**캡스톤 디자인 (1)**

**개별 보고서**

|  |  |
| --- | --- |
| 팀 | 4조 |
| 학과 | 컴퓨터 공학부 |
| 학번 | 201901531 |
| 이름 | 가수한 |

**0. 서론**

현대 게임 산업에서는 창의적이고 독창적인 게임 경험을 제공하여 플레이어들을 매료시키는 것이 중요한 과제 중 하나로 여겨지고 있다. 우리 팀은 이러한 관점에서 리듬과 로그 라이크 장르를 융합한 혁신적인 게임을 개발하기로 하였다. 이번 프로젝트를 통해 리듬 게임의 음악적 즐거움과 로그 라이크 게임의 전략적인 흥미를 결합하여 새로운 장르를 창출하고자 했다.

리듬 게임은 주로 음악적 감각을 요구하며, 로그 라이크 게임은 전략적 사고를 필요로 하는 특성을 가지고 있다. 이는 각각의 장르를 즐기는 데에 특화된 플레이어를 만들어내는 경향이 있다. 그 결과, 게이머들은 한 가지 장르에서만 플레이하고 그 한계를 경험하게 된다. 이러한 문제점을 극복하고자, 우리는 리듬과 전략의 조화로운 조합을 지향하는 게임을 기획하였다.

뿐만 아니라, 캡스톤 디자인 수업에서는 기존 시스템에 개선된 기술과 창의성을 적용하는 것이 핵심 과제이다. 이에 우리 팀은 Unity 엔진을 활용하여 리듬과 로그 라이크의 특성을 융합하는 게임을 개발하고, 창의적이고 독창적인 경험을 통해 기존의 게임 시스템에서의 한계를 극복하고자 하였다.

캡스톤 디자인 수업에서는 기존 시스템에 개선된 기술과 창의성을 적용하는 것이 핵심 과제이다. 기존 주제로는 기술적, 창의성, 독창성 면에서 큰 개선이 어려워 VR 게임 개발로 주제를 변경하게 되었다. VR은 현재 게임 산업에서 많은 주목을 받고 있으며, Unity에서 언리얼 엔진으로의 전환을 통해 높은 수준의 기술적 도전과 창의성의 발휘가 가능할 것으로 기대된다.

이 보고서에서는 대강적인 개발 주제와 그에 따른 개별적으로 진행한 개발 내용을 소개한다.

**1. 개요**

**(1) 개발의 필요성**

상세한 배경: 리듬과 로그 라이크 장르의 결합을 통한 게임 경험은 해당 장르의 특성상 플레이어들 간의 역량 차이로 갈리는 경우가 많다. 리듬 게임은 음악적 감각이 뛰어난 사람, 로그 라이크 게임은 전략적 사고가 필요한 사람이 주로 즐긴다. 그러나 이로 인해 한계점이 발생하고, 두 장르 간의 교류가 적은 편이다. 이러한 한계를 극복하고자, 플레이어가 양쪽 장르의 특성을 동시에 체험하며 높은 접근성을 가진 게임을 개발하고자 한다.

**(2) 개발 목표 및 가정**

전략적인 유저와 음악적 감각을 가진 유저 모두가 즐길 수 있는 게임을 개발하는 것이 목표이다. 게임은 두 장르를 조화시킨 새로운 경험을 제공하여, 리듬에 집중하면서도 전략적인 선택이 요구되는 독특한 플레이를 플레이어에게 제공한다.

장르 간의 장점을 결합하여 다양한 재미를 선사하는 것을 목표로 한다. 리듬 게임의 지루함과 로그 라이크 게임의 불가피한 실패에 따른 스트레스를 완화하기 위해, 게임은 플레이어의 능력과 선택에 따라 변화하는 동적인 플레이 스타일을 지향한다.



그림 1. 초기 기획안

**(3) 활용 분야**

텍스트, 스크린샷, 멀티미디어 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명장르 융합의 가능성: 게임 산업에서는 기존의 한계를 극복하고 새로운 트렌드를 만들 수 있는 장르의 융합이 주목받고 있다. 이 게임은 해당 트렌드에 발맞춰 두 장르를 성공적으로 결합하여 다양한 게임 시나리오와 플레이 방식을 지원하며, 다양한 게이머들에게 새로운 경험을 제공할 것이다.

그림 2. 로그라이크 게임 中 “뱀파이어 서바이벌”



그림 3. 리듬 게임 中 “DJMAX”

**(4) 접근 방법**

유니티 엔진을 활용하여 리듬과 로그 라이크의 특성을 조합하는 새로운 게임 시스템을 구축한다. 각 플레이어의 선택이 게임의 진행과 난이도에 영향을 미치기도 하며 어느 정도 피지컬 적인 요소가 필요하기도 한다. 이렇게 양쪽의 장점이 적절하게 조합되어 서로의 재미를 반감 시키지 않고 양쪽의 재미를 즐길 수 있는 게임 플레이를 설계한다

2. 개발 완료한 내용

가. 리듬 게임 PART

(1) 기본적인 UI 개발

리듬게임 시스템이 진행될 부분의 라인, 판정의 기준이 될 판정선을 직접 구현하였다. 실제로 라인의 적절한 위치에 노트가 생성되어 설정된 속도로 하강하고 판정하는 매커니즘이 해당 라인에서 진행된다.

