Introduction

this a report that details the procedure and steps of designing and building a Twiddle Lock ,A Twiddle Lock is a digital lock system that runs in two modes ,the secure mode and the unsecure mode ,it uses the raspberry pi model 3 B+ connected to a potentiometer and two push buttons to get inputs from the user. For the user to enter the password he/she must first press one of the two buttons to choose whatever mode he/she wants to operate the lock in.

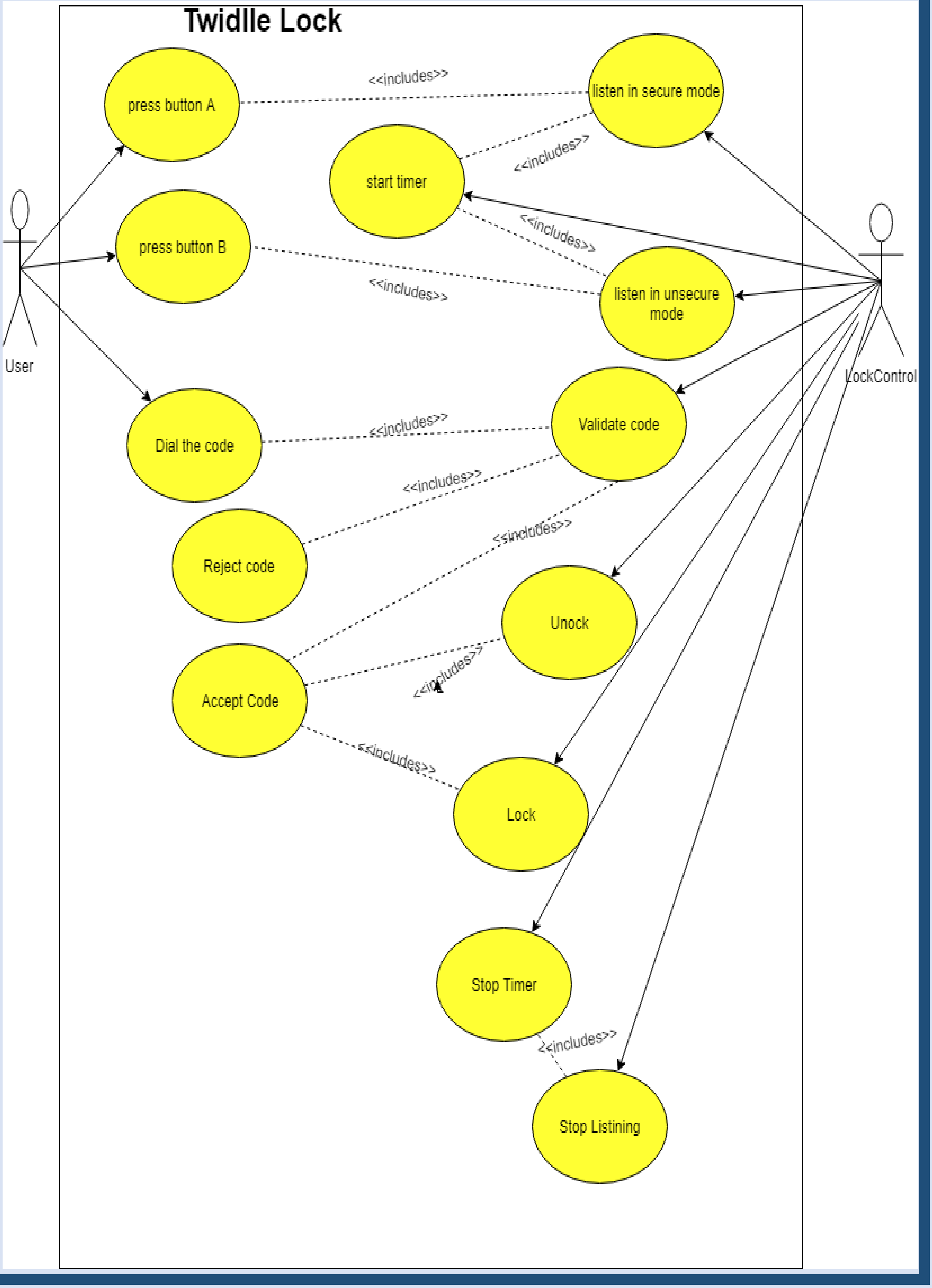
Once any button is pressed the countdown timer of forty seconds starts , when the timer hits zero it stops taking inputs from the user , if the timer has not ran out ,the use can enter inputs by fully turning the potentiometer fully clockwise or anticlockwise then the system will register the direction and the duration it took to fully turn the potentiometer ,if the user pause for more than three seconds the system checks if the register values are correct or not , when the values are correct the system displays a “correct password” and stops the counter then unlocks/locks else it displays “incorrect password “message and continues counting down…this is repeated until the timer hits zero ore the password is correct. This report details this is done by detailing the Requirements, Specifications ,   Design, Validation, Implementation ,Validation and Performance.

