**MoonLighter 기술문서**

* **개요**
* **GameObject, ObjectManager**
* **GameObject**
* **ObjectManager**
* **Player**
* **Move**
* **Collision**
* **Monster**
* **Monster**
* **BossGolem**
* **UI**
* **Inventory**
* **Progressbar**
* **DamageFont**
* **Etc**
* **Effect**
* **Item**
* **NPC**
* **Camera**
* **Lighting**

**개요**

게임 제목 : MoonLighter

제작 기간 : 2019.03.07 ~ 2019.03.15

팀 원 : 이주희, 정의찬, 이학영, 권순우

제작 환경 : C++

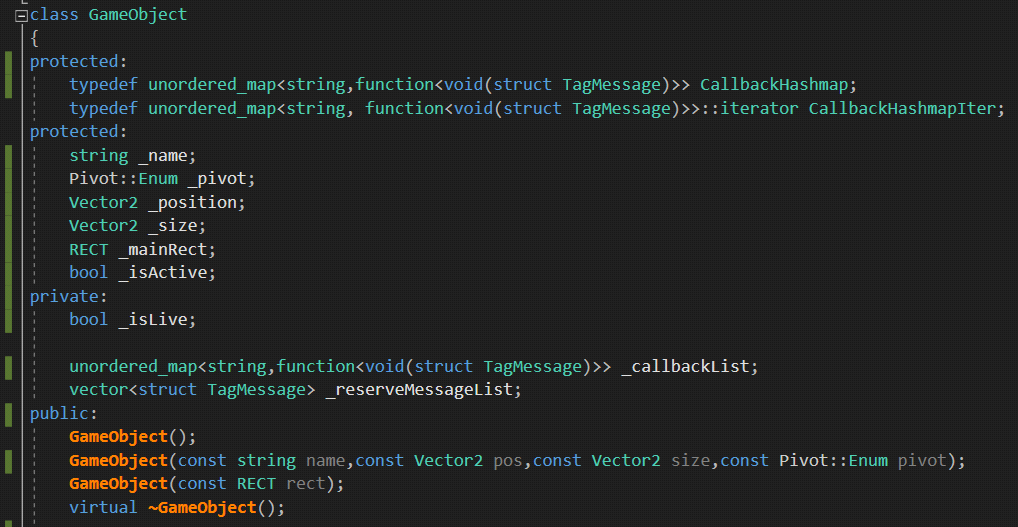
사용 라이브러리 : FMOD, Direct2D, DirectX11

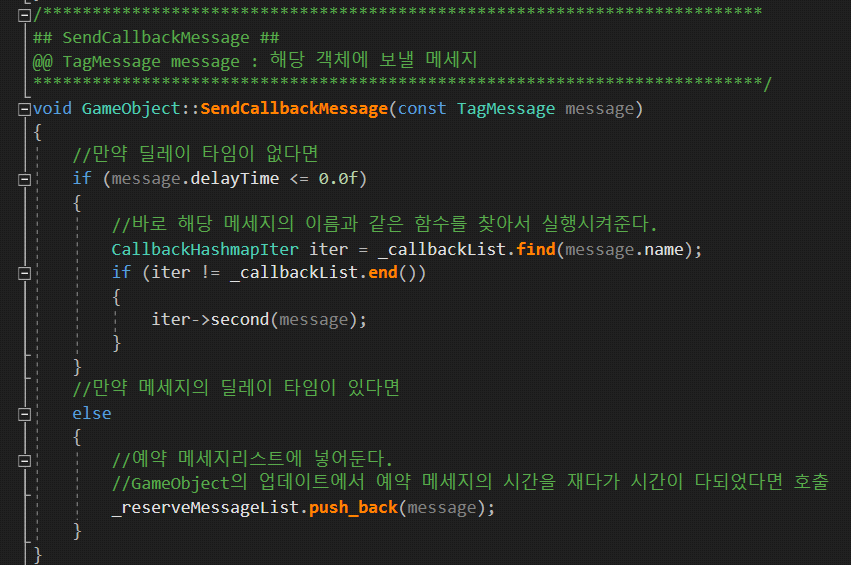
버전 관리 : GitHub, SourceTree

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | 이주희 | 정의찬 | 이학영 | 권순우 |
| Player | Player |  |  |  |  |
| UI | Inventory |  |  |  |  |
| Progressbar |  |  |  |  |
| DamageFont |  |  |  |  |
| Monster | Golem |  |  |  |  |
| Weed |  |  |  |  |
| BigSlime |  |  |  |  |
| Plankton |  |  |  |  |
| GolemBoss |  |  |  |  |
| Scene | Loading |  |  |  |  |
| Dungeon |  |  |  |  |
| Town |  |  |  |  |
| ETC | NPC |  |  |  |  |
| Item |  |  |  |  |
| Effect |  |  |  |  |
| System | Camera |  |  |  |  |
| Lighting |  |  |  |  |
| FrameWork |  |  |  |  |

