Game Console using Nucleo board and peripherals

Embedded Processor Application
Final Project Report

지도 교수: 이종원 교수님

21400726 조정훈

21500633 전주영

Table of Contents

I. Introduction

수업시간 때 사용한 NUCLEO-F411RE 보드를 사용하여 사용자가 원하는 게임 프로그램을 자유롭게 올려서 구동할 수 있는 게임기 플랫폼을 만들어 보았다. Nucleo 보드는 지원하는 interface 모듈이 다양하고, 프로그램을 올리고 수정하는 것이 매우 간편하여 다양한 게임을 바꿔 올릴 수 있는 게임기에 적합하다고 판단하였다.

이를 위해 입력 컨트롤러로는 joystick 을 사용하였고, 출력으로는 게임 화면을 보여주는 LCD 와 게임 시작과 끝을 알려주는 buzzer 를 사용하였다. 추가적으로 블루투스 통신을 이용하여 입력 컨트롤러와 게임기 core 부분이 나누어질 수 있게 하였다.

프로젝트를 진행하면서 설정한 목표들은 다음과 같다.

- SPI 통신 프로토콜을 이해하여 게임 화면을 OLED 에 띄울 수 있다.
- Pwm 을 사용하여 게임 구동에 필요한 음을 만들 수 있다.
- joystick 을 통해 사용자가 원하는 방향의 입력이 나오게 할 수 있다.
- 블루투스 통신을 통해 입력 컨트롤러와 게임기 간의 통신이 가능하게 할 수 있다.

위와 같은 프로젝트를 수행하기 위해 snake game 을 구현해 게임기 동작을 확인하였다. Snake game 의 동작 방식은 다음과 같다. 뱀이 먹이를 먹으면 계속 몸이 길어지게 되는데, 이때 벽에 몸이 닿거나 자신의 몸에 자신이 닿으면 게임이 종료된다. 또한 먹이를 먹을수록 속도가 빨라지게 된다. 본 프로젝트에서는 3 의 배수의 먹이를 먹을 때마다 속도가 더 빨라 지게끔 구현하였다. 이때 뱀은 대각선으로 움직일 수는 없으며 위, 아래, 왼쪽, 오른쪽 방향만 갖는데, 머리와 꼬리를 가지고 있으므로 위로 가고 있다가 바로 아래로 가거나 아래로 가고 있다가 바로 위로 가는 식의 동작은 불가하다.

II. Overall System

A. 모듈에 대한 설명

1. Piezoelectric Buzzer(피에조 부저)

피에조 부저는 피에조 효과를 이용하여 소리를 내는 장치이다. 피에조 효과란 압전 효과 라고도 불리며, 압전 물질, 즉, 수정이나 세라믹 같은 결정체에 압력을 주게 되면 변형이 일어나 표면에 전압이 생기고 반대로 전압을 걸어 압전 물질이 응축 또는 신장하게 하는 것이다. 압전 물질에 얇은 판을 대어 압전 물질에 소리가 나도록 하는 것이 피에조 부저이다. 피에조 부저에 음계별로 적절한 주파수를 넣어주면 해당하는 음계의 소리가 난다. 이 프로젝트에서는 피에조 부저를 사용하여 게임 음향을 구현하였다.

게임기 본체의 역할을 하는 Nucleo board에서 피에조 부저를 pulse width modulation(PWM)신호를 사용하여 제어한다. PWM신호는 디지털 신호로 아날로그 신호를 생성해는 방법이다. PWM신호는 펄스 폭(duty cycle, 듀티 사이클)과 pwm period로 정의가 되는데 Period는 하나의 사이클이 완료되는 시간을 가리키며 듀티 사이클은 신호가 켜져 있는 상태의 시간을 하나의 사이클을 완료하는데 총 시간(period)의 백분율로 나타낸 것이다. PWM에서 주파수는 하나의 사이클을 완료할 동안 걸리는 속도를 나타내며 high와 low 상태 전환하는 속도를 결정한다. Mbed 라이브러리를 통해 PWM신호를 제어할 수 있는데 적절한 주파수와 듀티 사이클을 설정하여 소리를 제어한다.

Figure 1. Piezoelectric Buzzer

2. HM-10

HM-10는 블루투스에서 시리얼로 신호를 전환시켜주는 마스터-슬레이브 모듈이다. 해당 모듈은 Bluetooth v4.0으로 통신하며 BLE(Bluetooth Low Energy)를 지원하는데 BLE란 저전력을 이용하여 통신을 한 다는 것이다. 해당 프로젝트에서는 HM-10으로 연결된 두 보드 간의 통신을 위에서 AT command를 사용하였다.

HM-10을 Nucleo board로 연결할 때에는 UART를 사용한다. UART통신이란 연속적으로 한번에 하나의 비트 단위로 데이터를 전송하는 시리얼(Serial)통신 방법 중 하나이다. 통신 프로토콜인 SPI나 I2C와 달리 microcontroller에 있는 물리적 회로이다. UART 기기는 통신할 때 송신과 수신을 동시에 받는 full duplex(전이중) 방식을 사용하며 asynchronous(비동기)하게 데이터를 전송하는데 이는 데이터가 클럭 신호에 동기가 되지 않는다는 것을 의미한다. 클럭 시그널 대신에 UART는 전송되는 패킷에 start bit와 stop bit를 붙여 보내게 된다. UART는 데이터 수신과 송신을 TX pin과 RX pin으로 한다. TX는 parallel비트를 serial비트로 전환시켜주며 데이터 버스로 parallel하게 들어오는 것을 TX가 serial한 비트로 변환하여 상대 보드의 RX에 보낸다. RX는 받은 serial 데이터를 다시 parallel하게 변환시켜 데이터 버스로 보낸다.

Figure 2. HM-10

3. OLED Adafruit 1.44 Color TFT LCD Display

게임 그래픽을 출력하기 위해서 OLED display를 화면으로 사용하였다. 해당 OLED display는 128x128 칼라 픽셀이 있으며 SPI를 이용하여 통신을 한다. 모듈 뒤에는 mircroSD카드 리더기가 있어 bitmap 사진과 같은 파일을 읽고 쓸 수 있다.

해당 OLED는 SPI 통신을 한다. SPI(Serial Peripheral Interface)는 master와 slave 기기 간에 full duplex로 synchronous 통신을 하는 프로토콜이다. 두 개의 제어 시그널과 두개의 데이터 시그널이 있다. 제어 시그널로는 클럭과 slave select(SS)가 있는데 클럭은 동기화 신호이며 SS는 master 기기가 여러 slave기기와 통신을 할 때 하나의 기기와 통신을 할 때에 사용한다. 데이터 시그널로는 master에서 slave로 가는 방향성 있는 데이터 선인 MOSI가 있으며 slave에서 master로 가는 데이터 선인 MISO가 있다. SPI는 full duplex의 성질과 길이를 조정할 수 있는 유연성, 단순한 하드웨어 인터페이스 처리 때문에 많이 사용된다.

Figure 3. Adafruit 1.44 Color TFT LCD Display

4. Joystick (조이스틱)

조이스틱은 입력 장치 중 하나로 게임기의 컨트롤러 부분을 담당한다. 조이스틱은 축을 사용하여 방향을 입력할 수 있고, 버튼을 눌러서 입력을 줄 수도 있다. 본 프로젝트에서는 스틱을 이용한 입력만을 다뤘다.

