UI UX게임

35기 이찬규

목차

[1. UI, UX 기획 4](#_Toc527077177)

[1.1. 로비 화면 5](#_Toc527077178)

[1.2. 인 게임 6](#_Toc527077179)

[1.3. 인 게임 메뉴 7](#_Toc527077180)

[1.4. 인벤토리 8](#_Toc527077181)

[2. 게임 엔진 설계 9](#_Toc527077182)

[2.1. DirectX Graphic Pipeline 9](#_Toc527077183)

[2.1.1. Input-Assembler Stage 10](#_Toc527077184)

[2.1.2. Vertex Shader Stage 11](#_Toc527077185)

[2.1.3. Pixel Shader Stage 12](#_Toc527077186)

[2.2. Alpha Blend 12](#_Toc527077187)

[2.3. Alpha Test 13](#_Toc527077188)

[3. 게임 개요 14](#_Toc527077189)

[3.1. 게임 소개 14](#_Toc527077190)

[3.2. 게임의 진행 14](#_Toc527077191)

[4. 게임 시스템 16](#_Toc527077192)

[4.1. Core 와 Sample 클래스 16](#_Toc527077193)

[4.2. SceneMgr와 Scene 클래스 17](#_Toc527077194)

[4.3. Shader와 Texture 클래스 19](#_Toc527077195)

[4.4. ObjectMgr과 Object 클래스 20](#_Toc527077196)

[4.4.1. Effect클래스 21](#_Toc527077197)

[4.4.2. Character 클래스 22](#_Toc527077198)

[4.5. State 23](#_Toc527077199)

[4.5.1. PlayerState 23](#_Toc527077200)

[4.5.2. EnemyState 23](#_Toc527077201)

[4.6. UI 25](#_Toc527077202)

[4.6.1. LobbyMenu 25](#_Toc527077203)

[4.6.2. Setting 26](#_Toc527077204)

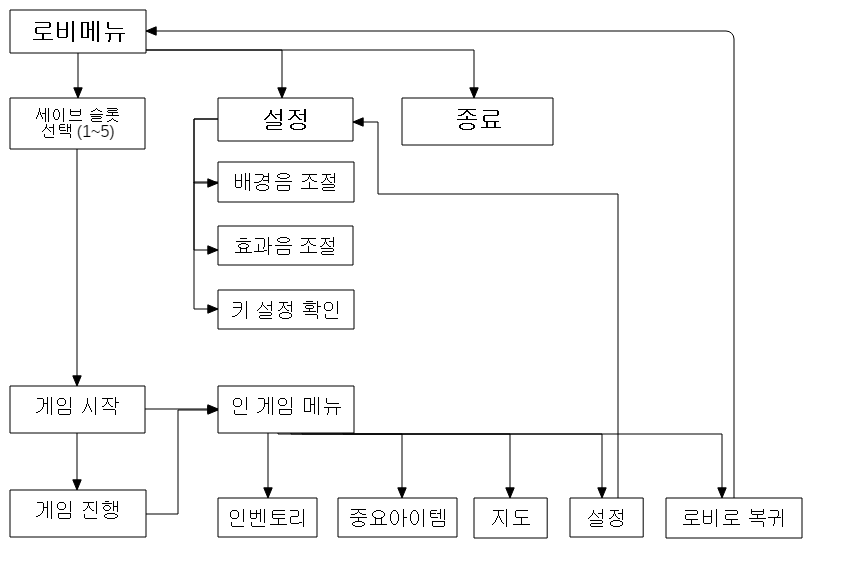
[4.6.3. GameUI 26](#_Toc527077205)

[4.6.4. InGameMenu 27](#_Toc527077206)

[4.6.5. Inventory와 Item 28](#_Toc527077207)

[5. 추가 필요 기능 및 개선점 29](#_Toc527077208)

# UI, UX 기획



[그림 1-1] : UI 기본 순서도

## 로비 화면

세이브 슬롯 1

세이브 슬롯 2

세이브 슬롯 3

세이브 슬롯 4

세이브 슬롯 5

설정

종료

[그림 1-2] : 로비 화면

게임의 로비화면에선 그림 1-2와 같이 세이브 슬롯을 택일하여 게임을 시작할 수도 있고, 설정을 들어가서 게임 내 설정을 바꿀 수도 있다. 종료는 프로그램을 종료한다.

배경음

효과음

키 설정 보기

이전으로

배경음 조절 바

효과음 조절 바

[그림 1-3] : 설정 화면

설정 화면은 배경음과 효과음을 조절할 수 있게 해서, 배경음과 효과음 오른쪽 끝에는 현재 볼륨의 크기를 알 수 있는 진행바를 배치한다.

또한 키 설정 보기는 게임에서 쓰이는 키에 대한 설명을 볼 수 있게 한다.

이전으로를 누르면, 로비화면으로 돌아가거나 게임 중에는 인게임메뉴로 돌아갈 수 있게 한다.

## 인 게임

[그림 1-4] : 인 게임 UI

퀵슬롯

체력 바

게임 화폐 표시

인 게임 UI는 왼쪽 위 모서리에 밀집해 있는 형태다. 퀵슬롯은 소모성 아이템을 등록하는 곳으로 플레이어가 사용할 아이템을 인벤토리에서 등록하면 게임내에서 바로 사용할 수 있게 되어있다. 퀵슬롯엔 최대 3개의 소모성 아이템을 등록할 수 있게 하고, 진행중에 언제든지 바꿀 수 있게 한다. 체력바는 플레이어의 체력을 확인할 수 있는 바 형태로 제작하고 체력이 가득 찼을 때는 붉은색으로 채워지게 하고 체력이 닳게 되면 붉은색 영역이 줄어드는 형태로 제작한다.

게임 화폐 표시 부분은 게임 중 얻은 화폐의 개수를 볼 수 있게 제작할 예정이다.

## 인 게임 메뉴

키 설명

5

버튼1

2

3

4

버튼 설명

[그림 1-5] : 인 게임 메뉴

인 게임 메뉴는 다섯 가지 버튼이 있고 그 왼쪽에 해당 메뉴에 대한 설명을 표시할 수 있게 구성한다. 버튼 1은 인벤토리이고, 2번은 중요아이템, 3번은 맵, 4번은 설정, 5번은 메인메뉴 돌아가기다.

