abstract— **The problem is to implement a vending machine using finite state machine (FSM) logic in Python.**

The goal is to create a vending machine that allows users to select juices, enter the amount of money, and dispense the chosen juice along with any necessary change. The machine maintains an initial stock of 50 cans for each juice, provides warnings for low stock, and allows users to refill when all cans are exhausted.The implementation follows an FSM approach, where states and transition conditions are defined. Initially, the states were implemented as methods of a Machine class, but later it was found easier to define them as separate classes. The main class, Machine, manages the current state, transition, and execution of behaviors. In summary, the code provides a functional simulation of a vending machine using FSM logic in Python. It allows users to interact with the machine by making selections, entering money, and receiving the chosen juice and change. The vending machine displays available juices and prices, and users enter a four-letter code to select their desired drink. After entering the code, users input the amount of money they want to insert. If the amount matches the juice's cost, no change is returned. If the amount is higher, the machine dispenses the juice and provides the appropriate change.The implementation also handles stock management, displaying warnings when stock is empty and allowing users to refill the machine by entering "REFILL" when all cans are empty.

The FSM can be used in automating motion in the field of robotics. It is also widely used in the development of AI.

I. INTRODUCTION

The task is to implement a vending machine using finite state machine (FSM) logic in Python or C++. The machine should display available juices and their prices, allow the user to enter their choice and the amount of money. It should dispense the juice and return change if necessary. The machine should have an initial stock of 50 cans for each juice, display a warning when stock is low, and allow the user to refill when all cans are exhausted. The implementation should define states and transition conditions without using if-else statements. I used classes to define the states of the machine. Initially I tried making a class Machine in which the states were the methods of the instances of the class. But later I found defining them as classes was easier to implement and I went with it.

II. PROBLEM STATEMENT

The task is to create a vending machine using finite state machine (FSM) logic in either Python or C++.

A finite state machine is a computational model that consists of a set of states and transitions between those states based on certain conditions. In this case, the vending machine will have various states and transition conditions to simulate its behavior.

The vending machine should initially display a list of available juices along with their respective prices. The user will then enter a four-letter code to select their desired drink. After that, the user will enter the amount of money they want to insert into the vending machine. If the amount matches the cost of the juice, no change will be returned. However, if the amount is higher, the vending machine should dispense the juice and return the appropriate change to the user.

To handle the stock of juices, each variety of juice should be initialized with 50 cans. If the stock of a particular juice reaches zero, the machine should display a warning message prompting the user to select another juice. In the event that all the cans for all varieties of juices are exhausted, the user will need to type "REFILL" to replenish the stock before using the machine again.

To implement the vending machine using FSM logic, it is important to define the different states the machine can be in, such as "Idle," "Juice Selection," and "Money Insertion". I have assumed that the "Idle" itself is "Juice Selection". The transition conditions will determine when the machine moves from one state to another based on user input or internal events.

III. RELATED WORK

One of the approaches is to use simple conditional statements. But it will be cumbersome and also it will be difficult to add a new state. We can also define the process of refilling and returning change as new states. But I have included them in the states of "Juice Selection" and "Money Insertion".

IV. INITIAL ATTEMPTS

Defining states : initial attempt was to create a class "Machine" and define the states as the methods of the instances of this class

V. FINAL APPROACH

The task is to implement the simulation of a vending machine in Python using a finite state machine (FSM) approach.

The FSM is a computational model that consists of a set of states and transitions between those states based on certain conditions. In the context of the vending machine, the FSM allows us to model the different states the machine can be in (such as "EnterCode" and "EnterMoney") and define the transitions between these states based on user input.

The Machine class serves as the main class representing the vending machine itself. It has several attributes: states

*

and Trans (dictionaries to store the available states and transitions), trans (to keep track of the current transition), and currState (to maintain the current state). The setState method is responsible for setting the current state based on the given state name, while the transition method sets the current transition, updates the current state, and executes the transition. The Execute method is used to execute the behavior of the current state or transition, taking an optional code argument.

The transition class represents a transition between states. It has a constructor that takes the name of the next state as an argument and an Execute method that simply prints a message indicating the transition.

The EnterCode class represents the state where the user enters the item code. It contains an Execute method that displays all available items and their corresponding codes, prompts the user to enter a code, checks if the entered code is valid, and then calls the vend method of the Items class to handle the vending process.

