컴퓨터공학설계및실험I 기말 프로젝트

<Amazing BeatFall!> 보고서

수학과 20161250 이지호

1. **Outline**

|  |
| --- |
| A picture containing object, clock, monitor, different  Description automatically generated  [Amazing BeatFall!]  <Amazing BeatFall!> 은 9-11주차 실습 <WaterFall>의 아이디어를 토대로 하여 만든 Beat-Playing Program이다. Generator, Pedal, Path를 원하는 위치에 설치하고 BeatBall을 생성하여 나만의 Beat가 연주되는 것을 청각뿐 아니라 시각적으로도 즐길 수 있다. 프로그램을 구성하는 주된 요소로는 공 낱개의 정보를 나타내는 BeatBall, BeatBall을 생성해주는 Generator, BeatBall을 튕겨내는 구조물 역할의 Pedal, Path가 있다. 사용자는 마우스 Drag&Drop으로 물체를 추가/삭제, 이동, 회전할 수 있고, BPM, Volume 조절 및 Save/Load 기능을 구현하여 더욱 다양하고 편리하게 프로그램을 즐길 수 있도록 하였다.  <Amazing BeatFall!>에는 총 5가지의 Pedal이 준비되어 있다. Kick, Snare, Hat, Ride, Crash Pedal이 있으며 각 Pedal마다 10가지의 Sound Sample중 하나씩을 랜덤하게 선택, 조합하여 프로그램이 시작된다. 원하는 대로 물체를 배치한 후 .abf 파일로 저장할 수 있으며, 마찬가지로 .abf 파일을 불러 저장되어 있던 설정을 불러올 수 있다.  <Amazing BeatFall!>은 C++와 OpenFrameWorks를 활용한 프로그램으로서 Mac OS X, Xcode 11 환경에서 프로그래밍되었으며, 아래에 본 프로그램을 구현하는 데 사용한 자료구조와 알고리즘을 소개한다. |

1. **자료구조**

|  |
| --- |
| 1. myBeat.h   A close up of a logo  Description automatically generated  [class myObj]  본 프로그램 상의 모든 물체(Generator, BeatBall, Pedal, Path)는 공통적으로 위치 정보를 갖고 있어야 한다. myObj라는 클래스를 선언하여, 각 물체들을 나타내는 클래스를 myObj의 상속 클래스로 만들어서 가장 기본적인 기능인 ‘위치 정보 갱신’을 편리하게 할 수 있도록 구현했다. 각 자식 클래스는 모두 myObj의 생성자를 따른다. bool move\_flag는 BeatBall을 제외한 나머지 세 개의 클래스를 위한 flag로, 사용자가 마우스로 물체를 옮기는 상태인지를 나타낸다.  A screenshot of a cell phone  Description automatically generatedA screenshot of a cell phone  Description automatically generated  [class Pedal : class myObj]  Pedal의 정보를 담은 클래스이다. 회전이 가능한 직선 모양의 물체이므로, 중심 좌표 float x, y 외에 float xinit, yinit, xfinish, yfinish, 그리고 float rotate\_degree를 멤버에 추가하였다. 또한, \_pedalType 을 선언하여 해당 객체가 어떤 종류(ex. Snare)의 Pedal인지 알 수 있도록 했다. 사용자가 마우스 클릭을 하였는지를 검사하는 flag들과, BeatBall과 충돌했는지를 검사하는 hit\_flag가 있다. 시각 효과를 위해 hit\_flag가 true로 set되면 flash() 함수에서 flash\_offset을 MAX\_FLASH\_OFFSET 으로 설정하여 순간적으로 Pedal의 투명도, 크기를 조절한다. ofApp::draw() 함수가 반복 호출됨에 따라 flash\_offset의 값은 점차 0으로 감소하고, 페달의 투명도와 크기는 다시 원래대로 돌아오게 된다.  Path 또한 실제로는 Pedal 클래스로 구현하는데, 거의 모든 기능을 공유하기 때문에 별도의 클래스로 분리하지 않았다. 다만 Pedal과 Path는 ofApp 실행 시 데이터를 따로 관리하고, Path는 \_pedalType과 flash()를 사용하지 않는다.  A screenshot of a cell phone  Description automatically generated  [class BeatBall : class myObj]  공 하나의 정보를 담고 있는 클래스이다. 중력이 있는 공간에서의 공의 낙하운동에 최대한 가깝게 구현하고자 velocity, acceleration x, y를 추가하였다.  