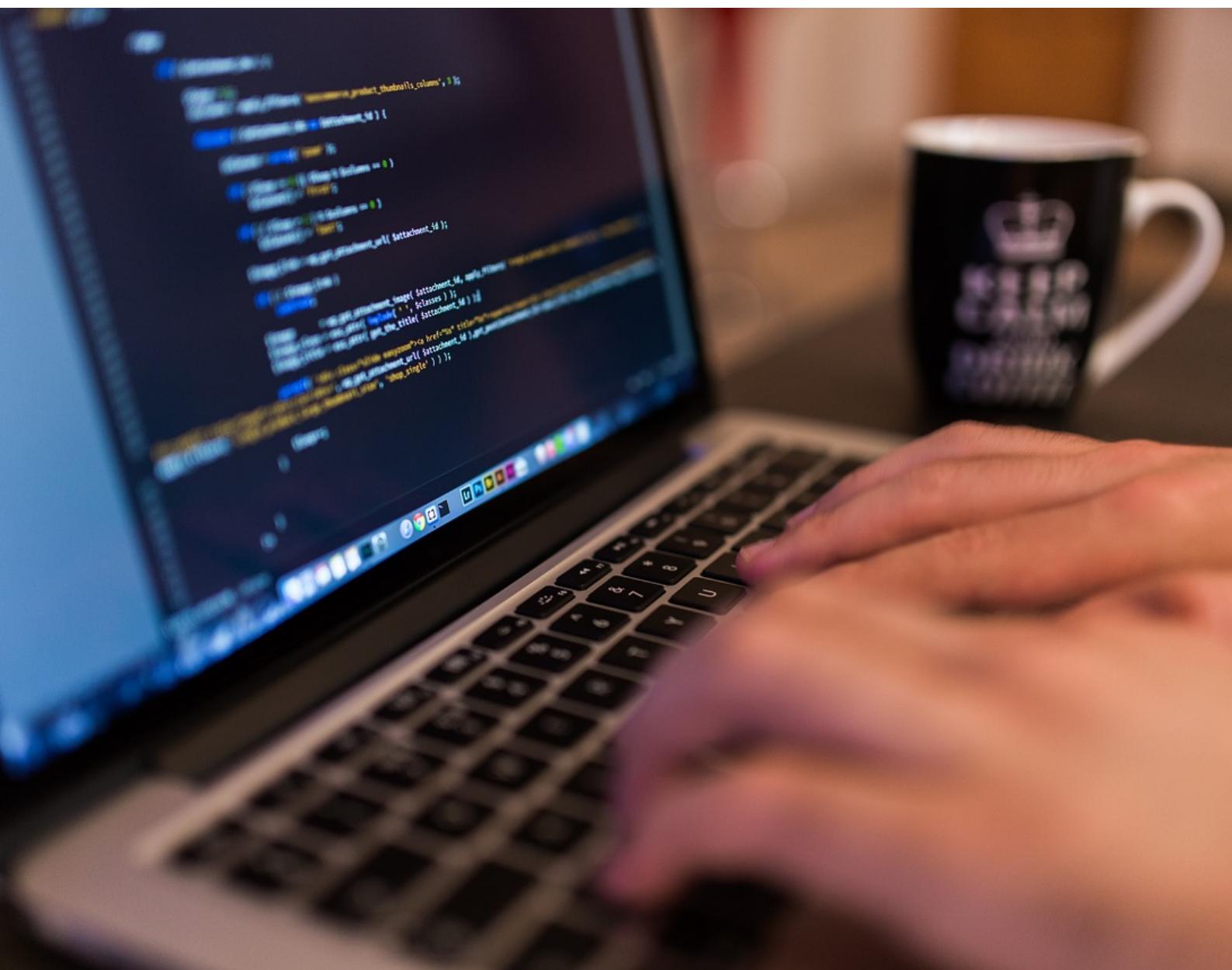


CSED232
Seungyong Lee
객체지향프로그래밍
49003157 최우석 (f2sound)
(한동대학교 학점교류)

Object_Oriented Programming :

Assignment 4



Honor Code

“나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다”

*보고서의 예시사진들은 모두 디버깅을 위한 코드가 포함되어있습니다. 제출한 파일에는 모두 제거하였습니다.

