목차

1. 프로젝트 제목

RHYTHM STAR for Arduino

https://youtu.be/ZMurwirTWfw?t=51

2. 주제 선정 배경

1. 프로젝트의 사용 가능 부품

센서류: 온도 센서, 조도 센서, IR 리모트 수신 센서, 조이스틱 모듈, 초음파 센서

디스플레이류: 16x2 문자형 LCD, 8x8 도트 매트릭스 모듈, 4열 7세그먼트 LED, 1열 7세그먼트 LED, LED

기타류: 스위치, 가변저항, 부저 등

2. 대주제 선정 논의

사용할 수 있는 부품이 제약되고, MCU로 ATmega328 한 개를 사용할 수 있는 상황에서 어떤 것을 만드는 것이 창의적이고, 완성도가 있으며, 기술적 난이도가 있는 프로젝트가 될 수 있는지 고민했다.

1. 기술적 난이도 측면

   본 과제에서 기술적 난이도라는 것은, 회로 설계, 복합 구현, MCU 제어와 소프트웨어 구현 등의 많은 것을 함축하는 표현입니다. 사용할 수 있는 부품이 제한된 상황에서 회로 설계로 높은 기술적 난이도를 보여주는 프로젝트를 달성하기는 어렵고,

복합 구현(많은 부품을 목적에 맞게 제어) 역시, 아두이노에서 사용 가능한 핀의 수가 제한되기 때문에, 일정 규모 이상의 복합 구현을 달성하기 어렵다고 생각했다. (대체로 많은 핀을 사용하는 디스플레이류 부품 하나와, 센서류, 기타류 부품 몇 개만 사용하면 핀을 모두 사용하게 됨) 따라서 MCU 제어와 소프트웨어 구현 측면에서 높은 기술적 난이도를 가지는 프로젝트를 목표로 진행하게 되었고, 소프트웨어 구현이 상당히 중요한 게임과 관련된 프로젝트를 구현하기로 했다.

2. 창의성 측면

   창의성이란 새로운 생각이나 개념을 찾아내거나 기존에 있던 생각이나 개념들을 새롭게 조합해 내는 것을 말합니다. 따라서 본 과제에서 말하는 창의적인 프로젝트란, 흔해 빠지지 않은, 또는 기성의 것들을 새로이 잘 조합하여 좋은 만듦새를 가진 작품을 만드는 것으로 이해했다. 키트 구성품으로 사용 부품이 제한되는 만큼, 어떤 실사용 용도를 가지는 측정 기기(온도, 빛, 거리), 변환 기기(예를 들어 입력을 모스 부호로 변환하여 출력한다던지)와 비슷한 카테고리의 프로젝트를 만들면 기존 과제와 비슷한 느낌을 주거나, 작위적이고, 상용화된 물건의 열화판으로 생각되는 프로젝트가 될 수 있을 것 같았다. 따라서 같은 부품을 사용하더라도, 조작과 프로그래밍 논리, 디스플레이 방법에 따라 다양한 창의성을 보여주기 쉬운 게임과 관련된 프로젝트를 구현하기로 했다.

3. 완성도 측면

   완성도라는 것은 어떤 일이나 작품 따위가 질적으로 완성된 정도를 나타내는 것으로, 우선 만들어야 하는 것이 정해지고 나서 생각할 수 있는 것이지만, 주제를 정하기 전에 완성도 측면에서 고려할 수 있는 것은 기술적 난이도 측면에서 논의한 것과 비슷하다고 생각됩니다. 왜냐하면 낮은 기술적 난이도를 가지는 프로젝트에서 가질 수 있는 완성도의 한계는 높은 기술적 난이도를 가지는 프로젝트에서 가질 수 있는 완성도 한계보다 낮기 때문입니다. 따라서 주제 선정 과정에서, 이 부분에서 고려한 것은 기술적 난이도 측면에서 논의된 것과 같다.

4. 최종 대주제 설정

   (1), (2), (3)에서 논의 된 것을 바탕으로 **게임과 관련된 완성도 있는 프로젝트**를 진행하는 것을 목표로 대주제를 정했다.

3. 소주제 설정 논의

위에서 논의한 대주제(게임과 관련된 완성도 있는 프로젝트)를 바탕으로 3가지 프로젝트 소주제를 생각하고, 이 중 한 개의 소주제를 정했다.

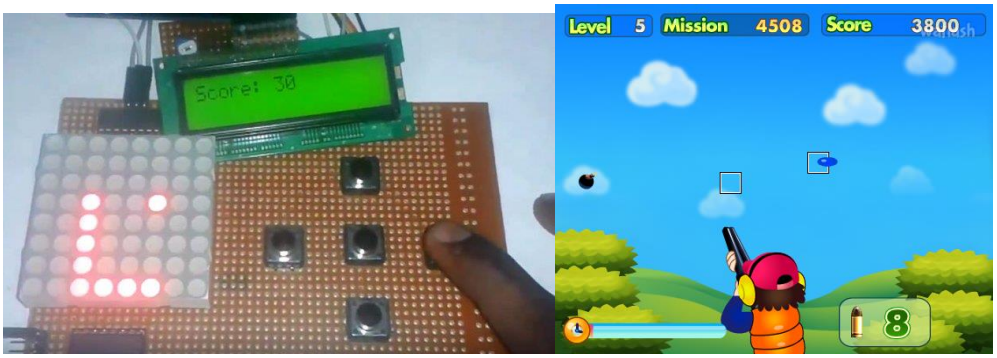
1. 게임 컨트롤러 (1안)

자이로 센서, 조이스틱, 스위치, 부저와 디스플레이류 장치 1개를 장착한 게임 컨트롤러

JSON(ArduinoJson) 또는 전용 포맷을 이용한 PC와의 씨리얼 통신으로, 프로젝트 기기는 컨트롤러의 역할만 담당하고 PC측 프로그램이 직렬 통신 신호를 마우스/키보드 제어로 변환해주어, PC 게임을 프로젝트 기기로 제어할 수 있음. 또한 전용 게임 소프트웨어를 이용하여 게임간 장착한 부저, 디스플레이류 장치에 피드백을 주는 것이 가능.

➔ 교수님께 드린 진행 관련 질문에서, 공정성 문제로 직렬 통신 신호를 시리얼 모니터가 아닌 외부 프로그램에서 파싱 하는 것(아직 배우지 않아서)이 허용되지 않는다는 답변을 들어서 폐기

2. 자이로 센서를 이용한 게임기 (2안)



8x8 Dot Matrix(MAX7219를 이용하여 5핀 사용으로 개선), 16x2 Character LCD, 부저, 자이로 센서, 기타류 장치를 이용한 자이로 센서 제어를 주로 한 미니게임 모음 게임기

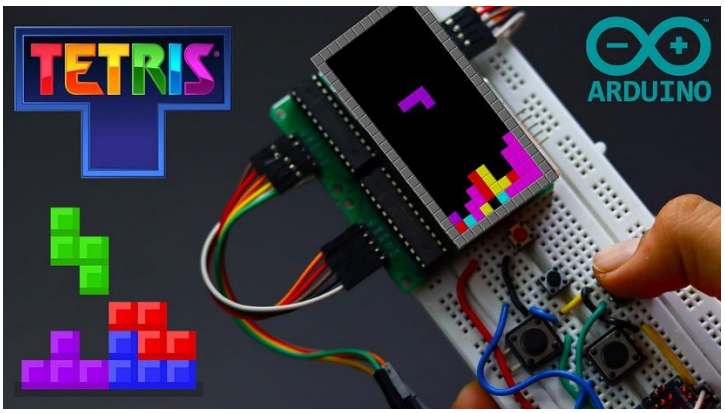
*. 예시 이미지와 같은 Snake 게임 등을 자이로 센서로 제어하거나, 원반 사격과 같은 게임을 8x8 도트 매트릭스와 자이로 센서, 스위치를 이용하여 구현

➔ 자이로 센서 납땜과 일부 테스트 회로 구성을 진행하였으나

- **사용 가능 부품**
  - LED, Switch, Buzzer, Thermistor, Photoresistor, Ultrasonic sensor, Joystick, 1-/4-Row 7-Segment, IR sensor + 리모콘, 8x8 Dot matrix, 16x2 LCD.
    - 이외 부품 사용금지,

교수님께 드린 질문에 대한 답변과 11/29에 공지된 사용 가능 부품 제한으로, MAX7219(키트 내 포함되었으나, 기존 과제에서 미사용된 부품)를 이용하여 8x8 도트 매트릭스의 핀 사용을 줄일 수 없게 되었고, 자이로 센서 역시 사용 불가하게 되어 폐기

3. 자작 게임기 (3안, 최종 결정)



8x8 도트 매트릭스 또는 16x2 캐릭터 LCD, 부저, 기타류 장치를 이용한 게임기

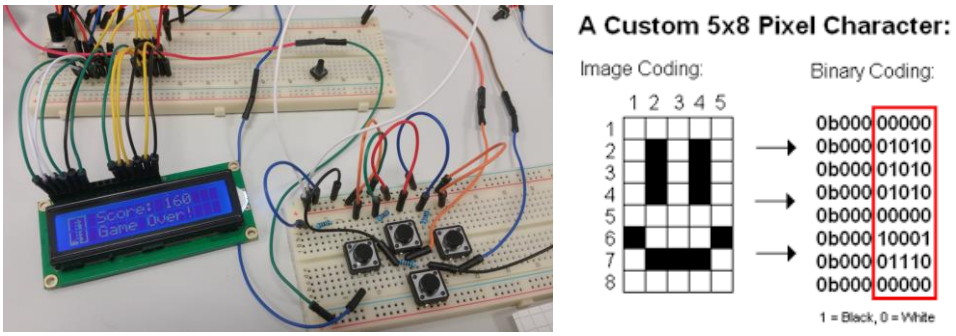
8x8 도트 매트릭스 또는 16x2 캐릭터 LCD에서 구현된 사례가 없는 완성도 있는 오리지널 게임을 해당 환경에서 구현

➔ 프로젝트 진행간 문제되는 부품이 없고, 대주제에 부합하는 구현안이기도 해서 해당 소주제로 프로젝트 진행을 결정하였다.

4. 소주제 심화

1. 디스플레이 장치 선택 (16x2 캐릭터 LCD, 8x8 도트 매트릭스)

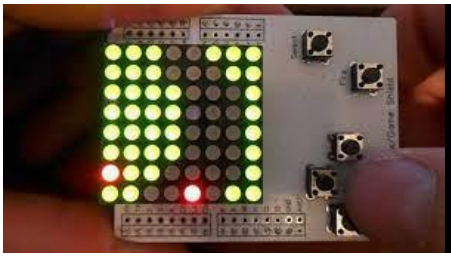
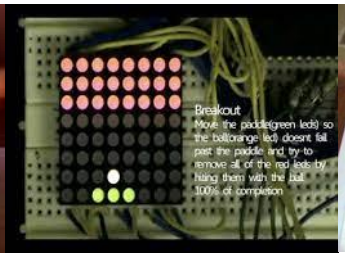
아두이노의 핀 수 제한으로, 16x2 캐릭터 LCD와 8x8 도트 매트릭스 모듈 둘다 이용하는 것은 제한이 있어, 둘 중 하나를 선택하여야 했는데, 16x2 캐릭터 LCD의 경우 텍스트 어드벤처 게임(옛날 PC 통신 시절 유행했던, MUD(Multi-user Dungeon) 게임과 같은)을 구현하는데 유리하나, 터미널(80x24) 급의 문자를 동시 처리하여, 아스키 아트나, 긴 게임 로그 등을 출력 가능했던, MUD 게임과 달리 16x2 캐릭터 LCD의 경우 다소 제약이 많기도 하고 현재와 같이 사용 가능한 부품이 제한된 환경에서는 아케이드 게임류를 구현하는 것이 낫다고 생각했다.

