**MoonLighter 기술문서**

1. **개요**
2. **GameObject, ObjectManager**

* **GameObject**
* **ObjectManager**

1. **Player**

* **Move**
* **Collision**

1. **Monster**

* **Monster**
* **BossGolem**

1. **UI**

* **Inventory**
* **Progressbar**
* **DamageFont**

1. **Etc**

* **Effect**
* **Item**
* **NPC**
* **Camera**
* **Lighting**

**1.개요**

게임 제목 : MoonLighter

제작 기간 : 2019.03.07 ~ 2019.03.15

팀 원 : 이주희, 정의찬, 이학영, 권순우

제작 환경 : C++

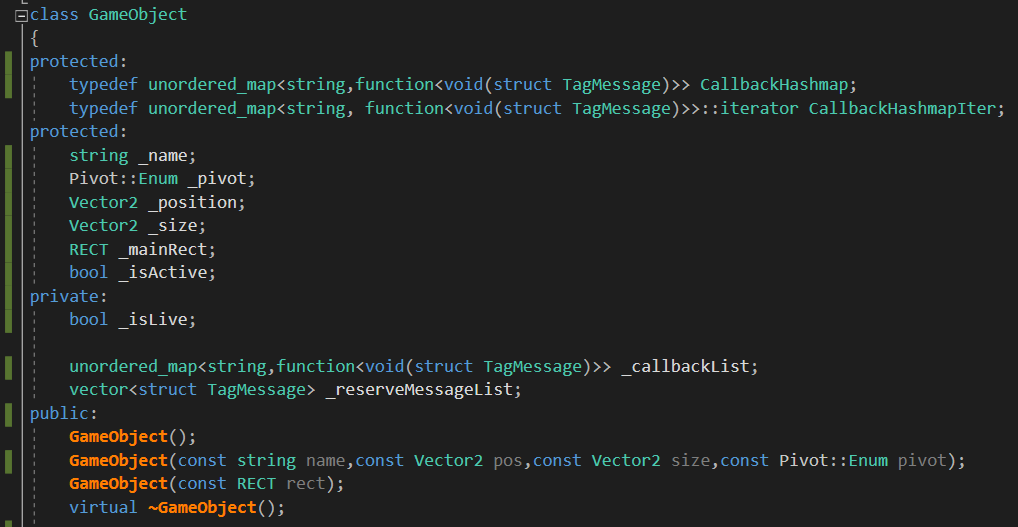
사용 라이브러리 : FMOD, Direct2D, DirectX11

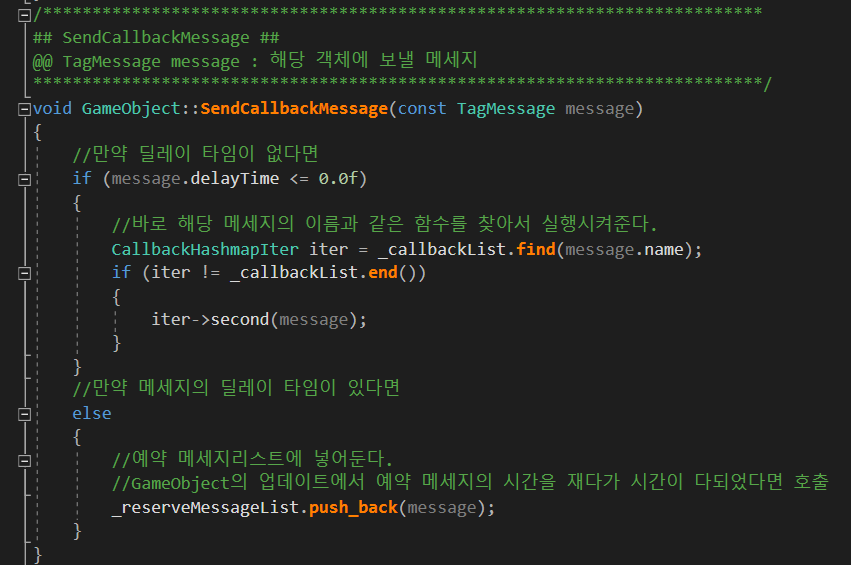
버전 관리 : GitHub, SourceTree

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | 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 |  |  |  |  |

1. **GameObject, ObjectManager**

**# GameObject**

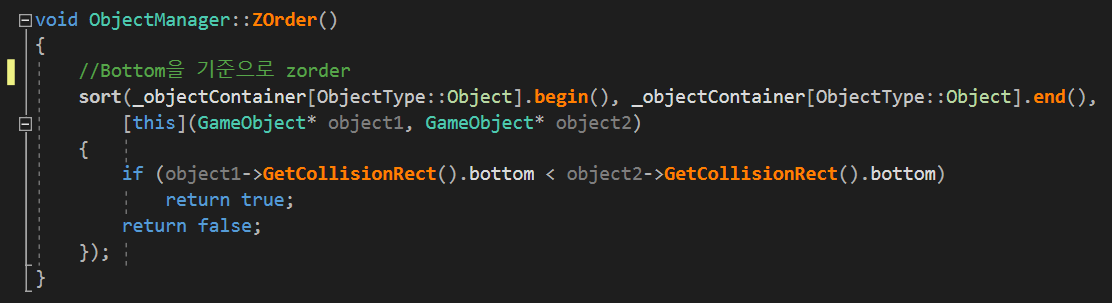




각 객체간의 커플링을 줄이기 위해 메시지 통신을 구현했으며 키값(string)과 함께 메시지를 받았을 때 실행될 함수를 해쉬맵에 등록하게 했습니다.