*macOS에서 과제를 진행하였는데 윈도우에서 컴파일 시 위젯사이즈 등이 매칭되지 않는 문제가 발생되어, 윈도우에서 잘 작동되도록 위젯사이즈, 글씨체 등을 수정하여 제출하였습니다. 이 부분은 [참고사항]에 기술하였습니다(p.13).

Problem 1: 구현완료

```
>> seq::m_notes size : 1
>> seq::put() 작동!
>> seq::m_notes size : 2
>> seq::put() 작동!
>> seq::m_notes size : 3
>> seq::put() 작동!
>> seq::m_notes size : 4
>> seq::put() 작동!
>> seq::m_notes size : 5
PASS  : TestIo::ReadWriteSong()
PASS  : TestIo::cleanupTestCase()
Totals: 4 passed, 0 failed, 0 skipped, 0 blacklisted, 2ms
***** Finished testing of TestIo *****
```

<test_io output>

```
bool Pitch::operator==(const Pitch &other) const
{
    return m_pitch_class == other.m_pitch_class &&
           m_octave == other.m_octave;
}
bool Note::operator==(const Note &other) const
{
    return m_start == other.m_start &&           △ comp
           m_duration == other.m_duration &&       △ comp
           m_pitch == other.m_pitch;
}
bool ISeq::operator== (const ISeq& other) const
{
    //std::cout<<" >> 시퀀스 == 연산자 작동!"<<std::endl;
    if(m_notes.size()!=other.getSize()){      △ comparison of integers of different signs: 'std::vector'
        //    std::cout<<" >> return false; 두 시퀀스의 size가 다릅니다."<<m_notes.size()<<"/"<<other.getSize()<<")
        return false;
    }

    else{
        for(int i=0; i<m_notes.size(); i++){      △ comparison of integers of different signs: 'int' and 'size_type'
            if(m_notes[i]->GetStart() != other.m_notes[i]->GetStart() ||          △ comparing floating point with
               m_notes[i]->GetDuration() != other.m_notes[i]->GetDuration() ||      △ comparing floating point with
               m_notes[i]->GetPitch().GetPitchClass() != other.m_notes[i]->GetPitch().GetPitchClass() ||  △ impl
               m_notes[i]->GetPitch().GetOctave() != other.m_notes[i]->GetPitch().GetOctave()){      △ implementation
                //std::cout<<" >> return false; 두 시퀀스의 내용이 다릅니다."<<std::endl;
                return false;
            }
        }
    }
    return true;
}
```

<Q_COMPARE 을 위한 비교연산자들 구현>

Problem 2: 구현완료

app/soundeffect.h

```
#ifndef __AS4_SOUNDEFFECT_H_
#define __AS4_SOUNDEFFECT_H_ △ macro name is a reserved word in C/C++

#include <QWidget>
#include <QTimer>
#include <QSoundEffect>

class DefaultSoundEffect : public QSoundEffect
{
    Q_OBJECT
public:
    DefaultSoundEffect();
    virtual ~DefaultSoundEffect();
    virtual void play();
};

class AutoStopSoundEffect : public DefaultSoundEffect
{
    Q_OBJECT
public:
    AutoStopSoundEffect(int Stop_milisec);
    virtual ~AutoStopSoundEffect();
    virtual void play();

private:
    QTimer* timer;
    int Stop_milisec;
};

#endif
<app/soundeffect.h>
```

Melody 트랙과 Drum 트랙의 재생방식 차이는 Duration이다. Melody는 정해진 duration 동안만 .wav 파일을 재생해야 하고, Drum은 duration과 상관 없이 .wav 파일을 끝날 때까지 재생해야 한다. 나는 그 차이를 QSoundEffect를 상속하는 두 클래스로 두었다.

DefaultSoundEffect는 저장된 .wav파일을 끝까지 재생하는 클래스이고, AutoStopSoundEffect는 duration 만큼 재생한 뒤 재생을 멈춘다. 이러한 구현을 가능하게 하기 위해, AutoStopSoundEffect는 QTimer 클래스를 이용하며, play() 내에서 정해진 duration이 지나면 재생하던 음원을 stop() 하는 시그널/슬롯을 추가하였다.

