CENTRAL CONTROL CENTER

MILESTONE 01 | GROUP 20C

Security System for the Pentagon

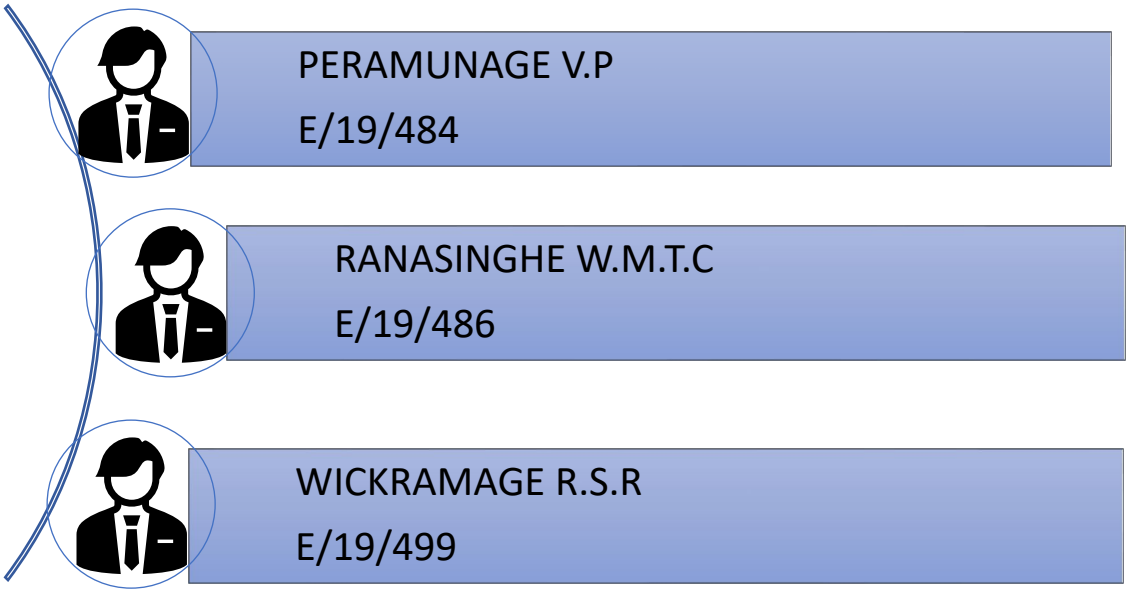
Overview

The Pentagon is the headquarters building of the United States Department of Defense. It was constructed on an accelerated schedule during World War II. As a symbol of the U.S. military, the phrase The Pentagon is often used as a metonym for the Department of Defense and its leadership.The Pentagon is the world's largest office building, with about 6.5 million square feet (150 acres; 60 ha) of floor space, of which 3.7 million sq ft (85 acres; 34 ha) are used as offices.[6][7] Some 23,000 military and civilian employees,[7] and another 3,000 non-defense support personnel, work in the Pentagon. It has five sides, five floors above ground, two basement levels, and five ring corridors per floor with a total of 17.5 miles (28.2 km)[7] of corridors.

Our Goals

1. To measure the temperature of the control center is monitored for fire detection
2. To build a MorseCode to enter secret passphrase to the server room using LDR and Light (PhoneTorch)

