증강현실을 활용한 협동 디펜스 게임

[Team pARty]
2017103998 송재혁
2018102226 장예원
2019102225 전세계

요 약

나날이 발전하는 증강현실(AR) 관련 기술과 이를 탑재한 스마트폰의 보급에도 불구하고 AR 게임의 성공 사례는 흔치 않다. 이는 증강현실에 대한 충분한 이해를 거치지 않아 증강현실만이 제공할 수 있는 경험을 사용자가 느낄 수 없었기 때문이다.

본 프로젝트에서는 현실과 가상세계와의 상호작용이라는 AR 의 장점을 살려 기존 AR 게임과 차별화한 게임을 개발하였다. Google ARCore 와 Unity 엔진 등의 기술을 기반으로 구현하고 멀티 플레이를 지원하며, 플레이어 간의 협동을 이용한 플레이로 게임의 몰입감을 높였다.

1. 서론

1.1. 연구배경

최근 출시되는 대부분의 스마트폰이 증강현실(Augmented Reality) 관련 기술을 지원함에 따라 증강현실 어플리케이션의 접근성이 향상되는 추세다. 대표적으로 삼성은 2016 년 3 월에 출시한 Galaxy S7 부터 ARCore 를 지원하고 있고, 2017 년 4 월에 출시한 Galaxy S8 부터 깊이(Depth) API 를 지원하고 있다. 애플은 이에 더해 2020 년 10 월에 출시한 iPhone12 시리즈부터 라이다(LiDAR) 센서를 탑재하여 사실적인 AR 경험을 제공하고 있다.

그러나 증강현실을 지원하는 스마트폰의 보급에도 불구하고 이를 적절히 활용한 어플리케이션은 많지 않다. 가장 흥행에 성공한 증강현실 게임인 '포켓몬 고(Pokémon GO)'의 경우에도 실제 환경을 배경으로 할 뿐이지 현실과 가상 사이의 상호작용이라는 증강현실의 장점을 살리지 못했다.

본 프로젝트에서는 스마트폰을 이용한 AR 멀티플레이어 협동 디펜스 게임을 개발한다. 실제 환경을 고려하여 적이 생성 및 이동하고, 플레이어도 마찬가지로 실제 지형지물을 고려하여 적을

처치한다. 이처럼 실제 지형지물을 이용한 플레이를 통해 현실 세계와 가상 환경 사이의 상호작용을 통해 증강현실의 활용도를 높인다. 더하여 플레이어의 행동반경을 제한하거나 예상치 못한 곳에서 몬스터가 스폰하여 두 플레이어가 협력해야만 승리할 수 있도록 협동 플레이를 유도하여 게임의 몰입감과 재미를 더한다.

1.2. 연구목표

기존 GPS 기반의 증강현실 게임에서 벗어나 증강현실 기술의 장점을 극대화하는 게임을 개발하는 것을 목표로 한다. 단순히 현실을 배경으로 삼고 가상의 물체를 띄우는 정도로만 증강현실을 활용하는 것을 지양한다. 이 목표를 달성하기 위해 필요한 세 가지의 세부 목표를 아래에서 설명한다.

첫번째, 현실과 가상 사이의 상호작용을 활용한다. 현실 세계의 지형 지물이 가상 세계에 작용하고, 가상 물체가 실제 환경과 충돌하는 등의 상호작용을 통해 실감나는 플레이 경험을 제공함과 동시에 증강현실의 장점을 살린다.

두 번째, 멀티 플레이를 지원하여 컨텐츠에 대한 몰입감을 높인다. 싱글 플레이가 아닌 두 플레이어가 협동하여 승리할 수 있도록 게임 컨텐츠를 제작하여 플레이어들이 보다 게임에 몰입할 수 있게 한다.

세 번째, 플레이어들의 전략적인 플레이를 유도한다. 단순히 다가오는 몬스터를 처치하며 넥서스를 지키는 디펜스 게임이 아닌, 적절한 타이밍에 스킬을 사용하도록 전략을 구상해야 하는 게임을 구현하여 게임의 완성도를 높일 뿐만 아니라 재미와 승부욕을 증대시킨다.

2. 관련연구

2.1. 증강현실 소프트웨어 개발 도구

2.1.1. ARCore

ARCore 는 Google 에서 개발한 증강현실 소프트웨어 개발 도구이다. Android, iOS, Unity, Unreal, Web 등의 많은 플랫폼에서의 개발을 지원한다. Hit-test, Placement, Depth, Lighting Estimation 등의 증강현실 어플리케이션을 개발하기 위한 기능들을 제공한다.