The EnterMoney class represents the state where the user enters the money. It contains an Execute method that takes the entered code as an argument, prompts the user to enter the amount of money, calculates the change to be returned, and prints the change amount.

The Items class is a subclass of Machine and represents the collection of items available in the vending machine. It has a constructor that initializes the attributes of each item, including the serial number, drink name, code, cost, and quantity. It also appends each item to the allItems list and updates the Itemdict dictionary with the item's code and cost. The vend method handles the vending process by checking the availability of the item, reducing its quantity when sold, and handling the refill process if all items are out of stock. The repr method provides a string representation of an item.

The main function serves as the entry point of the program. It starts by creating an instance of the Machine class named MyFSM. Then, it initializes several items using the Items class constructor, providing the item's attributes such as the serial number, drink name, code, cost, and quantity. Each item is appended to the allItems list and its code and cost are added to the Itemdict dictionary. The super().init() line ensures that the Machine class constructor is also called for each item.

Next, the states and transitions for the vending machine are set. The states dictionary of MyFSM is updated to include the "EnterCode" and "EnterMoney" states, with corresponding instances of the EnterCode and EnterMoney classes. The Trans dictionary of MyFSM is also updated to include the transitions "EnterCodeTransition" and "EnterMoneyTransition", which are instances of the transition class.

After setting up the states and transitions, the initial state is set to "EnterCode" using the setState method of MyFSM. Then, the Execute method of MyFSM is called, which executes the behavior of the current state (in this case, the "EnterCode" state).

The program then enters an infinite loop, allowing the user to interact with the vending machine multiple times.

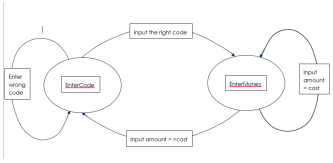

Fig. 1. FSM diagram

Inside the loop, the code of the current state is retrieved from MyFSM.currState.code. The transition method of MyFSM is called to transition to the "EnterMoney" state, followed by executing the state with the retrieved code as an argument. Then, the transition method is called again to transition back to the "EnterCode" state, and the state is executed again. This loop continues indefinitely, enabling the user to repeatedly enter item codes and money.

The Items class provides the vending machine with the functionality to handle items and their quantities. It maintains two important attributes: allItems, a list to store all the available items, and Itemdict, a dictionary to map item codes to their cost and quantity. The class constructor initializes the item's attributes and adds the item to allItems. It also updates Itemdict with the item's code, cost, and initial quantity.

The vend method of the Items class handles the vending process. It first calculates the total quantity of all items by iterating over Itemdict and summing up the quantities. If the count is zero, indicating that all items are out of stock, it prompts the user to type "REFILL" to replenish the stock. If the user types "REFILL", the quantity of each item is reset to 50, indicating a refill. If the quantity of the requested item is non-zero, it is decremented by one to signify that one item has been sold. If the quantity is zero, a message is displayed asking the user to choose another item.

In summary, the code provides a simulation of a vending machine using an FSM approach. It demonstrates how different states and transitions can be defined to handle the user's interaction with the machine, including entering item codes and money. The Items class manages the items available in the machine and handles the vending process. The code can be further customized and expanded to add more functionality, such as additional states, transitions, and item management features.

VI. Results and Observation

One of the major disadvantages of implementing a Finite State Machines is that the orders of state conversions are inflexible. I started with implementing the states as methods of the instances of a class. But planning it out further was a bit problematic and then I came across the idea of implementing the states as classes and it seemed easier to implement. So I chose to go with it. I have also used conditional statements inside states and in the process of vending, but I have avoided them in the implementation in the main function.

VII. Future Work

The program could have stored transaction history, it will help for analysis later on. We can also use a GUI to improve user interaction.

CONCLUSION

This problem gave an introduction to Finite State Machines. It took quite a lot of time to understand the working of FSMs and its implementation. It was a bit challenging as I was completely new to Finite State Machines. My initial attempt to implement states as methods also proved to be difficult. But then I found a way to implement them as classes.

In ARK, FSM models can be used in implementing automation of movements. The movements like moving straight, turning right or left can be considered as states and we can implement an FSM instead of simple conditional statements.

References

[1] https://www.cs.princeton.edu

[2] https://www.elprocus.com/finite-state-machine-mealy-state-machine-and-moore-state-machine/