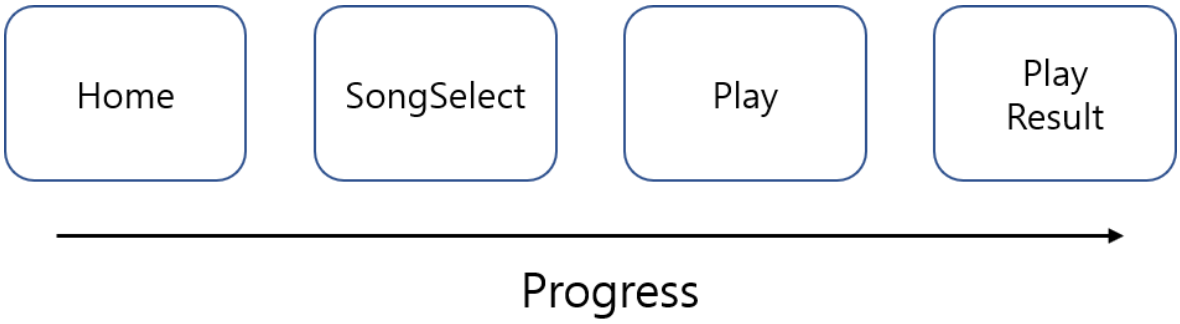
소개 Intro

유니티 리듬게임 Unity RhythmGame

노트타입 리듬게임입니다.

Github URL : https://github.com/LHEALP/UnityRhythmGame

씬 진행도 Scene Progress



Home : 첫 실행 시 나타나는 타이틀 씬

SongSelect : 플레이 할 음악을 선택하는 씬

Play : 노트가 생성되고 처리되는 씬 (※ core)

