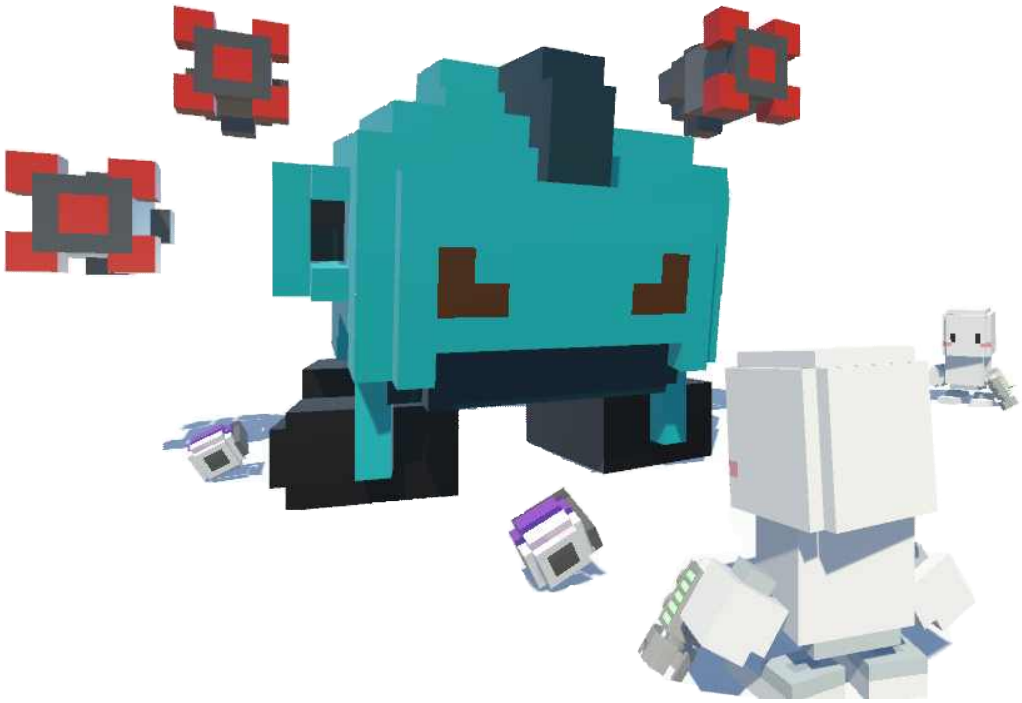
Yeungnam University

MY LITTLE FPS



[Revision history]

Revision date	Version #	Description	Author
	0.1	First document	

= Contents =

1. Introduction

1. Summary

현대 사회를 살아가는 사람들은 취미 생활로 게임을 즐긴다. 최근의 게임들은 화려한 그래픽과 웅장한 세계관을 기반으로 발전해 직관적인 재미를 추구하는 게임이 많이 줄었다. 이에 직관적 재미를 추구하는 사람들을 위해 만든 게임이 바로 “MY LITTLE FPS”이다.

2. introduce “MY LITTLE FPS”

이번에 제작하게 된 게임 “MY LITTLE FPS”는 일인칭 시점의 데스매치 방식으로 진행할 것이다. 해당 게임이 단순하고 직관적인 재미를 추구하지만, 게임 요소가 단순 슈팅이라면 재미를 저하할 수 있다고 판단하여 Enemy object를 추가하여 pve적 요소와 함께 새로운 아이템을 획득할 방법을 제공한다. 즉, (중립/적대) 성향을 띄는 Enemy object를 처치하여 얻은 아이템으로 서로를 처치하는 방식으로 진행되는 데스매치 게임이다.

