AI 2D 게임

35기 이찬규

목차

[1. 게임의 개요 3](#_Toc526878943)

[1.1. 게임 소개 3](#_Toc526878944)

[1.2. 게임의 진행 4](#_Toc526878945)

[2. 게임의 시스템 5](#_Toc526878946)

[2.1. Core 와 Sample 클래스 5](#_Toc526878947)

[2.2. SceneMgr 와 Scene 클래스 7](#_Toc526878948)

[2.3. Object클래스 9](#_Toc526878949)

[2.3.1. Terrain 10](#_Toc526878950)

[2.3.2. BKObject 11](#_Toc526878951)

[2.3.3. Effect 12](#_Toc526878952)

[2.3.4. Character 13](#_Toc526878953)

[2.3.5. State 14](#_Toc526878954)

[3. 추가 필요 기능 및 개선점 16](#_Toc526878955)

# 게임의 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

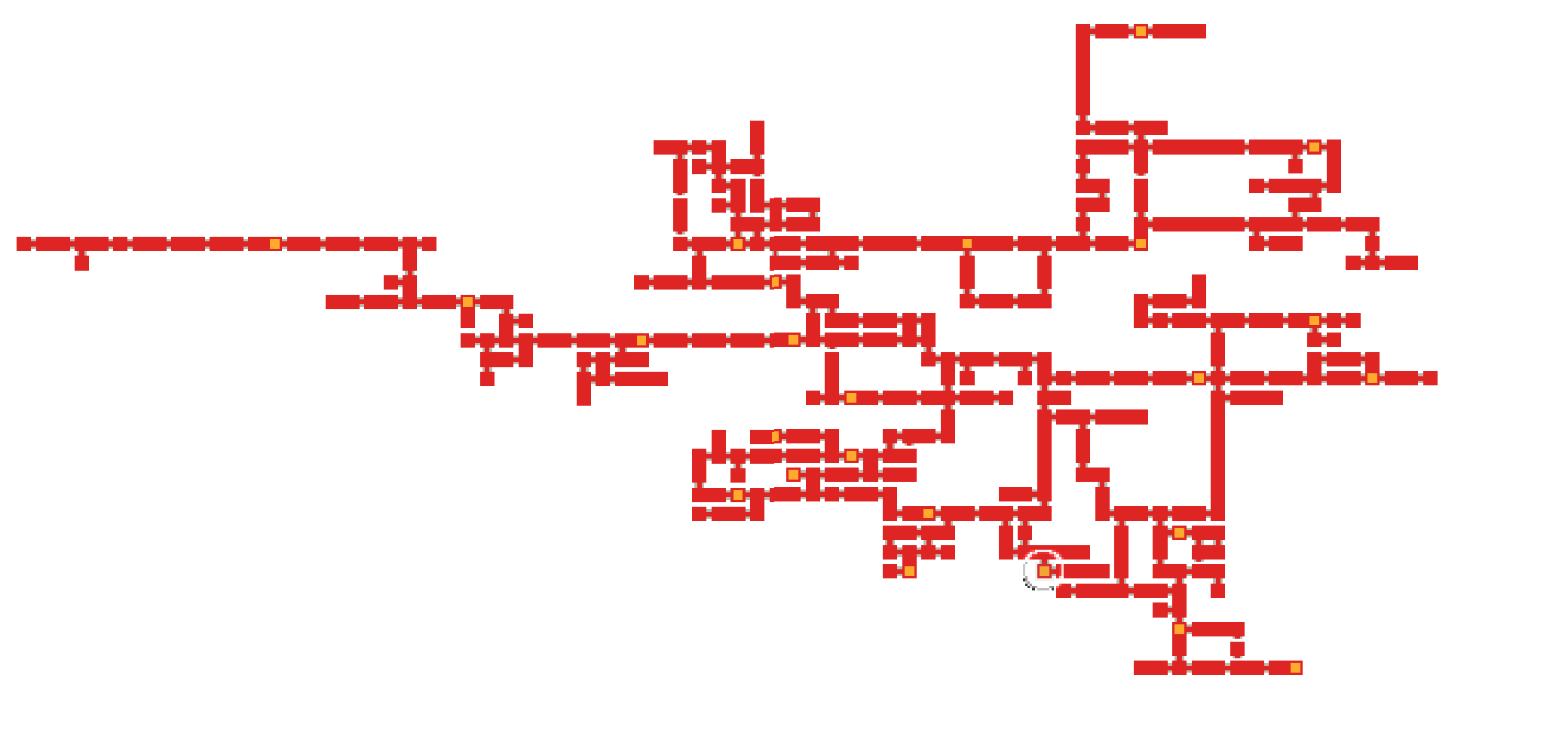
게임은 하나의 Scene안에서 진행이 되며 Scene끝으로 도달하게 되면 다음 Scene으로 넘어가는 방식으로 진행된다.