본 프로젝트에서는 Android 디바이스를 타겟으로 하기 때문에 ARCore 를 사용하여 개발한다.

2.1.2. ARKit

ARKit 은 Apple 에서 개발한 증강현실 소프트웨어 개발 도구이다. iOS 기반의 디바이스만 지원한다. ARCore 에서 제공하는 기능들을 대부분 지원하며 전후면 카메라 동시 지원, Motion Capture 등의 기능을 추가적으로 제공한다.

ARKit 은 iOS 디바이스만 지원하기 때문에 Android 디바이스를 타겟으로 하는 본 프로젝트에서는 사용할 수 없다.

2.2. 게임 개발 도구

2.2.1. Unity

Unity 는 게임 개발 환경을 제공하는 게임 엔진이다. Android, iOS, Windows, Mac, Linux, Web 등의 거의 모든 플랫폼을 지원하는 것이 특징이다. Unity 는 Component 기반의 모듈형 개발 방식을 채택하고 있기 때문에 다른 게임 엔진에 비하여 높은 생산성과 쉬운 개발 난이도를 갖는다.

AR Foundation 이라는 패키지를 통해 ARCore 와 ARKit 을 Extension 으로 지원한다. ARCore Extension 으로 세션을 구성하여 모든 AR 프로세스를 관리한다.

짧은 기간 내에 개발을 마쳐야 하는 본 프로젝트의 일정을 고려하여 생산성이 높은 Unity 와 AR Foundation 을 사용해 개발한다.

2.2.1. Unreal Engine

Unreal Engine 은 게임 개발 환경을 제공하는 게임 엔진이다. Unreal Engine 은 Lumen, Nanite 등의 최신 3D 기술을 탑재한 UE5 를 Early access 로 출시하면서 주목을 받고 있다. 모든 게임 엔진 중에서 가장 뛰어난 기술력과 성능이 특징이다.

C++를 통해 개발하므로 비교적 생산성이 떨어지며 다른 엔진과 비교했을 때 엔진 자체가 많은 리소스를 요구하는 등의 단점이 있다.

AR Core 를 플러그인(Plugin) 형태로 제공한다.

2.3. 깊이 인식

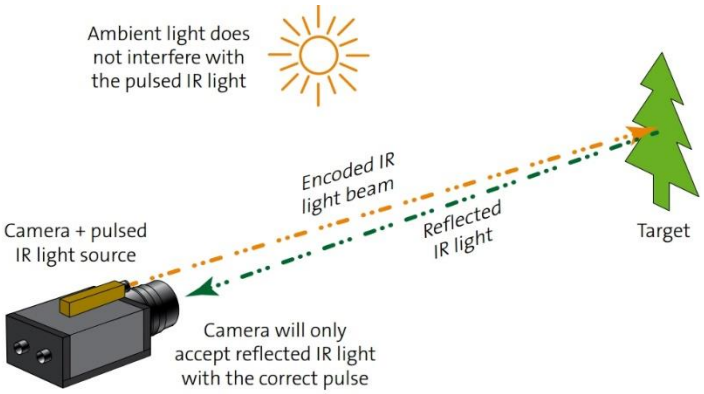
증강현실에서 현실의 장면을 3 차원 정보로 인식하고 정확한 Anchor 를 잡기 위해서는 깊이(Depth) 정보가 필요하다.

2.3.1. Depth-from-motion

ARCore 에서 제공하는 Depth API 는 Depth-from-motion 알고리즘을 사용한다. 사용자가 움직이면서 동일한 장면이 여러 각도에서 촬영될 때 특징점의 위치 변화를 감지해서 거리를 계산한다. 필요한 경우 머신러닝을 통해서 작은 움직임으로도 정확한 깊이를 계산할 수 있다.

따로 깊이 카메라(Depth Camera)가 없더라도 적용할 수 있는 방식이라는 이점이 있다. 반면 움직임이 적거나 특징점이 없는 환경에서 정확도가 떨어질 수 있다. ARCore 에서는 ToF(Time of Flight)센서 등의 하드웨어가 있는 경우 해당 정보도 활용해서 정확도를 높인다.

2.3.2. Time of Flight



[그림 1] Time of Flight 작동 원리

Time of Flight 는 물체와의 거리를 직접 구하는 방식으로 깊이 정보를 인식한다. 대상에게 빛을 쏘아 반사되어 돌아오는 시간을 측정하여 실제 거리를 계산한다.