## 인벤토리

5

버튼1

2

3

4

아이템 설명

None

아이템 1

아이템 2

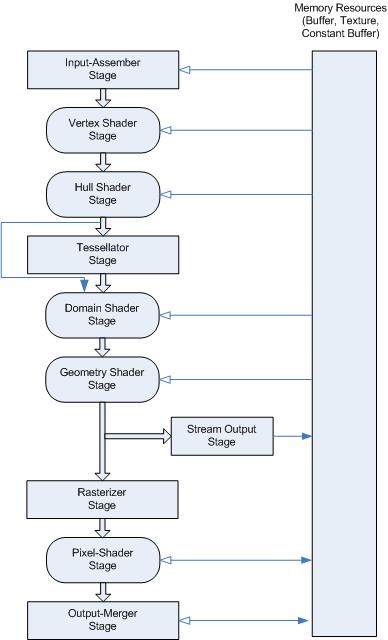
아이템 3

[그림 1-6] : 인벤토리

오른쪽 아이템 칸은 인벤토리에 저장된 아이템의 이름이 쭉 나열된다. 아이템은 소모성과 장비 두 가지로 나눠서 커서의 위치가 버튼 1,2,3이면 아이템 목록은 소모성 아이템만 나오게 하고, 커서가 버튼 4,5면 장비 아이템이 나오게 한다. 커서는 버튼 1~5까지 방향키로 자유자재로 움직이게 하고, 슬롯에 등록을 위해 A키를 누르면 인벤토리 아이템목록으로 커서가 넘어가게 한다. 커서의 알파값은 계속 변동을 주어서 투명도를 조절한다.

# 게임 엔진 설계

## DirectX Graphic Pipeline

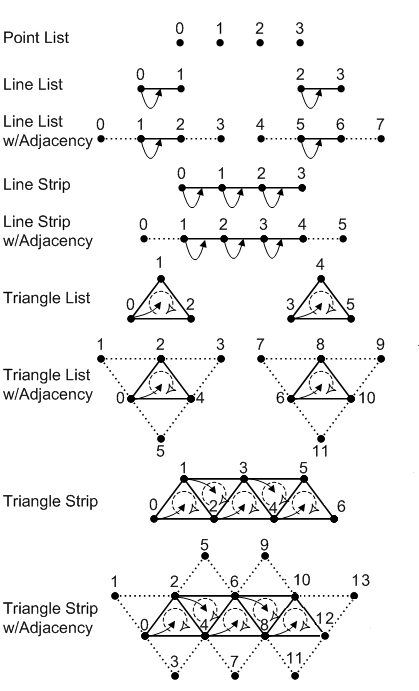


[그림 2-1] : DirectX Graphic Pipeline

DirectX 11은 그림 2-1과 같이 렌더링을 하기 위한 과정을 스테이지별로 나눠서 진행을 한다. 사용자는 버퍼를 Input-Assembler State에 바인딩하게 되며 바인딩 된 버퍼는 화살표 방향대로 스테이지를 거쳐가게 된다. 이 때 공통 쉐이더 코어(둥근 사각형 블록)이 있는 단계는 HLSL 프로그래밍 언어를 사용하여 프로그래밍 할 수 있다. 현재 프로그램에서 쓰인 쉐이더는 Vertex Shader와 Pixel Shader를 사용하였다.

### Input-Assembler Stage

어셈블러 단계의 목적은 사용자가 채운 버퍼에서 데이터를 읽고 다른 파이프라인 단계에서 사용할 기본 데이터로 데이터를 어셈블 하는 것이다. 어셈블러 단계는 다양한 PrimitiveTopology를 통해서 같은 데이터를 다르게 렌더링하는 것이 가능하다.



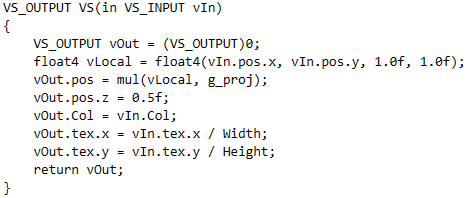
[그림 2-2] : PrimitiveTopology에 따른 차이

그림 2-2와 같이 정점 4개를 Topology를 LineList로 설정을 하면 직선 두개가 나오지만, LineStrip으로 설정하면 각각의 인덱스별로 선을 이어서 3개의 선이 만들어지게 된다.

추가적으로 어셈블러 스테이지에선 시멘틱이라고 불리는 시스템 생성 값을 첨부하여 다음 스테이지로 보내야 하는데, 이를 통해 쉐이더 코어는 해당 생성 값을 참조하여 처리할 수 있게 된다. 현재 프로그램에서는 여기에 추가적으로 정점 4개로 사각형 하나를 렌더링 할 수 있게 하기위해 인덱스 버퍼를 만들어서 바인딩하고 있다.

### Vertex Shader Stage

버텍스 쉐이더 스테이지는 입력 어셈블러에서 들어온 정점을 처리하는 곳으로, 정점 하나당 한번씩 호출이 된다. 파이프 라인을 실행하기 위해 항상 활성화가 되어 있어야 한다.



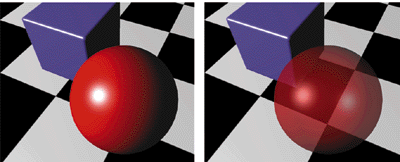
[그림 2-3] : VertexShader

현재 프로그램은 정점의 위치와 텍스쳐의 UV값을 화면 좌표계 기준으로 사용하고 있는데, 버텍스 쉐이더에서 다음 단계로 넘길 시에는 투영좌표 값으로 바꿔줘야 하기 때문에 정점의 위치 값은 행렬 곱을 통해 투영좌표 값으로 변경시키고 UV값은 텍스쳐 이미지의 폭과 높이를 상수 버퍼에 저장해서 그 값으로 나눠주는 과정을 거치고 있다.

### Pixel Shader Stage

버텍스 쉐이더가 정점 하나당 한번씩 호출되듯이, 픽셀 세이더 또한 픽셀 하나당 한번씩 호출이 된다. 픽셀 세이더는 픽셀 하나의 칼라 값을 조정할 수 있는데, 현재 프로그램에서는 텍스쳐의 칼라 값을 그대로 받아와 출력하고, 필요에 따라 알파 값을 조절하면서 페이드 인, 아웃 효과나, 캐릭터가 무적상태일 때 반짝거리는 효과를 주는 용도로 사용이 된다.

## Alpha Blend



[그림 2-4] : AlphaBlending

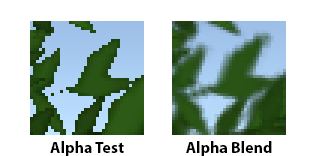
알파 블렌딩은 하나의 픽셀에 두개 이상의 텍스쳐를 표현하기 위해 사용한다. 일반적으로 불투명 텍스쳐 위에 투명 텍스쳐를 렌더링 하게 되는데 투명 텍스쳐의 알파값을 통해서 두 개의 텍스쳐의 색상을 섞는 방식이다.

