Device Communications for the Internet of Things Report
Snake-attack!

1. Project Idea

This report details the project, which involved building a snake game controlled by Arduino boards and a Python script game running on a laptop. In this project, I used two Arduino boards: one acting as a master and the other as a slave.

- The master Arduino board was responsible for interacting with a Python script and controlling a buzzer,
- while the slave Arduino board was connected to a joystick for controlling the snake movement in the game
- The game will be displayed on the laptop that is connected to the Master

2. Implementation Details

Arduino Board Connections

I connected the two Arduino boards using the Two-Wire-Interface (TWI) communication protocol, which allows for communication betIen a master (connected to the PC & Buzzer) and a slave (connected to the joystick controller). The master Arduino served as the bridge betIen the PC and the slave Arduino, enabling us to exchange data.

Joystick Integration

In this setup, the slave Arduino, assigned the I2C address 2, reads analog values from analog pin A0 and analog pin A1 connected to the joystick's X and Y values respectively. When a request is received from the master, a requestHandler function stores the collected values in an array to be passed from slave to master.

At the master arduino, a "*newData*" flag is used to serve as an indicator that new data is now available for retrieval. For debugging purposes, the values read from the analog pins A0 and A1, are also printed to the serial monitor.

Buzzer Integration

The master Arduino was equipped with a buzzer programmed to respond to specific events in the Python script. When the snake collided with the food or goes into Game-Over condition, a command is sent to the master Arduino to activate the buzzer, providing audio feedback to the player.

Python Game Script Integration

The master Arduino continuously checks if there are any incoming commands from the Python script using "*Serial.available()*". When a command is received, the arduino will activate the buzzer according to 3 conditions:

1. On default, there will be no buzzer sounds and the RGB LED will shine bright Green.
2. The "B" command reflects a game-over event, the buzzer will sound for 2 seconds and the RBG LED will shine bright Red.
3. The "F" command reflects a food event, buzzer sounds a short beep and the RGB LED will continue to shine bright Green..

The Python script uses the Pygame library to create a graphical interface for the snake game. Various different functions will draw the game elements (snake, food, wall, status) and render it on screen. A separate thread (*read_joystick_thread)* is created to read joystick values from the Arduino via a serial connection and updates the python variables.

When certain game events occur, such as the snake consuming food or the player losing, the Python script sends commands to the master Arduino via the serial connection. The 'F' command is sent to indicate a food event, and the length of the snake is passed as a parameter. The 'B' command is sent to indicate a game over event.

