2021 컴퓨터공학 설계 및 실험1 기말 프로젝트: visible piano

분반 : 3반　　학번 : 20200118　　이름 : 조창희

1. 프로젝트 목표 및 계획

나는 openframwork를 기반으로 한 두 프로젝트 중 waterfall 실습의 연장선으로, 프레임마다 움직이는 선을 이용하여 visible piano(보이는 피아노) 프로젝트를 제작할 예정이다.

visible piano 프로젝트는 음악을 꼭 청각으로 감상해야 하는가라는 의문으로부터 시작한다. 보통 음악을 감상하기 위해서는 청각을 필수적으로 요구한다. 하지만 이는 어떠한 이유로 청각을 사용할 수 없는 사람에게는 치명적으로 작용한다. 청각으로 음악을 듣는 것과 같을 수는 없더라도, 음의 높낮이를 시각화한다면 굳이 청각을 사용하지 않더라도 음악을 감상할 수 있을 것이다. 그뿐만 아니라, 계이름을 제대로 숙지하지 못하는 아이들에게 음을 시각화해서 보여준다면 더 쉽게 인지시킬 수 있을 것이다. 따라서 '음악의 시각화'를 프로젝트 목표로 삼아, 음악 중에서도 비교적 음의 구분이 명확한 '피아노'를 음악의 악기로 선정하여 구현해보고자 한다. 사용 프로그램은 ios 기반 앱인 GarageBand와 실습 당시 사용하였던 Openframwork이다. GarageBand를 사용하여 음을 구현하고, 이를 이용하여 Oenframwark를 통해 시각적인 이미지를 구현할 예정이다. 기본적인 Draw()를 통해 전반적인 피아노의 모습을 그림으로 구현한 뒤, 각각의 지점들을 설정하여 영역을 구분하고, waterfall 방식을 이용하여 흐르는 모양으로 음의 높낮이를 구현할 예정이다.

2. 설계 목표 세부화

2-1 피아노 악기 모양으로 디자인한 화면/환경을 구성한다.

2-2 특정 key를 입력하면 key에 해당하는 음이 나오도록 하고, 해당 음 높낮이에 알맞은 line의 길이가 '흘러내리는 모양으로' 움직인다.

2-3 특정 key를 입력하면 기록이 시작되고, 기록이 완료되면 기록동안 입력한 key값을 .txt 파일 형태로 출력한다.

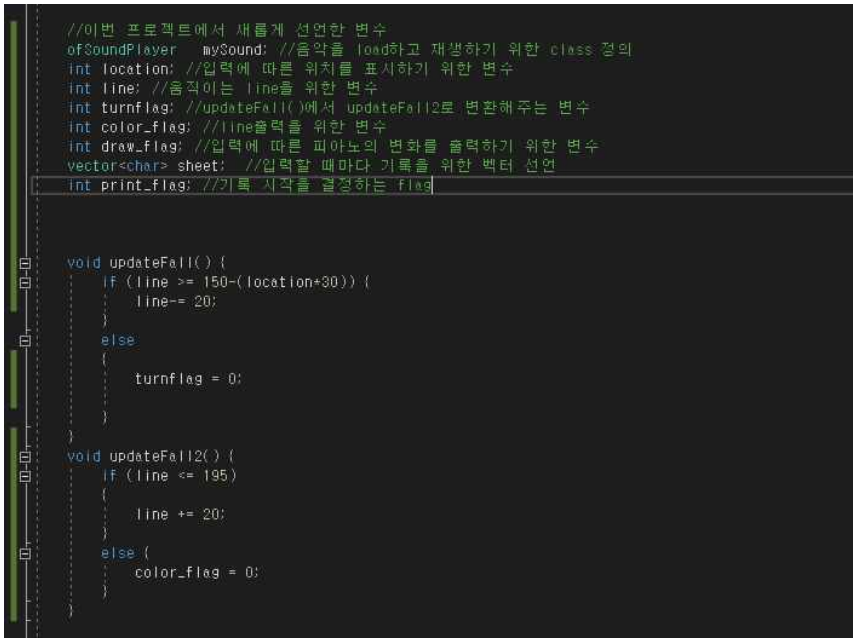
2-4 특정 key를 입력하면 메모리를 해제하고, 프로젝트를 종료한다.