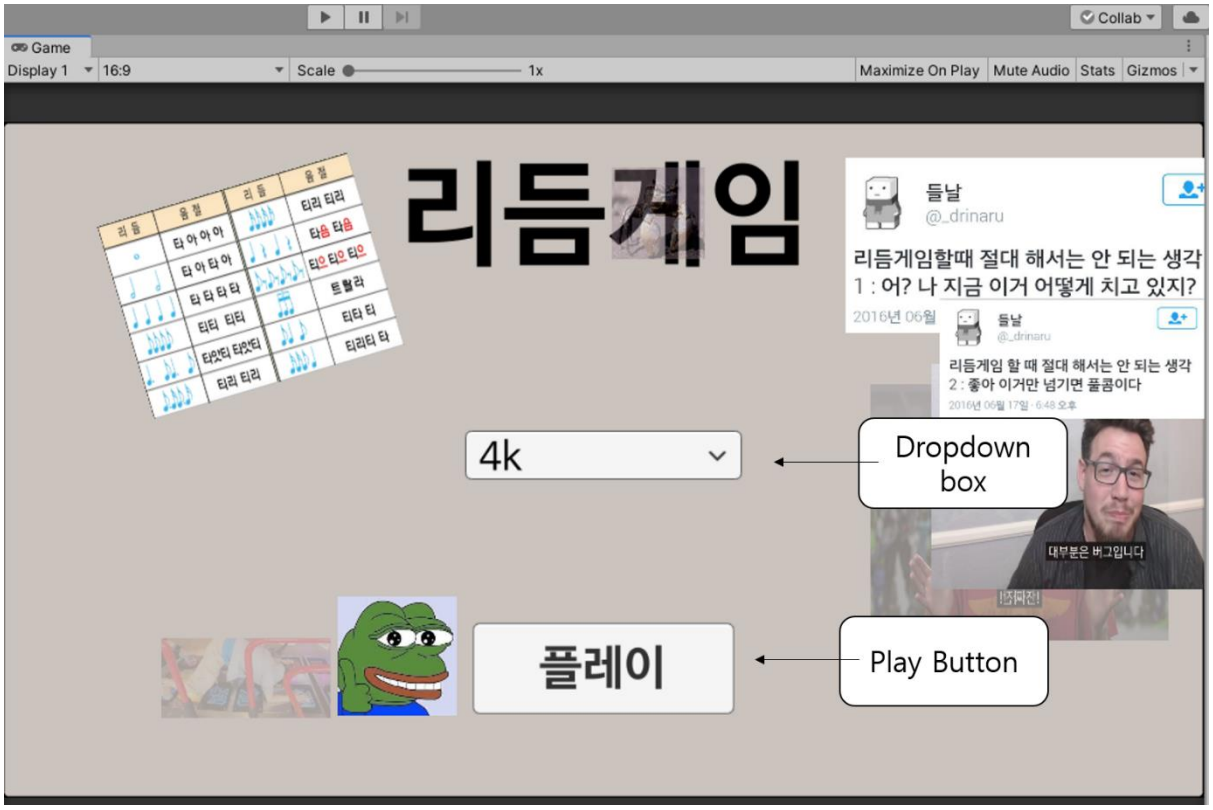
PlayResult : 플레이 결과(점수)를 보여주는 씬

※ readme

유니티 리듬게임을 학습하기 위해서..

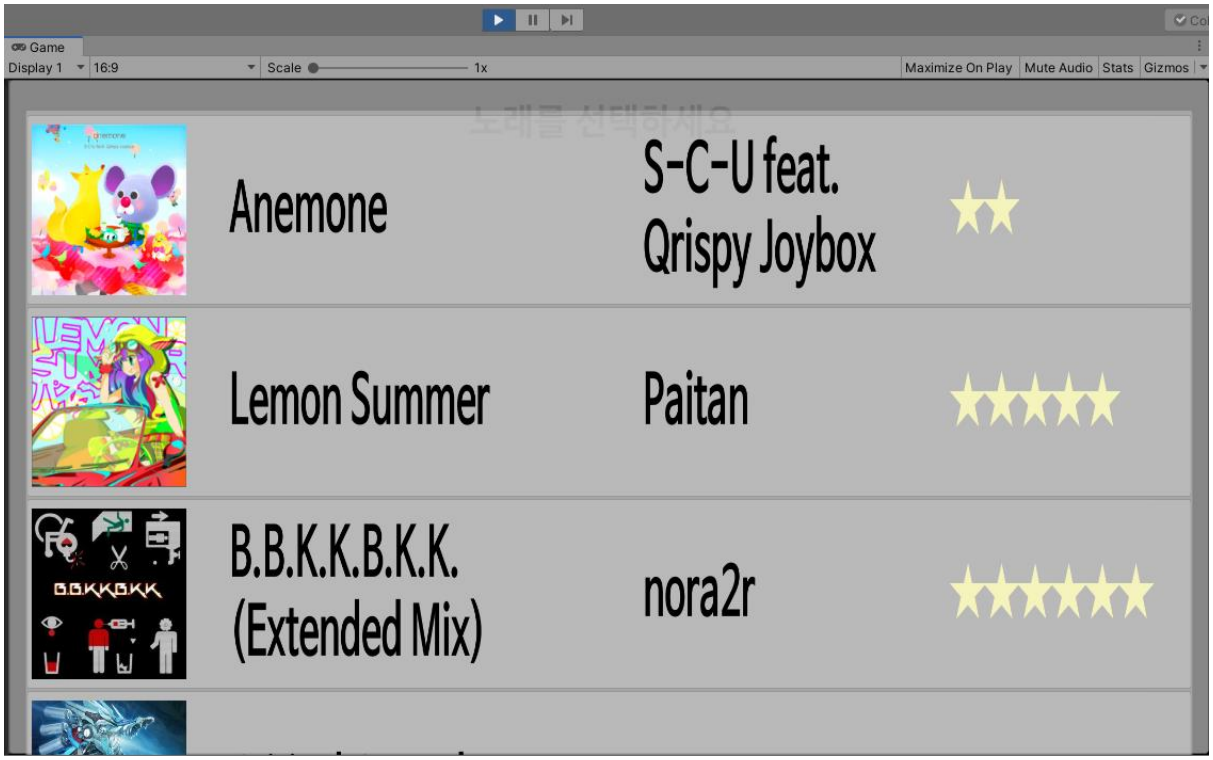
"Play를 제외한 씬들은 UI이기 때문에 중요하지 않습니다.
그러므로 간략하게 설명하고 넘어가도록 하겠습니다.
이해가 되지 않는 부분이 있다면 GitHub 페이지에 방문해서 글을 남겨주세요."

Home Scene



플레이할 키의 개수를 선택하고 플레이 버튼을 눌러 진행합니다.