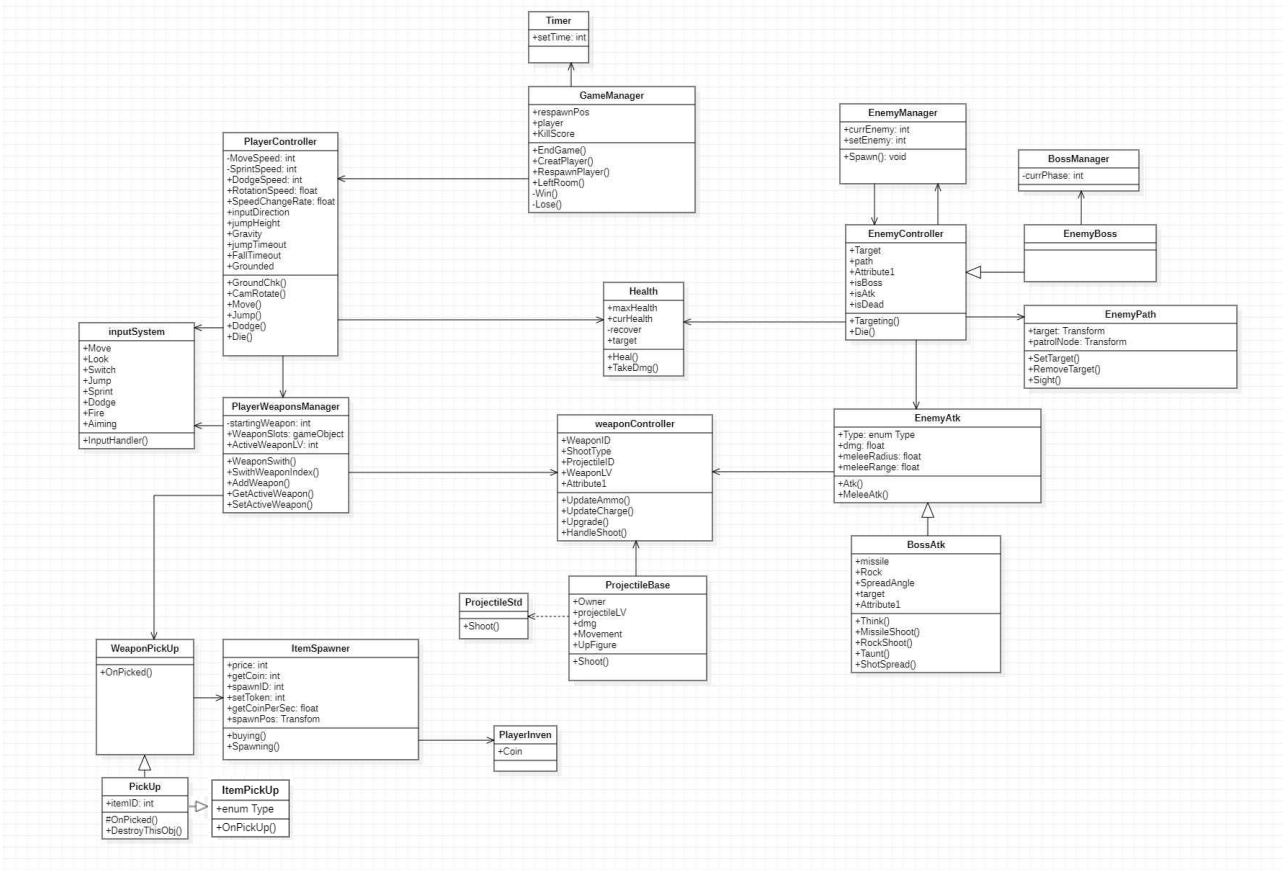
3. Goal

이번 Design 보고서에서는 Use case에 대해 Sequence Diagram을 그리고 동작 과정을 소개한다. 또한 해당 프로젝트에서 사용한 클래스에 대해 소개하며 어떤 기능을 하는지 설명한다. 게임이 완성되었을 때의 StateMachine Diagram을 그려 어떤 상태가 될 수 있는지 보여준다.

해당 보고서를 읽고 나면 “MY LITTLE FPS”가 어떤 방식으로 디자인 되었는지 알 수 있다.

2. Class diagram

해당 프로젝트는 유니티를 사용하여 진행하였다. 해당 프로젝트의 진행에 있어서는 코드를 재사용하는 전통적인 방법인 '상속' 대신 컴포턴트 패턴을 주로 사용하여 만들었다.



해당 클래스 다이어그램은 메인 게임씬의 클래스들은 표현한 다이어그램이다.
실제 구현에는 훨씬 많은양의 클래스가 사용되었지만 이는 UI적 요소를 나타내는데 사용되어 현재 클래스 다이어그램에는 제외하였다.
클래스 다이어그램에 나타난 대부분의 클래스는 Monobehaviour이라는 클래스를 상속한다. 이 클래스는 unity 스크립트가 파생되는 기본 클래스이기에 자동으로 상속된다. 해당 클래스에는 코루틴을 생성하는 함수와 Start, Update, 충돌 및 트리거의 발생을 감지하는 함수 등이 포함되어 있다.

메인 씬의 클래스 다이어그램에서 각각의 게임 오브젝트를 나타내는 클래스끼리 묶어 설명한다.
본 프로젝트의 대다수의 메소드는 public으로 구현되어 있다. 이는 UI적 요소를 위해서인데 UI를 당담하는 클래스에서 메소드의 실행 정보를 얻어 시각적으로 표시하는 과정을 위해서이다.

Player	
PlayerController	InputSystem으로부터 받은 입력을 통해 플레이어를 움직이게 하는 클래스이다. -해당 클래스에서 사용된 변수는 플레이어의 움직이는 방향과 속도, 회전에 관련된 변수들이다. +GroundChk() : 플레이어가 현재 점프 중인지 바닥에 착지한 상태인지 확인하는 메소드이다.
PlayerWeaponManager	플레이어가 소유하고 있는 무기를 관리하는 클래스이다. +WeaponSwitch() : 플레이어의 현재 장비 중인 무기를 교체하는 메소드이다. +SetWeaponIndex() : 리모트 플레이어가 로컬플레이어의 현재 장비중인 무기의 정보를 얻기 위해 사용되는 메소드이다. +AddWeapon() : 플레이어가 무기를 획득하는 메소드이다.
InputSystem	유저로부터 입력을 받아 다른 클래스로 전달하는 역할을 하는 클래스이다. 해당 클래스의 주요 역할은 다양한 입력장치 (모바일 기기의 터치 스크린, 콘솔의 게임패드, pc환경의 키보드와 마우스)를 하나의 입력으로 만들어주는 역할을 하는 클래스이다. (본 프로젝트에서는 사용환경을 pc로 제한하여 키보드와 마우스에 대한 입력처리만 수행한다.)

