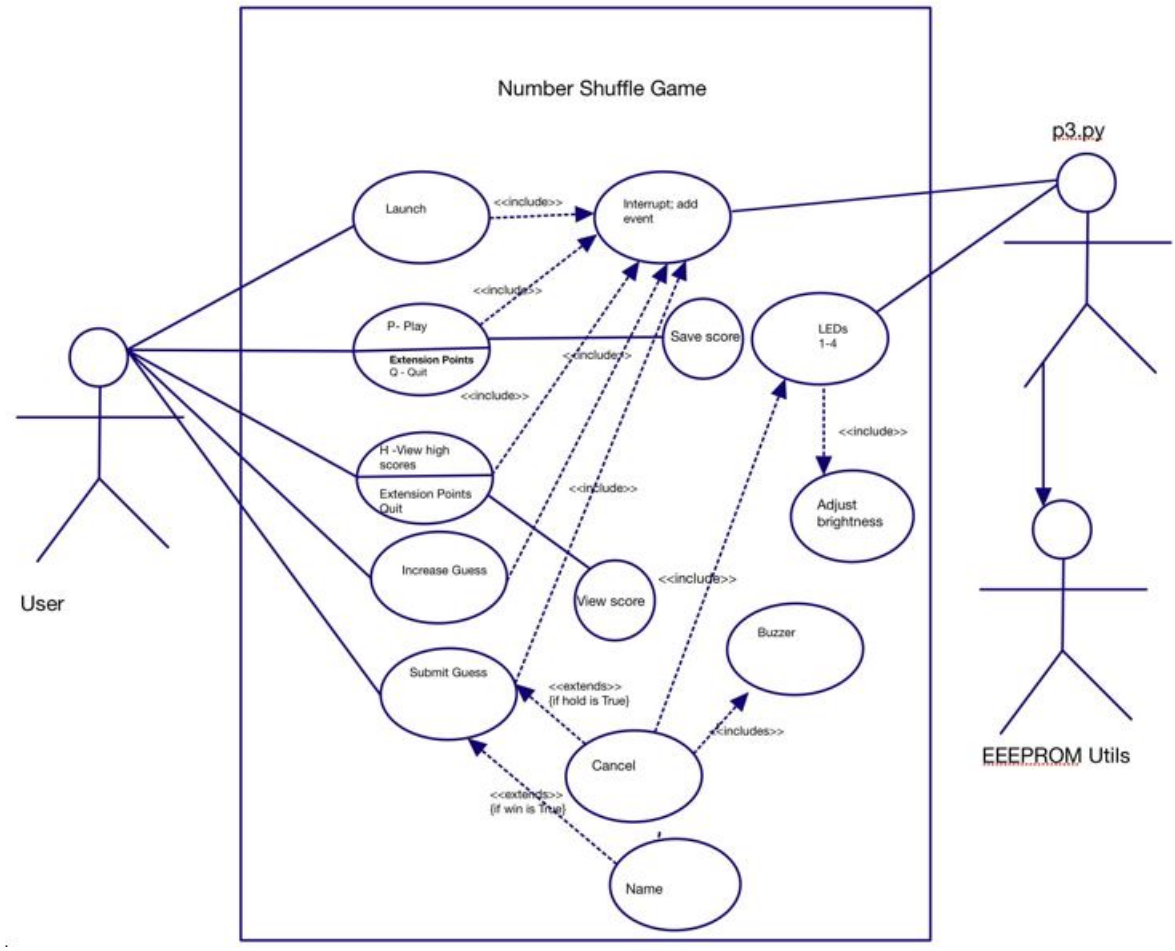
Work Package 3 - Practical

STLTSE004 MVMAMA001
EEE3096S 2021

The UML diagram is as follows:



The initialisations and imports are as follows:

```
# Import libraries
import RPi.GPIO as GPIO
import random
import ES2EEPROMUtils
import os

# 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
guessV = 0
counter = 0
```

```
fixed = 0
# DEFINE THE PINS USED HERE
eeprom = ES2EEPROMUtils.ES2EEPROM()
LED_value = [11, 13, 15]
LED_accuracy = 32
btn_submit = 16
btn_increase = 18
buzz = None
led_pwm = None
eeprom.write_byte(0, counter)
guess_num = 0
name = ""
buzzer = 33
eeprom = ES2EEPROMUtils.ES2EEPROM()
```

The `welcome` method is as follows:

```
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 Fam
```