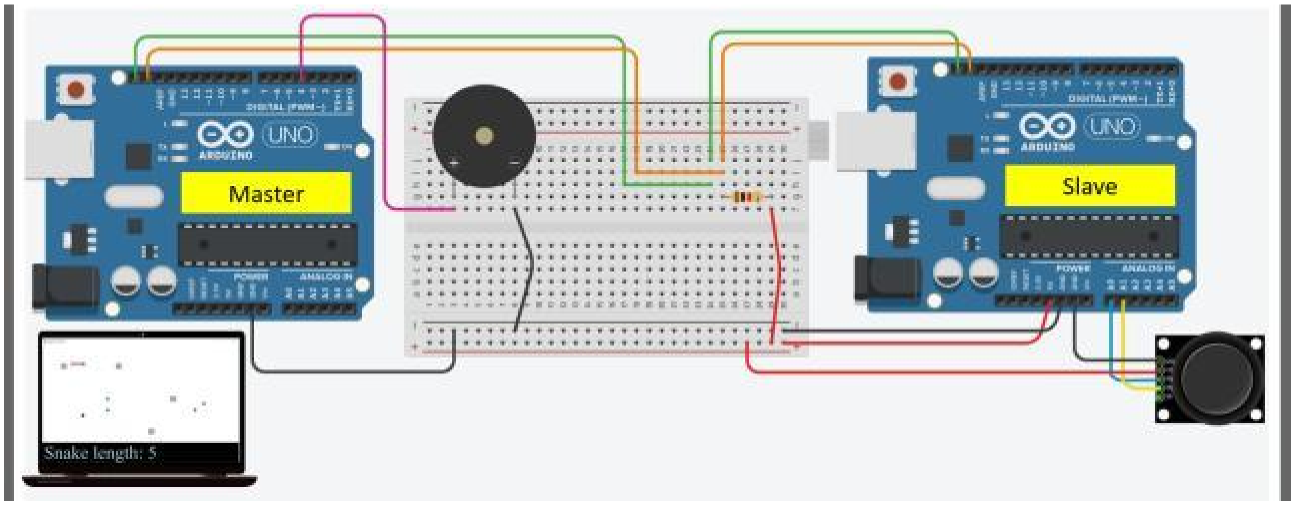
Future Work & its Challenges

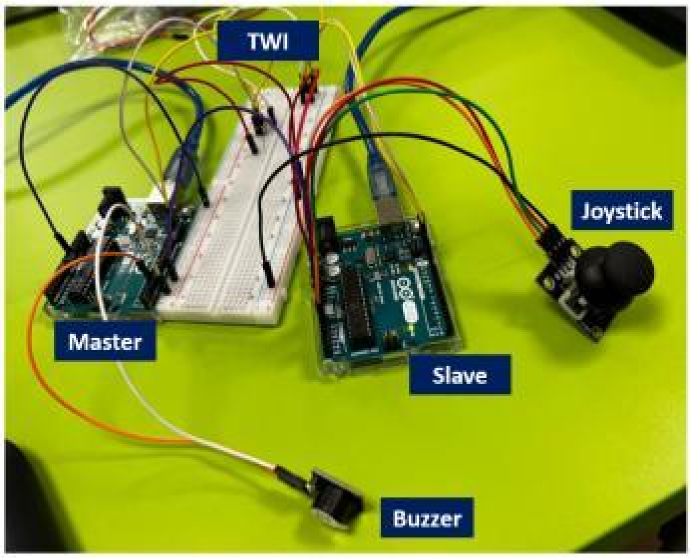
#1 - Actuation to the RGB LED color intensity (dimmer to brighter) based on the increasing snake length. Challenges - more actuations causes more time in the Arduinos to process and creates lag-time to reflect the user's joystick directions

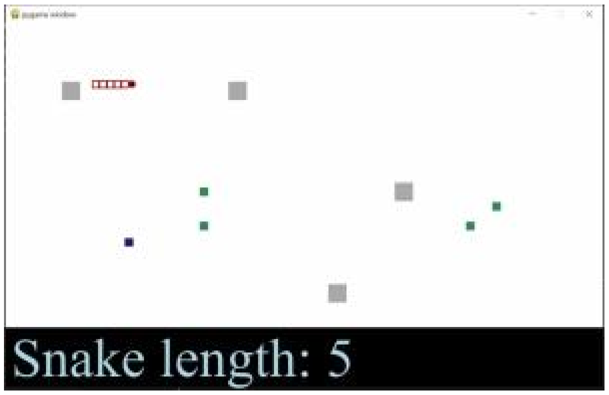
#2 Integration of a button for the user to restart the game or voluntarily end the game. Challenges - initializing the sensor onto Arduino and further python script enhancements.

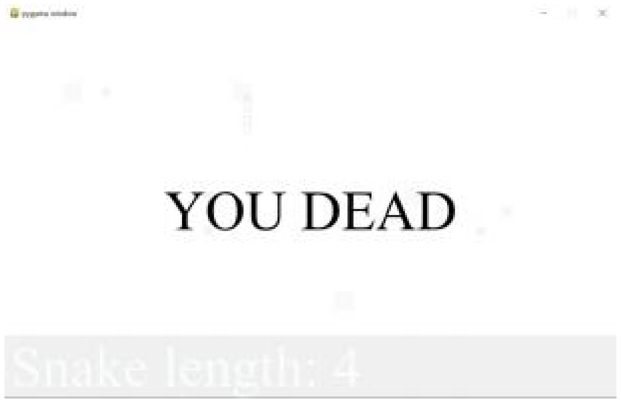
#3 Using a rotary encoder for the user to increase / decrease the speed of the snake's movement and the difficulty of the game. Challenges - initializing the sensor onto Arduino and further python script enhancements.

3. Circuit Connections









Appendix: Sthece Code

Arduino Sketch (Slave):

```
#include <Wire.h>
int newData=0;
uint16_t buffer[2];

//the request handler
void requestHandler(){
//Read from the ADC at I02 and I04
buffer[0]=analogRead(0);
buffer[1]=analogRead(1);
Wire.write((uint8_t*)buffer,sizeof(buffer));
//Set the newData flag
newData=1;
}

void setup(){
//put ythe setup code here,to run once:
Wire.begin(2);//I are slave #2
Serial.begin(115200);
Wire.onRequest(requestHandler);
}

void loop(){
//put ythe main code here,to run repeatedly:
//Loop until there is new data.newData is set by
//request handler.
while(!newData)
  delay(50);

newData =0;

//Write the values read
Serial.print("x(slave)=");
Serial.print(buffer[0]);
Serial.print("y(slave)=");
Serial.println(buffer[1]);
}
```