스크린샷, 멀티미디어, 디스플레이 장치, 전자 기기이(가) 표시된 사진

자동 생성된 설명

그림 4. 리듬 게임이 진행되는 부분

(2) 노래 정보 읽어오기

노트를 생성하기에 앞서 Sheet 클래스에 노래 정보와 노트 정보를 불러와 저장해놓는다. Sheet 클래스에는 제목, 작곡가, 음악 클립의 인덱스 번호, 노래가 시작될 때까지의 대기 시간인 offset, 가장 중요한 노트들의 라인 및 시간 정보, 한 마디 별 시간 등의 여러 정보를 담고 있다.

가장 먼저 게임이 시작하면 우선 새로운 Sheet 클래스를 만들고 SheetLoader.cs에서 지정된 경로에 있는 Sheet 파일을 모두 불러온다. 이렇게 불러온 Sheet 파일은 Parser.cs의 IEParse로 넘어가서 필요한 정보별로 정리해서 해당하는 변수에 저장하는 과정을 진행한다. 이 때 제목과 작곡가, 노트에 대한 정보들이 Sheet 클래스의 변수에 저장된다.

이제 플레이가 시작되면 제목을 매개변수로 넘겨서 해당하는 노래의 Sheet를 불러와 Sheet 속의 노트 정보가 담긴 notes를 기반으로 노트를 생성하고 판정한다.

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

그림 5. Sheet Class 생성 과정

(3) 노트 생성하기

이제 읽어온 Sheet 정보를 기반으로 Note를 생성한다. Short 노트의 경우와 Long 노트의 경우를 구분하여 노트를 생성했다.

- 오브젝트 풀링 기법

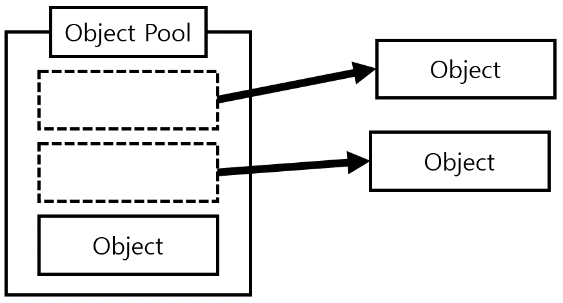
 앞으로 노트를 적게는 수 십개, 많게는 수백개를 생성하게 될 텐데 이러한 노트 오브젝트들이 가벼운 리소스를 사용한다고 하더라도 유니티의 특성 상 오브젝트 프리팹을 지속적으로 복사해서 생성하고 삭제하면 쓰레기 더미 파일이 계속 남아 성능에 과부하가 걸린다. 이를 방지하기 위한 대표적인 기법으로 오브젝트 풀링 기법을 사용해서, 노트 프리팹을 복사해서 사용한 후에 해당 오브젝트를 삭제하는 것이 아니라 비활성화 시킨 후 다시 필요할 때 활성화 시키고 적절한 위치로 다시 위치시킨 후에 사용할 수 있도록 한다. 이러한 방식을 위해서는 오브젝트들이 생성되고 비활성화 된 동안 보관될 일종의 수영장 같은 개념의 풀이 필요하다고 해서 오브젝트 풀링 기법이라고 불린다. 따라서 노트들이 저장될 풀을 직접 만들고 해당 풀에서 노트를 가져가서 사용하고 반환하는 방법을 사용하자.

그림 6. 오브젝트 풀링

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 1. Short Note 용 오브젝트 풀

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

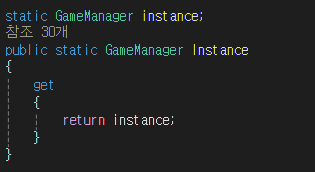
코드 2. Long Note 용 오브젝트 풀

위와 같이 각 노트 종류별 풀을 생성해주었다. 한가지 풀에 저장하지 않는 이유는 기본적으로 Long Note의 경우 Short note와는 별개로 Head, Tail, Line 등 더 많은 양의 데이터를 저장하고 있는 완전히 다른 데이터 타입이기 때문에 풀을 따로따로 만들어 주었다.

이제 위 (2)에서 저장된 Sheet 클래스의 notes를 읽어와서 노트를 생성한다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

코드 3. Sheet의 notes 불러오기

위와 같이 노래의 title을 매개변수로 사용해서 notes들의 정보를 불러온다. 이 때 게임 매니저의 인스턴스에 바로 접근하여 정보를 가져올 수 있는데, 이는 싱글톤 패턴으로 게임 매니저를 등록 해놓았기 때문이다. 오른쪽과 같이 실제 GameManger.cs에 작성되어 있는데 이렇게 작성하면 게임 매니저는 정적으로 선언되어 인스턴스가 메모리 상에 직접적으로 올라가 있게 된다. 그렇기에 다른 스크립트에서도 별도의 선언 없이 바로 접근해서 해당 인스턴스에 접근해 게임 매니저에 있는 함수나 변수를 읽거나 호출할 수 있게 된다. 이러한 싱글톤 패턴으로 게임 매니저를 사용한 이유는 게임 매니저가 전반의 게임 시스템에 단 하나만 존재하는 유일한 개념이고, 게임의 대표적인 정보 (Sheet 정보, 게임 속도, 플레이 상태) 등, 등의 정보를 모두 가지고 있어 다른 곳에서도 자유롭게 사용할 수 있도록 싱글톤 패턴으로 작성을 해놓았다.

코드 4. 싱글톤 패턴

이제 Sheet 정보를 읽어오고 난 후에는 currentBar 만큼의 노트들을 생성하기 위해 next를 증가 시킨다. 노트를 생성할 때 처음에 currentBar 만큼을 미리 생성하고 시작하고 노래가 진행되면서 특정 시간을 주기로 지속적으로 노트를 생성한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 5. Note 생성

실제로 노트가 생성되는 구간이다. 생성할 노트의 타입에 따라 아까 사전에 생성한 PoolShort에서 Short 노트를 가져오거나, PoolLong에서 Long노트를 가져온다. Sheet에서 읽어온 노트 정보를통해 오브젝트 풀에서 가져온 노트의 위치를 초기화 해준다. ( notes의 노트들에는 노트들의 판정 시간과 라인, 노트 타입의 정보들이 저장되어있다.



