리듬 타! (Rhythm Ta!)

프로젝트 명	리듬 타! (Rhythm Ta!) - 오픈 소스를 이용한 리듬 게임	
프로젝트 팀원	장혜원	15011171
	안재현	16011140
	차봉호	16011145
	유지인	16011191
	진예진	16011208

과목	오픈 소스 SW 개론
지도 교수님	장문정 교수님
제출 날짜	2018-05-27

SEJONG UNIVERSITY

목차

표 & 그림 목차

1. 프로젝트 개요

1.1 프로젝트 주제

오픈 소스 커뮤니티를 이용하여 개인 개발자의 리듬 게임 프로그램을 개선해서 다양한 레벨과 기능이 추가 된 리듬 게임 완성

1.2 주제 선정 이유

팀원 중 다수가 '소프트웨어 설계 기초' 강의를 수강했는데, 이 때 게임 개발 시 구현에 필요한 기능들에 대해 공부하고 팀 프로젝트로 자유 게임을 직접 개발하는 경험을 했다. '소프트웨어 설계 기초' 강의에서는 게임을 처음부터 끝까지 만들면서 간단한 수준의 게임을 구현했기 때문에 비교적 쉬운 콘솔 게임을 개발했음에도 불구하고 구현하려는 게임을 처음부터 직접 기획하고 제작하려고 하니 상당한 어려움과 시간이 소요됐다. 그 때 경험했던 어려움을 바탕으로 이번에는 오픈 소스 커뮤니티를 이용해서 다른 개발자들이 만든 소스 코드를 수정하고 게임개발 과정에서 소요되는 시간이 크게 단축되는 경험을 하기 위해 위 주제를 선택했다. 또, 기존의 오픈 소스 게임에 여러 가지 부족한 기능들을 추가해 게임을 보완하면서 이전 강의에서 만들었던 게임보다는 조금 더 수준 높은 게임을 완성한다.

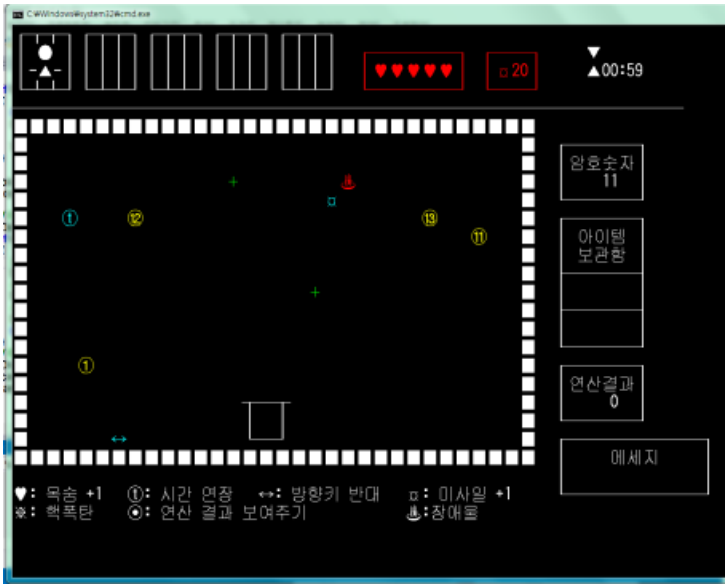


Figure 1 SW 설계 기초 강의 시간에 개발한 게임

1.3 프로젝트 목표

오픈 소스를 이용하여 게임을 만드는 과정에서 오픈 소스를 활용하는 방법과 사용하려는 오픈 소스의 라이선스를 바탕으로 바람직한 활용을 실전에서 경험하는 것이 프로젝트의 주요 목표이다. 또, 사용한 오픈 소스를 가독성이 높은 코드로 바꾸고, 좋은 알고리즘을 사용하며, 후에 우리와 혹은 다른 개발자들이 유지 보수를 쉽게 할 수 있도록 적절한 주석들을 작성하여 보완할 계획이다. 오픈 소스를 개선하면서 코드를 읽는 사람인 동시에 코드 작성자가 되기 때문에 유지보수와 확장이 쉬운 코드를 만들 수 있을 것이라고 생각합니다. Git 과 Github 를 사용하여 더 편리하게 오픈소스를 팀원들과 공동으로 개발하고 유지, 관리를 하면서 실력을 향상시키며 오픈소스의 가치를 알아가는 것이 목표이다.

선택한 리듬게임 오픈 소스의 코드를 강의시간에 배운 좋은 코드의 요소들을 적용시켜 가독성이 뛰어난 코드로 수정한다. 게임을 플레이 해보면서 발견했던 오류를 해결하고, 기존 코드에는 완성이 되지 않은 한 개의 곡만 실행 가능하지만, 채보 프로그램의 사용 방법을 익혀서 미완성의 채보를 완성시키고 실행 가능한 음악의 종류를 더 추가한다. 이외에도 랭킹 기능, 싱크로율 조절 기능, 일시 정지 기능과 같이 기존에 없던 기능들을 추가하여 더 발전된 결과물을 완성한다.

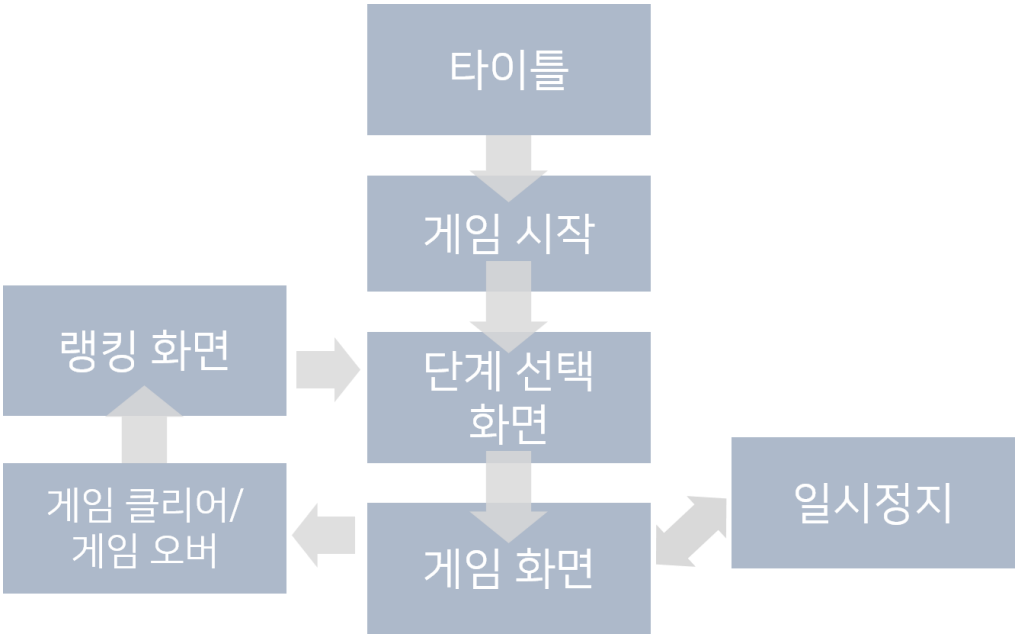


Figure 2 프로젝트 계획 단계의 프로그램 목표 구상도

1.4 프로젝트 계획 범위

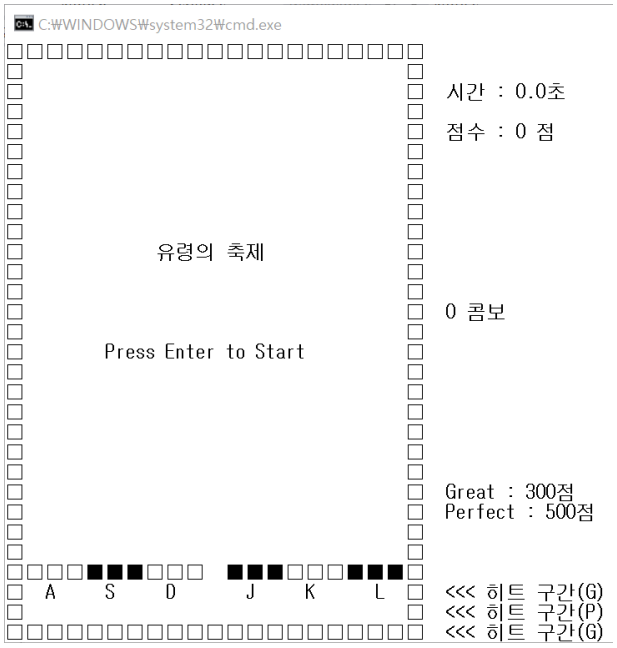


Figure 3 오픈 소스 리듬 게임 실행 화면

1.4.1 게임 맵 하단의 히트구간 수정

게임의 맵 하단의 히트구간은 Great 과 Perfect 구간으로 나누어져 있다. 그러나 게임을 직접 실행해 보았을 때 플레이어가 두 히트 구간을 구분 하는 것이 어려워 게임의 이해를 방해한다.

- 개선 방향 : 히트 구간에 서로 다른 색을 넣어 플레이어가 Perfect 구간에 노트를 맞추기 쉽도록 돕는다.

1.4.2 콤보 시스템의 개선

저희가 선택한 오픈 소스 리듬 게임에서는 콤보 기능이 있다. 노트가 히트 구간에 내려왔을 때 플레이어가 노트에 맞는 키를 연속으로 맞추면 콤보의 수가 늘어나게 된다. 하지만 이 콤보는 플레이어에게 히트의 연속 성공 횟수만 알려줄 뿐 특별한 기능을 하지 않고 동시에 내려오는 노트는 콤보의 수가 증가하지 않는다.

- 개선 방향 : 콤보에 따른 추가 점수를 부여하고 콤보 수 카운트 함수를 개선한다.

1.4.3 부실한 게임 설명

일반적으로 게임의 첫 화면은 게임의 전반적인 설명이 필요하지만 이 코드는 게임 설명이 충분하지 않다.

- 개선 방향 : 게임할 때 사용되는 키, 점수와 콤보 부분을 플레이어가 더 보기 쉽도록 UI 를

개선하고 콤보에 따른 점수 증가, 어떤 방식으로 게임이 진행되는지를 설명하는 화면을 추가한다.

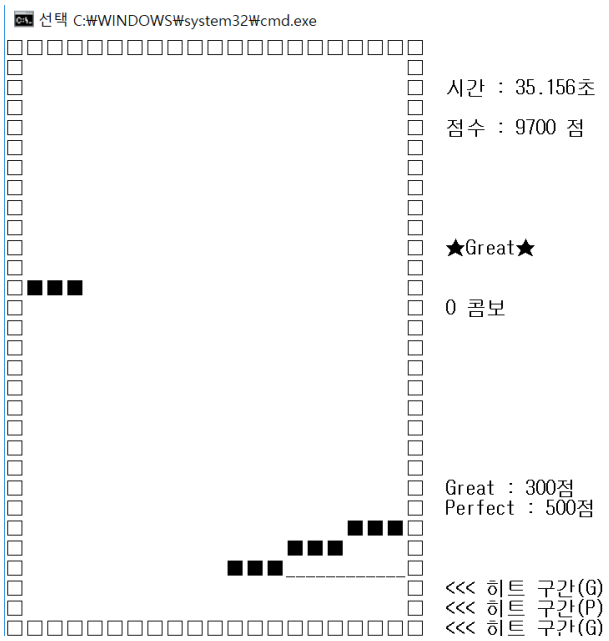


Figure 4 오픈 소스 리듬 게임 플레이 중 화면

1.4.4 맞지 않는 싱크로율

싱크로율은 Synchronize(동시에 발생하다)의 준말인 sync(싱크)와 비율의 합성어다. 노래의 리듬에 맞춰 노트들이 내려오는 것은 리듬게임의 핵심이다. 그러나 이 게임은 노트의 싱크로율이 리듬과 맞지 않고 어색하게 떨어져 플레이가 쉽지 않다.

- 개선 방향: 기존 코드는 시간의 경과에 따라 정해진 노트가 내려오도록 작성되어 있다. 노래를 조금 더 세분화하여 분석해서 노트가 리듬에 따라 정확하게 내려올 수 있도록 개선한다.

1.4.5 단순한 노트

노트들이 비교적 단순한 편이라 플레이어가 게임을 실행할 때 지루함을 느끼게 된다.

- 개선 방향 : key 입력을 지속해야 하는 긴 노트나, 내려오면서 모습이 보이지 않는 노트와 같이 다양한 모습의 노트를 추가해서 사용자에게 높은 난이도와 재미를 줄 수 있도록 한다.

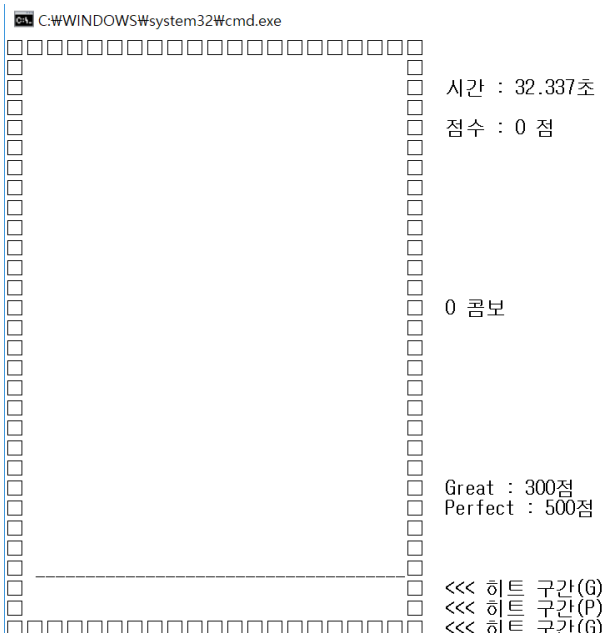


Figure 5 오픈 소스 리듬 게임 종료 화면

1.4.6 정해지지 않은 끝

게임 시작 후 30 초 이후부터 플레이 할 수 있는 노트들은 일정 시간 이후 내려오지 않지만 노래는 1 분 30 초가량 계속 재생된다.

- 개선 방향 : 곡의 길이를 줄이거나 노트들을 더 만들어 곡의 재생 시간과 노트의 출력 시간을 조정한다. 추가로 기존의 코드는 곡의 선택 없이 한 곡만 실행할 수 있지만 여러 개의 곡을 추가해서 선택하고 실행할 수 있도록 한다.

1.4.7 부실한 인터페이스

사용자에게 곡이 끝났음을 알려주는 화면이 구현되지 않았다.

- 개선 방향 : 처음으로 되돌아갈 수 있는 단축키를 만들고, 한 곡의 실행이 끝나면 다른 곡을 선택할 수 있도록 하거나 자신의 점수를 알 수 있는 랭킹 화면을 출력한다. 또, 게임의 끝을 알릴 수 있는 효과 사운드를 추가한다.