WeaponSpawner	
ItemSpawner	플레이어로부터 제화를 획득하고 그에 맞는 아이템을 생성하는 클래스이다. +buying() : 플레이어로부터 제화를 획득하는 클래스이다. +spawning() : 재화가 충분히 모였을 때 아이템을 생성하는 클래스이다.
ItemPickUp	플레이어가 아이템을 주웠을 때 발생하는 이벤트를 나타내는 클래스이다. +Type : 해당 아이템이 어떤 아이템인지 나타내는 변수이다. +OnPick() : 아이템을 주웠을 때 플레이어에게 이벤트를 발생시키는 메소드이다. 해당 메소드에서는 플레이어의 Status와 주로 상호작용을 한다.
WeaponPickUp	플레이어가 무기를 주웠을 때 발생하는 이벤트를 나타내는 클래스이다. 해당 클래스에서 itemID는 무기의 종류를 나타낸다.

	+OnPick() : 아이템을 주웠을 때 플레이어에게 이벤트를 발생시키는 메소드이다. 해당 메소드에서는 플레이어의 PlayerWeaponManager와 주로 상호작용을 한다.

WeaponSpawner	
ItemSpawner	플레이어로부터 제화를 획득하고 그에 맞는 아이템을 생성하는 클래스이다. +buying() : 플레이어로부터 제화를 획득하는 클래스이다. +spawning() : 재화가 충분히 모였을 때 아이템을 생성하는 클래스이다.
ItemPickUp	플레이어가 아이템을 주웠을 때 발생하는 이벤트를 나타내는 클래스이다. +Type : 해당 아이템이 어떤 아이템인지 나타내는 변수이다. +OnPick() : 아이템을 주웠을 때 플레이어에게 이벤트를 발생시키는 메소드이다. 해당 메소드에서는 플레이어의 Status와 주로 상호작용을 한다.
WeaponPickUp	플레이어가 무기를 주웠을 때 발생하는 이벤트를 나타내는 클래스이다. 해당 클래스에서 itemID는 무기의 종류를 나타낸다. +OnPick() : 아이템을 주웠을 때 플레이어에게 이벤트를 발생시키는 메소드이다. 해당 메소드에서는 플레이어의 PlayerWeaponManager와 주로 상호작용을 한다.

(Player & Enemy) 공통	
Health	+MaxHealth : 최대 체력 +Recover : 초당 체력 회복량 +Target : 자신을 공격한 게임 오브젝트 +Heal() : Recover에 설정한 수치만큼 회복하는 메소드이다. +TakeDamage() : 대상 게임 오브젝트가 피해를 받았을 때 자신을 공격한 오브젝트가 어떤 오브젝트인지 확인하는 메소드이다.
WeaponController	해당 게임 오브젝트의 투사체 발사를 관리하는 클래스이다. UpdateAmmo() : 투사체를 발사할 때 설정된 탄창으로부터 탄알의 수를 업데이트 하는 메소드이다.

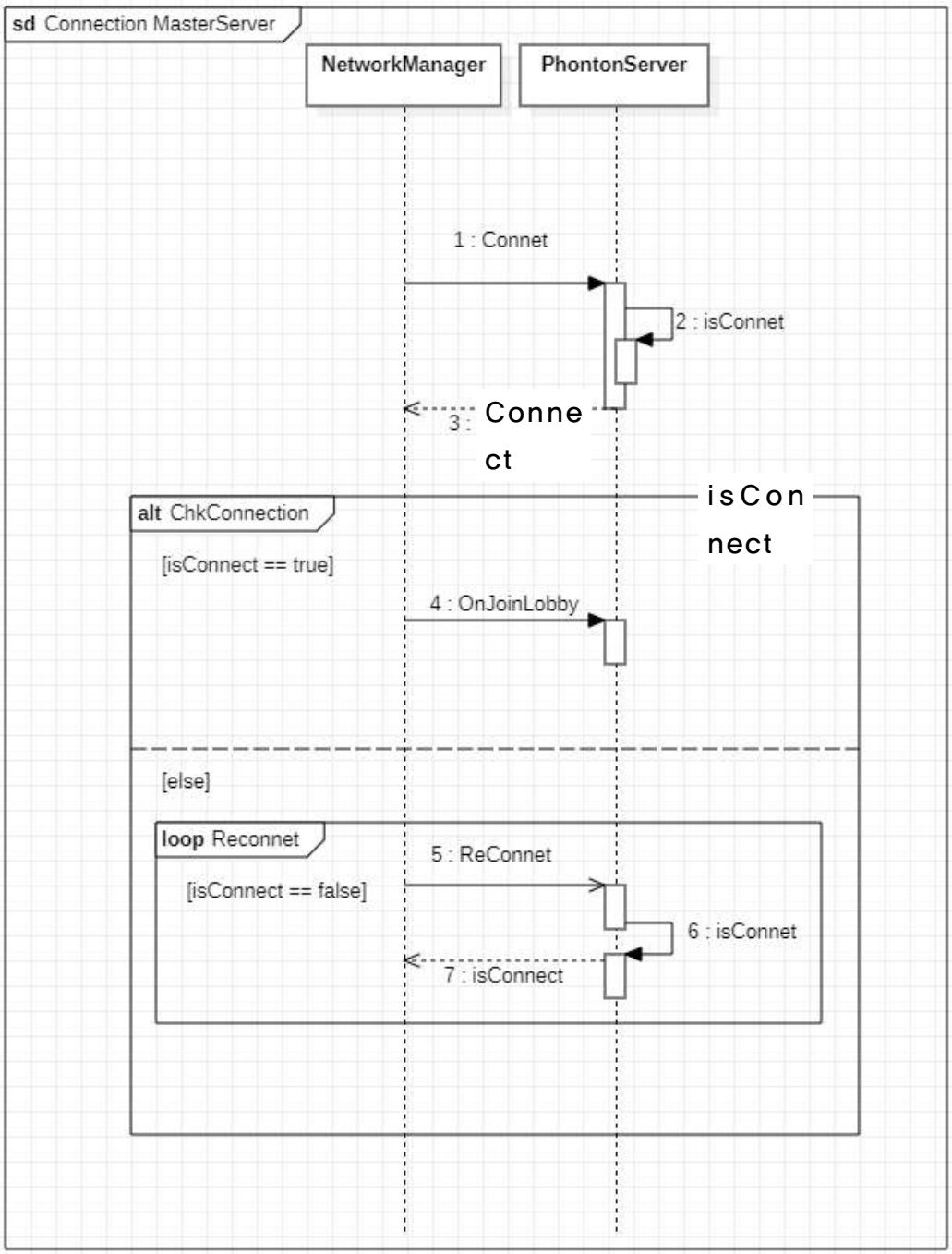
	UpdateCharge() : 투사체의 종류가 충전식무기라면 UI적으로 무기에서 충전되고 있음을 보여주는 메소드이다. Upgrade() : 플레이어의 경우 PlayerWeaponManager에서 무기 레벨이 증가하였고, 해당 무기에 강화된 투사체가 있는 경우 투사체의 종류를 바꾸는 메소드이다. HandleShoot() : 해당 무기에 설정된 투사체를 발사하는 메소드이다.
ProjectileBase	탄알의 기본적인 정보를 담고 있는 클래스이다. Owner : 해당 오브젝트를 발사한 주인이 누구인지 정보를 저장하고 있다. Onshoot() : 발사된 시점에서 해당 투사체의 방향을 설정하는 메소드이다.

