Finite State Machine

Owais Makroo

***Abstract*—This paper presents an improved solution to implementing a vending machine using Finite State Machine (FSM) logic. The vending machine allows users to select and purchase various drinks by entering a four-letter code and providing the required amount of money. The FSM approach eliminates the need for cumbersome if-else branching statements, providing ease of adding new states and transition conditions. The implementation maintains an inventory of drinks, tracks availability, and handles user interactions accordingly.**

I. INTRODUCTION

Vending machines are automated devices widely used in various settings to provide convenient drink access. This paper presents an improved solution for implementing a vending machine using FSM logic. The FSM approach enables a structured and scalable solution that simplifies the management of states and transitions.

Finite State Machines (FSMs) are mathematical models used to represent and control the behavior of systems with discrete states. In the context of a vending machine, the states represent different stages of the purchase process, such as idle, choose_drink, enter_money, dispense_drink, return_change, and refill. Transitions occur based on user inputs and internal conditions, moving the machine from one state to another.

II. PROBLEM STATEMENT

Implement a vending machine using Finite State Machine (FSM) logic. The vending machine should allow users to select drinks by entering a four-letter code and providing the required amount of money. If the entered amount matches the drink's cost, no change is returned. The machine should maintain an inventory of drinks and display appropriate warnings when a drink is out of stock or when all drinks are exhausted. The user should be able to refill all drink stocks before using the machine.

Requirements

1) Implement the vending machine using FSM logic to handle state transitions.
2) Avoid using if-else statements and utilize FSM principles for easy addition of new states.
3) Maintain a data structure to store drink information (code, name, cost, stock).
4) Implement methods for restocking drinks, retrieving cost and stock, and dispensing drinks.
5) Display warnings when a drink is out of stock or when all drinks are exhausted.
6) Allow users to refill drink stocks before using the machine.

Drinks Information

- Each drink is identified by a four-letter code, has a name, a cost, and an initial stock quantity of 50.
- The drink information is provided as a dictionary with keys: "Code", "Name", "Cost", and "Stock".

The implementation should fulfill these requirements and provide a working vending machine that can handle user interactions and manage drink inventory effectively using FSM logic.

III. VENDING MACHINE IMPLEMENTATION

Class Initialization The VendingMachine class initializes the drinks inventory using a Pandas DataFrame, which stores the drinks' codes, names, costs, and stocks. The *max_name_length* attribute represents the maximum length of the drink name for formatting purposes.

```
class VendingMachine:
    drinks_dictioinary = {
    "Code": ["PEPS", "MDEW", "DPEP", "COKE", "GATO",
         "DCOK", "MINM", "TROP"],
    "Name": [
        "Pepsi",
        "Mountain Dew",
        "Doctor Pepper",
        "Coke",
        "Gatorade",
        "Diet Coke",
        "Minute Maid",
        "Tropicana",
    ],
    "Cost": [30, 30, 50, 20, 20, 30, 25, 30],
    "Stock": [0, 0, 0, 0, 0, 0, 0, 0],
}
    def __init__(self) -> None:
        self.drinks = pd.DataFrame(VendingMachine.
            drinks_dictioinary)
        self.refill()
        self.max_name_length = 13
```

Refill Method The refill method restocks all drinks by setting the stock level to the specified *new_stock* value. Each drink's stock is individually updated using a loop, allowing inventory management flexibility. The method provides a message indicating the successful restocking of the vending machine.

```
    def refill(self, new_stock: int = 50) -> None:
        self.drinks.Stock = new_stock
        print("Refill Successful!!")
```

Change Stock Method The *change_stock* method updates the stock level of a specific drink based on the provided code and new stock value. If the new stock level is zero, the method displays a warning message indicating that the drink is exhausted.

```
    def change_stock(self, code: str, new_stock: int
        ) -> None:
        self.drinks.loc[self.drinks["Code"] == code,
             "Stock"] = new_stock
        while new_stock == 0:
            print(
```

```
                f"Stock of {self.drinks.loc[self.
                    drinks['Code'] == code, 'Name'].
                    values[0]} is EXHAUSTED!!"
            )
            break
```