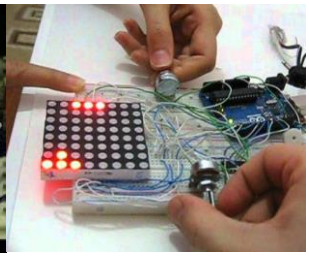
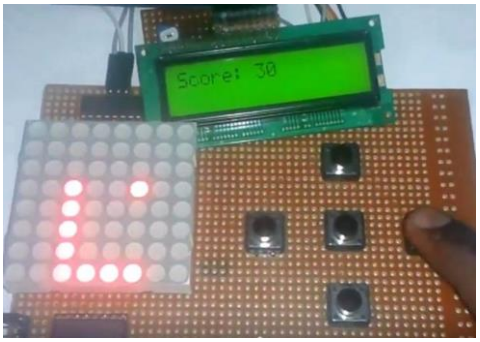


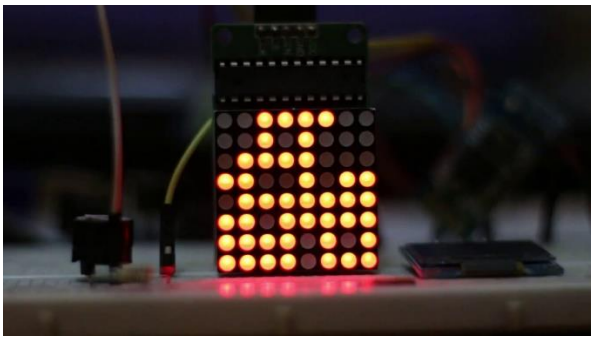
(출처: Hackaday, Tetris on Arduino + LCD)

16x2 LCD 역시 글자 한 칸을 5x8 크기의 점 배열을 통해 제어할 수 있지만, 크기가 너무작아 가시성이 좋지 않고, 구조적으로 각 글자 디스플레이칸마다 간격이 있어, 해당 부분의 점을 제어할 수 없는 한계 때문에 **LED의 크기가 커서 가시성이 좋은 8x8 도트 매트릭스로 디스플레이 부품을 선택하였다.**

2. 게임 종류 설정



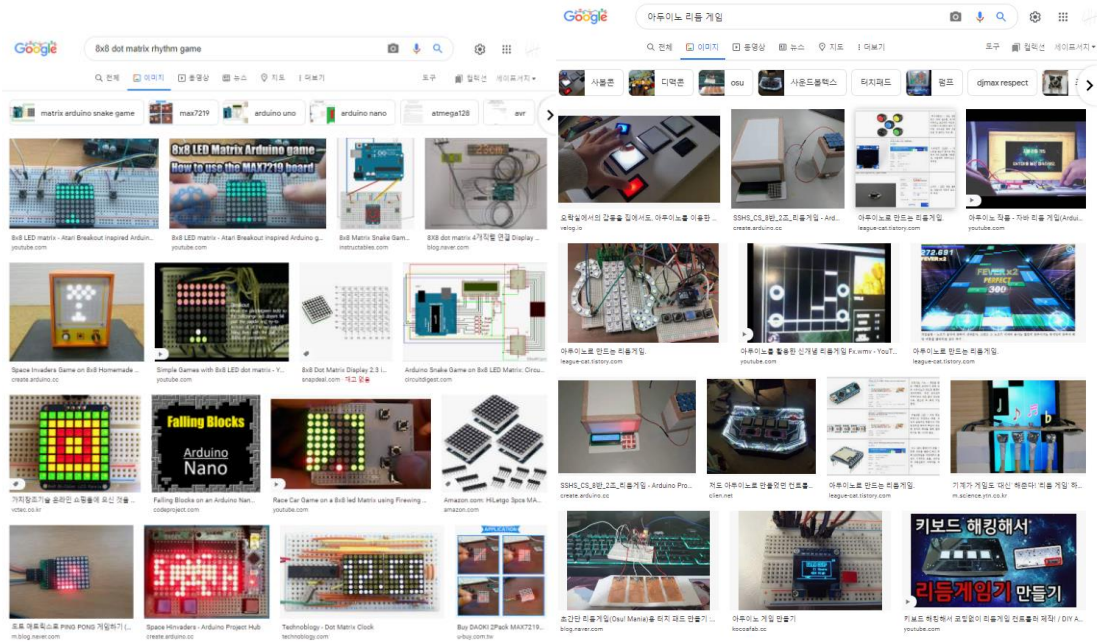
(8x8 도트 매트릭스 상에서 구현된 다양한 게임들, 좌측부터 자동차 경주, 벽돌 깨기, 핑퐁, 스네이크, 테트리스)

8x8 도트 매트릭스 환경에서 구현된 많은 게임들이 있지만, 이 중 8x8 도트 매트릭스로 구현된 사례가 없는 아케이드 게임류를 만들어보고 싶었다.

그러던 중, 어릴 적 피쳐폰을 사용하던 시절에 재밌게 했던 "리듬 스타"라는 건반 노트 기반 리듬 게임이 생각났다.

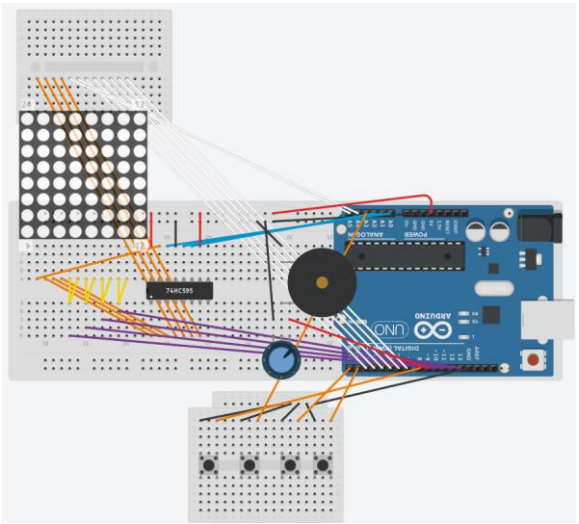
("리듬 스타"의 게임 화면)

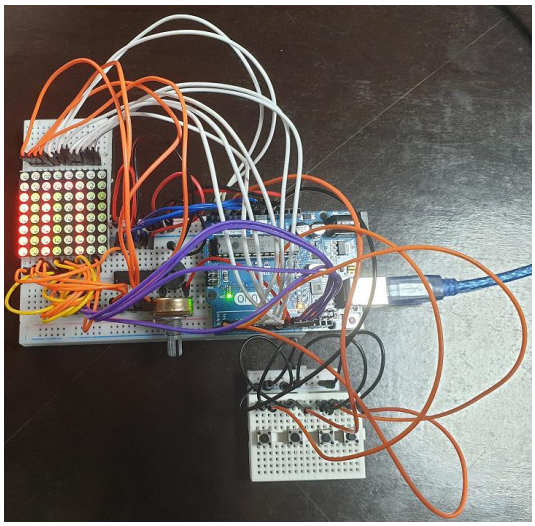
WIPI 기반으로 작성되어 2008년 출시한 **"리듬 스타"**는 피쳐폰에 내장된 MIDI API를 이용하여 곡을 재생하고 곡마다 설정된 노트 건반 기보를 출력하여 내려오는 타이밍에 맞게 건반에 해당하는 핸드폰의 버튼을 누르면 점수를 얻는 방식의 리듬 게임이었다.



8x8 도트 매트릭스로 건반 리듬 게임을 구현한 사례는 검색으로 찾아보기 힘들었고, 이 리듬 게임을 도트 매트릭스 구역을 8x2씩 나누어 4 건반 리듬 게임으로 아두이노에 구현하면 어떨까 하는 생각이 들어, 해당 게임을 아두이노에 구현하는 것을 주제로 정하게 되었다.

3. 시스템 구성도





핵심 사용 부품

아두이노 우노 R3(Arduino Uno R3) x1

8x8 도트 매트릭스 모듈(8x8 Dot Matrix) x1

시프트 레지스터(Shift Register IC) x1

수동 부저(Passive Buzzer) x1

택트 스위치(Tact Switch) x4

기타 사용 부품

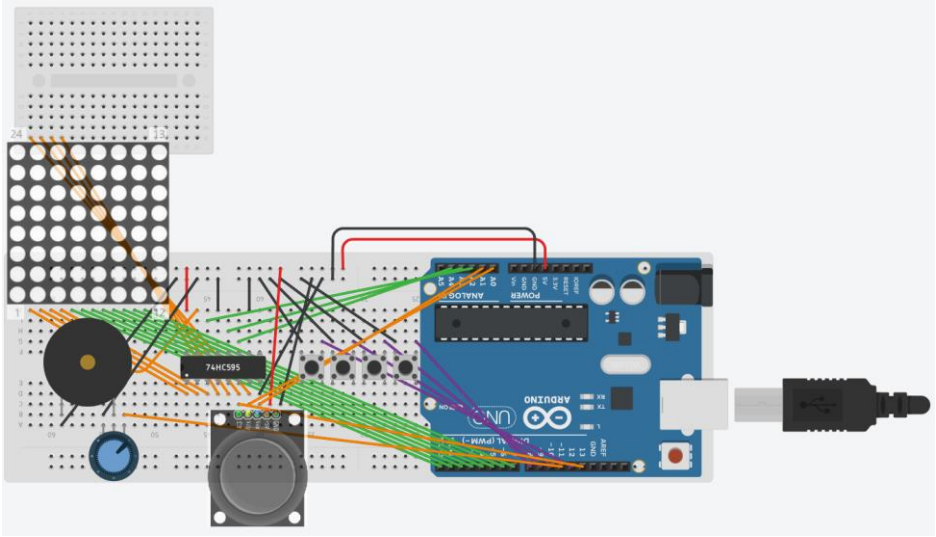
830 포인트 브레드보드(830 Point Breadboard) x1 **(그라인더로 잘라서 2분할 사용)**

10K 가변 저항(10K Variable Resistance) x1

점퍼 케이블(Jumper Cable)

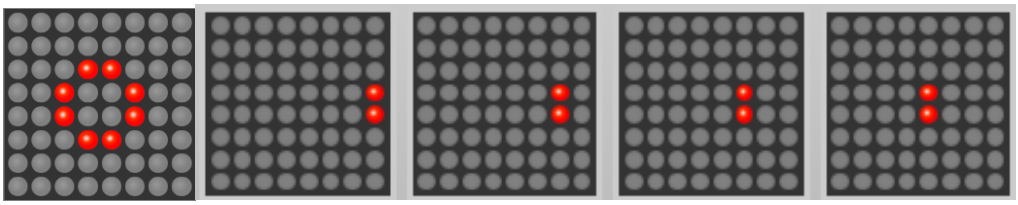
4. 변경 내용

제안서 회로도



(프로젝트 제안서의 시스템 구성도)

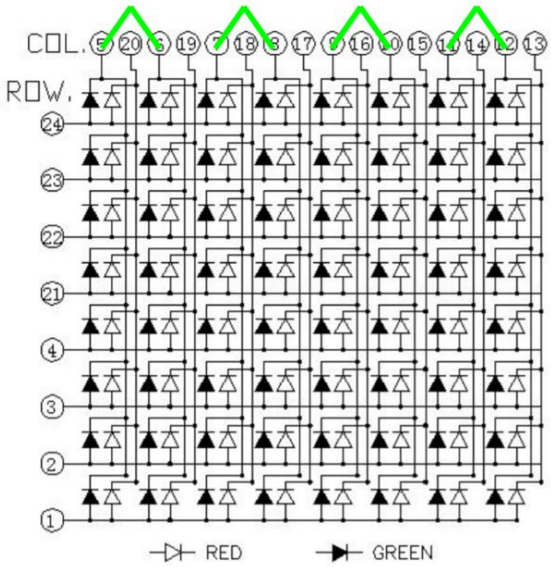
조이스틱 모듈 제거



(조이스틱을 통한 리듬게임 조작 시, 중앙에서 누름 피드백을 표시하고 상하좌우에서 노트가 내려오게 됨)

택트 스위치 조작으로도 UI 조작이 충분하고, 조이스틱을 이용한 리듬게임 조작 시 수직으로 반이 떨어지는 인터페이스보다 건반이 떨어지는 이펙트를 자세하게 표현하기 힘들 것 같아 조이스틱 모듈을 구성에서 제거했다.

도트 매트릭스 회로 변경



제안서 회로도의 단일 색상 제어 도트 매트릭스 회로에서(핀이 부족하여 단일 색상 제어로 구성함), 초록색도 사용이 가능하게끔 회로를 변경하였다. 초록색 LED를 2개씩 묶어 제어해서 핀 수를 아끼면서도 두가지 색깔을 사용할 수 있게 되었다.