1.4.8 런타임 에러

노트가 다 내려오고 난 후 50~55 초에 런타임 에러가 발생한다.

- 개선 방향 : 어느 지점에서 오류(무한 루프)가 발생하였는지 파악하여 코드를 수정한다.

1.5 프로젝트 환경

- OS : Windows 10 32bit / Tools : Visual Studio 2017 (sdk15 이상 권장) / Github / SourceTree

1.5.1 Windows 10 32 bit

Windows 환경을 선택한 이유 중 하나는 한국에선 Mac OS 보다 보편적으로 사용되고 있으며 제일 접근하기 쉬운 운영체제이고 팀원들도 5 명 전부 Windows OS 를 사용하고 있어서 프로젝트를 시작 하기 앞서 Windows OS 를 사용하는 것이 옳다고 판단했다. 또 Windows OS 를 선택한 가장 큰 이유는 선택한 리듬 게임이 라이센스가 없는 오픈 소스여서 개발자와 컨택을 했을 때 개발환경이 Windows OS 였다는 답변을 받았고, 프로젝트 운영체제 환경을 Windows OS 로 결정했다.

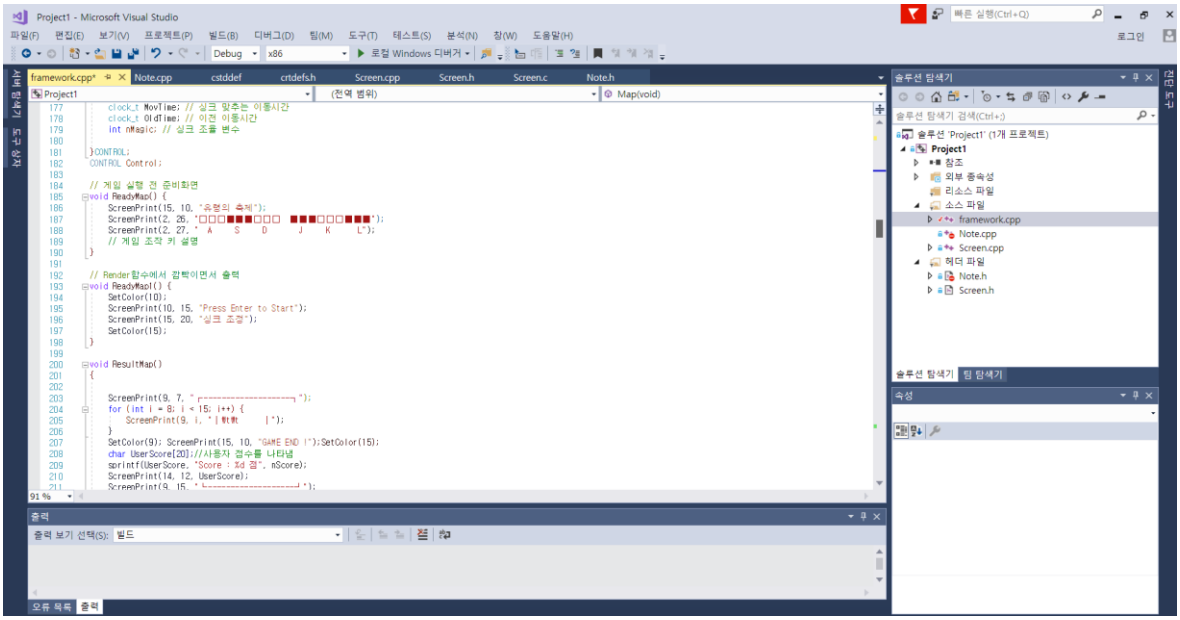


Figure 6 Visual Studio 2017 로 리듬 게임 프로젝트 실행

1.5.2 C++ / fmod

“fmod”는 Win32 API 로써 32 비트의 VisualStudio 환경에서 C++을 지원하는 라이브러리다. fmod 는 음향 미들웨어로써 VisualStudio 프로젝트에 다양한 음향 기능을 제공해준다. 가장 큰 특징은 여러 채널을 이용해서 사운드를 동시 사용가능하다는 것인데, fmod 를 이용해서 일시정지 기능을 사용하고자 한다.

1.5.3 소스트리(SourceTree)

강의를 들으면서 git bash 터미널을 이용해 팀끼리 협업 작업하는 것을 배우고 연습했지만 편집기의 사용이나 커밋, 브랜치를 한 눈에 보기 힘들다는 점 때문에 우리는 커밋, 브랜치 관리와 pull, push 를

쉽게 할 수 있는 프로그램으로 우리의 개발도구인 Visual studio 2017 에서 함께 관리를 하려 했으나 조원 중 몇 명의 개발환경에서 다른 사람이 push 한 내용이 바로 보이지 않는 등 작업이 원활하지 않았기 때문에 분산 버전 관리 시스템인 소스트리를 함께 사용하기로 했다. 소스트리(https://www.sourcetreeapp.com/)는 소프트웨어 개발자를 타겟으로 하는 기업인 아틀라시안에서 내보인 개발자들이 협업할 때 Git 기능들을 시각화해서 관리과 쉽고 개발이 편리하도록 돕는 프리 소프트웨어이다. 소스트리의 업데이트 소식이나 새로운 정보들을 보여주는 소스트리 블로그(https://blog.sourcetreeapp.com/)와 소스트리 튜토리얼을 알려주는 페이지(https://confluence.atlassian.com/bitbucket/tutorials)도 제공해주기 때문에 쉽게 소스트리에 적응할 수 있었다. Windows 와 Mac OS 에서 소스트리를 사용할 수 있고 해당 소프트웨어는 소스트리 홈페이지에서 다운로드 받을 수 있었다.