3. 피아노 음 구현을 위한 어플 소개

Garageband: ios 기반 앱으로, 다양한 악기를 이용한 작곡이 가능하다. 해당 앱을 이용하여 피아노 음을 추출한다. 이 .mp3 파일들은 bin\data 파일에 저장한다.

4. 분석

4.1 변수, 함수, 자료구조 선언



```
//이번 프로젝트에서 새롭게 선언한 변수
ofSoundPlayer   mySound; //음악을 load하고 재생하기 위한 class 정의
int location; //입력에 따른 위치를 표시하기 위한 변수
int line; //움직이는 line을 위한 변수
int turnflag; //updateFall()에서 updateFall2로 변환해주는 변수
int color_flag; //line출력을 위한 변수
int draw_flag; //입력에 따른 피아노의 변화를 출력하기 위한 변수
vector<char> sheet;  //입력할 때마다 기록을 위한 벡터 선언
int print_flag; //기록 시작을 결정하는 flag



void updateFall() {
    if (line >= 150-(location*30)) {
        line-= 20;
    }
    else
    {
        turnflag = 0;

    }
}
void updateFall2() {
    if (line <= 195)
    {
        line += 20;
    }
    else {
        color_flag = 0;
    }
}
```

4-1-1 ofSoundPlayer mySound:

음악을 load하고 재생하기 위한 class이다.mySound.load()를 통해 음악 파일을 불러오고, mySound.play()를 통해 재생한다.

4-1-2 int location

피아노 음을 구현하기 위해 각 음에 알맞은 key를 입력할텐데, 각 음에 알맞은 위치를 표현하기 위한 변수이다.

4-1-3 int line

각 음마다 고유 높이까지 올라갔다가 내려가는 것을 구현하기 위한 변수이다.

4-1-4 int turnflag

line을 움직이는 것을 구현하기 위해 두 함수를 사용하는데, 함수별 전환을 표시하기 위한 flag이다.

4-1-5 int color_flag

line 변수가 표함된 직선을 색에 맞게 출력하는 것을 결정할 flag이다.

4-1-6 int draw_flag

draw() 구현 여부를 결정할 flag이다.

4-1-7 vector<char>sheet

입력한 피아노 음을 기록할 때 사용하는데, 기록 값을 저장하기 위한 1차원 벡터이다.

4-1-8 print_flag

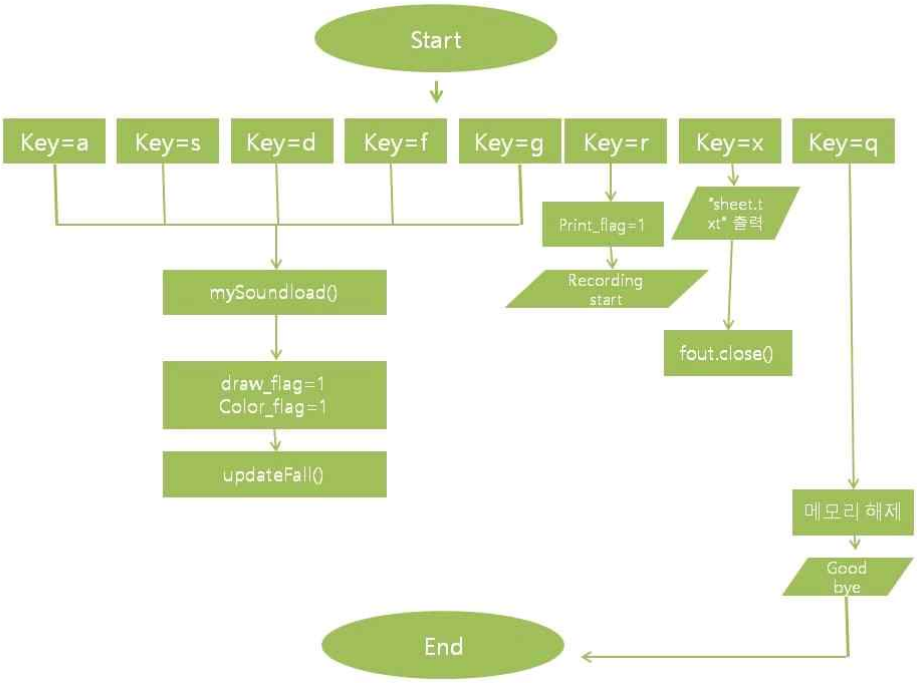
기록 여부를 결정하는 flag이다.