Our Team


PERAMUNAGE V.P
E/19/484
RANASINGHE W.M.T.C
E/19/486
WICKRAMAGE R.S.R
E/19/499

CONTENT

1. Thermiser Circuit Design
   - FLLOW CHART
   - CIRCUIT DIAGRAM
   - CODE SNIPPETS
   - EXPLANATION
   - FIRMWEAR

2. LDR Circuit Design
   - FLOW CHART
   - CIRCUIT DIAGRAM
   - CODE SNIPPET
   - EXPLANATION
   - FIRMWEAR

1.Flow Charts

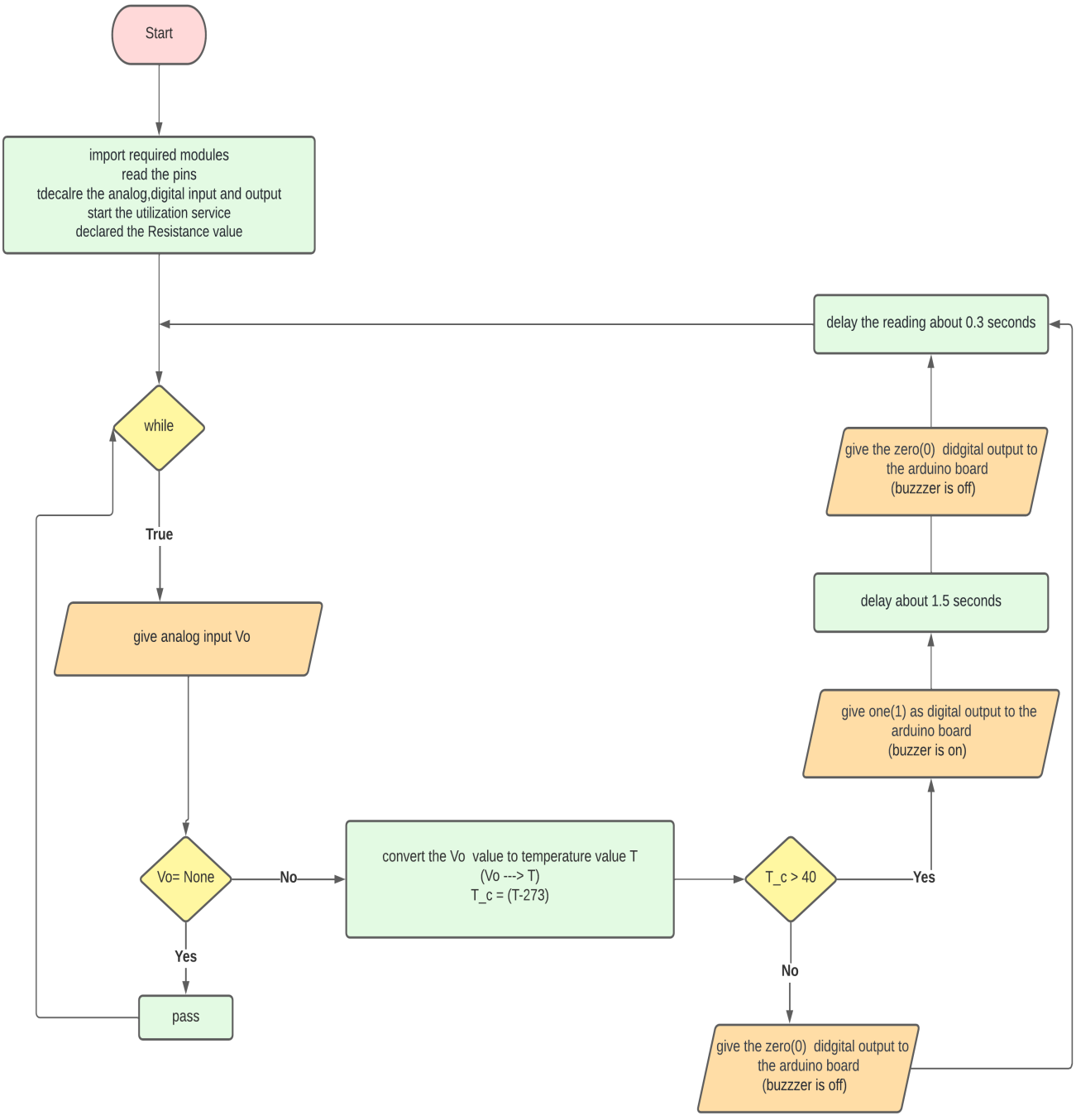


FIGURE 1.1

2.Circuit Diagrams

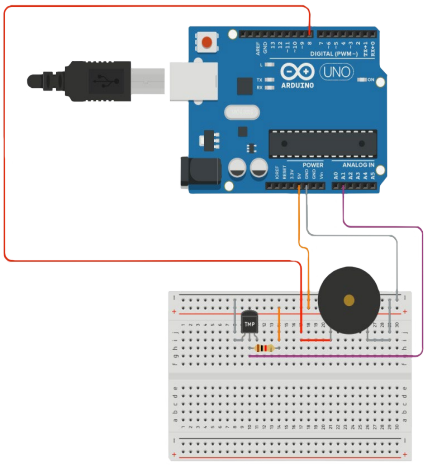


FIGURE 1.2

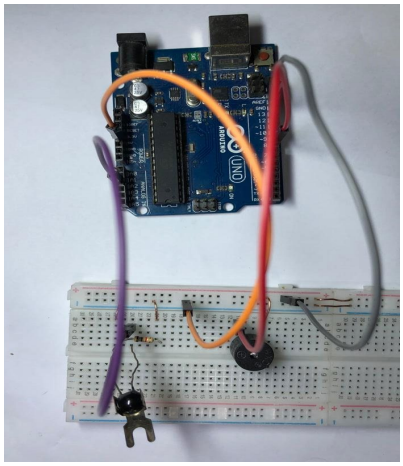


FIGURE 1.3

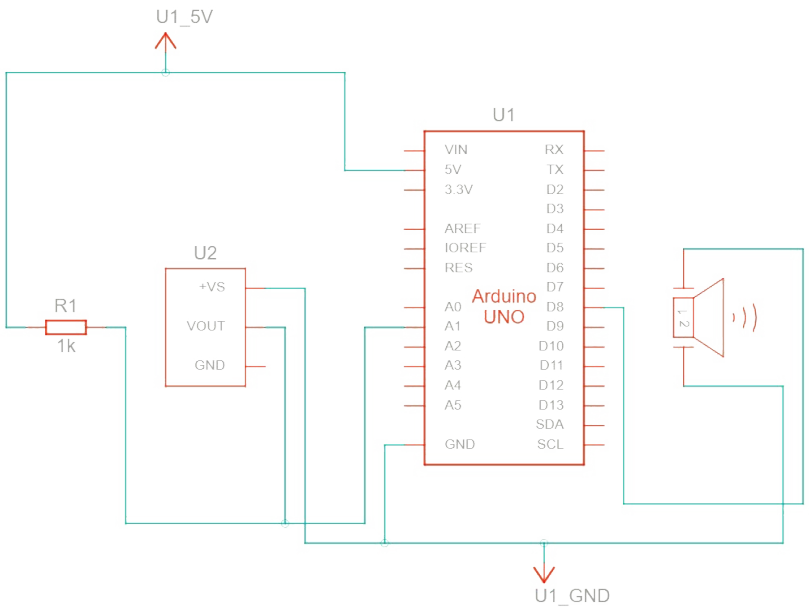


FIGURE 1.4

3.Code Snippet

thermister.py

```python
from pyfirmata import Arduino, util, INPUT,OUTPUT #import pyfirmata module
import math #import math module
import time #import time module

board = Arduino("COM3") #Arduino board port

thermo_pin = 1 #thermister pin
buzzer_pin = 8 #buzzer pin
switch_pin = 4  #switch pin

# analog,digital output & input
board.analog[thermo_pin].mode = INPUT
board.digital[buzzer_pin].mode = OUTPUT
board.digital[switch_pin].mode = INPUT

it  = util.Iterator(board)
it.start()

board.analog[thermo_pin].enable_reporting()

# thermister resistance
R = 1000


while True:
    # analog voltage value
    Vo = board.analog[thermo_pin].read()

    # pass the None value
    if Vo == None:
        pass

    else:
        # convert voltage into temperature value equations
        Vt = Vo*5
        Rt = (R * Vt)/(5-Vt)
        T = 1/((math.log(Rt/1000))/35441+(1/298.15))

        # temperature in celsius
        T_c = int(T-273)


        # print the temperature value
        print(T_c)

        # dager level temperature detected process
        if T_c >40:
            board.digital[buzzer_pin].write(1) #buzzer is on
            time.sleep(1.5) #delay the time upto 100 millisecond
            board.digital[buzzer_pin].write(0) # buzzer is off

        else:
            board.digital[buzzer_pin].write(0) # buzzer is off

        # delay time
        time.sleep(0.3)
```

FIGURE 1.5

4.Explanations

Hardware Components:

- Thermistor, 1 kilo-ohm resistor, jumper wires, Buzzer

Functionality of sensors:

- Thermistor:
  - Thermistor are used as temperature sensors. For example Fire Alarms have been included thermistor sensor. In thermistor, resistance decrease as the temperature increases. At low temperature, It has very high resistance and only little current can flow through it. At high temperature, It has low resistance and more current can flow through it. In this Practical, 1 kilo-ohm resistor were used. So At low temperature thermistor resistance is near to the 1 kilo-ohm value and At high temperature It near to the 0 kilo-ohm.

Hardware Explanation:

- In this circuit temperature is measured from the thermistor sensor. At the room temperature it has resistance value. Thermistor gives the voltage drop to the Arduino board as analog input. That analog output is number that is between 0 and 1 corresponding to the voltage drop. Then that number calculated through equation and convert into real world kelvin temperature value. When temperature is increased and it beyond the dangerous value Arduino board gives digital output to buzzer on. After delaying 1.5 seconds buzzer is off this happen infinitely the temperature value below the dangerous level.

Code Explanation:

Program workflow:

- first we give analog input and get the that analog value through the Arduino board.
- if the analog value is None then that value is passed. Else analog value converts into the temperature value by using standard equation.
- After, that temperature value converts into Celsius because that temperature value is kelvin.
- Then check whether the temperature value is above the dangerous temperature.
- If the temperature value is above the dangerous temperature we give digital output to the Arduino board and turn on the fire alarm buzzer for 1.5 second and turn off the buzzer. That is happening infinitely until temperature value goes down below the dangerous temperature level .if temperature value goes down below the dangerous temperature buzzer is off.

Usage of data structures:

- There are no data structures in the code.

5.Firmware

```
from pyfirmata import Arduino, util, INPUT,OUTPUT #import pyfirmata module
import math #import math module
import time #import time module

board = Arduino("COM3") #Arduino board port

thermo_pin = 1 #thermister pin
buzzer_pin = 8 #buzzer pin
switch_pin = 4  #switch pin

# analog,digital output & input
board.analog[thermo_pin].mode = INPUT
board.digital[buzzer_pin].mode = OUTPUT
board.digital[switch_pin].mode = INPUT

it  = util.Iterator(board)
it.start()

board.analog[thermo_pin].enable_reporting()

# thermister resistance
R = 1000


while True:
   # analog voltage value
   Vo = board.analog[thermo_pin].read()

   # pass the None value
   if Vo == None:
      pass

   else:
      # convert voltage into temperature value equations
      Vt = Vo*5
      Rt = (R * Vt)/(5-Vt)
      T = 1/((math.log(Rt/1000))/35441+(1/298.15))

      # temperature in celsius
      T_c = int(T-273)
```

```
# print the temperature value
print(T_c)

# dager level temperature detected process
if T_c >40:
   board.digital[buzzer_pin].write(1) #buzzer is on
   time.sleep(1.5) #delay the time upto 100 millisecond
   board.digital[buzzer_pin].write(0) # buzzer is off

else:
   board.digital[buzzer_pin].write(0) # buzzer is off

# delay time
time.sleep(0.3)
```