브레드 보드 분할

택트 스위치 부분을 중앙 회로쪽에서 분리하여 편리한 조작이 가능하도록 브레드 보드를 그라인더로 잘라서 분할하여 사용하였다. (기존 부품을 이용하는 규칙을 지키기 위해)

5. 기능 설명 및 구현 결과

메인 화면

"Rhythm Star!" 글귀를 Left Scroll 하면서 출력 (아무 버튼이나 누를 시, 곡 선택 화면 진

입)

곡 선택 화면

곡 이름을 Left Scroll 하면서 출력 (첫번째 버튼으로 이전 곡, 네번째 버튼으로 다음 곡 선택 이동, 두번째 버튼으로 곡 플레이, 세번째 버튼으로 곡 최고 점수 보기)

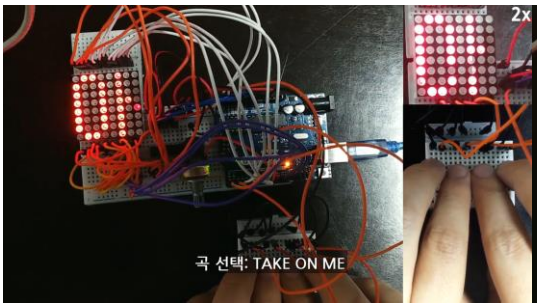
첫번째 버튼: 이전 곡으로 이동

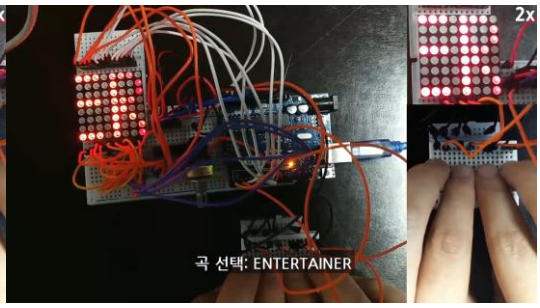
두번째 버튼: 게임 화면으로 이동 (곡 플레이)

세번째 버튼: 점수 확인 화면으로 이동 (곡 최고점수 보기)

네번째 버튼: 다음 곡으로 이동

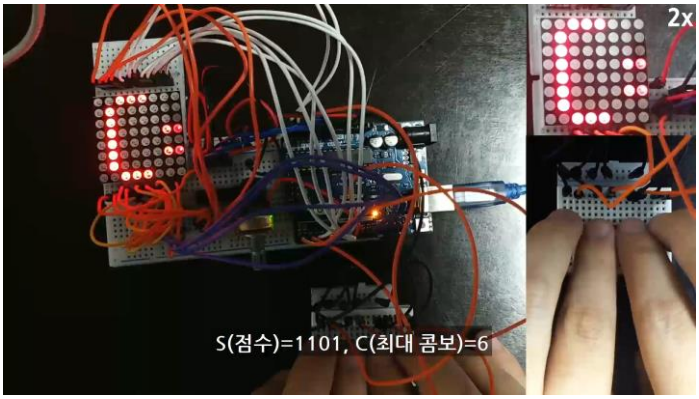
*. CUSTOM을 선택하였을 경우 EEPROM에 업로드한 곡이 재생됨 (후술)





점수 화면

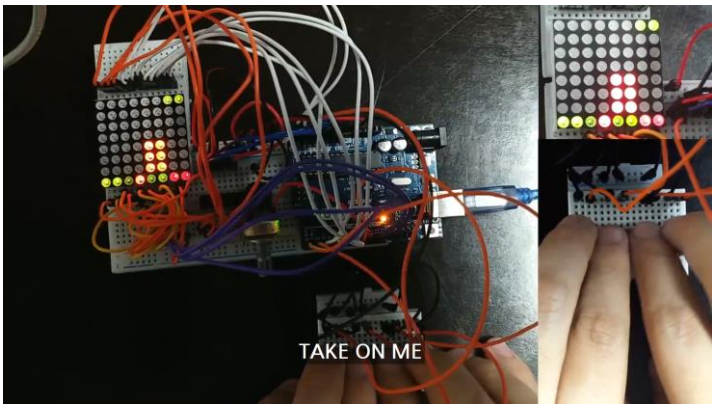
기존 달성한 최고 점수를 Left Scroll (아무 키나 누를 시, 곡 선택 화면 진입)



게임 화면

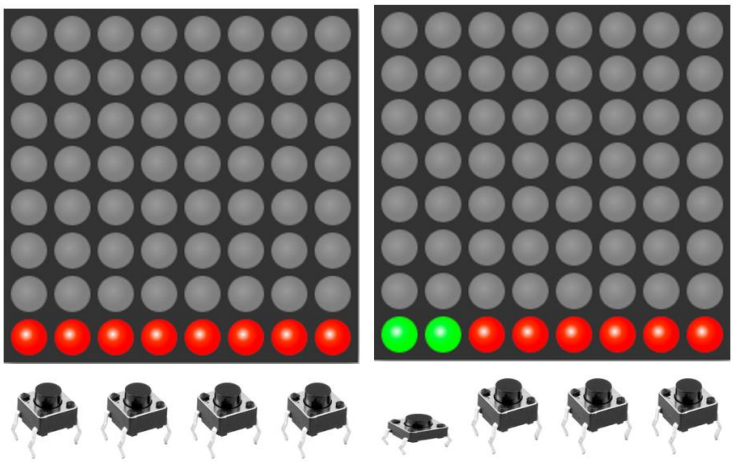
3, 2, 1, GO 화면 표시 이후 곡 재생과 함께 게임이 시작됨, 게임이 종료되면 현재 게임 점수를 출력하고 곡 선택 화면으로 이동됨 (버튼들을 4분할 건반으로 사용하고, 떨어지는 노트에 맞춰 버튼을 누르면 점수를 획득함)

*. 게임 점수는 각 곡마다, EEPROM R/W을 통해 관리.



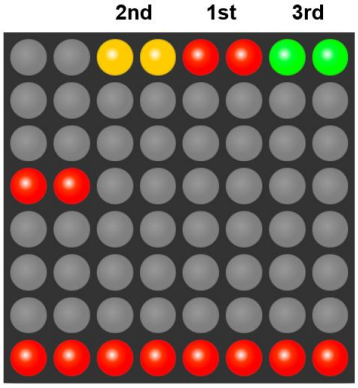
인게임 로직과 렌더링 상세

누름 피드백



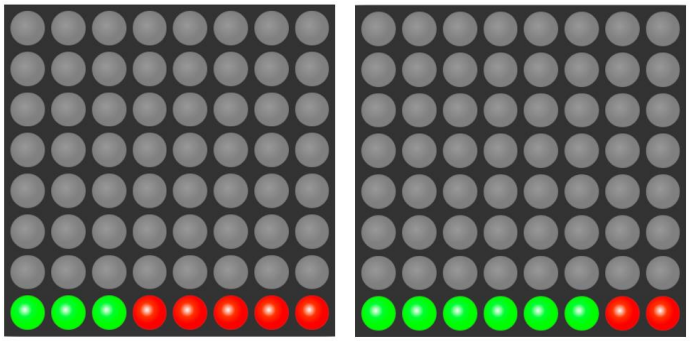
(버튼을 누르면 녹색 LED로 해당 건반에 하이라이팅 피드백을 표시)

다음 노트 표시



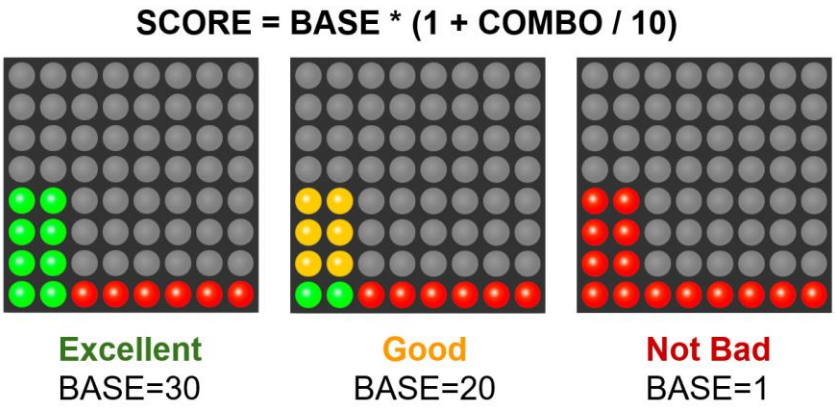
노트가 떨어지기 전에, 각 색깔별로 빨간색은 바로 다음 노트, 주황색은 그 다음 노트, 초록색은 그 다음 다음 노트로 상단에 표시해서 다음 차례의 노트가 떨어지는 것을 알 수 있게 구성했다.

콤보



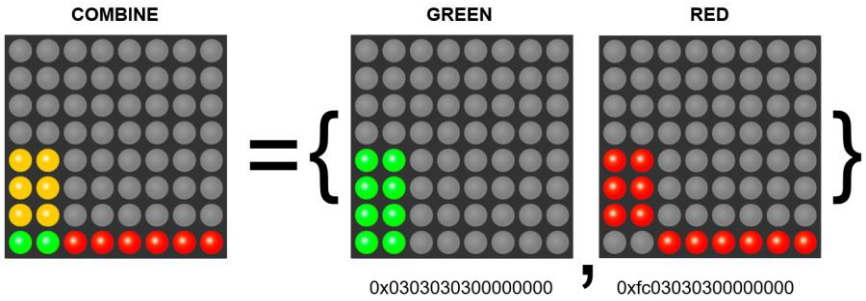
연속적으로 실수하지 않고 버튼을 눌렀을 때 콤보가 쌓이게 처리하고, 하단 바가 빨간색에서 초록색으로 콤보의 숫자만큼 채워지게 시각적 피드백을 표시하였다.

판정

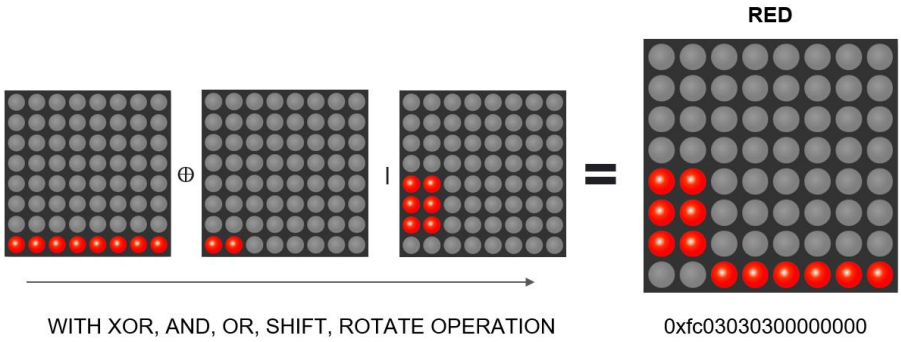


떨어질 때 가장 정확한 타이밍에 눌렀을 경우, 좀 부정확하게 누른 경우, 아주 부정확하게 누른 경우를 나누어서 서로 다른 색의 이펙트로 표시하고 이렇게 각자 다른 점수가 누적되게 처리했다.

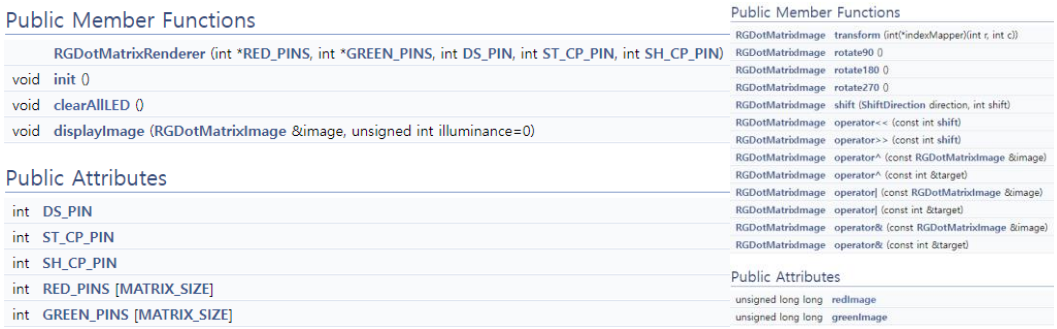
렌더링



(좌측에 표시된 상태를 표시하기 위해, 필요한 각각의 LED 색깔별 점등상태)



(64비트 부호없는 정수로 표현한 도트 매트릭스 이미지를 다양한 연산을 통해 처리)



Public Member Functions

	RGDotMatrixRenderer (int *RED_PINS, int *GREEN_PINS, int DS_PIN, int ST_CP_PIN, int SH_CP_PIN)
void	init ()
void	clearAllLED ()
void	displayImage (RGDotMatrixImage &image, unsigned int illuminance=0)

Public Attributes

int	DS_PIN
int	ST_CP_PIN
int	SH_CP_PIN
int	RED_PINS [MATRIX_SIZE]
int	GREEN_PINS [MATRIX_SIZE]

Public Member Functions

RGDotMatrixImage	transform (int(*indexMapper)(int r, int c))
RGDotMatrixImage	rotate90 ()
RGDotMatrixImage	rotate180 ()
RGDotMatrixImage	rotate270 ()
RGDotMatrixImage	shift (ShiftDirection direction, int shift)
RGDotMatrixImage	operator<< (const int shift)
RGDotMatrixImage	operator>> (const int shift)
RGDotMatrixImage	operator^ (const RGDotMatrixImage &image)
RGDotMatrixImage	operator^ (const int &target)
RGDotMatrixImage	operator| (const RGDotMatrixImage &image)
RGDotMatrixImage	operator| (const int &target)
RGDotMatrixImage	operator& (const RGDotMatrixImage &image)
RGDotMatrixImage	operator& (const int &target)

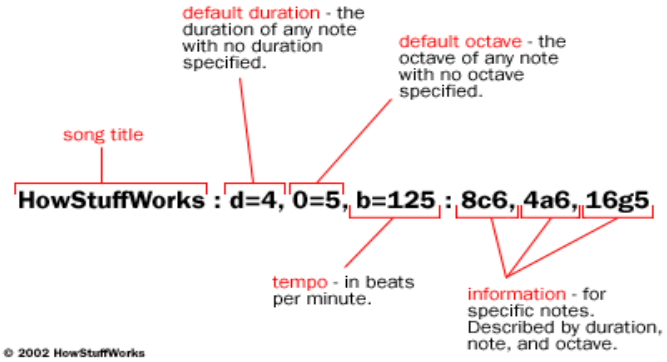
Public Attributes

unsigned long long	redImage
unsigned long long	greenImage

(RGDotMatrixRenderer, RGDotMatrixImage 클래스)

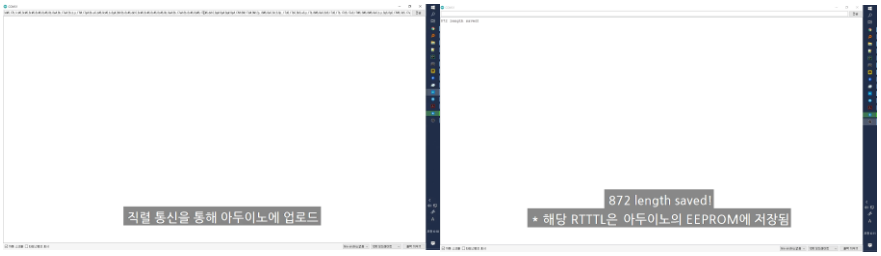
도트 매트릭스 이미지를 다양한 연산을 편리하게 할 수 있도록 객체지향적으로 모델링하고, 구조적으로 처리하기 위해 도트 매트릭스 렌더러를 직접 만들어 활용하였음

RTTTL(Ring Tone Text Transfer Language) 지원



노키아에서 벨소리를 텍스트 단문으로 처리하기 위해서 만든 포맷인 RTTTL(간단한 멜로디를 나타내기 위해서 범용적으로 사용되는 포맷)의 파싱을 통해 멜로디를 연주하고, 리듬 게임 건반을 자동 생성하여 각 곡마다 리듬 게임 노트를 따로 관리할 필요 없고, 곡 변경, 이식 부분에서 확장성과 범용성을 확보하였다.