4-1-9 updateFall, updateFall2

오픈프레임웍스 수업을 기반으로 새롭게 정의한 함수로, 음별로 특정 높이까지 올라가고 내려가는 것을 구현할 함수들이다. 이 외에도, 해당 프로젝트를 구현하기 위해 오픈프레임웍스 내부적으로 주어진 setup(), draw(), keyPressed(int key)를 이용할 예정이다.

4.2 flow차트

visible piano가 구현되기 위한 간단한 플로우차트는 다음과 같다.



4.3 구현 설명

4-3-1 setup()

```
//--------------------------------------------------------------
void ofApp::setup(){
    ofSetFrameRate(15);
    ofBackground(255,255,255);
    ofSetLineWidth(4);

    draw_flag = 0; //draw, color flag 초기화
    color_flag = 0;

}
```

배경색을 지정하고, draw_flag와 color_flag을 초기화해준다.

4-3-2 keyPressed(int key)

4-3-2-1 key==q(Q)인 경우

```
void ofApp::keyPressed(int key){ //key 동작에 따른 변화

    if (key == 'q' || key == 'Q'){ //종료

        location = 0;
        line = 0;
        sheet.clear(); //할당된 메모리 초기화
        cout << "Good bye!" << endl;
        _Exit(0); //종료
    }
```

이 경우는 함수를 종료하기 위한 명령어이다. location 변수와 line 변수를 0으로 초기화해주고 벡터 sheet을 clear()로 메모리 할당을 해제해준다. 그 다음 Good bye를 출력하고 함수를 종료한다.

4-3-2-2 key==r(R)이거나 key==x(X)인 경우

```
    }
    if (key == 'r' || key == 'R') {
        print_flag = 1; //기록 시작
        cout << "recording start" << endl;
    }

    if (key == 'x' || key == 'X') { //기록을 print하고 sheet.txt 파일에 저장
        ofstream fout("sheet.txt");
        sheet.push_back('\n');
        for (int i = 0; i < sheet.size(); i++) {
            cout << sheet[i] << " ";
            fout.put(sheet[i]);
        }
        cout << endl << "recording finish : sheet.txt" << endl;
        fout.close();
    }
```

key==r일 때에는 record 시작, key==x일때는 record 종료를 뜻한다. record 시작일때는 시작한다는 메시지와 함께, print_flag를 1로 설정한다. 이 flag는 이후 key를 입력할 때마다 vector sheet에 push 여부를 결정해준다.

그리고 key==x일때는 지금까지의 기록을 print 및 함수에 입력을 하고, 입력 완료 메시지를 프린트한다. 여기서 프린트는, 영어 계이름을 따라 도레미파솔은 'CDEFG' 형식으로 기록된다. 각 record마다 구분을 짓기 위해 처음에 '\n'을 push해준다. 작업이 완료된 후, 파일을 close 한다.

4-3-2-3 key==a(A)인 경우

```
if (key == 'a'|| key =='A'){  //피아노 건반 '도(C)' 를 나타내는 key
    mySound.load("C.mp3"); //C.mp3 음악 파일 불러오기
    mySound.play(); //재생
    location = 0; //각 변수 초기화
    draw_flag = 1;
    color_flag = 1;
    turnflag = 1;
    line = 200;
    if (print_flag) {
        sheet.push_back('C');
    }

}
```

여기서, key가 a,s,d,f,g인 경우에는 각각 location 변수가 1씩 증가하는 것 말고는 알고리즘적인 차이가 없기 때문에, 대표적으로 a만 설명할 예정이다. 먼저 개러지밴드로 추출한 피아노 음을 mySound.load()를 통해 불러온다. 그리고 mySond.play()를 통해 재생한다. 그러한 다음 알맞은 location값을 부여하고, draw_flag와 color_flag를 1로 세팅하여 draw()가 작동할 수

있도록 한다. turnflag=1은 updateFall()를 이용함을 의미한다. 그리고 line이 흘러내리는 초기 위치를 200으로 초기화한 다음, print_flag가 1일 경우 sheet 벡터에 음에 해당하는 계이름을 push한다.