1.Flow charts

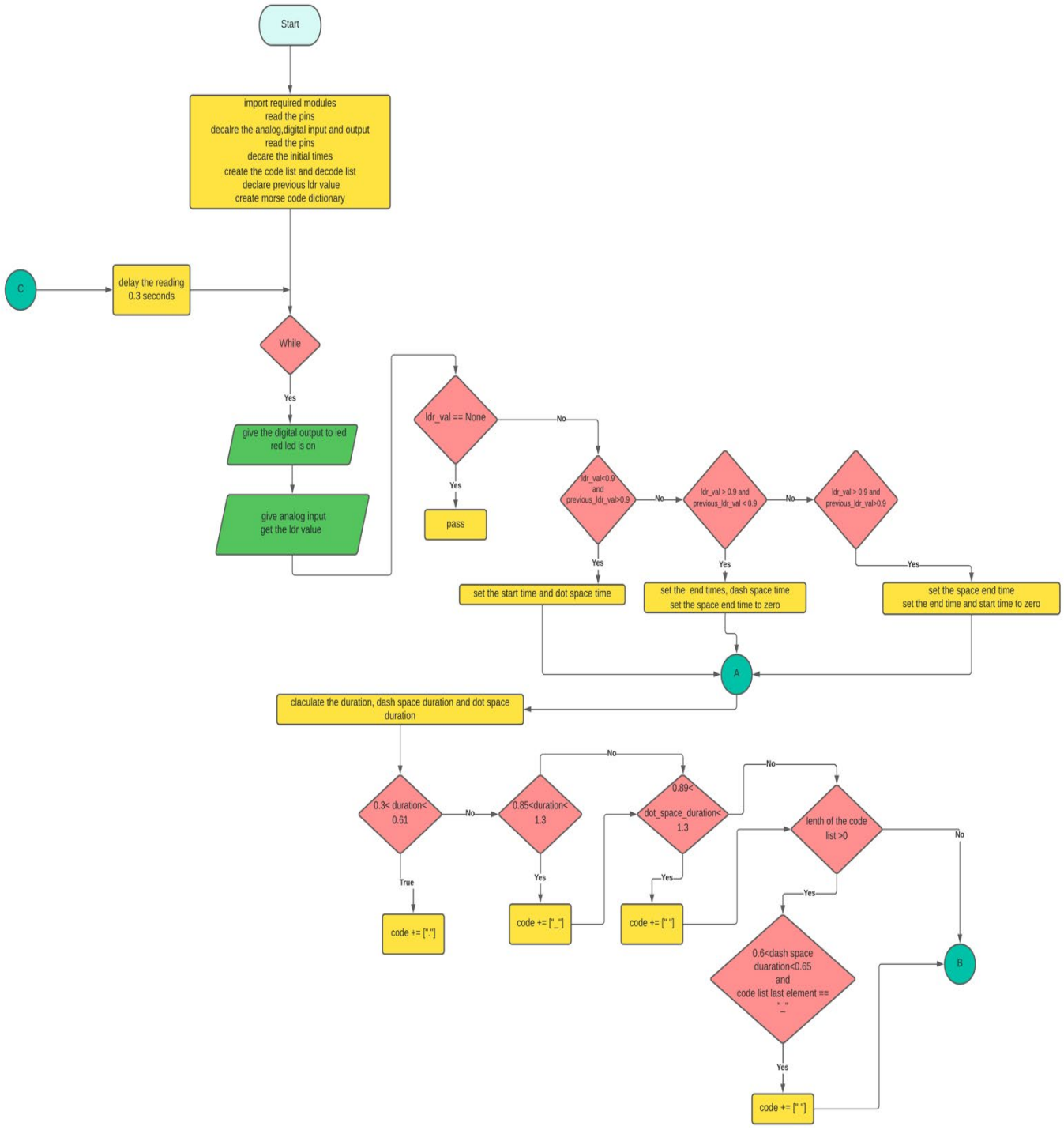


FIGURE 2.1

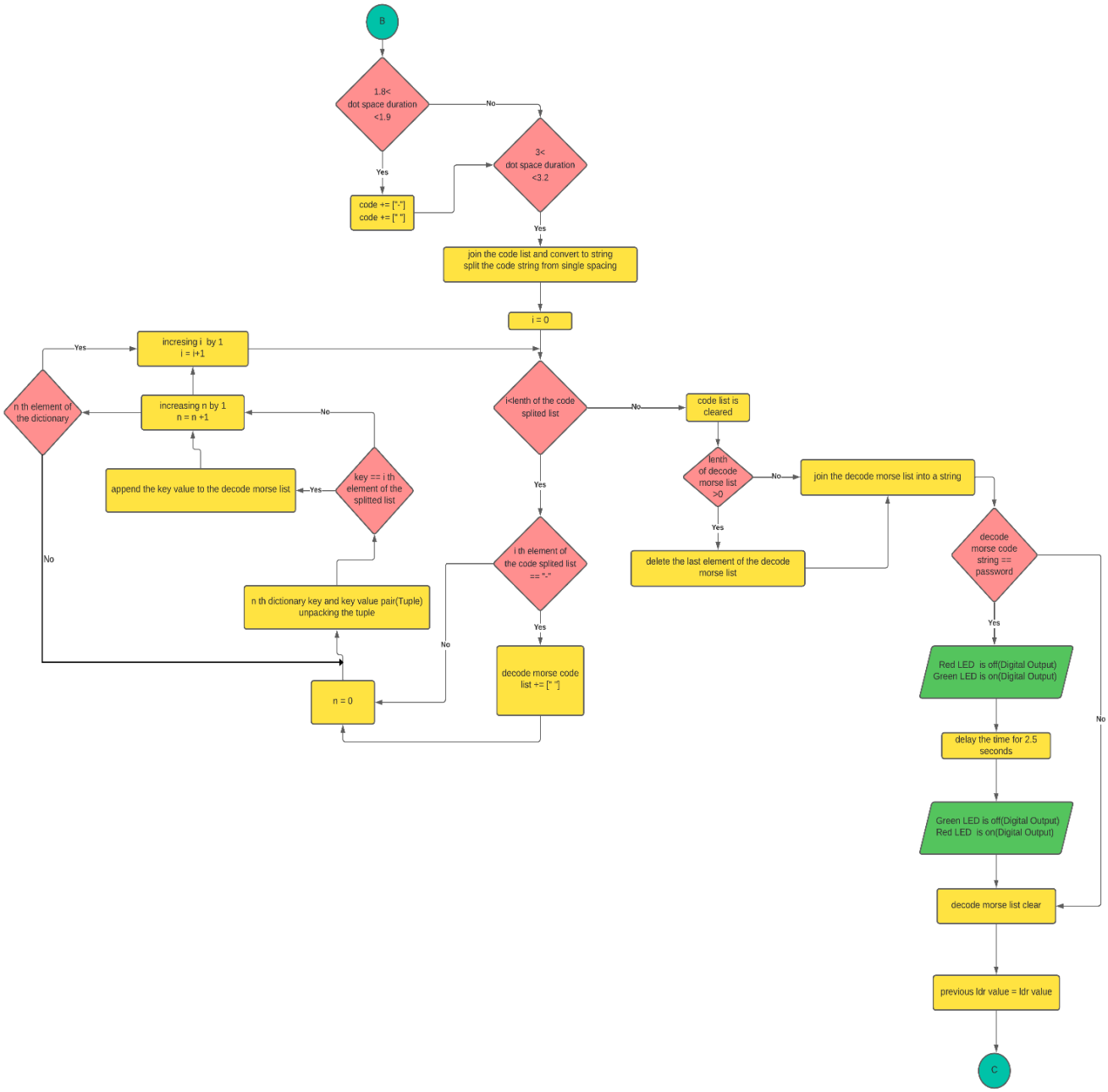

B
1.8< dot space duration <1.9
No
Yes
3< dot space duration <3.2
code += ["-"]
code += [" "]
Yes
join the code list and convert to string
split the code string from single spacing
i = 0
incresing i by 1
i = i+1
Yes
n th element of the dictionary
increasing n by 1
n = n +1
No
append the key value to the decode morse list
Yes
key == i th element of the splitted list
No
n th dictionary key and key value pair(Tuple) unpacking the tuple
n = 0
No
i<lenth of the code splited list
No
Yes
i th element of the code splited list == "-"
Yes
No
decode morse code list += [" "]
code list is cleared
lenth of decode morse list >0
No
Yes
delete the last element of the decode morse list
join the decode morse list into a string
decode morse code string == password
Yes
No
Red LED is off(Digital Output)
Green LED is on(Digital Output)
delay the time for 2.5 seconds
Green LED is off(Digital Output)
Red LED is on(Digital Output)
decode morse list clear
previous ldr value = ldr value
C