애플에서는 ToF 원리를 사용하는 라이다(LiDAR)를 아이폰 12 프로에 처음으로 탑재했다. 삼성에서는 DepthVision Camera 라는 이름으로 s20+, s20u 에 ToF 모듈을 탑재했다.

Time of Flight 는 Depth-from-motion 알고리즘에 비해 더 높은 정확도를 가졌지만 탑재된 기기가 제한적이라는 단점이 있어 많은 사용자를 대상으로 하는 어플리케이션에서는 활용하기 어렵다.

2.4. 기존 유사한 연구

본 프로젝트에서 구현할 AR 협동 디펜스 게임과 유사한 모바일 어플리케이션을 조사해본 결과, 실제 지형에서 평면을 인식한 후, 평면상에 한해서 게임이 펼쳐지는 타워 디펜스 게임이나 1 인칭 시점에서 다가오는 적을 처치하는 슈팅 게임이 주를 이뤘다. 이러한 게임들은 실제 지형지물과 가상 세계와의 상호작용이 없어 증강현실의 필요성을 느낄 수 없었다.

2.5. 기존 연구의 문제점 및 해결 방안

2.5.1. 연구의 문제점

기존의 AR 을 활용한 모바일 디펜스 게임의 경우 현실 세계가 게임의 배경이 될 뿐, 실제 지형지물이나 환경이 게임에 거의 작용하지 않는다. 증강현실 기술이 게임 요소에 잘 녹아들기 위해서는 현실 세계가 가상 오브젝트에 영향을 주는 등 현실과 가상 세계 사이의 상호작용이 필연적이다. 또한 기존 게임들이 의도했지만 다소 부족했던 현실 세계를 배경으로 플레이하고 있다는 몰입감을 증대하기 위한 방안을 마련해야 한다.

2.5.2. 해결 방안

2.5.2.1. 실제 지형지물에 따른 몬스터 생성 위치

본 프로젝트의 게임은 멀리서부터 넥서스를 향해 다가오는 몬스터가 존재한다. 이때 기존 AR 게임과 같이 단순히 카메라로부터 멀리 생성하고 사용자에게 다가오게 하는 것이 아니라, 실제 환경의 depth map 으로 얻어진 정보를 활용하여 몬스터가 자연스럽게 생성될 수 있도록 한다.

2.5.2.2. 협동 플레이

멀티 플레이를 지원하여 두 플레이어가 방 안의 적을 함께 처치하는 협동 게임을 제작한다. 게임 플레이어들이 대체로 비협동 게임에 비해 협동 게임을 플레이했을 때 사용자 경험과 더불어 게임의 몰입감과 재미 모두 향상된다는 연구결과[1]가 있는 만큼, 증강현실을 이용한 협동 게임은 플레이어들에게 실제로 방 안에서 생성되는 몬스터와 싸우고 있다는 경험을 불러올 것이다.

3. 프로젝트 내용

3.1. 게임 소개

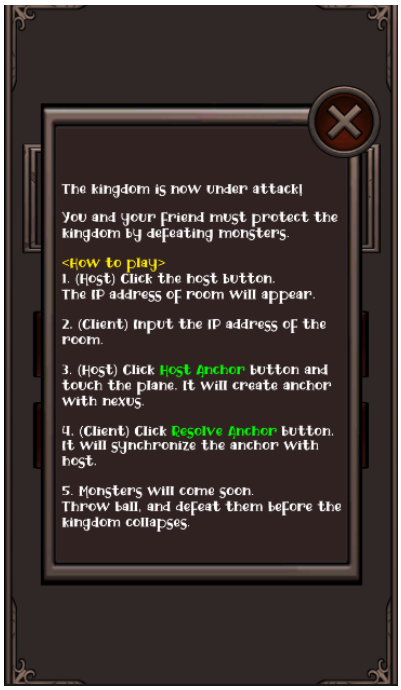
3.1.1 목표

스마트폰의 카메라를 통해 증강현실 환경에서 등장하는 적들을 파악하고, 몰려오는 적들을 친구와 함께 물리치는 멀티플레이어 디펜스 게임이다.

사용자는 스마트폰을 터치함으로써 넥서스(Nexus)라 불리는 오브젝트를 증강현실 환경 하에 생성할 수 있다. 몬스터들은 넥서스를 향해 다가올 것이고, 다가온 몬스터는 넥서스를 공격할 것이다. 몬스터의 공격으로부터 넥서스의 체력이 모두 떨어지기 전에 사용자는 몬스터에게 공격을 가해 몬스터를 쓰러뜨려야 한다.