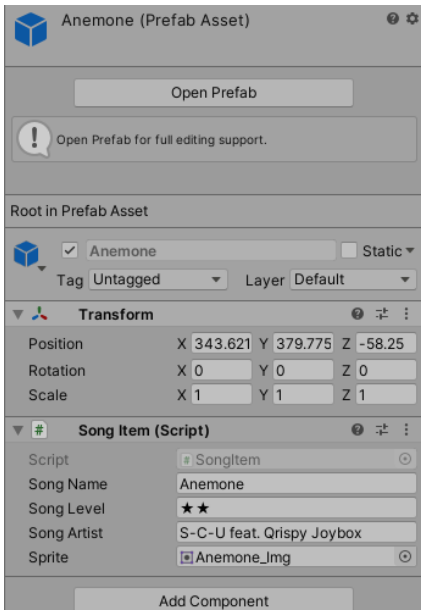
SongSelect Scene



원하는 음악를 선택할 수 있습니다.

마우스 스크롤을 통해 여러 개의 음악들을 확인할 수 있습니다.
항목을 한번 클릭하면 음악의 미리듣기가 가능하며, 한 번 더 누르면 Play 씬으로 넘어가게 되어 게임을 할 수 있습니다.
Listview 형태를 나타내고 있으며 https://youtu.be/GpPWbM_6DVY 의 도움을 받아 만들었습니다.

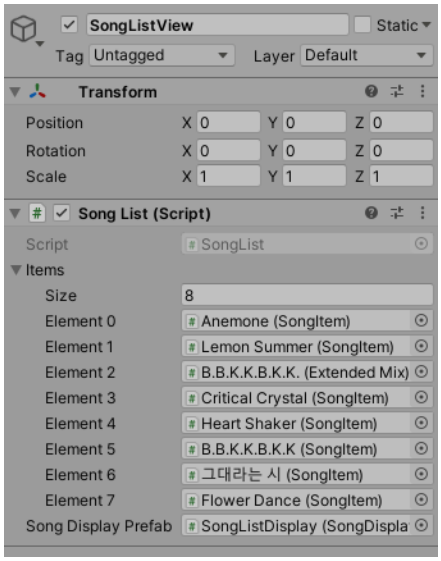
리스트뷰에 음악을 추가하는 방법



빈 게임오브젝트를 만들고 SongItem 스크립트를 추가하여 음악의 데이터를 적고 프리팹화 합니다.

이때 SongName 항목은 파일을 읽을 때 사용하기 때문에 **정확히 작성해야** 합니다.