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.

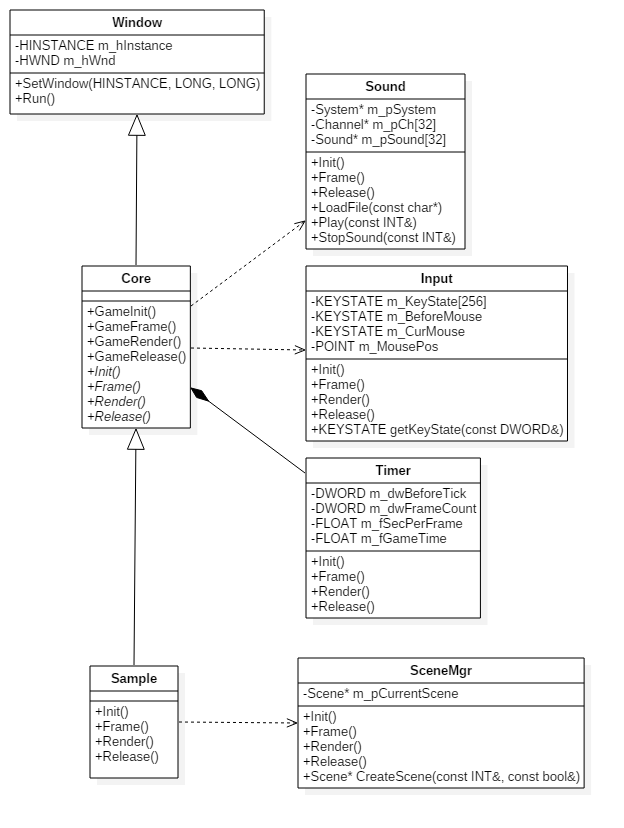
이런 방식은 기존에 익숙한 횡스크롤 게임과 크게 다른 점은 없다고 볼 수 있다.



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

# 게임의 시스템

## Core 와 Sample 클래스



[그림 2-1] : Core 와 Sample의 클래스 다이어그램

Core 클래스는 윈도우를 띄우는데 사용하는 Window 클래스를 상속받아 사용한다. Core의 멤버로는 Timer 클래스가 있다. Timer 클래스는 Frame()의 호출 시간을 체크하여 전에 호출된 시간과 비교한후 FPS와 SPF 그리고 게임이 진행된 총 시간을 계산한다.

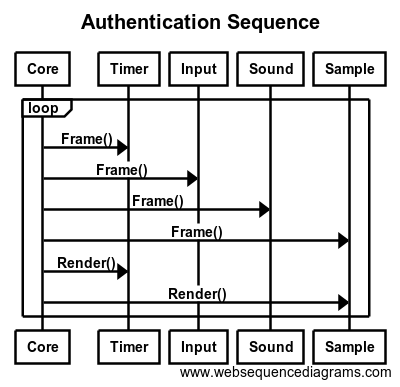
Input 클래스와 Sound 클래스는 싱글톤으로 정의가 되어있다.

Input같은 경우는 현재의 키 상태를 체크하는 getKeyState 함수가 있는데 이를 통해 플레이어의 입력을 처리할 수 있게 된다.

Sound 클래스는 게임 내 사운드를 출력하기 위한 클래스로 Core의 GameInit함수에서 사운드 파일을 불러들여 저장을 한 후, 필요할 때 재생을 시킬 수 있는 구조로 되어있다.

Core 클래스는 Init, Frame, Render, Release 함수를 가상함수로 정의해서 이를 상속하는 클래스의 필요에 따라 재정의 할 수 있게 되어있다.