* **GameObject, ObjectManager**

**# GameObject**

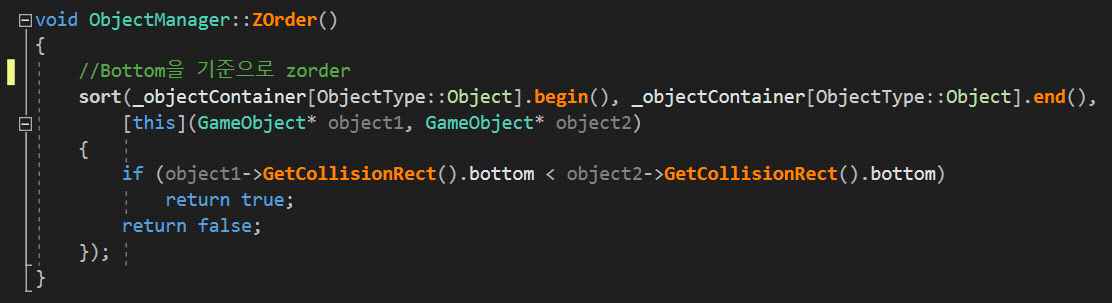




각 객체간의 커플링을 줄이기 위해 메시지 통신을 구현했으며 키값(string)과 함께 메시지를 받았을 때 실행될 함수를 해쉬맵에 등록하게 했습니다.

**# ObjectManager**



GameObject는 벡터에 담아 관리하게 하며 이 벡터들은 ObjectType의 열거자를 키값으로 해쉬맵에 담았습니다.

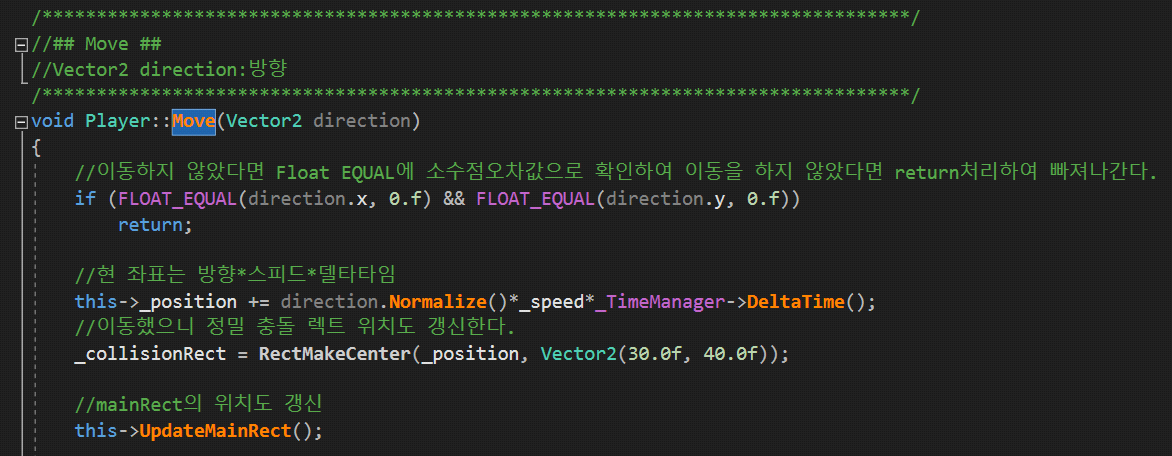
Object타입의 벡터에 있는 오브젝트들은 Y축에 따른 Z오더가 필요하기 때문에 퀵소팅을 사용해서 렌더링전에 정렬해 주었습니다

* **Player(example)**

# 플레이어 이동

키를 눌렀을시 바로 플레이어의 좌표에 스피드를 더하는 식으로 플레이어를 이동처리했더니 대각선으로 이동하게 되면 두번의 이동처리가 되어 더 많은 거리를 이동하는 문제가 있었습니다.

이를 해결하기 키를 누르면 이동방향 변수에 방향값을 더한 뒤 나온 벡터를 정규화하여 이동처리 하게 했습니다.

Ex) moveValue += Vector2(1,0);

# 플레이어 이동

키를 눌렀을시 바로 플레이어의 좌표에 스피드를 더하는 식으로 플레이어를 이동처리했더니 대각선으로 이동하게 되면 두번의 이동처리가 되어 더 많은 거리를 이동하는 문제가 있었습니다.

이를 해결하기 키를 누르면 이동방향 변수에 방향값을 더한 뒤 나온 벡터를 정규화하여 이동처리 하게 했습니다.