Enemy	
EnemyPath	Enemy의 이동경로를 관리하는 클래스이다. 기본적으로 설정된 순찰 경로를 따라 이동하지만 Target이 설정되면 해당 목표를 쫓는다. Sight() : Enemy의 시야에 적이 감지되면 현재 타겟을 적으로 설정하는 메소드이다.
EnemyController	Enemy의 행동을 관리하는 클래스이다. Targeting() : EnemyPath나 Enemy의 Health클래스에서 Target이 설정된 경우 해당 Enemy의 Target을 설정한다.
EnemyAtk	EnemyController에서 Target을 감지한 경우, 상대가 공격 범위 내에 들어왔을 때 공격을 시행한다.
EnemyManager	현재 게임에 생성된 Enemy의 수를 계산하고 Enemy가 처치되었다면 설정된 수치만큼 다시 Enemy를 생성하는 클래스이다. EnemySpawn() : Enemy를 생성하는 메소드이다.

BOSS Enemy	
EnemyPath (X)	기본적으로 BOSS Enemy는 Enemy를 구성하는 클래스를 상속받아 구현되었다. 하지만 BOSS의 경우 따로 이동하지 않음으로 EnemyPath를 상속받지 않는다.
EnemyBoss	Boss의 행동을 관리하는 클래스이다. Targeting() : Enemy의 Health클래스에서 Target이 설정된 경우 해당 Enemy의 Target을 설정한다.
BossAtk	Boss는 일반적인 Enemy보다 다양한 패턴의 공격을 수행한다. 기본적으로 근접 공격에 해당하는 메소드는 Enemy의 근접공격을 상속받아 사용하고 다른 패턴의 공격은 BossAtk에서 새롭게 정의한다.
BossManager	현재 게임에서 보스는 2개의 Phase로 구성되어있다. 첫 번째 보스가 처리되었을 경우 2번째 Phase를 활성화하는 역할을 하는 클래스이다.