Ex) Anemone -> Anemone.mp3, Anemone_data.txt

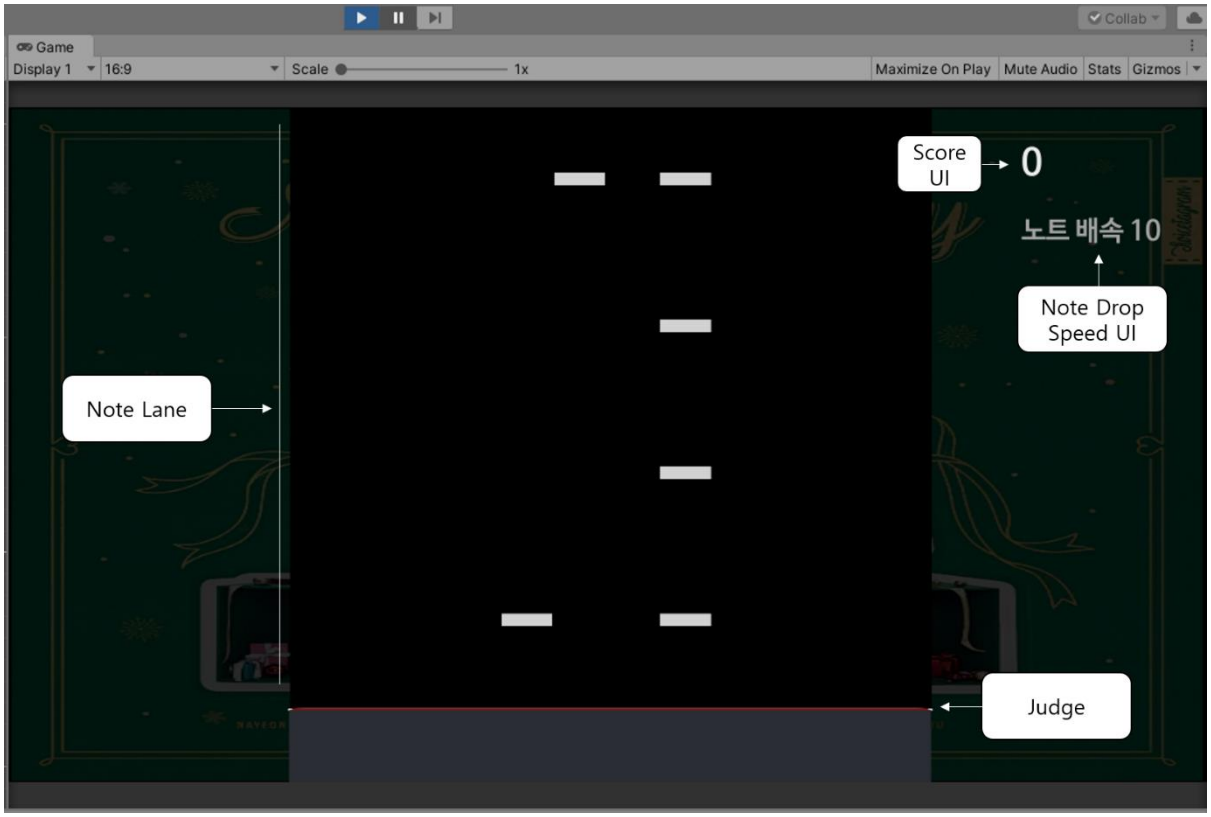


프리팹화 된 오브젝트를 SongListView 오브젝트의 SongList 스크립트에 추가합니다.

마지막으로 Resources 폴더에 SongName에 작성한 내용과 똑같은 폴더를 생성하고 음악파일은 SongName.* / Sheet(Note) 파일은 SongName_data.txt / 이미지 파일은 SongName_Img.* 확장자의 * 마크는 유니티에서 지원하는 확장자를 말합니다. (mp3, wav, jpg, png ....)

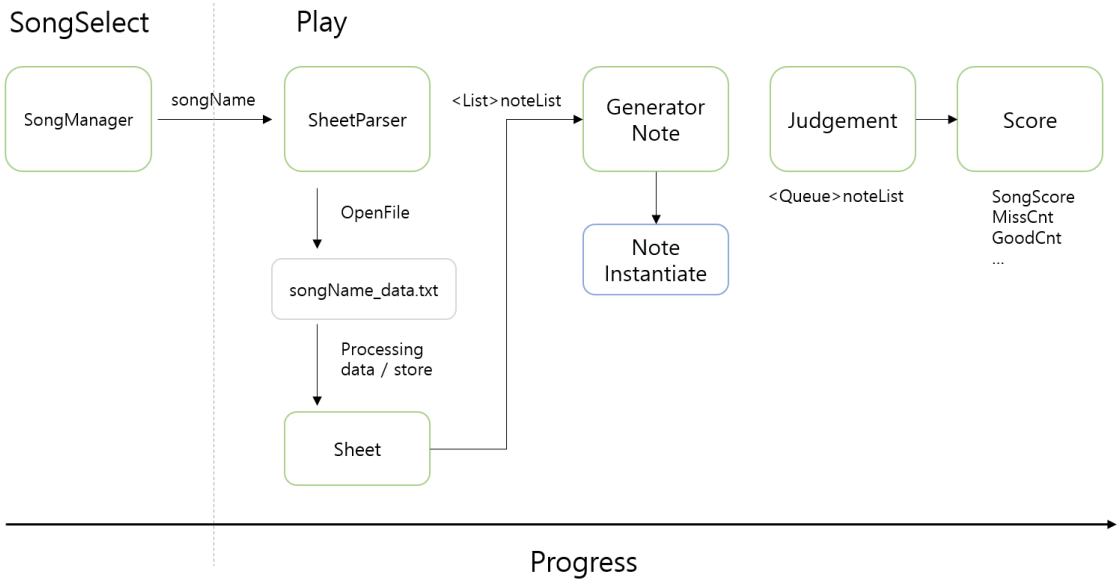
기회가 된다면 이 시스템을 자동화하는 것을 추천합니다.