본 프로젝트에서 사용된 Funduino Joystick Shield는 이러한 입력 기능 외에 여러 interface 모듈을 지니고 있다. nRF24L01 모듈은 2.4Ghz 주파수로 무선 통신을 가능하게 해주는 RF 모듈이다. 또한 Nokia 5110 LCD Interface 모듈이 있어, joystick에 LCD 모듈을 바로 연결하여 사용할 수도 있다. 이외에도 bluetooth interface, I2C interface 등이 구현되어 있어 여러가지 용도로 사용이 가능하다.

조이스틱 내부에는 2개의 가변 저항이 있고, x축과 y축의 방향에 따라서 이 저항 값이 변하고 이에 따라 전압 값이 변하는 것이다. 그러므로 사용자가 스틱을 조종하면 이에 해당하는 전압 값이 ADC를 거쳐 0에서 1 사이의 값으로 입력된다. 이를 위해 x의 전압 값과 y의 전압 값을 각각 읽어 들여야 한다.

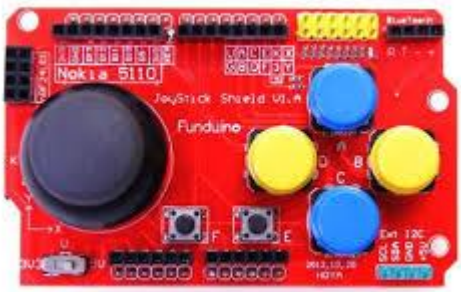

Figure 4. Nokia 5110 joystick

5. STM32F411RE nucleo board

본 프로젝트에 사용된 보드는 STM32F411RE nucleo 보드로 본 프로젝트의 중심 역할을 한다. Nucleo 보드는 ARM32-bit Cortex-M4 CPU with FPU 를 사용하며 100MHz의 max CPU frequency를 가지고, GPIO 핀들과 12-bit ADC, RTC, Timer, I2C, USART, SPI 등을 지원한다.

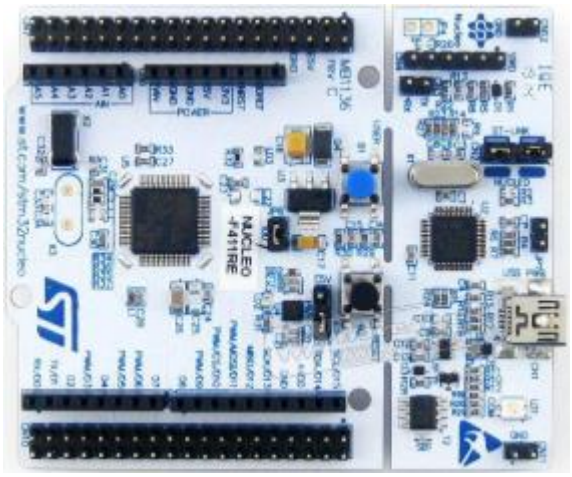

Figure 5. Nucleo-F4RE11

이 보드의 특징으로는 ST-Link/V2 가 보드에 포함되어 있어 pc와 usb를 통해 연결하기만 하면 프로그램을 쉽게 변경하고 지우고 다시 올릴 수 있다. 이를 통해 firmware upgrade도 손쉽게 할 수 있다. 또한 ARM에서 제공하는 mbed OS와 SDK library를 mbed 호환 보드에 올려서 사용할 수 있어서 어플리케이션 개발에 용이하다. 이 외에도 USB connector를 통해 power를 공급받으며, LED, push button, OSC clock, USART communication, Arduino connector 등등 다양한 모듈들이 존재한다. 아래 figure를 보면 이 뿐만 아니라 더 많은 모듈들이 있음을 알 수 있다.

