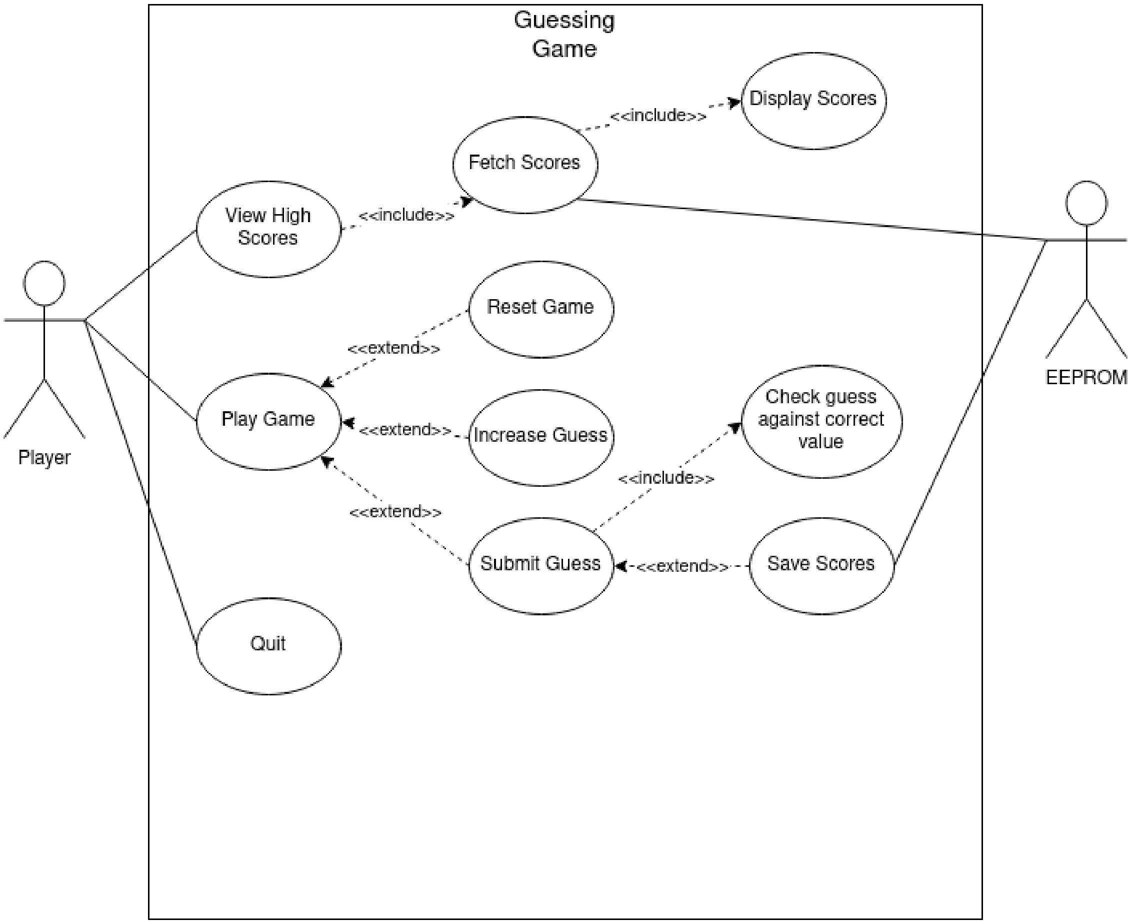
Prac3_BTJMAL001_LNGANG002

September 13, 2021

1 EEE3096S Practical 3

1.1 BTJMAL001 LNGANG002

1.1.1 Report



The intiazliations and imports are as follows:

[ ]:

```
# Import libraries
import RPi.GPIO as GPIO
import random
import ES2EEPROMUtils
import os
import time
from math import ceil

# 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_buzzer = None
USER_GUESS = 0
CORRECT_VALUE = None
GUESS_ATTEMPTS = 0
last_interrupt_time = 0

# DEFINE THE PINS USED HERE
LED_value = [11, 13, 15]
LED_accuracy = 32
btn_submit = 16
btn_increase = 18
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␣
 ↪Fame!")
```