FIGURE 2.2

2.Circuit Diagrams

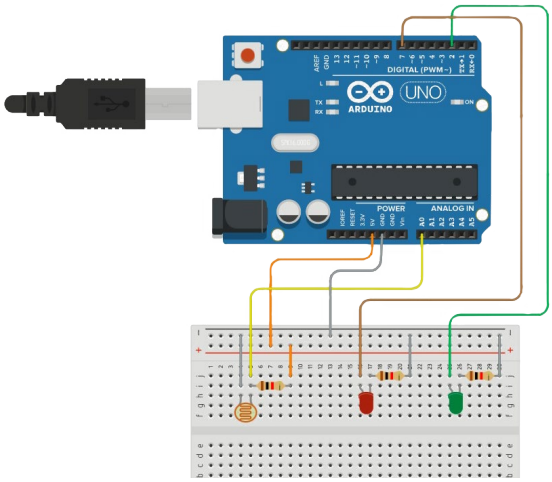


FIGURE 2.3

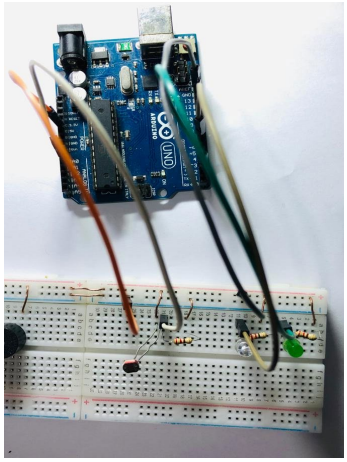


FIGURE 2.4

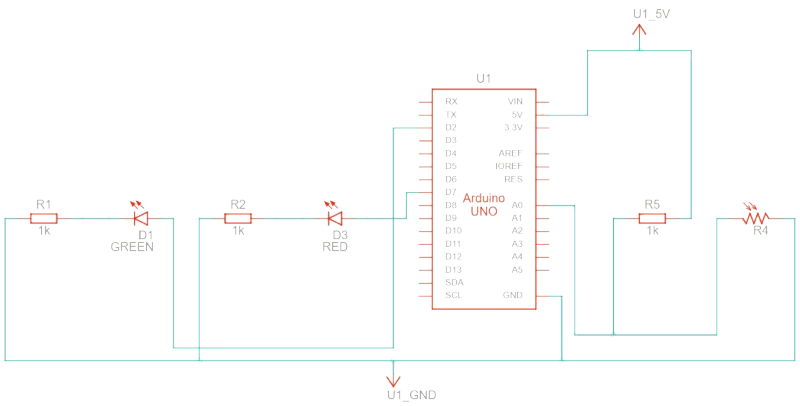
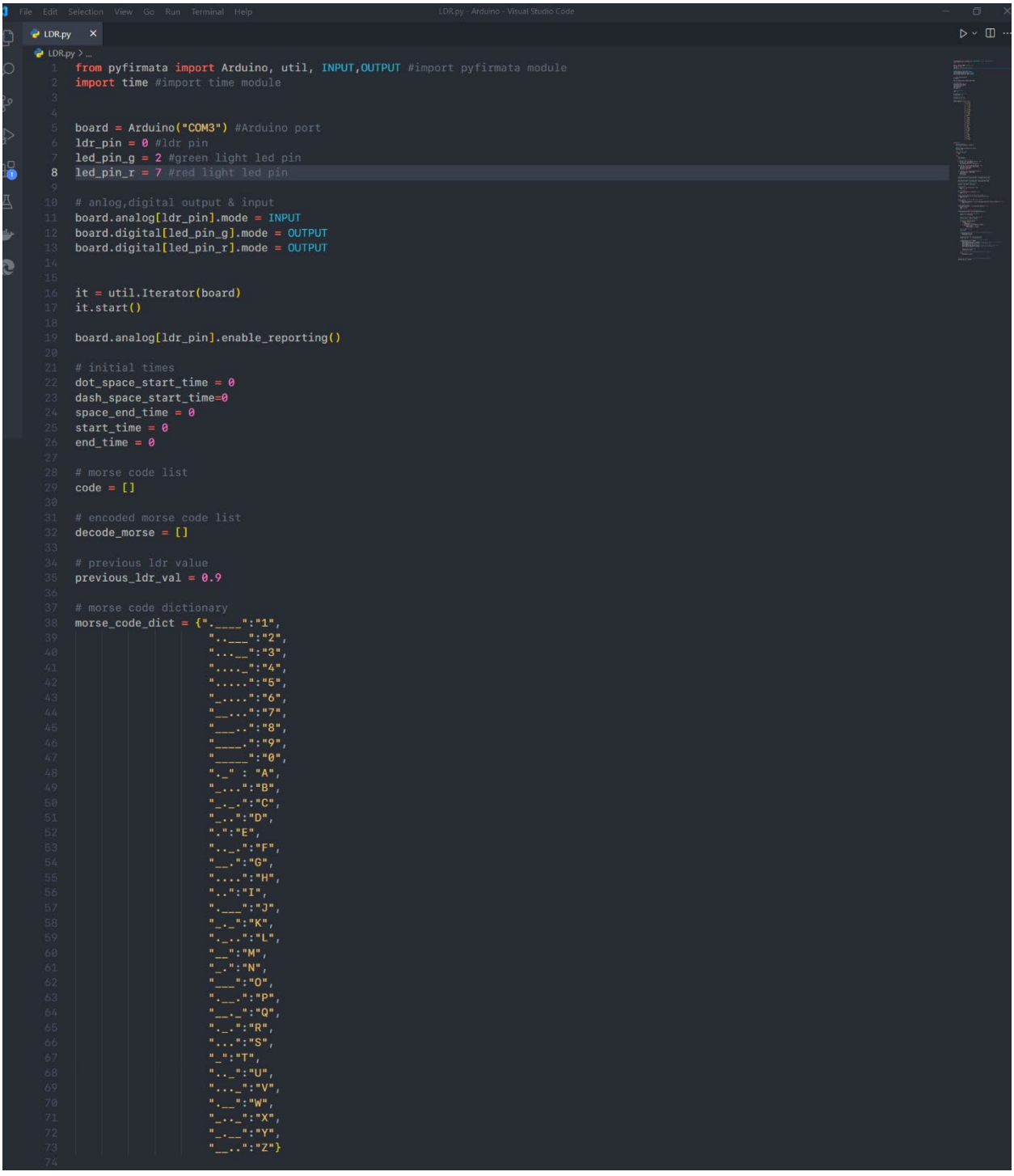


FIGURE 2.5

3.Code Snippet



```python
from pyfirmata import Arduino, util, INPUT,OUTPUT #import pyfirmata module
import time #import time module


board = Arduino("COM3") #Arduino port
ldr_pin = 0 #ldr pin
led_pin_g = 2 #green light led pin
led_pin_r = 7 #red light led pin

# anlog,digital output & input
board.analog[ldr_pin].mode = INPUT
board.digital[led_pin_g].mode = OUTPUT
board.digital[led_pin_r].mode = OUTPUT


it = util.Iterator(board)
it.start()

board.analog[ldr_pin].enable_reporting()

# initial times
dot_space_start_time = 0
dash_space_start_time=0
space_end_time = 0
start_time = 0
end_time = 0

# morse code list
code = []

# encoded morse code list
decode_morse = []

# previous ldr value
previous_ldr_val = 0.9

# morse code dictionary
morse_code_dict = {".____":"1",
                   "..___":"2",
                   "...__":"3",
                   "...._":"4",
                   ".....":"5",
                   "_....":"6",
                   "__...":"7",
                   "___..":"8",
                   "____.":"9",
                   "_____":"0",
                   "._" : "A",
                   "_...":"B",
                   "_._.":"C",
                   "_..":"D",
                   ".":"E",
                   ".._.":"F",
                   "__.":"G",
                   "....":"H",
                   "..":"I",
                   ".___":"J",
                   "_._":"K",
                   "._..":"L",
                   "__":"M",
                   "_.":"N",
                   "___":"O",
                   ".__.":"P",
                   "__._":"Q",
                   "._.":"R",
                   "...":"S",
                   "_":"T",
                   ".._":"U",
                   "..._":"V",
                   ".__":"W",
                   "_.._":"X",
                   "_.__":"Y",
                   "__..":"Z"}
```

