2. Finite State Machine

Pranaya Chowdary

Abstract— **This documentation presents the implementation of a finite state machine (FSM) to simulate the logic of a vending machine. The FSM approach eliminates the need for complex if-else statements and allows for easy addition of new states and transition conditions. Finite state machines (FSMs) are widely used in various applications to model and control systems with a finite number of states.**
The objective is to create a robust and efficient system that can handle user interactions, provide appropriate responses, and manage stock levels. The vending machine allows users to choose from a list of available juices, enter the amount of money, and receive their selected drink along with any necessary change. The FSM logic handles stock management, invalid inputs, and refill requirements. The implementation is presented in Python, following the defined states and transition conditions, using a class-based approach.

I. INTRODUCTION

This research focuses on implementing an FSM to simulate the logic of a vending machine. The vending machine simulation allows users to select their desired drink from a list, input the amount of money, and receive the drink along with any necessary change. The FSM handles invalid inputs, out-of-stock situations, and refill requirements seamlessly.

II. PROBLEM STATEMENT

The problem entails implementing a vending machine using FSM logic. The vending machine should display a list of available juices along with their prices. Users can choose a drink by entering the corresponding four-letter code. After selecting a drink, users need to input the amount of money they will feed into the vending machine. If the entered amount is exact as the cost of the juice, no change is returned. Otherwise, the machine should return the appropriate change after vending the selected juice.

The vending machine should initialize each variety of juice with 50 cans. If the stock of a particular juice is exhausted (reaches 0), the machine should display a warning message to the user, suggesting they choose a different juice. If all the cans of every juice are exhausted, the user needs to type in "REFILL" to replenish the stocks before using the machine.

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

The objective is to define the states of the vending machine and devise suitable transition conditions between the different states to implement the required functionality. The solution should avoid the use of if-else statements and prioritize the ease of adding new states by defining a few new transition conditions.

III. RELATED WORK

Finite state machines have been extensively used in the domain of vending machines.One example of related work is the study by Qureshi et al.[1], where they implemented an FSM-based vending machine using Verilog hardware description language. Their work focused on the design and implementation of the vending machine at the hardware level, leveraging the FSM concept for efficient control and state management.

IV. FINAL APPROACH

Link to Google Colab Notebook

The final approach involves implementing an FSM-based vending machine using Python. The FSM consists of several states, including IDLE, CHOOSE_DRINK, INSERT_MONEY, RETURN_CHANGE, DISPENSE_DRINK, and REFILL. Each state is associated with specific functionalities and transition conditions.

The implementation utilizes a dictionary to define the different states and their corresponding functions. The vending machine also maintains a dictionary of available juices, each containing information such as name, price, and stock. The FSM transitions between states based on user inputs and predefined conditions.

```
class VendingMachine:
  def __init__(self):
    self.states = {
        'IDLE' : self.idle_state,
        'CHOOSE_DRINK' : self.
            choose_drink_state,
        'INSERT_MONEY' : self.
            insert_money_state,
        'RETURN_CHANGE' : self.
            return_change_state,
        'DISPENSE_DRINK': self.
            dispense_drink_state,
        'REFILL' : self.refill_state
    }
    self.juices = {
        'PEPS': {'name': 'Pepsi', 'price
            ': 30, 'stock': 1},
```

```
            'MDEW': {'name': 'Mountain Dew',
                'price': 30, 'stock': 0},
            'DPEP': {'name': 'Dr. Pepper', '
                price': 50, 'stock': 0},
            'COKE': {'name': 'Coke', 'price
                ': 20, 'stock': 0},
            'GATO': {'name': 'Gatorade', '
                price': 20, 'stock': 0},
            'DCOK': {'name': 'Diet Coke', '
                price': 30, 'stock': 0},
            'MINM': {'name': 'Minute Maid',
                'price': 25, 'stock': 0},
            'TROP': {'name': 'Tropicana', '
                price': 30, 'stock': 1}
        }
        self.current_state = 'IDLE'
        self.drink = None
        self.money = 0

    def run(self):
        while True:
            self.states[self.current_state]()

    def idle_state(self):
        #print all drinks
        print("\n")
        print('Available drinks:')
        print('CODE - NAME - COST (in Rs.)')
        print("--"*20)
        for code, juice in self.juices.items
            ():
            print(code, '-', juice['name'],
                juice['price'])
        self.current_state = 'CHOOSE_DRINK'
        for drink in self.juices:
            if self.juices[drink]['stock'] !=
                0:
                break
        else:
            self.current_state = 'REFILL'

    def choose_drink_state(self):
        drink = input("\nEnter code of the
            drink you want to buy: ")
        if drink not in self.juices:
            print("Invalid code entered.
                Please enter a valid code.")
        elif self.juices[drink]['stock'] ==
            0:
            print("Sorry, this drink is out of
                stock. Pick another one.")
        else:
            self.drink = drink
            self.current_state = 'INSERT_MONEY
                '


    def insert_money_state(self):
        price = self.juices[self.drink]['
            price']
        print("\nYour drink costs Rs.",
            price)
        self.money = int(input("Write the
            amount you are entering: Rs."))
        if self.money < price:
            print("Insufficient amount entered
                . Please try again.")
        else:
            self.money -= price
            if (self.money > 0):
                self.current_state = '
                    RETURN_CHANGE'
            else:
                self.current_state = '
                    DISPENSE_DRINK'

    def return_change_state(self):
        print("Returning change of Rs. ",
            self.money)
        self.money = 0
        self.current_state = 'DISPENSE_DRINK
            '


    def dispense_drink_state(self):
        print("\nDispensing: ", self.juices[
            self.drink]['name'])
        self.juices[self.drink]['stock'] -=
            1
        print("Enjoy your drink!")
        self.drink = None
        self.current_state = 'IDLE'

    def refill_state(self):
        print("\nAll drinks are out of stock
            . Please enter 'REFILL' to refill
            ")
        refill = input("")
        if (refill == 'REFILL'):
            for drink in self.juices:
                self.juices[drink]['stock'] = 50
            print("All drinks have been
                restocked!")
        self.current_state = 'IDLE'
```