코드 6. 노트 초기 위치 계산

위와 같이 x좌표는 라인 번호에 맞는 좌표, y좌표의 경우에는 “**노트 시간 - 현재 음악 시간 + 판정 Line의 y좌표**”로 설정해주었다. 이렇게 y좌표를 계산하면 판정선(JudgeLine)에 원하는 시간에 정확하게 떨어진다. 이제 노트의 초기 위치 설정이 끝났다면 해당 오브젝트의 speed, life와 같이 변수를 초기화해주고 set Active로 활성화 시켜준 다음 Move를 실행해 노트가 떨어지도록 한다. 이제 이 노트는 앞으로 반환 받아야하는 노트이므로 toReleaseList에 저장해놓는다.

위와 같은 일련의 과정들로 Sheet의 notes로부터 노트 데이터를 읽어와 원하는 타입의 노트를 원하는 위치에 생성한다.

(4) 노트 반환하기

생성된 노트를 이제 반환 받을 수 있도록 반환 부분을 구현했다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 7. 노트 반환받기

반환 받아야하는 노트들은 모두 toReleaseList에 저장 되어있다. 일정 주기로 해당 리스트의 모든 노트들을 확인하면서 Life 여부를 확인해서 Life가 false 인 노트는 Release 시켜준다. 이 때 노트의 Life는 노트가 일정 높이 이하로 바뀌면 자동으로 false로 바뀌도록 Note.cs 안에 구현 되어있다. 이러한 방식으로 판정을 하지 못하고 일정 높이까지 추락한 노트는 자동으로 Release 되도록 구현하였다.

텍스트, 스크린샷, 디스플레이, 폰트이(가) 표시된 사진

자동 생성된 설명

코드 8. IEReleaseTimer

일정 주기로 Release로 하기 위해서 위와 같은 ReleaseTimer를 코루틴 함수로 돌려주면서 지속적으로 Release 함수를 호출한다. 이 때 interval 시간만큼 대기하고 Release 하는 것을 확인할 수 있다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 9. 추가적인 Release

위와 같이 주기 Release 말고도 바닥에 떨어진 노트를 Dequeue하는 함수와 판정된 노트를 그 즉시 release하고 비활성화하는 JudgeNoteRelease도 작성했다. 이러한 함수들을 작성한 이유는 추후에 해당 함수를 호출하는 부분에서 자세히 설명하겠다.

이렇게 생성되고 반환되는 note 오브젝트는 어떻게 구현되었는지는 바로 다음에 기술하였다.

(5) 노트 오브젝트

스크린샷, 소프트웨어, 텍스트, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 10. Note Obejct

노트 오브젝트는 다음과 같이 추상화를 사용해서 구현하였다. 추상 함수로 구현한 이유는 노트의 타입에 따라 SetPosition과 같이 위치 설정이나 Interpolate와 같은 위치 보정 함수가 달리 실행되어야 하므로 추상화를 사용하였다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 11. Short Note Object

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 12. Long Note Object

다음과 같이 각 타입 별 노트를 구현하였다. Long Note의 Move와 IEMove는 Short Note와 동일한 코드라 생략하였다. Long Note의 경우, Short Note와는 달리 Head, tail, Line transform을 모두 저장하고 있으며 위치 설정하거나 위치를 보정할 때도 모든 트랜스폼의 위치를 설정해준다.

(6) 시간 기반 판정 시스템

생성된 노트들의 판정 시스템을 다음과 같이 구현하였다. 이 때 기존에 개발된 위치 기반 판정을 시간 기반 판정으로 바꾸었다. 바꾼 이유는 아래 그림을 참고한다.

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

그림 7. 위치 기반 판정

시간 기반 판정의 수행 원리는 아래 그림과 같다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

그림 8. 시간 기반 판정 원리

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명 위와 같은 시간 기반 판정을 구현하기 위해 아래와 같이 코드를 작성했다.

코드 13. 시간 기반 판정 코드

먼저 라인별 노트 판정 시간을 저장해둔다. 위의 그림 8을 보면 알겠지만 판정에 필요한 시간은 총 2개로 키를 입력한 순간의 노래 시간과 노트가 판정되어야 하는 시간이다. 노트가 판정되어야 하는, 즉 판정선에 도착해야하는 시간은 기존에 노트를 생성할 때와 마찬가지로 Sheet의 notes의 note들을 자신이 떨어져야 할 라인에 해당하는 큐에 저장되게 된다. 이렇게 하면 각 라인별로 다음에 판정되어야 할 노트의 판정 시간을 쉽게 확인할 수 있게 된다. 판정 시간의 저장이 끝났으면 이제 Judge에서 해당 라인에 판정 시간과 위에서 언급했던 노래 시간과 비교해서 판정한다.