Play Scene



곡이 선택되면 나타나는 화면입니다.

아래는 대략적인 진행구조를 나타냅니다.



※ green: script / gray: text file / blue: prefab

노트를 파싱하기 앞서 OSU!의 데이터 형식을 참고하여 만들었기 때문에 설명 드리겠습니다.

예시) Whisper of the summer night(acoustic ver.)_data.txt

```
[SheetInfo]
AudioFileName=Whisper of the summer night(acoustic ver.).mp3
AudioViewTime=30
ImageFileName=Whisper of the summer night(acoustic ver.)_Img
BPM=71
Offset=0.3396
Beat=44
Bit=32
Bar=80
```

[SheetInfo]

음악의 전반적인 내용을 가지고 있습니다.

```
[ContentInfo]
Title=Whisper of the summer night(acoustic ver.)
Artist=SHK
Source=SHK
Sheet=LHEALP
Difficult=Lv.2.5
```

[ContentInfo] 게임에서 표기될 내용을 가지고 있습니다.

```
[NoteInfo]
448,192,3396,1,0,0:0:0:0:
64,192,4452,1,0,0:0:0:0:
192,192,4663,1,0,0:0:0:0:
320,192,4874,1,0,0:0:0:0:
448,192,5086,1,0,0:0:0:0:
64,192,6142,1,0,0:0:0:0:
192,192,6353,1,0,0:0:0:0:
320,192,6565,1,0,0:0:0:0:
```

[NoteInfo] 노트의 위치를 가지고 있습니다.

읽는방법

x, y, time, notetype, ...

여기서 사용될 데이터는 x 값과 time입니다. 이 외의 값은 파싱은 하지만 사용되지 않거나 파싱을 하지 않습니다.

Time은 밀리세컨드로 표기하며 330은 0.33초를 말합니다.
이 값은 노트를 판정하는데 쓰일 뿐만 아니라 노트가 최초 생성되고 떨어지는 위치를 결정하기도 합니다. 이 시간 값들을 기록하는 방식은 BPM의 개념이나 메트로놈을 만들어본 적이 있다면 어렵지 않게 이해할 수 있습니다.

(4/4 기준) 60BPM은 1초에 한 박을 재생한다고 볼 수 있습니다. 4개의 노트가 1초에 한 번씩 떨어진다고 한다면 1000, 2000, 3000, 4000으로 기록할 수 있습니다. (1초 = 1000밀리세컨드)
Anemone곡의 경우 200BPM이기 때문에 한 박의 단위는 0.33초 입니다. 이를 밀리세컨드로 변환하면 330이 됩니다.

노트파싱 NoteParsing

SheetParser.cs가 SongSelect 씬의 SongManager.cs 로부터 선택된 음악의 이름을 전달받으면 해당 파일을 Resources.Load하여 TextAsset 변수에 저장합니다. 그리고 텍스트파일을 한줄씩 읽으면서 필요한 데이터들을 구분하여 Sheet.cs 의 변수에 저장합니다.
일반적인 파싱의 원리와 특별히 다른 것은 없습니다.

노트 Note

Note.cs(NotePrefab)는 아래를 향해 떨어지거나 노트의 속도를 변경하는 역할만 합니다.
이 스크립트는 노트의 판정에 전혀 관여하지 않습니다. 그저 화면에 보여주기만 하고 판정은 백그라운드에서 따로 처리됩니다.

노트생성 NoteGenerator

GeneratorNote.cs 에서 Sheet.cs의 noteList에서 데이터를 한 개씩 불러옵니다. (foreach)
※ noteList 에는 노트의 시간 값이 저장 되어있습니다.

앞서 말했듯이 시간 값은 노트가 생성될 위치로도 사용됩니다.
코드의 일부분을 살펴보겠습니다.