커스텀 곡 업로드 지원

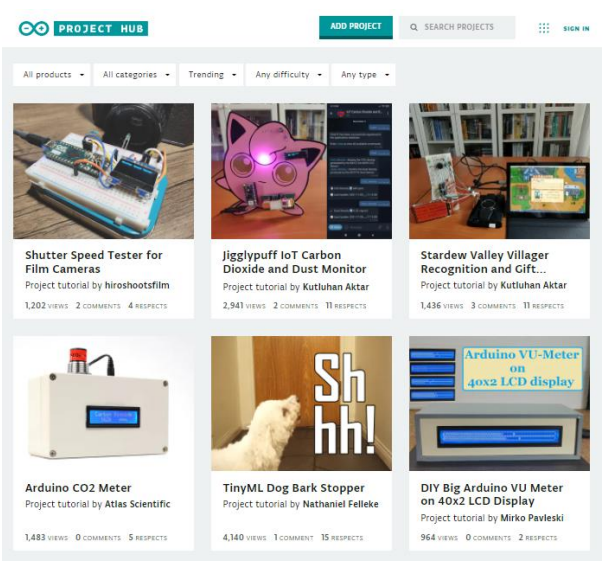


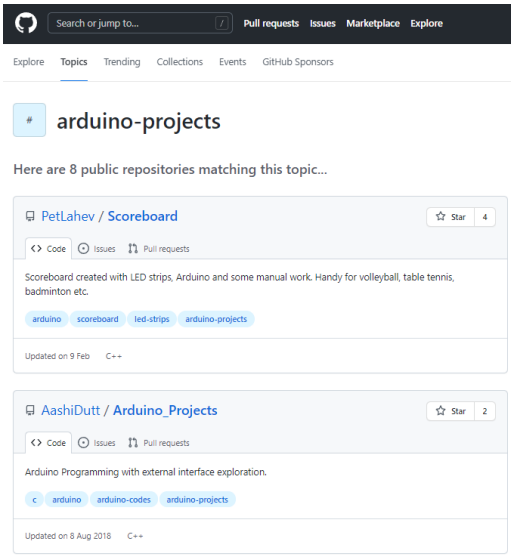
메인 화면에 위치한 상태에서 RTTTL을 시리얼 통신을 이용하여 보내면, EEPROM에 저장하여 CUSTOM 곡 선택으로 플레이 할 수 있도록 곡 업로드 기능을 지원하였다.

6. 결론

기대효과

8x8 도트 매트릭스와 아두이노 환경에서 작동하는 건반 리듬 게임을 구현

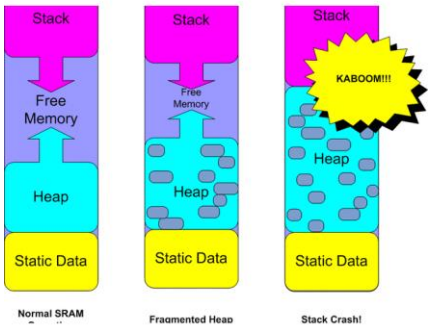




본 프로젝트는 완성 이후 소스 코드와, 회로도 등을 정리한 뒤 아두이노 프로젝트 허브와 GitHub 공개 저장소에 게시될 예정으로, 8x8 도트 매트릭스를 이용한 고전 리듬 게임기가 가지고 싶은 사람에게 실제 작동하는 구현체 스펙을 제시할 수 있고, 비슷한 기능을 구현하고 싶은 사람에게 레퍼런스 자료가 되거나 또는 다른 마이크로프로세서 프로젝트를 진행하는 사람들에게 영감이 될 수 있을 것으로 생각된다.

소감

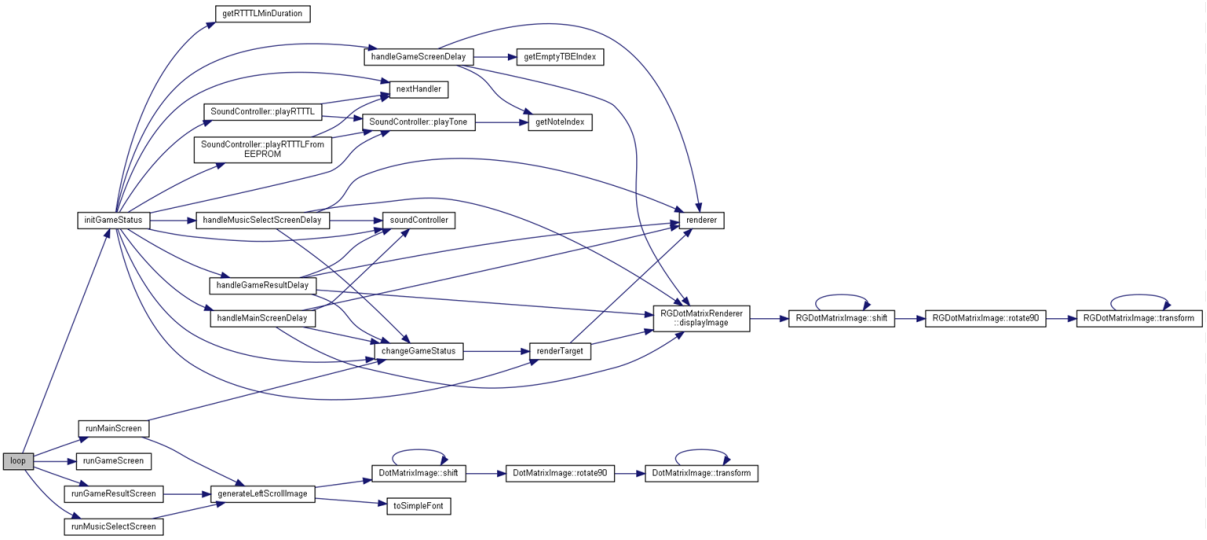
한 학기 동안 진행한 과제로 쌓은 역량을 총 동원해서 저만의 프로젝트는 만들어 보니, 그 과정속에서 얻어가는 것도 많았다.



메뉴 화면에서 Left Scroll간 도트 매트릭스를 최대한 끊김없이 스무스하게 제어하려고 미리 재생할 상을 동적 할당으로 저장하여 제어했다가, 메모리 부족으로 어려움을 겪어 실시간 상 생성 렌더링 방식으로 코드를 다시 고치기도 했고, EEPROM에 저장된 RTTTL(큰

문자열)을 메모리상에 올리고 함수를 호출하여 재생하려고 했다가 스택 메모리가 부족함을 깨닫고 하나씩 읽어서 처리하게 수정하기도 했고, 조금 더 나은 개발 환경을 위해서 PlatformIO도 공부하여 제게 익숙한 IDE에서 작업하는 방법도 이번 프로젝트를 수행하면서 알게 되었다. 이런 과정 속에서 제 자신의 마이크로프로세서 활용 역량을 증진시킬수 있어 뜻 깊은 시간이었던 것 같고, 그 과정이 매번 도전의 연속이었기에, 이번 학기 어떤 과목보다 재미있게 정말 시간 가는 줄 모르고 프로젝트를 만들었던 것 같다.

부록 – 소스코드



(메인 루프 다이어그램 by Doxygen)

GitHub: https://github.com/refracta/koreatech-assignment/tree/master/MicroprocessorNPractice/Project

*. 추후 별도 Repository로 분리 예정, doc/html 폴더에 Doxygen 포함

EPPROMUtils.h

```
1  /*! @file */
2 #ifndef EEPROMMANGER_H
3 #define EEPROMMANGER_H
4
5 #include <Arduino.h>
6 #include <EEPROM.h>
7
8 #define SCORE_START 0
9 #define CUSTOM_MUSIC_START 64
10 #define CUSTOM_MUSIC_DATA_START (CUSTOM_MUSIC_START + 2)
11
12 /*
13  * 한 saveIndex 당 3 바이트씩 처리
14  * SAVE_INDEX=0 [ [SCORE_PART1] [SCORE_PART2] [MAX_COMBO] ]
```

```
 * SCORE 는 2 바이트 unsigned int
 * MAX_COMBO 는 1 바이트 unsigned char
 */

#define READ_SCORE(I) ((EEPROM.read(SCORE_START + I * 3 + 1) << 8) +
EEPROM.read(SCORE_START + I * 3 + 0))
#define WRITE_SCORE(I, S) EEPROM.write(SCORE_START + I * 3 + 0, S &
0xFF); EEPROM.write(SCORE_START + I * 3 + 1, (S & 0xFF00) >> 8)
#define READ_COMBO(I) (EEPROM.read(SCORE_START + I * 3 + 2))
#define WRITE_COMBO(I, C) EEPROM.write(SCORE_START + I * 3 + 2, C)
#define READ_CUSTOM_MUSIC_LENGTH() ((EEPROM.read(CUSTOM_MUSIC_START +
1) << 8) + EEPROM.read(CUSTOM_MUSIC_START + 0))
#define WRITE_CUSTOM_MUSIC_LENGTH(L) EEPROM.write(CUSTOM_MUSIC_START +
0, L & 0xFF); EEPROM.write(CUSTOM_MUSIC_START + 1, (L & 0xFF00) >> 8)
#endif
```

RGDotMatrixConst.h

```
/*! ₩file */
#ifndef RGDOTMATRIXCONST_H
#define RGDOTMATRIXCONST_H
#define MATRIX_SIZE 8
#endif
```