(7) 판정

키를 입력 받았을 때 Judge 함수가 실행된다. 이 때 위 그림 8에서와 같이 누른 라인에 해당하는 큐의 맨 앞의 노트의 판정 시간과 내가 현재 누른 시간을 비교해서 차이가 일정 범위 안으로 들어오면 노트의 판정을 진행한다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 14. 판정 진행

이 때 max10 ~ max100은 판정 시간 범위를 저장해놓은 변수이다. 시간 차이가 maxbreak 범위 안에 들어오면 판정을 진행하는데 maxbreak 범위에는 들어오지만 max1에 들어오지 못하면 break로 판정하고 해당하는 콤보 증감을 진행한다. Max1 안에 들어온다면 어떤 범위에 해당하는 if-else 문으로 확인하여 해당하는 콤보 증감을 진행한다. 하지만 점수 시스템의 경우에는 바로 입력하지 않고 플레이 상태에 따라 진행 여부를 결정하는데 이는 로그 라이크 시스템의 플레이어가 사망했을 때는 더 이상 점수를 증가시키지 않도록 하기 위함이다.

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 15. 게임 플레이 상태에 따른 SetScore

이제 노트 종류에 따른 이펙트를 진행한다. 그러고 나면 Short 노트는 life가 false일 때까지 기다렸다가 release하는 것이 아니라 바로 JudgedNoteRelease라는 함수를 호출해 바로 반환 시킨다. (노트가 판정됨과 동시에 사라지도록 하기 위해서이다. Life가 false가 될 때까지 기다리면 노트가 판정 되어도 오브젝트가 사라지지 않고 그대로 내려가는 것이 플레이어에게 보인다. )

텍스트, 스크린샷, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 16. 롱 노트 판정

롱 노트의 판정의 경우 키를 입력했다가 땔 때도 추가적인 판정을 해야하기 때문에 CheckLongNote를 추가적으로 작성해주었다. 위에서 롱 노트를 입력한 순간 해당 라인에 맞는 longNoteCheck가 1로 변경되고 해당 라인의 키 입력을 땐 순간 longNoteCheck가 1이면 롱노트를 눌렀다가 땠다는 것을 의미하므로 판정을 진행한다.

이 때 롱노트의 경우 FallNoteDequeue를 진행하는데 해당 롱 노트는 끝까지 내려가서 판정이 끝난 노트이기 때문에 별도로 note 큐에서 dequeue 해줘야 시간판정이 정상 작동한다. 일종의 롱 노트용 JudgedNoteRelease와 동일한 기능을 하는 함수이다.

이제 키를 입력하지 못해 판정받지 못하고 그대로 진행되어 판정 범위를 벗어나는 경우 Miss 판정을 진행한다.

텍스트, 스크린샷, 소프트웨어, 디스플레이이(가) 표시된 사진

자동 생성된 설명

코드 17. Break 노트 판정

다음과 같이 미스 판정을 진행한다. 위의 시간 판정 원리와 동일하게 노트 큐의 판정시간과 노래의 진행 시간을 비교해서 판정이 가능한 범위를 벗어난다면 더 이상 해당 노트는 입력할 수 없음을 의미하고 따라서 Break 판정을 진행한다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 18. Break 노트 판정

Long 노트와 Short 노트로 각각 구분하여 해당 노트에 맞는 판정을 진행한다. 이 때 중요한 것은 게임이 플레이 상태가 아니라면 즉, 플레이어가 죽어서 게임 오버된 경우에는 굳이 노트를 릴리스를 진행하지 않는다. 어차피 알아서 시간이 지나면 일정 높이 이하로 떨어져 Life가 False로 변하고 ReleaseTimer가 호출하는 release에서 반환될 예정이기 때문이다.

(8) 점수 시스템

리듬 게임의 점수와 콤보 시스템은 Score안에 구현하였다. Score 스크립트 내부에서 판정에 따른 점수가 증가되고 점수와 콤보에 따른 UI를 지속적으로 업데이트하도록 구현하였다.

노트 판정이 진행되어 해당하는 판정에 따른 점수를 증가시키기 위해 SetScore함수가 실행된다.

텍스트, 스크린샷, 디스플레이, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 19. SetScore 코드

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명setScore가 진행되면 판정에 걸맞는 애니메이션이 실행되도록 Effect 매니저의 judgeEffect 함수를 호출하고 UI를 업데이트 해준다. 이 때 콤보가 2 이상일 때만 콤보 텍스트가 뜨도록 combo를 직접 비교한 2 이상이면 해당 숫자로, 2보다 작으면 공백으로 UI를 업데이트 해준다. 점수의 경우에는 바로 업데이트 해준다. 점수는 오른쪽과 같이 판정 종류별 개수를 통해 총 점수를 계산하는 방식으로 점수를 계산하도록 구현했다.

코드 20. 점수 증가

(9) 이펙트 매니저

이펙트 매니저에서 각종 모든 애니메이션을 총괄한다. 애니메이션으로 이펙트를 실행하기 때문에 이름을 이펙트 매니저로 명명하였다.

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

코드 21. CoolBomb\_Animation

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명노트의 판정 종류에 따라 다른 이펙트가 실행되도록 max\_index를 매개변수로 전달받아 해당 하는 값에 따른 애니메이션의 Trigger를 호출해 해당 애니메이션을 실행한다. 이 외에도 판정 여부에 따라 MAX 100%, MAX 90%와 같은 스프라이트가 화면에 출력되도록 JudgeEffect 함수 또한 구현하였다.

코드 22. JudgeEffect

(10) 입력 시스템

스크린샷, 텍스트, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명입력 시스템은 유니티에서 제공하는 Input System을 기반으로 구현하였다.

코드 23. Input System

각 입력에 따른 NoteLine을 매핑 시켜놓고 해당하는 NoteLine을 눌렀을 때 Judge가 진행되도록 구현하였다. 입력한 키를 땔 때는 CheckLongNote를 진행하도록 구현하였다.