GeneratorNote.cs ... IEnumerator GenNote()

```
...
foreach(float noteTime in sheet.noteList1)
            Instantiate(notePrefab, new Vector3(2.3f, noteStartPosY + sync.offset +  notePosY
* (noteTime * noteCorrectRate)), Quaternion.identity);
...
```

Vector3의 Y값에 적혀있는 변수들은 원하는 위치에 생성될 수 있게 보정하는 역할을 합니다.
여기서 noteTime에 noteCorrectRate를 곱해주는 이유는 아래와 같습니다.

noteStartPosY + sync.offset, notePosY * noteTime = too high Y Pos!!
330,1200,5000, ...

noteTime * noteCorrectTime = moderate Y Pos!!
330 * 0.001 = 0.33

싱크 Sync

Sync.cs는 노트의 싱크를 담당합니다. 눈여겨봐야 할 변수는 아래와 같습니다.
frequency, offset, scrollSpeed 정도입니다. 나머지는 PlayTik() 코루틴을 위한 것 또는 현재 프로젝트에서 크게 사용되는 부분이 아닙니다.

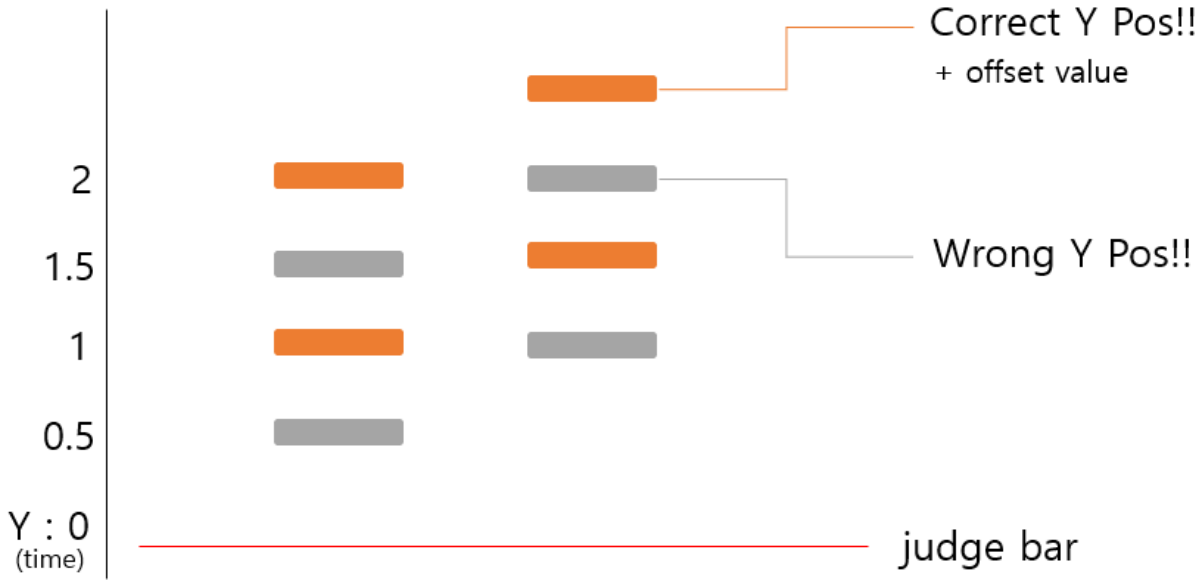
frequency : 현재 선택된 음악의 PCM 값을 가져옵니다. 이 것은 중요하기 때문에 노트판정 파트에서 설명하겠습니다.

offset : 동일한 음악의 파일을 여러 개 가지고 있다고 가정합니다. 이 음악파일들은 서비스 제공자가 음원을 제공받아 가공하는 과정에서 음악의 첫 비트가 시작되는 지점이 전부 다르게 설정되기 때문에 이 변수가 존재합니다. 예를 들어, 어떤 사람은 음악 파일을 실행하자 마자 재생되는 것을 원치 않아 오디오의 맨 앞에 1초의 무음 더미를 삽입했다고 합시다. 텍스트 파일에 기록된 노트들의 시간 값들은 이를(더미) 고려하지 않았기 때문에 offset 값이 필요합니다. 이해를 쉽게 하기 위해서 아래 예제를 봅시다.

```
[NoteInfo]
64,192,500,1,0,0:0:0:0:
320,192,1000,1,0,0:0:0:0:
320,192,1500,1,0,0:0:0:0:
64,192,2000,1,0,0:0:0:0:
```

위의 노트 데이터를 파싱해서 노트를 생성했다고 합시다.