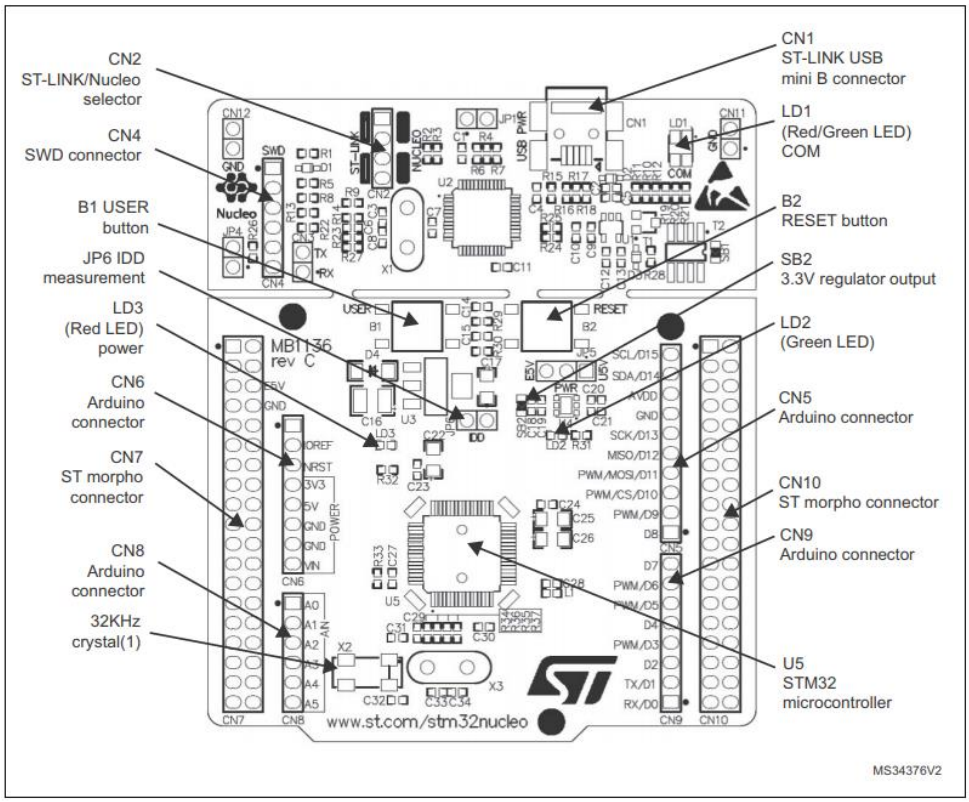

Figure 6. Nucleo top layout

B. 동작 개요

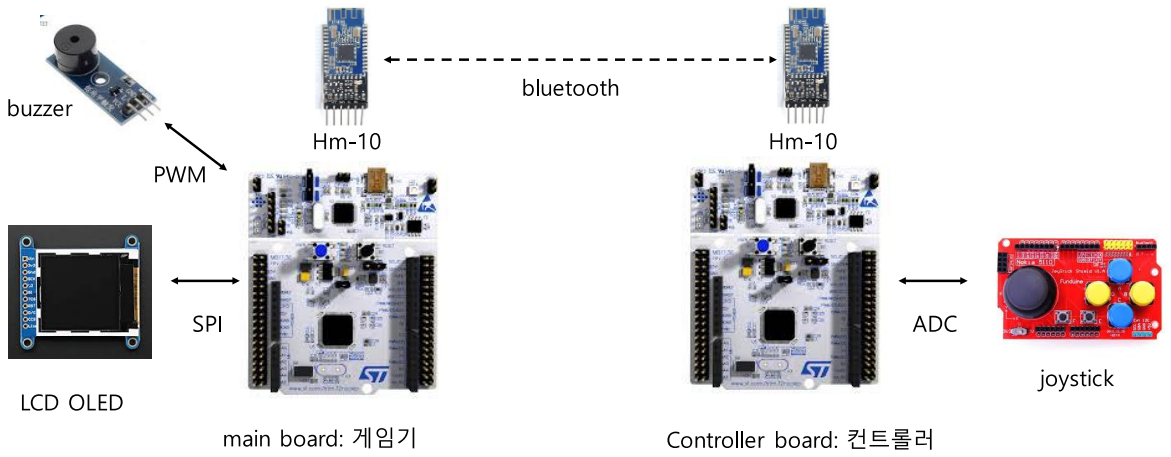

Figure 7. 프로젝트 구성도

위의 그림은 해당 프로젝트의 전체적인 구성도를 나타낸다. 우선 게임기는 크게 두 부분으로 나누어 볼 수 있는데, 게임기 역할을 하는 메인 보드와 컨트롤러 역할을 하는 컨트롤러 보드이다. 메인 보드에서는 게임 프로그램이 돌아가며 OLED에 게임 진행 상태를 출력한다. 컨트롤러 보드에서는 사용자로부터 입력 값을 joystick으로 받고 해당 정보를 메인 보드로 블루투스를 이용하여 넘겨준다. 메인 보드는 해당 정보를 받아 게임을 제어한다. 여기서 joystick과 컨트롤러 보드 사이는 ADC로 통신을 한다. 메인 보드 쪽에서는 OLED와는 SPI통신, Hm-10와는 UART통신, 그리고 buzzer와는 PWM통신을 한다.

III. Description of Operation

다음으로는 게임기 구현에 대해서 구체적으로 어떻게 게임과 각 모듈들이 동작하는지에 대해서 살펴볼 것이다. 게임기의 각 동작들에 대한 설명과 코드, 그리고 해당 코드에 대한 설명 순서대로 설명이 진행될 것이다.

A. Snake Game

게임기에서 동작될 게임으로 선택된 것은 snake game이다. Snake game은 뱀이 음식을 먹으며 꼬리를 키우고 벽이나 자기자신에게 닿지 않도록 하는 게임이다. 게임을 동작 시키기 위한 함수들은 다음과 같다.

Code

```
void initConsole();
void initSnake();
void removeSnakeBlock();
void addSnakeBlock();
int change_dir();
void createFood();
status moveSnake();
status checkGameOver();
```

코드의 중요 부분을 세가지로 좁혀 설명하자면, 첫 번째는 뱀이 움직이는 것, 두번째는 뱀이 방향을 바꾸는 것과 마지막으로 뱀이 벽 또는 자기 자신과 부딪혔는지 확인 하는 것이다. 뱀이 움직이는 것을 구현한 것은 `addSnakeBlock()`, `removeSnakeBlock()`, `moveSnake()` 이 세 함수이다. `change_dir()`는 뱀의 방향을 바꾸며 `checkGameOver()`함수는 뱀이 자신 또는 벽과 부딪혔는지 감지하고 부딪혔으면 게임을 끝내는 함수이다. 하지만 우선 각 함수들과 각 함수들의 동작을 설명하기 전에 프로그램에서 사용한 데이터 구조에 대한 설명을 하겠다.

Code

```
1. typedef enum {BLANK, BOUNDARY, SNAKE, FOOD} blocktype;
2. typedef enum {NONE, UP, DOWN, RIGHT, LEFT} direction;
3. typedef enum {GAMEON, GAMEOVER} status;
4.
5. typedef struct snake_blocks {
6.     int x, y;
7.     struct snake_blocks* prev;
8. } snake_blocks;
```

위 코드는 게임 구현을 위해 새롭게 정의된 데이터 타입들이다. 우선, 첫 줄에 정의된 blocktype은 게임 보드 한 블록에 들어가는 객체 종류를 가리킨다. 게임 보드에 들어가는 블록 종류로는 아무 것도 존재하지 않는 블록을 BLANK, 벽을 BOUNDARY, 뱀을 SNAKE, 그리고 뱀이 먹을 음식을 FOOD로 정의하였다.

두 번째 줄에 정의된 direction 데이터 타입은 뱀이 갖는 방향에 대한 것이다. 뱀은 총 동서남북의 네 가지 방향을 가질 수 있는데 NONE이 정의된 것은 게임이 시작되었을 때 뱀은 사용자가 입력을 주기 전까지 멈춰 있어야 하기 때문이다.

다음으로 정의된 데이터 타입은 게임의 진행 상태를 가리키는 status이다. 게임의 상태는 두가지로 게임이 진행되고 있다는 GAMEON과 게임이 끝난 상태인 GAMEOVER가 정의 되어있다.

마지막으로 뱀의 객체가 5번째 줄부터 8번째 줄까지 정의 되어있다. 뱀의 각 블록은 이 객체로 선언 되어있으며 연결 리스트(linked list)로 구현되어 있다. 해당 객체는 각 블록이 위치한 x, y 좌표와 연결 리스트이기에 앞에 연결된 블록 객체를 가리키는 포인터 변수가 정의 되어있다.

다음은 게임 초기화 상태, 즉 게임이 처음 켜질 때의 상태에 대한 설명이다. 밑의 코드는 선언된 변수에 대한 코드이다.

Code

```
1. blocktype
   board[BOARD_HEIGHT/BLOCK_SIZE][BOARD_WIDTH/BLOCK_SIZE];
2. direction curDir = NONE;
3. status gameStatus=GAMEON;
4. int snakeSize=3;
5. float speed=0.5;
6. snake_blocks* snakeHead, *snakeTail;
7. int isFood=0, eaten_stat=0;
8. snake_blocks *firstNewBlock=NULL, *lastNewBlock=NULL;
9. int score=0;
```

첫 줄은 게임이 진행될 보드에 대한 선언이다. 보드는 세로는 BOARD_HEIGHT/BLOCK_SIZE, 가로는 BOARD_WIDTH/BLOCK_SIZE로 선언되어 있는데 BOARD_HIEGT 와 BOARD_WIDTH는 각각 128로 정의 되어있으며 BLOCK_SIZE는 4로 정의 되어있다. 이는 우리가 사용한 OLED의 크기가 128x128 픽셀 값을 가지기 때문이다. BLOCK_SIZE는 각 블록의 가로 세로 크기를 임의로 정한 것인데 전체 보드의 가로와 세로 크기가 128이기 때문에 게임 보드가 너무 크거나 작지 않게 하기 위해서 4로 정의하였다. 그리하여 보드의 배열 크기는 32x32이다.