RGDotMatrixFont.h

```
  /*! ₩file */
#ifndef RGDOTMATRIXFONT_H
#define RGDOTMATRIXFONT_H
#define FONT_EMPTY 0x0000000000000000
#define FONT_FULL 0xffffffffffffffff
#define FONT_A 0x090909090f090906
#define FONT_B 0x0709090907090907
#define FONT_C 0x0e0101010101010e
#define FONT_D 0x0305090909090907
#define FONT_E 0x0f0101010f01010f
#define FONT_F 0x010101010701010f
#define FONT_G 0x0609090d01090906
#define FONT_H 0x090909090f090909
#define FONT_I 0x0f0404040404040f
#define FONT_J 0x030404040404040f
#define FONT_K 0x0909090503050909
#define FONT_L 0x0f01010101010101
#define FONT_M 0x0909090909090f09
#define FONT_N 0x0909090d0b090909
```

```
20 #define FONT_O 0x0609090909090906
21 #define FONT_P 0x0101010107090907
22 #define FONT_Q 0x0c060d0909090906
23 #define FONT_R 0x0909050307090907
24 #define FONT_S 0x070808080601010e
25 #define FONT_T 0x020202020202020f
26 #define FONT_U 0x0609090909090909
27 #define FONT_V 0x0205090909090909
28 #define FONT_W 0x090f090909090909
29 #define FONT_X 0x0909090609090909
30 #define FONT_Y 0x080808080e090909
31 #define FONT_Z 0x0f01010204080800f
32 #define FONT_a 0x0609090e08070000
33 #define FONT_b 0x0609090701010000
34 #define FONT_c 0x0e010101010e0000
35 #define FONT_d 0x0609090e08080000
36 #define FONT_e 0x0e01010f09060000
37 #define FONT_f 0x0202020f020c0000
38 #define FONT_g 0x07080e0909060000
39 #define FONT_h 0x0909090701010000
40 #define FONT_i 0x0202020200020000
41 #define FONT_j 0x0304040400040000
42 #define FONT_k 0x0905030509010000
43 #define FONT_l 0x0e01010101010000
44 #define FONT_m 0x0909090f0f060000
45 #define FONT_n 0x0909090909060000
46 #define FONT_o 0x0609090909060000
47 #define FONT_p 0x0101070909060000
48 #define FONT_q 0x08080e0909060000
49 #define FONT_r 0x0101010907010000
50 #define FONT_s 0x07080806010e0000
51 #define FONT_t 0x0c02020f02020000
52 #define FONT_u 0x0609090909090000
53 #define FONT_v 0x0205090909090000
54 #define FONT_w 0x060f0f0909090000
55 #define FONT_x 0x0909090609090000
56 #define FONT_y 0x0708080e09090000
57 #define FONT_z 0x0f010204080f0000
58 #define FONT_0 0x0609090d0b090906
59 #define FONT_1 0x0f04040404040406
60 #define FONT_2 0x0e01010608080806
61 #define FONT_3 0x0708080806080807
62 #define FONT_4 0x0404040f05050604
```

```
63  #define FONT_5 0x070808080601010f
64  #define FONT_6 0x060909090701010e
65  #define FONT_7 0x010202040608080f
66  #define FONT_8 0x0609090609090906
67  #define FONT_9 0x0708080e09090906
68  #define FONT_DOT 0x0303000000000000
69  #define FONT_COMMA 0x0302000000000000
70  #define FONT_LEFT_DOUBLE_QUOTATION_MARK 0x00000000000a0a00
71  #define FONT_RIGHT_DOUBLE_QUOTATION_MARK 0x0000000000050500
72  #define FONT_LEFT_SINGLE_QUOTATION_MARK 0x0000000000000404
73  #define FONT_RIGHT_SINGLE_QUOTATION_MARK 0x0000000000000101
74  #define FONT_UNKNOWN1 0x0000000000000103
75  #define FONT_QUESTION_MARK 0x020002020e080807
76  #define FONT_EXCLAMATION_MARK 0x0303000303030303
77  #define FONT_UNKNOWN2 0x0609010d0d090906
78  #define FONT_UNDERSCORE 0x0f00000000000000
79  #define FONT_UNKNOWN3 0x0000000000020702
80  #define FONT_UNKNOWN4 0x06060f06060f0606
81  #define FONT_DOLLAR 0x04070c0c06050e04
82  #define FONT_UNKNOWN5 0x0d0d010102040b0b
83  #define FONT_UNKNOWN6 0x06090d0902090906
84  #define FONT_LEFT_PARENTHESIS 0x0c0201010101020c
85  #define FONT_RIGHT_PARENTHESIS 0x0304080808080403
86  #define FONT_PLUS 0x0000020702000000
87  #define FONT_MINUS 0x0000000f00000000
88  #define FONT_SLASH 0x0101020204040808
89  #define FONT_UNKNOWN7 0x0303000303000000
90  #define FONT_UNKNOWN8 0x0302000303000000
91  #define FONT_LEFT_ANGLE_BRACKET 0x0804020101020408
92  #define FONT_EQUAL_SIGN 0x00000f000f000000
93  #define FONT_RIGHT_ANGLE_BRACKET 0x0102040808040201
94  #define FONT_LEFT_BRACKET 0x0f0101010101010f
95  #define BACK_SLASH 0x0808040402020101
96  #define FONT_RIGHT_BRACKET 0x0f0808080808080f
97  #define FONT_UNKNOWN9 0x0000000000050502
98  #define FONT_UNKNOWN10 0x0000000000000403
99  #define FONT_LEFT_BRACE 0x0c0202030302020c
100 #define FONT_VERTICAL_BAR 0x0202020202020202
101 #define FONT_RIGHT_BRACE 0x0304040c0c040403
102 #define FONT_UNKNOWN11 0x000000050a000000
103
104 #define FONT_GAME_COL1 0x0003030300000000
105 #define FONT_GAME_COL2 0x000c0c0c00000000
```

```
106 #define FONT_GAME_COL3 0x0030303000000000
107 #define FONT_GAME_COL4 0x00c0c0c000000000
108 #define FONT_UNDERLINE 0xff00000000000000
109
110 #define FONT_NOTE_COL1 0x0300000000000000
111 #define FONT_NOTE_COL2 0x0c00000000000000
112 #define FONT_NOTE_COL3 0x3000000000000000
113 #define FONT_NOTE_COL4 0xc000000000000000
114
115 #define FONT_NOTE_COL1_2x 0x0303000000000000
116 #define FONT_NOTE_COL2_2x 0x0c0c000000000000
117 #define FONT_NOTE_COL3_2x 0x3030000000000000
118 #define FONT_NOTE_COL4_2x 0xc0c0000000000000
119
120 #define NUM_OF_SIMPLE_FONT 42
121 const unsigned long long SIMPLE_FONTS[] = {FONT_0, FONT_1, FONT_2,
122 FONT_3, FONT_4, FONT_5, FONT_6, FONT_7, FONT_8,
123                                             FONT_9, FONT_DOT,
124 FONT_COMMA, FONT_EXCLAMATION_MARK,
125                                             FONT_SLASH,
126 FONT_EQUAL_SIGN, FONT_A, FONT_B,
127                                             FONT_C, FONT_D, FONT_E,
128 FONT_F, FONT_G, FONT_H, FONT_I, FONT_J, FONT_K,
129                                             FONT_L, FONT_M, FONT_N,
130 FONT_O, FONT_P, FONT_Q, FONT_R, FONT_S, FONT_T,
131                                             FONT_U, FONT_V, FONT_W,
132 FONT_X, FONT_Y, FONT_Z, FONT_EMPTY};
133 const char SIMPLE_FONT_CHARACTERS[] = {
134         '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '.', ',',
135 '!', '/',
136         '=', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K',
137 'L', 'M', 'N', 'O',
138         'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', ' '};
139
140 unsigned long long toSimpleFont(char ch) {
141     for (int i = 0; i < NUM_OF_SIMPLE_FONT; i++) {
142         if (ch == SIMPLE_FONT_CHARACTERS[i]) {
            return SIMPLE_FONTS[i];
        }
    }
    return FONT_EMPTY;
}
```

```
// from https://xantorohara.github.io/led-matrix-
editor/#0000000000000000|090909090f090906|0709090907090907|0e01010101
01010e|0305090909090907|0f0101010f01010f|010101010701010f|0609090d010
90906|090909090f090909|0f0404040404040f|030404040404040f|090909050305
0909|0f01010101010101|0909090909090f09|0909090d0b090909|0609090909090
906|0101010107090907|0c060d0909090906|0909050307090907|0708080806010
0e|0202020202020f|0609090909090909|0205090909090909|090f09090909090
9|0909090609090909|080808080e090909|0f0101020408080f|0609090e08070000
|0609090701010000|0e010101010e0000|0609090e08080000|0e01010f09060000|
0202020f020c0000|07080e0909060000|0909090701010000|0202020200020000|0
304040400040000|0905030509010000|0e01010101010000|0909090f0f060000|09
09090909060000|0609090909060000|0101070909060000|08080e0909060000|010
1010907010000|0708080601e0000|0c02020f02020000|0609090909090000|0205
090909090000|060f0f0909090000|0909090609090000|0708080e09090000|0f010
204080f0000|0609090d0b090906|0f04040404040406|0e01010608080806|070808
0806080807|0404040f05050604|070808080601010f|0609090907010e0|0102020
40608080f|0609090609090906|0708080e09090906|0303000000000000|03020000
00000000|00000000000a0a00|0000000000050500|0000000000000404|000000000
0000101|0000000000000103|020002020e080807|0303000303030303|0609010d0d
090906|0f00000000000000|0000000000020702|06060f06060f0606|04070c0c060
50e04|0d0d010102040b0b|06090d0902090906|0c0201010101020c|030408080808
0403|0000020702000000|0000000f00000000|0101020204040808|0303000303000
000|0302000303000000|0804020101020408|00000f000f000000|01020408080402
01|0f0101010101010f|0808040402020101|0f0808080808080f|000000000005050
2|0000000000000403|0c0202030302020c|0202020202020202|0304040c0c040403
|000000050a000000
#endif
```

RGDotMatrixImage.h

```
1  /*! Wfile */
2 #ifndef RGDOTMATRIXIMAGE_H
3 #define RGDOTMATRIXIMAGE_H
4
5 #include "RGDotMatrixConst.h"
6
7 enum ShiftDirection {
8     UP, DOWN, LEFT, RIGHT
9 };
10
11 static inline int rotate90IndexMapper(int r, int c) {
12     return (MATRIX_SIZE - c - 1) * MATRIX_SIZE + r;
13 }
14
```

```
15 static inline int rotate180IndexMapper(int r, int c) {
16     return (MATRIX_SIZE - 1 - r) * MATRIX_SIZE + MATRIX_SIZE - 1 - c;
17 }
18
19 static inline int rotate270IndexMapper(int r, int c) {
20     return c * MATRIX_SIZE + MATRIX_SIZE - 1 - r;
21 }
22
23 inline unsigned long long transform(unsigned long long value,
24 int(*indexMapper)(int r, int c)) {
25     unsigned long long result = 0;
26     for (int r = 0; r < MATRIX_SIZE; r++) {
27         for (int c = 0; c < MATRIX_SIZE; c++) {
28             int flatIndex = r * MATRIX_SIZE + c;
29             int rotatedIndex = indexMapper(r, c);
30             unsigned long long rotatedIndexValue = value & (1ULL <<
31 rotatedIndex);
32             result += (1ULL << flatIndex) * (rotatedIndexValue ?
33 1ULL : 0ULL);
34         }
35     }
36     return result;
37 }
38
39 class DotMatrixImage {
40 public:
41     unsigned long long rawImage;
42
43     inline DotMatrixImage transform(int(*indexMapper)(int r, int c)) {
44         return {::transform(rawImage, indexMapper)};
45     }
46
47     inline DotMatrixImage rotate90() {
48         return transform(rotate90IndexMapper);
49     }
50
51     inline DotMatrixImage rotate180() {
52         return transform(rotate180IndexMapper);
53     }
54
55     inline DotMatrixImage rotate270() {
56         return transform(rotate270IndexMapper);
57     }
```

```
58
59      DotMatrixImage shift(ShiftDirection direction, int shift) {
60          switch (direction) {
61              case UP:
62                  return *this >> shift * MATRIX_SIZE;
63              case DOWN:
64                  return *this << shift * MATRIX_SIZE;
65              case LEFT:
66                  return (rotate90() >> shift *
67  MATRIX_SIZE).rotate270();
68              case RIGHT:
69                  return (rotate90() << shift *
70  MATRIX_SIZE).rotate270();
71          }
72      }
73
74      DotMatrixImage operator<<(const int shift) {
75          return {rawImage << shift};
76      }
77
78      DotMatrixImage operator>>(const int shift) {
79          return {rawImage >> shift};
80      }
81
82      DotMatrixImage operator|(const DotMatrixImage &image) {
83          return {rawImage | image.rawImage};
84      }
85
86      DotMatrixImage operator|(const int &target) {
87          return {rawImage | target};
88      }
89
90      DotMatrixImage operator&(const DotMatrixImage &image) {
91          return {rawImage & image.rawImage};
92      }
93
94      DotMatrixImage operator&(const int &target) {
95          return {rawImage & target};
96      }
97  };
98
99  class RGDotMatrixImage {
100 public:
```

```
101     unsigned long long redImage;
102     unsigned long long greenImage;
103 
104     inline RGDotMatrixImage transform(int(*indexMapper)(int r, int c))
105 {
106         return {::transform(redImage,
107 indexMapper), ::transform(greenImage, indexMapper)};
108     }
109 
110     inline RGDotMatrixImage rotate90() {
111         return transform(rotate90IndexMapper);
112     }
113 
114     inline RGDotMatrixImage rotate180() {
115         return transform(rotate180IndexMapper);
116     }
117 
118     inline RGDotMatrixImage rotate270() {
119         return transform(rotate270IndexMapper);
120     }
121 
122     RGDotMatrixImage shift(ShiftDirection direction, int shift) {
123         switch (direction) {
124             case UP:
125                 return *this >> shift * MATRIX_SIZE;
126             case DOWN:
127                 return *this << shift * MATRIX_SIZE;
128             case LEFT:
129                 return (rotate90() >> shift *
130 MATRIX_SIZE).rotate270();
131             case RIGHT:
132                 return (rotate90() << shift *
133 MATRIX_SIZE).rotate270();
134         }
135     }
136 
137     RGDotMatrixImage operator<<(const int shift) {
138         return {redImage << shift, greenImage << shift};
139     }
140 
141     RGDotMatrixImage operator>>(const int shift) {
142         return {redImage >> shift, greenImage >> shift};
143     }
```

```
144
145     RGDotMatrixImage operator^(const RGDotMatrixImage &image) {
146         return {redImage ^ image.redImage, greenImage ^
147 image.greenImage};
148     }
149
150     RGDotMatrixImage operator^(const int &target) {
151         return {redImage ^ target, greenImage ^ target};
152     }
153
154     RGDotMatrixImage operator|(const RGDotMatrixImage &image) {
155         return {redImage | image.redImage, greenImage |
156 image.greenImage};
157     }
158
159     RGDotMatrixImage operator|(const int &target) {
160         return {redImage | target, greenImage | target};
161     }
162
        RGDotMatrixImage operator&(const RGDotMatrixImage &image) {
            return {redImage & image.redImage, greenImage &
    image.greenImage};
        }

        RGDotMatrixImage operator&(const int &target) {
            return {redImage & target, greenImage & target};
        }
    };


    #endif
```