A screenshot of a cell phone  Description automatically generated  [class Generator : class myObj]  BeatBall을 생성하는 클래스이다. vector<BeatBall>을 멤버로 갖고 있는데, 이는 각 Generator마다 자신의 위치에서 BeatBall을 생성해야 하기 때문이다. 대신 이 vector는 별도의 push, pop은 수행하지 않고, 고정된 사이즈 내에서 운용된다(ofApp::update() 참조). genBall() 함수를 호출하면 인덱스 번호를 인자로 전달하여 Generator 내의 ball[idx] 를 필드에 활성화시킨다.   1. ofApp.h   A screenshot of a cell phone  Description automatically generated  [프로그램 기본 구성요소]  프로그램의 뼈대가 되는 구성요소들이다. std::vector는 push와 pop이 매우 용이하고 메모리를 효율적으로 사용하면서도, 정적배열처럼 모든 원소를 매우 쉽고 빠르게 참조할 수 있기 때문에 이번 프로그램의 메인 자료구조로 채택했다. 각 물체들을 모두 vector로 선언하여 동적으로 데이터를 관리하였고, 사용자가 물체를 추가하면 push, 삭제하면 erase하여 메모리 공간을 절약한다. 위 코드에서 ofColor 클래스 vector genColor를 선언하였는데, 이는 Generator객체마다 색상을 다르게 지정하여 생성되는 공 또한 자신이 속한 Generator의 색상을 따르도록 하기 위함이다. Generator가 추가/삭제될 때마다 genColor 도 같이 push/pop하여 size를 동일하게 유지한다.  A screenshot of a cell phone  Description automatically generated  [openframeworks 전용 변수들]  ofColor, ofSound, ofTrueToneFont, ofRectangle, ofImage 변수들이다. 물체 종류 별로 색상, 음성, string 등을 저장하여 draw() 함수 등에서 편리하게 해당 물체의 정보를 확인할 수 있도록 구현했다. ofImage TrashsCan은 프로그램 실행 시 삭제 칸에 삽입할 휴지통 아이콘을 load할 변수이다.  A screenshot of a computer  Description automatically generated  [String Format]  프로그램의 현재 정보나, 파일을 save/load할 때 반복적으로 사용될 문구를 string 상수로 정의해놨다. alert는 ofApp Window 및 콘솔에 출력할 프로그램 확인 메시지로 특정 함수 실행마다 계속 갱신된다. status는 현재 ofApp의 frame rate, 그리고 사용자가 설정한 BPM을 나타내는 변수로 마찬가지로 계속 갱신된다. 그 외에는 모두 상수처럼 사용된다.  A screenshot of a cell phone  Description automatically generated  [Flags]  프로그램 실행 중에 사용될 flag들이다. BeatBall을 생성할 건지를 결정하는 start\_flag, 물체를 새로 생성하는 지를 검사하는 create\_flag, 그리고 파일이 가장 최근에 load된 시점으로부터 수정된 것이 있는지를 검사하는 edit\_flag가 있다. edit\_flag가 true로 활성화되어 있을 때만 save 기능이 활성화된다.  A screenshot of a cell phone  Description automatically generated  [File Save & Load]  파일을 저장하고 불러오는 함수이다. 사용자가 S키를 눌러 save()를 호출하면, 내부에서 setInfoString()을 실행하여 infoSaved에 현재 BeatFall의 정보를 string으로 변환, 저장한다. 그 후, processSaveFile()에서 파일 출력 스트림을 생성하고 디렉토리에 .abf파일(텍스트)로 저장한다.  마찬가지로, 사용자가 L키를 눌러 파일을 선택하면, 해당 파일명을 load()함수에 전달하여 현재 실행 중이던 BeatFall을 clear하고 해당 파일의 정보를 load한다.  A screenshot of a cell phone  Description automatically generated  [User-Defined Functions & Control Functions]  playSoundSample()은 Pedal의 hit\_flag가 활성화될 때마다 호출되어, 해당 Pedal의 \_pedalType을 읽어 그에 맞는 sound sample을 재생해준다. 결과적으로는 BeatBall이 Pedal에 부딪힐 때마다 그에 맞는 소리가 나게 되는 것이다. clearall()은 사용자가 C키를 눌러 직접 호출할 수도 있고, 혹은 L키를 눌러 파일을 load할 때에 자동으로 호출되어 실행된다. 현재 스크린 상의 모든 물체를 초기화(삭제)한다.  그 밖에 다른 ofApp 기본 제공 함수들은 아래에서 다루도록 한다. |