두 번째 줄에 선언되어 있는 curdir 변수는 뱀이 움직이고 있는 방향을 가리킨다. 게임이 초기화 되었을 때 뱀은 움직이지 않고 있어야 하기 때문에 NONE으로 초기화 해주었다. 세 번째 줄의 gameStatus는 게임오버가 되었는지 확인하는 변수이다. 네 번째 줄의 snakeSize는 뱀의 길이, 다섯 번째 줄의 speed는 뱀이 움직이는 속도이다. 여섯 번째 줄의 snakeHead와 snakeTail은 각 머리와 꼬리 snake_block 구조체를 가리키는데 이는 snake가 움직일 때 사용된다. 일곱 번째 줄의 isFood, eaten_stat은 각각 음식을 새로 생성시키기 위해 음식이 보드에 있는지 아니면 뱀이 먹었는지 확인하는 변수이며 eaten_stat는 뱀이 현재 음식을 먹었는지, 먹었다면 몇 개 먹었는지 확인하는 변수이다. 여덟 번째 줄의 firstNewBlock과 lastNewBlock은 뱀이 음식을 먹은 상태에서 몸이 길어지지 않았을 때 가장 먼저 먹은 음식의 위치와 가장 마지막에 먹은 음식의 위치를 포인터로 가리킨다. 그리고 마지막으로 선언되어 있는 score는 현재 먹은 음식의 총 개수를 토대로 사용자가 몇 점을 받았는지를 저장하는 변수이다.

게임시작에 관련된 함수는 `initConsole()`와 `initSnake()`두 개이다. 우선 `initConsole()`는 보드를 생성하며 보드의 가장자리에 벽을 두고 정 중앙에 사이즈가 3 인 뱀을 보드에 할당한다. 보드의 나머지 부분은 BLANK 로 둔다. `initSnake()`는 뱀의 연결 리스트를 생성 및 초기화 해주는 함수이다. 3 개로 구성된 각 뱀 블록은 각각 그 위치를 저장하고 서로를 연결한다.

다음으로는 게임의 전체적인 흐름을 메인 함수의 코드를 통해 알아볼 것이다. 다음은 메인 함수의 일 부분이다. 다른 모듈에 대한 코드부분을 대부분 제외하였다.

Code

```
1. int main(void){
2.     srand(time(NULL));
3.
4.     initSnake();
5.     initConsole();
6.     displayConsole();
7.     food_checker.attach(&createFood, 0.001);
8.
9.     pc1.printf("start\r\n");
10.    bluetooth.attach(&change_dir);
11.
12.    while(1){
13.      if(curDir==NONE) continue;
14.      gameStatus=moveSnake();
15.      if(gameStatus==GAMEOVER)
16.          break;
17.      displayConsole();
18.      wait(speed);
19.    }
20.    pc1.printf("end\r\n");
21.}
```

네 번째 줄에서부터 여섯 번째 줄은 뱀과 보드를 초기화하고 그것을 OLED 에 출력하는 것까지의 동작에 대한 코드이다. 일곱 번째 줄은 Ticker 을 사용하여 `createFood()`함수를 interrupt 를 통해 0.001 초 간격으로 발생시키는 것이다. `createFood()`는 음식이 현재 보드에 있는지 없는지 확인하고 없으면 생성시켜주는 함수이다. 처음 일곱 번째 줄이 불리면 보드에 음식이 없기 때문에 음식을 랜덤으로 생성해 준다. 게임에 필요한 객체들을 초기화 한 후에 컴퓨터에 start 를 출력하고 게임을 시작한다. 사용자의 입력을 받기 전 뱀은 움직이지 않기

때문에 멈춰 있고 사용자의 입력을 받고 서야 뱀이 움직이기 시작한다. 사용자 입력은 bluetooth 모듈에 값이 들어오면 interrupt 를 통해서 방향을 바꿔준다. 열여덟 번째 줄은 뱀의 속도만큼 기다렸다가 출력하는 것을 가리킨다. 즉, 해당 현재의 속도를 시간 간격으로 `moveSnake()`를 호출한다. 만약 게임오버가 되면 gameStatus 가 GAMEOVER 값으로 변하기 때문에 while 문을 나와 컴퓨터에 end 를 출력하고 프로그램이 끝이 난다.

다음으로는 게임의 세가지 중요 동작에 대해서 설명하겠다. 우선 가장 중요한 것은 뱀이 움직이는 로직이다. 뱀이 움직이는 것은 `moveSnake()` 함수를 중심으로 돌아간다. 해당 함수 코드는 다음과 같다.

Code

```
1. status moveSnake(){
2.     status stat;
3.     addSnakeBlock();
4.     stat=checkGameOver();
5.     if(board[snakeHead->y][snakeHead->x]==FOOD){
6.         snake_blocks *temp =
   (snake_blocks*)malloc(sizeof(snake_blocks));
7.         temp->x= snakeHead->x;
8.         temp->y= snakeHead->y;
9.         temp->prev=NULL;
10.        if(firstNewBlock==NULL){
11.          firstNewBlock=temp;
12.          lastNewBlock=firstNewBlock;
13.        } else {
14.          lastNewBlock->prev=temp;
15.          lastNewBlock=temp;
16.        }
17.        isFood=0;
18.        eaten_stat++;
19.        score+=10;
20.        if(score%30==0){
21.            speed/=2;
22.            pc1.printf("%.3f\r\n", speed);
23.        }
24.        createFood();
25.    }
26.    board[snakeTail->y][snakeTail->x]=BLANK;
27.    board[snakeHead->y][snakeHead->x]=SNAKE;
28.    if(eaten_stat>0&&firstNewBlock->x==snakeTail->prev-
   >x&&firstNewBlock->y==snakeTail->prev->y){
29.        snakeSize++;
30.        eaten_stat--;
```

```
31.         snake_blocks *temp= firstNewBlock;
32.         firstNewBlock=firstNewBlock->prev;
33.         free(temp);
34.     }else{
35.         removeSnakeBlock();
36.     }
37.     return stat;
38. }
```

해당 함수에서는 충돌을 감지하는 checkGameOver()를 호출하여 매 움직임 마다, 즉 moveSnake()가 호출될 때마다 게임오버가 났는 지의 상태를 반환해 준다.

moveSnake()는 크게 두가지 부분으로 나뉘며 그것은 뱀이 아무 것도 먹지 않은 상태로 움직일 때와 음식을 먹었을 때의 움직임이다. 뱀이 움직이는 기본적 로직은 뱀이 향하는 방향 위치에 새로운 snake_block 객체를 생성해주고 해당 블록을 snakeHead 로 설정해주며 뱀의 꼬리 블록에 할당된 메모리를 해체하고 그 앞에 연결된 블록을 snakeTail 로 설정해주는 것이다. 이 과정은 뱀의 머리가 새로 생성되고 꼬리를 없앰으로 뱀으로 하여금 움직이는 것처럼 보이게 한다. 뱀의 머리가 향하는 위치에 새로운 블록을 생성해주는 함수가 addSnakeBlock()이며, 꼬리 부분의 블록을 해체하는 함수는 removeSnakeBlock()이다.

만약 뱀이 음식을 섭취하고 뱀의 꼬리가 먹은 음식의 위치를 지날 때 뱀은 길어져야 한다. 먼저 뱀이 처음 음식을 먹었을 때 먹은 음식에 대한 연결 리스트를 생성해준다. 여기서 먹은 음식을 연결 리스트로 연결한 것은 뱀이 음식을 먹고 몸이 늘어나기 전에 또 다른 음식을 섭취했을 때 음식을 먹은 두 위치를 지날 때 꼬리가 늘어나야 하기 때문이다. 그렇기 때문에 연결 리스트를 사용하여 첫번째로 먹은 음식의 위치를 지나고 몸이 늘어나면 포인터가 다음으로 늘어날 블록의 위치를 가리켜 다음 블록을 지날 때 몸이 늘어나게 한다. 음식이 뱀의 머리를 닿을 때마다 음식의 연결 리스트는 늘어나고 음식 연결 리스트의 head 를 snakeTail 이 지날 때마다 음식 연결 리스트는 줄어들고 뱀이 길어진다. 뱀이 길어지는 원리는 뱀이 움직일 때 꼬리부분의 블록을 해체하지 않게 함으로 뱀이 길어지는 것처럼 보이게 한다. 또한 뱀이 음식을 섭취하면 점수를 10 점 올리고 점수가 30 의 배수가 될 때마다 뱀의 속도를 두배로 빠르게 하여 난이도를 더 어렵게 하며 그 속도를 컴퓨터에 출력한다.