나. 로그 라이크 PART

(1) 게임 오버 구현

로그 라이크 부분에서는 게임 오버를 판정하고 게임 State를 바꾸고 다시하기 버튼을 누르면 게임이 다시 시작되는 로직을 구현하였다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

코드 24. 체력 확인

체력이 0보다 작아지게 되면 GameOver 함수를 호출한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

코드 25. GameOver

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

코드 26. DisabledNote

게임 오버가 되면 게임 state를 바꿔주고 노래를 멈춘다. 그러고 난 후 NoteGenerator에 DisabledNote를 호출하는데 DisabledNote는 모든 노트를 비활성화해주는 역할을 한다. 이렇게 하면 죽은 순간 모든 노트들이 숨겨져 죽고 나서도 노트가 계속 내려가는 것처럼 보이는 것을 방지한다. 그러고 난후 GameOver ui를 활성화 시켜 게임 오버 사실을 알리고 다시하기 버튼을 누를 수 있도록 한다.

텍스트, 스크린샷, 멀티미디어 소프트웨어, 디자인이(가) 표시된 사진

자동 생성된 설명

그림 9. 게임 오버 UI 및 다시하기 버튼

다시하기 버튼을 누르면 SceneManager의 LoadScene(0)으로 게임 씬 자체를 다시 불러오면서 다시 게임을 시작하도록 구현했다.

다. VR PART

(1) 눈높이 위치 보정 구현

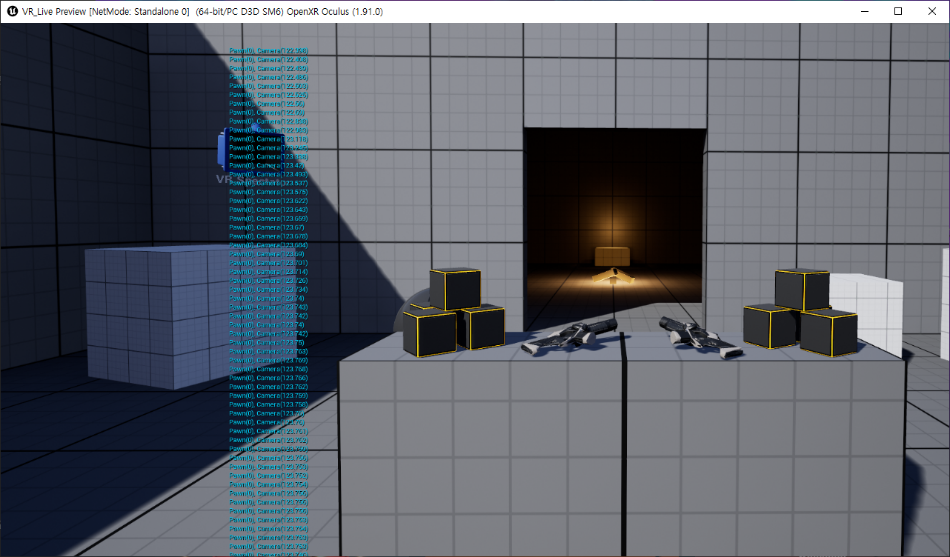


그림 10. 눈높이 위치 확인

VR 헤드셋을 끼고 나서 실시간으로 Pawn과 카메라의 위치를 찍어보면 카메라의 높이가 헤드셋의 높이에 따라 변하는 것을 확인할 수 있다. 실제 Floor 레벨은 초기 VR 설정 과정에서 안전 구역으로 설정한 영역의 Floor 레벨을 기준으로 현재 헤드셋과 높이 차이가 실제로 보정이 되어 BP Pawn에 적용이 되어서 카메라의 높이에 적용된다고 할 수 있다.

목표 높이를 DesiredCameraHeight를 Float형 변수로 선언해준 다음 초기 값을 75로 설정한다.

스크린샷, 멀티미디어 소프트웨어, 그래픽 소프트웨어, 3D 모델링이(가) 표시된 사진

자동 생성된 설명

원하는 높이랑 실제 카메라 높이가 다르다고 하면 그 차이만큼 VR 루트의 위치를 보정시키게 된다면 우리가 원하는 위치로 카메라를 고정할 수 있다. 이렇게 하면 앉아있던 일어서 있던 상관없이 내가 원하는 높이에서 카메라의 위치를 보정 시켜서 시작할 수 있다.

스크린샷, 사각형, 타일이(가) 표시된 사진

자동 생성된 설명

그림 11. 눈높이 보정 완료

일어선 상태에서 실행시켰음에도 불구하고 높이가 75로 다시 낮아진 상태로 카메라가 보정된 것을 확인할 수 있다.

(2) Hand 액터 추가

Pawn 안에 왼손, 오른손 스켈레탈 메쉬를 모두 넣어두는 기존 UE5 VR Template과는 다르게 별도로 Pawn이 실행되면 알아서 Hand Actor를 생성해줘서 계층 구조로 핸드 액터를 추가해주도록 구현했다. 이렇게 Hand Actor를 빼내서 왼손 오른손 공용으로 구현함으로써 기존 폰 BP에 모션 컨트롤러와 Hand의 여러 기능이 추가 되면서 BP가 복잡해지는 현상을 방지하면서 별도의 Hand 액터를 관리함으로써 모션 컨트롤러를 사용하는 여러 기능을 구현함에 있어서 더 편리하고 확장성 있도록 구현하 였다.

스크린샷, 회로이(가) 표시된 사진

자동 생성된 설명

그림 12. Hand Mesh 불러오기

BeginPlay 이벤트시 왼손 오른손 여부에 따라 적절한 Mesh를 불러오도록 Mesh 설정을 구현하였다. 이러한 Hand Actor를 Pawn에서 불러오도록 구현했다.