음악파일의 오프셋이 0일 때는 별다른 조치를 하지 않아도 되지만 0.5초의 더미가 삽입된 파일의 경우 해당 값만큼 노트를 생성할 때 Y값에 추가를 해줘야 싱크가 일치합니다.



scrollSpeed : 노트들의 간격을 지정합니다. 이는 플레이어가 스크롤 속도를 키를 눌러 원하는 속도로 바꿀 수 있습니다.

키 입력 Input Key

InputKey.cs에서는 키보드 입력들을 담당하고 있습니다. 이 게임은 4개의 노트를 입력 받기 때문에 D, F, J, K를 입력 받고 있음을 코드에서 확인할 수 있습니다. 키를 누를 때마다 노트를 판정하기 위해서 판정 클래스의 노트판정 메소드를 호출합니다. 그리고 스크롤속도를 바꾸기 위해 F5, F6키를 사용하고 있는 것을 볼 수 있습니다. 키를 눌러 속도를 조절하면 Note 태그를 가지고 있는 게임 오브젝트를 찾아 Y 포지션을 변경하고 노트가 내려오는 속도도 변경됩니다.

노트판정 Note Judgement

노트판정을 다루기 앞서 먼저 이해해야 할 주제가 있습니다.
싱크 파트에서 언급한 frequency 값입니다.

사진에서 동그라미 쳐진 값을 말하는데, 이는 1초에 44100번의 음원조각들을 재생한다는 의미이다. (음원의 frequency 값은 AudioClip.frequency 변수를 통해 받아올 수 있다.)

그리고 음악이 얼만큼 진행했는지 알 수 있는 보다 상세한 값을 AudioSource.timeSamples 변수를 통해 받아올 수 있다. 이를 이해하기 위해 아래 예제를 확인합시다.

44100Hz의 음악파일이 재생된 지 2초가 지났다면 timeSamples 변수는 44100Hz에 2를 곱한 88200Hz를 반환합니다.
참고자료 https://docs.unity3d.com/kr/2019.3/ScriptReference/AudioSource-timeSamples.html

나는 음원이 가지고 있는 PCM 샘플 값을 이용하여 판정을 처리했다.
먼저 판정을 인정해주는 범위를 지정해야 한다. 이 또한 샘플 값에 기반하고 있다.
Judgement.cs

```
... // 44100 = 1초
float greatRate = 3025f; // 3025 ≒ 0.06초
float goodRate = 7050f; // 7050 ≒ 0.15초
```

```
float missRate = 16100f; // 16100 ≒ 0.36초
…
```

이 값들은 자신이 원하는 만큼 조절해서 입력하면 된다.

판정에 현재 재생중인 음악의 timeSample 값이 필요하기 때문에 Update() 안에서 선언한다.

```
…
currentTime = songManager.music.timeSamples;
…
```

이하 코드들을 분석할 때 판정할 노트의 시간 값은 모두 1000(1초)라고 가정합니다.

미스판정 (노트가 판정바 아래로 내려갔을 때)

```
…
if (judgeLane1.Count > 0) // 큐에 데이터가 더 이상 존재하지 않으면 실행하지 않음
{
    currentNoteTime1 = judgeLane1.Peek(); // 데이터를 빼내지 않고 보기만 한다.
    currentNoteTime1 = currentNoteTime1 * 0.001f * songManager.music.clip.frequency;
    // 현재 노트 시간값을 PCM Sample값으로 변환하는 방법
    // 노트의 시간 값이 1000(1초)일 때 위 수식을 실행하면 44100이란 값이 나온다.

    if (currentNoteTime1 + missRate <= currentTime)
    // 44100 + 16100 <= currentTime(현재 음악의 재생 위치)
    // = 1초 + 0.36초
    {
          judgeLane1.Dequeue();
          score.ProcessScore(0);
    }
}
…
```

나머지 판정들

```
…
// 현재 노트 시간이 현재 시간에서 판정 범위를 계산한 것보다 크거나 작으면 해당 파트 판정

if (currentNoteTime1 > currentTime - missRate && currentNoteTime1 < currentTime + missRate)
// 노트 체킹 범위 : 범위를 지정해주지 않는다면 순식간에 모든 노트들을 데큐해버릴 수 있다.
그 범위는 현재 음악의 재생위치 ± 0.36초
{
     if (currentNoteTime1 > currentTime - goodRate && currentNoteTime1 < currentTime +
goodRate)
     {
        // 그레잇판정 ± 0.15초
        if (currentNoteTime1 > currentTime - greatRate && currentNoteTime1 < currentTime +
greatRate)
        {
              judgeLane1.Dequeue();
              score.ProcessScore(2);
```