뱀이 방향을 제어하는 함수는 change_dir()이다. 뱀의 방향을 바꾸는 코드는 joystick 에서 방향을 정하는 것과 거의 유사하다. 뱀의 방향이 위인 경우 아래로 내려갈 수 없으며 오른쪽인 경우 왼쪽으로 갈 수 없다. 이 역 또한 마찬가지 임으로 그러한 입력들은 무시하며 그 외의 입력에 대해서 방향을 바꾸어 준다.

마지막으로 게임의 status 를 확인하는 코드를 살펴보겠다.

Code

```
1. status checkGameOver(){
2.     if(board[snakeHead->y][snakeHead->x]==SNAKE||\
3. board[snakeHead->y][snakeHead->x]==BOUNDARY){
4.         return GAMEOVER;
5.     }
6.     return GAMEON;
7. }
```

위의 코드는 뱀의 머리가 뱀에 닿아 있는지 또는 벽에 닿아 있는지 확인하며 만약 그렇다면 status 를 GAMEOVER 로 반환해준다. 만약 함수가 호출되었는데 뱀의 머리가 BLANK 에 닿으면 게임은 계속 진행되어야 함으로 GAMEON 을 반환해 준다. Board 와 snake 연결 리스트는 한번에 변하지 않으며 먼저 snake 연결리스트가 변하고 해당 함수를 호출하여 뱀이 벽 또는 스스로와 충돌했는지 확인한후 snake 위치를 보드에 업데이트 한다.

여기까지 게임의 구동 원리였으며 다음으로는 각 모듈들에 대한 코드와 동작 원리에 대해서 설명하겠다.

B. OLED Adafruit 1.44 Color TFT LCD Display

OLED Adafruit 1.44 Color TFT LCD Display 는 TFT driver(ST7735R)을 지원함으로 ST7735R 라이브러리를 찾아서 사용하였다. 해당 라이브러리를 사용하기 위해서 Adafruit_GFX 라이브러리 또한 import 하였다. ST7735R 라이브러리에서 지원하는 함수 중 프로젝트에 사용한 것은 `fillScreen(uint16_t color)` 와 `fillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color)` 이다. 첫 번째 함수는 화면을 색으로 채우는 것이며 두 번째 사용한 함수는 어느 위치에 가로와 세로, 그리고 색갈을 지정한 직사각형을 그리는 함수이다. 게임의 배경을 첫 번째 함수를 사용하여 검정색으로 칠했고, snake 게임의 snake, food, boundary 모두 두번째 함수를 사용하여 직사각형으로 구현하였다.

OLED 객체 선언은 다음과 같이 해주었다.

Code

```
Adafruit_ST7735 tft(D11,NC, D13,D10, D5, D4);
```

파라미터는 각각 mosi, miso, sck, cs, rs, reset 의 핀을 가리킨다.

OLED 를 구현한 코드는 메인 함수의 일부분과 디스플레이를 함수이다. 메인함수에서는 `initR()`를 호출하여 화면을 초기화 해주며 `fillScreen()`으로 배경화면을 검정색으로 만들었다. 게임 초기 상태를 설정해주고 `displayConsole()`함수를 불러 OLED 화면을 출력하였다. `displayConsole()`코드는 다음과 같다.

Code

```
1. void displayConsole(){
2.     for(int i=0; i<BOARD_HEIGHT/BLOCK_SIZE; i++){
3.         for(int j=0;j<BOARD_WIDTH/BLOCK_SIZE; j++){
4.             if(board[i][j]==BOUNDARY){
5.                 tft.fillRect(i*4, j*4, 3, 3, 0xFFFF);
6.             }else if(board[i][j]==SNAKE){
7.                 tft.fillRect(i*4, j*4, 3, 3, 0x07E0);
8.             }else if(board[i][j]==FOOD){
9.                 tft.fillRect(i*4, j*4, 3, 3, 0x001F);
10.             }else if(board[i][j]==BLANK){
11.                 tft.fillRect(i*4, j*4, 3, 3, 0x0000);
```

```
            }
        }
    }
}
```

위의 코드는 32x32 크기의 보드를 모두 훑으며 각각 블록에 해당하는 색깔의 직사각형을 그린다. 벽의 경우 흰색 블록으로, 뱀의경우 초록색 블록으로, 음식의 경우 빨간색 블록으로 그리고 비어있는 경우 검정색 블록으로 정사각형을 그린다. 각 정사각형은 3x3 인데 이는 블록과 블록사이 빈틈을 주기 위해서 4x4 가 아닌 3x3 으로 설정하였다.

C. Joystick

Joystick은 입력 컨트롤러이기 때문에 뱀의 움직임 방향을 조절한다. 그러므로 아래 코드에 나와있듯이 enum 타입으로 direction을 설정하였다.

joystick으로부터 출력되는 0부터 1 사이의 값을 읽어 들이기 위해 AnalogIn class를 사용하였다. 여기서 x축, y축이 독립적이므로 각각 나눠서 읽어 들였다.

Main 함수를 보면 먼저 게임기와 입력 컨트롤러 간의 bluetooth 연결을 수행한다. 그리고 연결이 되면 while 문 안에 들어와서 계속 direction을 읽는다. 이때, 방향이 바뀌고, bluetooth가 writeable하면 direction을 update한다.

Code

```
typedef enum direction{NONE, UP, DOWN, RIGHT, LEFT}
direction;

AnalogIn A_x(A0);
AnalogIn A_y(A1);

direction curDir=NONE;
direction prevDir=NONE;

void readDir();

int main(void){

 ...        //bluetooth connection
```

```
14.
15.while(1){
16.    readDir();
17.      if(prevDir!=curDir&&bluetooth.writeable()){
18.        prevDir=curDir;
19.        bluetooth.printf("h%d", curDir);
20.        pc.printf("current direction: %d\r\n", curDir);
21.      }
22.  }
23.}
```

다음 코드는 위에 나온 readDir 함수이다. joystick으로부터 읽어 들인 값인 Ax와 Ay는 0부터 1사이의 값을 갖는다. Joystick의 구동 방향에 따라 전체를 원으로 보면 위, 아래, 왼쪽, 오른쪽으로 4개로 4등분할 수 있다. 이렇게 나눈 이유는 사용자가 joystick으로 언제나 정확하게 위, 아래, 왼쪽, 오른쪽 방향을 줄 수는 없을 것이라고 생각했기 때문에 최대한 사용자의 의도에 맞추기 위해 이와 같이 범위를 나누어서 설정하였다.