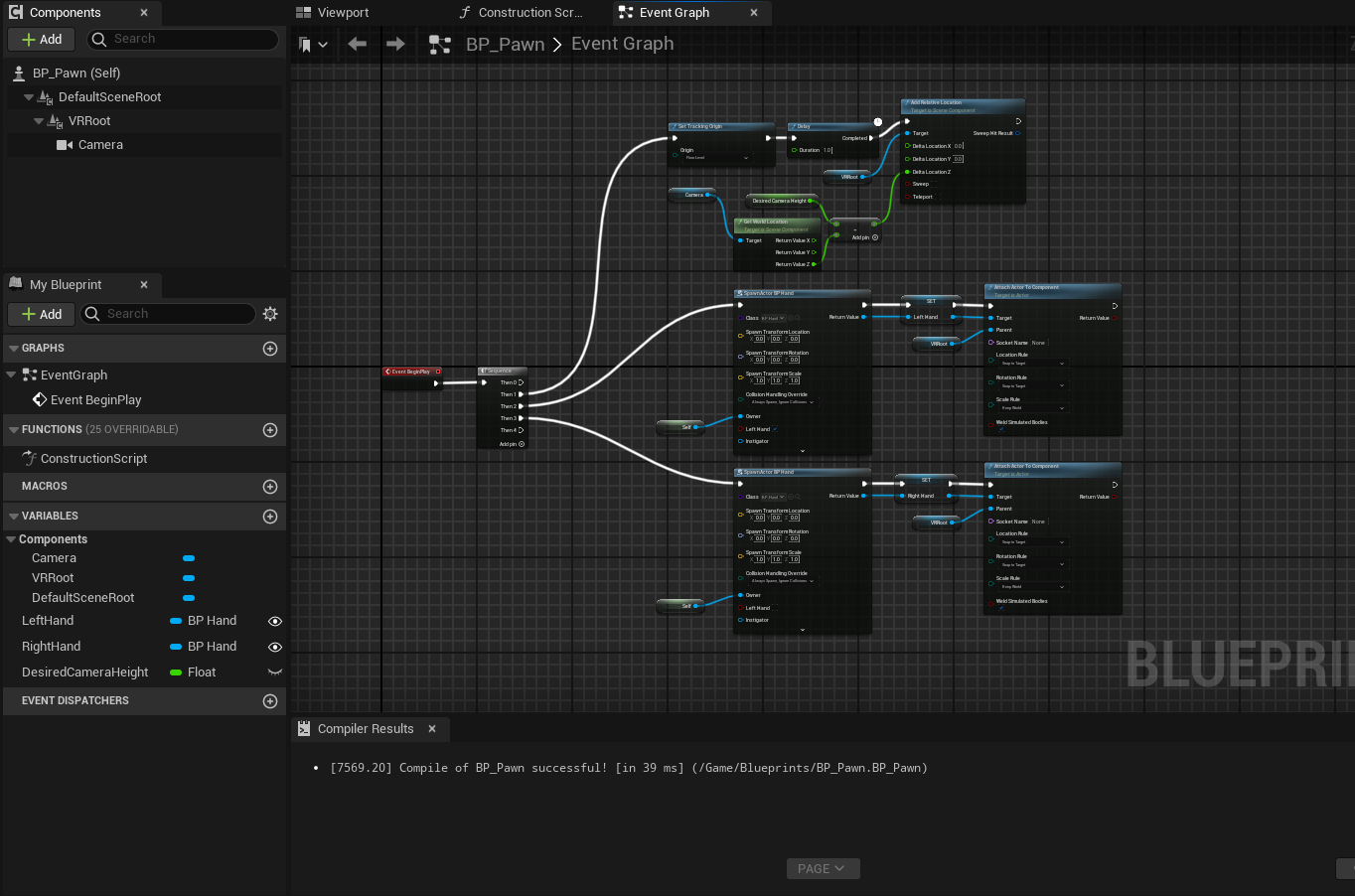


그림 13. BP\_Pawn

스크린샷, 멀티미디어 소프트웨어, 그래픽 소프트웨어, 3D 모델링이(가) 표시된 사진

자동 생성된 설명스크린샷, 멀티미디어 소프트웨어, 그래픽 소프트웨어, 텍스트이(가) 표시된 사진

자동 생성된 설명

그림 14. Hand 추가

BP\_Pawn이 시작되면 (1)에서 구현한 눈높이 보정 로직을 실행한 다음 BP\_Hand를 Spawn한 다음 VR Root의 자식 오브젝트로 넣어준다. 이렇게 하면 실제로 BP\_Pawn 안에는 어떠한 Hand나 모션 컨트롤러도 없지만 시작되고 나면 손이 정상적으로 생성된다.

(3) IMC, IA 구현

텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

그림 15. IMC\_VR\_Default

언리얼에서 제공하는 Input Mapping Context와 Input Action을 이용해 입력을 받고 처리하도록 구현하였다. 이렇게 VR 기기(현 프로젝트는 오큘러스의 메타 퀘스트 3로 테스트를 진행)의 입력에 따라 인풋 액션을 매핑 시켰다. 하지만 여기까지는 언리얼에서 자체적으로 제공하는 입력 시스템을 사용했다고 볼 수 있다. 하지만 입력에 따른 이동 방식과 회전 방식을 직접 구현하였다. 다음 내용에 기술하였다.

(4) 보행 이동 구현

언리얼의 기본적인 VR 이동 시스템은 텔레포트 시스템이다. 하지만 이러한 방식이 실제로 사용하기에는 불편하고 뒤로는 이동할 수 없다는 단점이 있어서 다음과 같이 FloatingPawnMovenent 컴포넌트를 이용해 부드럽게 이동하도록 이동을 구현하였다.