3.2. 시나리오

3.2.1. 홈 화면



[그림 2] 홈 화면 [그림 3] Game Rule

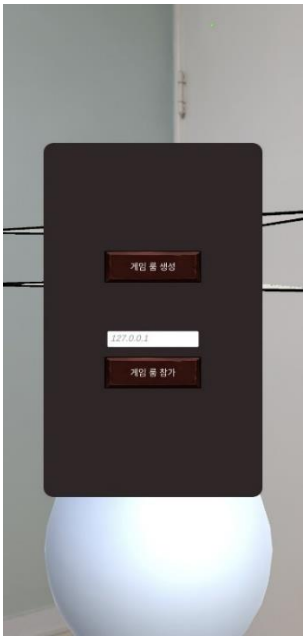
[그림 2]는 어플리케이션을 실행하면 가장 먼저 사용자에게 보여지는 홈 화면이다. 홈 화면에서는 게임명인 "AR Defence"를 표시하며, <Game Start>와 <Game Rule>의 두 버튼을 표시한다. 이 중 하단의 <Game Rule> 버튼을 클릭하면 **[그림 3]**의 게임 방법이 화면에 출력된다. **[그림 3]**에서 사용자에게 제공되는 게임 방법은 다음과 같다.

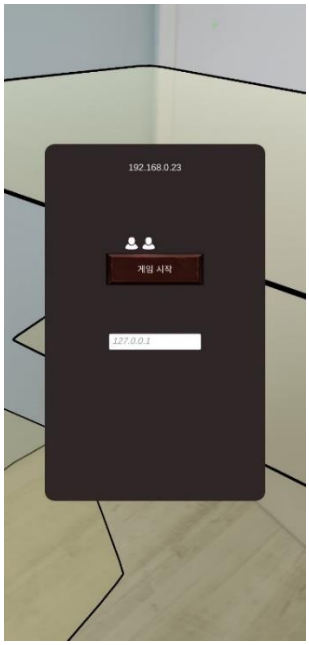
<Game Rule>

1. **호스트의 경우**, host 버튼을 클릭하고 게임 화면에서 <게임 룸 생성> 버튼을 클릭한다. 화면에 생성된 게임 룸의 IP 주소가 나타날 것이다.
2. **클라이언트의 경우**, 화면에 나타난 input 레이블에 호스트로부터 건네받은 게임 룸의 IP 주소를 입력하고 <게임 룸 참가> 버튼을 클릭한다.
3. **호스트의 경우,** <Host anchor> 버튼을 클릭하고 화면 상 평면(plane)을 터치한다. 터치를 통해 앵커와 함께 넥서스가 생성될 것이다.
4. **클라이언트의 경우,** <Resolve anchor> 버튼을 클릭한다. Host 와 좌표계가 동기화될 것이며, 성공적으로 수행되었다면 host 와 같은 위치에 넥서스가 생성될 것이다.
5. 몬스터가 곧 생성되어 넥서스를 침공할 것이다. 넥서스의 체력이 모두 소진되기 전에 몬스터에게 공을 던져서 몬스터를 모두 무찔러야 한다.

게임 방법을 확인한 사용자는 홈 화면의 <Game Start> 버튼을 클릭함으로써 후술할 게임 룸 화면으로 이동할 수 있다.

3.2.2. 게임 룸 화면



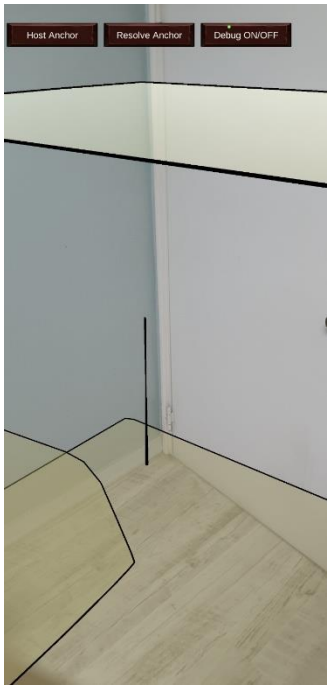


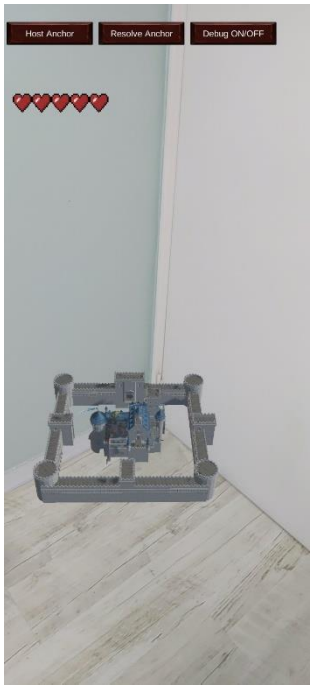
[그림 4] 게임 룸 화면 [그림 5] 게임 룸 입장 화면 (Host 디바이스)