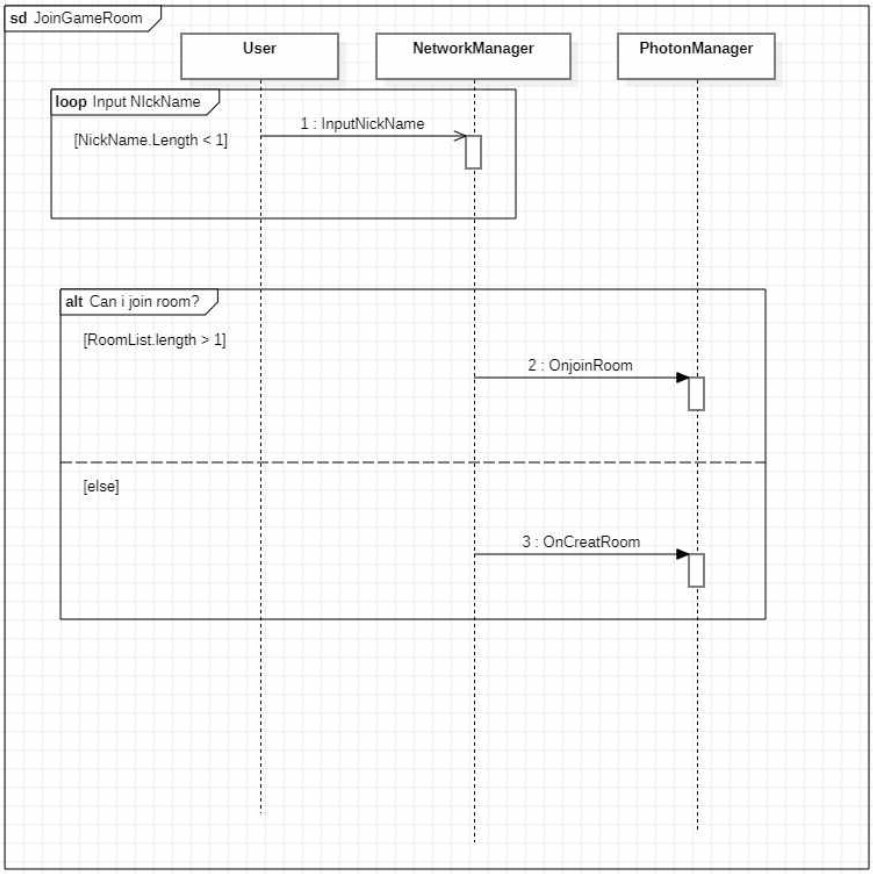
3. Sequence diagram

1) Connection Master Server

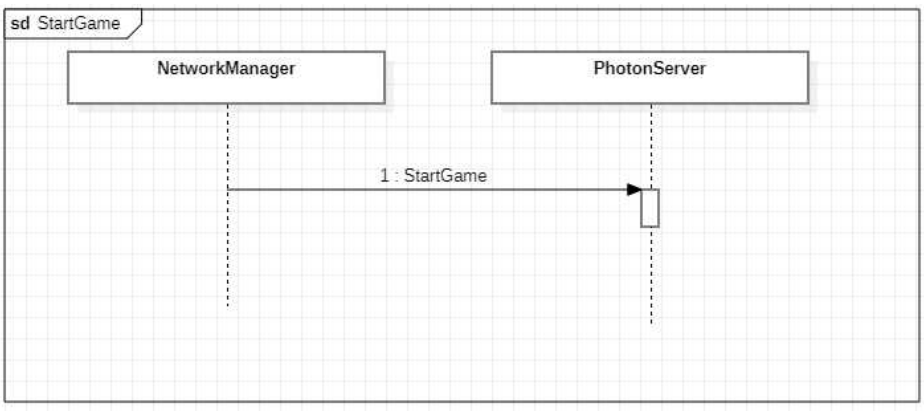


Connection MasterServer Use case 에서의 Sequence Diagram이다. 플레이어가 NetworkManager에서 Connect를 호출하면 포톤서버에서 서버와 연결되었는지 확인하고 연결유무를 NetworkManager에게 알려준다. 만약 NetworkManager에서 포톤서버와 연결이 확인되었으면 해당 플레이어를 로비로 호출하고 연결에 실패했으면 재접속을 요청한다.