즉 DefaultSoundEffect는 QT에서 제공하는 QSoundEffect 클래스와 기능적으로 완전히 동일하지만, play() 메소드를 virtual로 선언할 필요가 있었기에 DefaultSoundEffect를 추가하여 play() 메소드를 virtual로 상속하였다. 이로 인해 이후 다른 악기의 트랙을 구현할 때에, Melody처럼 정

해진 duration 내에 재생을 멈춰야 하는 트랙은 AutoStopSoundEffect, Drum처럼 duration과 상관 없이 음원을 끝까지 재생해야 하는 트랙은 DefaultSoundEffect로 저장한 뒤 동일하게 모두 play() 메소드로 음원을 재생할 수 있게 되었다.

app/tracks.h

```
class DefaultTrack : QWidget
{
    Q_OBJECT
public:
    explicit DefaultTrack(QWidget *parent = nullptr);
    virtual ~DefaultTrack();
    virtual void setPath(as4::model::ISeq* seq);
    virtual void setVector();
    void play();

protected slots:
    void playUnit();

protected:
    std::string file;
    std::vector< std::vector<DefaultSoundEffect*>> Sound;
    int increasingIdx;
    QTimer* timer;
    as4::model::TimeConfig timeConfig;
};

class DrumTrack : public DefaultTrack
{
public:
    DrumTrack();
    ~DrumTrack();
    virtual void setPath(as4::model::ISeq* seq);
    virtual void setVector();

};

<app/tracks.h>
```

DefaultTrack 클래스는 현재 widget에 입력된 노트들을 재생 가능한 형태로 변환하는 모든 과정을 포함한다. 2차원 형태로 선언된 벡터는 각 노트들의 SoundEffect 객체를 저장하는데, 2차원 벡터의 행은 각 unit의 단위, 즉 시간을 나타내고, 열은 해당 시간에 재생되어야 할 SoundEffect 객체들을 저장한다. 그러므로 Sound[0] 는 0번째 unit에 재생되어야 할 SoundEffect 객체들의 집

합인 벡터를 저장하고 있고, Sound[4] 는 4번째 unit에 재생되어야 할 SoundEffect 객체들의 집합인 벡터를 저장하고 있다. 즉, QTimer를 이용하여 한 유닛만큼의 시간간격을 기준으로

/ Sound[0] play /, / Sound[1] play /, / Sound[2] play /, ...

를 진행하며 트랙을 재생한다.

setPath 메소드는 현재 화면상에 보이는 widget에 저장된 모든 노트들을 start, duration, pitch, resource path 등의 정보가 담긴 형식화된 .txt 파일로 저장한다. 이 때, melody 는 리소스 파일의 audio/melody/ … 이고, drum 은 audio/drum/ … 이기에 이 둘을 구분하여 입력하기 위해 Track 을 상속으로 나타내었고, setPath() 메소드를 virtual로 선언하였다. DefaultTrack은 melody 트랙을 의미한다. 뿐만 아니라, 실제 재생 객체 생성시에도 melody 와 drum에 따라 DefaultSoundEffect / AutoStopSoundEffect 로 구분하여 생성해야 하므로, 이러한 역할을 하 는 setVector() 메소드 역시 virtual로 선언하였다.

app/playsong.h

앞서 본 track 클래스를 멤버변수로 가지고, mainwindow에서 play 버튼을 누르는 front 적인 움직임과, 실제 음이 재생되는 back 적인 연산을 이어주는 클래스이다. SetAndPlay는 mainwindow 의 play버튼이 click() 시그널을 보냈을 때 실행되며, SetAndPlay 를 통해 멤버변수인 Track 내에서 실제 SoundEffect 객체들이 생성되고 재생된다. 이 때 QTabWidget* 을 인자로 받게 되는데, 그 이유는 현재 유저가 melody 위젯과 drum 위젯 중 무엇을 선택한 채 play 버튼을 눌렀는지 구분하기 위해서다. 즉, 현재 탭 위젯이 melody이면 Track 은 DefaultTrack으로 동적할당되고, 현재 탭 위젯이 drum이면 Track 은 DrumTrack으로 동적할당되는 식이다.

```
class PlaySong
{
public:
    PlaySong();
    ~PlaySong();
    void SetAndPlay(Song* FullSong, QTabWidget* tabWidget);

private:
    stack<DefaultTrack*> TrackMemory;
    DefaultTrack* Track;
};
```

<app/playsong.h>

특히, 매 play 마다 동적생성되는 DefaultTrack의 메모리를 스택에 저장해둠으로써, play 버튼을 누를 때 우선 스택을 참조하여 이전의 play가 아직 재생중인지를 판단하며, 재생중이라면 현재의 play를 pass 하고, 재생중이지 않다면 스택에 저장된 메모리를 delete하고 현재 play를 진행한다.