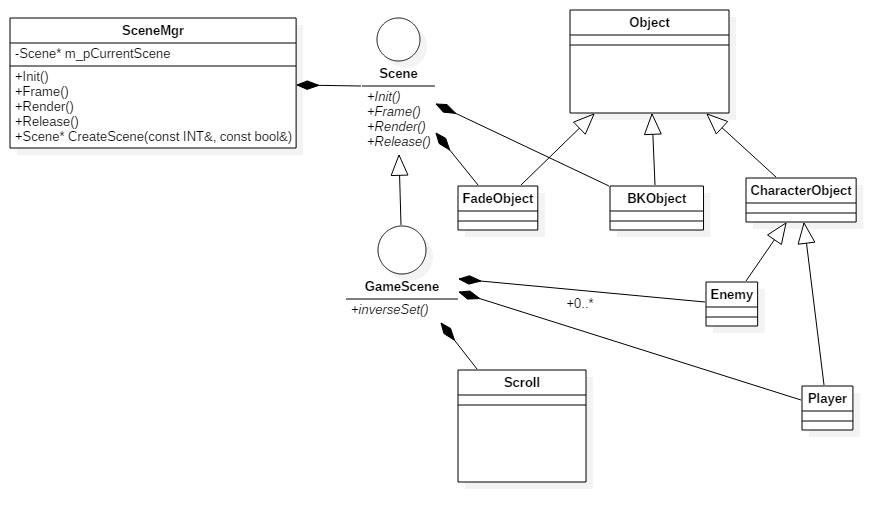
Sample 클래스는 Core 클래스를 상속하는 클래스로 게임의 전반적인 내용을 처리하는 클래스이다.



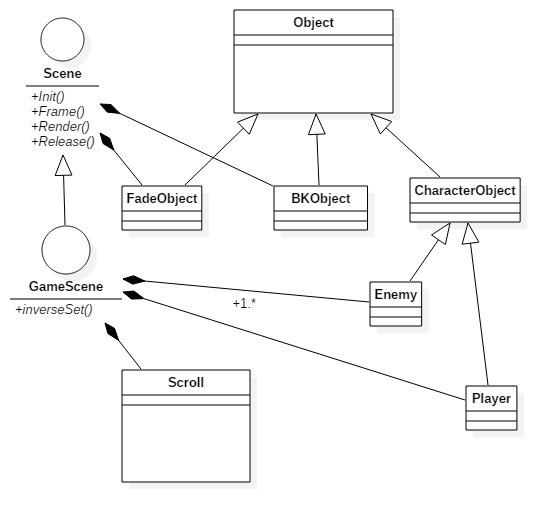
[그림 2-2] : Core의 시퀀스 다이어그램

Core의 GameFrame함수가 호출되면 순서대로 Timer, Input, Sound, Sample의 Frame이 호출되고, 그 후에 GameRender를 통해 Timer와 Sample의 Render를 호출하는 방식으로 구성 돼있다.

## SceneMgr 와 Scene 클래스



[그림 2-3] : SceneMgr의 클래스 다이어그램



[그림 2-4] : Scene의 클래스 다이어그램

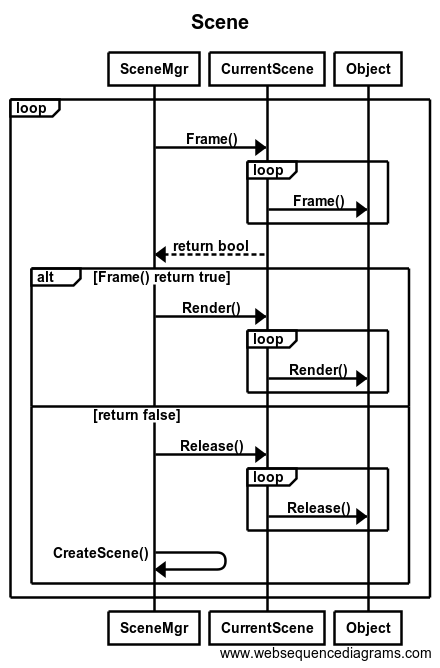
SceneMgr은 Scene을 멤버로 가지고 있고 Frame과 Render에서 현재 Scene 의 Frame과 Render를 호출하는 방식으로 진행된다.

Scene의 전환은 Frame의 반환값으로 파악하며 True가 반환되면 계속 진행하고 False가 반환하면 현재 Scene을 소멸시키고 다음 Scene을 생성해서 진행하는 형태이다.

Scene의 멤버는 여러 오브젝트들이 있는데, 공통적으로 배경을 담당하는 BKObject와 Scene전환마다 페이드아웃효과를 주는 FadeObject클래스가 있다.

