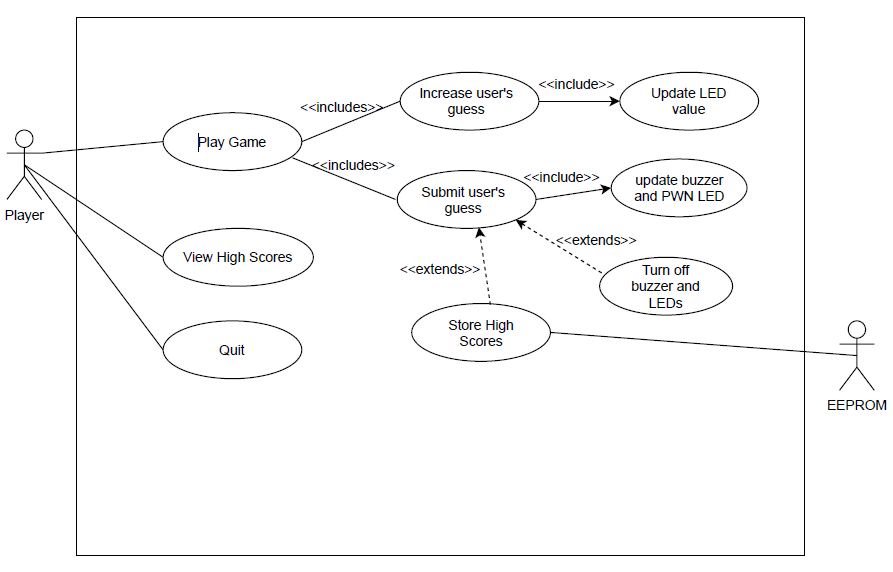
Work Package 4 - Practical

CLXBHE005, KMPKWE002

EEE3096S 2021

[Demonstration Video](https://www.youtube.com/watch?v=SH_DTlAQyDY)

1. **A UML use-case diagram of the system:**



1. **The Prac3.py code contents**

The initialisations and imports are as follows:

# Import libraries

import traceback

import RPi.GPIO as GPIO

import random

import ES2EEPROMUtils

import os

import time

import time

# some global variables that need to change as we run the program

end\_of\_game = None  # set if the user wins or ends the game

pwm\_led = None

pwm\_trans = None

guessed\_number = 0

number\_of\_guesses = 0

player\_name = None

value = None

# DEFINE THE PINS USED HERE

LED\_value = [11, 13, 15]

LED\_accuracy = 32

btn\_submit = 16

btn\_increase = 18

buzzer = None

trans\_pin = 33

eeprom = ES2EEPROMUtils.ES2EEPROM()

The welcome method is as follows:

# Print the game banner

def welcome():

    os.system('clear')

    print("  \_   \_                 \_                  \_\_\_\_\_ \_            \_\_  \_\_ \_")

    print("| \ | |               | |                / \_\_\_\_| |          / \_|/ \_| |")

    print("|  \| |\_   \_ \_ \_\_ \_\_\_ | |\_\_   \_\_\_ \_ \_\_  | (\_\_\_ | |\_\_  \_   \_| |\_| |\_| | \_\_\_ ")

    print("| . ` | | | | '\_ ` \_ \| '\_ \ / \_ \ '\_\_|  \\_\_\_ \| '\_ \| | | |  \_|  \_| |/ \_ \\")

    print("| |\  | |\_| | | | | | | |\_) |  \_\_/ |     \_\_\_\_) | | | | |\_| | | | | | |  \_\_/")

    print("|\_| \\_|\\_\_,\_|\_| |\_| |\_|\_.\_\_/ \\_\_\_|\_|    |\_\_\_\_\_/|\_| |\_|\\_\_,\_|\_| |\_| |\_|\\_\_\_|")

    print("")

    print("Guess the number and immortalise your name in the High Score Hall of Fame!")

The menu method is as follows:

def menu():

    global end\_of\_game

    global value

    option = input("Select an option:  H - View High Scores  P - Play Game  Q - Quit\n>>>")

    option = option.upper()

    if option == "H":

        os.system('clear')

        print("HIGH SCORES!!")

        s\_count, ss = fetch\_scores()

        display\_scores(s\_count, ss)

    elif option == "P":

        os.system('clear')

        print("Starting a new round!")

        print("Use the buttons on the Pi to make and submit your guess!")

        print("Press and hold the guess button to cancel your game")

        value = generate\_number()

        while not end\_of\_game:

            pass

    elif option == "Q":

        print("Come back soon!")

        exit()

    else:

        print("Invalid option. Please select a valid one!")