The `menu` method is as follows:

```
# Print the game menu
def menu():
    global end_of_game
    global fixed
    option = input("Select an option: H — View High Scores P — Play Game Q — Quit\
    option = option.upper()
    if option == "H":
        os.system('clear')
        print("HIGH SCORES!!")
        s_counter, ss = fetch_scores()
        display_scores(s_counter, 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(counter, raw_data):
    # print the scores to the screen in the expected format
    print("There are {} scores. Here are the top 3!".format(counter))
    # print out the scores in the required format
    if(counter == 1):
        print("1  Empty")
        print("2  Empty")
        print("3  Empty")
    if(counter == 1):
        print("1  " +raw_data[0][0]+ " took " + str(raw_data[0][1]) + " guesses")
        print("2  Empty")
        print("3  Empty")
    elif(counter == 2):
        print("1  " +raw_data[0][0]+ " took " + str(raw_data[0][1]) + " guesses")
        print("2  " +raw_data[1][0]+ " took " + str(raw_data[1][1]) + " guesses")
        print("3  Empty")
    elif(counter >= 3):
        print("1  " +raw_data[0][0]+ " took " + str(raw_data[0][1]) + " guesses")
        print("2  " +raw_data[1][0]+ " took " + str(raw_data[1][1]) + " guesses")
        print("3  " +raw_data[2][0]+ " took " + str(raw_data[2][1]) + " guesses")
```

The `setup` method is as follows:

```
# Setup Pins
def setup():
    global led_pwm
    global buzz
    # Setup board mode
    GPIO.setmode(GPIO.BOARD)
    # Setup the regular GPIO
    GPIO.setup(LED_value[0],GPIO.OUT)
    GPIO.setup(LED_value[1],GPIO.OUT)
    GPIO.setup(LED_value[2],GPIO.OUT)
    GPIO.setup(LED_accuracy,GPIO.OUT)
    GPIO.setup(buzzer,GPIO.OUT)
    GPIO.setup(btn_submit,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(btn_increase,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    # Setup the PWM channels
    led_pwm = GPIO.PWM(LED_accuracy,1000)
    led_pwm.start(0)
    buzz=GPIO.PWM(buzzer,1000)
```

```
    # Setup the debouncing and callbacks
    GPIO.add_event_detect(btn_submit,GPIO.FALLING,
    callback = btn_guess_pressed, bouncetime = 500)
    GPIO.add_event_detect(btn_increase,GPIO.FALLING,
    callback = btn_increase_pressed, bouncetime = 500)
    #initial values
    GPIO.output(buzzer,False)
    GPIO.output(LED_value[0],False)
    GPIO.output(LED_value[1],False)
    GPIO.output(LED_value[2],False)
```

The `fetch_scores` method is as follows:

```
# Load the high scores
def fetch_scores():

    score_count = eeprom.read_byte(0)
    results = eeprom.read_block(1,score_count*4)
    arr1 = []
    arr2 = []
    i = 3
    num1 = score_count*4
    while True:
        time.sleep(0.1)
        if i >= num1:
            break
        else:
            str1 = chr(results[i-3])+""+chr(results[i-2])+""+chr(results[i-1])
            guess = results[i]
            i = i + 4
            arr1.append(str1)
            arr1.append(guess)
    for z in range(score_count):
        arr1.append([])
    index = 0
    x = 0
    while True:
        if x >score_count-1:
            break
        else:
            arr2[x].append(arr1[index])
            arr2[x].append(arr1[index+1])
            index = index + 2
            x = x + 1
    arr1.sort(key=lambda x: x[1])
    return score_count, arr2
    # Get the scores
```

The `save_scores` method is as follows:

```
# Save high scores
def save_scores():
    global name
    global guess_num
    global led_pwm
    global buzz
    # fetch scores
    counter, data = fetch_scores()
    counter = counter + 1
    data.append([])
    name = name[0:3]
    data[counter-1].append(name)
    data[counter-1].append(guess_num)
    # include new score
    # sort
    # update total amount of scores
    eeprom.write_byte(0, counter)
    results = data
    results.sort(key=lambda x: x[1])
    data_to_write = []
    for score in results:
    # get the string
        for letter in score[0]:
            data_to_write.append(ord(letter))
        data_to_write.append(score[1])
    eeprom.write_block(1, data_to_write)
    guess_num = 0
```

The `generate_number` method is as follows:

```
# Generate random guess number
def generate_number():
    return random.randint(0, pow(2, 3)-1)
```

The `btn_increase_pressed` method is as follows:

```
# Increase button pressed
def btn_increase_pressed(channel):
    global counter
    global guessV
    global led_pwm
    led_pwm.ChangeDutyCycle(0)
    counter =guessV
    if(counter==0):
        GPIO.output(LED_value[0],False)
        GPIO.output(LED_value[1],False)
        GPIO.output(LED_value[2],False)
```

```
    if(counter==1):
        GPIO.output(LED_value[0],True)
        GPIO.output(LED_value[1],False)
        GPIO.output(LED_value[2],False)
    if(counter==2):
        GPIO.output(LED_value[0],False)
        GPIO.output(LED_value[1],True)
        GPIO.output(LED_value[2],False)
    if(counter==3):
        GPIO.output(LED_value[0],True)
        GPIO.output(LED_value[1],True)
        GPIO.output(LED_value[2],False)
    if(counter==4):
        GPIO.output(LED_value[0],False)
        GPIO.output(LED_value[1],False)
        GPIO.output(LED_value[2],True)
    if(counter==5):
        GPIO.output(LED_value[0],True)
        GPIO.output(LED_value[1],False)
        GPIO.output(LED_value[2],True)
    if(counter==6):
        GPIO.output(LED_value[0],False)
        GPIO.output(LED_value[1],True)
        GPIO.output(LED_value[2],True)
    if(counter==7):
        GPIO.output(LED_value[0],True)
        GPIO.output(LED_value[1],True)
        GPIO.output(LED_value[2],True)
    counter = counter + 1
    if(counter ==8):
        counter = 0
    print("Your guess is:", generate_number(),", the guesses used:",counter)
    counter = counter+1
    guessV+=1
```