1. **알고리즘**

|  |
| --- |
| 1. myBeat.cpp   A screenshot of a cell phone  Description automatically generated  [myObj::myObj(float \_x, float \_y)]  부모 클래스의 생성자이다. Generator, BeatBall, Pedal 모두 이 생성자를 상속받아 생성된다.  A screenshot of a cell phone  Description automatically generated  [Pedal::Pedal(), set()]  Pedal 클래스의 생성자와 set() 함수이다. 생성자는 myObj 클래스를 물려받아 rotate\_deg와 pedalID를 추가로 설정할 수 있게 구현했다. set() 함수도 생성자와 거의 유사하게, 좌표와 회전 상태, 그리고 pedalID를 입력하면 그에 맞게 설정되도록 하였다.  A screenshot of a cell phone  Description automatically generated  [Pedal::flash(), getRadian()]  flash()는 Pedal이 BeatBall과 부딪혔을 때 flash\_offset에 비례하는 offset만큼 투명도와 크기가 변하도록 해주는 함수이다. hit\_flag가 활성화된 순간, flash\_offset은 MAX 값으로 set된다. 이후 ofApp::draw()함수에서 1/60초마다 반복 호출되며 flash\_offset의 값은 static float unit\_offset의 값만큼 감소한다. 0이 되면 Pedal의 투명도와 크기는 다시 원래대로 되돌아온다.  getRadian()은 degree 단위로 저장된 Pedal(혹은 path)의 회전 정보를 radian 단위로 변환해주는 함수이다.  A screenshot of a cell phone  Description automatically generated  [BeatBall::BeatBall(), set(), collide()]  BeatBall의 생성자와 set() 함수는 좌표 설정만 하면 되므로 간단하게 나타냈다.  A screenshot of a cell phone  Description automatically generatedA screenshot of a cell phone  Description automatically generated  collide() 는 BeatBall객체 하나가 필드 위의 모든 pedal, path와의 위치를 고려하여 BeatBall의 다음 frame에서의 운동상태를 갱신해주는 함수이다. 이를 위해 parameter로 vector 포인터를 전달하여, Pedal이나 Path의 정보가 (추가/삭제/이동 등의 이유로) 변하더라도 해당 vector에 직접 참조할 수 있도록 하였다.  collide() 함수 내부로 진입하기 위한 조건은 ‘BeatBall이 Pedal(혹은 path)와 충분히 가까운가?’이다. 이를 위해 최소 거리는 BeatBall의 반지름으로 설정했고, 거리를 계산하기 위해 이후에 설명할 getDistance() 함수가 이용된다. BeatBall이 해당 거리 내에 진입했다면, Pedal의 넓은 면에 충돌했는지 아니면 끄트머리에 충돌했는지를 확인한다. 이는 좀 더 사실적이고 다양한 BeatBall의 운동을 구현하기 위함이다. Pedal의 끝 점에서 충돌이 일어났다고 확인되면, BeatBall의 입사각(incidence)과 Pedal의 경사(angle)을 비교하여 ‘끄트머리 충돌’이 일어날 수 있는 조건인지를 확인한다. 만약 ‘끄트머리 충돌’이 일어난 것이라면, Pedal이 90도 회전되어 있는 상태라고 가정하고 반사각(reflection)을 계산한다. Pedal의 옆면은 넓은 면과 수직이기 때문이다.  반사각 계산이 끝나면, BeatBall의 충돌 직전 속력을 계산하고 감속 계수(SPRING)를 곱하여 충돌 이후의 속력을 설정한다. 가속도와 속도 설정을 마치면 함수를 종료한다.  A screenshot of a cell phone  Description automatically generated  [BeatBall::move(), getProjectionY(), getDistance()]  BeatBall의 움직임과 collision을 계산하기 위해 선언된 함수들이다. move()는 단위 frame마다 가속도만큼 속도를 증가시키고 속도만큼 x, y좌표를 증가시킨다. getProjectionY()는 BeatBall이 인자로 전달된 \_pedal에 y축 방향과 나란하게 다가갈 경우의 y좌표, 즉 pedal위에 생긴 BeatBall 그림자의 y좌표를 계산하는 함수이다. 