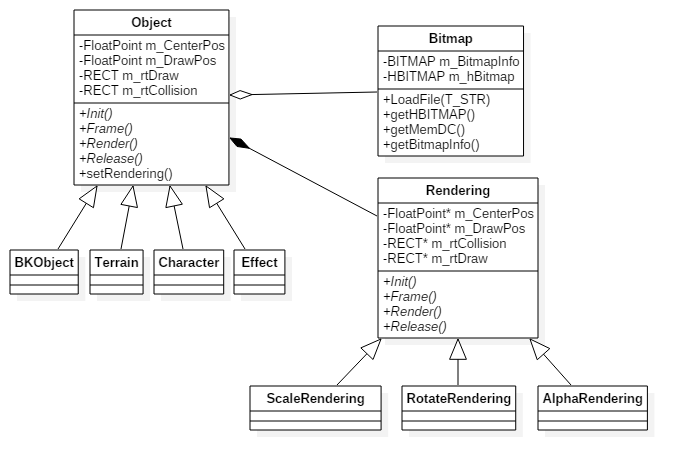
또한 GameScene같은 경우는 Player와 Enemy클래스를 멤버로 가지고, 추가적으로 캐릭터의 이동시에 다른 오브젝트들을 이동시키는 Scroll 클래스가 있다.

현재 프로그램에는 오브젝트를 관리하는 클래스가 존재하지 않고, 이렇게 Scene내에 오브젝트들을 배치하고 관리하는 방식으로 진행된다.



[그림 2-5] : SceneMgr와 Scene의 시퀀스 다이어그램

## Object클래스



[그림 2-6] : Object의 클래스 다이어그램

Object 클래스를 구성하는 멤버 중 하나인 Bitmap은 비트맵 파일을 읽어 드려서 그 파일의 정보를 저장하는 클래스이다.

Rendering 클래스는 화면에 출력을 담당하는 클래스다.

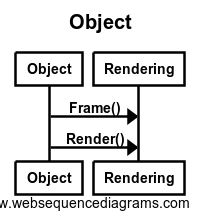
Object는 Render함수가 호출 되면 현재 자신의 Rendering의 Render를 호출하는 방식으로 구성돼 있다.

Rendering은 확대와 반전이 가능한 ScaleRendering, 회전을 줄 수 있는 RotateRendering, 알파블랜드를 할 수 있는 AlphaRendering으로 나뉘어져 있다.

Rendering은 생성시에 해당 Object객체를 인자로 받아서 그 객체의 센터 값, 출력위치 값, 충돌크기, 비트맵의 포지션 값을 포인터로 저장한다.

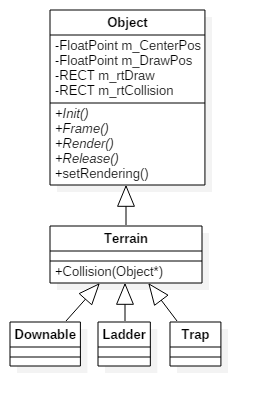
그렇게 함으로써 Object의 Frame()에서 바뀌는 여러 데이터들을 Render를 할 때 포인터의 값을 읽어드리면서 그대로 출력을 하게 된다.

또한 Object에는 setRendering 함수가 구현 돼있어서, 언제든지 다른 Rendering으로 바꿀 수 있도록 되어있다.



[그림 2-7]: Object의 시퀀스 다이어그램

### Terrain



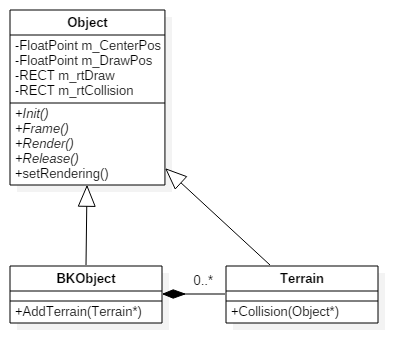
[그림 2-8] : Terrain의 클래스 다이어그램

Terrain은 게임에서 쓰이는 지형을 표현하기 위한 클래스다.

기본적인 지형 외에 플레이어가 아래로 떨어질 수 있는 Downable과 사다리 역할을 하는 Ladder, 그리고 닿으면 즉사를 하는 Trap클래스가 존재한다.