#플레이어 애니메이션

플레이어의 상태별로 재생해야 할 애니메이션들을 Map의 상태 enum을 키값으로 담아 관리하였습니다.

#사망

플레이어가 던전에서 전투 중 사망할 경우 모든 키 값을 받지 않고 애니메이션 렌더 후 던전 로비로 씬을 이동합니다.

#공격

공격은 검과 화살 두가지로 구성되어있습니다. 검은 입력시 렌더에 맞춰 충돌할 RECT를 생성하여 몹에게 데미지를 전달하도록 구성하였고, 활은 공격시 화살 객체를 생성하여 오브젝트매니저에 등록했습니다.

Enemy와 오브젝트와의 충돌을 플레이어가 아닌 화살 객체에서 처리하도록 했습니다.

#충돌

이동 시 물체들과의 충돌은 ObjectManager를 통해 Object타입의 벡터를 불러와 충돌 검사를 했습니다. 이때 몬스터, 아이템과 물리적 충돌이 발생해 플레이어가 이동하지 못하는 문제가 생겼었습니다. 이를 해결하기 위해 충돌 처리 전에 dynamic\_cast를 이용해서 해당 GameObject\*가 Item인지 Enemy인지 검사 후 아니라면 충돌처리를 실시하게 하였습니다.

플레이어가 구르기를 시전할 경우 Enemy와 충돌하지 않고 데미지를 입지 않는 상태를 만들기 위해서 구르기 상태에 진입할 경우 bool값을 사용하였습니다.

#NPC

npc마다 이동하는 좌표를 vector에 담아 관리하였고, 앵글값을 구하여 해당 위치로 이동하고 거리 값을 구하여 지정 위치까지 도착할 경우 다음 배열의 이동 좌표 배열을 변경하여 다시 이동하도록 만들었습니다.

**Monster**

여러 종류의 몬스터를 구현하기에 앞서 몬스터들의 부모클래스인 Enemy클래스를 GameObject를 상속받아 구현했습니다. Enemy클래스에는 여러 종류의 몬스터들이 공통적으로 필요한 것들을 구현했습니다.

# Golem

골램은 플레이어를 바라보는 시야각을 계산해서 골렘이 보는 방향과 공격방향을 결정하게 하였습니다. 평상시에는 플레이어를 향하는 방향의 각도를 구해 플레이어를 추격하며 자신의 일정 거리안에 들어오게 될 경우 공격상태로 전환하게 했습니다.

# Weed

골렘과 마찬가지로 플레이어를 향하는 각도를 구해 움직이게 구현했으며 피격시 쫒던 방향의 반대방향으로 밀려나가도록 했으며 카메라 흔들림을 통해 타격감을 향상시켰습니다.

# BigSlime

여러 애니메이션에 대한 정보를 상태값 enum을 키값으로 stl::map에 담아서 관리했습니다. 만약 이 몬스터에 다른 기능이 추가되더라도 기존에 적었던 코드들과 충돌을 최소화시킬 수 있는 코드가 제작이 가능하게되었습니다.

#Boss Golem

보스는 플레이어를 인식하기 전의 기본상태와 인식후 생성상태 및 사망상태와 3가지 공격 패턴을 가지고 있습니다.

보스도 굉장히 많은 상태와 그에따른 정보량이 많으므로 map으로 분류하였고,

첫번째 패턴인 손날리기 스킬은 손을 날린 순간부터 플레이어를 쫒아가며 일정 시간이 지나면 바닥으로 떨어지면서 플레이어를 공격합니다.

두번째 패턴인 돌떨어트리기는 일정 딜레이 후에 여러방향으로 바위 객체를 생성합니다. 바위객체는 생성과 동시에 아래로 떨어지며 떨어진 위치에 플레이어가 있다면 데미지를 줍니다. 바위는 일정 시간이 지나면 천천히 사라지게 했습니다.

세번째 패턴은 여러마리의 플랑크톤을 생성해 플레이어의 움직임을 자유롭게 하지 못하게 했습니다.