이를 토대로 getDistance()에서 BeatBall과 해당 pedal간의 거리를 계산할 수 있다. BeatBall이 pedal 의 x좌표 범위 내에 있다면 pedal까지의 수직 거리를 return하고, 그렇지 않다면 pedal의 양 끝 점 중 더 가까운 곳까지의 거리를 return한다. 이 함수를 통해 collide()에서 BeatBall과 Pedal의 거리를 계산하여 이것이 최소 거리보다 작은지, 즉 충돌 조건을 만족하는지를 검사할 수 있다.  A screenshot of a cell phone  Description automatically generated  [Generator::Generator(), set(), genBall()]  Generator의 생성자는 다른 클래스들과 거의 비슷하지만, 생성 시 자동으로 vector<BeatBall> 멤버의 size를 MAX로 고정하는 것이 추가되었다. 이 vector size는 프로그램이 종료될 때까지 변하지 않는다.  genBall() 은 해당 Generator가 멤버로 갖고 있는 ball의 idx번째 원소를 activate하고 필드에 생성하여 내보내는 함수이다. ofApp::update()에서 설명하겠지만, 설정해둔 BPM에 맞게 idx를 1씩 증가시키며 genBall() 함수가 호출된다. idx가 MAX가 되면 다시 0으로 설정하여, 정해진 size 내의 원소들만 활용한다. 이는 볼링장에서 볼링공을 굴리고 나면(Generate & Launch), 레인 위에서 움직이다가(move & collide) 끝에 도달하면(deactivate) 바닥 밑 레일을 통해 다시 출발점으로 돌아오는 것과 매우 유사한 알고리즘이다. 볼링공의 개수는 정해져 있지만, MAX를 적당한 값으로 설정하면 사용자가 봤을 때 마치 볼링공이 끊임없이 생성되는 것처럼, 즉 BeatBall이 무한히 생성되는 것처럼 구현할 수 있는 것이다.   1. ofApp.cpp   A screenshot of a cell phone  Description automatically generated  [ofApp::setup()]  배경의 모든 outline을 setup하는 함수이다. 본 프로그램에서는 배경 및 각 물체들의 color, Pedal들의 sound sample, 텍스트를 draw할 때 사용될 font 등등 전반적인 요소들을 알맞게 설정한다. 또한, 화면 오른쪽에 나타낼 side menubar나 왼쪽 아래에 표시되는 status monitor를 그리기 위해 ofRectangle 변수들의 위치와 크기를 설정해준다.  Sound setup에서, load할 sound sample file 이름을 랜덤하게 지정하는데, 본 프로젝트에 포함된 모든 sound sample file의 이름은 “pedal type” + “#” + “.wav”의 형식으로 되어있다(ex. Hat4.wav). 각각 1에서 10번까지 10개의 sample이 있으므로, 랜덤하게 하나를 골라 해당 sample을 ofSoundPlayer에 load한다. 사용자는 프로그램을 실행할 때마다 색다른 조합의 sound sample을 체험할 수 있다.    A screenshot of a cell phone  Description automatically generated  [ofApp::update()]  frame rate에 따라 반복 호출되는 갱신 함수이다. fps를 읽어 왼쪽 아래에 텍스트로 보이도록 한다. 또한, 사용자가 설정한 BPM대로 BeatBall을 생성하기 위해 자체적으로 clock 기능을 수행하도록 코드를 짰다. BPM이 설정되면 BeatBall이 생성될 주기(period)가 정해지고, clock을 0으로 set한다. update() 함수는 1/60초 (정확히는 1/framerate)마다 호출되므로, 단위 시간은 1/60(초)로 설정하여 시간을 계산한다. 한 주기만큼의 시간이 흐르면, 모든 Generator의 genBall 함수를 호출하여 공을 생성하고, idx를 한 칸 뒤로 이동시킨다. 위에서 설명한 대로, 주기마다 1번 볼링공, 2번 볼링공, …, n번 볼링공을 출발시키다가 idx가 MAX에 도달하면 다시 1번 볼링공을 출발점에 set하고 launch하는 알고리즘이다. 이로 인해 정적 할당을 했음에도 불구하고 무한히 연속적인 BeatBall 생성을 구현할 수 있는 것이다.  A screenshot of a cell phone  Description automatically generated  [ofApp::draw()]  update()와 마찬가지로 1/60초마다 반복 호출되는 함수이다. 이 프로그램에서 사용될 모든 color의 정보는 ofApp.h에서 배열로 선언했으므로, 적절한 인덱스의 원소를 참조하여 color를 setup하고 그림을 그린다.  