4-3-3 draw()

4-3-3-1 피아노 기본 세팅

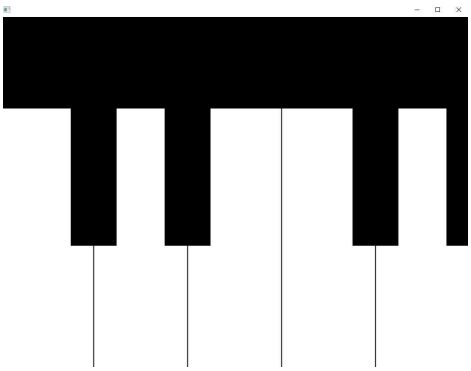
```
void ofApp::draw() {
    ofSetColor(0, 0, 0);

    //피아노 그리기
    ofDrawRectangle(0, 0, 1024, 200);
    ofSetLineWidth(2);
    for (int k = 0; k < 5; k++) {
        ofDrawLine(204.8*(k + 1), 200, 204.8*(k + 1), ofGetWidth());
        if (k != 2) {
            ofDrawRectangle(204.8*(k + 1) - 50, 200, 100, 200);
        }
    }
```

draw()도 부분으로 나눠서 살펴볼 예정이다. 먼저 위 코드는 draw_flag에 상관없이 기본적으로 실행되는 화면으로, 피아노의 기본적인 외형을 구현한다.

먼저, ofSetColor로 색을 0으로 설정한 뒤에, ofDrawRectangle(0,0,1024,200)으로 피아노 위 라인이 움직일 수 있는 검은 배경을 구성한다. 그 뒤에 피아노는 도~솔까지 5개의 음을 구현할 예정이므로, 가로값인 1024를 5로 나눈 204.8에 건반을 구분지을 선을 그려준다. 그리고 for루프를 이용하여 k가 증가할 때마다 건반을 계속해서 구분한다. 그리고 적절하게 샵/플랫을 나타내는 검은 건반을 구성한다. 미~파로 넘어가는 부분은 검은 건반이 없으므로 이를 주의하며 그려준다.

<실제 피아노 건반 모습>

<구현한 화면 모습>



4-3-3-2 피아노 그림자 처리

```
ofSetLineWidth(5);
if (draw_flag) { //key에 의해 피아노가 선택되었을때, 색 변환

    ofSetColor(230, 230, 230); //회색 지정
    ofDrawRectangle(205.8*location, 200, 204.8, ofGetWidth());
    if ((location == 0) || (location == 3)) { //건반 위치에 따른 다른 색 변환
        ofSetColor(0, 0, 0);
        ofDrawRectangle(204.8*(location + 1) - 50, 200, 100, 300);
    }
    if ((location == 1) || (location == 4)) {
        ofSetColor(0, 0, 0);
        ofDrawRectangle(204.8*(location + 1) - 50, 200, 100, 300);
        ofDrawRectangle(204.8*location - 50, 200, 100, 300);
    }
    if (location == 2) {
        ofSetColor(0, 0, 0);
        ofDrawRectangle(204.8*location - 50, 200, 100, 300);
    }
    draw_flag -= 0.5;
}
```

피아노 음을 입력할 때, 입력한 표시가 나지 않으면 자신이 무슨 음을 누르고 있는지 알지 못하여 불편할 수 있다. 따라서 입력한 key에 따라 해당 피아노에 음영처리를 하여 키를 눌렀음을 시각적으로 보인다. gray 색상으로 ofSetColor(230,230,230) 으로 지정한 뒤에, 받은 location값을 이용하여 해당 location을 gray처리한다.