Requirements

The twiddle locks works in the following way:

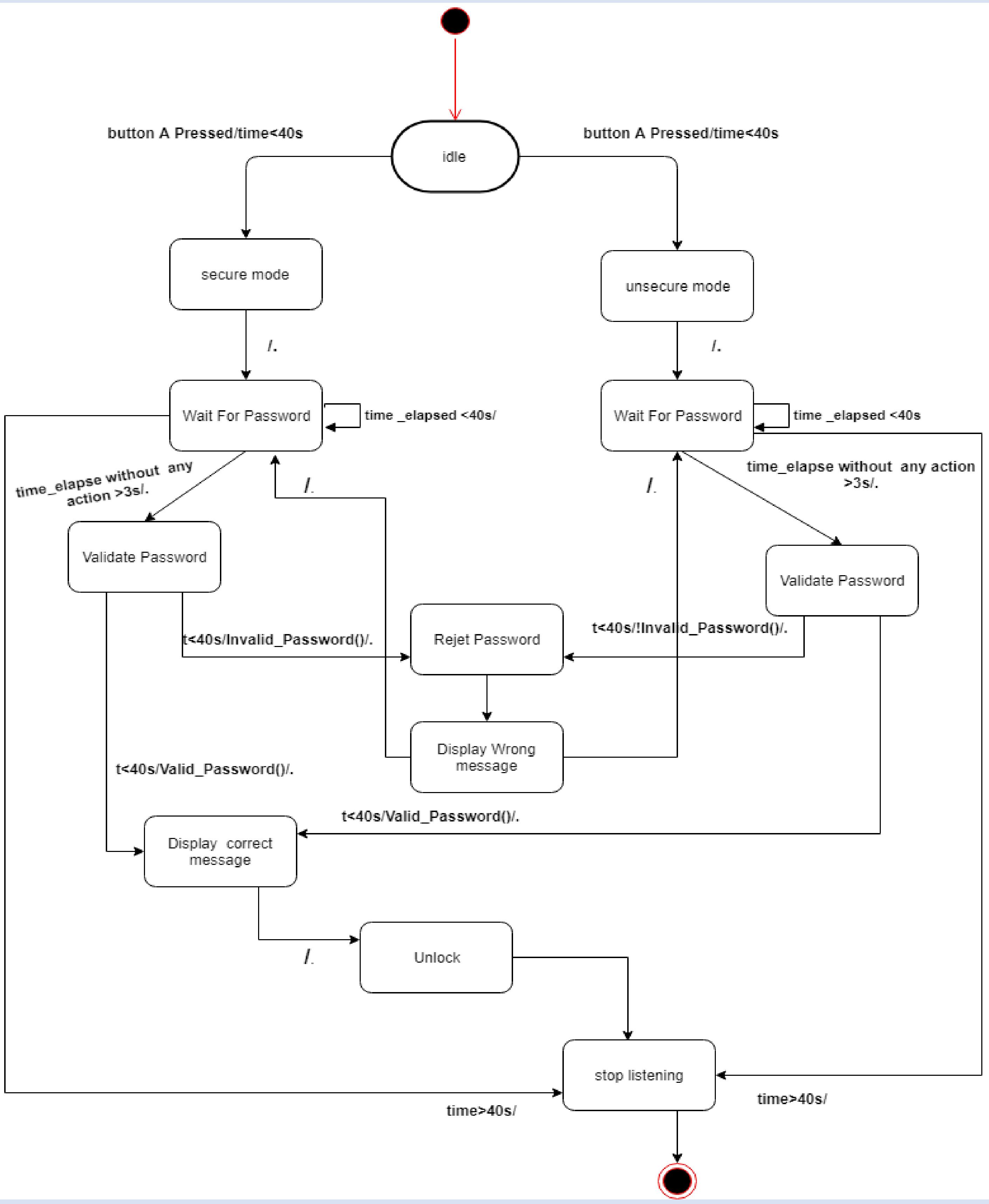
- Has two modes, secure and unsecure
- Two push buttons to choose on the modes
- Potentiometer to dial the passwords
- A count down timer to limit time listening for code
- A count up timer to check how long it took to rotate the pot and dial it
- A validating timer that validates the codes after three seconds without any activity from the user
- When the countdown timer is up it stops listening for codes