가장 먼저 배경 frame(outline)을 먼저 그리고, 그 다음 오른쪽 side menubar에 들어갈 각 물체들의 그림을 그려준다.  A screenshot of a cell phone  Description automatically generated  화면에서 쓰일 모든 Text 문구들을 그린다. ofTrueTypeFont 클래스에 포함된 drawstring 함수를 이용해, load 된 font로 화면에 글씨를 그린다.  status monitor를 띄우기 위해 stringstream 클래스를 이용했다. string 문자열을 10진수, 16진수 정수로, 혹은 그 반대로 변환하기에 굉장히 편리한 클래스가 namespace std에 정의되어 있다. fps는 float형 실수로 저장되어 있고, 기본적으로 소수점 이하 6자리의 정확도를 갖고 있다. 모니터에는 소수점 이하 2자리만 표시해도 충분하므로, std::fixed와 std::setprecision(2)를 stringstream의 옵션으로 추가하여 fps를 소수점 이하 2자리까지만 표시된 string으로 변환한다. BPM은 int형으로 선언되어 있으므로, 마찬가지로 std에 정의된 to\_string() 함수를 이용해 string으로 변환 후 저장한다. 이를 string status에 옮겨 drawString()으로 그린다.  A screenshot of a cell phone  Description automatically generated  모든 배경을 다 그렸으면, 물체들을 그 위에 그린다. 가장 먼저 Generator와 각 BeatBall들을 모두 그린다. 모든 Generator가 BeatBall vector를 가지고 있으므로 nested for loop를 이용하여 모두 그려준다. 다만, MAX 개의 BeatBall을 전부 그리는 것이 아니라, active\_flag가 true로 set되어 있는 BeatBall만을 그린다. 그리고 BeatBall을 그리기 전, collide()와 move()를 호출하여 매 frame마다 BeatBall이 올바른 위치에 그려지고 다음 방향으로 진행할 수 있도록 한다. 즉, draw() 함수에서 BeatBall의 위치(및 운동 상태) 갱신과 Draw 둘 다 수행하는 것이다. BeatBall이 스크린 범위를 벗어났다면 active\_flag를 clear하여 collide(), move()를 수행하지 않도록 한다.  A screenshot of a cell phone  Description automatically generated  그 후, Pedal vector를 참조하여 모든 pedal들을 그린다. 이때, 종류가 같은 것끼리는 같은 color로, 다른 것끼리는 다른 color로 그려야 하므로 for 문 내부에서 ofSetColor를 통해 각기 다른 color를 호출한다. ofDrawLine 함수로 그리며, 모든 Pedal은 양 끝점의 x, y좌표를 이미 저장하고 있으므로 쉽게 그릴 수 있다. 추가로, BeatBall과의 collision 발생 시 bounce effect를 구현하기 위해 길이와 투명도에 offset을 설정하여 생동감있는 시각 효과를 연출한다.  Path도 Pedal과 비슷하게 그리지만, Pedal처럼 충돌 시 시각 효과가 연출되는 것이 아니므로, 별도의 offset은 설정하지 않는다.  A screenshot of a cell phone  Description automatically generated  [ofApp::mousePressed()]  사용자가 마우스 Left click을 했을 때 함수 내부로 진입한다. 이 key command로 가능한 동작은 물체의 이동 및 회전, 추가, 삭제이다. 가장 먼저, 클릭한 위치가 Generator, Pedal, Path 중 어느 물체의 위치인지 판단한다. 해당되는 물체가 있다면, 그 물체의 isClicked flag를 활성화시킨다. Pedal과 Path의 경우에는 이동 외에도 회전 기능이 있으므로, 중심을 클릭했을 때와 양 끝점을 클릭했을 때 다른 flag를 활성화하도록 한다. 이 flag는 mouseDragged()와 mouseReleased()에서 검사하여 그에 맞는 동작을 수행한다.  A screenshot of a cell phone  Description automatically generated  만약 위에서 해당되는 물체가 없었다면, 그 다음으로 사용자가 물체를 추가하려는 것인지를 판단한다. 클릭 위치를 검사하여 side menubar 내부인지를 확인하고, 맞다면 그에 맞는 객체를 생성하여 해당 vector에 push한다(= 새로운 물체를 필드에 생성한다).  