2,3) Join GameRoom & Start Game

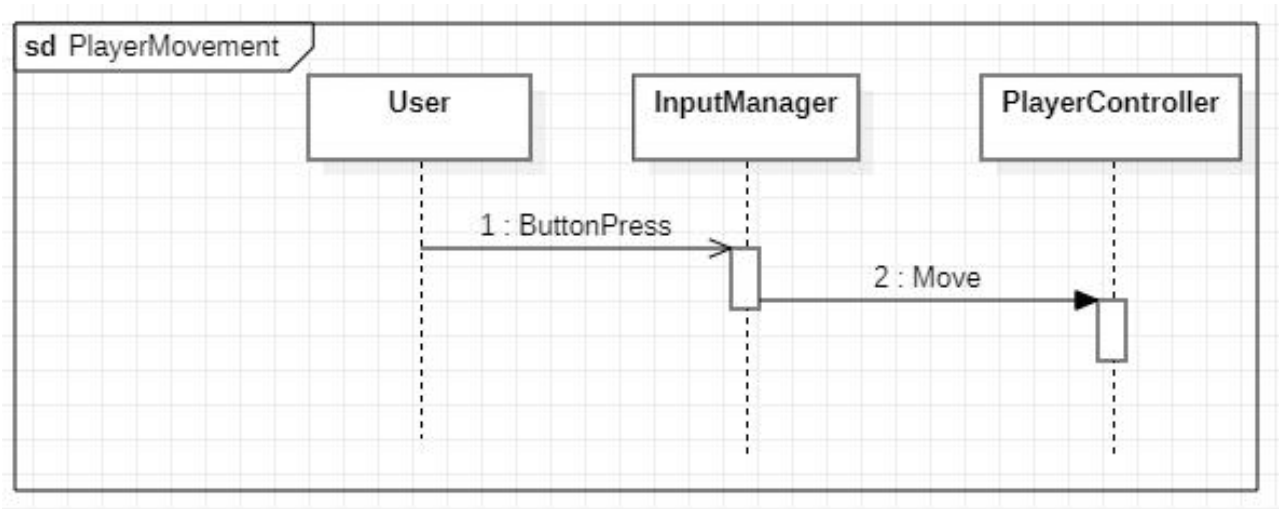


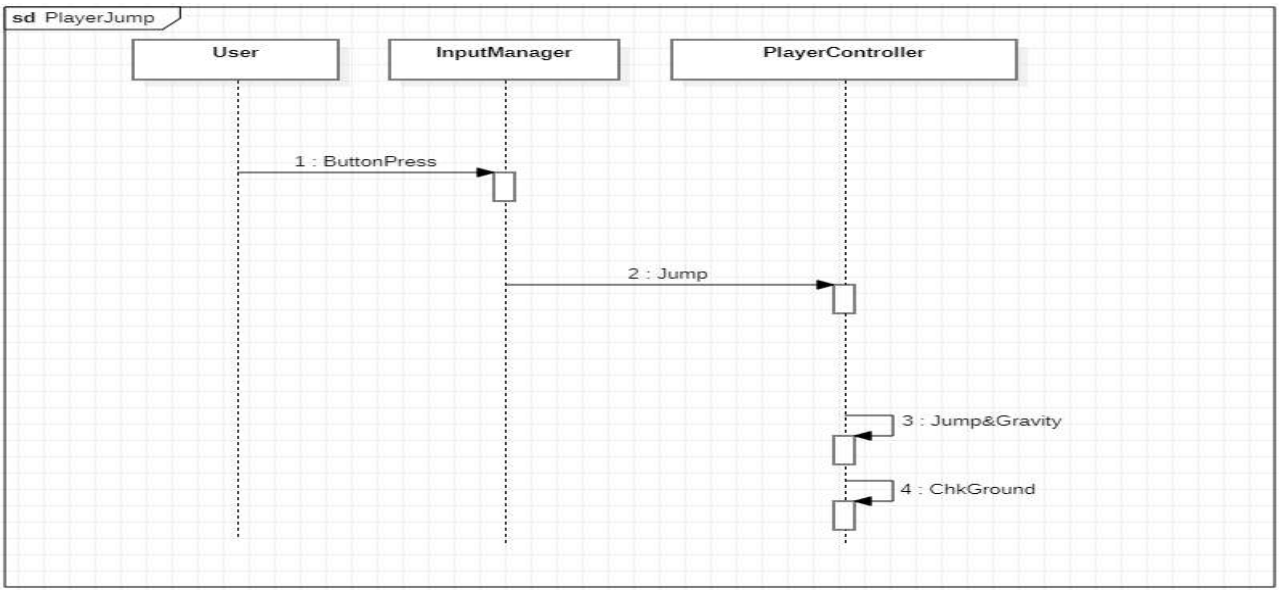
로비에 접속한 플레이어에게 해당 서버에서 사용할 닉네임을 설정하는 Use case에 대한 Sequence Diagram이다. User가 입력한 닉네임의 글자 수가 0자인 경우 입력을 하지 않고 접속을 시도한 것임으로 다시 닉네임을 다시 입력하게 한다.
이후 룸에 접속을 시도했을 때 현재 만들어진 방이 존재하면 룸에 참여하고, 룸이 존재하지 않는 상황이면 자신이 룸을 생성한다.

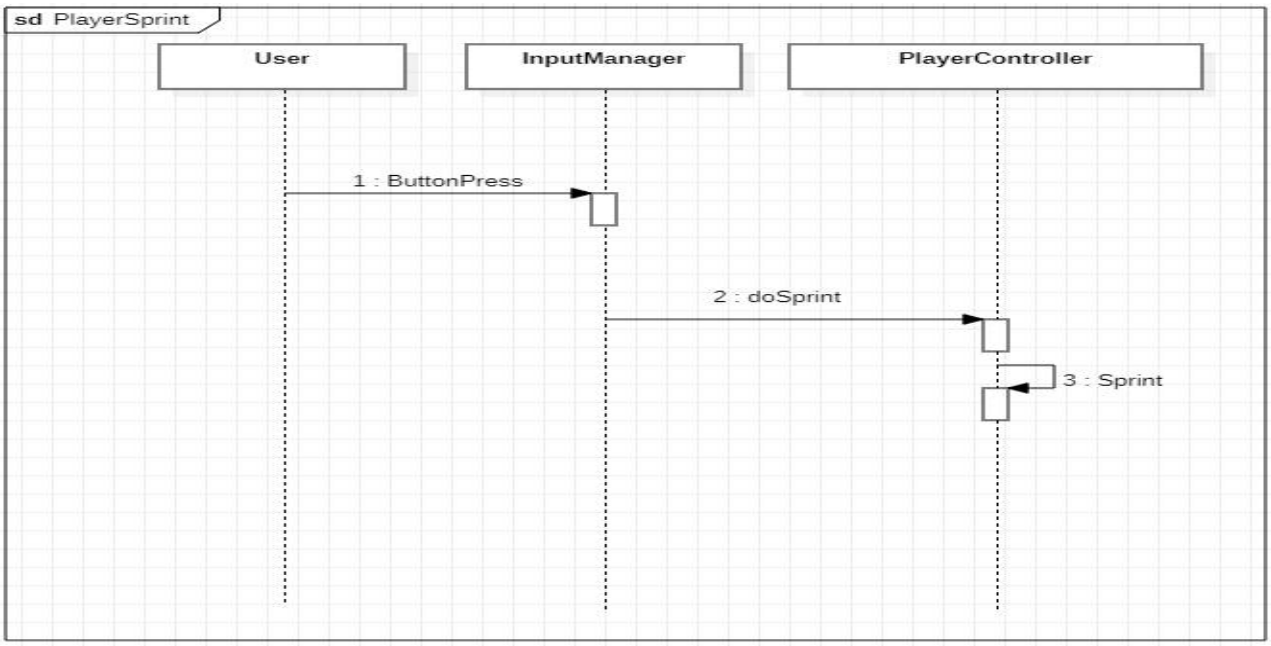


룸에 접속한 플레이어 중 마스터 클라이언트(방장)이 게임을 시작하는 use case에 대한 Sequence Diagram이다.

