Practical 3 Number Shuffle

Done By Jonathan Campbell and Akhil Jacob

The initialisations and imports are as follows:

```
# Import libraries
import RPi.GPIO as GPIO
import random
import ES2EEPROMUtils
import os
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
guess = 0
scores = 0
ran_num = 0
name = ""
# DEFINE THE PINS USED HERE
LED_value = [11, 13, 15]
LED_accuracy = 32
btn_submit = 16
btn_increase = 18
buzzer = 33
eeprom = ES2EEPROMUtils.ES2EEPROM()
PWM_one = None
PWM_two = None
# 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:

```
# Print the game menu
def menu():
    global end_of_game, ran_num
    end_of_game = False
    #eeprom.populate_mock_scores()
    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")
        ran_num = generate_number()
        #print (ran_num)
        while not end_of_game:
            pass
    elif option == "Q":
        print("Come back soon!")
        GPIO.cleanup()
        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
    print("There are {} scores. Here are the top 3!".format(*count))
    # print out the scores in the required format
    for i in range(3):
        print("{} - {} took {} guesses".format(i+1,raw_data[i][0],raw_data[i][1
]))
```

The Setup method is as follows:

```
# Setup Pins
def setup():
    global PWM_one,PWM_two
    # Setup board mode
    GPIO.setmode(GPIO.BOARD)
    # Setup regular GPIO
    GPIO.setup(LED_value[0],GPIO.OUT)
    GPIO.setup(LED_value[1],GPIO.OUT)
    GPIO.setup(LED_value[2],GPIO.OUT)
    GPIO.output(LED_value[0],0)
    GPIO.output(LED_value[1],0)
    GPIO.output(LED_value[2],0)
    GPIO.setup(LED_accuracy,GPIO.OUT)
    PWM_one = GPIO.PWM(LED_accuracy,1000)
    PWM_one.start(0)
    # Buzzer PWM channel
    GPIO.setup(buzzer,GPIO.OUT)
    PWM_two =GPIO.PWM(buzzer,1)
    PWM_two.start(0)
    # Setup PWM channels EEPROM
    GPIO.setup(5,GPIO.OUT)
    GPIO.setup(3,GPIO.OUT)
    # Setup debouncing and callbacks
    GPIO.setup(btn_submit,GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(btn_increase,GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.add_event_detect(btn_submit, GPIO.FALLING, btn_guess_pressed, bounceti
me=300)
    GPIO.add_event_detect(btn_increase, GPIO.RISING, btn_increase_pressed, boun
cetime=300)
```

The Fetch Scores method is as follows:

```
# Load high scores
def fetch_scores():
    # get however many scores there are
    global score_count
    score_count = eeprom.read_block(0,1)[0]
    time.sleep(0.01)
    # Get the scores
    score = []
    for i in range(score_count):
        List = eeprom.read_block((i+1)*10,16)
        time/sleep(0.01)
        name = ''
        l=0
        while l<=15:
            l+=1
            if List[l]!=None:
                name += chr(List[l])
        score = List[15]
        scores.append([name,score])
    # return back the results
    return score_count, scores
```

The Save Scores method is as follows:

```
# Save high scores
def save_scores():
    # fetch scores
    score_count,scores = fetch_scores()
    # include new score
    name = input("Please enter your name")
    new_score = [name,score]
    # update total amount of scores
    score_count += 1
    time.sleep(0.01)
    eeprom.write_block(0,score_count)
    time.sleep(0.01)
    # sort
    scores.append(new_scores)
    scores = sorted(scores, key=lambda x: x[1])
    # write new scores
    for i, high_scores in enumerate(scores):
        write = []
        for letter in high_scores[0]:
            write.append(ord(letter))
        while (len(data_to_write)<15):
            data_to_write.append(None)
        data_to_write.append(high_scores[1])
        time.sleep(0.01)
        eeprom.write_block((i+1)*10,data_to_write,4)
        time.sleep(0.01)
```

The Generate number method is as follows:

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

The LED cleanup method is as follows:

```
def LED_cleanup():
    global LED_value
    GPIO.output(LED_value[0],0)
    GPIO.output(LED_value[1],0)
    GPIO.output(LED_value[2],0)
```

The Button Increase pressed method is as follows:

```
# Increase button pressed
def btn_increase_pressed(channel):
    # Increase the value shown on the LEDs
    # You can choose to have a global variable store the user's current guess,
    # or just pull the value off the LEDs when a user makes a guess
    global guess, LED_value
    LED_cleanup()
    trigger_buzzer()
    accuracy_leds()
    if guess<=6:
        guess +=1
        print ("guess + 1 = ",guess)
    else:
        print ("Max guess number set guess back to 0")
        guess = 0

    if (guess == 1):
        GPIO.output(LED_value[0],1)
    elif (guess == 2):
        GPIO.output(LED_value[1],1)
    elif (guess == 3):
        GPIO.output(LED_value[0],1)
        GPIO.output(LED_value[1],1)
    elif (guess == 4):
        GPIO.output(LED_value[2],1)
    elif (guess == 5):
        GPIO.output(LED_value[0],1)
        GPIO.output(LED_value[2],1)
    elif (guess == 6):
        GPIO.output(LED_value[1],1)
        GPIO.output(LED_value[2],1)
    elif (guess ==7):
        GPIO.output(LED_value[0],1)
        GPIO.output(LED_value[1],1)
        GPIO.output(LED_value[2],1)
    else:
        LED_cleanup()
```

The Button Guess pressed method is as follows:

```
# Guess button
def btn_guess_pressed(channel):
    # If they've pressed and held the button, clear up the GPIO and take them b
ack to the menu screen
    # Compare the actual value with the user value displayed on the LEDs
    # Change the PWM LED
    # if it's close enough, adjust the buzzer
    # if it's an exact guess:
    # - Disable LEDs and Buzzer
    # - tell the user and prompt them for a name
    # - fetch all the scores
    # - add the new score
    # - sort the scores
    # - Store the scores back to the EEPROM, being sure to update the score cou
nt
    global guess, ran_num, btn_submit
    start = time.time()
    time.sleep(1)
    if GPIO.input(btn_submit) == GPIO.LOW:
        LED_cleanup()
        #GPIO.cleanup()
        PWM_one.stop()
        PWM_two.stop()
        guess = 0
        end_of_game = True
        menu()
    else:
        LED_cleanup()
        PWM_one.stop()
        PWM_two.stop()
        save_scores()
        end_of_game = True
        menu()
```

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 answ
er
    # - The % brightness should be directly proportional to the % "closeness"
    # - For example if the answer is 6 and a user guesses 4, the brightness sho
uld be at 4/6*100 = 66%
    # - If they guessed 7, the brightness would be at ((8-7)/(8-6)*100 = 50%

    global guess, PWM_one, ran_num
    PWM_one.start(50)
    if (guess <= ran_num):
        PWM_one.ChangeDutyCycle(int(round((guess/ran_num)*100)))
        print ("guess <= ran_num", int(round((guess/ran_num)*100)))
    else:
        PWM_one.ChangeDutyCycle(int(round(((8-guess)/(8-ran_num))*100)))
        print ("else", int(round(((8-guess)/(8-ran_num))*100)))
```

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 th
e 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 onc
e every second
    # If the user is off by an absolute value of 2, the buzzer should sound twi
ce every second
    # If the user is off by an absolute value of 1, the buzzer should sound 4 t
imes a second