A screenshot of a cell phone  Description automatically generated  A screenshot of a cell phone  Description automatically generated  [ofApp::mouseDragged]  Left Click 상태에서 Drag했을 때에만 함수 내부로 진입한다(그 외에는 mouse drag로 수행되는 동작이 없기 때문). Pedal과 Path의 경우, 중심부가 클릭되었다면 move\_flag를, 양 끝이 클릭되었다면 rotate\_flag를 활성화하고 mouseReleased()로 넘어간다.  A screenshot of a cell phone  Description automatically generated  [ofApp::mouseReleased]  마지막으로 마우스 click & drag가 끝나면 (즉 drop이면) 모든 flag 를 검사하여 그에 맞게 alert string을 설정하고 물체를 set한다.  A screenshot of a cell phone  Description automatically generatedA screenshot of text  Description automatically generated  만약, 사용자가 물체를 생성하려고 오른쪽 메뉴로부터 클릭을 했다가 다시 원위치에 내려놓는 경우, 물체 생성을 취소하고 확인 메시지를 출력한다. 클릭한 시점에서 물체가 이미 생성되어 vector에 push되었으므로, 이 객체는 vector의 맨 뒤에 위치할 것이고 따라서 pop만 해주면 바로 해당 객체가 삭제된다. 모든 동작이 끝난 후 alert에 저장된 메시지를 스크린과 콘솔에 출력한다.  A screenshot of a cell phone  Description automatically generated  [ofApp::save()]  사용자가 S key를 눌렀을 경우에 호출되는 저장 함수이다. .abf 형식의 파일로 저장하며, Dialog를 화면에 띄워 디렉토리와 파일명을 입력하면 해당 경로에 파일을 생성해준다. 성공 시 true, 실패 시 false를 return한다.  A screenshot of text  Description automatically generatedA screenshot of a cell phone  Description automatically generated  [ofApp::setInfoString]  현재 필드의 모든 정보를 string 형식으로 저장하고, 이 string을 return하는 함수이다. ofApp.h 에서 정의한 String Format대로 infoString을 완성한다. 파일이 성공적으로 저장되면 다음과 같은 형태의 .abf파일을 확인할 수 있다.    [init.abf]  A screenshot of a cell phone  Description automatically generated  [ofApp::openFile]  사용자가 L key를 누르면 호출된다. Dialog를 띄워 파일을 선택하고 해당 파일명을 return한다. 만약 유효하지 않은 파일을 선택했다면 빈 string을 return한다.  A screenshot of a cell phone  Description automatically generated  [ofApp::load()]  openFile()의 return 값(string fileName)을 그대로 인자로 받아 해당 파일을 load하는 함수이다. 만약 전달된 fileName이 empty string이라면 openFile() 함수에서 file open에 실패한 것이므로 그대로 false를 반환한다.  가장 먼저 파일의 확장자를 검사한다. .abf 형식이 아니라면 마찬가지로 false를 반환한다. 확장자가 올바르다면, buffer에 모든 텍스트 정보를 옮기고 iterator를 선언하여 line-by-line 참조를 한다. BPM과 volume 정보를 읽어 그대로 저장한다. ofApp.h에 정의해둔 string 형 Label을 비교하여 파일이 올바른 형식으로 저장되어 있는지를 검사한다.  A screenshot of a cell phone  Description automatically generated  iter를 현재 위치에서 다음 label이 나올 때까지, line별로 Pedal과 Path의 정보를 읽는다. color 정보는 16진수로 저장되어 있으므로 알맞게 변환하여 color setup을 한다. sound sample의 정보도 불러들이는데, 본 프로젝트의 sound sample은 bin/data/sample/에 위치해 있으므로 미리 매크로 상수로 경로를 정의해놓으면 나중에 파일명만 가지고도 해당 경로에 있는 파일을 load할 수 있다.  A screenshot of a cell phone  Description automatically generated  color, sound setup이 끝났으면 각 객체별 위치 정보를 읽는다. 마찬가지로 iter를 다음 label에 도달할 때까지 반복문을 돌며 vector에 차례로 push한다.  A screenshot of a cell phone  Description automatically generated  A screenshot of a cell phone  Description automatically generated  load가 끝날 때까지 예외 처리되지 않으면 true를 반환한다. |