Up 방향의 입력이 들어온 경우를 보면 스틱이 조금 왼쪽으로 가거나 조금 오른쪽으로 가더라도 뱀을 위로 가게 하려는 것이 사용자의 의도이므로 이에 맞춰 범위를 설정해 두었다. 이때 주의해야 할 점은 뱀의 꼬리가 바로 머리가 되면 안 되기 때문에 아래로 가고 있을 때 UP 신호가 오더라도 이는 무시해야 한다는 거다. 그래서 현재 direction이 down이 아닐 때에만 direction이 up으로 업데이트되게끔 하였다.

이는 왼쪽, 오른쪽 방향 변환에도 똑같이 적용된다.

Code

```
1. void readDir(){
2.   float Ax, Ay;
3.   Ax = A_x.read();
4.   Ay = A_y.read();
5.
6.   //up
7.   if((0.24f < Ax && Ax < 0.76f) && (0.74f < Ay && Ay <
   1.01f)){
8.       if(curDir!=DOWN)
9.           curDir=UP;
10.}
```

```
11.
12.  //down
13.  else if((0.24f < Ax && Ax < 0.76f) && (-0.01f < Ay &&
   Ay < 0.26f)){
14.      if(curDir!=UP)
15.          curDir=DOWN;
16.}
17.
18.  //left
19.  else if((-0.01f < Ax && Ax < 0.26f) && (0.24f < Ay &&
   Ay < 0.76f)){
20.      if(curDir!=RIGHT)
21.          curDir=LEFT;
22.}
23.
24.  //right
25.  else if((0.74f < Ax && Ax < 1.01f) && (0.24f < Ay &&
   Ay < 0.76f)){
26.      if(curDir!=LEFT)
27.          curDir=RIGHT;
28.  }
29.}
```

D. Buzzer

구현한 게임기에서 buzzer는 게임 시작할 때, 게임 종료할 때, 그리고 뱀이 움직일 때 소리가 나게끔 하였다. 이를 위해 enum type으로 GAMEON과 GAMEOVER의 status를 정의하였다.

buzzer는 pwm 신호로 제어되므로 PwmOut class를 사용해 정의하였다. 그 다음에 나오는 freqGameStart, freqGameOver, freqMove는 음계별 주파수를 담은 array이다. 생성하고자 하는 음에 따라 설정해주었고, 음계별 주파수에 대한 표는 코드 아래에 첨부하였다. 그 다음 나오는 beat는 박자를 설정해주는 array이다. 이는 아래 함수 설명에서 다루겠다.

먼저 게임을 시작하면서 buzzer에 1을 write한다. 이는 duty cycle을 100%로 설정하여 소리가 안나는 상태가 된다. 그리고 gameStartBGM()을 통해 게임 시작 음악을 출력하고, while문안에서 게임이 종료되었을 때 gameOverBGM이 나온다. 그리고 changeDir 함수 안에서 방향이 업데이트 될 때 moveBGM이 출력된다.

Code

```
typedef enum status{GAMEON, GAMEOVER}status;

PwmOut buzzer(D3);

float freqGameStart[]={698.46, 880.00, 783.99, 698.46,
1396.91, 698.46, 698.46, 880.00, 932.33, 1046.50,
1174.66, 1046.50, 698.46, 698.46};
float
freqGameOver[]={261.63,246.94,233.08,220.00,207.65,196.0
0};
float freqMove[]={261.63, 196.00};
int beatGameStart[]={6,4,4,6,8,12,6,4,4,4,4,4,3,3};
int beatGameOver[]={8,4,8,4,16,4};
int beatMove[]={2, 4};

status gameStatus=GAMEON;
int main(void){
    buzzer=1.0;
      ...         //bluetooth connection

    initSnake();
    initConsole();
    displayConsole();
    gameStartBGM();

      ...

    while(1){
      if(curDir==NONE) continue;
      gameStatus=moveSnake();
      if(gameStatus==GAMEOVER)
          break;
      displayConsole();
      wait(speed);
    }
    gameOverBGM();
    pc1.printf("end\r\n");
}

void change_dir(){
  char ch;
  if(bluetooth.getc()=='h'){
    ch = bluetooth.getc();
    if(ch=='4'){
        if(curDir!=RIGHT){
            curDir=LEFT;
```

```
43.                 moveBGM();
44.             } }   …   } }
```

아래 코드는 게임 시작할 때 나오는 소리를 출력하는 함수이다.

출력할 음계의 개수만큼 for문이 수행된다. 초기 상태에서 1로 설정해둔 duty cycle을 0.9로 바꿔줌으로써 duty cycle을 90%로 설정한다. 이 이유는 duty cycle이 0.9일 때가 가장 원하는 음의 소리가 나왔기 때문이다.

그리고 각각의 음계별 주파수를 통해 us 단위의 주기를 구해서 이를 buzzer의 주기로 설정한다. 이로써 원하는 음이 buzzer를 통해 나오게 된다. 또한, 위에서 설정한 beatGameStart는 박자를 의미한다. 16MHz의 CPU clock을 가진 8bit timer에서는 16Mhz/2^8 =62.5Khz가 play rate이다. 여기에 원하는 박자 값을 곱해서 wait를 주면 음악의 박자 효과를 낼 수 있다. 이는 gameOverBGM, moveBGM 등 모든 소리를 출력하는 함수에 동일하게 적용된다.

Code

```
1. void gameOverBGM(){
2.     int period_us;
3.     int beat_ms;
4.
5.     for(int i=0;i<gameOverBGMNum; i++){
6.         buzzer=0.9;
7.         period_us=1000000/freqGameOver[i];
8.         beat_ms=62.5*beatGameOver[i];
9.         buzzer.period_us(period_us);
10.         wait_ms(beat_ms);
11.     }
12.     buzzer=1.0;
13.}
14.
15.void gameOverBGM(){
16.     int period_us;
17.     int beat_ms;
18.
19.     for(int i=0;i<gameOverBGMNum; i++){
20.         buzzer=0.9;
21.         period_us=1000000/freqGameOver[i];
22.         beat_ms=62.5*beatGameOver[i];
23.         buzzer.period_us(period_us);
24.         wait_ms(beat_ms);
```

```
25.     }
26.     buzzer=1.0;
27.}
28.void moveBGM(){
29.     int period_us;
30.     int beat_ms;
31.
32.     for(int i=0;i<moveBGMNum; i++){
33.         buzzer=0.9;
34.         period_us=1000000/freqMove[i];
35.         beat_ms=62.5*beatMove[i];
36.         buzzer.period_us(period_us);
37.         wait_ms(beat_ms);
38.     }
39.     buzzer=1.0;
40.}
41.
```

표 1. 음계별 주파수

옥타브 / 음계	1	2	3	4	5	6	7	8
C(도)	32.70	65.41	130.81	261.63	523.25	1046.50	2093.00	4186.01
C#	34.65	69.30	138.59	277.18	554.37	1108.73	2217.46	4434.92
D(레)	36.71	73.42	146.83	293.66	587.33	1174.66	2349.32	4698.64
D#	38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	4978.03
E(미)	41.20	82.41	164.81	329.63	659.26	1318.51	2637.02	5274.04
F(파)	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	5587.65
F#	46.25	92.50	185.00	369.99	739.99	1479.98	2959.96	5919.91
G(솔)	49.00	98.00	196.00	392.00	783.99	1567.98	3135.96	6271.93
G#	51.91	103.83	207.65	415.30	830.61	1661.22	3322.44	6644.88
A(라)	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00	7040.00
A#	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	7458.62
B(시)	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	7902.13

E. HM-10

블루투스의 경우 HM-10 에서 HM-10 으로 통신해야 하기 때문에 AT command 를 사용하였다. AT command 를 모듈로 전달해주는 방법으로는 UART 를 사용하였다. 블루투스의 master-slave 관계에 있어서 master 을 컨트롤러로 두고 slave 를 메인 보드로 두었다. 각 master 와 slave 에서 사용한 AT command 는 다음과 같다.