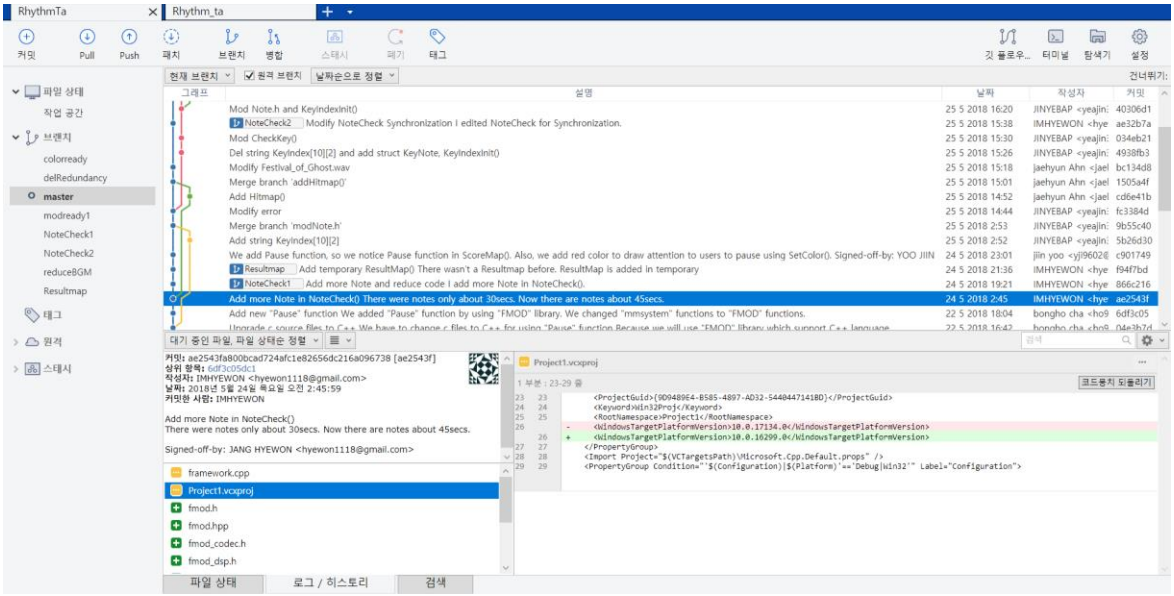


Figure 7 소스트리 실행 화면

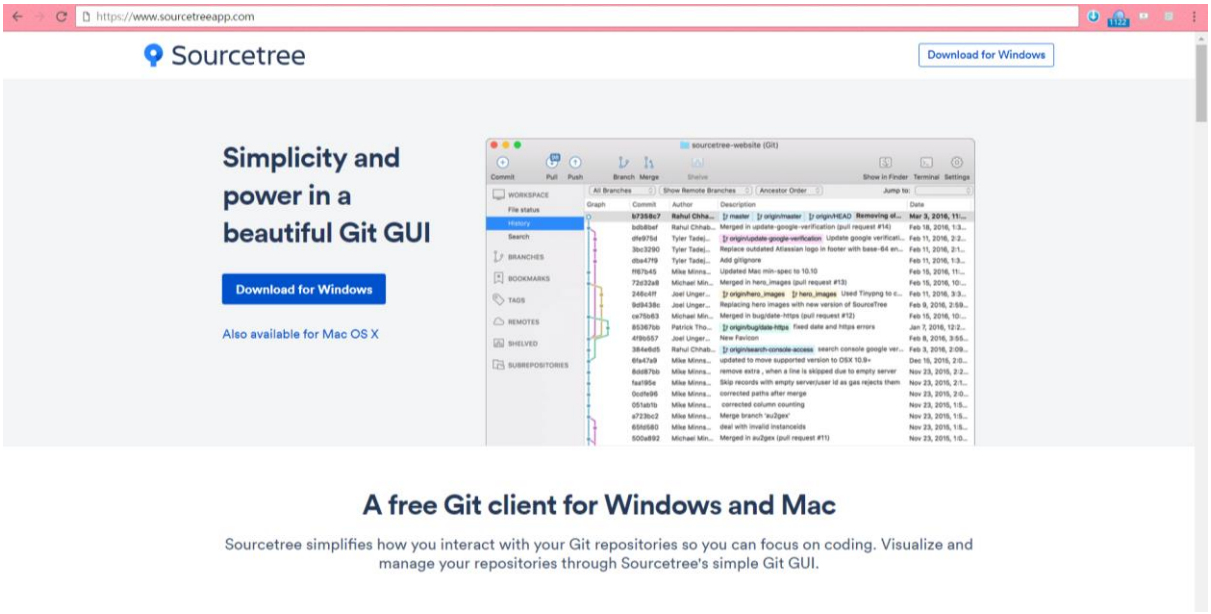


Figure 8 소스트리 홈페이지

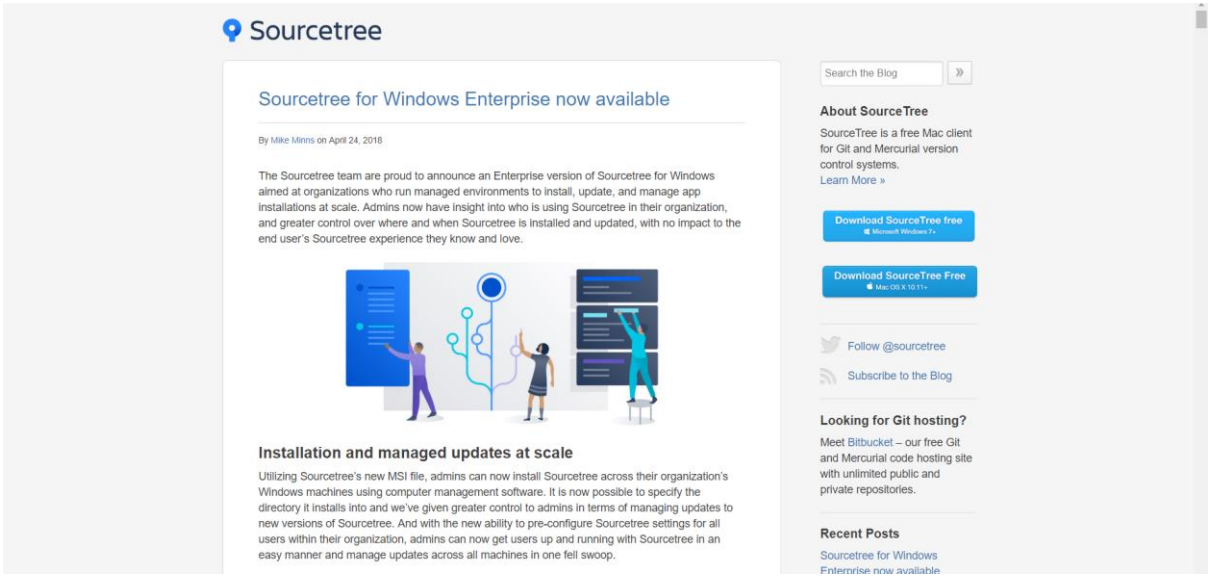


Figure 9 소스트리 블로그

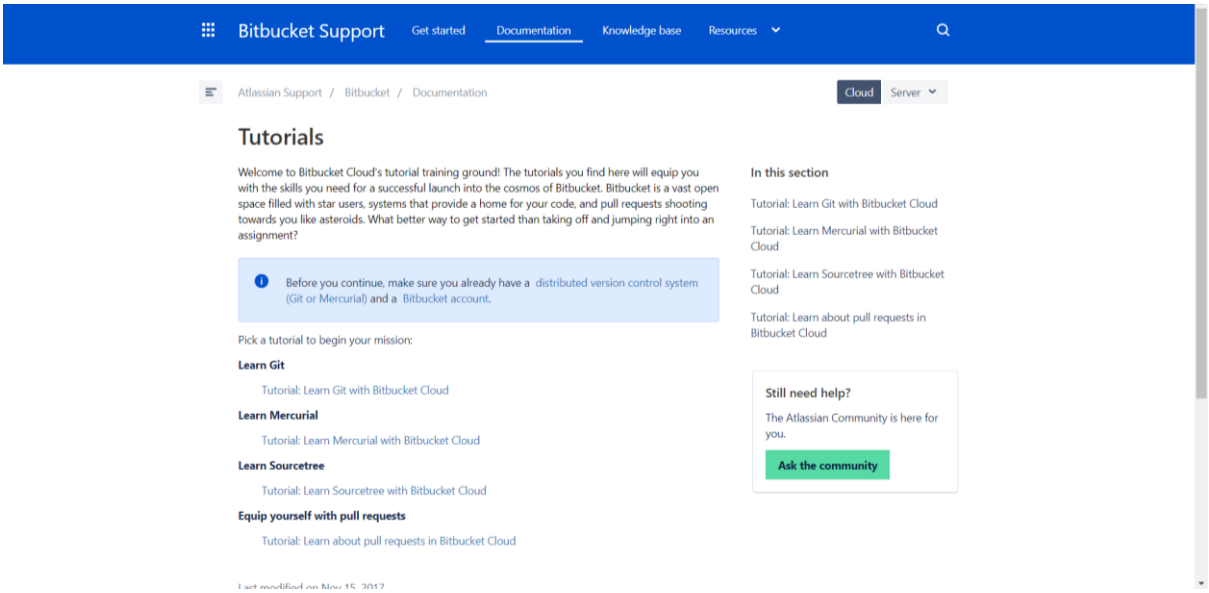


Figure 10 소스트리 튜토리얼

2. 오픈 소스 SW

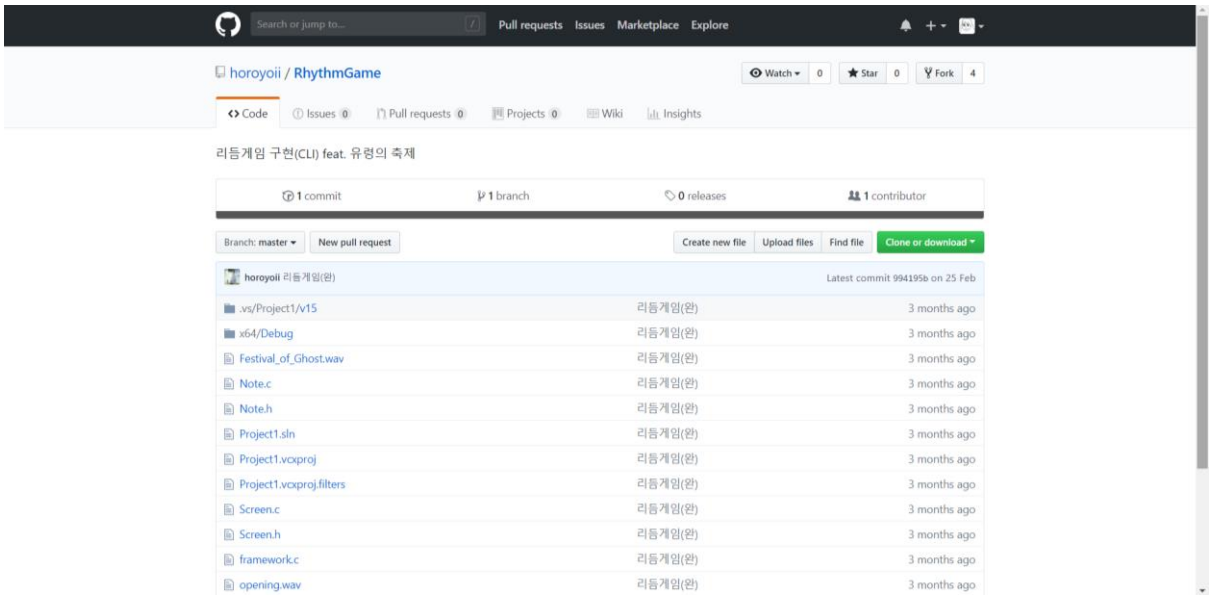


Figure 11 Github horoyoii 리듬 게임

2.1 기존 오픈 소스 SW

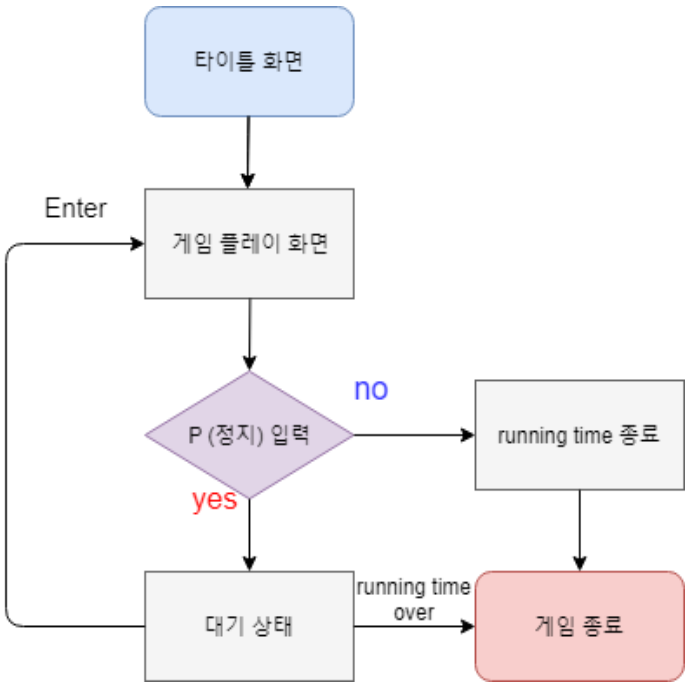


Figure 12 기존 오픈 소스 리듬게임의 전체 Flow – chart

2.1.1 기존 오픈 소스 SW 설명

이 오픈 소스는 개인 개발자가 github 에 올린 리듬게임 오픈 소스이다. Readme 가 따로 없었기 때문에 폴더에 함께 첨부된 기획.hwp 를 참고했다. Rhythm Game 이라는 제목의 이 오픈 소스는 유령의 축제라는 노래의 리듬을 맞추는 게임으로, 노래의 총 길이는 약 1 분 37 초이며 30 초 동안 리듬에 맞는 노트가 떨어진다. 플레이어는 떨어지는 노트에 맞게 키보드의 키를 사용해서 노트를 맞추며 점수를 획득할 수 있다. 이 게임의 기능은 크게 볼 때 두 가지로 정리되어 있다. 첫째로, A, S, D, J, K, L 키를 사용하여 각각의 위치에서 떨어지는 노트를 맞추며 점수를 얻는다. 히트 구간은 Perfect 와 Great 의 두 구간으로 나뉘어져 있으며 각 위치에서 노트에 맞는 키를 입력하면 구간에 맞는 점수를 얻을 수 있다. 이 두 가지 기능 이외에도 게임을 시작하는 Enter 기능, 플레이 시간과, 사용자 점수, 히트 판정 등 UI 를 제공한다.

개발자는 스테이지 구성을 준비, 러닝, 중지, 결과 4 가지로 구성했다. 준비 화면에서는 플레이어의 시작 신호를 기다리며 플레이어가 게임 시작을 위해 Enter 를 입력하면 스테이지가 러닝 단계로 바뀌며 게임이 실행된다. 게임 시작 후 3 초가 지나면 노래에 맞춰서 노트가 떨어지고, 플레이어는 게임을 하며 점수를 얻는다. P 를 입력하면 콘솔창의 우측에 있는 시간이 멈추며 중지 상태가 된다. 개발자는 결과 스테이지를 구성했으나 구현하지는 않았고 플레이어는 결과화면을 볼 수 없다.

러닝 화면에서 떨어지는 노트를 구현 할 때 개발자는 “2 차원 배열 속에 노트를 다 넣고, 2 차원 배열을 한 킨 씩 내려 출력한다”고 설명했다. 이에 따라 게임 상에서 나오는 노래에 맞춰 정해진 노트 배열이 떨어지는 것처럼 보인다. 게임 맵은 y 축을 1-28 칸, x 축을 1-42 칸으로 구현하고 □ ■ 문자를 사용해서 화면을 구성했다.

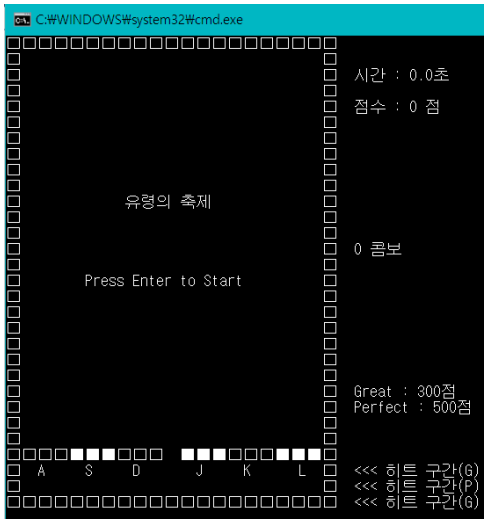


Figure 13 기존 리듬 게임 시작 화면

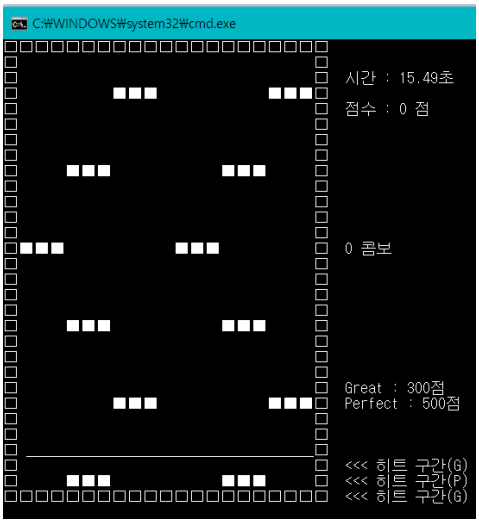


Figure 14 기존 리듬 게임 실행 화면

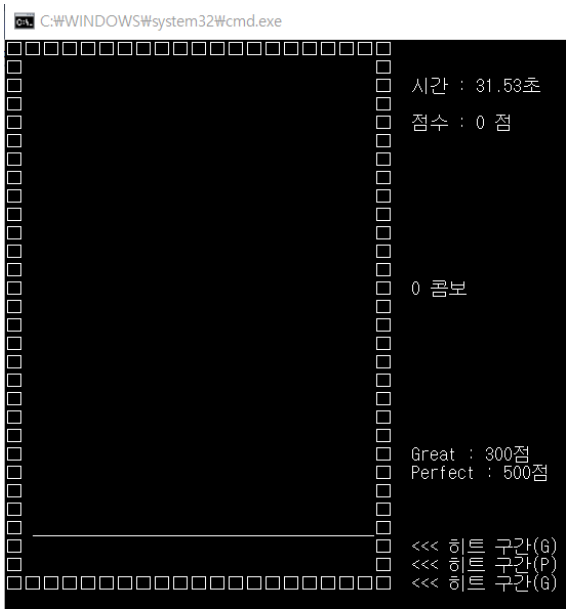


Figure 15 기존 리듬 게임 완료 화면

2.1.2 기존 오픈 소스 SW 프레임워크

기존 오픈 SW 에 있는 Screen.cpp 는 게임 프레임워크이다. 프레임워크란 소프트웨어 어플리케이션 이나 솔루션 개발을 수월하게 하기 위해 소프트웨어의 구체적 기능을 제공하는 부분을 제공하는 소프트웨어 환경을 말한다. (자주 쓰는 기능들을 재사용 가능하도록 하는 뼈대)

2.1.2.1 ScreenInit() : 두 개의 버퍼 생성 후 커서 설정하는 함수이다, 이 함수에서는 두개의 버퍼(g_hScreen[0], g_hScreen[1]) 을 만든 뒤 커서의 정보(안보이게)를 이 두 버퍼에 설정한다.

2.1.2.2 ScreenFlipping() : 버퍼를 교환하고 , 활성화된 버퍼를 화면에 그리는 함수(Render 에서 호출한다.

- SetConsoleActiveScreenBuffer() 함수는 인자로 넘겨주는 버퍼를 콘솔화면에 그린다. 이 함수는 Render() 에서 호출되며 바뀐 정보들을 버퍼에 저장하고 다른 버퍼에 넘겨준다.

2.1.2.3 ScreenClear(): 화면을 지워주는 함수

- COORD : 화면의 위치를 나타내는 변수
- FillConsoleOutputCharacter(buffer , ‘ ’ , 80*25 , Coor(0,0) , &dw)는 buffer 의 (0.0) 부터 시작하여 80*25 만큼 공백 만큼 채운다는 뜻이며 dw 는 버퍼에 기록된 문자수를 받는 변수이다.

2.1.2.4 ScreenRelease() : 화면 해제 하는 함수

- CloseHandle 함수는 핸들을 닫는 기능이다

2.1.2.5 ScrenPrint() : x, y 좌표에 String 을 출력하는 함수이다.

- 인자로 x, y, string 을 받아 x,y 좌표로 커서의 위치를 옮긴 뒤 그 위치에 string 을 출력한다.

2.1.2.6 SetColor() : 색상을 변경하는 함수이다

- SetConsoleTextAttribute() 함수는 인자로 버퍼와 컬러값(1-15)을 받게되는데 텍스트의 색상을 바꿔준다

2.2 완성 프로젝트 리듬 게임 설명

Rhythm Ta!리듬 게임은 원 개발자(horoyoii)의 리듬게임 오픈 소스를 clone 해서 여러 기능을 추가하고 기존의 소스코드에 있던 함수, 변수들을 보완해 이전의 오픈 소스 리듬 게임보다 발전된 모습의 리듬 게임이다. 'A, S, D, J, K, L' 키를 사용하여 각각의 위치에서 떨어지는 노트를 Perfect 와 Great 두 구간에 맞추며 점수를 얻는다는 기본적인 틀은 같게 유지하였지만, 비교적 단순해서 플레이어로 하여금 크게 흥미를 끌지 못했던 노트들을 조금 더 빠르고 다양한 모습의 노트를 추가해서 재미를 주었다. 기존의 오픈 소스에서는 플레이 시간과 사용자 점수 등을 알려주는 UI 가 있었지만 플레이어가 실행할 때 편리하다고 느끼는 수준은 아니었다. 처음 실행한 플레이어는 일시 정지 기능의 유무도 파악하기 힘들었고, 일시 정지를 해도 음악은 계속해서 재생됐으며 떨어지는 노트가 점수를 얻는 구간에 맞췄는지 한눈에 파악하기가 어려웠다. 또, 재생되는 음악 시간보다 적은 노트가 내려왔으며 게임의 끝이 정해지지 않아 플레이어를 당황하게 만들었다. 이러한 점들을 UI 를 보완해서 게임 중 일시 정지와 다시 게임 실행을 할 수 있는 방법을 알려주었다. 게임 실행을 하면서 플레이어가 내려오는 노트에 잘 맞췄는지 파악하기 쉽게 하기 위해 색과 문자 변환을 추가했다. 게임은 음악 재생 시간에 맞추어 끝나고 플레이어가 획득한 점수를 보여주는 결과화면과 다시 게임을 실행할 수 있도록 했다. 또 플레이어가 직접 싱크로율을 조절할 수 있는 함수를 추가했다.