1. **실행 결과**

|  |
| --- |
| A screenshot of a cell phone  Description automatically generated  [init.abf]  A screenshot of a computer screen  Description automatically generated  [File Save]  A screen shot of a clock  Description automatically generated  [Loaded File]  A picture containing clock, computer  Description automatically generated  [Quit Program] |

1. **프로그램의 개선점 & 새롭게 배운 점**

|  |
| --- |
| <Amazing BeatFall!> 은 사용자가 마치 벽돌 튕기기 게임처럼 시각적, 청각적인 요소를 활용해 즐길 수 있는 프로그램이다. 처음 자료구조를 세울 때부터 구현하고자 하는 모든 기능을 염두에 두고 프로그래밍한 것이 아니어서, 개발 과정에서 현재 자료 구조로는 구현하기 어렵겠다고 판단하여 부족한 부분이 일부 있었다.  첫 번째는 sound sample을 load하는 기능이 파일명 형식으로부터 자유롭지 못했다는 것이다. 본 프로그램은 각 Pedal type 별 sound sample의 파일명 형식을 모두 통일하여 해당 형식의 파일명을 랜덤으로 생성하여 load하는 방식인데, 이 구조의 문제점은 파일명이 다른 (혹은 확장자가 .wav가 아닌) sample은 프로그램에 이용하기 어렵다는 것이다. 이를 개선하기 위해 굳이 파일명을 통일하지 않아도 내가 원하는 디렉토리에 어떠한 파일이 있는지, 그리고 그 파일의 확장자가 무엇인지를 검사할 수 있는 함수를 이용한다면 이 문제를 해결할 수 있을 것이다.  두 번째는 시간의 흐름을 완벽하게 정확히 측정하지 못한다는 점이다. 본 프로그램은 ofApp::update() 함수가 호출될 때마다 단위 시간이 흘러가는 것을 count하여 시간이 얼마나 지났는지를 판단하도록 설계되어 있다. 그렇기 때문에, 어떤 순간에 많은 양의 함수가 동시에 호출된다면(예를 들어 100개의 Pedal의 hit\_flag가 동시에 set되는 경우) update()가 정확히 1/60초만에 호출되지 않을 수 있고, 그러므로 시간의 흐름을 계산하는 데에 오차가 생길 수밖에 없다. 실제로 status monitor에 fps(frame per second)를 출력하여, 매 순간 정확히 60을 유지하는 것이 아니라 59.4~60.2 의 범위에서 변동이 있는 것을 확인했다. 그러므로 update()의 호출 주기에 의존하지 않고, time.h 관련 함수를 사용한다면 훨씬 정확한 시간 측정이 가능할 것이다.  이번 기말 프로젝트를 진행하며, 객체 지향 언어에 대한 전반적인 이해도가 높아졌다. 프로그래밍 언어라고는 C언어밖에 없던 나에게 OpenFrameWorks와 C++을 활용한 프로그래밍은 흥미있으면서도 하나의 커다란 벽처럼 느껴졌다. 그럼에도 불구하고, 강의자료 및 StackOverFlow같은 온라인 커뮤니티를 적극 활용하며 C++에 대해 공부했고, 덕분에 클래스를 다루는 능력을 길렀으며 콘솔 I/O뿐 아니라 내가 코딩한 프로그램을 시각화하는 방법을 터득할 수 있었다. 단순히 알고리즘 문제를 풀기 위한 코드가 아니라 하나의 완성된 프로그램을 만들기 위해, 시행착오도 많이 겪었고 방대한 양의 코드 속에서 디버깅하느라 당혹스러울 때도 많았지만, 처음부터 끝까지 온전히 혼자의 힘으로 만든 첫 프로그램이라는 사실이 뿌듯했고 동시에 프로그래머로서 나의 부족한 부분이 무엇인지 객관적으로 판단할 수 있는 계기가 되었다. 더욱 거대한 프로그램을 만들고 다루기 위해서는, 자료구조 과목을 잘 복습하고 익혀서 더욱 효율적으로 메모리 관리를 할 수 있어야겠다고 느꼈다. 또한, 내가 필요한 기능을 구현하기 위해 인터넷에 무엇을 어떻게 검색해서 찾아야 하는지를 아는 것 또한 매우 중요하다고 느꼈다. 좋은 프로그래머란 이론에 해박하고 동시에 내가 모르는 것을 실전 상황에서도 바로바로 해결할 수 있는 능력을 갖춘 사람이고, 그것을 위해서 복습과 실전 문제 해결 경험이 모두 중요하다는 것을 깨닫게 된 한 학기였다. |