DOCUMENTATION FOR ARK PERCEPTION TEAM TASK

TASK-2 Finite state machine

I.INTRODUCTION

The Vending Machine using Finite State Machine (FSM) logic is a program implemented in C++ to simulate a vending machine. It allows users to select a drink from a list, enter the amount of money, and receive the drink along with any necessary change. The implementation follows the principles of FSM, avoiding clumsy if-else statements and allowing easy addition of new states by defining transition conditions.

II. PROBLEM STATEMENT

The goal of this project is to implement a vending machine that can handle the following functionalities:

- The vending machine will display a list of available juices along with their prices.
- The user can enter the code for their desired drink.
- The user can enter the amount of money they will feed into the vending machine. If the amount entered is equal to the cost of the juice, no change is returned. Otherwise, the change is returned to the user after vending the juice.
- Each variety of juice is initially initialized to a stock of 50. If the stock of a particular juice is exhausted, a suitable warning message is displayed to the user, asking them to choose another juice.
- If all the cans of juice are exhausted, the user needs to type "REFILL" to replenish the stocks of all the juices before using the machine.
  The list of drinks is as follows:

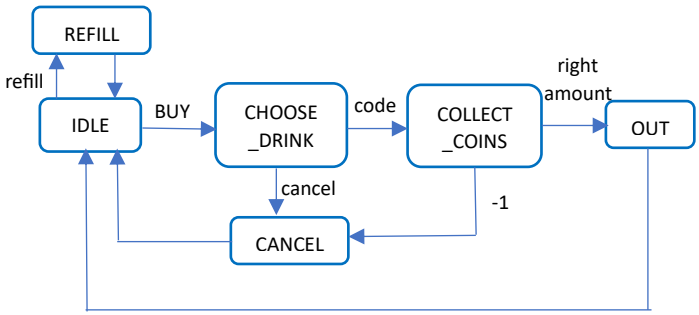
Sl. no.	Drink	Code	Cost
1	Pepsi	PEPS	30
2	Mountain Dew	MDEW	30
3	Dr. Pepper	DPEP	50
4	Coke	COKE	20
5	Gatorade	GATO	20
6	Diet Coke	DCOK	30
7	Minute Maid	MINM	25
8	Tropicana	TROP	30

III. RELATED WORK

We can use if/else conditioning again and again to achieve the desired result. We can also use a switch statement in an infinite loop with the different states as different cases. But this code will be very clumsy and hard to understand. Rather using the concept of Finite state machines is a better approach in terms of readability and maintenance of the code.

IV. INITIAL ATTEMPTS

The initial approach was to define different states and just specify what needs to be done in each state. Then I thought of connecting these states with different events considering the following diagram:



V. FINAL APPROACH

The final approach was implementing the code (in C++). The explanation of the code is as follows:

1. The code defines an enumeration class State that represents the different states the vending machine can be in. The states are IDLE, CHOOSE_DRINK, COLLECT_COINS, OUT, REFILL, and CANCEL.
2. A class named **VendingMachine** is defined. The constructor of this class initializes the initial state of the machine (**currentState**) as IDLE, the selected drink (**selectedDrink**) as an empty string, the inserted amount (**insertedAmount**) as 0, and the total count of available drinks (**totalCount**) as 400.
3. The '**run**' function represents the main loop of the vending machine. It runs indefinitely until the program is terminated. In each iteration, it calls the **displayCurrentState** function to show the current state of the machine and then calls the **handleState** function to handle the current state.
4. The members of the class are then defined. currentState stores the current state of the machine, selectedDrink stores the code of the drink selected by the user, insertedAmount stores the amount of money inserted by the user, and totalCount stores the total count of available drinks. Two unordered maps are also defined: drinkStocks stores the quantity of each drink, and drinkCosts stores the cost of each drink.
5. The displayCurrentState function displays the current state of the vending machine. It prints a separate line, the label "Current State: ", and then uses a switch statement to print the name of the current state. It also calls the displayDrinkMenu function when the state is IDLE to show the available drink options. The displayDrinkMenu function prints the drink menu, along with the

codes, costs, and quantities of each drink. It uses the drinkStocks unordered map to retrieve the quantities.