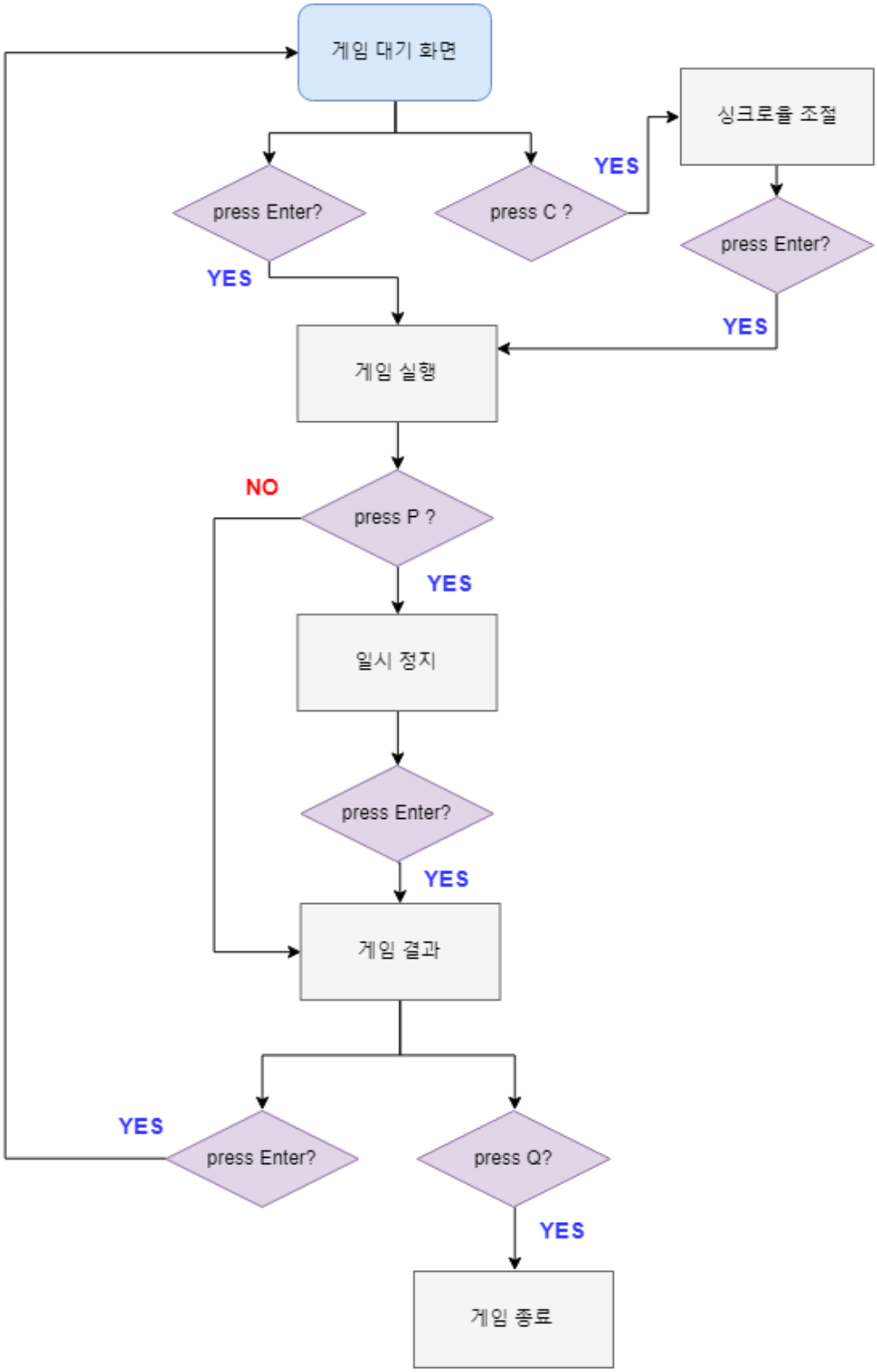


Figure 16 완성된 프로젝트 Flow - chart

2.3 완성 프로젝트 함수 설명

2.3.1 ScoreMap()

게임의 시작화면 우측에 경과 시간, 점수, 히트구간 등을 보여주는 함수이다.

2.3.2 ReadyMap(), ReadyMap1()

게임의 첫 시작 화면을 출력 해준다.

2.3.3 ResultMap()

게임의 끝을 알리고 플레이어의 획득 점수를 보여준다

2.3.4 SyncMap()

플레이어가 싱크로율을 조절할 수 있는 화면을 출력해준다.

2.3.5 ShowNote()

2 차원 배열을 아래로 떨어지게끔 보여주는 함수이다.

2.3.6 Init()

Control 구조체의 변수들을 초기화 해주는 함수이다. 주로 음악에 맞춰 노트들이 내려올 수 있도록 하는 싱크로율들을 조율하는 변수들이 Control 구조체 안에 포함되어 있다.

2.3.7 Update()

Stage 상태(현재 게임 상태)에 따라 다른 기능을 하는 함수이다. 현재 프로그램이 실행된 시간 Curtime 과 대기할 때까지의 시간 Oldtime, 그리고 게임이 시작된 시간 RunningTime 을 구해준다.

2.3.8 Render()

Render()는 이 게임의 진행 단계인 Ready 상태일 때, Running 상태일 때의 모습을 보여준다. 각 단계에서 어떤 역할을 하는지 구현하고 있다. 아래 그림에서 두 단계의 모습을 볼 수 있다.

2.3.9 secondkbhit()

두 번째로 입력된 키를 받기 위한 함수이다.

2.3.10 main()

프로젝트 파일의 메인함수이다. Init()함수를 호출하고 플레이어의 키를 입력받아 게임을 실행한다. 데이터를 갱신하는 Update()함수와 화면 출력을 담당하는 Render()함수도 호출한다.

2.3.11 NoteInit()

NoteInit()에서는 게임의 실행 음악인 'Festival_of_Ghost'에 맞는 채보 노트를 저장하고 있다.

2.3.12 KeyIndexInit()

키와 노트 String을 KeyNote 구조체에 초기화 시켜주는 함수이다.

2.3.13 GetKeyType()

키의 문자열을 반환해주는 함수이다.

2.3.14 isTwoKey

현재 노트가 두 개의 노트인지 확인해주는 함수이다.

2.3.15 HitNote

Hit가 됐을 때 가시적인 기능을 해주는 함수이다.

2.3.16 ControlSync()

방향키를 통해 싱크로율을 조절 할 수 있는 함수이다.

2.3.3 CheckKey

main에서 해당 키 입력 시 호출되는 함수이다.

2.3 다운로드 및 출처

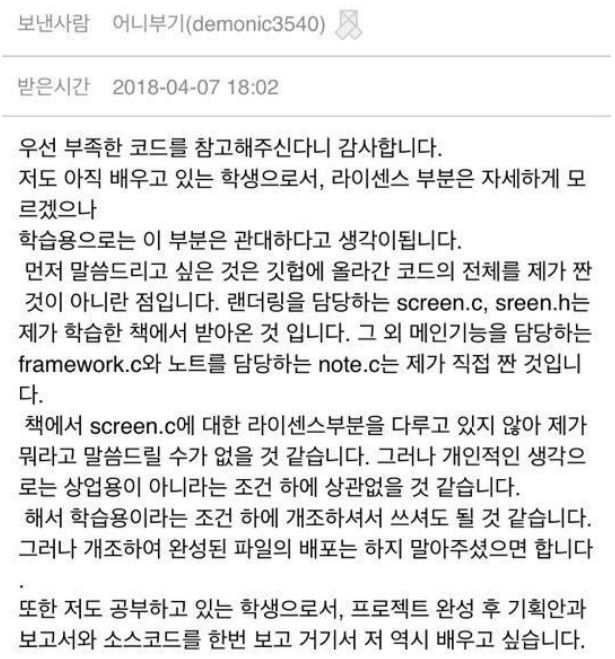
2.3.1 기존 오픈소스 SW 리듬게임 (hotoyoii) : https://github.com/horoyoii/RhythmGame.git (마지막 commit : 2018.01.25)

2.3.2 프로젝트 완성 리듬 게임 (Rhythm Ta) : https://github.com/IMHYEWON/RhythmTa.git

2.3.3 fmod : https://www.fmod.com/

2.4 라이선스

2.4.1 기존 오픈 소스 SW 라이선스


보낸사람 어니부기(demonic3540)

받은시간 2018-04-07 18:02

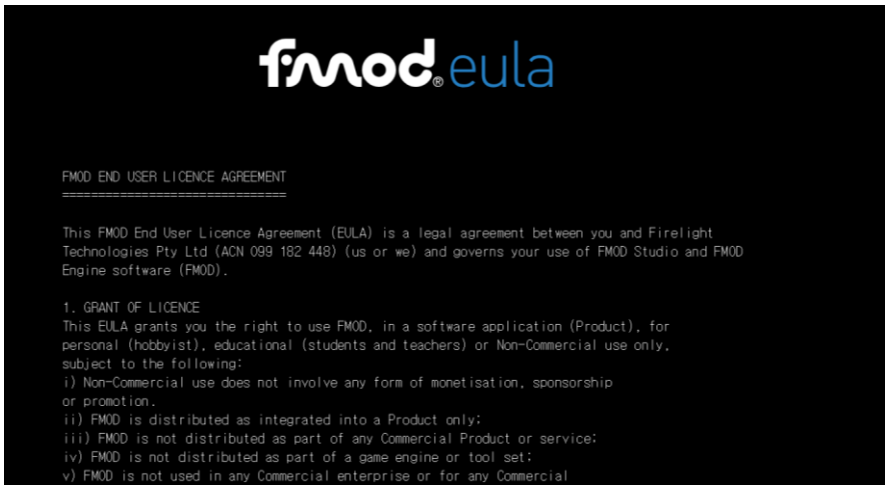
우선 부족한 코드를 참고해주신다니 감사합니다.
저도 아직 배우고 있는 학생으로서, 라이센스 부분은 자세하게 모르겠으나
학습용으로는 이 부분은 관대하다고 생각이됩니다.
먼저 말씀드리고 싶은 것은 깃헙에 올라간 코드의 전체를 제가 짠 것이 아니란 점입니다. 랜더링을 담당하는 screen.c, sreen.h는 제가 학습한 책에서 받아온 것 입니다. 그 외 메인기능을 담당하는 framework.c와 노트를 담당하는 note.c는 제가 직접 짠 것입니다.
책에서 screen.c에 대한 라이센스부분을 다루고 있지 않아 제가 뭐라고 말씀드릴 수가 없을 것 같습니다. 그러나 개인적인 생각으로는 상업용이 아니라는 조건 하에 상관없을 것 같습니다.
해서 학습용이라는 조건 하에 개조하셔서 쓰셔도 될 것 같습니다. 그러나 개조하여 완성된 파일의 배포는 하지 말아주셨으면 합니다.
또한 저도 공부하고 있는 학생으로서, 프로젝트 완성 후 기획안과 보고서와 소스코드를 한번 보고 거기서 저 역시 배우고 싶습니다.

Figure 17 원 개발자와 주고 받은 쪽지

선택한 리듬게임 오픈 소스는 라이선스가 정의되어있지 않아서 개발자에게 직접 메일을 이용하여 연락을 했다. Screen.c와 Screen.h는 'C를 이용한 게임 프로그래밍(이태성 저)'이라는 책을 참고 하였고, 따라서 상업적 이용만 아닌 학습용의 목적으로는 개조해도 된다고 답장을 받았다.

2.4.2 fmod 라이선스

fmod 는 최종 이용자 라이선스 계약(eula)을 사용한다. 요약하자면 fmod 는 개인, 교육 또는 비상업적 용도로 사용할 때 무료로 사용할 수 있다. 이용자가 본 계약에 동의하지 않는다면 fmod 가 제공하는 서비스를 이용할 수 없다.


fmod eula

FMOD END USER LICENCE AGREEMENT

This FMOD End User Licence Agreement (EULA) is a legal agreement between you and Firelight Technologies Pty Ltd (ACN 099 182 448) (us or we) and governs your use of FMOD Studio and FMOD Engine software (FMOD).

1. GRANT OF LICENCE
This EULA grants you the right to use FMOD, in a software application (Product), for personal (hobbyist), educational (students and teachers) or Non-Commercial use only, subject to the following:
i) Non-Commercial use does not involve any form of monetisation, sponsorship or promotion.
ii) FMOD is distributed as integrated into a Product only;
iii) FMOD is not distributed as part of any Commercial Product or service;
iv) FMOD is not distributed as part of a game engine or tool set;
v) FMOD is not used in any Commercial enterprise or for any Commercial

Figure 18 fmod license 페이지 (https://www.fmod.com/resources/eula)

3 프로젝트 수행

3.1 프로젝트 수행 내용

3.1.1 점수화면 수정

ScoreMap()에 ScreenPrint(44, 6, "Press ₩'p₩' to Pause")를 추가하여 화면의 (44,6)좌표에 게임을 정지하려면 'p'키를 누르라는 말을 추가했다.

위에서 추가한 기능을 사용자들이 잘 확인할 수 있도록 SetColor(12)를 설정하여 빨간 글자로 보이도록 설정하였다. 그리고 SetColor(15)를 주어 원래 글자색인 흰색으로 글자 색을 다시 설정해 주었다.

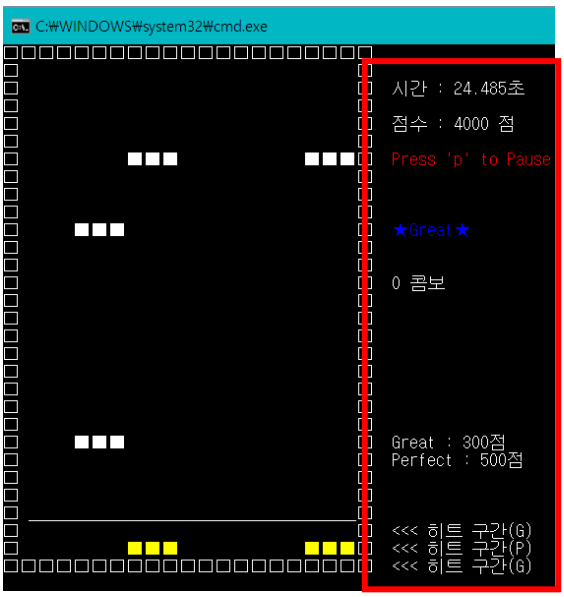


Figure 19 화면 상의 ScoreMap 출력

3.1.2 시작 화면 수정

ReadyMap(), ReadyMap1()는 아래 그림과 같이 게임의 첫 시작 화면을 출력해주는 함수인데 코드 리뷰를 했을 때 ReadyMap 과 ReadyMap1 함수의 분리를 한 눈에 알아 보기 힘들었기 때문에 후에 나와 같이 이 오픈 소스를 활용할 다른 개발자들을 위해 두 함수의 분리에 대한 간략한 설명을 주석으로 추가했다.

ReadyMap1() 함수는 “Press Enter to Start”를 출력해 플레이어에게 Enter 를 입력해 시작을 유도하는 문구인데 눈에 더 잘 띄게 보일 수 있도록 색을 추가했다. 색이 있는 문구는 원 개발자가 출력에 관한 함수를 저장해둔 Screen.cpp 에 SetConsoleTextAttribute(g_hScreen[g_nScreen-Index],color);를 이용해 SetColor 함수를 따로 만들어두었기 때문에 이 함수를 이용해서 편리하게 색을 추가할 수 있었다.