The below use case Diagram goes into detail on how the Twiddle lock works


Twidlle Lock
User
LockControl
press button A
press button B
Dial the code
Reject code
Accept Code
listen in secure mode
start timer
listen in unsecure mode
Validate code
Unock
Lock
Stop Timer
Stop Listining
<<includes>>

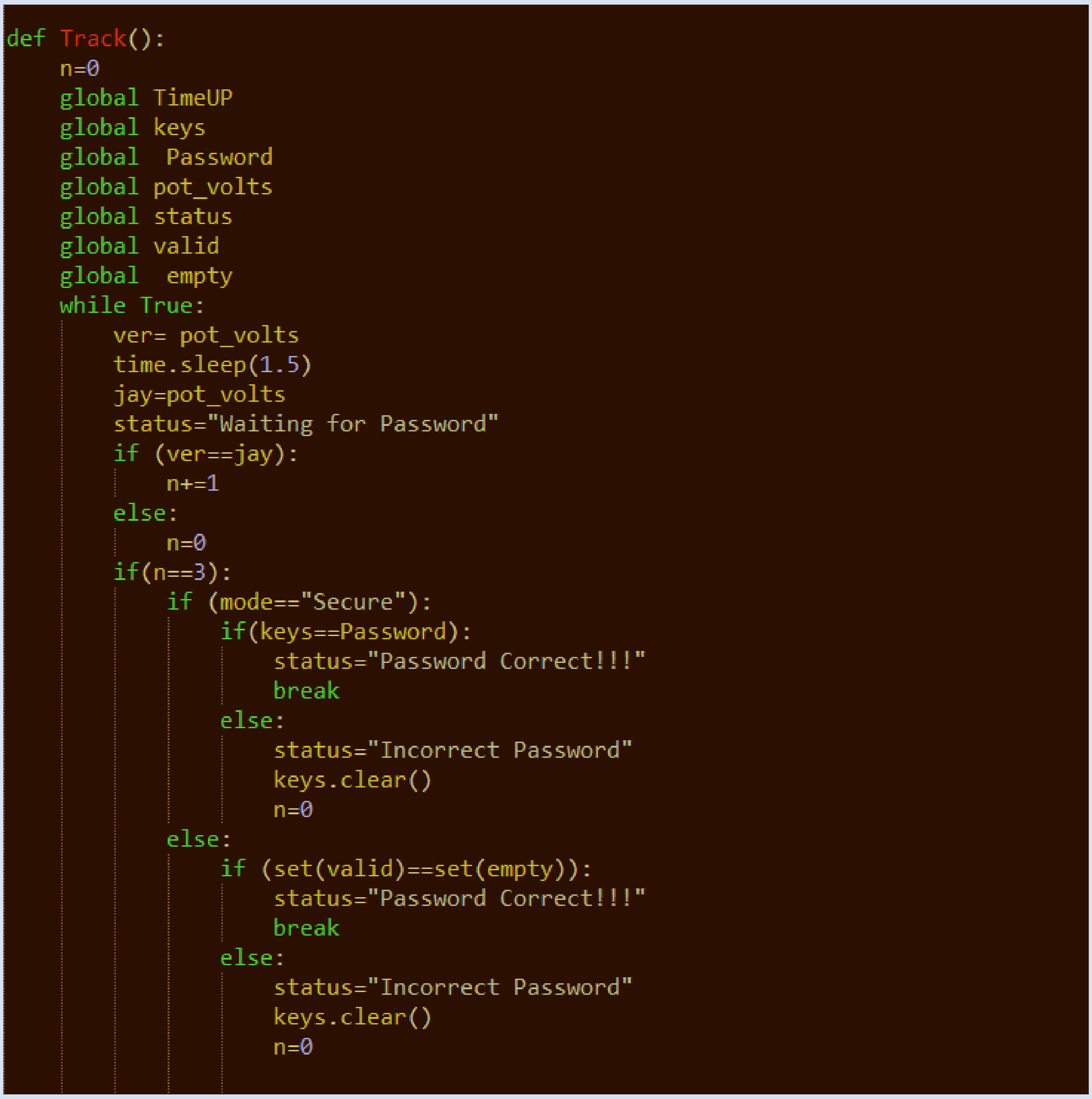
Specifications and Design

The UML diagram shows the main operation of the Twiddle lock.



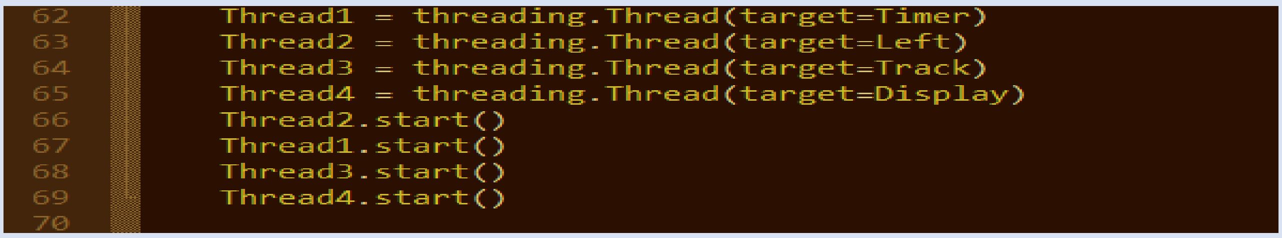
Implementation

Since I am designing a lock one of the most important thing was to make sure that the keys (Password)entered are correct the following code snippet shows how this was accomplished



```
def Track():
    n=0
    global TimeUP
    global keys
    global  Password
    global pot_volts
    global status
    global valid
    global  empty
    while True:
        ver= pot_volts
        time.sleep(1.5)
        jay=pot_volts
        status="Waiting for Password"
        if (ver==jay):
            n+=1
        else:
            n=0
        if(n==3):
            if (mode=="Secure"):
                if(keys==Password):
                    status="Password Correct!!!"
                    break
                else:
                    status="Incorrect Password"
                    keys.clear()
                    n=0
            else:
                if (set(valid)==set(empty)):
                    status="Password Correct!!!"
                    break
                else:
                    status="Incorrect Password"
                    keys.clear()
                    n=0
```

This function is ran under a thread so that it runs in the background without slowing down the code because it has high time requirements