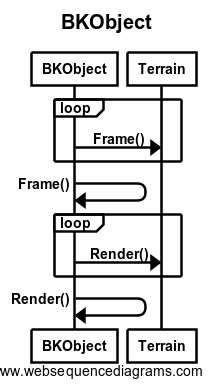
또한 다른 오브젝트와 충돌 처리를 하는 Collision함수가 들어 있다. Collision함수는 다른 오브젝트와의 충돌을 계산할 때, 충돌 여부를 계산한 후에 충돌이 일어났으면 충돌영역을 다시 계산하여 충돌의 방향을 계산하는데 이를 통해 Downable같은 경우는 플레이어의 좌우와 밑에서 위로오는 충돌은 무시하면 위에서 오는 충돌만 계산을 한다.

### BKObject



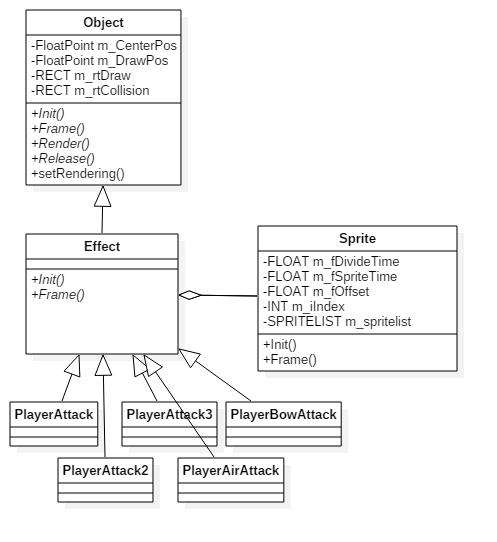
[그림 2-9] : BKOBject의 클래스 다이어그램

BKObject는 게임의 배경을 담당하는 클래스이며, 추가적으로 Scene에서 지형들을 배치하고 그 지형들을 해당 Scene의 BKObject에 추가해서 배경과 지형을 같이 BKObject에서 관리를 한다.



[그림 2-10]: BKOBject의 시퀀스 다이어그램

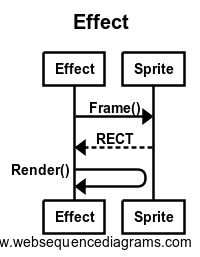
### Effect



[그림 2-11]: Effect의 클래스 다이어그램

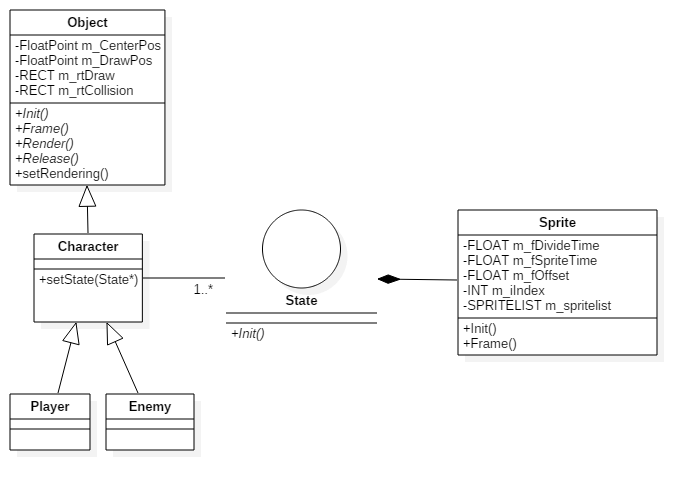
Effect클래스는 현재 플레이어의 근접공격3개와 공중 공격, 활 공격 총 5개가 존재한다.

멤버로 Sprite클래스를 가지고 있는데 Sprite는 STL 벡터에 RECT를 저장하고 Frame호출 시에 오프셋만큼 시간이 경과하면 다음 RECT를 가리키고 Effect에서 RECT를 불러와서 비트맵의 불러오는 위치를 조정하여 애니메이션 효과를 줄 수 있게 하였다.