후에 플레이어가 싱크로율을 조절할 수 있는 함수를 따로 만들고 ReadyMap 에 사용자가 싱크로율을 조절 하는 화면으로 이동할 수 있도록 ‘Press c to Syncmap’ 문자열 출력을 추가했다.

Figure 20 화면 상의 ReadyMap, ReadyMap1 출력

3.1.3 게임의 결과 화면 추가

기존 코드의 Stage 구조체에는 {READY, RUNNING, PAUSE, RESULT} 4 개의 Stage 가 있었으나 코드 내의 switch case 문에는 RESULT stage 를 이용한 코드가 존재하지 않았고, 게임 완료 후에도 게임의 끝을 알려주는 인터페이스나 점수 화면같이 종료 화면을 출력해주지 않았기 때문에 Result-Map()을 구현해 게임완료와 획득 점수를 출력해주는 함수를 만들었다.

RUNNING 단계에서 52 초가 넘어가면 RESULT 단계로 넘어가야 하기 때문에 52 초가 넘어가면 Render()함수에 Stage 상태를 RESULT 로 바꿔주는 조건문을 추가한다. 사용자가 게임 결과를 볼 수 있게 하기 위해서 Switch 문에 RESULT 단계를 추가하여 노래가 끝난 시점인 52 초가 넘어가면 Result-Map()함수를 호출하여 게임 결과를 화면에 출력해준다.

게임이 종료된 후 다시 시작할 수 있는 기능을 추가했기 때문에 플레이어들에게 알리기 위한 문자열인 ‘Press enter to restart’를 추가했다.

ResultMap()에 Best Score 와 User Score 을 보여주는 랭킹 기능을 추가로 삽입하려 할 때, ‘GAME END ! ’와 점수를 출력하는 틀이 입력 받은 플레이어의 이름이 길어지면 틀 밖을 벗어나는 상황이 생기게 되어서 틀을 아예 지우기로 결정했다.

Figure 21 화면 상의 ResultMap 출력

3.1.4 채보 노트의 추가

오픈 소스 주제로 리듬 게임을 선택했을 계획단계에서 어떻게 음악에 맞는 채보 노트를 추가할 것인가에 대해 고민을 했었는데, 처음엔 Osu! 라는 다른 리듬 게임을 이용할 계획이었다. Osu!에서 내가 삽입한 음악에 맞춰 노트를 만들고 그 곡을 플레이할 수 있는데, 이렇게 해서 만들어진 노트를 텍스트 파일로 변환 후 그 파일을 읽어 노트를 NoteCheck()함수 내에 추가하거나 또는 NoteCheck()함수가 호출되었을 때 채보 노트가 담긴 텍스트 파일을 읽어오는 방식으로 구현하려 했으나 Osu!에서 만들 어진 노트 파일(.osz)이 암호화가 되어있어 읽을 수 없었기 때문에 음악을 듣고 직접 일일이 NoteCheck()내에서 노트를 작성하고 실행해 본 후 작동과 싱크로율을 확인하는 작업을 계속 반복해서 완성했다.

게임의 BGM 인 ‘유령의축제(Festival of Ghost)’를 재생하다 보면 비슷한 부분이 몇 번 반복 되는 것을 들을 수 있다. 이렇게 조금씩 반복 되는 부분이나 연속으로 키를 눌러야 하는 노트의 경우에는 for 문을 사용해서 NoteCheck 함수의 길이를 줄이도록 노력했다.

Figure 22 암호화 된 osz 파일

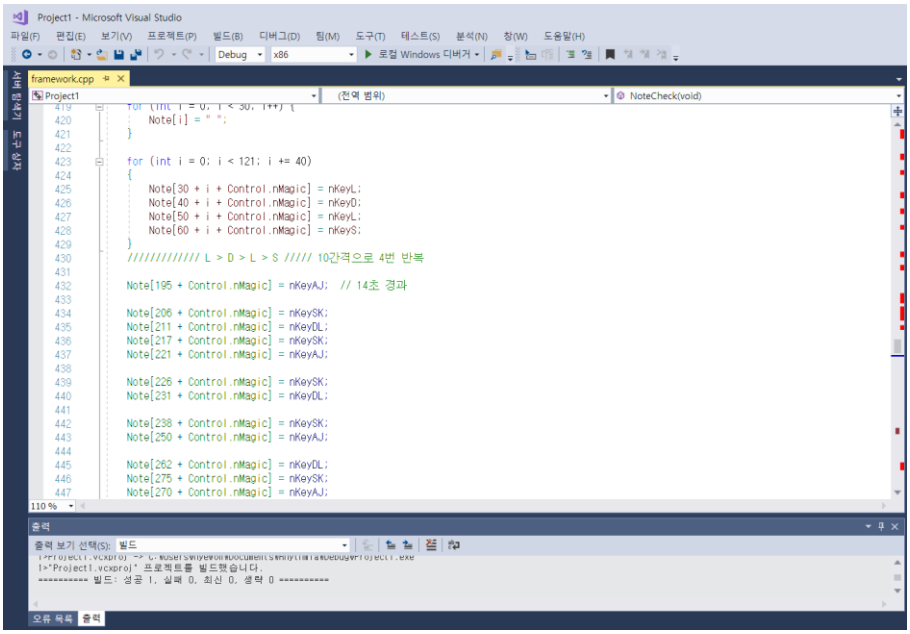


Figure 23 NoteCheck 함수

기존 소스 코드에서는 노트가 다소 단순한 편이었기 때문에 간격이 좁은 노트나 연속으로 계속 키를 누르고 있어야 하는 키들을 추가했지만, 연속으로 키를 누르는 노트의 경우 콘솔창에서 키를 눌렀을 때 getch()가 키를 입력 받는 작업이 조금 느린 편이기 때문에 코드를 작성할 때 예상한 점수와 실제 실행했을 때의 점수가 큰 격차가 생긴다. 예를 들어 5 개의 노트가 연속적으로 내려올 때 코드를 작성할 때 예상한 점수는 노트 5 개의 점수지만 키를 정확히 5 번 입력할 때만 예상 점수가 생기고 키를 떼지 않고 누른 상태에서 5 개의 노트가 끝난 경우에 키를 뗐을 경우에는 5 개보다 많은 점수가 입력 받기 때문에 이 부분 보완의 어려움으로 연속적인 키를 한 칸씩 띄우는 코드로 수정했다.

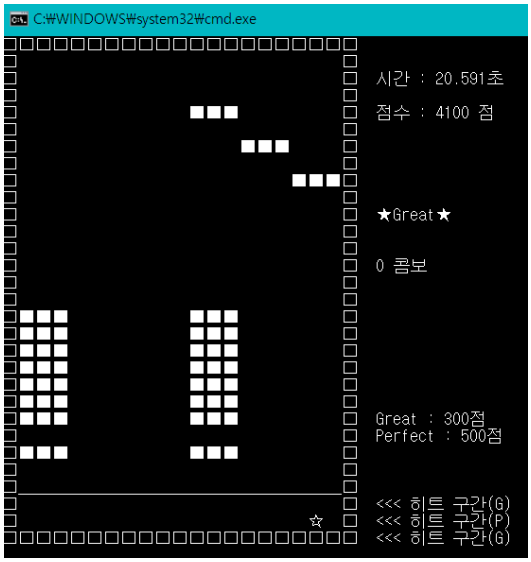


Figure 24 연속 노트 수정 후

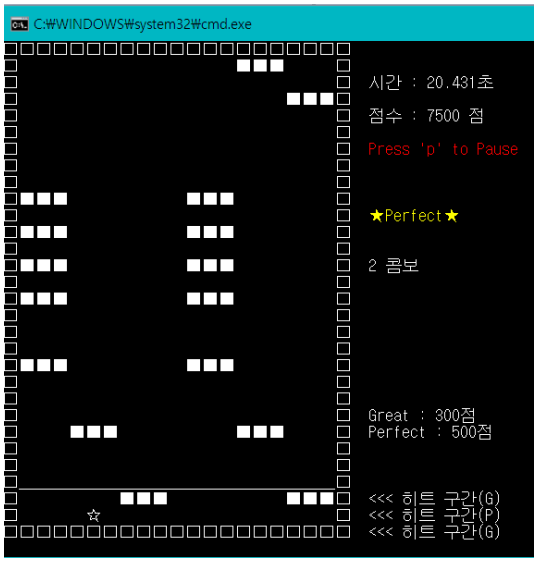


Figure 25 연속 노트 수정 전

음악에 맞춰 노트를 히트 구간에 잘 맞게 하도록 코드를 거듭 수정해서 싱크로율을 맞췄으나 컴퓨터 환경에 따라 미묘하게 느려지거나 다시 실행하면 또 원래 속도로 돌아오는 등 노트가 내려오는 속도가 실행할 때 마다 미세하지만 조금씩 바뀌었기 때문에 결국 플레이어가 싱크로율을 직접 조정할 수 있도록 하는 화면과 기능을 추가하기로 결정하였다.

NoteCheck()는 데이터를 초기화하는 Init(), 데이터를 갱신하는 Update()함수에서 각각 한번씩 호출된다. Init()함수에서는 모든 노트를 공백으로 초기화하는 코드가 있는데(for(int i = 0; i 〈 ALLNOTE; i++) { Note[i] = “ ”; }), 기존 코드는 NoteCheck()가 위의 공백으로 초기화하는 코드의 앞에서 호출됐기 때문에 악보를 불러 놓고서 그것을 다시 공백으로 초기화 해버리는 결과가 만들어졌다. 따라서 for 문의 앞에 있던 NoteCheck()를 뒤쪽으로 옮겨서 노트를 불러올 수 있도록 수정했다.

Update()함수에서는 마지막에 NoteCheck()를 호출하는 부분이 있지만 이미 Init()함수에서 NoteCheck()를 호출해 초기화한 상태여서 불필요하게 컴파일되는 부분이라고 판단해서 삭제했다. 삭제 후 실행해 봤을 때 문제없이 실행됐다.

최종적으로 코드를 정리하고 가독성이 좋은 코드로 바꾸면서 ‘NoteCheck’라는 이름이 적합한 것 같지 않다고 판단해서 NoteInit()으로 이름을 바꾸었다.

3.1.5 두 개의 키를 처리

기존에는 두 개의 노트가 떨어질 때 한 개만 일치해도 점수와 콤보가 올라갔다. 이를 보완하기 위하여 소스트리를 사용하여 코드를 수정해서 두 개의 노트에 대한 입력 처리 기능을 추가했다.

CheckA, CheckS, CheckD, CheckJ, CheckK, CheckL, CheckAJ, CheckSK, CheckDL 함수에서 중복되는 내용을 합친 CheckKey 함수를 선언했다. 기존에 ShowNote 에서 Check, Check1, Check2 에 Note[curNoteIndex], Note[curNoteIndex -1], Note[curNoteIndex +1] 배열을 각각 대입하여 현재 입력 키와 같은 지 확인하는 방식에서 Check, Check1, Check2 변수를 제거하고 Note 배열과 직접 확인하도록 하였다. Note.h 에 입력 키 string, 키에 해당하는 노트 string 을 포함하는 구조체를 선언한 후에 노트 종류 개수에 해당하는 구조체 배열 KeyIndex 를 선언하였다. 그리고 framework.cpp 에서 각 입력 키에 해당하는 노트 string 들로 초기화를 해준 후에 GeyKeyType 이라는 함수를 선언하여 인자로는 입력 키 string 을 받아서 이것과 같은 string 을 구조체 배열에서 찾아서 노트 string 을 반환해주도록 했다. main 함수에서 키 입력을 받으면 이것을 string 으로 변환한 뒤에 CheckKey 의 인자로 주었다. Note[curNoteIndex], Note[curNoteIndex +1], Note[curNoteIndex -1]이 두 개의 노트를 가진 문자열이라면 사용자로부터 키 입력을 하나 더 받을 수 있다. 키를 입력 받으면 CheckKey 의 인자로 먼저 입력받았던 키 string 변수에 더한 후에 넘겨준다. 이 때에 알파벳 오름차순으로 문자열을 배치했다. CheckKey 는 이 넘겨받은 string 과 KeyIndex 구조체 배열의 inputKey 와 같은 string 을 찾아서 해당 구조체 인덱스의 nKey 와 현재 떨어지고 있는 Note 배열과 같은 지 확인하여 perfect 구간, great 구간을 처리해준다.

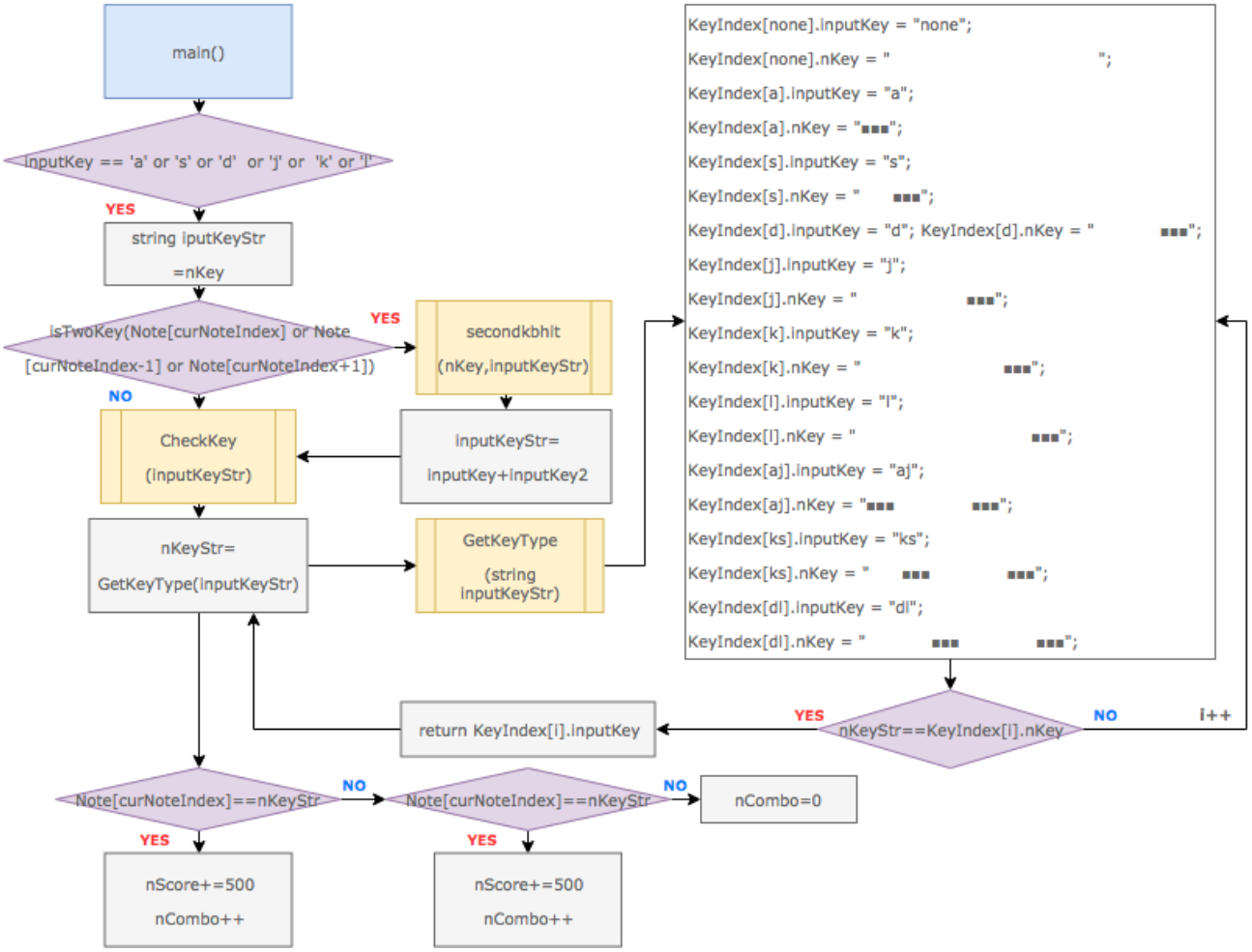


Figure 26 두 개의 키 입력 flow – chart

3.1.6 일시 정지 기능의 추가

기존에는 Pause 기능이 구현되지 않아서, P 를 눌러도 Pause 기능을 사용할 수 없었다. 이를 수정하기 위해서 P 를 눌렀을 때, 노트와 노래가 일시 정지를 할 수 있도록 코드를 수정했다.