[그림 4] 은 홈 화면에서 <Game Start> 버튼을 통해 진입한 게임 룸 화면이다. 홈 화면의 Game Rule 에서 상술했듯, 같이 게임을 플레이할 사용자들 중 호스트 역할을 맡을 디바이스에서 <게임 룸 생성> 버튼을 클릭하면 화면 상에 IP 주소가 출력된다. **[그림 5]**에서 화면 중앙 캔버스의 상단에 IP 주소가 출력된 것을 확인할 수 있다. 이렇게 출력된 IP 주소를 클라이언트 역할을 맡은 디바이스에서 **[그림 4]** 중앙부의 input 레이블에 입력하면 두 디바이스간 통신이 활성화되게 된다. 이 때 주의할 점은, 같이 게임을 플레이할 유저들은 반드시 같은 wifi 에 접속해 있어야 한다는 점이다.

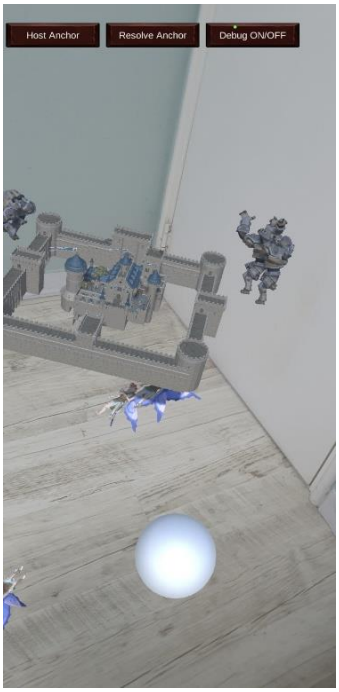
클라이언트가 호스트의 게임 룸에 접속하면 호스트의 화면에는 **[그림 5]**와 같이 클라이언트가 접속했음을 표현하는 아이콘이 출력된다. 사람의 인원을 의미하는 아이콘이 두 개 나타난 것을 통해 두 대의 디바이스가 게임 룸을 통해 서로 통신 중이라는 것을 알 수 있다. 또한 게임 룸을 생성한 호스트의 디바이스에서는 **[그림 4]**의 <게임 룸 생성> 버튼이 **[그림 5]**에서와 같이 <게임 시작> 버튼으로 대체된다. 게임 인원이 충족되었고 호스트가 이 버튼을 클릭하면 호스트와 클라이언트는 본격적인 게임 화면으로 진입한다.

3.2.3. 게임 화면





[그림 6] 게임 화면 진입 [그림 7] Resolve Anchor 결과 (Client 디바이스)



[그림 8] 몬스터 생성 (Host 디바이스) [그림 9] 몬스터 처치 (Host 디바이스)

AR Plane 은 현실 환경에서의 바닥, 벽 등의 평면을 인식하여 표현된 AR 상의 평면이다. **[그림 6]**에서 AR Plane 이 노란색으로 시각화되어 사용자의 화면에 표시되고 있는 것을 볼 수 있다. 호스트 플레이어는 <Host Anchor> 버튼을 클릭하고 화면 상의 AR Plane 을 터치하면 anchor 가 호스팅된다. Anchor 의 호스팅과 함께 호스트의 화면에서는 AR Plane 상에 Nexus 가 생성되었음을 확인할 수 있다. Anchor 의 호스팅에는 약간의 시간이 소요되며, 주위 환경의 인식 정도 등에 따라 호스팅에 실패할 수도 있는데, 이 때 <Debug On / Off> 버튼을 통해 런타임 상 디버그 텍스트를 출력시키면 anchor 호스팅이 제대로 수행되었는지를 알 수 있다.

호스팅이 성공적으로 수행되었다면 클라이언트는 <Resolve Anchor> 버튼을 클릭하여 호스팅된 anchor 를 resolve 할 수 있다. 만약 resolve 가 성공적으로 수행되었다면, 클라이언트 디바이스에서도 호스트 디바이스에서 지정한 nexus 의 위치가 동기화되어 화면에 나타날 것이다. 즉, 이 과정을 통해 호스트와 클라이언트는 동기화된 nexus 와 좌표계(클라우드 앵커 좌표계)를 공유하게 된다.