스크린샷, 텍스트, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

그림 16. 보행 이동 구현

이후에 바라보고 있는 방향으로 이동하는 것이 더 자연스러울 것 같아서 다음과 같이 업데이트하였다.

텍스트, 스크린샷, 멀티미디어 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

그림 17. 보행 이동 개선

(5) 고개 회전 구현

스크린샷, 텍스트, 멀티미디어 소프트웨어, 그래픽 소프트웨어이(가) 표시된 사진

자동 생성된 설명

그림 18. snapTurn

다음과 같이 고개 회전을 구현하였다. 하지만 구현하고 나서 문제점을 발견했는데 다음과 같은 문제점이 생겼다.

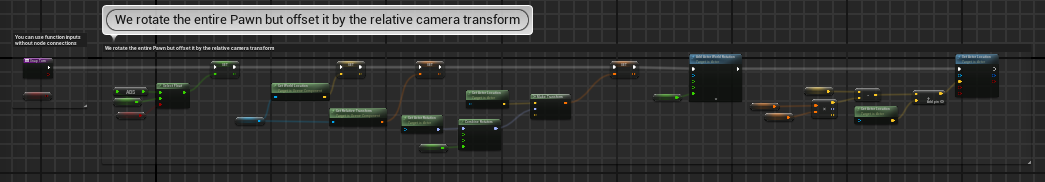
텍스트, 도표, 원, 폰트이(가) 표시된 사진

자동 생성된 설명

그림 19. 고개 회전 문제점

폰을 회전시키다 보니 카메라를 기준으로 위치를 잡고 있는 플레이어가 느꼈을 때 고개를 내민 상태에서 회전을 하면 회전을 많이 한 것 같은 느낌이 들게 된다. 쉽게 말해 허리를 숙인 상태에서 회전을 하면 고개를 회전 하는 것이 아니라 허리를 굽힌 상태에서 발의 방향을 돌린 듯한 느낌이 든다는 것이었다. 이를 해결하기 위해 언리얼에서 자체적으로 제공하는 SnapTurn을 확인해서 분석해보았다.

기존의 언리얼에 내장된 snapTurn은 아래와 같이 구현되어 있다.



노드들도 복잡하고 설명도 부족하지만 실제 작동 원리를 그림으로 나타내면 아래와 같다. 텍스트, 도표, 라인, 폰트이(가) 표시된 사진

자동 생성된 설명

그림 20. SnapTurn의 작동원리

그림이 난해하지만 결국 Pawn을 직접 회전 시켰을 때 카메라가 순간적으로 많이 회전하는 현상을 방지하기 위해 위의 그림에서 “원하는 벡터”를 구해서 회전 후 Pawn의 위치를 보정 시켜서 카메라를 기준으로 회전한 거 같은 느낌이 들도록 하는 것이다.

스크린샷, 멀티미디어 소프트웨어, 그래픽 소프트웨어, 3D 모델링이(가) 표시된 사진

자동 생성된 설명

그림 21. 직접 구현한 SnapTurn

이것은 직접 구현한 snapTurn 이다. 위의 언리얼에서 제공하는 snapTurn보다 직관적이고 간단하게 snapTurn을 구현할 수 있었다. 원리를 간단히 설명하자면 기존의 카메라의 월드 로케이션 정보를 선언해둔 변수에 저장해두고 회전(공전)을 진행한다. 그런 다음 변경된 현재 카메라의 위치에서 기존 카메라의 위치를 빼서 아까 위에서 표현했던 “원하는 벡터”인 카메라의 이동 벡터를 찾는다. 그런 다음 해당 벡터만큼 BP\_Pawn의 월드 좌표를 이동시켜주면 SnapTurn 기능을 더 간단하게 구현할 수 있다.

(6) Grab 구현

Grab의 순서는 기존의 IMC, IA에 Trigger을 매핑시킨다음 Grab을 하는 순간 Handle Grab이 실행 되도록 한다. Grab의 기본적인 원리는 Attach Component to Component를 통해 Hand의 모션 컨트롤러를 Grab Component의 부모 오브젝트에 붙여버리는 것이다.

스크린샷, 3D 모델링이(가) 표시된 사진

자동 생성된 설명

그림 22. Handle Grab

Handle Grab이 시도되면 그 이후에 물건을 잡으려고 시도했는지 파악하기 위해 Try Grab을 호출한다. 스크린샷, 3D 모델링이(가) 표시된 사진

자동 생성된 설명

그림 23. Try Grab

이 때 Grab Type에 따라 Free, ObjToHand , HandToObj로 구분하는데 Free의 경우 잡는 부분을 기준으로 물체를 제어한다. ObjToHand는 오브젝트를 원하는 위치로 끌어와 원하는 위치를 잡은 것 같은 효과를 보여준다. (총과 칼과 같은 경우 해당 물체를 끌어와 손잡이가 잡힌 것처럼 연출) HandToObj의 경우 핸드가 오브젝트로 이동해서 오브젝트의 특정 부분을 잡은 것처럼 표현한다. (서랍을 연다고 하면 서랍의 손잡이가 손의 위치로 오는 것보다 손이 서랍의 손잡이에 붙은 것이 훨씬 자연스러운 연출)