이후 최후에 남은 DefaultTrack 메모리는 프로그램이 종료될 때 PlaySong의 소멸자를 통해 제거된다.

```
>> PLAY BUTTON click!
> DefaultTrack destructor!
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/11_4.wav
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/4_4.wav
>> SetAndPlay() exit!
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 1
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 4
>> PLAY BUTTON click!
>> 이미 재생중입니다.
>> PLAY BUTTON click!
>> 이미 재생중입니다.
>> 벡터 idx 40 도달
>> PLAY BUTTON click!
> DefaultTrack destructor!
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/11_4.wav
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/4_4.wav
>> SetAndPlay() exit!
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 1
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 4
> DefaultTrack destructor!
```

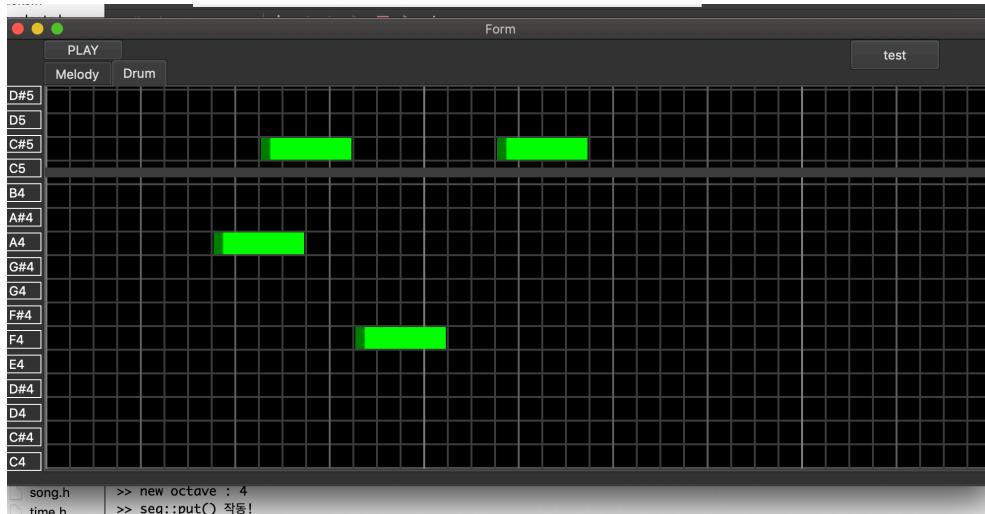
<Play Log for debugging>

위 로그에 보이는 바와 같이, 이미 재생중일 때에는 Play button 을 클릭해도 play를 위한 DefaultTrack이 생성되지 않으며, 재생이 끝난 이후라면(>> 벡터 idx 40 도달) Play button 을 클릭할 시 이전의 DefaultTrack의 메모리를 해제하고 새로운 DefaultTrack 이 생성되도록 구현했다. (>> play! 는 각 SoundEffect 객체의 play() 가 작동되었음을 의미한다.)

```
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/9_4.wav
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/6_4.wav
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/9_4.wav
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/7_4.wav
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/0_5.wav
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/4_4.wav
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/0_5.wav
>> 벡터 안에서 입력된 소스경로 : file:///audio/melody/audio/melody/9_4.wav
>> 시계가 동작한다네
>> SetAndPlay() exit!
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 4
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 9
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 12
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 18
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 20
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 23
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 24
AutoStopSoundEffect.play()!
>> play!
>> 현재 벡터 idx : 26
>> 벡터 idx 40 도달
```