RGDotMatrixRenderer.h

```
1   /*! Wfile */
 2 #ifndef RGDOTMATRIXRENDERER_H
 3 #define RGDOTMATRIXRENDERER_H
 4
 5 #include <Arduino.h>
 6 #include "RGDotMatrixConst.h"
 7 #include "RGDotMatrixImage.h"
 8 #include "RGDotMatrixFont.h"
 9
10 int generateLeftScrollImage(DotMatrixImage images[], String text) {
```

```
    int len = text.length();
    int index = 0;
    for (int i = 0; i < len; i++) {
        unsigned long long currentFont = toSimpleFont(text.charAt(i));
        unsigned long long nextFont = toSimpleFont(text.charAt((i +
1) % len));
        unsigned long long doubleNextFont =
toSimpleFont(text.charAt((i + 2) % len));
        DotMatrixImage currentImage = {currentFont};
        DotMatrixImage nextImage = {nextFont};
        DotMatrixImage doubleNextImage = {doubleNextFont};

        images[index++] = currentImage | nextImage.shift(RIGHT, 5);
        images[index++] = currentImage.shift(LEFT, 1) |
nextImage.shift(RIGHT, 4);
        images[index++] = currentImage.shift(LEFT, 2) |
nextImage.shift(RIGHT, 3);
        images[index++] = currentImage.shift(LEFT, 3) |
nextImage.shift(RIGHT, 2) | doubleNextImage.shift(RIGHT, 7);
        images[index++] = nextImage.shift(RIGHT, 1) |
doubleNextImage.shift(RIGHT, 6);
    }
    return index;
}

DotMatrixImage generateLeftScrollImage(String text, int frame) {
    int charIndex = frame / 5;
    int animateIndex = frame % 5;
    int len = text.length();
    unsigned long long currentFont =
toSimpleFont(text.charAt(charIndex));
    unsigned long long nextFont = toSimpleFont(text.charAt((charIndex
+ 1) % len));
    unsigned long long doubleNextFont =
toSimpleFont(text.charAt((charIndex + 2) % len));
    DotMatrixImage currentImage = {currentFont};
    DotMatrixImage nextImage = {nextFont};
    DotMatrixImage doubleNextImage = {doubleNextFont};

    DotMatrixImage result = nextImage.shift(RIGHT, 5 - animateIndex);
    if (animateIndex < 4) {
        result = result | (animateIndex == 0 ? currentImage :
currentImage.shift(LEFT, animateIndex));
```

```
    }
    if (animateIndex > 2) {
        result = result | doubleNextImage.shift(RIGHT, 10 -
animateIndex);
    }
    return result;
}


class RGDotMatrixRenderer {
private:
    void shiftRegister(unsigned char data) {
        digitalWrite(ST_CP_PIN, LOW);
        for (int i = 0; i < 8; i++) {
            digitalWrite(SH_CP_PIN, LOW);
            digitalWrite(DS_PIN, (data & (0x80 >> i)) ? HIGH : LOW);
            digitalWrite(SH_CP_PIN, HIGH);
        }
        digitalWrite(ST_CP_PIN, HIGH);
    }

    inline void controlRowPulse(unsigned char data) {
        shiftRegister(0x80 >> data);
    }

public:
    int DS_PIN;
    int ST_CP_PIN;
    int SH_CP_PIN;

    int RED_PINS[MATRIX_SIZE];

    int GREEN_PINS[MATRIX_SIZE];

    RGDotMatrixRenderer(int *RED_PINS, int *GREEN_PINS, int DS_PIN,
int ST_CP_PIN, int SH_CP_PIN) {
        for (int i = 0; i < MATRIX_SIZE; i++) {
            this->RED_PINS[i] = RED_PINS[i];
            this->GREEN_PINS[i] = GREEN_PINS[i];
        }
        this->DS_PIN = DS_PIN;
        this->ST_CP_PIN = ST_CP_PIN;
        this->SH_CP_PIN = SH_CP_PIN;
```

```
97      };
98
99      void init() {
100         for (int i = 0; i < MATRIX_SIZE; i++) {
101             pinMode(RED_PINS[i], OUTPUT);
102             pinMode(GREEN_PINS[i], OUTPUT);
103         }
104         pinMode(DS_PIN, OUTPUT);
105         pinMode(ST_CP_PIN, OUTPUT);
106         pinMode(SH_CP_PIN, OUTPUT);
107     }
108
109     void clearAllLED() {
110         for (int i = 0; i < MATRIX_SIZE; i++) {
111             for (int j = 0; j < MATRIX_SIZE; j++) {
112                 digitalWrite(RED_PINS[j], HIGH);
113                 digitalWrite(GREEN_PINS[j], HIGH);
114                 controlRowPulse(i);
115             }
116         }
117     }
118
119     void displayImage(RGDotMatrixImage &image, unsigned int
120 illuminance = 0) {
121         for (int r = 0; r < MATRIX_SIZE; r++) {
122             RGDotMatrixImage current = image.shift(UP, r) & 0xFF;
123
124             for (int c = 0; c < MATRIX_SIZE; c++) {
125                 digitalWrite(RED_PINS[c], bitRead(current.redImage,
126 c) ? LOW : HIGH);
127                 digitalWrite(GREEN_PINS[c],
    bitRead(current.greenImage, c) ? LOW : HIGH);
                 controlRowPulse(r);
                 delayMicroseconds(illuminance);
                 controlRowPulse(-1);
             }

             for (int c = 0; c < MATRIX_SIZE; c++) {
                 digitalWrite(RED_PINS[c], HIGH); // LED OFF
                 digitalWrite(GREEN_PINS[c], HIGH); // LED OFF
             }
         }
     }
```

```
};

#endif
```

SoundController.h

```
/*! Wfile */
#ifndef SOUNDCONTROLLER_H
#define SOUNDCONTROLLER_H

#include <Arduino.h>
#include <EEPROM.h>

#define IS_DIGIT(N) (N >= '0' && N <= '9')
#define IS_PLAYABLE_NOTE(N) (N == 'c' || N == 'C' || N == 'd' || N == 'D' || N == 'e' || N == 'f' || N == 'F' || N == 'g' || N == 'G' || N == 'a' || N =='A' || N == 'b' || N == 'p')
#define OCTAVE_OFFSET 0

#define NOTE rtttl.charAt(cursor)
#define NOTE_PP rtttl.charAt(cursor++)

/*
 * data from https://www.arduino.cc/en/Tutorial/BuiltInExamples/toneMelody
 * 아두이노 튜토리얼 문서의 헤더 파일을 배열 형태로 정리한 것, 각 음계의 진동수의 상수 정의이다.
 */
const int frequencies[] = {
        31,
// octave 0
        33, 35, 37, 39, 41, 44, 46, 49, 52, 55, 58, 62,
// octave 1
        65, 69, 73, 78, 82, 87, 93, 98, 104, 110, 117, 123,
// octave 2
        131, 139, 147, 156, 165, 175, 185, 196, 208, 220, 233, 247,
// octave 3
        262, 277, 294, 311, 330, 349, 370, 392, 415, 440, 466, 494,
// octave 4
        523, 554, 587, 622, 659, 698, 740, 784, 831, 880, 932, 988,
// octave 5
        1047, 1109, 1175, 1245, 1319, 1397, 1480, 1568, 1661, 1760, 1865, 1976,    // octave 6
```

```
39          2093, 2217, 2349, 2489, 2637, 2794, 2960, 3136, 3322, 3520,
40 3729, 3951,     // octave 7
41          4186, 4435, 4699, 4978
42 // octave 8
43 };
44
45 // note에 해당하는 notes의 index를 반환한다.
46 inline int getNoteIndex(char note) {
47     switch (note) {
48         case 'c':
49             return 0;
50         case 'C':
51             return 1;
52         case 'd':
53             return 2;
54         case 'D':
55             return 3;
56         case 'e':
57             return 4;
58         case 'f':
59             return 5;
60         case 'F':
61             return 6;
62         case 'g':
63             return 7;
64         case 'G':
65             return 8;
66         case 'a':
67             return 9;
68         case 'A':
69             return 10;
70         case 'b':
71             return 11;
72         default:
73             return -1;
74     }
75 }
76
77 int getRTTTLMinDuration(String &rtttl) {
78     int minDuration = 32767;
79
80     int size = rtttl.length();
81     byte defaultDuration = 4;
```

```
82      int bpm = 63;
83      int num;
84      long noteTime;
85      long duration;
86      int cursor;
87
88      // RTTTL 기본 포맷: d=N,o=N,b=NNN:
89      while (NOTE != ':') {
90          cursor++;
91      }    // 곡 제목 넘김
92      cursor++;
93      // ':' 넘기기
94
95      // get default duration
96      if (NOTE == 'd') {
97          cursor++;
98          cursor++;
99          // "d=" 넘기기
100         num = 0;
101         while (IS_DIGIT(NOTE)) {
102             num = (num * 10) + (NOTE_PP - '0');
103         }
104         if (num > 0) {
105             defaultDuration = num;
106         }
107         cursor++;
108         // "," 넘기기
109     }
110
111     // 기본 옥타브 설정
112     if (NOTE == 'o') {
113         cursor++;
114         cursor++;
115         cursor++;
116         // "," 넘기기
117     }
118
119     // BPM 설정
120     if (NOTE == 'b') {
121         cursor++;
122         cursor++;
123         // "b=" 넘기기
124         num = 0;
```

```
125         while (IS_DIGIT(NOTE)) {
126             num = (num * 10) + (NOTE_PP - '0');
127         }
128         bpm = num;
129         cursor++;
130         // "," 넘기기
131     }
132     noteTime = (60 * 1000L / bpm) * 4;
133     // 메인 루프
134
135
136     while (cursor < size) {
137         // 가능한 경우, 노트 박자 계산
138         num = 0;
139         while (IS_DIGIT(NOTE)) {
140             num = (num * 10) + (NOTE_PP - '0');
141         }
142
143         if (num) {
144             duration = noteTime / num;
145         } else {
146             duration = noteTime / defaultDuration;
147         }
148
149         cursor++;
150
151         if (NOTE == '#') {
152             cursor++;
153         }
154
155         // "." 스트로크 박자 처리
156         if (NOTE == '.') {
157             duration += duration / 2;
158             cursor++;
159         }
160
161         // 옥타브 처리
162         if (IS_DIGIT(NOTE)) {
163             cursor++;
164         }
165
166         if (NOTE == ',') {
167             cursor++;
```

```
        }

        if (duration < minDuration) {
            minDuration = duration;
        }
    }
    return minDuration;
}

struct NoteData {
    char note;
    int octave;
    int duration;
};

class SoundController {
private:
    int buzzerPin;

    void (*delayFunction)(unsigned long);

public:
    SoundController(int buzzerPin, void (*delayFunction)(unsigned
long)) : buzzerPin(buzzerPin),

delayFunction(delayFunction) {}

    void init() {
        pinMode(buzzerPin, OUTPUT);
    }

    void playTone(char note, int octave, int duration) {
        if (note == 'p') {
            delayFunction(duration);
            return;
        }
        int frequency = frequencies[getNoteIndex(note) - 11 + octave *
12];
        tone(buzzerPin, frequency, duration);
        delayFunction(duration);
        noTone(buzzerPin);
    }