FIGURE 2.6

```
while True:
    # red led light on
    board.digital[led_pin_r].write(1)

    # ldr value
    ldr_val = board.analog[ldr_pin].read()
    print(ldr_val)

    # pass the None value
    if ldr_val == None:
        pass

    else:
        # time claculation process
        if ldr_val < 0.9 and previous_ldr_val > 0.9:
            start_time = time.time()
            dot_space_start_time = time.time()

        elif ldr_val > 0.9 and previous_ldr_val < 0.9:
            dash_space_start_time = time.time()
            end_time = time.time()
            space_end_time = 0

        elif ldr_val > 0.9 and previous_ldr_val>0.9:
            space_end_time = time.time()
            end_time=0
            start_time=0

        # time duration between letter space or word space after dash
        dash_space_duration =space_end_time - dash_space_start_time

        # time duration between letter space or word space after dot
        dot_space_duration = space_end_time - dot_space_start_time

        # time duration between dot or dash
        duration = end_time - start_time

        # find the dot
        if duration > 0.3 and duration < 0.61:
            code += ["."]

        # find the dash
        elif (duration >0.85 and duration < 1.3):
            code += ["_"]

        # space between letters or numbers after the dot
        if dot_space_duration > 0.89 and dot_space_duration< 0.95:
            code += [" "]

        # space between letter or number after the dash
        if len(code)>0:
            if (dash_space_duration > 0.6 and dash_space_duration< 0.65) and code[-1] == "_":
                code += [" "]

        # space between words
        if dot_space_duration> 1.8 and dot_space_duration < 1.9:
            code += ["-"]
            code += [" "]
```

FIGURE 2.7

```
        # letter, word or sentence encoding process
        if dot_space_duration >3 and dot_space_duration <3.2:

            # mosh code string with dot and dash
            code_str = "".join(code)

            # split list that split from single spacing
            code_str_split = code_str.split(" ")

            # morse code encoding processing
            for morse in code_str_split:
                if morse == "-":
                    decode_morse += [" "]
                for key,value in morse_code_dict.items():
                    if morse == key:
                        decode_morse += [value]

            # morse code list clear
            code.clear()

            # remove the unnesasery last element of the decode_morse list
            if len(decode_morse)>0:
                decode_morse.pop(-1)

            # convert decode_morse list to the string
            decode_morse_str = "".join(decode_morse)

            # check whether the password is correct or incorrect
            if decode_morse_str == "67AB":
                board.digital[led_pin_r].write(0) #if password is correct red led is off
                board.digital[led_pin_g].write(1) # green light is on
                time.sleep(2.5) # delay time
                board.digital[led_pin_g].write(0) # after 2.5 sec green light is off
                board.digital[led_pin_r].write(1) # red light is on again

                # decode_morse list clear
                decode_morse.clear()

            # if password is incorrect decode_morse list is cleared
            else:
                decode_morse.clear()



        # assign the previous ldr value to previous_ldr_val variable
        previous_ldr_val = ldr_val

        # delay time
        time.sleep(0.3)
```


FIGURE 2.8

4.Explanations

Hardware Components:

- LDR Sensor, Three 220-ohm resistor, jumper wires, Green LED and Red LED

Functionality of the sensors:

- LDR Sensor:
  - LDR Sensor is used to detect light levels. For example, Security light are manufactured by using LDR. In the dark or at the low light levels, the resistance of the LDR is very high and little current can through of it. At the high light levels or in the bright light, In the resistance of the LDR is low and current flow can flow through it.

Hardware Explanation:

- In this Circuit password is given as morse code by using light. So, LDR was used for detecting the morse code. When program is running, LDR resistance is very high so current not flowing through the resistance. Therefore, whole current is flowing through the 220-ohm resistor. So, ldr value is 1. When we give the light resistance of the LDR is low and current flow through it. So ldr value gives analog value between 0 and 1. Then it detect the that pattern and give the morse code password.
- In the circuit, red and green LED were used. Initially red light is on. If password is correct the red light is off and then, green light were on. After some 1.5 seconds green light is off and red light is on again.

Code Explanation:

Program Workflow:

- **Important words:**

  Code list : code is the list that has dot and dashes.

  Decode morse list: list that has human language letters

- First we declare the initial times then declare the **code list** and declare the **decode morse** list and also previous ldr value set to 0.9.
- When we enter the dot or dash previous ldr value is > 0.9 and ldr value<0.9 in that set the start time.
- Ldr value >0.9 and previous ldr value<0.9 set the end time.
- After calculate the duration about dot and dash.
- Commonly dot : dash has time ratio 1:3 so duration so duration in that range dot or dash append to **the code list**.
- Completing the first morse code corresponding to letter there is space time gap between two morse codes if that space duration is in the relevant range space is append to **the code list**.

- If morse code has words they have also the relevant space time if that space time in the relevant duration this symbol (“-”) append to **the code list**.
- After completing the whole morse code there is time duration to detect whether you enter the whole code.
- After that detection **code list** convert to string and then that string split again to another list. That is **splitter code list**.
- Nested “for loop” process
    - Then that **splitter code list** iterates through the for “for loop”. (In this process gives **splitter code list element** one by one.)
    - If splitter code list has this (“-”) symbol , append the single space(“ ”) to the decode morse list. From this process separated the dot-dash morse code words**(this means list has meaning full words in dot(“.”) and dashes(”_”))** by single space.
    - After **morse code dictionary** iterate through the for loop and if **dictionary key** equal to the **splitter code list element** , append the **key value**(**human language letters**) to the decode morse list.
- After completing that process, **code list clear**.
- Then again decode morse list element join together and create a message(**human language**) as string.
- That is the process how morse code message into human language.
- After that message equal to our password, access is granted to the CCC room.