The display\_scores method is as follows:

def display\_scores(count, raw\_data):

    # print the scores to the screen in the expected format

    number\_of\_scores\_to\_print = 3

    print("There are {} scores. Here are the top 3!".format(count))

    # print out the scores in the required format

    score\_count = 1

    name = ""

    for i in range(1,4\*number\_of\_scores\_to\_print+1):

        if(i%4==0):

             print("{} - {} took {} guesses".format(score\_count, name, raw\_data[i-1]))

             name = ""

             score\_count +=1

        else:

            name += str(raw\_data[i-1])

The setup method is as follows:

# Setup Pins

def setup():

    global pwm\_led

    global pwm\_trans

    global trans\_pin

    # Setup board mode

    GPIO.setmode(GPIO.BOARD)

    # Setup regular GPIO

    #LEDS Setup

    for pinNo in LED\_value:

        GPIO.setup(pinNo, GPIO.OUT, initial=GPIO.LOW)

    GPIO.setup(LED\_accuracy, GPIO.OUT, initial=GPIO.LOW)

    #Button Setup

    GPIO.setup(btn\_submit, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

    GPIO.setup(btn\_increase, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

    #trans\_pin Setup

    GPIO.setup(trans\_pin, GPIO.OUT,initial=GPIO.HIGH)

    # Setup PWM channels

    pwm\_led = GPIO.PWM(LED\_accuracy,2000)

    pwm\_trans = GPIO.PWM(trans\_pin, 1)

    # Setup debouncing and callbacks

    GPIO.add\_event\_detect(btn\_increase, GPIO.FALLING, callback=btn\_increase\_pressed, bouncetime=300)

    GPIO.add\_event\_detect(btn\_submit, GPIO.FALLING, callback=btn\_guess\_pressed, bouncetime=1000)

The fetch\_scores method is as follows:

# Load high scores

def fetch\_scores():

    # get however many scores there are

    score\_count = eeprom.read\_block(0,1)[0]

    # Get the scores

    scores\_bin = eeprom.read\_block(1,4\*score\_count)

    scores = []

    # convert the codes back to ascii

    for i in range(1,4\*score\_count+1):

        if(i%4==0):

            scores.append(scores\_bin[i-1])

        else:

            scores.append(chr(scores\_bin[i-1]))

    # return back the results

    return score\_count, scores

The save\_scores method is as follows:

# Save high scores

def save\_scores():

    global player\_name

    global number\_of\_guesses

    # fetch scores

    score\_count, scores = fetch\_scores()

    scores\_2d = []

    name = ""

    for i in range(1,4\*score\_count+1):

        if(i%4==0):

            scores\_2d.append([name,scores[i-1]])

            name = ""

        else:

            name += str(scores[i-1])

    # include new score

    scores\_2d.append([player\_name,number\_of\_guesses])

    # sort

    scores\_2d.sort(key=lambda x: x[1])

    # update total amount of scores

    eeprom.write\_block(0, [len(scores\_2d)])

    # write new scores

    for i, score in enumerate(scores\_2d):

        data\_to\_write = []

        # get the string

        for letter in score[0]:

            data\_to\_write.append(ord(letter))

        data\_to\_write.append(score[1])

        eeprom.write\_block(i+1, data\_to\_write)

The generate\_number method is as follows:

# Generate guess number

def generate\_number():

    return random.randint(1, pow(2, 3)-1)

The toBinary method is as follows:

#Convert Number to binary and return the binary number in an array

def toBinary(decimal):

    binary = []

    for bit in bin(decimal).replace("0b", "").zfill(3):

        binary.append(eval(bit))

    return binary

The btn\_increase\_pressed method is as follows:

# Increase button pressed

def btn\_increase\_pressed(channel):

    global value

    global guessed\_number

    if(value!=None and not end\_of\_game):

        if(guessed\_number >= 7):

            guessed\_number =-1

        # You can choose to have a global variable store the user's current guess

        guessed\_number +=1

        LEDOutPut = toBinary(guessed\_number)

        # Increase the value shown on the LEDs

        GPIO.output(LED\_value[0],LEDOutPut[0])

        GPIO.output(LED\_value[1],LEDOutPut[1])

        GPIO.output(LED\_value[2],LEDOutPut[2])

The accuracy\_leds method is as follows:

# LED Brightness

def accuracy\_leds():

