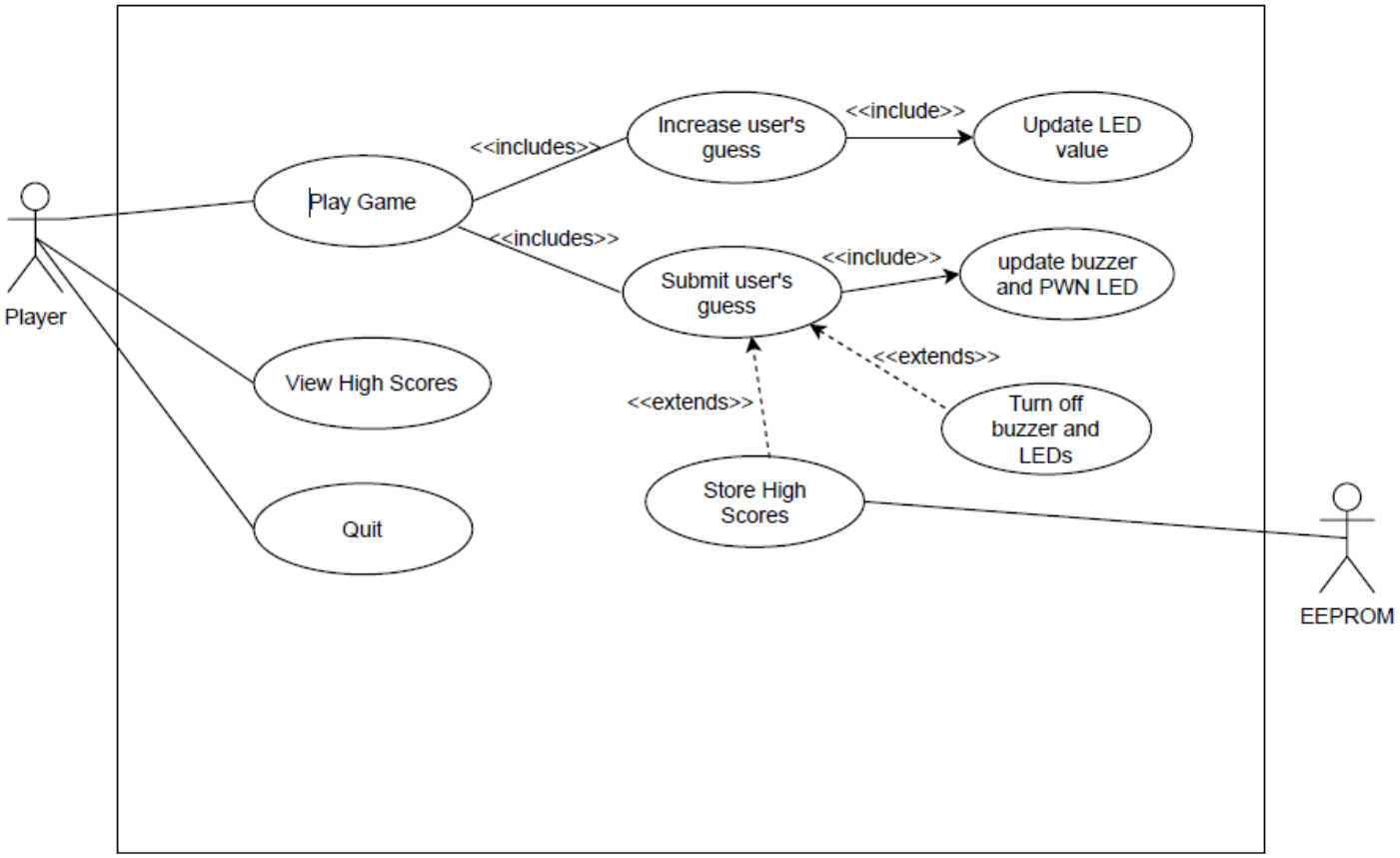
Work Package 4 - Practical

CLXBHE005, KMPKWE002

EEE3096S 2021

[Demonstration Video]

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



2. 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, bounce
time=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 seco
nd
    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 sec
ond
    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 secon
d
    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()
```