그런데, 여기서 문제는 음영처리만 하면, 검은 건반까지 모두 음영처리가 되어서 보기에 어색하다. 따라서 누른 화면에 검은 건반은 다시 검은색으로 덧씌운다. 여기서 각 음에 따라 검은 건반의 개수가 다르기 때문에, 케이스를 구분하여 음영처리를 해준다. 예를 들어 '도'와 '파'은 오른쪽 검은 건반 1개, '레'와 '솔'는 양쪽 건은 건반 두 개, 미는 왼쪽 검은 건반 1개이다.

draw함수는 flag가 1인 한 프레임별로 계속 반복되므로, 음영이 계속 남아있게 된다. 건반은 한번 치는데 음영이 계속 남아있으면 어색하므로, 마지막에 draw_flag를 감소시켜 2프레임 뒤에 바로 사라지도록 한다.

4-3-3-3 waterfall 적용 :: 흘러내리는 물을 높낮이별로 구현

```
    if (color_flag) {

        int r = ofRandom(100, 255); //랜덤 색 정의
        int g = ofRandom(100, 255);
        int b = ofRandom(100, 255);


        ofSetColor(r, g, b);
        for (int k = 0; k < 5; k++) { //건반 중앙에 line 위치
            ofDrawLine((204.8*(location)+204.8*(location + 1)) / 2 - 20+(k+10), 200, (204.8*(location)+204.8*(location + 1)) / 2 - 20 + (k + 10), line);
        } //line 표현
        if (turnflag == 1) {
            updateFall();
        }
        else
            updateFall2();

    }
}
```

디자인적 요소를 위해 line 색은 ofRandom()을 이용한다. 범위는 100부터 255까지 잡아 비교적 밝은 색이 오도록 한다. 이는 검은 배경과 대조되어 강조되는 효과를 준다. 또한, 디자인적 요소를 위해 for loop을 이용하여 얇은 선 5개가 같은 움직임을 하여 단조로움을 피한다. 여기서 프레임별로 움직이는 line이 실습을 바탕으로 한 요소이다. 이 line이 움직이기 위해 updateFall, updateFall2를 호출한다.

4-3-4 updateFall(), updateFall2()

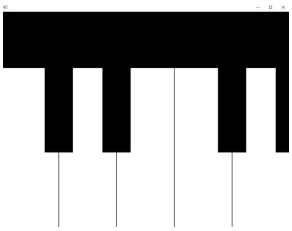
```
void updateFall() {
    if (line >= 150-(location*30)) {
        line-= 20;
    }
    else
    {
        turnflag = 0;

    }
}
void updateFall2() {
    if (line <= 195)
    {
        line += 20;
    }
    else {
        color_flag = 0;
    }
}

};
```

draw() 함수에서 호출하고 있는 함수이다. updateFall은 line이 감소하는, 눈으로 보면 마치 위로 올라가는 모양의 막대를 매 프레임별로 구현한다. 계속 올라가다가 일정 높이에서 멈춰야 하는데, 그 높이는 피아노의 건반 별로 다르다. 나는 최대 높이를 150으로 잡고, (최대 높이는 가장 낮게 올라감을 의미한다.) location에 따라 line의 최대 높이를 조절하며 마치 높은 음으로 갈수록 높이 올라가도록 구현한다.

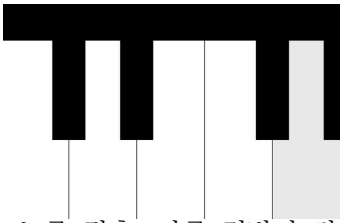
원하는 지점까지 올라간 뒤에는 turnflag를 0으로 하여 updateFall2 함수가 작동하도록 한다. 그리고 다시 line이 증가하여 원위치로 돌아오면, color_flag를 0으로 만들어 더 이상 위 두 함수가 작동하지 않도록 한다.