- Free Type

스크린샷, 멀티미디어 소프트웨어, 비디오 게임 소프트웨어, PC 게임이(가) 표시된 사진

자동 생성된 설명

- ObjToHand

스크린샷, 멀티미디어 소프트웨어, 그래픽 소프트웨어, 3D 모델링이(가) 표시된 사진

자동 생성된 설명 스크린샷, 3D 모델링이(가) 표시된 사진

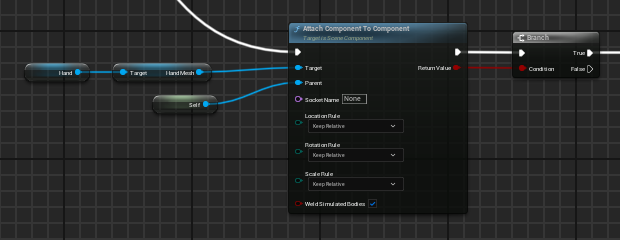
자동 생성된 설명

- HandToObj

스크린샷, 그래픽 소프트웨어, 멀티미디어 소프트웨어, 3D 모델링이(가) 표시된 사진

자동 생성된 설명

\* Set Should simulate On Drop은 물체를 잡기 전의 물리 효과 적용 여부를 기억해서 해당 물체를 놓는 순간 해당 물체의 물리 효과를 다시 복원 시키기 위한 함수이다.

스크린샷, 텍스트이(가) 표시된 사진

자동 생성된 설명

그림 25. HandToObj

이 때 중요한 점은 HandToObj의 경우 위의 블루 프린트와 같이 모션 컨트롤러를 직접 보내는 것이 아니라 핸드 메쉬를 Grab Component 위치로 보내서 부착시키도록 조정한다. 이 때 손을 놓을때 핸드 메쉬를 다시 내 모션 컨트롤러로 가지고 돌아와야 하기 때문에 위치를 사전에 미리 기억해 놓고 그랩을 풀면 핸드 메쉬를 저장한 위치로 복귀 시키도록 구현한다.

\* 핸드 메쉬를 이동시키는 이유? 모션 컨트롤러를 직접적으로 이동시키게 되면 모션 컨트롤러가 물체에 부착된 상테에서 움직이게 된다면 모션 커트롤러의 기존 위치를 잃어버리게 되고 모션 트래킹이 깨지게 된다. 즉, 내가 쥐고 있는 모션 컨트롤러의 위치에 정상적으로 컨트롤러가 위치하지 않게 되는 현상이 발생한다. 따라서 모션 컨트롤러는 절대 직접적으로 위치를 수정해서는 안된다. 모션 컨트롤러를 움직일 수 없으니 핸드 메쉬만 움직여서 손이 가서 잡은 것 같은 효과를 주는 것이다.

스크린샷, 3D 모델링이(가) 표시된 사진

자동 생성된 설명

그림 26. BP Hand

스크린샷, 비디오 게임 소프트웨어, 3D 모델링, PC 게임이(가) 표시된 사진

자동 생성된 설명

다음과 같이 BP\_Hand Mesh의 초기 상대 위치를 변수에 저장해두고 Release 시에 기억해놓은 트랜스폼 위치에 핸드메쉬를 다시 부착시켜준다.

스크린샷, 멀티미디어 소프트웨어, 그래픽 소프트웨어, 소프트웨어이(가) 표시된 사진

자동 생성된 설명

(7). Try Release

Release는 Grab을 했던 것을 반대로 Detach 해줘서 다시 Hand를 떨어뜨리고 기존 물체의 물리 효과를 복원 시킨다. HandToObj의 경우에는 ResetHandMesh를 진행해 물체에 부착되어 있는 핸드 메쉬를 내 모션 컨트롤러의 원래 위치로 돌아오게 한다.

스크린샷, 3D 모델링, 그래픽 소프트웨어, 비디오 게임 소프트웨어이(가) 표시된 사진

자동 생성된 설명

3. 향후 개발할 계획

(1) Shoot

총을 쥔 상태에서 IMC를 새로 매핑 시켜주어, 총을 쥔 상태로 Shoot Trigger를 당기면 총알이 나가도록 총을 구현할 예정이다.

스크린샷, 3D 모델링, 회로이(가) 표시된 사진

자동 생성된 설명

스크린샷, 3D 모델링, 그래픽 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명

실제로 IMC, IA는 모두 제작하여 매핑해두었고 로직도 구현해놓았는데, 총알 BP를 구현하지 않아서 아직까지 완전히 구현에 성공했다고 할 수 없다. 추후에 직선으로 나가는 총알 Class를 구현해서 해당 총알이 Spawn 되도록 할 예정이다.

(2) 총&칼 로직 구현

위와 같이 총의 구현이 마무리 되면 양손의 무기를 이용해 날라오는 박스를 파괴하는 로직을 구현할 예정이다.

(3) 여러가지 기능 개선

VR의 여러가지 기능을 추가적으로 구현하면서 시스템 적으로 개선할 수 있는 부분이 있으면 개선할 예정이다.

4. 첨부자료

(1) 본인이 단독으로 작성한 소스파일들 [소스파일들은 프로젝트 파일 내부에 첨부되어 있습니다]

(**\*팀원들과 함께 작업, 파랑색은 단독 작성)**

**리듬 게임 PART**

- EffectManager.cs \*

- GameManager.cs \*

- AuidoManager.cs

- InputManager.cs

- Judgement.cs

- NoteGenerater.cs

- NoteObject.cs

- Parser.cs

- Score.cs

- Sheet.cs

- SheetLoader.cs

- Sync.cs

**VR PART**

- IA, IMC

- BP\_GameMode

- BP\_Pawn

- BP\_Gun

- BP\_Hand

- BP\_GrabComponent

- BP\_Cube \*

- BP\_Saber \*

(2) 본인이 제작한 컨텐츠들

- 리듬 게임 UI

- 리듬 게임 이펙트 및 애니메이션 에셋 추출