<Play Log for debugging>

Problem 3: 구현완료



```
song.h      >> new octave : 4
time.h       >> seq::put() 작동!
util         >> seq::m_notes size : 5
ces          >> 슬롯 실행 : RemoveNote
sers/user/Library >> drum size : 4
io           >> start : 8
seqio.cpp    >> target start : 8
songio.cpp   >> target pitch : 9
model        >> target octave : 4
model        >> ISeq target 제거완료

using namespace as4::model;

namespace Ui {
class mainwindow;
}

class mainwindow : public QWidget
{
    Q_OBJECT
public:
    explicit mainwindow(QWidget *parent = nullptr);
    ~mainwindow(); △ '~mainwindow' overrides a dest

protected:
    void mousePressEvent(QMouseEvent* event) override;
    void mouseMoveEvent(QMouseEvent* event) override;

signals:
    void PutInSequence(int start, int duration, int pitch, int octave);
    void PutInSequenceDrum(int start, int duration, int pitch, int octave);
    void RemoveFromSequenceMelody(int start, int duration, int pitch, int octave);
    void RemoveFromSequenceDrum(int start, int duration, int pitch, int octave);

protected slots:
    void PutMelodyNote(QPoint pos, ClickableLabel* tab);
    void PutDrumNote(QPoint pos, ClickableLabel* tab);
    void RemoveMelodyNote(QPoint pos, ClickableLabel* tab);
    void RemoveDrumNote(QPoint pos, ClickableLabel* tab);

private slots:
    void on_PlayButton_clicked();
    void on_pushButton_clicked();

private:
    Ui::mainwindow *ui;
    ClickableLabel* MelodyRoll;
    ClickableLabel* DrumRoll;
    QVector<VisualNote*> melody_notes;
    QVector<VisualNote*> drum_notes;
    Song *FullSong;
    PlaySong Player;
};

<app/mainwindow.h>
```

a. Play

PLAY 버튼을 누르면 on_PlayButton_clicked() 슬롯이 실행되고, 이 슬롯은 Player.SetAndPlay() 메소드를 실행시킨다. 버튼은 Design Form 으로 구현하였다.

b. 음계 표시

세로 칸을 기준으로 각 음계를 Text로 나타낸다. Design Form 으로 구현하였다.

c, d, e. Note

유저가 하나의 노트를 왼쪽 마우스 클릭으로 찍으면, 현재 위젯에 따라 PutMelodyNote, PutDrumNote 슬롯 중 하나가 실행된다. 우선, 이러한 행위의 구현을 위해 tabWidget에 클릭 시 그널이 구현되어있어야 했다. 그래서 clickablelabel 클래스를 만들었다.

```
namespace Ui{
class ClickableLabel;
}

class ClickableLabel : public QLabel
{
    Q_OBJECT

public:
    explicit ClickableLabel(QWidget* parent = Q_NULLPTR, Qt::WindowFlags f = Qt::WindowFlags());
    ~ClickableLabel();

protected:
    void mousePressEvent(QMouseEvent* event);

signals:
    void Put(QPoint pos, ClickableLabel* );
    void Remove(QPoint pos, ClickableLabel* );
};

<app/clickablelabel.h>
```

clickablelabel 은 피아노를 나타내는 이미지를 보여줘야 한다. 그렇기 때문에 QPixmap을 보여줄 수 있는 QLabel을 상속하도록 했으며, 마우스의 왼쪽클릭, 오른쪽클릭에 따라 각각 Put, Remove 시그널을 보내도록 구현했다.

이 클릭 가능한 레이블은 mainwindow에 Design Form 으로 구현된 탭 위젯의 각각 페이지마다 보여야 하기에, 프로그램에서 이 클릭 가능한 레이블은 총 두 개가 생성된다(Melody tab, Drum tab). 이러한 Qt 상속은 mainwindow의 생성자에서 이뤄지도록 구현했다.

```
mainwindow::mainwindow(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::mainwindow)
{
    ui->setupUi(this);
    MelodyRoll = new ClickableLabel(ui->tab);
    DrumRoll = new ClickableLabel(ui->tab_2);
    FullSong = new Song;
    connect(MelodyRoll, SIGNAL(Put(QPoint, ClickableLabel*)), this, SLOT(PutMelodyNote(QPoint, ClickableLabel*)));
    connect(DrumRoll, SIGNAL(Put(QPoint, ClickableLabel*)), this, SLOT(PutDrumNote(QPoint, ClickableLabel*)));
    connect(MelodyRoll, SIGNAL(Remove(QPoint, ClickableLabel*)), this, SLOT(RemoveMelodyNote(QPoint, ClickableLabel*)));
    connect(DrumRoll, SIGNAL(Remove(QPoint, ClickableLabel*)), this, SLOT(RemoveDrumNote(QPoint, ClickableLabel*)));
}
```

<app/mainwindow.cpp ; constructor>

즉, 클릭레이블의 Put, Remove 시그널은 각각 PutMelodyNote, PutDrumNote, RemoveMelodyNote, RemoveDrumNote 슬롯과 연결되어 있다. 이 connect 도 mainwindow 의 생성자에 선언되어있다.