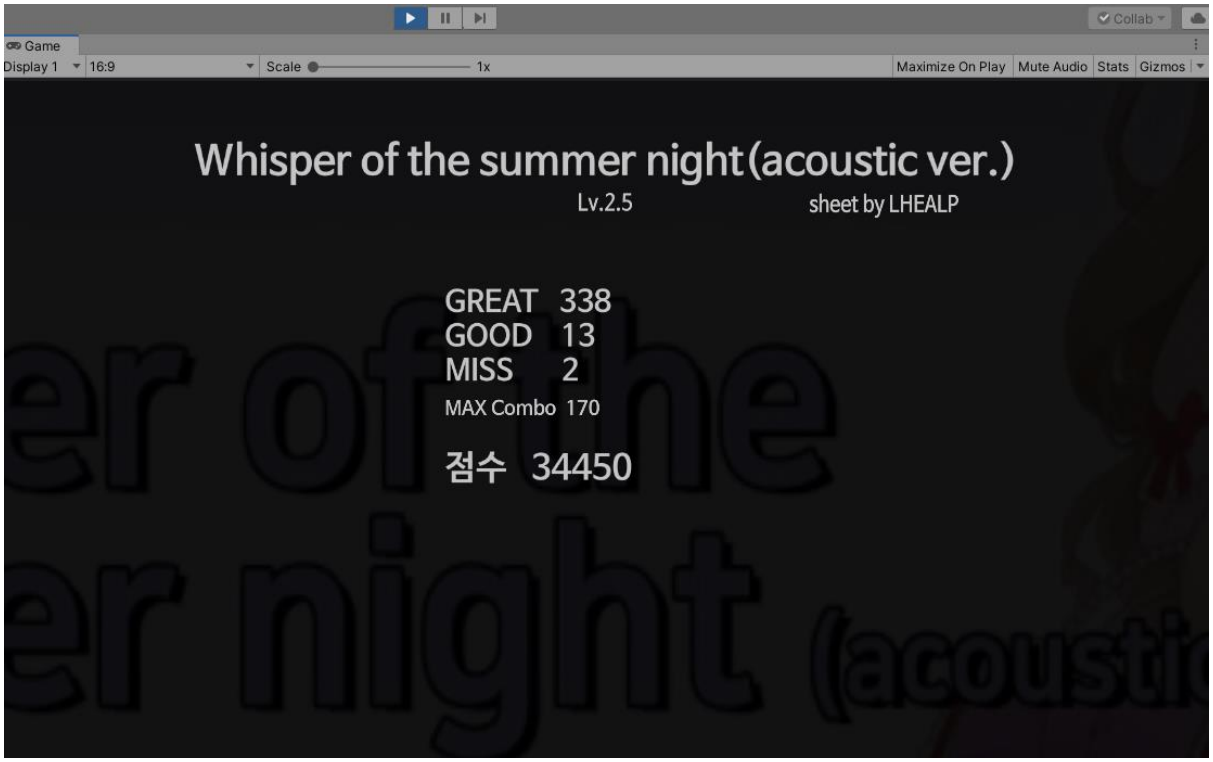
```
        }
        // 굿판정 ± 0.06초
        else
            {
                judgeLane1.Dequeue();
                score.ProcessScore(1);
            }
        }
        // 너무 빨리 입력했을때 미스처리( 키보드를 막 눌러서 노트를 처리하는 것을 방지한다 )
        else
        {
              judgeLane1.Dequeue();
              score.ProcessScore(0);
         }
    }
...
```

판정할 때 시간 값이 ± 인 이유는 노트를 빠르게 치는 것뿐 아니라 늦게 치는 것도 동일한 판정을 내리기 위해서입니다. 이는 리듬게임을 즐겨한다면 알 수 있는 정보이므로 자세한 설명은 생략합니다.

점수 Score

아마도 이제까지 적힌 내용 및 코드를 이해했다면 설명할 필요가 없으니 하지 않겠습니다.

PlayResult Scene



플레이가 끝나면 결과 씬으로 전환되며 획득한 스코어를 확인할 수 있습니다.