대표적인 방법 중에 하나로 선형 합성이 있는데, 선형합성은

최종 색 = 배경 색 \* (1.0f – 소스알파 값) + (소스 색 \* 소스 알파 값) 의 식을 통해 두개 이상의 텍스쳐를 하나의 픽셀에 렌더링 하게 된다.

## Alpha Test

알파 블렌딩이 배경의 이미지와 현재 렌더링 하고자 하는 텍스쳐의 알파값을 섞어서 쓰는 방식이라면, 알파 테스트는 현재 텍스쳐의 알파값을 0 또는 1 두가지로 바꾼다.



[그림 2-5] : Alpha Test와 Alpha Blend

이로 인해 가장 두드러지게 나타나는 부분이 텍스쳐의 가장자리 부분에 있다. 그림 2-5와 같이 알파 테스팅은 알파값이 0과 1로만 표현을 하다 보니 가장자리 부분이 부자연스럽게 표현이 된다. 반면에 알파블렌딩은 0과 1사이에 알파값 또한 배경과 연산을 통해 색을 섞기 때문에 표현이 좀더 부드럽게 나오는 것을 볼 수 있다.

결과적으로 알파테스트는 디테일한 부분에선 알파블렌딩에 비해 떨어지지만 알파값을 0과 1로만 계산을 하기 때문에 비교적 속도가 빠르고, 알파블렌딩은 속도는 떨어지지만 렌더링이 좀 더 자연스럽게 된다.

# 게임 개요

|  |  |
| --- | --- |
| **모모도라: 달 아래의 진혼곡** **Momodora: Reverie Under the Moonlight** | |
| [파일:external/40f582ef0563db78a09d4ae667c39c474365a1f54811e43f01dd4d036f7792d7.jpg](https://namu.wiki/w/%ED%8C%8C%EC%9D%BC:external/40f582ef0563db78a09d4ae667c39c474365a1f54811e43f01dd4d036f7792d7.jpg) | |
| **개발** | [Bombservice](http://bombservice.com/) |
| **플랫폼** | PC, [PS4](https://namu.wiki/w/PS4), [XBOX ONE](https://namu.wiki/w/XBOX%20ONE) |
| **출시** | PC: 2016년 3월 4일 PS4: 2017년 3월 16일 XBOX ONE: 2017년 3월 17일 |
| **장르** | [메트로배니아](https://namu.wiki/w/%EB%A9%94%ED%8A%B8%EB%A1%9C%EB%B0%B0%EB%8B%88%EC%95%84) |
| **엔진** | [게임메이커 스튜디오](https://namu.wiki/w/%EA%B2%8C%EC%9E%84%EB%A9%94%EC%9D%B4%EC%BB%A4%20%EC%8A%A4%ED%8A%9C%EB%94%94%EC%98%A4) |
| **링크** | [스팀](http://store.steampowered.com/app/428550/) |

## 게임 소개

제작 게임은 모모도라 시리즈의 네 번째에 해당하는 작품을 모작하였다.

장르는 메트로배니아 형태로, 흔히 얘기하는 2D 횡스크롤 즉, 사이드뷰 플랫폼이 기반으로 된 게임으로, 게임상에 하나의 거대한 맵 이 존재하고 플레이어는 그 맵 을 하나하나 밝히면서 게임을 진행하는 방식이다.

맵 이 거대하기 때문에 탐색의 비중이 높고, 그에 따라 맵 곳곳에 숨겨진 아이템을 찾는 요소도 많이 들어있다.

그 밖에도 게임 초반에 갈 수 없는 지형들도 존재하며, 이런 지형들은 플레이어가 플레이 도중 얻은 능력들로 갈 수 있게 되어있고, 이런 능력들은 단순히 갈 수 없는 지형을 가게 되는 효과외에 전투에도 다양한 도움이 되어서 플레이어의 성장을 느낄 수 있다.

## 게임의 진행

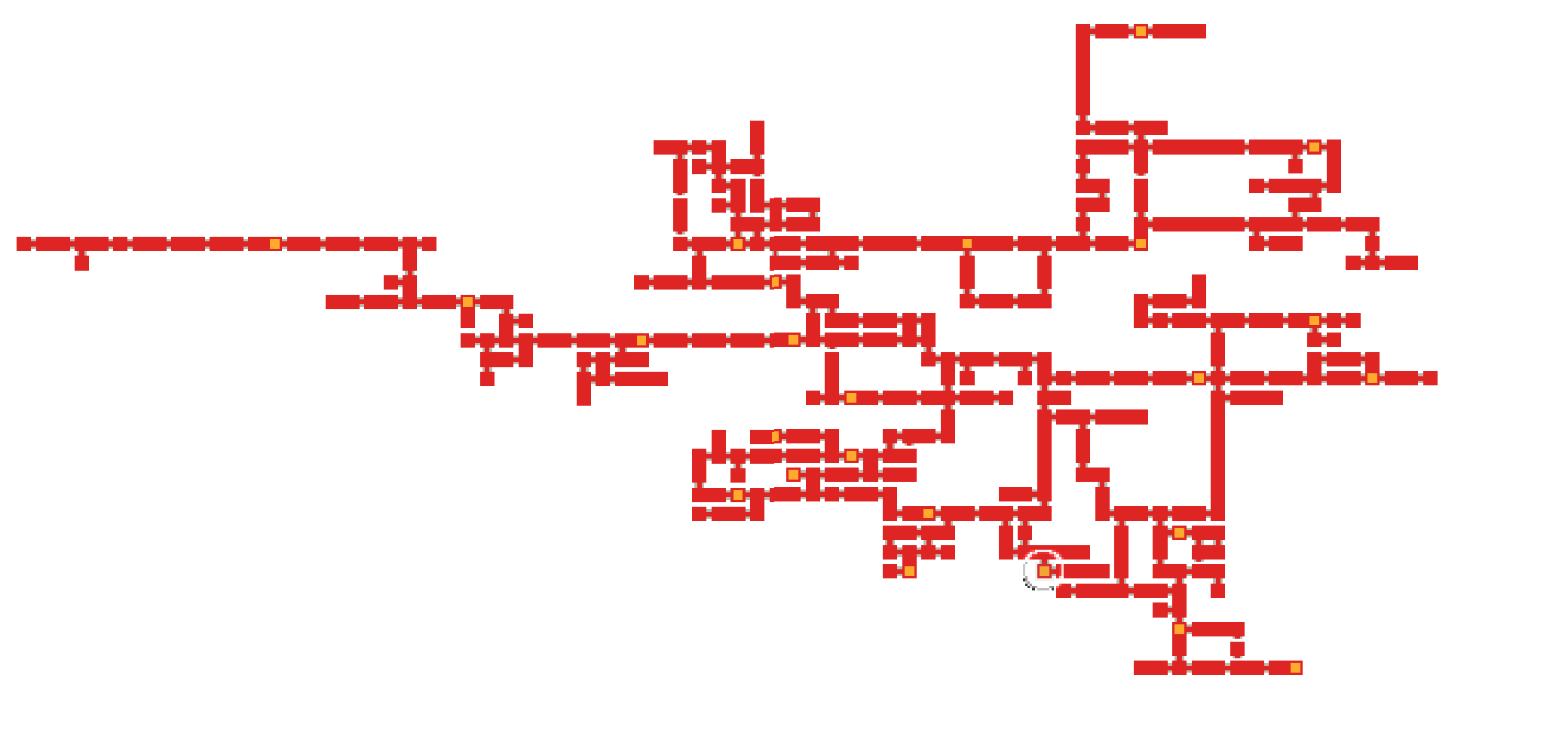
게임은 하나의 Scene안에서 진행이 되며 Scene끝으로 도달하게 되면 다음 Scene으로 넘어가는 방식으로 진행된다.