하나의 노트 이미지를 나타내는 클래스는 visualnote.h 에 선언되어있다.

```
namespace Ui{
class VisualNote;
}

class VisualNote : public QLabel
{
    Q_OBJECT

public:
    explicit VisualNote(QWidget* parent = Q_NULLPTR, Qt::WindowFlags f = Qt::WindowFlags());
    ~VisualNote();
    int getX() const;
    int getY() const;
    void set(QPoint pos);
    bool find(int x, int y);

private:
    int x;
    int y;
};

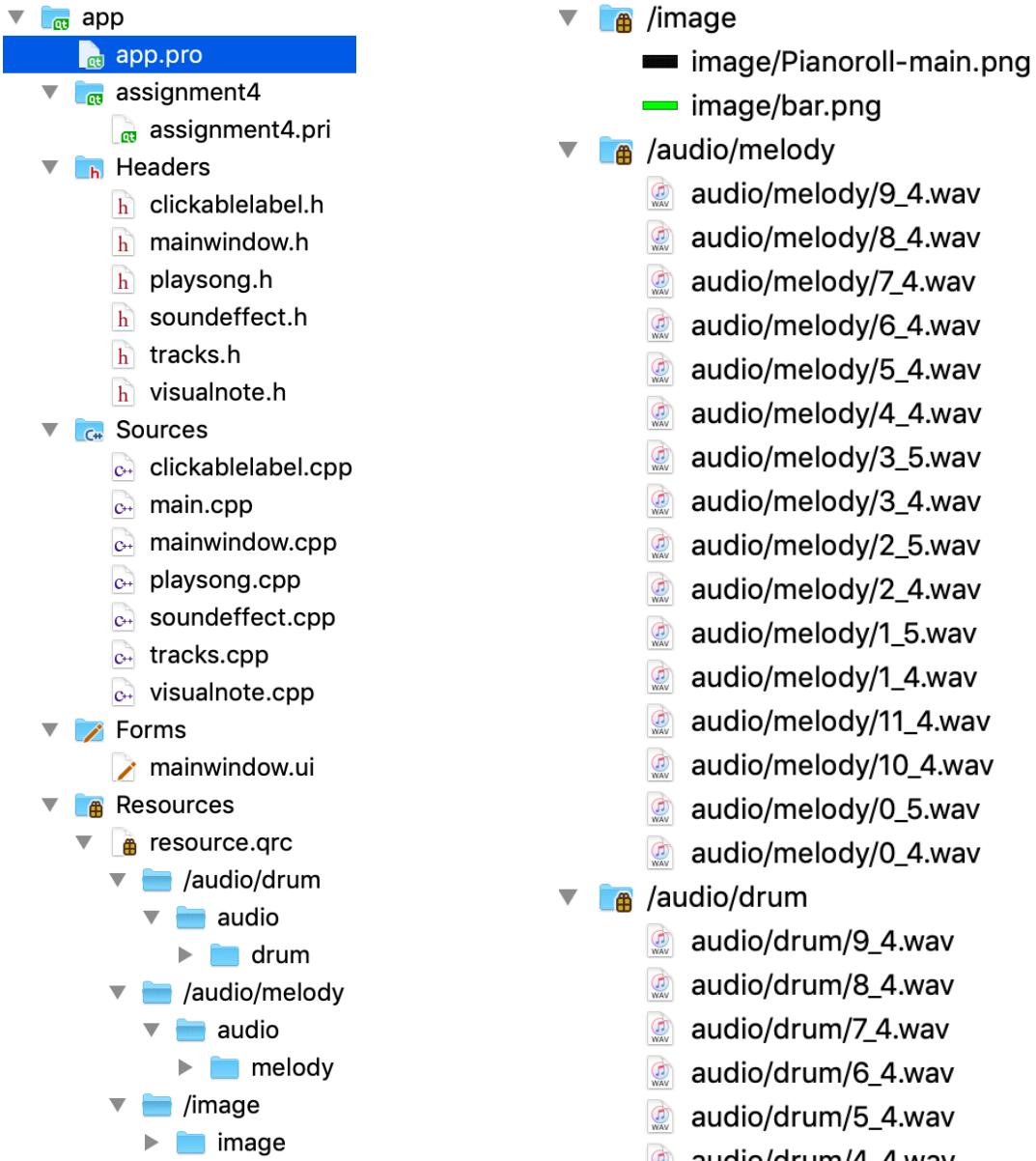
<app/visualnote.h>
```