The code is implemented using an object of vending machine, and runs on user inputs.

```
vendingmachine = VendingMachine()
vendingmachine.run()
```

V. RESULTS AND OBSERVATIONS

The implemented FSM-based vending machine was tested extensively to evaluate its performance and effectiveness. The results were compared with other approaches, such as if-else branching, to assess the advantages of using FSMs.

The FSM-based approach demonstrated several benefits. Firstly, it provided a more modular and maintainable code structure compared to if-else branching. The separation of states and transitions made it easier to add new functionality and handle complex interactions.

Additionally, the FSM approach improved the readability and comprehensibility of the code. The use of descriptive state names and transition conditions enhanced the understanding of the logic and facilitated future modifications.

However, it is worth noting that the FSM implementation also had some limitations. The system relied on user inputs and assumed correct behavior. Invalid inputs could lead to unexpected states and behavior. To mitigate this, proper error handling and user prompts were implemented. Systems that need an indeterminate number of states can't be modeled by FSM, so they're not suitable for every problem.

Overall, the implemented FSM-based vending machine achieved the desired functionality, effectively handling user interactions, stock management, and monetary transactions.

VI. FUTURE WORK

- Integration of graphical user interface (GUI): Enhancing the user experience by developing a GUI for the vending machine, allowing users to interact with the system through a visually appealing interface.
- Error Handling and Recovery: Enhance the FSM's error handling capabilities by incorporating error states and recovery mechanisms. This can enable the system to gracefully handle unexpected situations and recover from errors.
- Parallel and Concurrent FSMs: Explore techniques for designing and managing parallel or concurrent FSMs. This can be useful in systems with multiple concurrent tasks or in distributed systems where different components operate independently but need to coordinate their actions.

CONCLUSION

In summary, this paper presented the implementation of a finite state machine to simulate a vending machine. The FSM approach provided a modular and extensible solution, avoiding complex if-else statements and allowing easy addition of new states and transitions. The implemented vending machine successfully handled user interactions, stock management, and monetary transactions. Results highlighted the advantages of FSMs in terms of maintainability and readability. Overall, the FSM-based vending machine demonstrated the benefits of using FSMs for modeling and controlling complex systems.

REFERENCES

[1] Qureshi, Muhammad Ali, et al. "Design and Implementation of Vending Machine using Verilog HDL." 2011 2nd International Conference on Networking and Information Technology, IPCSIT. Vol. 17. 2011.

[2] https://github.com/kaushik3012/fsm-examples/blob/main/FSM.ipynb

[3] Theory of Computation With Finite State Machines