하나의 스테이지가 존재하고 그 안에 여러 Scene이 포함되 있는 형태이며, 스테이지가 모두 모여 게임의 전체적인 맵 이 형성 돼있다.

단순히 일방적인 길이 아닌 중간에 아래로 떨어지거나 위로 올라가는 길도 존재하며 그 과정에서 플레이에 도움이 되는 아이템 등을 얻을 수 있다.

플레이어는 두 가지 공격방식으로 전투를 진행하게 되는데, 안정적이나 피해량은 낮은 원거리 공격을 할 수도 있고 몬스터와 붙어서 위험성은 크나 피해량은 높은 근접 공격을 사용할 수도 있다.

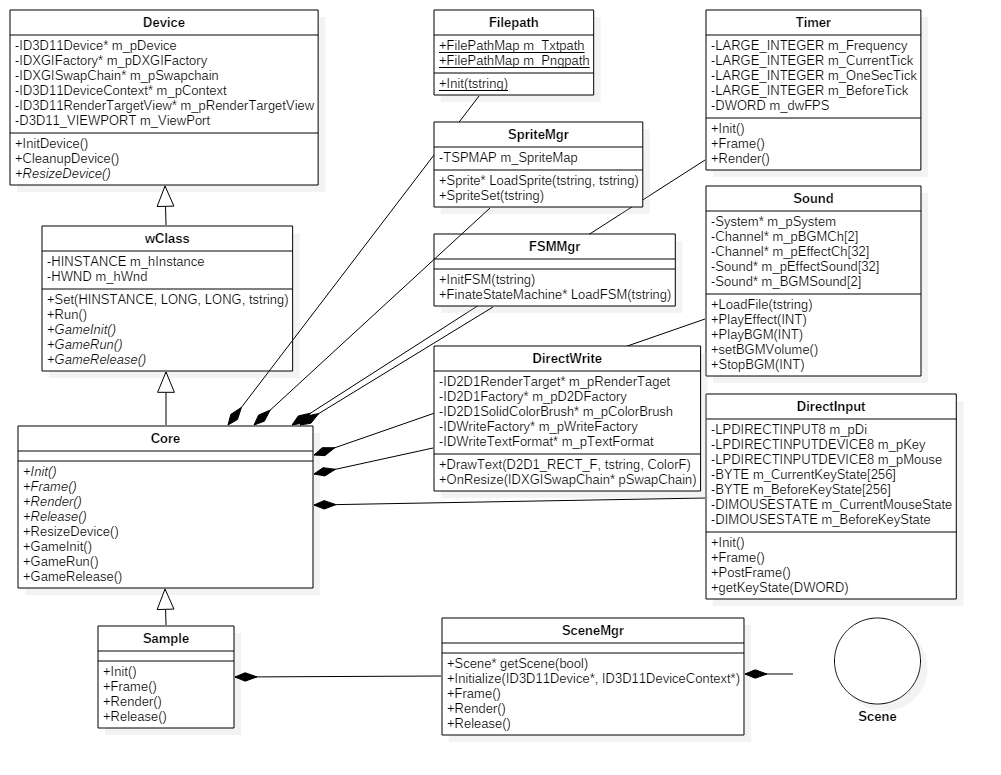
이런 방식은 기존에 익숙한 횡스크롤 게임과 크게 다른 점은 없다고 볼 수 있다.



[그림 3-1] : 게임의 맵. 칸 하나가 하나의 Scene을 이룬다.

# 게임 시스템

## Core 와 Sample 클래스



[그림 4-1] : Core 와 Sample의 클래스 다이어그램

Device는 DirectX를 사용하기 위해 기본적인 장치들을 생성하는 클래스로, 이를 윈도우를 띄우는 wClass가 상속하는 구조로 되어있다.

이를 다시 Core 클래스가 상속하는데, Core에서는 게임에 사용되는 전반적인 클래스들을 초기화 한다. Filepath 클래스는 게임에 쓰이는 리소스들의 경로를 저장하는데 쓰이는데, 함수와 멤버 모두 정적으로 구성되어있다.

SpriteMgr와 FSMMgr는 각각 Sprite 와 FSM을 초기에 데이터를 저장하여 필요할 때 불러올 수 있도록 구성하였다.

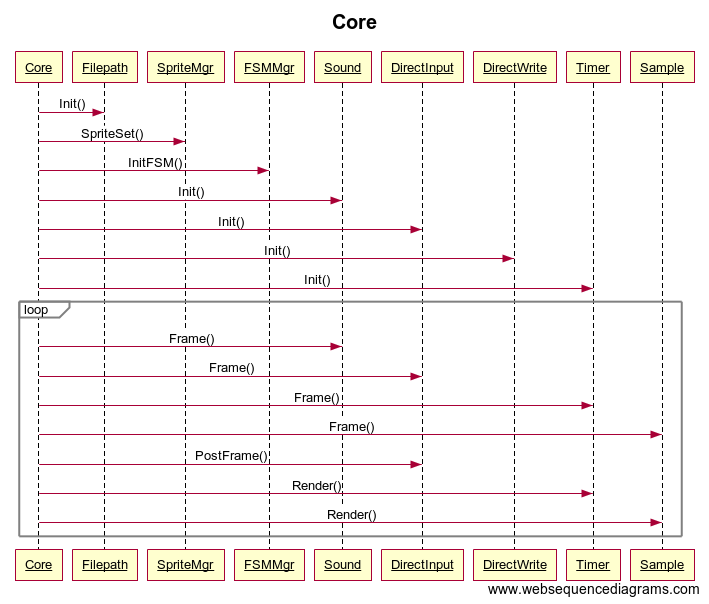
DirectWrite는 화면에 문자를 출력하기 위한 클래스인데, 현재는 FPS와 게임시간 SPF를 출력하는 용도로만 사용하고 있다.