    global guess, ran_num

    PWM_two.start(50)
    a=0
    a = abs(guess-ran_num)
    PWM_two.ChangeFrequency((4 * 2**(1 - a)))
```

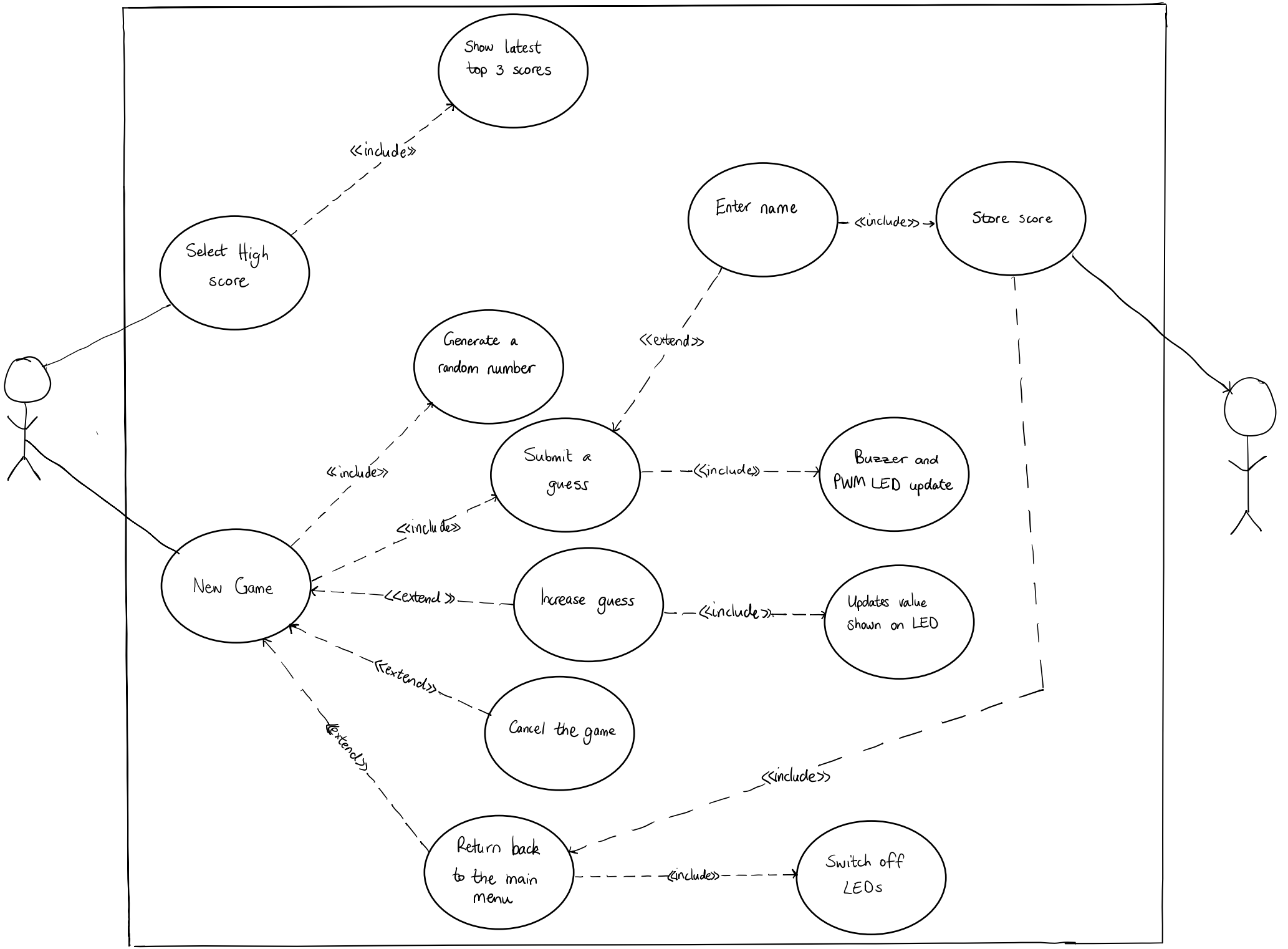
Finally the main program when executed does the following:

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

Demo Video is in the Github folder

GITHUB LINK

https://github.com/Jonathan5320/EEE3096S.git


Show latest top 3 scores
«include»
Select High score
Enter name
«include»
Store score
Generate a random number
«extend»
«include»
Submit a guess
«include»
Buzzer and PWM LED update
«include»
New Game
«extend»
Increase guess
«include»
Updates value shown on LED
«extend»
Cancel the game
«extend»
«include»
Return back to the main menu
«include»
Switch off LEDs