ATM Finite State Machine

Design & developed by:

1. Sunny Subash(M00833071)
2. Pouria Sabouri(M00737178)
3. Amir Ali Zahedi(M00770346)
4. Jenish Paudel(M00805644)

Submitted to:

Bob Fields

Introduction:

This Project is about an Automated teller Machine (ATM) provided by the banking authorities for their customers so that they can withdraw their cash.

Functionality:

This machine will be on stand-by mode and the customers are required to insert the bank card in order to use it. After that, the machine will instruct the customer to enter the pin and then it will check whether the pin entered is correct or not. If the pin entered is correct, then it will shift to the next window but if the pin is wrong then it will again ask customer to retry. During this process if the customer fails to enter the correct pin for three times, then the card will be blocked, and the customer will need to contact his/her bank. After the successful completion of pin entering process, the customers will be asked to enter the amount to withdraw cash. If the customer confirms the amount, then the machine will start to communicate with the server to check if there is enough amount or not to continue the transaction. If there is not enough amount on the customer's account, then the machine will display that you don't have enough balance to continue and that's why bank card will be released, and the machine will return to its initial state. But if there is enough amount on customer account then first it will release the card and afterward the machine will start to dispatch the cash entered. Once the cash is dispatched and collected by customer the machine will return to its initial state. But if the customer doesn't want to confirm that amount whether he can cancel the process or enter another amount.

Explanation

State Table

The picture below shows a code for our machine's state-table which is the base of our project because it tells our functions what is the name of our next state and what event will lead us to that state.

```
(define state-table
  '(
    (("standby mode" "insert card") "enter pin")
    (("enter pin" "pin entered") "checking pin")

    ;case 1
    (("checking pin" "wrong") "enter pin")
    (("checking pin" "3 times pin is wrong") "card is blocked")
    (("card is blocked" "releasing card") "card released")
    (("card collected" "next customer") "standby mode")

    ;case 2
    (("checking pin" "correct") "enter amount")
    (("enter amount" "amount entered") "confirm?")

    ;case 2A
    (("confirm?" "yes") "connected to bank")
    (("connected to bank" "checking balance") "balance checked"
    (("balance checked" "not enough balance") "card declined")
    (("card declined" "releasing card") "card released")

    ;case 2B
    (("balance checked" "enough balance") "card released")
    (("card released" "collecting card") "card collected")
    (("card collected" "dispaching cash") "cash collected")
    (("cash collected" "next customer") "standby mode")

    ;case 3
    (("confirm?" "no") "enter amount")
    (("enter amount" "cancel button pressed") "cancelled")
    (("cancelled" "next customer") "standby mode")
    ))
```

Next State

The picture below shows a code of a Function named next_state which checks the given arguments (state1 and event) are string and if they are string then check whether they are in the state table. If yes, then print the next state or else the function calls itself until the state table becomes empty.

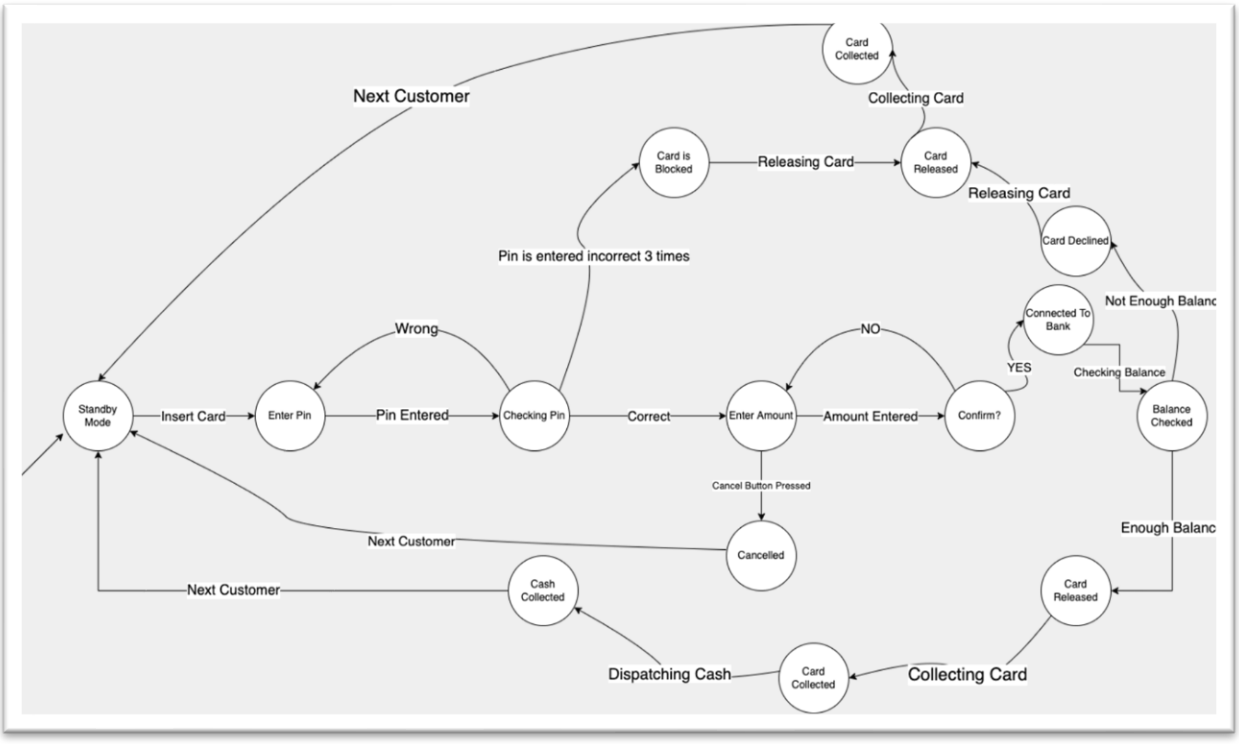
```
(define next_state (λ (State1 event state-table)
                     (cond
                       ((and (string? State1) (string? event))
                        (cond
                          ((empty? state-table) #f)
                          ((and
                            (equal?  State1  (first(first (first state-table))))
                            (equal?  event   (first (reverse (first (first state-table)))))
                            (first (reverse (first state-table)))))
                          (#t
                           (next_state State1  event (rest state-table)))))
                       (#t #f))))
```

Run Sequence

The picture below shows a code of a Function named run-seq which have 3 arguments with format (run-seq "initial-state" '(event-seq) state-table) which will print the sequence of latest states until the list of event sequence becomes empty

```
(define run-seq (λ (init-state event-seq state-table )
                  (cond
                    ((empty? event-seq) #f)
                    (#t
                     (let ([latest-state  (next_state init-state (first event-seq) state-table)])
                       (println latest-state) (sleep 1)
                       (run-seq latest-state (rest event-seq) state-table)
                       (sleep 0)
                       )))))
```

Graphical Representation of Our FSM



Sobs

10,12,21,113,107,217,14,7,8,101,102,143

Sob 14

Within the past few years, the number of fraud cases with respect to ATM has been increased. With the cash of the individuals moving towards the computerized platform, ATM stealing has gotten to be an issue that has eventually led to a worldwide objection. The present review examines the serious repercussions of ATM card cloning and the relation to privacy, ethical and legitimate concerns. The preventive measures which must be be taken and embraced by the government specialists to moderate the issue.
(https://pubmed.ncbi.nlm.nih.gov/29717470/)

THANK YOU