4,5,6) PlayerMovement

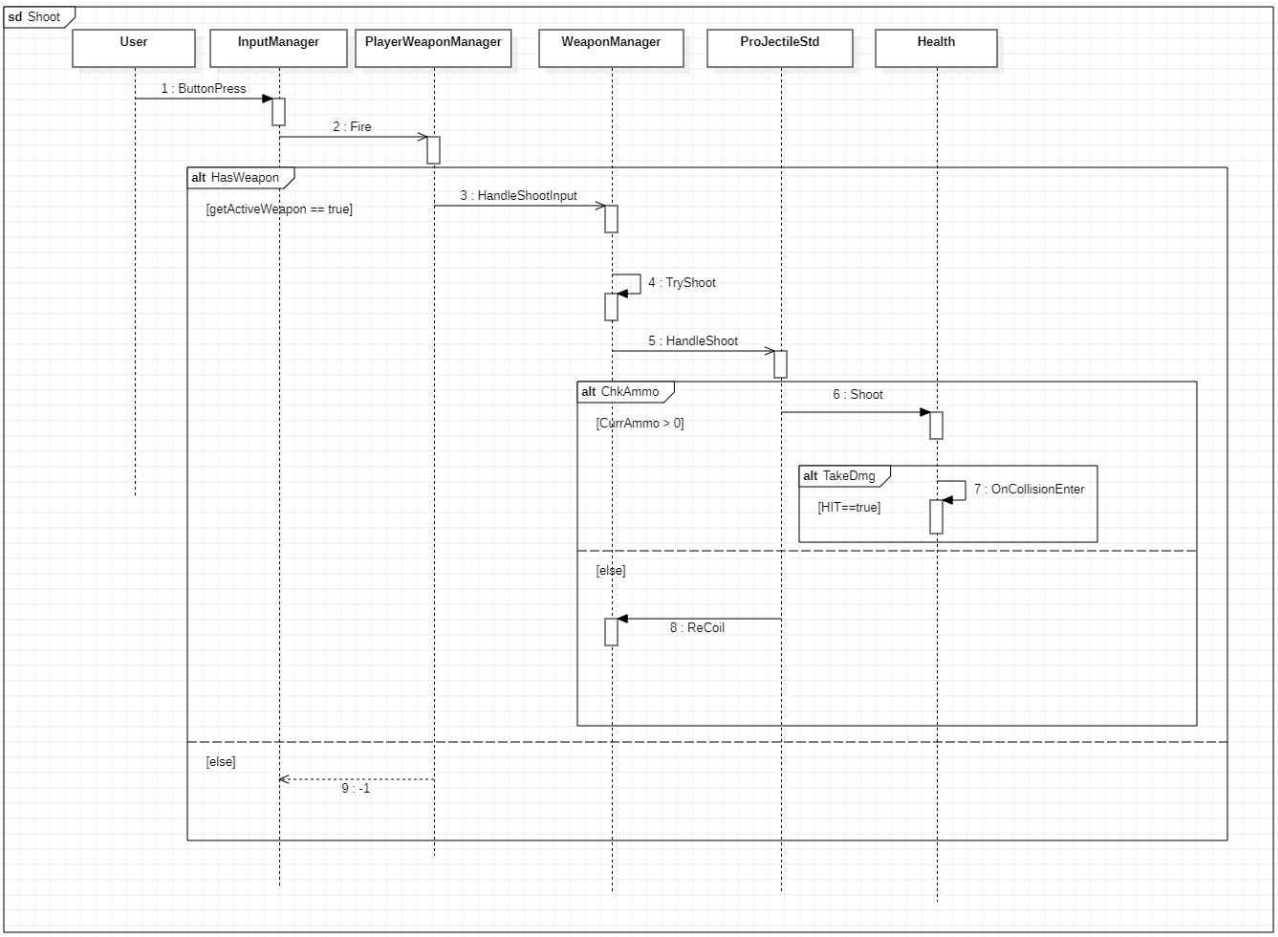






PlayerMovent에 관련된 Use case에서의 Sequence Diagram이다. 일반적인 순서는 User가 버튼을 눌렀을 때 InputManager가 이를 처리해서 어떤 행동을 하기 위해 버튼을 눌렀는지 판단한다. 이후 InputManager에서 처리된 행동을 PlayerController에서 수행한다. Jump의 경우는 Move계열과는 다르게 점프 이후에 플레이어가 땅에 착지했는지 여부를 확인한다. 이는 ChkGround함수를 통해 수행한다.