The *menu* method is as follows:

[ ]:
```
# Print the game menu
def menu():
    global end_of_game, CORRECT_VALUE
    end_of_game = None
```

```python
    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")
        CORRECT_VALUE = generate_number()
        #print(f"Correct Value = {CORRECT_VALUE}")
        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:

[ ]:

```python
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(i+1, ". ", raw_data[i][0], ": ", raw_data[i][1], sep="")
```

The *setup* method is as follows:

[ ]:

```python
# Setup Pins
def setup():
    # Setup board mode
    GPIO.setmode(GPIO.BOARD)

    # Setup regular GPIO
    # LEDS
    for i in range(3):
        GPIO.setup(LED_value[i],GPIO.OUT)
        GPIO.output(LED_value[i], GPIO.LOW)
    GPIO.setup(LED_accuracy, GPIO.OUT)
    # Btns
    GPIO.setup(btn_submit, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.setup(btn_increase, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    # Buzzer
    GPIO.setup(buzzer, GPIO.OUT)

    global pwm_led, pwm_buzzer
    # Setup PWM channels
    pwm_led = GPIO.PWM(LED_accuracy, 1000)
    pwm_led.start(0)
    pwm_buzzer= GPIO.PWM(buzzer, 1)
    pwm_buzzer.start(0)

    # Setup debouncing and callbacks
    GPIO.add_event_detect(btn_increase, GPIO.FALLING,␣
↪callback=btn_increase_pressed, bouncetime=200)
    GPIO.add_event_detect(btn_submit, GPIO.FALLING,␣
↪callback=btn_guess_pressed,bouncetime=200)
```

The *fetch_scores* method is as follows:

[ ]:
```
# Load high scores
def fetch_scores():
    global eeprom
    score_count = eeprom.read_byte(0)
    scores = []

    # Get the scores
    for i in range(score_count):
        scores.append(eeprom.read_block(i+1, 4))

    # convert the codes back to ascii
    for i in range(len(scores)):
        name = chr(scores[i][0]) + chr(scores[i][1]) + chr(scores[i][2])
        scores[i] = [name, scores[i][3]]

    scores.sort(key=lambda x: x[1])
    # return back the results
    return score_count, scores
```

The *trim_name* method is as follows:

[ ]:
```
def trim_name(name):
    # Trim name to 3 characters
    user_name = name[0] + name[(len(name)//2)] + name[-1]
    return user_name
```

The *save_scores* method is as follows:

[ ]:

```
# Save high scores
def save_scores():
    score_count, scores = fetch_scores()
    user_name = input("Please enter your name: ")
    if len(user_name) > 3:
        user_name = trim_name(user_name)
    # include new score
    scores.append([user_name, GUESS_ATTEMPTS])
    # sort
    scores.sort(key=lambda x: x[1])
    # update total amount of scores
    score_count += 1
    eeprom.write_block(0, [score_count])

    # write new scores
    data_to_write = []
    for letter in user_name:
        data_to_write.append(ord(letter))
    data_to_write.append(GUESS_ATTEMPTS)

    eeprom.write_block(score_count, data_to_write)
    print("Writing Scores")
```

The *generate_number* method is as follows:

[ ]:

```
# Generate 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 USER_GUESS, last_interrupt_time