```

```
211     void playTone(NoteData &data) {
212         playTone(data.note, data.octave, data.duration);
213     }
214 
215     void playRTTTL(String &rtttl, void (*nextHandler)(NoteData *queue,
216 int size),
217                    int numOfNext) {
218         int size = rtttl.length();
219         byte defaultDuration = 4;
220         byte octave = 6;
221         int bpm = 63;
222         int num;
223         long noteTime;
224         long duration;
225         char note;
226         byte scale;
227         int cursor;
228 
229 
230 
231         // RTTTL 기본 포맷: d=N,o=N,b=NNN:
232         while (NOTE != ':') {
233             cursor++;
234         }    // 곡 제목 넘김
235         cursor++;
236         // ':' 넘기기
237 
238         // get default duration
239         if (NOTE == 'd') {
240             cursor++;
241             cursor++;
242             // "d=" 넘기기
243             num = 0;
244             while (IS_DIGIT(NOTE)) {
245                 num = (num * 10) + (NOTE_PP - '0');
246             }
247             if (num > 0) {
248                 defaultDuration = num;
249             }
250             cursor++;
251             // "," 넘기기
252         }
253 
```

```
254         // 기본 옥타브 설정
255         if (NOTE == 'o') {
256             cursor++;
257             cursor++;
258             // "o=" 넘기기
259             num = NOTE_PP - '0';
260             if (num >= 3 && num <= 7) {
261                 octave = num;
262             }
263             cursor++;
264             // "," 넘기기
265         }
266 
267         // BPM 설정
268         if (NOTE == 'b') {
269             cursor++;
270             cursor++;
271             // "b=" 넘기기
272             num = 0;
273             while (IS_DIGIT(NOTE)) {
274                 num = (num * 10) + (NOTE_PP - '0');
275             }
276             bpm = num;
277             cursor++;
278             // "," 넘기기
279         }
280         noteTime = (60 * 1000L / bpm) * 4;
281         // 메인 루프
282 
283         NoteData *queue = (NoteData *) malloc(sizeof(NoteData) *
284 numOfNext);
285         int queueCursor = 0;
286         while (cursor < size) {
287             // 가능한 경우, 노트 박자 계산
288             num = 0;
289             while (IS_DIGIT(NOTE)) {
290                 num = (num * 10) + (NOTE_PP - '0');
291             }
292 
293             if (num) {
294                 duration = noteTime / num;
295             } else {
296                 duration = noteTime / defaultDuration;
```

```
297             }
298 
299             note = NOTE;
300             cursor++;
301 
302             if (NOTE == '#') {
303                 note = toupper(note);
304                 cursor++;
305             }
306 
307             // "." 스트로크 박자 처리
308             if (NOTE == '.') {
309                 duration += duration / 2;
310                 cursor++;
311             }
312 
313             // 옥타브 처리
314             if (IS_DIGIT(NOTE)) {
315                 scale = NOTE - '0';
316                 cursor++;
317             } else {
318                 scale = octave;
319             }
320 
321             scale += OCTAVE_OFFSET;
322 
323             if (NOTE == ',') {
324                 cursor++;
325             }
326 
327             if (queueCursor == numOfNext) {
328                 nextHandler(queue, queueCursor);
329                 playTone(queue[0]);
330                 for (int i = 1; i < numOfNext; i++) {
331                     queue[i - 1] = queue[i];
332                 }
333                 queueCursor--;
334             }
335             if (IS_PLAYABLE_NOTE(note)) {
336                 queue[queueCursor++] = {note, scale, duration};
337             }
338 
339         }
```

```
340         int leftSize = queueCursor;
341         for (int i = 0; i < leftSize; i++) {
342             nextHandler(queue + i, numOfNext - i);
343             playTone(queue[i]);
344         }
345         free(queue);
346     }
347
348 #define NOTE EEPROM.read(cursor)
349 #define NOTE_PP EEPROM.read(cursor++)
350
351     void playRTTTLFromEEPROM(int start, int end, void
352 (*nextHandler)(NoteData *queue, int size),
353                              int numOfNext) {
354         int size = end - start;
355         byte defaultDuration = 4;
356         byte octave = 6;
357         int bpm = 63;
358         int num;
359         long noteTime;
360         long duration;
361         char note;
362         byte scale;
363         int cursor = start;
364
365         // RTTTL 기본 포맷: d=N,o=N,b=NNN:
366         while (NOTE != ':') {
367             cursor++;
368         }    // 곡 제목 넘김
369         cursor++;
370         // ':' 넘기기
371
372         // get default duration
373         if (NOTE == 'd') {
374             cursor++;
375             cursor++;
376             // "d=" 넘기기
377             num = 0;
378             while (IS_DIGIT(NOTE)) {
379                 num = (num * 10) + (NOTE_PP - '0');
380             }
381             if (num > 0) {
382                 defaultDuration = num;
```

```
383             }
384             cursor++;
385             // "," 넘기기
386         }
387 
388         // 기본 옥타브 설정
389         if (NOTE == 'o') {
390             cursor++;
391             cursor++;
392             // "o=" 넘기기
393             num = NOTE_PP - '0';
394             if (num >= 3 && num <= 7) {
395                 octave = num;
396             }
397             cursor++;
398             // "," 넘기기
399         }
400 
401         // BPM 설정
402         if (NOTE == 'b') {
403             cursor++;
404             cursor++;
405             // "b=" 넘기기
406             num = 0;
407             while (IS_DIGIT(NOTE)) {
408                 num = (num * 10) + (NOTE_PP - '0');
409             }
410             bpm = num;
411             cursor++;
412             // "," 넘기기
413         }
414         noteTime = (60 * 1000L / bpm) * 4;
415         // 메인 루프
416 
417         NoteData *queue = (NoteData *) malloc(sizeof(NoteData) *
418 numOfNext);
419         int queueCursor = 0;
420         while (cursor < end) {
421             // 가능한 경우, 노트 박자 계산
422             num = 0;
423             while (IS_DIGIT(NOTE)) {
424                 num = (num * 10) + (NOTE_PP - '0');
425             }
```

```
426
427             if (num) {
428                 duration = noteTime / num;
429             } else {
430                 duration = noteTime / defaultDuration;
431             }
432
433             note = NOTE;
434             cursor++;
435
436             if (NOTE == '#') {
437                 note = toupper(note);
438                 cursor++;
439             }
440
441             // "." 스트로크 박자 처리
442             if (NOTE == '.') {
443                 duration += duration / 2;
444                 cursor++;
445             }
446
447             // 옥타브 처리
448             if (IS_DIGIT(NOTE)) {
449                 scale = NOTE - '0';
450                 cursor++;
451             } else {
452                 scale = octave;
453             }
454
455             scale += OCTAVE_OFFSET;
456
457             if (NOTE == ',') {
458                 cursor++;
459             }
460
461             if (queueCursor == numOfNext) {
462                 nextHandler(queue, queueCursor);
463                 playTone(queue[0]);
464                 for (int i = 1; i < numOfNext; i++) {
465                     queue[i - 1] = queue[i];
466                 }
                queueCursor--;
            }
```

```
                if (IS_PLAYABLE_NOTE(note)) {
                    queue[queueCursor++] = {note, scale, duration};
                }

            }
            int leftSize = queueCursor;
            for (int i = 0; i < leftSize; i++) {
                nextHandler(queue + i, numOfNext - i);
                playTone(queue[i]);
            }
            free(queue);
        }

    #undef NOTE
    #undef NOTE_PP
    };

    #endif
```

Tune.h

```
/*! Wfile */
#ifndef TUNE_H
#define TUNE_H

#define SUCCESS_TUNE(S) S.playTone('g', 5, 200);
#define FAIL_TUNE(S) S.playTone('F', 6, 100); S.playTone('F', 6, 100);

#endif
```

Project.h

```
   /*! Wfile */
  #ifndef PROJECT_H
  #define PROJECT_H

  #include <Arduino.h>
  #include "RGDotMatrixRenderer.h"
  #include "RGDotMatrixFont.h"
  #include "SoundController.h"
  #include "Tune.h"
  #include "EEPROMUtils.h"

  #define NOTE_PREVIEW 3
```

```
#define NUM_OF_MUSIC_DATA 4
#define CUSTOM_MUSIC_INDEX 3

// 디버그 플래그
#define DEBUG_MODE
#ifdef DEBUG_MODE
#define debug(P) Serial.print(P)
#define debugln(P) Serial.println(P)
#else
#define debug(P)
#define debugln(P)
#endif

// 버튼
#define BUTTON1_PIN 8
#define BUTTON2_PIN A3
#define BUTTON3_PIN 0
#define BUTTON4_PIN 1
#define NUM_OF_BUTTON 4

#define IS_BUTTON_PRESSED(BUTTON_PIN) (!digitalRead(BUTTON_PIN))
#define IS_ANY_BUTTON_PRESSED() (IS_BUTTON_PRESSED(BUTTON1_PIN) ||
IS_BUTTON_PRESSED(BUTTON2_PIN) || IS_BUTTON_PRESSED(BUTTON3_PIN) ||
IS_BUTTON_PRESSED(BUTTON4_PIN))

#define DISABLE_SERIAL_PIN_BUTTONS() pinMode(BUTTON3_PIN, INPUT);
pinMode(BUTTON4_PIN, INPUT);
#define ENABLE_SERIAL_PIN_BUTTONS() pinMode(BUTTON3_PIN,
INPUT_PULLUP); pinMode(BUTTON4_PIN, INPUT_PULLUP);

const int BUTTON_PINS[] = {BUTTON1_PIN, BUTTON2_PIN, BUTTON3_PIN,
BUTTON4_PIN};

// 렌더러와 렌더 버퍼
RGDotMatrixRenderer renderer((int[]) {A4, A5, 2, 3, 4, 5, 6, 7},
(int[]) {10, 10, 11, 11, 12, 12, 13, 13}, A0, A1, A2);
DotMatrixImage *rib = nullptr; // Single renderImageBuffer
int ribSize;
int playIndex = 0;
RGDotMatrixImage renderImage;

// 딜레이 하이재킹
void (*_delay)(unsigned long) = delay;
```

```
56
57 #define delay(MS) delayWithCustomHandler(MS)
58
59 void (*customHandler)(unsigned long, unsigned long);
60
61 void delayWithCustomHandler(unsigned long ms) {
62     unsigned long current = millis();
63     unsigned long delta;
64     while ((delta = millis() - current) < ms) {
65         if (customHandler) {
66             customHandler(current, ms - delta);
67         }
68     }
69 }
70
71 // 음악 데이터
72 String musicData[NUM_OF_MUSIC_DATA];
73 int musicDataCursor = 0;
74 SoundController soundController(9, delayWithCustomHandler);
75
76 void initMusicData() {
77     musicData[0] = F(
78             "TAKE ON
79 ME:d=4,o=4,b=160:8f#5,8f#5,8f#5,8d5,8p,8b,8p,8e5,8p,8e5,8p,8e5,8g#5,8
80 g#5,8a5,8b5,8a5,8a5,8a5,8e5,8p,8d5,8p,8f#5,8p,8f#5,8p,8f#5,8e5,8e5,8f
81 #5,8e5,8f#5,8f#5,8f#5,8d5,8p,8b,8p,8e5,8p,8e5,8p,8e5,8g#5,8g#5,8a5,8b
82 5,8a5,8a5,8a5,8e5,8p,8d5,8p,8f#5,8p,8f#5,8p,8f#5,8e5,8e5");
83     musicData[1] = F(
84             "THE
85 SIMPSONS:d=4,o=5,b=160:c.6,e6,f#6,8a6,g.6,e6,c6,8a,8f#,8f#,8f#,2g,8p,
86 8p,8f#,8f#,8f#,8g,a#.,8c6,8c6,8c6,c6");
87     musicData[2] = F(
88
89 "ENTERTAINER:d=4,o=5,b=140:8d,8d#,8e,c6,8e,c6,8e,2c.6,8c6,8d6,8d#6,8e
90 6,8c6,8d6,e6,8b,d6,2c6,p,8d,8d#,8e,c6,8e,c6,8e,2c.6,8p,8a,8g,8f#,8a,8
91 c6,e6,8d6,8c6,8a,2d6");
92     musicData[3] = F("CUSTOM:");
93 }
94
95 // 게임 상태
96 enum GameStatus {
97     MAIN_SCREEN, SELECT_MUSIC_SCREEN, GAME_SCREEN, GAME_RESULT_SCREEN
98 };
```

```
 99  boolean isInited = false;
100  GameStatus previousStatus;
101  GameStatus currentStatus = MAIN_SCREEN;
102
103  // 인게임 노트 데이터 처리
104  char nextNotes[3];
105  boolean isHit = false;
106  int BASE_TIME;
107
108  // 인게임 렌더링
109  unsigned long long FONT_NOTE_COLS[] = {FONT_NOTE_COL1,
110  FONT_NOTE_COL2, FONT_NOTE_COL3, FONT_NOTE_COL4};
111  unsigned long long FONT_GAME_COLS[] = {FONT_GAME_COL1,
112  FONT_GAME_COL2, FONT_GAME_COL3, FONT_GAME_COL4};
113  struct TimeBasedRGDotMatrix {
114      RGDotMatrixImage image;
115      long targetTime;
     };
     #define NUM_OF_TIME_BASED_EFFECT 3
     TimeBasedRGDotMatrix timeBasedEffects[NUM_OF_TIME_BASED_EFFECT];

     // 인게임 데이터
     int maxCombo = 0;
     int combo = 0;
     unsigned int score = 0;

     // 스크롤 메시지 버퍼
     #define MESSAGE_BUFFER_LENGTH 20
     char message[MESSAGE_BUFFER_LENGTH];
     int messageLen;

     void changeGameStatus(GameStatus);

     #endif
```