Timer는 게임의 시간에 관련된 부분들을 계산하는 클래스로, 위에서 말한 FPS와 게임시간 SPF를 Timer에서 계산을 한다. Sound는 게임에서 쓰는 사운드들을 출력하기 위한 클래스로 게임내에서 배경음과 효과음을 따로 조절하기 위해 채널과 사운드를 배경과 효과음 따로 나눠서 저장하고 있다.

DirectInput은 키의 입력을 받아 처리하는 클래스인데, 키의 입력을 홀드상태와 업상태를 구분하기 위해서 현재의 키 상태와 이전 프레임의 키 상태를 저장한다.

Frame을 호출한 뒤에 코어에 마지막에 PostFrame을 호출해서 현재 상태를 이전 상태에 저장하는 방식이다.

Sample은 이 Core를 상속하는 클래스로, 내부에서 SceneMgr 프레임과 렌더를 호출해서 게임이 진행이 된다.

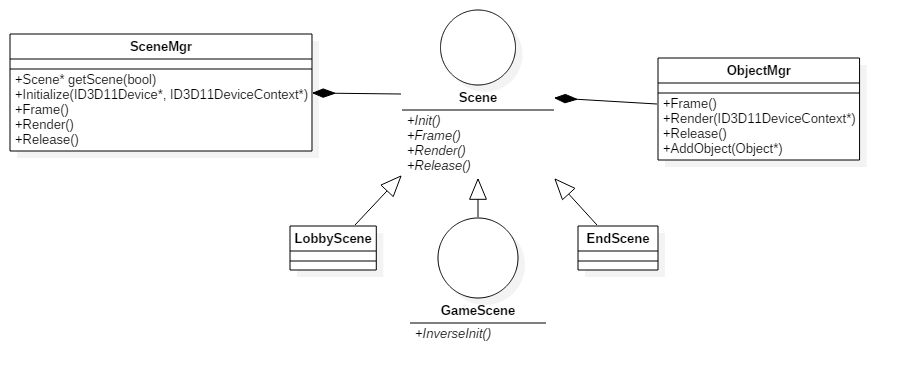


[그림 4-2] : Core의 시퀀스 다이어그램

## SceneMgr와 Scene 클래스

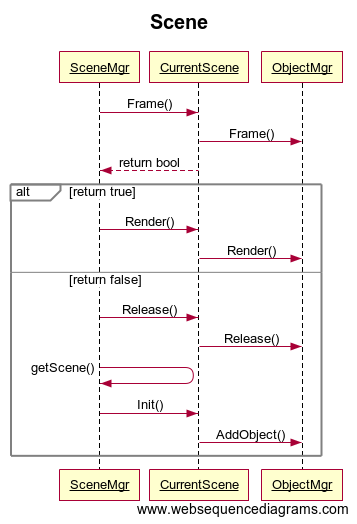
SceneMgr는 Scene을 멤버로 둬서 현재의 Scene의 Frame과 Render를 호출한다. Scene의 전환이 일어나면 현재의 Scene을 Release한 뒤, getScene함수를 통해서 바뀌는 Scene을 생성해서 다시 Frame과 Render를 호출한다.

Scene은 생성을 할 때, 오브젝트들을 생성해서 배치 한 뒤, ObjectMgr에 추가하여 ObjectMgr의 Frame과 Render를 호출한다.



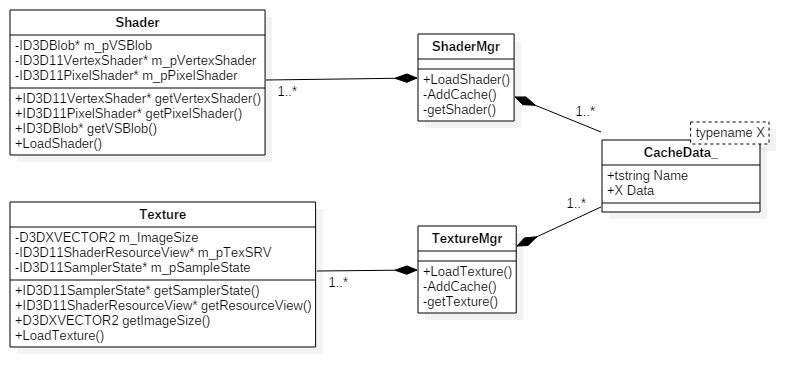
[그림 4-3] : Scene과 SceneMgr의 클래스 다이어그램

게임 진행중에 지나온 Scene도 다시 돌아갈 수 있게끔, GameScene은 추가적으로 InverseInit 함수를 재정의하게 하여, 플레이어가 뒤 Scene에서 앞 Scene으로 올 때에는 오브젝트를 반대로 배치할 수 있도록 하였다.