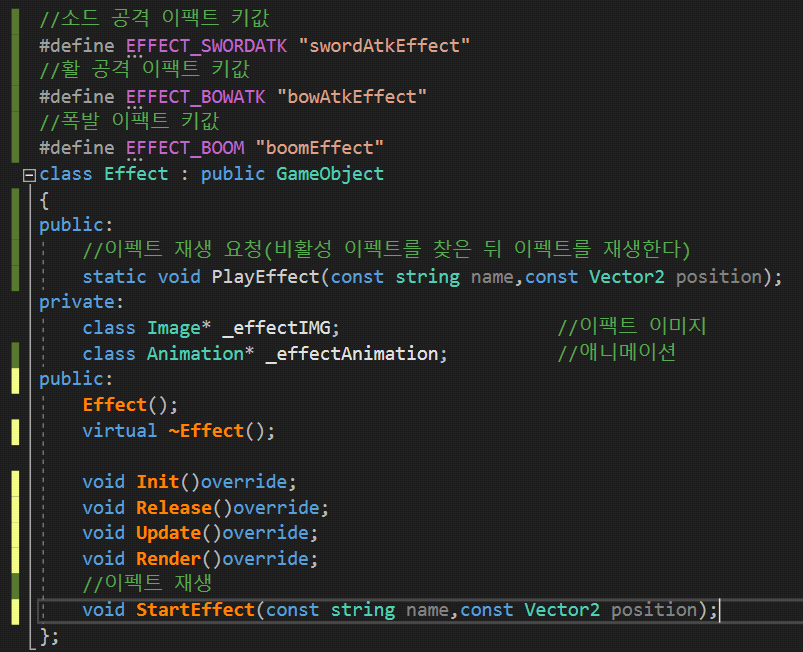
* **UI**

# DamageFont

데미지 폰트는 플레이어와 몬스터 모두가 요청할 수 있어야 하기 때문에 싱글톤 패턴으로 구현했습니다. 데미지 폰트에 필요한 정보들을 구조체 안에 묶고, 요청이 들어오면 구조체에 정보를 담아 초기화한 후에 벡터에 담아 관리했습니다.

# Effect

이펙트는 GmaeObject를 상속받아 구현했으며 미리 지정해놓은 임의의 수만큼 할당해놓은 뒤, 플레이하라는 요청이 들어오면 이미지의 프레임을 재생한 후 다시 비활성 상태로 바뀝니다.

이펙트를 사용하는 클래스들의 편의성을 고려해서 static함수를 선언해 재생하는 이펙트 이미지의 키값과 좌표만 입력하면 사용 가능하게 구현했습니다. 

# Item

아이템은 플레이어가 몬스터 처치 시 생성되는 무브아이템과 플레이어가 화면에 생성된 아이템을 습득해 인벤토리에 생성되는 아이템 2가지가 있습니다.

생성된 무브 아이템은 화면에 이미지를 그려주는 역할을 담당하고 3가지 상태를 처리합니다.

1. 플레이어를 향하는 각도를 계산해 해당 좌표로 움직여 충돌하는 상태

2. 충돌된 아이템이 가방 UI좌표를 향해 이동하는 상태

3. 가방 UI에 충돌되면 화면에서 아이템을 지우고 인벤토리에 추가되는 상태

#Progressbar

프로그래스 바는 최대 체력과 현재 체력의 비율을 구해 구현했습니다.

프로그래스 바 길이 : 프로그래스 바 현재 HP / 최대 HP \* 이미지 길이;

문라이터 프로그래스 바의 한 번에 깎이지 않고 천천히 깎이는 연출을 구현하기 위해서 프로그래스 바의 현재 HP와 플레이어의 현재 HP와 다르다면 값이 같아질 때까지 천천히 줄어들도록 구현했습니다.

# Inventory

[5종류의 인벤토리 슬롯]

캐릭터 슬롯/ 가방 슬롯/ 무기 슬롯/ 방어구 슬롯/ 포션 슬롯 5종류로 구분하였습니다.

문라이터는 인벤토리 안에 여러 기능이 섞여있고 캐릭터가 죽게되면 가방 슬롯 안에든 아이템만 사라지기 때문에 각 기능마다 구분지어 만들었습니다.

[인벤토리 아이템 생성과 개수 추가]

캐릭터가 던전에서 몬스터를 처치해 아이템이 드랍하고 습득하게 되면 ㅇ인벤토리는 캐릭터가 습득한 아이템의 이름을 판단해 아이템을 생성하는데 2가지 경우를 체크합니다.

우선적으로 같은 이름의 아이템이 들어가 있는 슬롯을 탐색하여 해당 슬롯이 아이템이 추가될 수 있는 상태(5개 이하)라면 해당 슬롯에 아이템 카운트롤 추가하고 아니라면 비어있는 슬롯을 찾아서 추가하게 했습니다. 2가지 경우를 모두 만족하지 않는다면 아이템은 추가되지 않습니다.

**Lighting**

Direct2D프레임워크를 사용하면서 라이팅 기능을 추가 하기 위해 DirectX11의 SwapChain으로부터 Direct2DRenderTarget을 생성했습니다.

전체 화면을 Direct2D렌더 타겟을 통해 렌더링 한 후 SRV로 PixelShader에 바인딩 해 라이팅을 처리했습니다.

전역광과 점광은 상수버퍼를 통해 쉐이더에 바인딩했으며 점광의 좌표 및 거리는 화면공간으로 변환 후 바인딩했습니다.