표 2. Master 에서 사용한 AT command 와 각 command 에 대한 설명

AT+RENEW	공장 초기화 상태로 모듈 값 복구
AT+IMME1	모듈이 초기화 되었을 때 AT command 에만 반응
AT+ROLE1	모듈을 master 로 설정
AT+DISC?	주위의 장치 스캔
AT+CON[Address]	주소를 통하여 장치 연결 시도

표 3. Slave 에서 사용한 AT command 와 각 command 에 대한 설명

AT+RENEW	공장 초기화 상태로 모듈 값 복구
AT+ROLE0	모듈을 slave 로 설정

Master 의 AT command 중 AT+DISC?는 주위에 있는 장치의 주소를 얻기 위해서만 사용되었기 때문에 코드에서는 사용하지 않았다. AT+DISC?로 메인 보드의 주소를 알아 낸 결과, 0CB2B778AFB7 이었고 AT+CON 뒤에 이 주소를 써 줌으로 연결을 하였다. 위의 과정을 거치게 되어도 연결이 이루어진 것을 확인해야 하기 때문에 REQ & ACK 과정을 거쳐서 연결을 확인하였다. 자세한 것은 코드를 통하여 설명하겠다.

우선 Controller 쪽 블루투스 코드는 다음과 같다.

Code

```
1. Serial bluetooth(D8, D2, 9600);
2.
3. int main(void){
4.   while(!bluetooth.writeable());
5.   pc.printf("Bluetooth starting...\r\n");
6.
7.   bluetooth.printf("AT+RENEW"); wait(0.5);
8.   bluetooth.printf("AT+IMME1"); wait(0.5);
9.   bluetooth.printf("AT+ROLE1"); wait(0.5);
10.   while(bluetooth.readable())
    pc.putc(bluetooth.getc());
11.   wait(2);
12.   bluetooth.printf("AT+CON0CB2B778AFB7");
13.   pc.printf("Connecting...\r\n");
14.   wait(2);
15.
16.   bluetooth.putc('&');
17.   pc.printf("Waiting for ACK\r\n");
18.   while(bluetooth.getc()!='&'){bluetooth.putc('&');}
19.   pc.printf("ACK received\r\n");
20.
21.   while(1){
22.       readDir();
23.       if(prevDir!=curDir&&bluetooth.writeable()){
24.         prevDir=curDir;
25.         bluetooth.printf("h%d", curDir);
26.         pc.printf("current direction: %d\r\n", curDir);
27.       }
28.   }
29.}
```

우선 블루투스 객체를 하나 생성한다. 블루투스 통신을 할 때 baud rate 를 9600 으로 설정한다. Main 의 가장 처음에 해당 블루투스에 command 를 보낼 수 있는지 writeable()을 통해 확인한다. 사용할 수 있는 상태가 되었을 때, 위에서 언급한 AT+RENEW, AT+IMME1, 그리고 AT+ROLE1 의 AT command 들을 블루투스로 보낸다. 그 다음 열 번째 줄에 블루투스에 모든 버퍼를 비운다. AT+CON 명령어를 사용하여 slave 와 연결을 하면 연결이 완성된다. 하지만 연결이 안 되었을 경우를 위해서 REQ 메시지로 '&'를 보내고 '&' ACK 메세지가

올 때까지 REQ 메시지를 전송하며 기다려 주며 ACK 가 오면 사용자 입력을 받고 메인 보드 쪽으로 데이터를 보낸다.

메인 보드의 블루투스 코드는 다음과 같다.

Code

```
1. Serial bluetooth(D8, D2, 9600);
2.
3. int main(void){
4.    ...
5.     while(!bluetooth.writeable());
6.     pc1.printf("Bluetooth starting...\r\n");
7.     bluetooth.printf("AT+RENEW\r\n");
8.     wait(1);
9.     bluetooth.printf("AT+ROLE0\r\n");
10.     while(bluetooth.readable())
11.         pc1.putc(bluetooth.getc());
12.     while(bluetooth.getc()!='&'){}
13.     bluetooth.putc('&');
14....
15.     bluetooth.attach(&change_dir);
16....
17.}
```

slave 쪽에서도 bluetooth 객체를 master 와 같은 방식으로 선언해준다. 메인 함수에서는 AT command 를 사용하여 모듈을 초기화하며 역할을 slave 로 정해 master 인 controller 가 커넥션을 이룰 때까지 기다린다. Master 가 연결을 하면 '&'라는 메시지가 올 것이고 그렇게 되면 slave '&'를 ACK 메시지로 보내준다. 그러면 연결이 완성되고 bluetooth 에 데이터가 들어오면 interrupt 메커니즘을 통하여 방향을 바꾸어 준다.

IV. Problems and Solutions

게임 프로그램을 코딩하면서 또, 여러 모듈들을 사용하여 게임기를 구현하면서 여러가지 문제들이 있었다. 그 중 해결한 문제가 있고 해결하지 못한 문제도 있었다. 또한 게임기를 보완시킬 수 있는 방법 또한 생각해 보았다.

A. 해결한 문제

1. 프로그램 버그

처음 프로그램을 작성했을 때, 뱀이 음식을 먹었을 때 그 위치에서 늘어나야 하는데 이 정보를 어떻게 저장할 지 고민하다가 변수 두개를 선언하여 x, y 좌표를 저장하였다. 하지만 하나의 변수로 위치를 저장하면 뱀이 음식을 먹고 길이가 늘어나기 전에 또 음식을 먹게 되면 해당 변수 x, y 좌표가 두 번째 먹은 음식에 좌표만을 가리키기 때문에 첫 번째 먹은 음식에 대해서 점수는 올라가지만 길이는 늘어나지 않게 된다. 뱀의 길이가 짧을 때는 이러한 경우가 잘 없지만 뱀의 길이가 길 때에는 이러한 현상이 자주 나타날 것으로 예측되기에 먹은만큼 먹이의 위치를 기록하기 위해서 linked list 로 위치 정보를 저장하였다. 길이가 늘어나기 전 먹이를 얼마나 먹었는지 모르기 때문에 연결 리스트가 적당하다고 판단하였다. 또한 pointer 를 처음 먹은 먹이 위치로 가리키고 있고 그 구조체는 다음 먹은 먹이를 가리키고 있다. 뱀은 먹이를 먹은 순서대로 그 위치에서 늘어나기 때문에 연결리스트를 사용하는 것이 적절하다. 해당 구조체를 바꿈으로 버그를 해결하였다.

2. 블루투스 통신 자동 연결 문제

블루투스를 AT command 를 사용해서 자동으로 연결하려고 했지만 주변에 있는 슬레이브를 탐색하는 명령어, AT+INQ 가 실행되지 않아 다른 직접 슬레이브의 주소를 AT+DISC 를 사용하여 알아내고 주소를 통해 연결하도록 설정하였다.