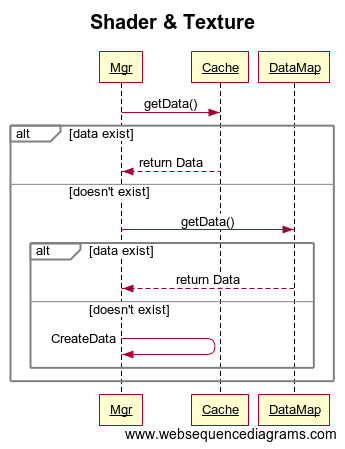
[그림 4-4] : Scene과 SceneMgr의 시퀀스 다이어그램

## Shader와 Texture 클래스



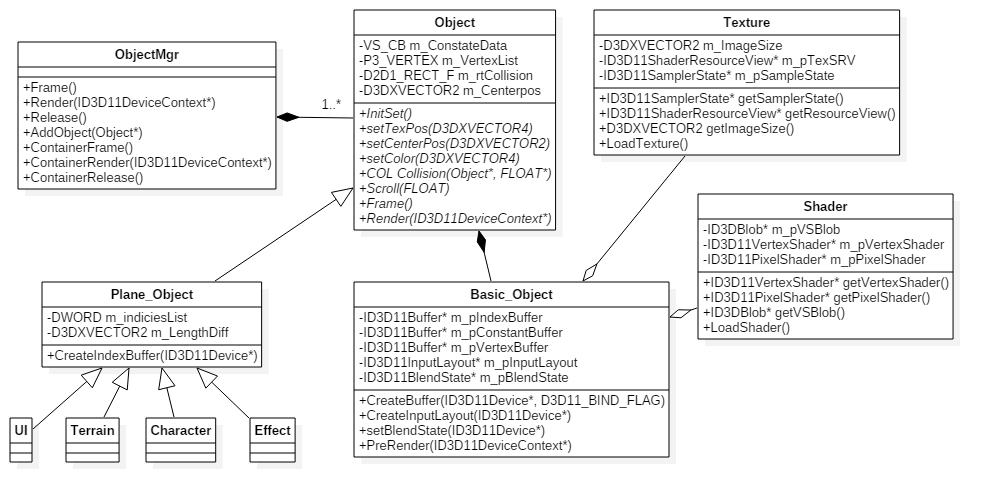
[그림 4-5] : Shader와 Texture의 클래스 다이어그램

Shader와 Texture는 모두 매니저 클래스를 통해 관리하며, 매니저의 Load함수를 호출하면 우선 사용자가 원하는 데이터가 캐시에 있는지 확인을 한다. 있으면 바로 그 데이터를 반환하고, 없을 경우는 전체 데이터를 통해서 똑같은 작업을 한다. 전체데이터에도 없으면 만들어서 반환하는 구조로 되어있다.



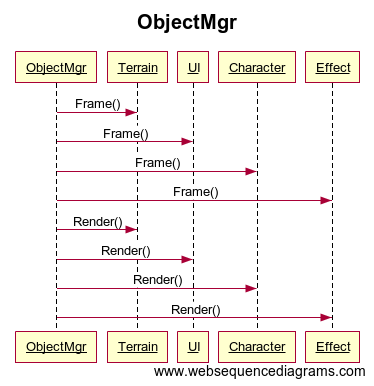
[그림 4-6] : ShaderMgr과 TextureMgr의 시퀀스 다이어그램

## ObjectMgr과 Object 클래스



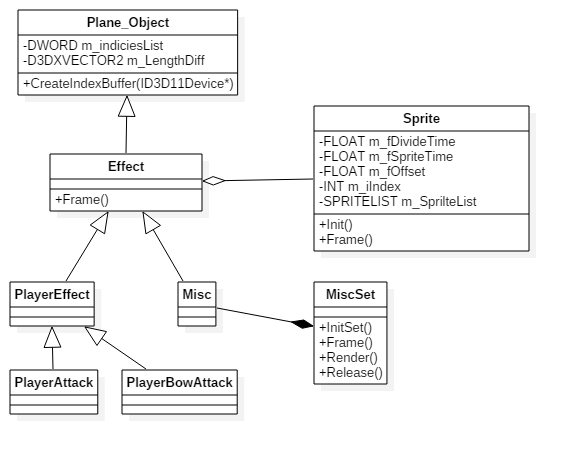
[그림 4-7] : ObjectMgr과 Object의 클래스 다이어그램

Basic\_Object는 오브젝트 생성시에 필요한 기본적인 버퍼 등을 가지고 있는 클래스로, Object는 이 Basic\_Object를 멤버로 가진다. Object는 상수 버퍼를 사용하기위한 구조체와 정점들을 STL 벡터에 보관하며 충돌체크를 위한 충돌크기와 센터위치 값을 가진다. Plane\_Object는 정점 4개로 사각형을 출력하기 위한 클래스로 이를 위해 정점의 인덱스들을 벡터에 저장하게 하였다. ObjectMgr은 오브젝트들을 관리하기 위한 클래스로 Plane\_Object를 상속받는 클래스별로 따로 관리를 한다. ObjectMgr엔 컨테이너를 순회하며 Frame과 Render를 호출하는 함수가 템플릿으로 정의를 해서, 각각 컨테이너별로 따로 Frame과 Render를 호출한다.



[그림 4-8] : ObjectMgr의 시퀀스 다이어그램

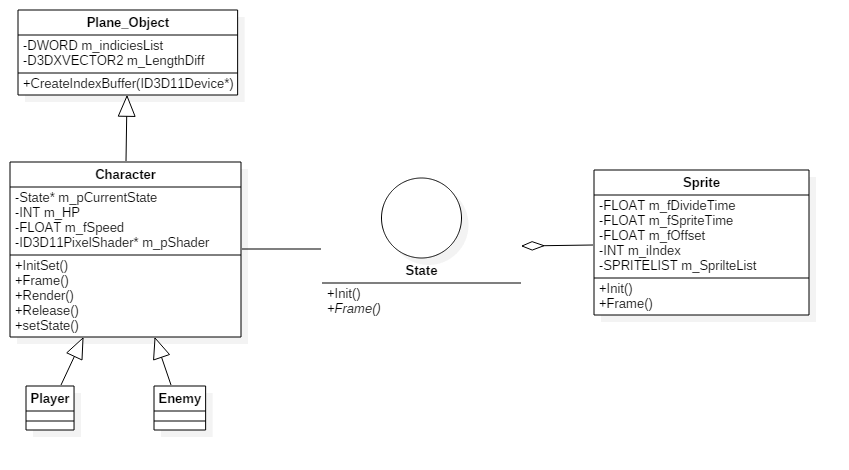
### Effect클래스



[그림 4-9] : Effect의 클래스 다이어그램

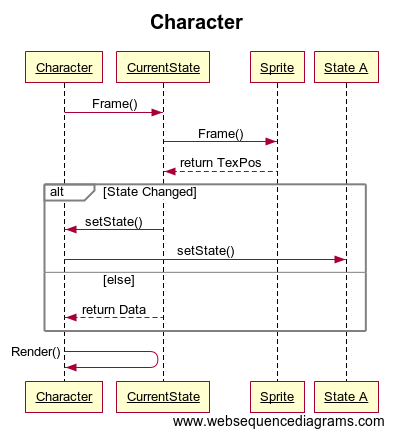
PlayerEffect는 플레이어의 공격에 관련된 이펙트가 있다. 근접공격 3단계와 공중 공격, 화살이펙트가 이 클래스를 상속한다. Misc는 플레이어의 버프효과를 나타내기 위한 클래스로 MiscSet은 이 Misc클래스를 배열로 둬서 하나의 세트를 구성한다. 현재 구현되 있는 아이템인 수정꽃을 사용하면 플레이어의 주위에 4개의 이펙트가 돌아다니게 되는데, 이를 통해 버프의 지속시간을 확인할 수 있게하였다.