역시 pixmap을 이용하기 위해 QLabel을 상속하며, 그 이외에 현재 노트의 위치를 나타내는 x 와 y 값을 정보로 가지고 있고 이들을 다루는 메소드들을 구현하였다. 이 노트들은 모두 mainwindow의 멤버변수로 선언된 QVector<VisualNote*> 안에서 저장되고 삭제됨으로써 GUI 에서도 나타나고 삭제된다. 특히, 이 노트 객체들은 모두 현재의 레이블, 즉 ClickableLabel 을 QT 상속하게 됨으로써 멜로디와 드럼 위젯에 각각 독립적으로 생기고 제거된다.

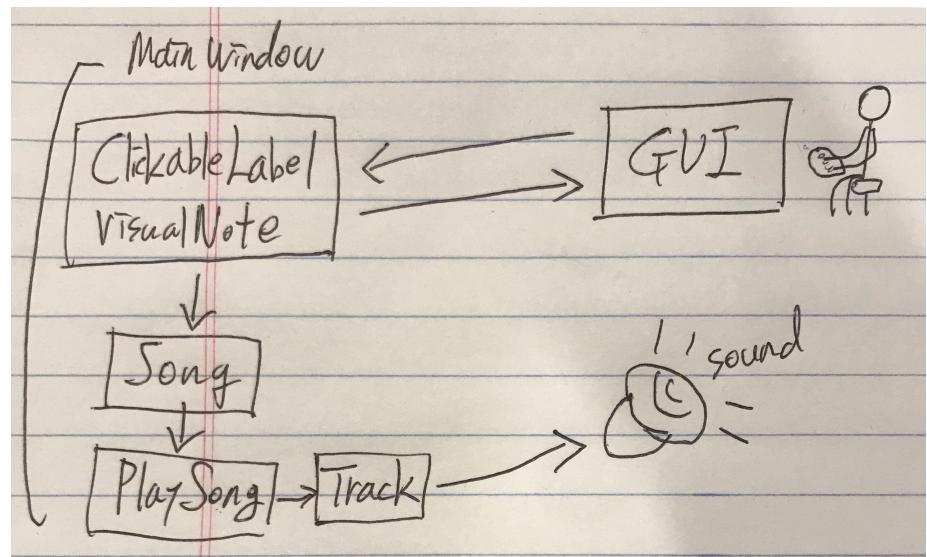
그 외

✓ app의 디렉토리 구조 및 리소스파일 .qrc

내가 정의한 파일은 모두 app에 있다. as4 디렉토리에는 몇몇 코드만 추가했을 뿐(주석처리) 파일을 추가하거나 제거한 것은 없으며, test_io, test_model 또한 마찬가지다.



✓mainwindow 내에서의 상호작용을 그림으로 나타내면 아래와 같다(..)