4.3 구현 이미지

4-3-1 기본 피아노 모습

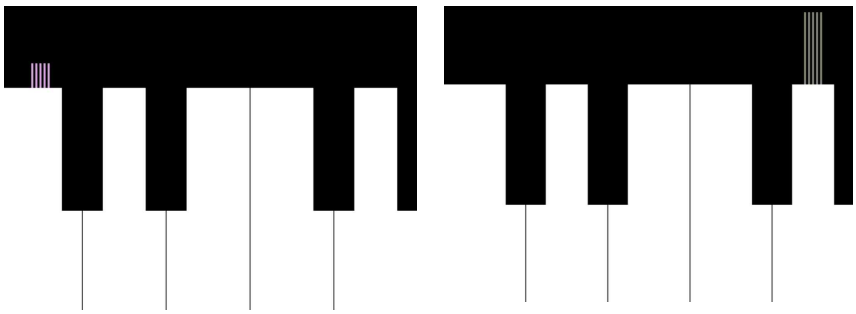


4-3-2 음영이 생기는 모습



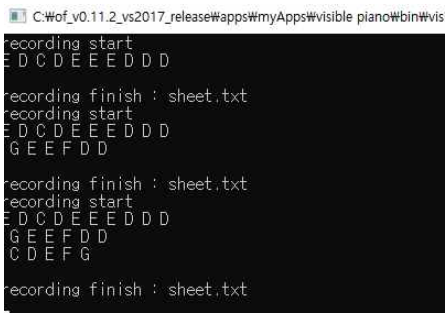
누른 직후, 다른 건반과 달리 음영이 생기는 모습을 확인할 수 있다.

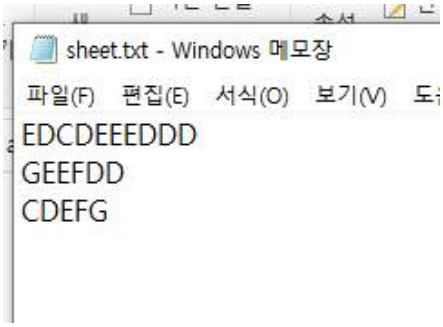
4-3-3 막대가 움직이는 모습



피아노 음계에 따라 서로 다른 높이로 움직임을 확인할 수 있다.

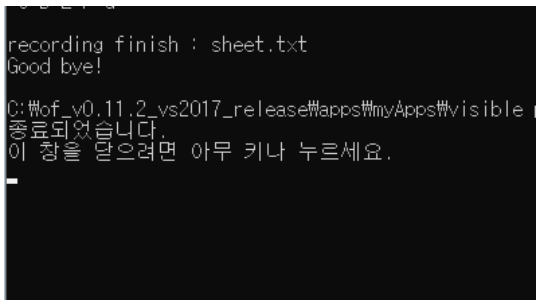
4-3-4 recording하고 기록하기





여러 번 recording할 때마다 줄바꿈이 되어 기록된다. 기록된 값이 텍스트 파일에 보내졌음을 확인할 수 있다.

4-3-5 종료



모든 메모리를 해제한 후 종료한다.

4.3 시간과 공간 복잡도

Big-O표기법을 이용하여 시간과 공간 복잡도를 평가해보겠다. 우선 공간복잡도는 동적할당된 sheet의 길이가 HEIGHT라고 가정할 때, O(HEIGHT)이다. 계속해서 push_back을 하며 공간을 늘려나가기 때문이다.

그리고 시간 복잡도도 마찬가지로 O(HEIGHT)이다. key==x인 경우 HEIGHT만큼 for문을 돌리면서 출력하고 입력하는 작업을 거치기 때문이다. 그 외 상황에서는 모두 O(1)이다.

5. 느낀 점 및 개선 사항

평소 예술 쪽에도 관심이 많아 코딩과 음악/디자인적인 측면을 결합하고 싶었는데, 이번 프로젝트를 계기로 구상할 수 있어서 좋은 경험이었다. 그리고 이러한 코딩과 예술의 결합을 하기에는 오픈프레임웍스가 최적의 오픈 소스 라이브러리라고 생각한다.

실습시간에 waterfall에서 키별로 색변화를 준 것에 더 나아가서 키별로 색변화와 음의 변화까지 준 점, 물이 떨어지는 현상을 이용하여 피아노의 음계를 표현했다는 점, 개러지밴드와 결합하여 만든 점은 좋았다고 생각한다. 다만 너무나도 음이 한정된 상황이라는 점, recording을 할 때 음만 기록되고 박자는 기록되지 않는다는 점은 아쉬운 측면이다. 이 프로젝트를 더 발전시키게 된다면, 조금 더 다양한 음을 지원하고, recording을 정교하게 해내어서 더욱더 발전된 '음악의 시각화'를 이루고 싶다. 가능하다면 피아노 말고 다른 악기에 대해서도 적용해보고 싶다.