    # debounce
    start_time = time.time()
    if (start_time - last_interrupt_time > 0.2):

        USER_GUESS += 1
        USER_GUESS = USER_GUESS % 8
        value_dict = {
            0: [GPIO.LOW, GPIO.LOW, GPIO.LOW],
            1: [GPIO.LOW, GPIO.LOW, GPIO.HIGH],
            2: [GPIO.LOW, GPIO.HIGH, GPIO.LOW],
            3: [GPIO.LOW, GPIO.HIGH, GPIO.HIGH],
            4: [GPIO.HIGH, GPIO.LOW, GPIO.LOW],
            5: [GPIO.HIGH, GPIO.LOW, GPIO.HIGH],
            6: [GPIO.HIGH, GPIO.HIGH, GPIO.LOW],
            7: [GPIO.HIGH, GPIO.HIGH, GPIO.HIGH],
            }
        # Increase the value shown on the LEDs
        for i in range(3):
            GPIO.output(LED_value[i], value_dict[USER_GUESS][i])
    last_interrupt_time = start_time
```

The *reset_GPIO* method is as follows:

[ ]:

```
def resetGPIO():
    global USER_GUESS, pwm_led, pwm_buzzer, last_interrupt_time
    last_interrupt_time = 0
    # Set the User guess to 7
    USER_GUESS = 7
    # then increase it and set the LEDS to correct value
    btn_increase_pressed(0)
    # Turn accuracy LED and buzzer off
    pwm_led.ChangeDutyCycle(0)
    pwm_buzzer.ChangeDutyCycle(0)
```

The *btn_guess_pressed* method is as follows:

[ ]:

```
# Guess button
def btn_guess_pressed(channel):
    global end_of_game, last_interrupt_time, GUESS_ATTEMPTS
    GUESS_ATTEMPTS += 1
    long_press = False
    start_time = time.time()
    # Measure the button press time
    while GPIO.input(btn_submit) == GPIO.LOW:
        time.sleep(0.1)
        btn_press_length = time.time() - start_time
        if btn_press_length > 1:
            long_press = True
            break

    # If longer than a second, wait for button release
    if long_press:
        print("Waiting for button release")
        while GPIO.input(btn_submit) == GPIO.LOW:
            pass
        # Reset GPIO and restart the game
        resetGPIO()
        welcome()
        end_of_game = True
    else:
        # If its a short press
        # debounce
        # Check if game is won, otherwise update LED and buzzer
        if (start_time - last_interrupt_time > 0.2):
            if USER_GUESS == CORRECT_VALUE:
                game_win()
                pass
            accuracy_leds()
            trigger_buzzer()
        last_interrupt_time = start_time
```

The *game_win* method is as follows:

[ ]:
```
def game_win():
    # Procedure for game win
    global GUESS_ATTEMPTS, end_of_game
    # Turn off GPIO
    pwm_led.stop()
    pwm_buzzer.stop()
    for i in range(3):
        GPIO.output(LED_value[i], GPIO.LOW)

    USER_GUESS = 0
    print(f"Congratulations! You guessed correctly!! It took you␣
 ↪{GUESS_ATTEMPTS}", "guesses!" if GUESS_ATTEMPTS > 1 else "guess!")
    # Store score
    save_scores()
    end_of_game = True
```

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
    led_val = 100.0-(abs(USER_GUESS-CORRECT_VALUE)/7*100.0)
    pwm_led.ChangeDutyCycle(round(led_val))
```

The *trigger_buzzer* method is as follows:

[ ]:
```
# Sound Buzzer
def trigger_buzzer():
    # The buzzer duty cycle should be left at 50%
    pwm_buzzer.ChangeDutyCycle(50.0)
    # If the user is off by an absolute value of 3, the buzzer should sound␣
 ↪once every second
    if (abs(USER_GUESS-CORRECT_VALUE) == 3):
        pwm_buzzer.ChangeFrequency(1)
```

```python
    # If the user is off by an absolute value of 2, the buzzer should sound twice every second
    elif (abs(USER_GUESS-CORRECT_VALUE) == 2):
        pwm_buzzer.ChangeFrequency(2)
    # If the user is off by an absolute value of 1, the buzzer should sound 4 times a second
    elif (abs(USER_GUESS-CORRECT_VALUE) == 1):
        pwm_buzzer.ChangeFrequency(4)
    else:
        pwm_buzzer.ChangeDutyCycle(0)
```

Finally the main program when executed does the following:

[ ]:

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