Arduino Sketch (Master):

```
#include <Wire.h>
int buzzerPin = 4;  // Buzzer pin
int greenPin = 10;  // Green LED pin
int redPin = 9;     // Red LED pin

void setup() {
  Serial.begin(115200);
  Wire.begin();
  pinMode(buzzerPin, OUTPUT);    // set buzzer pin as OUTPUT
  pinMode(greenPin, OUTPUT);     // set green LED pin as OUTPUT
  pinMode(redPin, OUTPUT);       // set red LED pin as OUTPUT
  digitalWrite(buzzerPin, LOW);  // turn off buzzer initially
  digitalWrite(greenPin, HIGH);   // turn off green LED initially
  digitalWrite(redPin, LOW);     // turn off red LED initially
}

void loop() {

  uint16_t data[2];
  char *ptr;

  ptr = (char *)data;

  Wire.requestFrom(2, 4);
  while (Wire.available() > 0) {
    *ptr = Wire.read();
    ptr++;
  }

  Serial.print(data[0]);
  Serial.print(",");
  Serial.println(data[1]);

  // Check if there's any incoming command from Python
  if (Serial.available()) {
    char command = Serial.read();
    switch (command) {
      case 'B':                          // Dead
        digitalWrite(buzzerPin, HIGH);   // turn on buzzer
        delay(2000);                     //  keep it on for 2 seconds
        digitalWrite(buzzerPin, LOW);    // turn off buzzer
        digitalWrite(redPin, HIGH);      // turn on red LED
        digitalWrite(greenPin, LOW);     // ensure green LED is off
        break;
      case 'F':                                // Food
        // int snakeLength = Serial.parseInt();  // Read the integer value for
snake length
        digitalWrite(buzzerPin, HIGH);         // short beep
        delay(200);
        digitalWrite(buzzerPin, LOW);
        // int intensity = map(snakeLength, 5, 20, 0, 255);  // Map snake length
to intensity
        // analogWrite(greenPin, intensity);                 // Set intensity of
green LED
        // digitalWrite(redPin, LOW);                        // Ensure red LED is
off
        break;

    }
  }


delay(100);
}
```