우선 해야 할 일은 3 가지였다. P 를 눌렀을 때 (1) 노트가 멈춰야 하며, (2) 시간도 멈춰야 하고, (3) 노래가 멈춰야 했다. 그래서 노트가 멈춰야 하기 때문에, Stage 가 PAUSE 일 때는 노트가 떨어져선 안 된다. 따라서 Render() 함수에서 Stage 가 PAUSE 일 때 return 을 통해 Render() 함수에서 빠져나온다. 여기서 break 를 하지 않는 이유는, break 를 했을 경우에는 Screenflipping()함수로 인해서 화면이 보여지는데, Switch 문 밑에 있는 Screenflipping()을 호출하지 않기 위해서 return 을 해줬다.

이제 Enter 를 누르면 게임 화면으로 돌아와야 한다. 돌아왔을 때 노트는 정상적으로 내려와야 하며, 게임 플레이 시간도 다시 진행되어야 한다. 하지만 여기서 문제점은 게임 플레이 시간을 구하는 방식에서 정지 시간을 고려하지 않는다는 것이다.

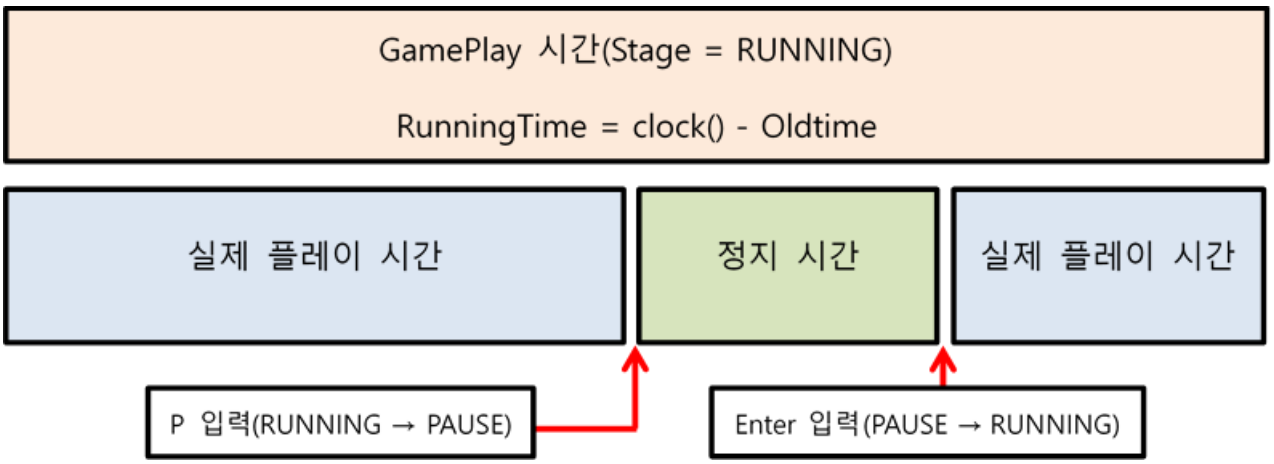


Figure 27 PauseTime 계산

게임에서 플레이 시간은 clock() 함수를 이용해서 구하기 때문에 게임 플레이 시간인 RunningTime 은 Curtime - Oldtime(Enter 를 누르기 전 대기 시간)이라는 공식을 따른다. 하지만 실제로는 P 가 입력되고 Enter 가 눌릴 때까지 즉 PAUSE 일 때는 RunningTime 에서 고려하면 안된다. 따라서 RunningTime 에 PauseTime 을 빼주는 공식을 사용하기로 했다. 이때 PauseTime 은 P 가 눌려서 Pause 상태가 되었을 때의 시간인 PauseStart 와 Enter 를 눌러 다시 RUNNING 상태가 되었을 때의 시간 PauseEnd 를 이용해서 PauseEnd - PauseStart 로 구해준다. 이때 한 게임에서 Pause 기능을 하는 것은 여러 번이 될 것이기 때문에 PauseTime += PauseEnd - PauseStart 이라는 식을 이용해서 정지된 시간을 계속해서 누적을 해줘야 한다.

하지만 일시 정지 기능에서 가장 중요한 기능인 노래를 멈췄다가 다시 재생하는 기능을 구현하는 것은 불가능 했다. 왜냐하면 현재 이 RhythmGame 오픈소스에서 사용하는 mmsystem 은 하나의 트랙으로 노래를 재생하고, 중지하는 기능밖에 하지 못한다. 이것의 한계를 느끼고 노래를 일시 정지할 수 있는 기능을 사용하기 위해서 “fmod”라는 라이브러리를 사용하고자 했다.

“fmod”는 VisualStudio 프로젝트에 다양한 음향 기능을 제공해준다. 특히 일시 정지 기능을 구현하기 위해 확장된 음향 기능을 위해서 “fmod”를 사용하기로 했다. 하지만 이 fmod 의 기능들은 대부분 클래스를 사용하는 C++을 지원하기 때문에 C 로 이루어진 RhythmGame 오픈소스를 고칠 필요가 있었다.

가장 먼저 framework.c, Note.c, Screen.c 파일을 cpp 로 바꿨다. 이때 문제가 생겼다. c 에서는 암묵적으로 문제가 되지 않던 문법이 C++로 바뀌면서 문제가 되었기 때문이다. (1) 예를 들어서 char *와 const char*는 같지 않지만 기존 프로젝트에서 C 문법일 때는 모든 문자열의 초기화를 쌍 따옴표로 정해진 const char*로 대입했다. 이러한 문제는 char*를 C++ 문법인 string 클래스로 바꿔서 선언하고, 인자 타입도 string 클래스로 바꿨다. (2) 또한 기존에서는 conio.h 에서 제공하는 _kbhit(), _getch() 기능을 사용을 conio.h 를 include 하지 않고 사용하고 있었기에, conio.h 를 include 해줬다. 이러한 문제들을 해결하고 난 뒤 실행은 문제없이 됐다.

이제 “fmod”를 사용하기로 했다. 우선 선언을 하기 위해서 준비해둔 FMOD 폴더에 있는 .hpp 파일, .h 파일, lib 파일, dll 파일이 필요했다. 이제 이 파일들을 각자 로컬에 존재하는 Visual Studio 프로젝트에 경로를 입력을 해줘야 하는데, 방법은 다음과 같다.

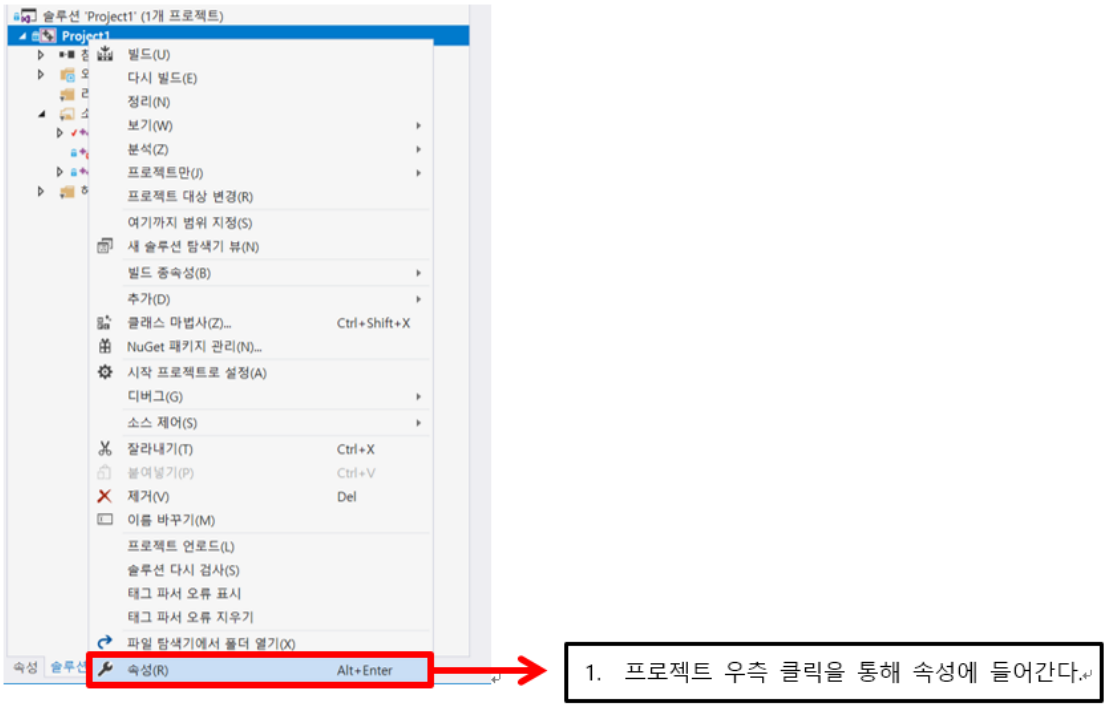


Figure 28 프로젝트 속성

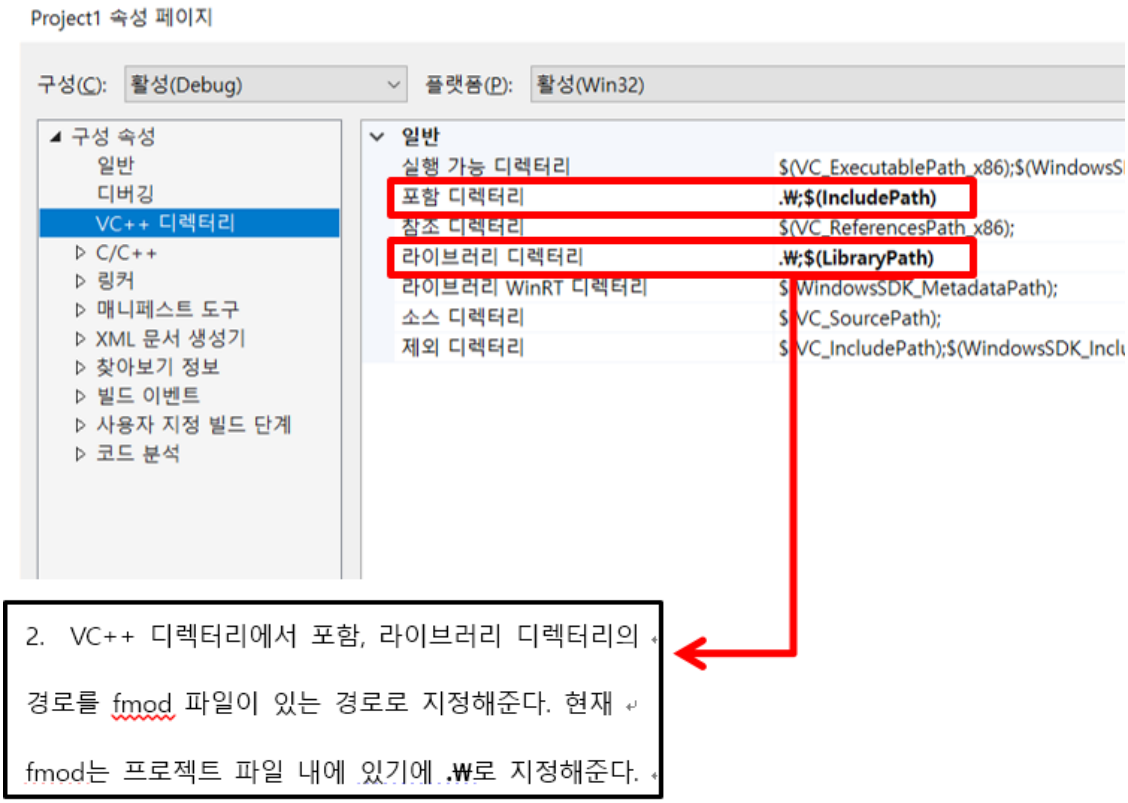


Figure 29 VC++ 디렉터리 경로 지정

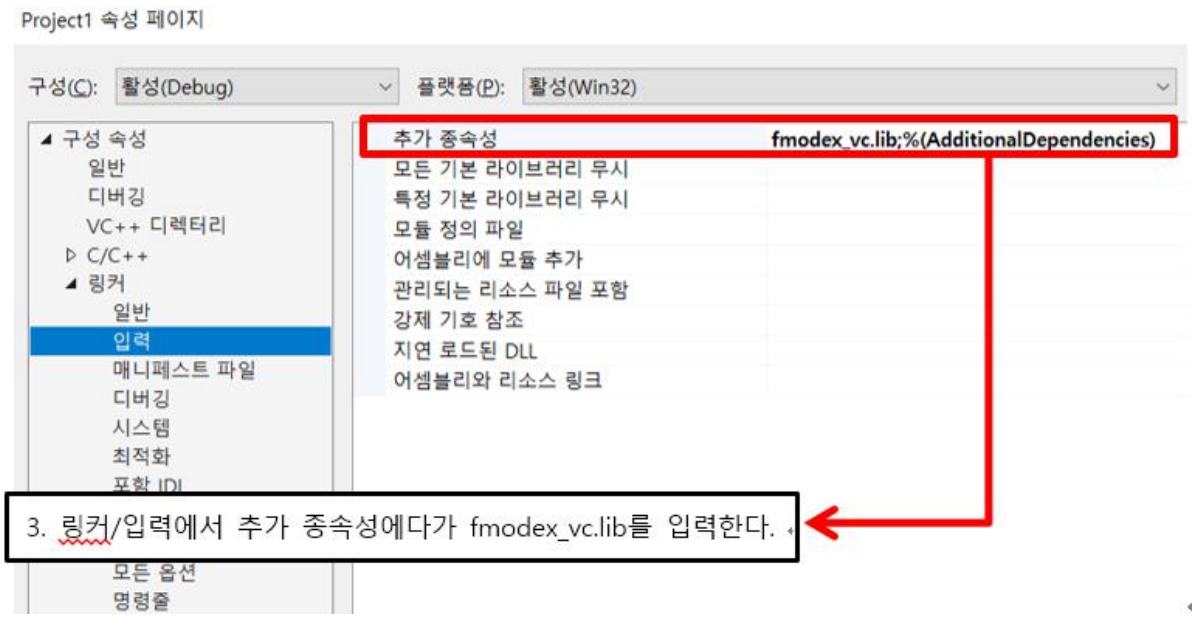


Figure 30 종속성 추가

이러한 방식을 통해서 fmod를 프로젝트와 연결해줄 수 있다. 이때 만약 dll파일이 없다는 오류가 생긴다면 fmodex.dll 파일을 C:₩Windows₩System32 와 C:₩Windows₩SysWOW64 에 넣어주면 오류는 사라지게 된다. 이제 fmod 와 project 가 연결되었기 때문에 fmod 를 사용할 수 있게 됐다. “fmod.hpp”를 include를 하고 기능을 사용해보기로 했다.