### Character 클래스



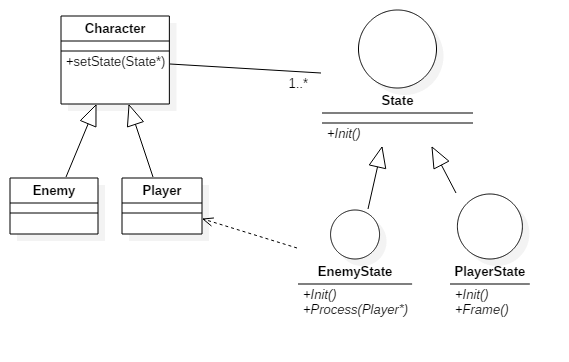
[그림 4-10] : Character의 클래스 다이어그램

Character는 State 클래스와 함께 사용이 된다. State클래스는 캐릭터의 상태를 구현하는 클래스로 캐릭터의 이동이나 점프 공격 등 모든 상태를 구현하고 그 상태를 통해서 행동을 제약하는 방식으로 구성이 돼있다. Character는 자신의 모든 상태를 STL map에 저장하고 있다. State와 Character는 상호 참조 가능한 형태로, 만약 현재 상태에서 다른 상태로의 변화가 발생하면 State는 Character의 setState로 현재의 상태를 변경 시키고, Character는 해당하는 상태로 현재상태를 바꾸고 계속해서 Frame과 Render를 호출하는 방식이다.



[그림 4-11] : Character의 시퀀스 다이어그램

## State



[그림 4-12]: State의 클래스 다이어그램

State는 EnemyState와 PlayerState로 나뉘어져있다.

### PlayerState

플레이어의 상태는 키보드 입력키에 의해 변하게 된다.

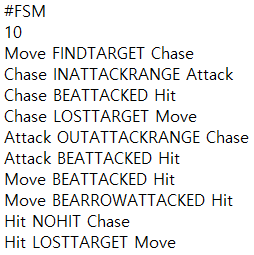
방향키로 이동을 하고 A키로 점프, S키로 근접공격, D키로 활공격을 할 수 있다. 실제 원작 게임에선 플레이어의 상태가 42개가 구현되어 있으나, 현재까지는 22개의 상태가 구현이 되어있다.

### EnemyState

#### 

[그림 4-13]: EnemyState의 클래스 다이어그램

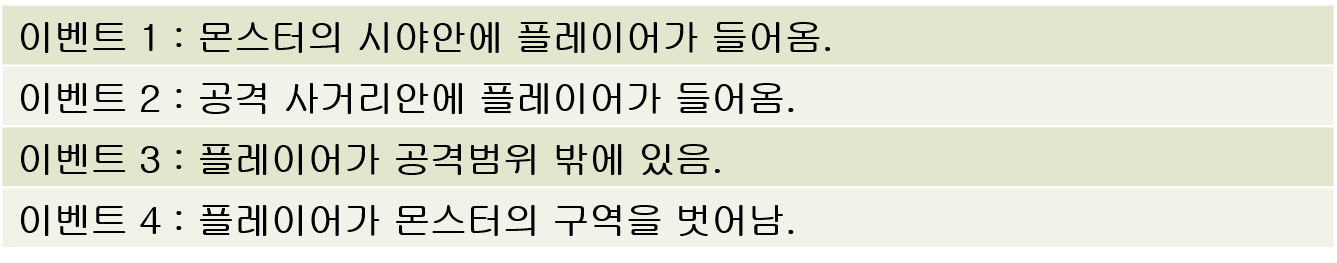
키보드로 상태가 변하는 플레이어와 달리 적 같은 경우는 상태별로 이벤트 발생 시 변경될 상태가 정의되어 있는 FSM이 필요하다.



[그림 4-14]: 스크립트로 저장한 FSM

적들은 생성시에 FSMMgr을 통해서 하나의 FSM을 받아서 그 FSM에 맞게 행동을 한다. 이벤트가 발생하면 FinateStateMachine의 StateTransition 함수를 호출해서 반환되는 상태값으로 변경이 되는 방식이다. 그림 2-18을 예로들어서 현재 적이 Move 상태 일 때 플레이어를 발견해서 FINDTARGET이 발생하면 플레이어를 추적하는 Chase 상태가 된다.

현재 게임에서 구현 되어있는 FSM은 한 종류로 다음과 같다.



**이벤트1**

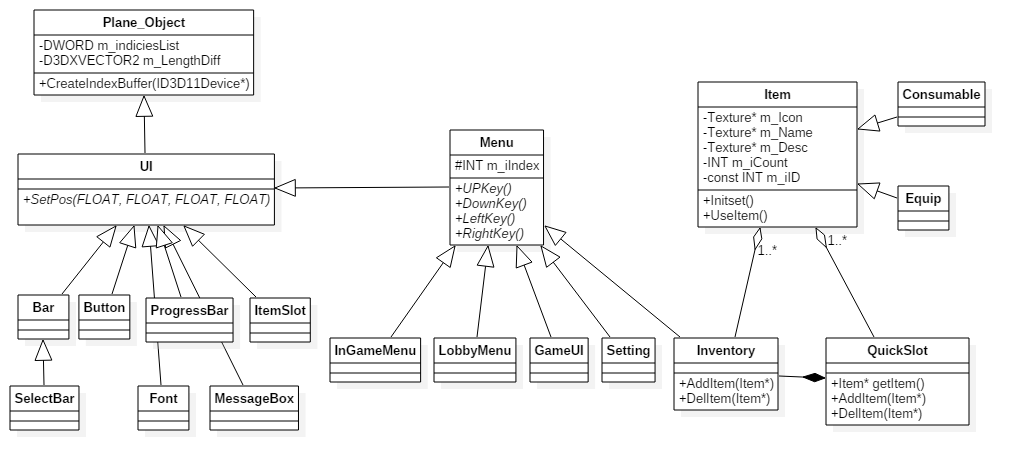
**이벤트4**

**이벤트2**

**이벤트3**

[그림 4-15] : 몬스터 FSM

## UI



[그림 4-16] : UI의 클래스 다이어그램

UI는 기본 구성요소들과 그 기본 구성요소들을 합하여 하나의 완성된 UI인 Menu 클래스 두 종류로 나눌 수 있다.