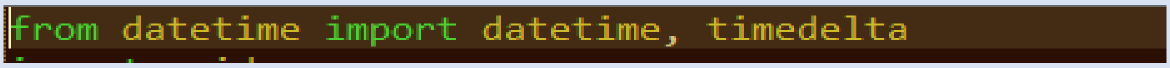


```
    Thread1 = threading.Thread(target=Timer)
    Thread2 = threading.Thread(target=Left)
    Thread3 = threading.Thread(target=Track)
    Thread4 = threading.Thread(target=Display)
    Thread2.start()
    Thread1.start()
    Thread3.start()
    Thread4.start()
```

In line 64 we initiate a thread for the function and then we start the thread in line 68, what the function does is that it checks if the user has paused for more than 3 seconds if that's the case then It validates the password entered by the user

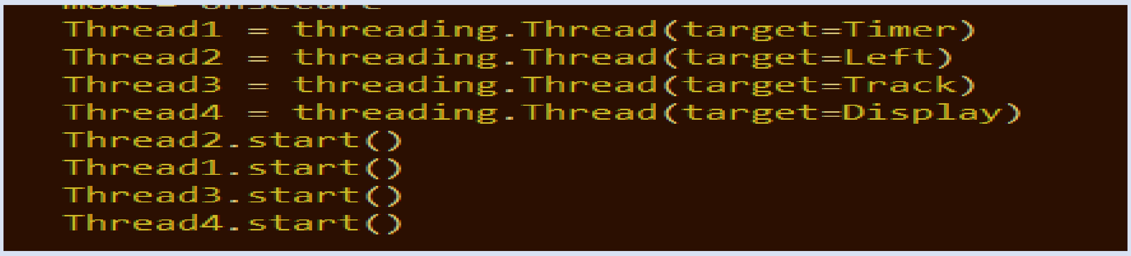
Validation and Performance

To make sure that the program was very fast and was operating on real time I had to make sure that I use the built it python time libraries



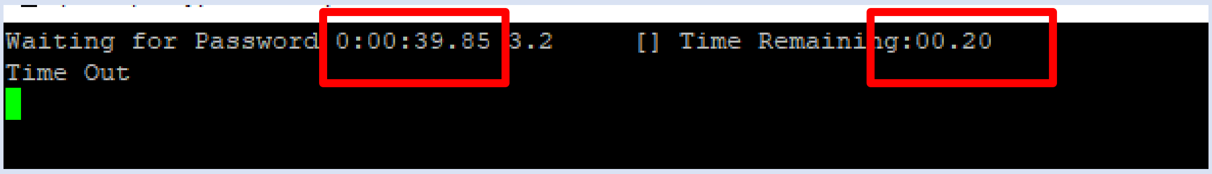
```
from datetime import datetime, timedelta
```

This made that I use the CPU clock to count the time passed with very high accuracy. And to ensure that the program runs smoothly without hang or lag I used a ported concurrency (heaving multiple threads running at once to accomplish a task



```
    Thread1  = threading.Thread(target=Timer)
    Thread2  = threading.Thread(target=Left)
    Thread3  = threading.Thread(target=Track)
    Thread4  = threading.Thread(target=Display)
    Thread2.start()
    Thread1.start()
    Thread3.start()
    Thread4.start()
```

And to ensure that program works correctly I display the time passed and the time remaining , the sum total must be 40 seconds highlighted in red



```
Waiting for Password 0:00:39.85 3.2      [] Time Remaining:00.20
Time Out
```

Conclusion

When I started the coding for the system I wrote a serial code without any parallelism ,I later found the serial code to be very useless because it was failing to meet the time deadlines , Improved the system by using Parnellism and concurrency , this made it possible for the system to run with real time requirements and very "smooth" and responsive , the system could be with all things regarded I don't think the system can be a useful product because of the price of the components involved in making the system, the raspberry pi 3+ is expensive in south Africa , the system would be successful if we use a cheaper micro controller as the controller

References

- https://softwareengineering.stackexchange.com/questions/120859/uml-diagrams-of-multi-threaded-applications
- https://creately.com/diagram/example/hfflld5a1/Parallelism%20Thread%20Diagram
- https://docs.python.org/3/library/datetime.html