가장 먼저 사용 준비를 해야 한다. fmod 사용 가이드는 포털 사이트를 통하면 쉽게 접할 수 있다. 따라서 가이드를 따라서 아래의 변수 선언과 함수 정의에 사용된 명령어들을 사용할 수 있다.

```
// 사운드 변수
System* pSystem;
Sound* pSound[2];
Channel* pChannel[1];


//사운드 함수
void SoundSystem() {
        System_Create(&pSystem);

        pSystem->init(4, FMOD_INIT_NORMAL, NULL);

        pSystem->createSound("opening.wav", FMOD_LOOP_NORMAL | FMOD_HARDWARE, NULL, &pSound[0]);
// 오프닝음악
        pSystem->createSound("Festival_of_Ghost.wav", FMOD_DEFAULT, NULL, &pSound[1]);
// 게임음악
}
void Play(int Sound_num) {

        pSystem->playSound(FMOD_CHANNEL_FREE, pSound[Sound_num], 0, pChannel);

}
```

Figure 31 fmod 변수 선언과 함수 정의

선언된 pSystem 은 fmod 시스템을 할당해준다고 생각할 수 있다. 따라서 System_Create(&pSystem)을 통해 pSystem 을 생성해주고 init 을 통해 초기화를 해준다. 그리고 pSystem 을 이용해서 pSound 에다가 참조할 노래 경로와, 노래의 타입(루프인지, 아닌지)를 결정해준다. 마지막으로 Play 함수에서 pSound 를 인덱스로 접근해서 해당 노래를 재생시킨다.

이렇게 선언된 pSystem, pSound, pChannel 은 fmod 의 음향 기능을 담고 있다. 그 중 pChannel[0]->setPaused(bool) 기능을 이용해서 현재 pChannel 에서 재생중인 pSound[Sound_num] 노래를 정지하고자 한다. setPaused()에서 인자로 true 를 넣는다면 pChannel 에서 재생중인 노래의 상태를 일시 정지로 만들어준다. 반대로 false 를 넣어준다면 일시 정지 상태가 풀리게 된다. 이것을 이용해서 main() 에서 만약 p 를 눌렀을 경우 setPause(true)를 해주고, 다시 Enter(=₩r)를 눌렀을 때는 setPause(false)를 해주면서 노래를 일시 정지할 수 있게 됐다.

3.1.6 히트 구간 판단

기존에 있던 Game 에서는 Note 가 Hit 구간에 왔을 때 키보드를 누르면 Perfect, Great 인지 판단해준다. 하지만 판단했을 때 사용자에게 가시적으로 눈에 띄지 않았다. 그래서 Hit 가 됐을 때 가시적인 기능을 해주는 HitMap() 함수를 구현했다.

우선 HitMap()의 기능은 떨어지는 노트 밑에 있는 Hit 구간에 ☆를 출력해 Hit 를 판단하는 기능을 넣기로 했다.

Figure 32 ☆ 를 출력하는 히트 구간

하지만 이 기능은 키보드가 눌린 것에 대한 가독성은 좋지만, Note 가 Hit 됐는지에 대한 여부는 모호했다. 또한 두 개의 노트가 내려올 때 별이 두 개가 나오지 않아서 한계가 있었다. 따라서 HitMap()보단 노트가 Hit 됐을 때, Note 를 변환시켜주는 HitNote() 기능을 구현하기로 했다.

기존에 있던 Game 에서는 Note 가 Hit 구간에 왔을 때 키보드를 누르면 Perfect, Great 인지 판단해준다. 하지만 판단했을 때 사용자에게 가시적으로 눈에 띄지 않았다. 그래서 Hit 가 됐을 때 가시적인 기능을 해주는 HitMap()함수를 구현했다.

가장 먼저 Note 를 Hit 를 했을 때, CheckNote() 함수에서는 현재 키 입력을 받았을 때, Note 의 위치를 비교해서 Perfect 와 Great 을 판단한다. 따라서 판단을 해서 Hit 가 됐을 때 Hit 가 된 노트를 변환시켜주는 HitNote() 함수를 구현했다.

```
string HitNote(string inputKey) {
        for (int i = 0; i < inputKey.length(); i++) {
                if (inputKey[i] != ' ') {
                        inputKey[i] = 'oo';
                }
        }
        return inputKey;
}
```

Figure 33 Hitnote 함수

현재 Hit 가 된 노트인 inputKey 를 이용해서 인덱스로 접근한다. 그리고 접근한 인덱스가 ‘ ‘가 아닐 때, 즉 ■ 일 때 그 인덱스 자리를‘oo’로 바꿔준다. 바뀐 Note 는 return 을 시켜 Note 배열에 있는 해당 인덱스의 Note 를 변경한다. 그래서 Hit 가 된 Note 를 사용자에게 더욱 가시적이게 바꿀 수 있었다. 추가로 Hit 구간에 들어갔을 때, Note 가 노란색으로 변하도록 수정하였다.

Figure 34 리듬 게임 실행 중 히트 구간

3.1.7 싱크로율 조절

노트들은 모두 배열로 선언되어 있으며 시간에 따라 노트 배열의 인덱스를 증가시키며 화면에 보여준다. 따라서 혹시라도 유저의 컴퓨터의 성능이 떨어지거나 다른 프로그램을 이용하고있어 배열을 읽는 속도 가 떨어진다면 미리 정의해 둔 싱크가 맞지 않을 수 있다. 따라서 그것을 조율 할 수 있는 변수 Control.nMagic 을 조정 해 줌으로 서 싱크를 바꿀 수 있다. Control 구조체에 존재하는 정수형변수 .nMagic 을 바꾸게되면 배열의 인덱스가 nMagic 만큼 밀리게 된다. 따라서 nMagic 이 증가하게 된다면 노트들은 더욱 늦게 내려 올 것이고 반대로 nMagic 이 감소하게 된다면 노트는 빠르게 내려 올 것이다.

사용자는 reddymap 에서 'C' key 를 입력 받게 되면 싱크를 조절 할 수 있는 SyncMap 이 나타나게 된다. 사용자는 화살표를 통하여 싱크를 조정 할 수 있다. Stage 가 Ready 상태 일 때 c 키를 누르게되면 stage 를 SYNC 로 바꾸며 SyncMap 함수를 호출한다. SyncMap()은 ScreenPrint() 함수를 사용한다. ScreenPrint(x 좌표 , y 좌표 , 문자열) 이 함수는 좌측 상단을 기준으로 x 좌표와 y 좌표에 문자열을 표시하는 함수이다. SyncMap() 이 호출되면 화면에 싱크를 조절할 수 있는 텍스트가 나오며 int 형 변수인 Syncnum 을 string 변수로 바꾸어주어 화면에 출력 해 준다. 또한 Stage 가 SYNC 일때

사용자는 좌 우 방향키를 이용하여 syncnum(-30 ~ 30) 을 수정 할 수 있는데 이 syncnum 의 값을 nmagic 에 넣는다.

```
Note[206 + Control.nMagic] = nKeySK;
Note[211 + Control.nMagic] = nKeyDL;
Note[217 + Control.nMagic] = nKeySK;
Note[221 + Control.nMagic] = nKeyAJ;

Note[226 + Control.nMagic] = nKeySK;
Note[231 + Control.nMagic] = nKeyDL;

Note[238 + Control.nMagic] = nKeySK;
Note[250 + Control.nMagic] = nKeyAJ;

Note[262 + Control.nMagic] = nKeyDL;
Note[275 + Control.nMagic] = nKeySK;
Note[270 + Control.nMagic] = nKeyAJ;
Note[274 + Control.nMagic] = nKeyDL;    // 22초

Note[290 + Control.nMagic] = nKeySK;

Note[298 + Control.nMagic] = nKeyDL;
Note[302 + Control.nMagic] = nKeySK;
Note[306 + Control.nMagic] = nKeyDL;
Note[310 + Control.nMagic] = nKeySK;
Note[314 + Control.nMagic] = nKeyAJ;
```

Figure 35 NoteInit 함수 내에서의 nMagic

nMagic 을 바꾸게되면 배열의 인덱스가 nMagic 만큼 밀리게 된다. 따라서 nMagic 이 증가하게 된다면 노트들은 더욱 늦게 내려 올 것이고 반대로 nMagic 이 감소하게 된다면 노트는 빠르게 내려 올 것이다.

사용자는 readymap 에서 'C' key 를 입력 받게 되면 싱크로율을 조절 할 수 있는 SyncMap 이 나타나게 된다. 사용자는 화살표를 통하여 싱크를 조정 할 수 있다.

Figure 36 싱크로율 조절 화면

3.1.8 재시작 기능 추가

Restart 기능 구현 : 게임이 끝나고 끝나는 일회성 게임이 아니라, 다시 시작을 할 수 있는 게임으로 만들고자 했다. 따라서 Result 화면으로 넘어갔을 때 특정 키를 입력했을 때 다시 게임 준비화면으로 넘어갈 수 있게끔 코드를 수정했다.

Restart 기능을 추가할 때, 우선 반복되는 기능을 생각했다. 계속해서 이 게임이 반복되기 위해선 모든 기능을 포함하는 반복문이 필요하다고 생각했고 main() 안에 있는 모든 기능을 while(1)로 묶었다. 그 후 게임이 끝났을 때, 즉 Stage 가 RESULT 일 때, 특정 키(Enter)를 입력했을 때 다시 게임 준비화면으로 넘어갈 수 있게, 키 입력을 받는 부분에서 Stage == RESULT 일 때, Stage 를 READY 로 바꾸고 Break 를 했다. 따라서 Stage 가 READY 로 바뀌고 기존에 있던 while 문을 빠져나가기 때문에 READY 에 상응하는 게임 화면이 다시 시작됐다. 그리고 ResultMap()에서 보었여주는 문구에 Press Enter to Restart를 넣어주면서 Enter를 눌렀을 때 다시 시작을 할 수 있다는 메시지를 알려줄 수 있게 됐다.

또한 추가해야 하는 기능은 다시 시작을 하지 않고 게임을 끄는 경우를 구현했다. 특정 키(q)를 눌렀을 때, 게임을 끝낼 수 있게끔 했다. 그래서 enum 에 END 라는 Stage 상태를 넣고, END 일 때 나갈 수 있게끔 게임을 반복할 수 있게 선언했던 while(1)에 있는 조건문에 (Stage != END)를 넣어줬다.

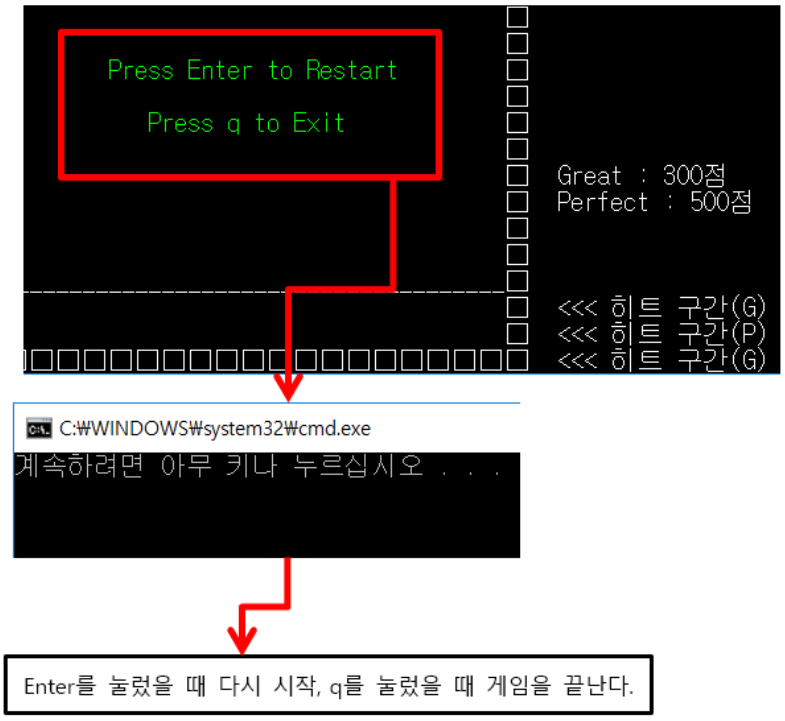


Figure 37 Restart 함수 구현

3.1.9 랭킹 시스템 구현

기존 Game 은 User 의 점수만 출력해줬다. 하지만 다시 시작 기능이 추가 됐기 때문에 현재까지의 User 점수중 Best 점수를 구해서 출력해준다.

Figure 38 저장되어 있는 BestScore

Figure 39 처음 게임 실행 시 랭킹 화면

우선 다시 시작될 때마다 초기화가 되는 User 점수와는 다르게 갱신되기 전까지 변하지 않는 BestScore 가 필요했다. 또한 화면에 출력되기 위해선 string 형 변수가 필요했기 때문에 User 의 Score 를 저장해주는 UserScore 와, BestScore 를 저장해주는 BestScore 를 선언했다. UserScore 는 nScore 를 이용해서 구하며 만약 nScore 가 현재 BestScore 보다 넘어갔을 경우에는 BestScore 에 있는 점수를 nScore 로 교체한다.