B. 해결하지 못한 문제

1. 입력이 짧은 시간 많이 들어갔을 때의 버그

뱀은 자신의 speed 만큼 기다렸다가 움직인다. 하지만 뱀이 기다리는 순간 많은 입력을 넣어주면 게임이 비정상적으로 종료되는 버그가 있다. 해당 버그를 해결하지 못하였지만 의심되는 원인으로는 board 와 snake 위치가 atomic 하지 않게 움직이기 때문이다. 그렇기 때문에 입력이 한번에 여러 개 들어오면 뱀의 위치는 변했는데 게임 보드가 갱신되지 않은 상태에서 또 다른 입력을 받아 잘못 인식하여 게임을 비정상적으로 종료한다고 추측하였다.

C. 보완할 점

1. microSD 활용

프로젝트에서 사용한 Adafruit OLED 모듈에 microSD 리더기 또한 포함되어있어 microSD 카드를 활용하면 좋겠다는 아이디어가 나왔다. Sd 카드로 파일을 읽고 쓸 수 있다. 프로젝트에서 사용한 OLED 의 경우 bitmap 파일을 출력할 수 있는 라이브러리 함수가 제공되는데 sd 카드에 bitmap 파일을 저장한 상태에서 게임이 시작하고 끝날 때 bitmap 파일을 불러와서 이미지로 출력을 하여 게임을 꾸밀 수 있다. 또는 역대 최고기록을 SD 카드에 저장하여 게임이 꺼지더라도 최고 기록을 파일로 저장하고 게임이 끝날 때 마다 자신의 점수화 함께 최고기록을 출력해 줄 수 있다.

2. joystick 버튼을 이용하여 기능 추가

현재는 funduino joystick shield 의 joystick 부분만을 사용하고 있지만 버튼 부분을 활용하여 다양한 기능을 추가할 수 있다. 부저를 비활성화 시켜 음향을 출력하지 않게 하는 버튼, 게임을 일시 정지 하는 버튼, 게임을 초기화 시키는 버튼이 그 예이다.

V. Conclusion

NUCLEO-F411RE 보드를 사용하여 주변 기기들과 통신함으로써 게임기를 구현해보았다. 사용한 주변 기기들로는 게임 화면을 출력해주는 OLED, 게임 입력 컨트롤러 역할을 하는 joystick, 게임의 동작 상황에 따라 소리를 출력하는 buzzer, 사용자로부터 입력 값을 joystick으로 받아서 이를 Bluetooth 통신을 통해 main 게임기로 넘겨주는 hm-10을 사용하였다. 각 주변 기기마다 사용하는 통신 프로토콜과 동작 방식이 달라서 이에 대한 이해가 필요했다.

전체적인 구성은 게임기를 담당하는 main board와 controller를 담당하는 controller board로 구성된다. 그리고 이 main board와 OLED가 SPI로 통신하며 게임 화면을 띄우고, pwm 방식으로 buzzer를 통해 게임 상황에 따른 소리를 출력한다. 이와 동시에 컨트롤러 보드에서는 joystick으로부터 사용자가 입력한 값을 입력 받아서 이를 Hm-10을 통해 Bluetooth 방식으로 main board로 정보를 전달하였다.

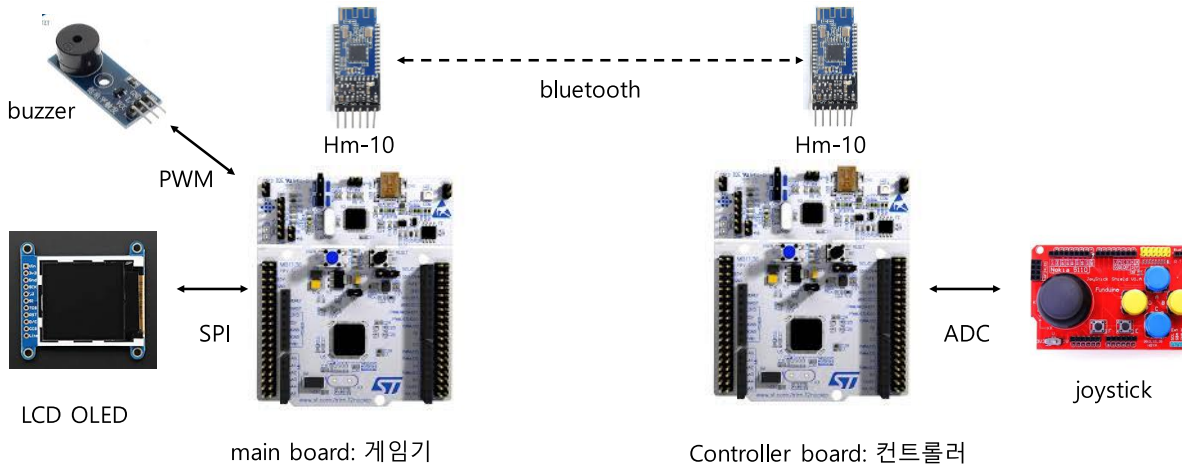

Figure 8. 전체적인 구조

다음은 완성된 게임기의 모습이다.

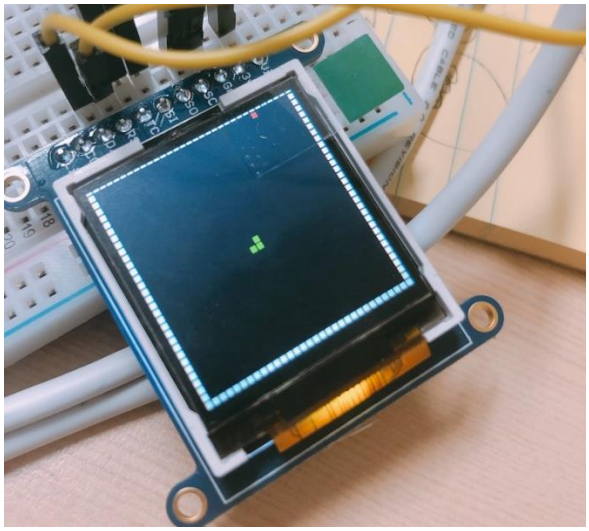

Figure 9. OLED 화면

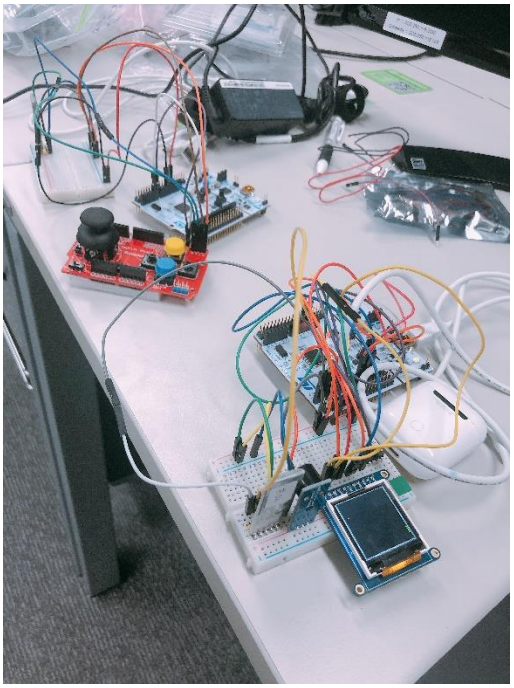

Figure 10. 메인보드와 컨트롤러