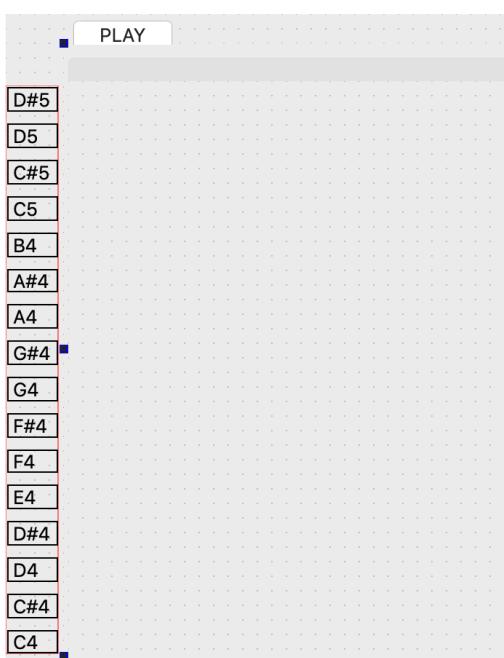


참고사항

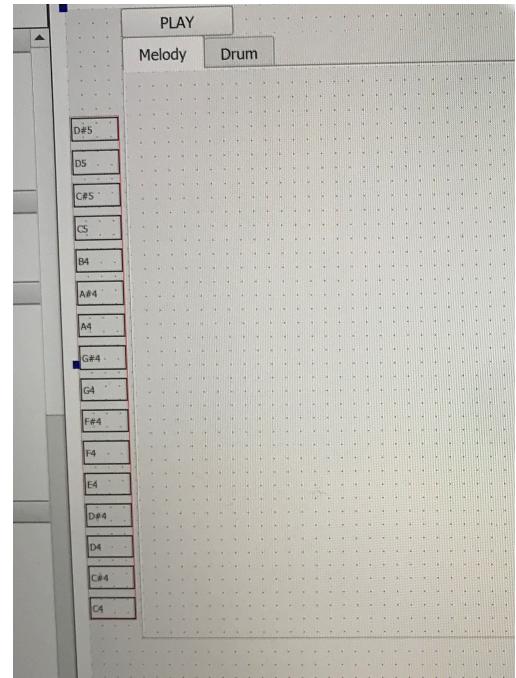
- ✓ 개발환경은 macOS였는데, 윈도우에서 컴파일 시 윈도우와의 글씨체 등의 호환성 문제로 좌측의 음표를 나타내는 스케일바와 실제 피아노롤이 맞지 않거나, 피아노를 최하단 일부(C4)가 잘려나가는 현상이 있었다. 최종적으로 윈도우에서 프로그램이 정상적으로 출력되도록 음표바와 탭위젯의 사이즈를 조절하였다. 밑의 사진은 그렇게 수정하여 윈도우에서 실행한 화면이다(맥에서의 실행화면은 Problem 3, p.8에 있다).



수정사항으로는, qt designer form에서 좌측 스케일바의 글씨체를 축소하였고, 스케일바를 나타내는 위젯의 사이즈를 조정하였으며, 탭위젯 사이즈를 늘렸다.



<맥 버전 Qt designer form>



<윈도우버전 Qt designer form>

느낀점

개인적인 취미로 DAW 프로그램에 익숙한 나에게, GUI 구현이 피아노롤임을 알았을 땐 흥미가 마구마구 솟구쳤다. 하지만 처음 1주동안은 QT의 개념을 잘 이해하지 못해 아무것도 손을 대지 못했다. 매번 과제를 해야겠다고 앉아서는 혼자 꿩끙앓며 고민만 하다가 결국 아무 것도 못 하고 잠에 들었을 때는 정말 포기해야겠다는 생각을 수도 없이 했었다. 하지만 QT라고 해서 뭔가 거창한 무언가가 있는게 아니라 지금까지 배워왔던 클래스와 다를 게 없고, 모든 QT의 라이브러리는 상속을 통해 내가 직접 다른 기능을 덧붙이거나 하는 형태로 사용할 수 있음을 깨닫고는, 본격적으로 과제가 진행되기 시작했다. 그 이후로는 QT Document를 뒤져보는 것도 재미있어졌고 조금 더 욕심을 부려 다른 기능들도 추가해보고 싶었다. 처음에는 굉장히 스트레스를 많이 받았지만 결국 해낸 것 같아 굉장히 보람찬 과제였다.

다른 한 편으로는, 내가 이제껏 사용해왔던 DAW 프로그램은 도대체 어떻게 프로그래머를 갈아넣었기에 그토록 수많은 기능을 담고 있는지 감이 오질 않았으며, 경이로움을 느꼈다. 실제 상용화되어있는 DAW 프로그램에서는 하나의 트랙을 표현하기 위해 어떤 자료구조를 사용하고 있는지가 제일 궁금했다. 내가 사용하는 Logic Pro X 프로그램의 경우 $\frac{1}{240}$ 유닛까지 표현이 가능한데, 이러한 트랙을 내가 진행한 이번 과제처럼 배열로 구현하는 것은 매우 비효율적이고, 특히 수많은 트랙이 모두 이러한 구조로 구현되는 것은 불가능할 것이다. 이러한 궁금증을 품고 앞으로 더욱 열심히 공부를 하며 호기심을 해결해나가는, 즐거운 학부생활을 할 생각에 벌써 가슴이 두근두근거린다.