Project.cpp

```
1   /*! Wfile */
 2 #include "Project.h"
 3
 4 void renderTarget(unsigned long current, unsigned long left) {
 5     renderer.displayImage(renderImage);
 6 }
 7
```

```
 8 void handleMainScreenDelay(unsigned long current, unsigned long left)
 9 {
10     renderer.displayImage(renderImage);
11     if (IS_ANY_BUTTON_PRESSED()) {
12         changeGameStatus(SELECT_MUSIC_SCREEN);
13         SUCCESS_TUNE(soundController);
14     }
15 }
16
17 void handleMusicSelectScreenDelay(unsigned long current, unsigned long
18 left) {
19     renderer.displayImage(renderImage);
20     if (IS_BUTTON_PRESSED(BUTTON1_PIN)) {
21         musicDataCursor = (musicDataCursor - 1 + NUM_OF_MUSIC_DATA) %
22 NUM_OF_MUSIC_DATA;
23         changeGameStatus(SELECT_MUSIC_SCREEN);
24         SUCCESS_TUNE(soundController);
25     } else if (IS_BUTTON_PRESSED(BUTTON4_PIN)) {
26         musicDataCursor = (musicDataCursor + 1) % NUM_OF_MUSIC_DATA;
27         changeGameStatus(SELECT_MUSIC_SCREEN);
28         SUCCESS_TUNE(soundController);
29     } else if (IS_BUTTON_PRESSED(BUTTON2_PIN)) {
30         changeGameStatus(GAME_SCREEN);
31         SUCCESS_TUNE(soundController);
32     } else if (IS_BUTTON_PRESSED(BUTTON3_PIN)) {
33         score = READ_SCORE(musicDataCursor);
34         maxCombo = READ_COMBO(musicDataCursor);
35         changeGameStatus(GAME_RESULT_SCREEN);
36         SUCCESS_TUNE(soundController);
37     }
38 }
39
40 int getEmptyTBEIndex() {
41     for (int i = 0; i < NUM_OF_TIME_BASED_EFFECT; i++) {
42         if (timeBasedEffects[i].targetTime <= 0) {
43             return i;
44         }
45     }
46     return 0;
47 }
48
49 void handleGameScreenDelay(unsigned long current, unsigned long left)
50 {
```

```
51      RGDotMatrixImage temp = renderImage;
52      for (int i = 0; i < NUM_OF_BUTTON; i++) {
53          if (IS_BUTTON_PRESSED(BUTTON_PINS[i])) {
54              renderImage = renderImage & (RGDotMatrixImage)
55  {~FONT_NOTE_COLS[i], FONT_FULL} |
56                              (RGDotMatrixImage) {FONT_EMPTY,
57  FONT_NOTE_COLS[i]};
58          }
59      }
60
61      int timeFactor = left / BASE_TIME;
62      int targetIndex;
63      unsigned long long targetNote;
64      boolean isValid;
65
66
67      int previewEffectIndexes[3] = {-1, -1, -1};
68      if (nextNotes[0]) {
69          previewEffectIndexes[0] = targetIndex =
70  getNoteIndex(nextNotes[0]) % 4;
71          targetNote = FONT_NOTE_COLS[targetIndex];
72          isValid = IS_BUTTON_PRESSED(BUTTON_PINS[targetIndex]);
73
74          if (!isHit) {
75              if (timeFactor < 8) {
76                  renderImage = renderImage | (RGDotMatrixImage)
77  {targetNote, FONT_EMPTY}.shift(UP, timeFactor);
78              } else {
79                  renderImage = renderImage | (RGDotMatrixImage)
80  {targetNote, FONT_EMPTY}.shift(UP, 7);
81              }
82              if (isValid) {
83                  isHit = true;
84                  combo++;
85                  if (combo > maxCombo) {
86                      maxCombo = combo;
87                  }
88                  long effectTime = current + 250;
89                  if (timeFactor < 4) {
90                      score += 30 * (1 + combo / 10.0);
91                      timeBasedEffects[getEmptyTBEIndex()] = {
92                              (RGDotMatrixImage) {FONT_EMPTY,
93
```

```
94
95 FONT_GAME_COLS[targetIndex]}, effectTime};
96                 } else if (timeFactor < 8) {
97                     score += 20 * (1 + combo / 10.0);
98                     timeBasedEffects[getEmptyTBEIndex()] = {
99                             (RGDotMatrixImage)
100 {FONT_GAME_COLS[targetIndex],
101
102 FONT_GAME_COLS[targetIndex]}, effectTime};
103                 } else {
104                     score += 1 * (1 + combo / 10.0);
105                     timeBasedEffects[getEmptyTBEIndex()] =
106 {(RGDotMatrixImage) {FONT_GAME_COLS[targetIndex]},
107
108 effectTime};
109                 }
110             }
111         }
112     }
113
114     for (int i = 1; i < 3; i++) {
115         char nextNote = nextNotes[i];
116         if (nextNote) {
117             targetIndex = getNoteIndex(nextNote) % 4;
118             boolean isNewPosition = targetIndex !=
119 previewEffectIndexes[0] && targetIndex != previewEffectIndexes[1] &&
120                                     targetIndex !=
121 previewEffectIndexes[2];
122             previewEffectIndexes[i] = targetIndex;
123             targetNote = FONT_NOTE_COLS[targetIndex];
124             if (isNewPosition) {
125                 renderImage = renderImage | (RGDotMatrixImage) {i ==
126 1 ? targetNote : FONT_EMPTY, targetNote}.shift(UP,
127
128 7);
129             }
130         }
131     }
132
133
134     for (int i = 0; i < NUM_OF_TIME_BASED_EFFECT; i++) {
135         if (0 < timeBasedEffects[i].targetTime && current <=
136 timeBasedEffects[i].targetTime) {
```

```
137             renderImage = renderImage | timeBasedEffects[i].image;
138         } else {
139             timeBasedEffects[i].targetTime = 0;
140         }
141     }
142 
143     int maxRenderCombo = combo < 8 ? combo : 8;
144     for (int i = 0; i < maxRenderCombo; i++) {
145         renderImage.redImage ^= 1ULL << (7 * 8 + i);
146         renderImage.greenImage |= 1ULL << (7 * 8 + i);
147     }
148 
149     renderer.displayImage(renderImage);
150     renderImage = temp;
151 }
152 
153 void handleGameResultDelay(unsigned long current, unsigned long left)
154 {
155     renderer.displayImage(renderImage);
156     if (IS_ANY_BUTTON_PRESSED()) {
157         changeGameStatus(SELECT_MUSIC_SCREEN);
158         SUCCESS_TUNE(soundController);
159     }
160 }
161 
162 void changeGameStatus(GameStatus status) {
163     customHandler = renderTarget;
164     previousStatus = currentStatus;
165     currentStatus = status;
166     isInited = false;
167 }
168 
169 void nextHandler(NoteData *notes, int size) {
170     for (int i = 0; i < NOTE_PREVIEW; i++) {
171         nextNotes[i] = (i < size && notes[i].note != 'p') ?
172 notes[i].note : NULL;
173     }
174     if (!isHit) {
175         combo = 0;
176     }
177     isHit = false;
178 }
179 
```

```
180 void initGameStatus() {
181     free(rib);
182     ribSize = 0;
183     playIndex = 0;
184     isInited = true;
185     if (currentStatus == MAIN_SCREEN) {
186         strcpy(message, "RHYTHM STAR!");
187         messageLen = strlen(message);
188         ribSize = messageLen * 5;
189         customHandler = handleMainScreenDelay;
190     } else if (currentStatus == SELECT_MUSIC_SCREEN) {
191         String &data = musicData[musicDataCursor];
192         data.substring(0, data.indexOf(":")).toCharArray(message,
193 MESSAGE_BUFFER_LENGTH);
194         messageLen = strlen(message);
195         message[messageLen++] = ' ';
196         message[messageLen] = 'WO';
197         ribSize = messageLen * 5;
198         customHandler = handleMusicSelectScreenDelay;
199     } else if (currentStatus == GAME_SCREEN) {
200         customHandler = renderTarget;
201         renderImage = {(DotMatrixImage{FONT_3}).shift(RIGHT,
202 2).rawImage, FONT_FULL};
203         soundController.playTone('g', 5, 200);
204         delay(800);
205         renderImage = {(DotMatrixImage{FONT_2}).shift(RIGHT,
206 2).rawImage, FONT_FULL};
207         soundController.playTone('g', 5, 200);
208         delay(800);
209         renderImage = {(DotMatrixImage{FONT_1}).shift(RIGHT,
210 2).rawImage, FONT_FULL};
211         soundController.playTone('g', 5, 200);
212         delay(800);
213         renderImage = {((DotMatrixImage) {FONT_G} | (DotMatrixImage)
214 {FONT_O}.shift(RIGHT, 4)).rawImage, FONT_FULL};
215         soundController.playTone('g', 7, 200);
216         delay(800);
217 
218         customHandler = handleGameScreenDelay;
219         renderImage = {FONT_UNDERLINE, FONT_EMPTY};
220         String &data = musicData[musicDataCursor];
221         BASE_TIME = getRTTTLMinDuration(data) / 8;
222         // BASE_TIME = 30; fix base time
```

```
223
224         combo = 0;
225         score = 0;
226         if (strcmp(message, "CUSTOM ") == 0) {
227
228 soundController.playRTTTLFromEEPROM(CUSTOM_MUSIC_DATA_START,
229
230 CUSTOM_MUSIC_DATA_START + READ_CUSTOM_MUSIC_LENGTH(), nextHandler,
231                                                     NOTE_PREVIEW);
232         } else {
233             soundController.playRTTTL(data, nextHandler,
234 NOTE_PREVIEW);
235         }
236         nextNotes[0] = nextNotes[1] = nextNotes[2] = 0;
237         if (score >= READ_SCORE(musicDataCursor)) {
238             WRITE_SCORE(musicDataCursor, score);
239             WRITE_COMBO(musicDataCursor, maxCombo);
240         }
241         changeGameStatus(GAME_RESULT_SCREEN);
242     } else if (currentStatus == GAME_RESULT_SCREEN) {
243         sprintf(message, "S=%u C=%u ", score, maxCombo);
244         messageLen = strlen(message);
245         customHandler = handleGameResultDelay;
246     }
247 }
248
249 void runMainScreen() {
250     renderImage = {generateLeftScrollImage(message,
251 playIndex).rawImage, FONT_FULL};
252     delay(100);
253     playIndex = (playIndex + 1) % ribSize;
254
255     int numOfRead = 0;
256     while (Serial.available()) {
257         EEPROM.write(CUSTOM_MUSIC_DATA_START + numOfRead++,
258 Serial.read());
259         _delay(9);
260     }
261     if (numOfRead == 7) {
262         for (int i = 0; i < EEPROM.length(); i++) {
263             EEPROM.write(i, 0);
264         }
265         Serial.println("clear eeprom successfully!");
```

```
266     } else if (numOfRead) {
267         WRITE_CUSTOM_MUSIC_LENGTH(numOfRead);
268         Serial.print(READ_CUSTOM_MUSIC_LENGTH());
269         Serial.println(" length saved!");
270         /*
271         for (int i = CUSTOM_MUSIC_DATA_START; i <
272 CUSTOM_MUSIC_DATA_START + numOfRead; i++) {
273             Serial.print((char) EEPROM.read(i));
274         }
275         Serial.println();
276         */
277         musicDataCursor = CUSTOM_MUSIC_INDEX;
278         changeGameStatus(SELECT_MUSIC_SCREEN);
279         _delay(300);
280     }
281 }
282
283 void runMusicSelectScreen() {
284     renderImage = {generateLeftScrollImage(message,
285 playIndex).rawImage, 0};
286     delay(100);
287     playIndex = (playIndex + 1) % ribSize;
288 }
289
290 void runGameScreen() {
291     delay(1000);
292 }
293
294 void runGameResultScreen() {
295     renderImage = {generateLeftScrollImage(message,
296 playIndex).rawImage, 0};
297     ribSize = messageLen * 5;
298     delay(100);
299     playIndex = (playIndex + 1) % ribSize;
300 }

    void setup() {
        Serial.begin(1200);
        renderer.init();
        soundController.init();
        initMusicData();

        pinMode(BUTTON1_PIN, INPUT_PULLUP);
```

```
    pinMode(BUTTON2_PIN, INPUT_PULLUP);
    pinMode(BUTTON3_PIN, INPUT_PULLUP);
    pinMode(BUTTON4_PIN, INPUT_PULLUP);

    changeGameStatus(MAIN_SCREEN);
}

void loop() {
    if (isInited) {
        switch (currentStatus) {
            case MAIN_SCREEN:
                runMainScreen();
                break;
            case SELECT_MUSIC_SCREEN:
                runMusicSelectScreen();
                break;
            case GAME_SCREEN:
                runGameScreen();
                break;
            case GAME_RESULT_SCREEN:
                runGameResultScreen();
                break;
        }
    } else {
        initGameStatus();
    }
}
```