The `btn_guess_pressed` method is as follows:

```
# Guess button
def btn_guess_pressed(channel):
    global end_of_game, led_pwm, buzz,guessV
    global counter
    global led_pwm
    global guess_num
    counter = 0
    guess_num = guess_num + 1

    welcome()
    if guessV == guess_num:
```

```
        name = input("\nEnter your name: ")
        while len(name)<3:
            name = input("\nEnter your name: ")
        pwm_led.stop()
        pwm_buzz.stop()
        print(f"{name}, you won!\n")

    else:
        print("\nSo close, yet so far!\n")
        # accuracy_leds()
        trigger_buzzer()

    end_of_game = True
    start_time = time.time()
    while GPIO.input(channel) == 0:
        time.sleep(0.1)
    button_time = time.time() - start_time
    if .1<=button_time <=0.7:
        accuracy_leds()
        trigger_buzzer()
    elif 0.7<button_time:
        led_pwm = None
        #buzz = None
        guess_num = 0
        name = ""
        GPIO.remove_event_detect(btn_submit)
        GPIO.remove_event_detect(btn_increase)
        GPIO.output(LED_value[0],False)
        GPIO.output(LED_value[1],False)
        GPIO.output(LED_value[2],False)
        GPIO.output(buzzer,False)
        GPIO.cleanup()
        setup()
        welcome()
        while True:
            time.sleep(0.1)
            menu()
            pass
    time.sleep(0.1)
```

The `accuracy_leds` method is as follows:

```
# LED Brightness
def accuracy_leds():
    global guessV
    global fixed
    global led_pwm
    # 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 4/6100 = 66%
    # If they guessed 7, the brightness would be at ((87)/(86)100 = 50%
    dc = 0
    if(guessV <= fixed):
        dc = (guessV/fixed)*100
    else:
        dc = ((8-guessV)/(8-fixed))*100
    led_pwm.ChangeDutyCycle(dc)
    time.sleep(0.1)
```

The `trigger_buzzer` method is as follows:

```
# Sound Buzzer

def trigger_buzzer():
    global buzz
    global guessV
    global fixed
    global name
    global led_pwm
    absV = abs(fixed-guessV)
    buzz.start(0)
    buzz.ChangeDutyCycle(0)
    start = time.time()
    if(absV==3):
        buzz.ChangeDutyCycle(50)
        while True:
            time.sleep(0.1)
            try:
                buzz.ChangeFrequency(1)
                time.sleep(0.1)
                stop = time.time()
                if((stop-start)>5):
                    stopAlertor()
                    break
            except Exception as e:
                print("some error")
    if(absV==2):
        buzz.ChangeDutyCycle(50)
        while True:
            time.sleep(0.1)
            try:
                buzz.ChangeFrequency(2)
                time.sleep(0.1)
                stop = time.time()
                if(stop-start>5):
                    stopAlertor()
```

```
                break
            except Exception as e:
                print("some error")
    if(absV==1):
        buzz.ChangeDutyCycle(50)
        while True:
            time.sleep(0.1)
            try:
                buzz.ChangeFrequency(4)
                time.sleep(0.1)
                stop = time.time()
                if(stop-start>5):
                    stopAlertor()
                    break
            except Exception as e:
                print("some error")
    if(absV == 0):
        print("Correct!!!!")
        name = input("Enter name with 3 or more characters:\n")
        save_scores()
        led_pwm = None
        buzz = None
        guess_num = 0
        name = ""
        GPIO.remove_event_detect(btn_submit)
        GPIO.remove_event_detect(btn_increase)
        GPIO.output(LED_value[0],False)
        GPIO.output(LED_value[1],False)
        GPIO.output(LED_value[2],False)
        GPIO.output(buzzer,False)
        GPIO.cleanup()
        setup()
        welcome()
        while True:
            time.sleep(0.1)
            menu()
            pass
    stopAlertor()
    time.sleep(0.1)
```

The `timer` method is as follows:

```
def timer(x):
        GPIO.output(33, True)
        time.sleep(x)
        GPIO.output(33, False)
        time.sleep(x)
```

The `stopAlertor` method is as follows:

```
def stopAlertor():
    buzz.stop()
```

The `startAlertor` method is as follows:

```
def startAlertor():
    buzz.start(50)
```

Finally the main program when executed does the following:

```
if __name__ == "__main__":
    try:
        # Call setup function
        setup()
        welcome()
        while True:
            menu()
            pass
    except Exception as e:
        print(e)
    finally:
        GPIO.cleanup()
```