3.2 시행착오

3.2.1 오랫동안 Pull 을 하지 않아서 생긴 오류

한 팀원(안재현)이 branch 를 만들어서 싱크로율 조절 기능을 만드는 중에 여러 개의 push 가 이루어 졌고, 원래 작업 하던 것은 'new' branch 에서 commit 을 해놓은 후에 master branch 에서 pull 을 했다. 그리고 작업하던 branch 를 병합하려고 하니 충돌이 많은 곳에서 일어났고 수동으로 코드를 수정하고 병합을 성공시킨 후 commit 을 해둔 뒤에 다시 pull 을 했고, 그 사이에 다른 팀원(유지인)이 한 개의 함수를 고친 commit 을 push 했는데 또 많은 충돌이 일어났다. 수동으로 코드를 수정하는 과정에서 현재 원격 branch 의 코드 내용과 많이 달라진 것으로 추정되었다.

그래서 git bash 에서 ‘git reset –merge ORIG_HEAD’ 명령어를 이용하여 merging 중인 것을 해제하고 master branch 에서 fetch 의 hard 옵션을 통해 팀원(안재현)의 master 브랜치를 원격 브랜치와 같도록 만들었다. 그리고 master branch 의 코드에 싱크 조절 작업을 하던 ‘new’ 브랜치의 코드 내용을 직접 끼워 넣는 작업을 한 뒤에 push 를 하여 해결하였다.

프로젝트 수행을 시작한지 얼마 되지 않을 때, 강의시간에 교수님의 파일로 연습하는 것이 아니라 실전에서의 Pull 과 Push 에 익숙해진 상태가 아니어서 벌어진 실수로, 처음에는 새로운 레포지토리를 만들어 다시 clone 해 시작하려 했다. 그러나 git bash 에서 명령어로 해결했기 때문에 새 레포지토리는 C 에서 C++로 업그레이드할 때처럼 코드에 큰 변화가 생길 때 종종 모의로 실험을 하는 레포지토리로 남겨두었다.

3.2.2 fmod 디렉터리의 변경

fmod 라이브러리의 사용을 위해 C 에서 C++로 업그레이드 했을 때, fmod 라이브러리 파일 중 fmodex.dll 이 push 를 하려는 팀원(차봉호)의 Sourecetree 내의 stage 에 올라가지 않았다. 필요한 fmod 파일이 모두 있어야만 음악 재생을 할 수 있기 때문에 fmod 라이브러리를 압축한 파일을 push 했다. 이 때 디렉터리 경로의 설정을 절대경로가 아니라 상대경로로 설정하였다. 다른 팀원들이 로컬 PC 에서 pull 하고 작업할 때 압축 파일을 풀고 실행하면 되는데, 한 팀원(장혜원)이 작업 후 다시 push 를 할 때 이 압축을 푼 파일들을 다시 함께 push 를 하는 실수를 해서 디렉터리 경로가 변경되는 상황이 발생했다. 이 실수로 디렉터리 복사를 필수로 자주 해야 한다는 깨달음을 얻었다.

3.3 버전 관리

Rhythm Ta! 리듬 게임은 다음과 같이 버전을 나누었다.

표 1 Rhythm Ta! 리듬 게임의 버전

커밋	작성자	태그	설명
3510e0b	IMHYEWON	V1.0.0	초기 상태
04e3b7d	bongho cha	V2.0.0	C->C++로 변경
6df3c05	bongho cha	V2.1.0	일시 정지 기능 추가
Fc3384d	JINYEBAP	V2.1.1	노트 string 변수 중복 선언 오류 수정
1505a4f	jaehyun Ahn	V2.2.0	Hit 된 노트에 대한 표시 기능 추가
Bc134d8	jaehyun Ahn	V2.2.1	노래 길이 수정
169e404	bongho cha	V2.2.2	Hit 된 노트에 대한 표시 오류 수정
88aa7be	JINYEBAP	V2.3.0	두 개의 키 입력 기능 추가
B9b31d2	jiin yoo	V2.4.0	게임 종료 후 최종 점수를 띄워주는 결과 기능 추가
215a014	jaehyun Ahn	V2.5.0	싱크로율을 조절할 수 있는 기능과 화면 추가
C1f81d8	IMHYEWON	V2.5.1	키 입력 인식이 힘든 연속된 노트들 수정
1af9ee5	bongho cha	V2.6.0	게임이 끝난 후에 다시 시작 기능 추가
C06abd6	jaehyun Ahn	V2.6.1	싱크로율 조절 키 설명 추가, 화면 위치가 맵을 벗어나지 않게 조절
5ab3a48	bongho cha	V2.7.0	게임이 끝난 후에 게임 종료 기능 추가
bcbadda	bongho cha	V2.8.0	랭킹 기능 추가
a3f6fe9	bongho cha	V2.9.0	콤보에 대한 점수 누적 기능 추가
42fa72b	bongho cha	V2.9.1	콤보 누적 오류 수정

4. 프로젝트 수행 소감

장혜원

나는 팀원들과 다르게 소프트웨어 설계 기초 강의를 수강하지 않았고 함께 팀 프로젝트를 수행한 경험이 없기 때문에 완성된 소프트웨어 소스코드를 리뷰하거나 팀원들과 협업하면서 작업한 경험이 이번이 처음이었다. 1 학년 때 학교에서 튜터의 지도로 가계부 프로그램을 만든 적이 있었는데, 그 때는 Github 나 소스트리같이 팀원들과 함께 작업할 수 있는 방법들을 몰랐기 때문에 카카오톡으로 디렉토리를 통째로 보내가며 작업을 하느라 굉장히 번거로웠던 기억이 있다. 프로젝트를 시작하면서 내가 push 를 하면 다른 팀원들의 작업과 충돌이 나거나 오류를 불러 일으킬까 두려워서 익숙해지는데 조금 시간이 걸렸지만 적응하고 나니 팀원들의 커밋 메시지를 확인하면서 쉽게 작업 내역을 확인하고 내가 수정한 부분과 병합할 수 있었다. 코드를 리뷰하면서 굉장히 허술한 리듬 게임이라고 생각하고 내가 맡은 함수 부분에서 어떻게 이런 계산이 나왔는지 이해가 되지 않았다. 그런데 내가 프로젝트를 본격적으로 작업하면서 내가 맡은 부분을 직접 수정하고 보완하니 그제서야 원 개발자의 의도와 계산이 보였고, 그 때 내가 했던 코드 리뷰가 겉핥기 식이었다는 것을 깨닫고 반성했다. 그렇지만 나는 처음 코드를 읽는 입장에서도 충분히 파악이 쉬운 코드가 되기를 원했기 때문에 팀원들과 함께 코드를 보완하고 함수를 추가할 때는 함수 이름이나 주석에 대해 충분히 고민하고 작성했다.

안재현

이번 오픈소스를 수업을 수강하면서 리눅스 및 깃에 대해서 왜 배우는지 이해가 되지 않았다. 하지만 이번 과제 5 를 수행하며 깃이 왜 필요하며 프로젝트를 다른 협업자들과 할 때 아주 중요한 역할을 한다는 것을 알게 되었다. 깃을 처음 사용하면서 오류도 많았다. 다른 협업자와 내가 고친 코드가 충돌되었을 때 이상한 점을 눈치채지 못하고 억지로 push 를 하여 다른 협업자들에게 피해를 주었다. 그에 대해서 마스터 브랜치를 제가 커밋한 곳까지 되돌린 후 수동으로 병합 한 뒤 push 했다. 또한 pull 을 오랜 시간동안 하지 않으면서 나 자신도 커밋 및 push 를 하지 않아서 병합하는데 시간이 많이 걸렸습니다. 이러한 오류가 많은 이유는 지금 우리가 채택 한 이 코드는 framework.cpp 에 많은 코드들이 연관되어있어 많은 협업자들이 이 framework.cpp 를 고치게 되었는데 이 구조는 Github 를 완전히 이용하지 못했다고 생각한다. 만약 코드들이 객체화 되어 협업자들이 각자 개별적으로 맡게 된다면 충돌도 얼마 없을 것이며 깃 허브를 완전히 이용하게 될 것 이라고 생각한다.

차봉호

Git 을 사용해보면서 가장 큰 장점은 프로젝트를 수행하는데 있어서 서로 간의 작업 내역이 뚜렷하고 관리가 쉬웠다는 것이다. 2 학년때 들었던 소프트웨어설계기초 강의와 비교할 때, 그 당시에 서로 만나서 입으로, 노트로, 메신저로 나누던 작업 내용들은 무척 복잡하고 관리가 힘들었다. 이미 정해놨던 기능들에 대해서 헷갈리고, 작업하는 프로젝트도 제출용 프로젝트, 테스트용 프로젝트 등으로 다양해지면서 관리가 힘들었다. 하지만 Git 을 사용하면서 Branch 로 인한 작업 분할을 하고 master 로 병합을 하면서 작업했던 내용들을 합치면서 과거에 겪었던 어려움들은 느껴지지 않았다. 이번에 처음 배우는 기능이라서 사용하데 겁이 많았지만, 사용에 익숙해지면서 익숙해졌다. 하지만 가장 큰 문제점은 Git 사용을 편하게 하기 위해서 사용했던 소스트리를 잘 사용하지 못한 것이었다. 소스트리를 잘 사용하지 못해서 현재 clone 했던 폴더를 편하게 다시 옮기면서 소스트리가 그것을 찾지 못하면서, 북마크를 삭제하고, 그러다 보니 다시 클론을 하게 되면서 지금까지 사용하던 branch 가 사라지게 되었다. Git bash 에서는 Clone 한 폴더 자체를 옮기는 것은 문제되지 않았지만 소스트리는 참조하는 경로가 정해져 있었기 때문에 문제가 생겼다. 이러한 문제를 토대로 더욱 조심하고, 소스트리의 경로를 재설정 해주는 기능을 찾아 다시는 이런 일이 일어나지 않도록 경각심을 가졌다. 또한 Git 을 사용하지 않았을 때는 프로젝트를 넘겨 받았을 때, 수정된 내용을 일일이 찾아야 했는데 Git 을 사용할 때는 변경된 내용을 확인할 수 있고 커밋된 내용을 토대로 어떤 업데이트가 생겼는지 판단할 수 있었다. 이러한 장점들을 토대로 개발자에게 있어 Git 과 GitHub 는 뗄 수 없는 것이라고 생각했으며 오픈소스 수업에서 배운 내용을 바탕으로 더욱 공부를 해서 전보다 발전된 프로젝트 수행을 할 것이라는 목표가 생겼다.

유지인

오픈 소스를 활용해서 리듬 게임을 만들어보았다. 작년에도 소프트웨어 설계 기초라는 과목에서 게임을 만들었었는데 그때는 오픈 소스를 사용하지 않고 아예 처음부터 우리가 구현했었다. 이번에 오픈 소스를 활용해서 게임을 만들어보니 이미 구현되어있는 코드를 우리가 수정하고, 추가시키면서 완성시키는 것이었기 때문에 훨씬 편리했고, 이미 구현되어있는 코드가 어떤 식으로 구성되어있는지 배울 점도 많았고 미비한 부분, 코드가 깔끔하지 않은 부분은 다시 구현해 보면서 코드에 대해 더 생각해 볼 수 있었다.

수업시간에는 Git Bash 를 사용했는데 우리는 소스트리를 사용해서 Github 과 Visual studio 2017 을 연동해서 사용해보았다. Visual studio 2017 에서 코드를 수정하면 소스트리에 수정된 내용으로 코드가 변경되어 있었다. 소스트리에서도 조원들이 한 작업을 pull 하거나 내가 한 작업을 push 하면 코드가 모두 바뀌어있었다. 이러한 점이 각자 수정한 파일을 서로 주고 받고 하지 않아도 돼서 편리했고, 커밋 메시지를 작성했기 때문에 주석을 달아서 내가 어떤 기능을 추가했는지 따로 설명을 하지 않아도 돼서 편리했다. 하지만 두 사람이 모두 pull 을하고 있어서 오류가 발생하는 경우도 있고, 처음 사용해봤기

때문에 내 마음대로 push 했다가 모든 조원들의 코드가 바뀌는 일이 발생하는 경우도 있었기 때문에 신중하게 작업했다. 소스트리를 사용해 보면서 점점 익숙해 졌지만, 아직도 미숙한 점이 많다. Git 과 Github 은 앞으로도 여러 사람들과 협동해서 코딩을 하기 위해서는 중요한 것이기 때문에 이번 과제로 그치지 않고, 앞으로 더 알아보고 사용하면서 익숙해져야겠다고 생각했다.

진예진

지난 학기에 소프트웨어 설계 기초 강의에서 4 명이 함께 게임을 만들어 보았었다. 그 때에는 한 명이 코드 작성을 끝내면 다음 사람이 넘겨 받아서 코드를 수정했어야 했고, 두 명이 동시에 고치면 병합이 어려웠다. 그래서 코드를 작성하는 데에 드는 시간보다 어느 부분을 수정했는지, 코드의 오류가 무엇인지를 찾는 시간이 훨씬 오래 걸렸다. 하지만 git 과 git hub 의 사용으로 그러한 시간들을 단축할 수 있었다. 어느 부분이 변경되었는지 라인 단위로 표시되어서 쉽게 알 수 있었고 브랜치라는 것이 존재해서 공통으로 사용하는 master 브랜치 말고 각자의 로컬 브랜치에서 작업을 할 수 있어서 여러 명이 같은 시간에 작업을 할 수 있는 것이 가장 편리했다. 그리고 로컬 내에서도 내가 원하는 작업마다 브랜치를 만들어 commit을 하여 기록을 남기고, 다시 되돌릴 수 있는 기능이 잘못 수정했을 때에 굉장히 유용했다. 그리고 full 할 때에 병합에서 원격 master 브랜치와 로컬 master 브랜치와 충돌이 발생하면 어느 부분에서 충돌이 났는지 표시되고 그 과정에서 다른 사람이 수정한 부분이 더 효율적인 코드일 경우에는 직접 수정하여 그 코드를 선택할 수 있는 점이 좋았다. 그리고 다른 부분을 고쳤을 경우에는 병합이 알아서 되는 점도 편했다. 협업에 있어서 git 과 Github 는 꼭 필요한 환경이라고 생각한다.

5. 참고문헌 및 사이트

3 FMOD(2018), https://www.fmod.com/

4 네이버 블로그(2012), API 에서 FMOD 사용법, http://blog.goldface.kr/31