Get Cost and Get Stock Methods The *get_cost* method retrieves the cost of a drink based on the provided code. The *get_stock* method retrieves the current stock level of a drink based on the provided code.

```
def get_cost(self, code: str) -> int:
    return self.drinks.loc[self.drinks["Code"]
        == code, 'Cost'].values[0]

def get_stock(self, code: str) -> int:
    return self.drinks.loc[self.drinks["Code"]
        == code, 'Stock'].values[0]
```

str Method The *__str__* method generates a formatted string representation of the vending machine's current inventory. The method iterates over the drinks DataFrame, excluding drinks with zero stock, and constructs the string with code, name, and cost information.

```
def __str__(self) -> str:
    return_string: str = (
        "Code".ljust(6, " ")
        + "Name".ljust(self.max_name_length + 3,
            " ")
        + "Cost".ljust(5, " ")
        + "\n"
    )
    for _, drink in self.drinks.iterrows():
        while drink["Stock"]:
            return_string = (
                return_string
                + drink["Code"].ljust(6, " ")
                + drink["Name"].ljust(self.
                    max_name_length + 3, " ")
                + str(drink["Cost"]).ljust(5, "
                    ")
                + "\n"
            )
            break

    return return_string
```

Get Drink Method The *get_drink* method allows users to select a drink by entering its four-letter code. It validates the code, checks the availability of the drink, and prompts the user for the required amount of money. If the user enters sufficient funds, the method dispenses the drink and returns any change due. It also handles scenarios where the drink is out of stock, providing appropriate error messages.

```
def get_drink(self):
    while (
        code := input("Enter the code (NOT case
            sensitive.): ").upper()
    ) not in VendingMachine.drinks_dictioinary["
        Code"]:
        print("The entered key was not
            recognized!! Please try again...")

    current_stock: int = self.get_stock(code)

    while current_stock:
        cost: int = self.get_cost(code)
        while (amount := int(input("Enter the
            amount: "))) < cost:
            print("Insufficicent funds!! Please
                try again...")

        self.change_stock(code, current_stock -
            1)
        pr_str = f"Here is your {self.drinks.loc
            [self.drinks['Code'] == code, 'Name
            '].values[0]}!!"
        while amount - cost > 0:
            pr_str = pr_str + f" And the change
                of {amount - cost}"
            break

        print(pr_str)
        return

    print("The requested drink is out of stock!!
         Please try again...")
    self.get_drink()
```

Main Function and Menu Options The *main* function implements the main control loop of the vending machine. The menu options are defined in the *menu_options* dictionary, mapping user inputs to corresponding methods. The user is prompted to enter a key, and the corresponding action is executed until an unrecognized key is entered.

```
menu = """Please select an option!!
'g': Get a drink.
'REFILL': Restock.
's': Print the menu.
'q': Quit!"""

menu_options = {
                "g": vm_1.get_drink,
                "REFILL": vm_1.refill,
                "q": quit,
                "s": lambda: print(vm_1),
                }


def main() -> None:
    print(menu)
    while (option := input("Enter the key: ")) in
        menu_options.keys():
        menu_options[option]()
    print("The entered key was not recognised!!
        Please try again...")
    main()


if __name__ == "__main__":
    main()
```

Here is a link to the complete code.

CONCLUSION

The improved solution presented in this paper demonstrates the effective implementation of a vending machine using FSM logic. The solution provides flexibility and scalability by utilizing a Pandas DataFrame for storing and managing the drinks inventory. The FSM approach simplifies the control flow and eliminates the need.

REFERENCES

[1] Adam Sawicki, "Rise of the finite-state machines, Medium, 2020
[2] Wikipedia, Finite-state machine
[3] Mark Shead, Understanding State Machines, Medium , 2018