     # Set the brightness of the LED based on how close the guess is to the answer

    # - The % brightness should be directly proportional to the % "closeness"

    # - For example if the answer is 6 and a user guesses 4, the brightness should be at 4/6\*100 = 66%

    # - If they guessed 7, the brightness would be at ((8-7)/(8-6)\*100 = 50%

    global guessed\_number

    global value

    if(guessed\_number<=value):

        pwm\_led.start((guessed\_number/value)\*100)

    elif(guessed\_number>value):

        pwm\_led.start(((8-guessed\_number)/(8-value))\*100)

The btn\_guess\_pressed method is as follows:

# Guess button

def btn\_guess\_pressed(channel):

    global value

    global end\_of\_game

    global guessed\_number

    global LED\_accuracy

    global number\_of\_guesses

    global player\_name

    global pwm\_led

    global pwm\_trans

    global trans\_pin

    global firstRun

    if(value!=None and not end\_of\_game):

        time.sleep(1.8)

        if(GPIO.input(btn\_submit)<0.5):

            GPIO.cleanup([LED\_value[0], LED\_value[1], LED\_value[2], LED\_accuracy, trans\_pin, btn\_submit, btn\_increase])

            end\_of\_game = True

        elif(guessed\_number>0):

                # Compare the actual value with the user value displayed on the LEDs

                number\_of\_guesses +=1

                # Change the PWM LED

                accuracy\_leds()

                # if it's an exact guess:

                if(value == guessed\_number):

                # - Disable LEDs and Buzzer

                    GPIO.output(LED\_value[0], GPIO.LOW)

                    GPIO.output(LED\_value[1], GPIO.LOW)

                    GPIO.output(LED\_value[2], GPIO.LOW)

                    GPIO.setup(trans\_pin, GPIO.OUT,initial=GPIO.HIGH)

                    pwm\_trans.stop(0)

                    GPIO.output(trans\_pin, GPIO.HIGH)

                    pwm\_led.stop(0)

                    GPIO.output(LED\_accuracy, GPIO.LOW)

                    GPIO.cleanup(trans\_pin)

                    # - tell the user and prompt them for a name

                    print(">>>>> Correct Guess <<<<< ")

                    player\_name = input("Enter your three letter name to be diplayed on the score board.\n>>>")

                    if(len(player\_name)>3):

                        player\_name = player\_name[:3]

                    elif(len(player\_name)<3):

                        player\_name = player\_name.ljust(3, 'X')

                    # - fetch all the scores

                    # - add the new score

                    # - sort the scores

                    # - Store the scores back to the EEPROM, being sure to update the score count

                    save\_scores()

                    end\_of\_game = True

                else:

                    # if it's close enough, adjust the buzzer

                    trigger\_buzzer()

The trigger\_buzzer method is as follows:

# Sound Buzzer

def trigger\_buzzer():

    # The buzzer operates differently from the LED

    # While we want the brightness of the LED to change(duty cycle), we want the frequency of the buzzer to change

    # The buzzer duty cycle should be left at 50%

    # If the user is off by an absolute value of 3, the buzzer should sound once every second

    global guessed\_number

    global value

    global pwm\_trans

    global trans\_pin

    GPIO.setup(trans\_pin, GPIO.OUT,initial=GPIO.HIGH)

    pwm\_trans.stop(0)

    GPIO.output(trans\_pin, GPIO.HIGH)

    GPIO.cleanup(trans\_pin)

    if(abs(guessed\_number-value) == 3):

        GPIO.setup(trans\_pin, GPIO.OUT,initial=GPIO.HIGH)

        pwm\_trans.start(50)

        pwm\_trans.ChangeFrequency(1)

    # If the user is off by an absolute value of 2, the buzzer should sound twice every second

    elif(abs(guessed\_number-value) == 2):

        GPIO.setup(trans\_pin, GPIO.OUT,initial=GPIO.HIGH)

        pwm\_trans.start(50)

        pwm\_trans.ChangeFrequency(2)

    # If the user is off by an absolute value of 1, the buzzer should sound 4 times a second

    elif(abs(guessed\_number-value) == 1):

        GPIO.setup(trans\_pin, GPIO.OUT,initial=GPIO.HIGH)

        pwm\_trans.start(50)

        pwm\_trans.ChangeFrequency(4)

Finally the main program when executed does the following:

if \_\_name\_\_ == "\_\_main\_\_":

    try:

        # Call setup function

        welcome()

        setup()

        while True:

            menu()

    except Exception as e:

        print(e)

    finally:

        GPIO.cleanup()