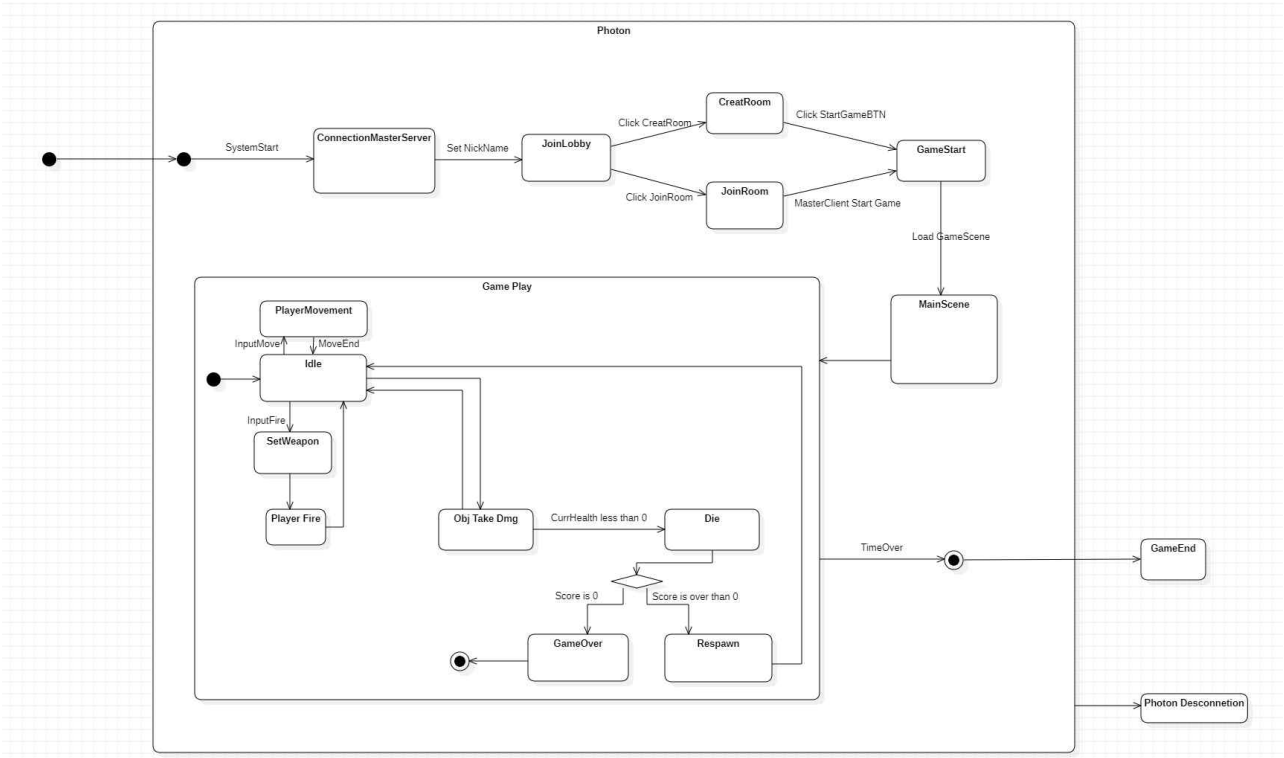
7) Shoot



플레이어가 무기를 발사할 때 Use case에 대한 Sequence Diagram이다. 플레이어가 무기를 발사하기 위해서 버튼을 누르면 inputManager에서 이를 감지하고 PlayerWeaponManager로 발사하라는 신호를 보낸다. PlayerWeaponManager에서는 현재 장비 중인 무기를 확인하고 장비 중이라면 WeaponManager에 발사를 요청한다. WeaponManager에서 현재 무기가 발사 가능한 상태인지 확인하고 총알이 있으면 투사체를 발사한다. 그렇지 않으면 발사하지 않고 재장전을 한다.

투사체를 발사한 경우 WeaponManager에서는 WeaponManger에 등록된 투사체에 발사를 요청한다. 투사체 클래스에서는 해당 무기가 바라보고 있는 방향으로 투사체를 발사한다. 발사한 투사체가 Health클래스를 가지고 있는 게임 오브젝트와 충돌했을 경우 해당 클래스에서 TakeDmg메소드를 실행한다.

4. State machine diagram



유저가 게임을 시작하면(처음 시작화면에서 PVP를 선택한 경우) 서버와 접속하기 위한 화면이 나온다. 접속을 시도하면 닉네임을 설정하고 로비에 참가한다. 로비화면에서는 자신이 방을 만들지 이미 만들어진 방에 참여할지 선택하는 화면이 나온다. 해당 화면에서 선택할 수 있는 방이 없다면 자신은 방의 이름을 설정하고 방을 만든다. (혹은 빠른 참여를 선택하면 방이 없는 경우 방-번호 이름으로 방이 생성된다. 방이 존재하는 경우 최상단에 존재하는 게임룸에 참여한다) 이후 방에 함께 게임할 유저가 모두 방에 접속했다면 마스터 클라이언트(처음 해당 게임룸을 만든 유저)가 게임 스타트 버튼을 누르면 게임이 시작된다. 게임이 시작되면 해당 룸에있는 모든 유저는 메인 씬으로 이동하게 된다.

메인 씬으로 이동한 유저들은 각각 자신의 플레이어가 생성되고 움직임에 해당되는 버튼을 누름으로 플레이어를 이동할 수 있다. 플레이어나 적이 발사한 투사체에 맞게 되면 체력이 감소하고 체력이 0이되면 점수를 1점 감점 시키고 플레이어를 부활시킨다. 점수가 0점에 다다르면 플레이어는 부활하지 못한다.

제한시간이 다 되었을 때 점수가 가장 높은 유저가 승리하고 게임은 끝이 난다.

5. Implementation requirements

5.1 H/W platform requirements

- cpu : intel i3+
- RAM : 4G+
- HDD / SSD :10G+

5.2 S/W platform requirements

– OS : Windows7+

– Implement Language : C# (Unity)

6. Glossary

Unity : 게임 개발을 위한 통합개발환경

GameObject : 게임 세계에서 표현되는 모든 객체를 말함
게임의 구성요소로 사용되며, 스크립트와 상호작용하여 동작하는 객체

Sequence Diagram : 객체 간의 동적 상호작용을 시간적 개념으로 모델링하여 나타낸 다이어그램

StateMachine Diagram : 객체 LifeTime동안의 변화될 될 수 있는 모든 상태를 정의해 둔 다이어그램

7. References

–강의자료 : Structural Modeling II, Behavior Modeling I, II

–참고자료 : https://docs.unity3d.com/
https://www.photonengine.com/