Usage of Data Structures:

Lists:

code : code is the list that has dot and dashes.

decode_morse : list that has human language letters

Dictionary:

morse_code_dict : dictionary that has morse code and human language letters and numbers corresponding to morse code**(e.g. :- {“._”: “A”})**

5.Firmware

```
from pyfirmata import Arduino, util, INPUT,OUTPUT #import pyfirmata module
import time #import time module


board = Arduino("COM3") #Arduino port
ldr_pin = 0 #ldr pin
led_pin_g = 2 #green light led pin
led_pin_r = 7 #red light led pin

# anlog,digital output & input
board.analog[ldr_pin].mode = INPUT
board.digital[led_pin_g].mode = OUTPUT
board.digital[led_pin_r].mode = OUTPUT


it = util.Iterator(board)
it.start()

board.analog[ldr_pin].enable_reporting()

# initial times
dot_space_start_time = 0
dash_space_start_time=0
space_end_time = 0
start_time = 0
end_time = 0

# morse code list
code = []


# encoded morse code list
decode_morse = []
```

```
# previous ldr value
previous_ldr_val = 0.9

# morse code dictionary
morse_code_dict = {".__":"1",
                   "..__":"2",
                   "...__":"3",
                   "...._":"4",
                   ".....":"5",
                   "_....":"6",
                   "__...":"7",
                   "_..":"8",
                   "___.":"9",
                   "____":"0",
                   "._" : "A",
                   "_...":"B",
                   "..":"C",
                   "_..":"D",
                   ".":"E",
                   "..._":"F",
                   "__.":"G",
                   "....":"H",
                   "..":"I",
                   "._":"J",
                   ".":"K",
                   "._..":"L",
                   "__":"M",
                   "_.":"N",
                   "_":"O",
                   ".__.":"P",
                   "_.":"Q",
                   "._.":"R",
                   "...":"S",
                   "_":"T",
                   "..._":"U",
                   "...._":"V",
                   ".__":"W",
                   "..":"X",
                   "._":"Y",
                   "__..":"Z"}
```

```
while True:
    # red led light on
    board.digital[led_pin_r].write(1)

    # ldr value
    ldr_val = board.analog[ldr_pin].read()
    print(ldr_val)

    # pass the None value
    if ldr_val == None:
        pass

    else:
        # time claculation process
        if ldr_val < 0.9 and previous_ldr_val > 0.9:
            start_time = time.time()
            dot_space_start_time = time.time()

        elif ldr_val > 0.9 and previous_ldr_val < 0.9:
            dash_space_start_time = time.time()
            end_time = time.time()
            space_end_time = 0

        elif ldr_val > 0.9 and previous_ldr_val>0.9:
            space_end_time = time.time()
             end_time=0
            start_time=0

       # time duration between letter space or word space after dash
        dash_space_duration =space_end_time - dash_space_start_time

        # time duration between letter space or word space after dot
        dot_space_duration = space_end_time - dot_space_start_time

        # time duration between dot or dash
        duration = end_time - start_time
```

```
# find the dot
if duration > 0.3 and duration < 0.61:
    code += ["."]

# find the dash
elif (duration >0.85 and duration < 1.3):
    code += ["_"]

# space between letters or numbers after the dot
if dot_space_duration > 0.89 and dot_space_duration< 0.95:
    code += [" "]

# space between letter or number after the dash
if len(code)>0:
    if (dash_space_duration > 0.6 and dash_space_duration< 0.65) and code[-1] == "_":
        code += [" "]

# space between words
if dot_space_duration> 1.8 and dot_space_duration < 1.9:
    code += ["-"]
    code += [" "]

# letter, word or sentence encoding process
if dot_space_duration >3 and dot_space_duration <3.2:

    # mosh code string with dot and dash
    code_str = "".join(code)

    # split list that split from single spacing
    code_str_split = code_str.split(" ")

    # morse code encoding processing
    for morse in code_str_split:
        if morse == "-":
            decode_morse += [" "]
        for key,value in morse_code_dict.items():
            if morse == key:
                decode_morse += [value]

    # morse code list clear
    code.clear()
```

```
    # remove the unnesasery last element of the decode_morse list
    if len(decode_morse)>0:
        decode_morse.pop(-1)

    # convert decode_morse list to the string
    decode_morse_str = "".join(decode_morse)

    # check whether the password is correct or incorrect
    if decode_morse_str == "67AB":
        board.digital[led_pin_r].write(0) #if password is correct red led is off
        board.digital[led_pin_g].write(1) # green light is on
        time.sleep(2.5) # delay time
        board.digital[led_pin_g].write(0) # after 2.5 sec green light is off
        board.digital[led_pin_r].write(1) # red light is on again

        # decode_morse list clear
        decode_morse.clear()

    # if password is incorrect decode_morse list is cleared
    else:
        decode_morse.clear()



# assign the previous ldr value to previous_ldr_val variable
previous_ldr_val = ldr_val

# delay time
time.sleep(0.3)
```

END..