6. The handleState function handles the current state of the vending machine. It takes the current state as input and executes the appropriate function based on the state using a switch statement. Each state has a corresponding function that performs the necessary actions.
7. Then different functions are defined for performing specific actions in a certain state.
8. The **idle_state()** function represents the IDLE state of the vending machine. If there are no more drinks left (totalCount is less than or equal to 0), it prompts the user to refill the machine by typing "REFILL". If the user enters "REFILL", the state is changed to REFILL.
   If there are drinks available, it prompts the user to type "BUY" to proceed with buying a drink. If the user enters "BUY", the state is changed to CHOOSE_DRINK. Otherwise, it displays an "Invalid input" message.
9. The **search_drink()** function takes a drink code as input and searches for it in the drinkStocks unordered map. It iterates through an array of drink codes and compares them with the input code. If the code is found and the quantity of the corresponding drink in drinkStocks is greater than 0, it returns 1 to indicate that the drink is available. If the code is found but the quantity is 0, it returns 0 to indicate that the drink is out of stock. If the code is not found, it returns -1 to indicate an invalid code.
   The **select_drink()** function prompts the user to enter the code for the desired drink. It takes the input using take_input() and assigns it to the code variable. If the user enters "CANCEL", the state is changed to CANCEL. If the search_drink() function returns 1 (indicating that the drink is available), the selectedDrink variable is set to the input code, and the state is changed to COLLECT_COINS. If the search_drink() function returns 0 (indicating that the drink is out of stock), it displays a message informing the user that the drink is out of stock. If the search_drink() function returns -1 (indicating an invalid code), it displays an "Invalid input" message.
10. The **collect_coins()** function prompts the user to enter the amount of money to pay for the selected drink. It takes the input using cin and assigns it to the amount variable. If the user enters -1, indicating cancellation, the state is changed to CANCEL. If the entered amount is greater than or equal to the cost of the selected drink (drinkCosts[selectedDrink]), the insertedAmount variable is set to the entered amount, and the state is changed to OUT. If the entered amount is less than the required amount, it displays a message informing the user that the amount entered is insufficient and shows the required amount for the selected drink.
11. The **give_drink()** function represents the OUT state of the vending machine. It decrements the quantity of the selected drink in drinkStocks and decreases the totalCount variable to track the total number of available drinks. It then displays a message to the user asking them to collect the selected drink. If the amount inserted is greater than the cost of the selected drink, it displays the change to be collected. Finally, it resets the insertedAmount, selectedDrink, and currentState variables to their initial values, preparing the machine for the IDLE state.
12. The **refill()** function represents the REFILL state of the vending machine. It iterates over each pair in the drinkStocks unordered map and sets the quantity of each drink to 50, effectively refilling the stocks. It also sets the totalCount variable to 400 to represent the total number of available drinks. It displays a message confirming that the stocks have been refilled, and then sets the currentState to IDLE.
13. The **cancel()** function represents the CANCEL state of the vending machine. It displays a message indicating cancellation. Finally, it sets the currentState to IDLE.
14. The **main()** function creates an instance of the VendingMachine class and calls its run() method to start the simulation.

VI. RESULTS AND OBSERVATION

The Vending Machine using FSM logic has been successfully implemented. It allows users to interact with the machine, select drinks, enter money, and receive the drink along with any necessary change. The FSM-based approach provides a clear structure and avoids the need for complex if-else branching.

Observations:

- The program ensures that the user cannot select an out-of-stock drink.
- If the inserted amount is less than the required amount, the user is informed of the discrepancy.
- The program correctly handles the vending of the drink and returning the change, if any.
- When the stock of a particular drink is exhausted, the user is prompted to choose another drink.
- If all the drinks are exhausted, the user is required to refill the machine before further use.

VII. CONCLUSION

The Vending Machine using Finite State Machine (FSM) logic has been successfully implemented, providing a functional simulation of a vending machine. The FSM approach allows for easy addition of new states and transition conditions,

making the implementation modular and flexible. By adhering to FSM principles, the implementation avoids complex if-else branching, resulting in cleaner code and easier maintenance. The vending machine is capable of handling drink selection, money collection, and providing the drink along with any necessary change. Future enhancements and extensions can be explored to further improve the functionality and user experience of the vending machine.

Understanding the concept of finite state machines was a bit difficult in the beginning. After reading some documentations, it was found that FSMs make the system much more efficient and are actually easier to implement.

VIII. REFERENCES

- Implementing a Finite State Machine in C++ | Aleksandr Hovhannisyan
- https://www.codeproject.com/Articles/1087619/State-Machine-Design-in-Cplusplus-2