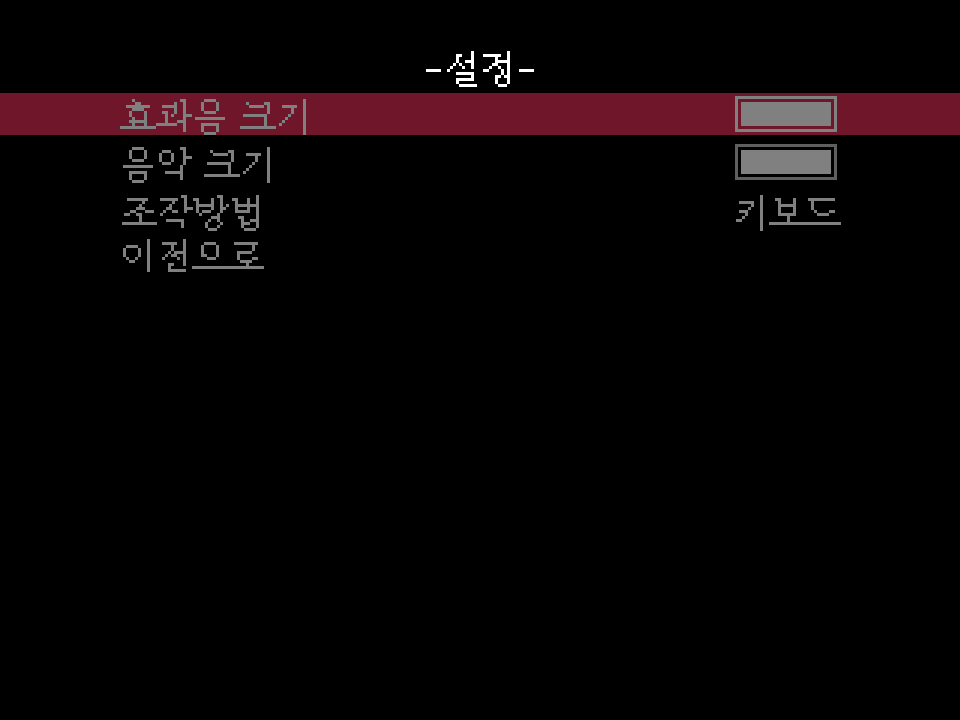
### LobbyMenu



[그림 4-17] : LobbyMenu

로비 메뉴는 6개의 Bar와 1개의 Select Bar로 이루어져 있다. 방향키 위, 아래 버튼을 누르면 활성화된 슬롯이 바뀌는 방식으로 작동한다.

### Setting



[그림 4-18] : 설정 화면

설정화면은 3개의 Bar와 1개의 SelectBar 그리고 2개의 ProgressBar로 이루어져 있다. 로비와 마찬가지로 위, 아래 키로 슬롯을 이동할 수 있으며, 추가적으로 효과음과 음악에 커서가 위치하면 화살표 좌, 우 키를 통해 볼륨을 조절할 수 있는데, 줄이게 되면 오른쪽에 바가 작아지고 키우게 되면 바가 커지는 형식으로 만들어져 있다.

### GameUI



[그림 4-19] : GameUI

GameUI는 3개의 Bar와 1개의 ProgressBar와 한 개의 ItemSlot, 4개의 폰트로 제작하였다.

GameUI는 게임이 진행되는 동안 항상 왼쪽 위 모서리에 고정된 상태로 출력이 되며, 왼쪽 끝에 있는 아이템 슬롯은 현재 퀵슬롯에 등록된 아이템의 아이콘과 해당 아이템의 개수를 보여준다. 또한 퀵슬롯에 다른 아이템이 등록되어 있다면, E키를 통해서 언제든지 다음 아이템으로 전환할 수도 있다. HP바는 플레이어의 HP가 최대치일때는 그림과 같이 바가 가득 찬 상태로 있고, HP가 줄어들면 바의 크기도 같이 줄어들게 된다.

그 아래의 숫자 3개는 게임상 화폐를 보여주는 UI인데, 현재까지는 구현은 되어있지 않다.

### InGameMenu



[그림 4-20] : InGameMenu

인게임메뉴는 게임 진행중 왼쪽 Shift키를 통해 열 수 있다. 배경에 쓰이는 Bar 2개와 Button 5개, 버튼에 해당하는 메뉴설명인 Font로 구현을 하였다.

화살표 좌, 우 키로 이동하고 원하는 메뉴에서 A키를 누르면 넘어가게 되는데, 처음부터 장비, 중요아이템, 지도, 설정, 메인메뉴로 돌아가기 순으로 되어있다.

현재 지도는 구현이 되어있지 않다.

### Inventory와 Item



[그림 4-21] : Inventory

Item은 소모성 아이템과 장비아이템으로 나눌 수 있다. 활성 슬롯 밑에 있는 슬롯3칸은 퀵슬롯으로 소모성 아이템 같은 경우는 이곳에 등록을 한 뒤 게임에서 등록된 다른 아이템으로 바꿔가면서 사용할 수 있다. 지속 슬롯 밑 2칸은 장비 아이템을 등록하는 슬롯이다.

인벤토리는 소모성 아이템과 장비 아이템을 따로 저장하여 관리하며, 그로 인해 활성 슬롯에 커서가 올라가 있으면 인벤토리 내 소모성 아이템만 출력이 되게 하고, 지속 슬롯에 커서가 올라가게 되면 장비 아이템만 출력이 된다. 인벤토리에서 아이템을 빼서 퀵슬롯에 등록하게 되면 아이템은 인벤토리에서 삭제되고 퀵슬롯으로 옮겨가게 된다. 아이템을 저장하는 곳이 두 곳인 셈이다. 플레이어의 슬롯에 등록된 아이템은 언제든지 다시 인벤토리로 옮길 수 있는데 해제하고 싶은 아이템 커서에다 A키를 누르면 커서가 인벤토리로 옮겨지는데, 아이템을 바꾸고 싶다면 바꾸고 싶은 아이템으로 커서를 옮겨 A키를 누르고 장착해제하고 싶다면 None에다 두고 A키를 누르면 된다. 퀵슬롯에서 아이템을 해제하면 반대로 퀵슬롯에서 인벤토리로 아이템이 넘어가게된다. 인벤토리는 아이템이 들어올 때 정렬을 하게 되는데, 각 아이템은 고유의 ID값이 존재하고 인벤토리 내에서는 이 ID값을 기준으로 정렬을 한다.

# 추가 필요 기능 및 개선점

추가 필요 기능은 우선 타격 시 이펙트를 구현하고, 그에 맞게 타격감을 느껴볼 수 있게 몬스터가 플레이어의 공격에 피해를 입으면 화면에 약간의 흔들림을 주는 것이다. 현재 구현되 있는 부분까지 플레이하면 게임을 플레이한다는 느낌이 많이 부족하다. 그 이유가 게임 내 효과가 많이 구현 돼있지 못한 게 크다.

그리고 씬 자체도 좀 더 늘려서 그에 맞게 다양한 적들과 오브젝트들을 만드는 것이다.

개선점으로는 FSM을 하나 더 늘려서 두 종류로 만들고자 했는데, 새로 만든 FSM 같은 경우 상태 순환이 원활이 되지 않아서 뺏는데, 이 부분을 고쳐서 잘 작동하는 FSM을 추가하는 것이 있다.