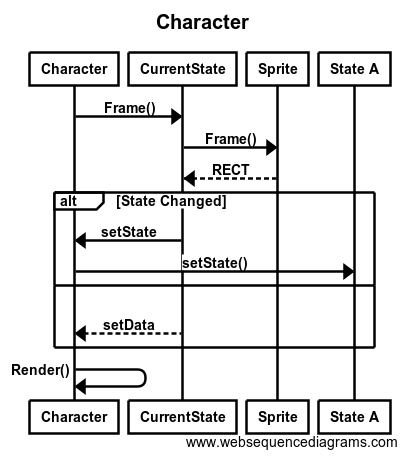
[그림 2-12]: Effect의 시퀀스 다이어그램

### Character



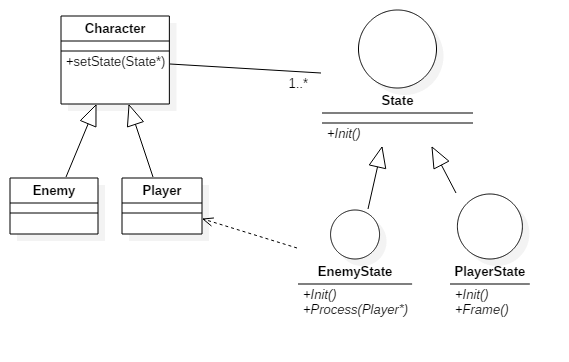
[그림 2-13]: Character의 클래스 다이어그램

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 2-14]: Character의 시퀀스 다이어그램

### State



[그림 2-16]: State의 클래스 다이어그램

State는 EnemyState와 PlayerState로 나뉘어져있다.

#### PlayerState

플레이어의 상태는 키보드 입력키에 의해 변하게 된다.

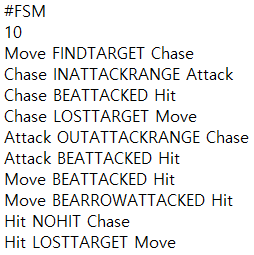
방향키로 이동을 하고 A키로 점프, S키로 근접공격, D키로 활공격을 할 수 있다. 실제 원작 게임에선 플레이어의 상태가 42개가 구현되어 있으나, 현재까지는 22개의 상태가 구현이 되어있다.

#### EnemyState

#### 

[그림 2-17]: EnemyState의 클래스 다이어그램

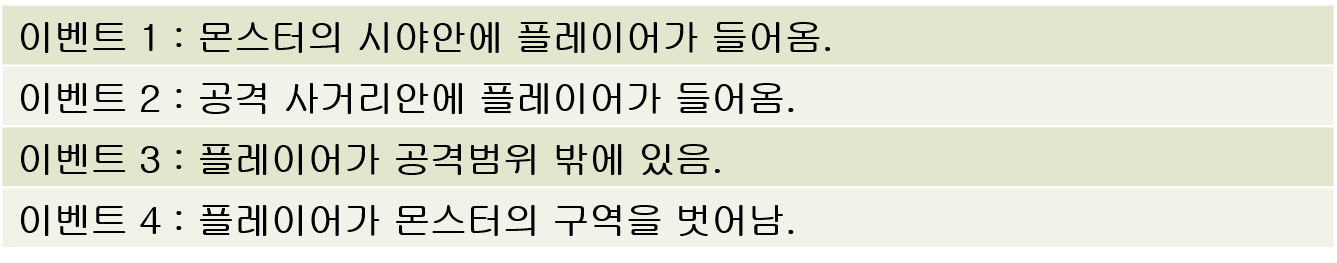
키보드로 상태가 변하는 플레이어와 달리 적 같은 경우는 상태별로 이벤트 발생 시 변경될 상태가 정의되어 있는 FSM이 필요하다.



[그림 2-18]: 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**

# 추가 필요 기능 및 개선점

가장 부족하게 느껴지는 것은 이펙트가 많이 없다는 점이다.

공격을 하고 그에 맞게 충돌처리를 해서 적을 잡기는 하지만 이펙트가 없어서 게임이 굉장히 심심한 느낌이 많이 들어서 이펙트의 소중함을 새삼 느끼는 계기가 됐다. 그 밖에도 아이템을 구현을 못한 점 또한 많이 아쉽다.

우선 아이템을 구현하기 위해 인벤토리나 게임내 메뉴와 같은 UI의 구현이 우선되어야 할 것 같다.

마지막으로 적의 종류 또한 현재는 한 종류만 나와서 적의 종류를 좀더 다양화하고 그에 맞게 FSM또한 추가가 필요해 보인다.

프로그램의 구조적으로 보면, 프로젝트 기한이 다가올수록 클래스를 설계할 때 재사용성이나 유연성을 깊게 생각하지 않고 구현에 급급하게 만들어서 코딩이 난잡하게 된 거 같다. 다음에 만들 때는 프로그램의 설계단계에서 좀 더 많은 고민을 해서 객체 지향 프로그래밍을 해야 할 것 같다.