Python script:

```
import sys
import random
import pygame
from pygame.color import THECOLORS as COLORS
from pygame.locals import *

import serial
import threading
import time


def draw_background():
    # white background
    screen.fill(COLORS[ 'white'])
    pygame.draw.rect(screen, COLORS[ 'black'],
                     (-100, GAME_SIZE[1], 3000, 200), 0)


def draw_wall():
    for xy in wall_list:
        pygame.draw.rect(screen, COLORS[ 'darkgray'], (xy[0]-WALL_WIDTH/2,
                         xy[1]-WALL_WIDTH/2, WALL_WIDTH, WALL_HEIGHT), 0)


def draw_snake():
    head = snake_list[0]
    pygame.draw.circle(screen, COLORS[ 'darkred'],
                       (head[0], head[1]), int(SNAKE_WIDTH/2), 0)
    for xy in snake_list[1:]:
        pygame.draw.rect(screen, COLORS[ 'darkred'], (xy[0]-SNAKE_WIDTH/2,
                         xy[1]-SNAKE_WIDTH/2, SNAKE_WIDTH, SNAKE_HEIGHT), 2)


def draw_food():
    for xyz in food_list:
        pygame.draw.rect(screen, FOOD_COLORS[xyz[2]-1], (xyz[0] -
                         FOOD_WIDTH/2, xyz[1]-FOOD_WIDTH/2, FOOD_WIDTH,
FOOD_HEIGHT), 0)
```

```python
def draw_context():
    txt = FONT_M.render(
        'Snake length: '+str(len(snake_list)-1), True, COLORS[ 'lightblue'])
    x, y = 10, GAME_SIZE[1]+(int((SIZE[1]-GAME_SIZE[1])/2))
    y = int(y-FONT_M.size('Count')[1]/2)
    screen.blit(txt, (x, y))


def draw_pause():
    s = pygame.Surface(SIZE, pygame.SRCALPHA)
    s.fill((255, 255, 255, 220))
    screen.blit(s, (0, 0))
    txt = FONT_M.render( 'PAUSE', True, COLORS['darkgray'])
    x, y = SIZE[0]/2, SIZE[1]/2
    x, y = int(x-FONT_M.size('PAUSE')[0]/2), int(y-FONT_M.size('PAUSE')[1]/2)
    screen.blit(txt, (x, y))


def draw_dead():
    s = pygame.Surface(SIZE, pygame.SRCALPHA)
    s.fill((255, 255, 255, 240))
    screen.blit(s, (0, 0))
    txt = FONT_M.render( 'YOU DEAD', True, COLORS['black'])
    x, y = SIZE[0]/2, SIZE[1]/2
    x, y = int(x-FONT_M.size( 'YOU DEAD')
               [0]/2), int(y-FONT_M.size( 'YOU DEAD')[1]/2)
    screen.blit(txt, (x, y))


def rect_cover(rect1, rect2):
    left1 = int(rect1[0])
    right1 = int(rect1[0]+rect1[2])
    up1 = int(rect1[1])
    down1 = int(rect1[1]+rect1[3])
    left2 = int(rect2[0])
    right2 = int(rect2[0]+rect2[2])
    up2 = int(rect2[1])
    down2 = int(rect2[1]+rect2[3])
    if not (right2 <= left1 or left2 >= right1 or down2 <= up1 or up2 >= down1):
        return True
    return False


def add_food():
    while (True):
        xyz = [random.choice(X_LIST), random.choice(
            Y_LIST), random.choice([1, 2, 3, 4])]
        if xyz not in wall_list:
            food_list.append(xyz)
            break


def add_body(length=1):
    for c in range(length):
        # 尾巴加一节
        last2, last1 = snake_list[-2], snake_list[-1]
        if last2[0] == last1[0]:  # 竖着的两段
            if last2[1] > last1[1]:  # 朝下
                snake_list.append([last1[0], last1[1]-SNAKE_WIDTH])
            else:
                snake_list.append([last1[0], last1[1]+SNAKE_WIDTH])
        else:  # 横着的两段
            if last2[0] > last1[0]:  # 朝右
                snake_list.append([last1[0]-SNAKE_WIDTH, last1[1]])
            else:
                snake_list.append([last1[0]+SNAKE_WIDTH, last1[1]])


def check_food():
    # 头与食物
    first = snake_list[0]
    snake_head_rect = (first[0]-SNAKE_WIDTH/2, first[1] -
                       SNAKE_WIDTH/2, SNAKE_WIDTH, SNAKE_HEIGHT)
    for i in range(len(food_list)):
        xyz = food_list[i]
        food_rect = (xyz[0]-FOOD_WIDTH/2, xyz[1] -
                     FOOD_WIDTH/2, FOOD_WIDTH, FOOD_HEIGHT)
        if rect_cover(snake_head_rect, food_rect):
            add_body(xyz[2])
            snake_length = len(snake_list) - 1
            # 'F' command for Food, folloId by the snake length
            ser.write( 'F{}'.format(snake_length).encode())
            del food_list[i]
            return True
    return False


def check_dead():
    first = snake_list[0]
    snake_head_rect = (first[0]-SNAKE_WIDTH/2, first[1] -
                       SNAKE_WIDTH/2, SNAKE_WIDTH, SNAKE_HEIGHT)
    # 头与边缘
    if first[0] < 0 or first[0] > GAME_SIZE[0] or first[1] < 0 or first[1] >
GAME_SIZE[1]:
        return True
    # 头与墙壁
    for xy in wall_list:
        wall_rect = (xy[0]-WALL_WIDTH/2, xy[1] -
                     WALL_WIDTH/2, WALL_WIDTH, WALL_HEIGHT)
        if rect_cover(snake_head_rect, wall_rect):
            return True
    # 头与自身
    for xy in snake_list[1:]:
        body_rect = (xy[0]-SNAKE_WIDTH/2, xy[1] -
                     SNAKE_WIDTH/2, SNAKE_WIDTH, SNAKE_HEIGHT)
        if rect_cover(snake_head_rect, body_rect):
            return True
    return False


def read_joystick_values():
    try:
        # Read data from Arduino over serial port
        data = ser.readline().decode('utf-8').strip()
        print("Received data: ", str(data))  # show the raw data received
        x_val, y_val = [int(i) for i in data.split(",")]
```

```python
        return x_val, y_val
    except ValueError:
        return 508, 521


def beep_buzzer():
    ser.write(b'B')


# def eat_buzzer():
#     ser.write(b'b')


# Global variables
joystick_values = (508, 521)
joystick_values_lock = threading.Lock()


def read_joystick_thread():
    global joystick_values
    while running and not dead:
        x_val, y_val = read_joystick_values()
        with joystick_values_lock:
            joystick_values = (x_val, y_val)


if __name__ == "__main__":
    # init pygame
    pygame.init()

    # Connect to Arduino over serial
    ser = serial.Serial( 'COM6', 115200)
    time.sleep(2)

    infoObject = pygame.display.Info()
    SCREEN_WIDTH, SCREEN_HEIGHT = infoObject.current_w, infoObject.current_h
    # to leave a margin around the game for other UI elements:
    MARGIN = 300
    GAME_SIZE = [SCREEN_WIDTH - MARGIN, SCREEN_HEIGHT - MARGIN]
```

```
original_game_size = [900, 900]
scaling_factor_x = GAME_SIZE[0] / original_game_size[0]
scaling_factor_y = GAME_SIZE[1] / original_game_size[1]

# constant
SIZE = [GAME_SIZE[0], GAME_SIZE[1]+100]
FONT_S = pygame.font.SysFont( 'Times', 50)
FONT_M = pygame.font.SysFont( 'Times', 90)
DIRECTION = ['up', 'right', 'down', 'left']
X_LIST = [x for x in range(GAME_SIZE[0])]
Y_LIST = [y for y in range(GAME_SIZE[1])]
FOOD_COLORS = ((46, 139, 87), (199, 21, 133), (25, 25, 112), (255, 215, 0))

# wall
wall_list = [[100, 200], [600, 500], [350, 200], [500, 800]]
wall_list = [[int(x * scaling_factor_x), int(y * scaling_factor_y)]
             for x, y in wall_list]
WALL_WIDTH, WALL_HEIGHT = 30, 30

# food
food_list = [(150, 200, 1), (300, 500, 1), (740, 542, 1),
             (300, 600, 1), (700, 600, 1)]
food_list = [[int(x * scaling_factor_x), int(y * scaling_factor_y), z]
             for x, y, z in food_list]
FOOD_WIDTH, FOOD_HEIGHT = 14, 14

# create screen 500*500
screen = pygame.display.set_mode(SIZE)

# variable parameter
snake_list = [[100+12*4, 300], [100+12*3, 300],
              [100+12*2, 300], [100+12*1, 300], [100, 300]]
SNAKE_WIDTH, SNAKE_HEIGHT = 12, 12
snake_v = 0
count_time = 0

# level
frame = 0.05
```

```python
level = 1

key_states = {"left": False, "right": False, "up": False, "down": False}

# main loop
running = True
pause = False
dead = False
head = 'right'

# Start a separate thread for reading joystick values
joystick_thread = threading.Thread(target=read_joystick_thread)
joystick_thread.start()

while running:
    with joystick_values_lock:
        x_val, y_val = joystick_values
    if x_val is not None and y_val is not None:
        # Use the joystick's X and Y values to determine movement direction
        if y_val <= 100:  # Adjust the threshold as needed
            head = 'left'
        elif y_val >= 900:  # Adjust the threshold as needed
            head = 'right'
        elif x_val >= 900:  # Adjust the threshold as needed
            head = 'up'
        elif x_val <= 100:  # Adjust the threshold as needed
            head = 'down'

    # update data
    if not pause and not dead:
        count_time += frame*level
        first = snake_list[0]
        snake_list[1:] = snake_list[:-1]
        if head == 'up':
            snake_list[0] = [first[0], first[1]-SNAKE_WIDTH]
        elif head == 'down':
            snake_list[0] = [first[0], first[1]+SNAKE_WIDTH]
        elif head == 'left':
            snake_list[0] = [first[0]-SNAKE_WIDTH, first[1]]
        elif head == 'right':
            snake_list[0] = [first[0]+SNAKE_WIDTH, first[1]]

    # background
    draw_background()
    # tunnel
    draw_wall()
    # choose item
    draw_snake()
    # food
    draw_food()
    # point
    draw_context()
    # pause
    if not dead and pause:
        draw_pause()

    # dead
    if dead:
        beep_buzzer()
        draw_dead()
        pygame.display.flip()

        waiting_for_input = True
        while waiting_for_input:
            for event in pygame.event.get():
                if event.type == QUIT or event.type == KEYDOWN:
                    waiting_for_input = False
                    break
        # Break out of the main loop after handling the dead state.
        break

    # flip
    pygame.display.flip()
    # pause 50ms
    pygame.time.delay(int(frame/level*2500))
    # check win or not
    dead = check_dead()
    if check_food():
        add_food()

# Ensure the joystick thread stops
joystick_thread.join()
ser.close()  # Close the serial connection
pygame.quit()
```