**# 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

1. **Scene**

씬에서 다음 씬으로 전환할 때 바로 넘어가려 하면 느려지는 문제가 있기때문에 로딩씬을 거쳐 로딩이 끝나면 다음 씬으로 넘어가게 했습니다. 로딩은 쓰레드를 통해 구현했으며 다음 씬의 Init(초기화)부분이 끝나면 넘어가게 했습니다.

1. **Player(example)**

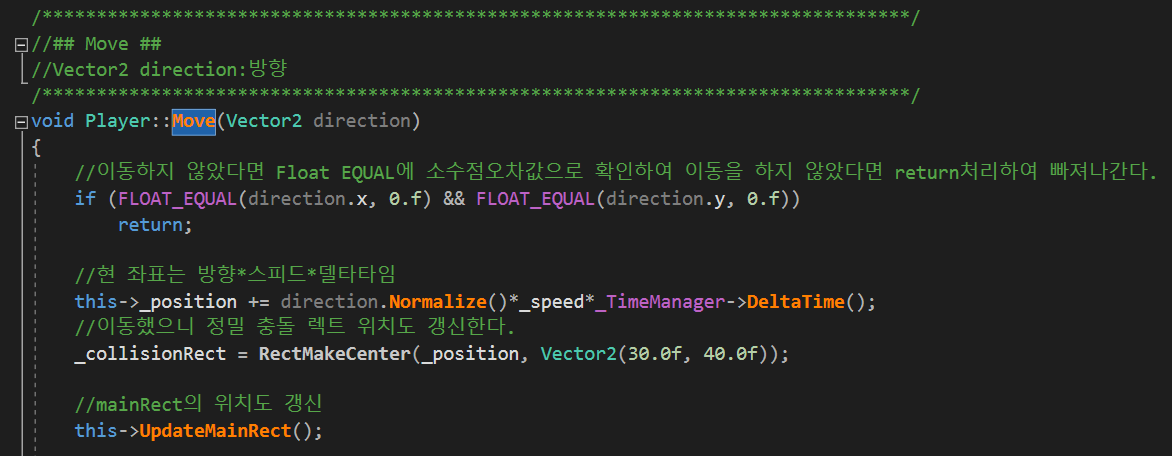
아래처럼 어떠한 문제가 있었는데 어떻게 해결을 했다 하는 식으로 구현을 해결방법과 엮는 것은 좋은 기능의 설명 방법 중 하나다. 단, image\*가 null이었는데 알고보니 초기화를 안해줬었던 문제였다 같은 문제들은 ㄴㄴ. 로직의 해결법이 중요.

눈에 띄는 문제라면 보정 전,보정 후 비교 인게임 샷을 함께 첨부해 보여주는 것도 좋은 방법

# 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);

.

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.

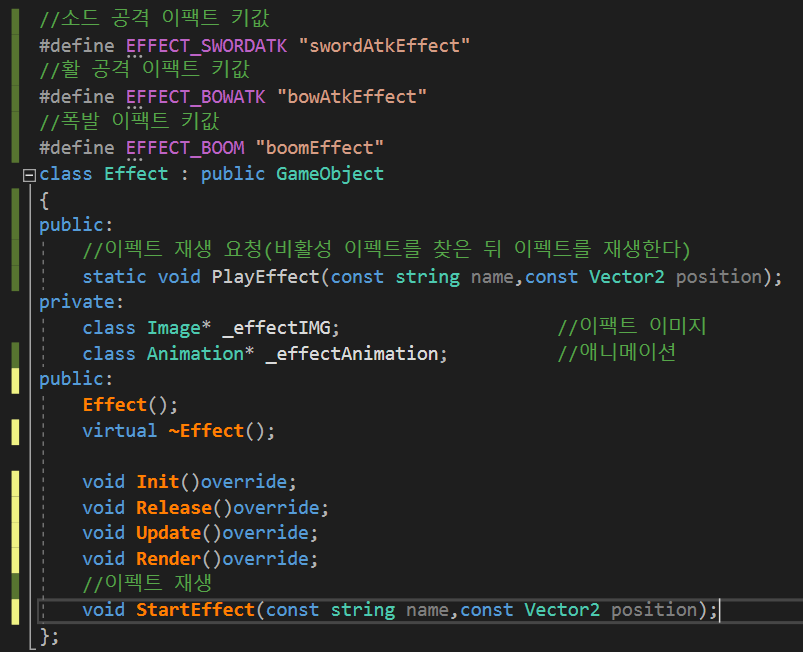
1. **UI**

# 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 MoveItem클래스와 InventoryItem으로 나뉩니다. 무브 아이템은 씬에 이미지를 그려주는 역할을 담당하고 생성되면 3가지 상태를 처리합니다.

# Inventory

인벤토리는