IS4151/IS5451 – AIoT Solutions and Development

AY 2024/25 Semester 2

Practical Lab 02 – Single-board Microcontroller (II)

sender and sequence number, 5 alphabet unique identifier

Part 1 – Basic Programming

refer to lab 1 code for qn 4.

check if the message has been received before, random number

PE02-1-1 – Intermediate Messenger

This exercise continues from PE01-1-4. Recall that in PE01-1-4, you have created a basic peer-to-peer messenger that assumes that all micro:bit devices that are involved in the communication are within transmission range of each other.

The micro:bit devices can be programmed to transmit at the maximum power of 7 (+4 dBm). This provides a maximum transmission range of about 70 m with direct line of sight. Suppose you would like to extend the transmission range to 1 km using a mesh network of micro:bit devices. In a mesh network, theoretically, a micro:bit device that receives a text message will rebroadcast the same message to other surrounding micro:bit devices. In this way, one device can easily send a message to another device that is more than 70 m away.

Modify the existing messaging protocol to implement a mesh network taking into consideration:

- The technical characteristics of micro:bit.
- The infinite echo problem – When recipient devices start to rebroadcast a message, very quickly, all the devices will end up infinitely rebroadcasting the rebroadcasted message.

Document your mesh messaging protocol. Explain in detail how your protocol mitigates the infinite echo problem.

sequence number can run out, also question on how long can it be.

PE02-1-2 – Advanced Messenger

This exercise continues from PE02-1-1. Suppose that the mesh network is going to be deployed in a rectangular plot of land. At one corner of the rectangle, there is a fog server implemented using a conventional desktop or laptop computer.

Modify the existing messaging protocol to allow a designated micro:bit device that is nearest to the fog server to relay the message to the fog server. Upon receiving the messages, the fog server will display them on the monitor.

Hint: The pySerial module can be used to perform serial port programming in Python easily. pySerial can be installed using the pip command `python -m pip install pySerial`

The following code fragment demonstrates how to read and write string data from/to a micro:bit device using the newline character as the delimiter:

```
import serial

ser = serial.Serial(port='COM6', baudrate=115200)

# Read newline terminated data
msg = ser.readline()
smsg = msg.decode('utf-8').strip()
print('RX:{}'.format(smsg))

# Write newline terminated data
ser.write(b'Ack\r\n')
```

PE02-1-3 – Home Security System

This is a pair exercise. Using one or more micro:bit devices together with Grove shields and Grove modules, conceptualise, design and implement a home security system that provides as many different types of protection mechanisms as possible.

The home security system should exhibit the functional requirements of a/an alert system and reactive system.

Briefly document the design of your home security system.

Part 2 – Intermediate Programming

PE02-2-1 – The CHASER! Game

Recall that the micro:bit device has a 25 LEDs arranged in a 5x5 grid. Each LED is individually addressable using a pair of (x, y) coordinates where x is horizontal, and y is vertical. The coordinates are zero-based starting from the top-left corner. The complete coordinates map is shown in the table below:

(0, 0)	(1, 0)	(2, 0)	(3, 0)	(4, 0)
(0, 1)	(1, 1)	(2, 1)	(3, 1)	(4, 1)
(0, 2)	(1, 2)	(2, 2)	(3, 2)	(4, 2)
(0, 3)	(1, 3)	(2, 3)	(3, 3)	(4, 3)
(0, 4)	(1, 4)	(2, 4)	(3, 4)	(4, 4)

The `led.plot(x,y)` function and `led.unplot(x,y)` function can be used to turn on/off each individual LED, respectively.

Using this flexible technique of controlling the micro:bit device's LEDs, write a program to implement a chaser game. In the chaser game, the micro:bit device will randomly move an enemy LED around the screen. The player will control a chaser LED to chase after the enemy

LED. The game ends when the chaser LED successfully chases or catches up with the enemy LED.

The basic reference game protocol is given in the table below:

S/N	Stage	Enemy LED	Chaser LED
1	Splash Screen	• Shows a welcome splash screen. • Prompt player to press button A to start game.	
2	Game Initialisation	• Generate a random starting coordinate except (2, 2) • Generate a random initial direction (1 – Up, 2 – Right, 3 – Down, 4 – Left) • Set initial speed to 900 ms delay, i.e., move one LED position every 900 ms.	• Set starting coordinate to (2, 2) • Set initial direction to Up • Set initial speed to 1,200 ms delay.
3	Game Play	• Enemy moves one LED position every 900 ms • During each movement, determine whether to change direction with 0.5 probability. • If need to change direction, generate new random direction before movement. • Move enemy by one position in current direction. • Movement will "wrap around" to the other side of the LED screen. E.g., if the current position is (4,2) and the enemy LED moves one position to the right, the new position will be (0,2)	• Chaser moves one LED position every X ms • The chaser moves in three speeds – 1,200 ms (slowest), 900 ms and 600 ms (fastest) • Move chaser by one position in current direction. • Movement will likewise "wrap around" to the other side of the LED screen.
4		-	• If player presses button AB, toggle the chaser's speed in this order – 1,200 ms → 900 ms → 600 ms → 1,200 ms → …
5		-	• If player presses button A, rotate the chaser direction in an anticlockwise order – Left → Down → Right → Up → Left → …
6		-	• If player presses button B, rotate the chaser direction in a clockwise order – Right → Down → Left → Up → Right → …
7	Game End	• End the game by showing a smiling face, i.e., IconNames.Happy, when the chaser LED chases or catches up with the enemy LED. • I.e., both LEDs up on the same position or (x, y) coordinates. • Press button A to restart game.	

Table 1 – Chaser basic reference game protocol.