위의 시각 자료에서 **[그림 7]**은 클라이언트 디바이스, **[그림 8, 9]**는 호스트 디바이스이다. 두 디바이스에서 모두 비슷한 위치에 nexus 가 생성되었음을 확인할 수 있다.

이제 지켜야 할 Nexus 도 생성되었으니, 잠시 후 nexus 를 파괴하려는 몬스터들이 자동 스폰될 것이다. **[그림 8]**의 하단에 흰색 공이 존재하는 것을 볼 수 있다. 플레이어는 스크린에서 이를 드래그하여 증강 현실 내에서 던질 수 있으며, 플레이어의 드래그 방향 및 속도 등에 따라

던지는 공의 물리량이 결정된다. 플레이어들이 넥서스를 향해 달려오는 몬스터들에게 공을 던지면 공에 충돌한 몬스터는 쓰러지게 된다. 몬스터들은 플레이어의 시야의 외부를 포함한 다양한 지점에서 등장하므로, 각 플레이어들은 서로의 시야의 사각을 보완하며 등장하는 몬스터들을 쓰러뜨리기 위해 협동할 필요가 있다. 이렇게 목표치만큼의 몬스터를 제거하면 게임을 승리할 수 있다. 반대로, 몬스터들이 nexus 를 공격하여 **[그림 7]**의 좌측 상단에서 확인할 수 있는 하트 모양 UI 로 표현된 nexus 의 체력이 모두 떨어지게 된다면 플레이어들은 게임에서 패배하게 된다.

[그림 10] 플레이어 아바타

추가로, 동료 플레이어의 디바이스의 위치를 네트워크를 통해 동기화하여 파악한다. 이를 이용하여 스크린에 보이는 동료 플레이어의 외관에 아바타를 입힐 수 있다. 동기화되는 Pose 값은 position 과 rotation 을 모두 포함한 값이므로 플레이어가 이동하거나 회전하더라도 그에 맞춰 아바타가 업데이트된다.

3.2. 요구사항

3.2.1. Android Device 에 대한 요구사항

- AR foundation 의 ARCore 를 지원하면서 Depth API 를 지원하는 Android 기기.
(https://developers.google.com/ar/devices)

- Android 7.0 이상, API 레벨 24 이상.

3.2.2. Cloud Anchor를 이용한 통신에 대한 요구사항

- 게임을 함께 플레이하는 유저들 간의 게임 룸이 생성되도록 구현한다.
- 네트워킹을 담당하는 별도의 서버를 두지 않고, Wi-Fi를 통한 LAN 연결을 통한 직접 연결으로 호스트가 되는 플레이어의 디바이스에서 서버 역할을 수행하도록 한다.
- 게임을 시작하기 전에 호스트와 클라이언트 간 연결이 정상적으로 진행되었는지를 확인할 필요가 있다. 호스트에서 생성한 게임 룸에 클라이언트가 접속했는지의 여부를 확인할 수 있도록 별도의 UI가 요구된다.
- 게임 플레이 중 필요한 부분에 대해서 실시간으로 게임 오브젝트의 Pose 값이 갱신되어 각 플레이어의 디바이스에 동기화되어야 한다. Pose는 오브젝트의 rotation(회전) 정보와 position(위치) 정보를 통칭한다.
- 게임 플레이어들은 증강 현실 내에서 동일한 좌표계를 가져야 한다. 이를 위해 호스트와 클라이언트 디바이스에서 Cloud Anchor를 host하고 resolve하는 과정을 거칠 필요가 있다.

3.2.3. Depth Image를 이용한 몬스터 스폰 위치 getter에 대한 요구사항

- 3.2.1에서 상술한 Depth API가 지원되는 디바이스가 요구된다.
- Google ARCore에서는 AR Occlusion Manager로부터 depth map을 얻을 수 있다. 따라서 구현 과정에서 ar session origin이 AROcclusionManager 컴포넌트를 가져야할 필요가 있다.
AR Occlusion Manager는 카메라로부터 얻은 이미지로부터 depth를 판단하여 Occlusion(폐색)을 관리하는데, occlusion을 통해 더욱 사실적인 증강 현실 경험을 가질 수 있다.
예를 들어, 현실 세계의 탁자의 위에 가상의 오브젝트를 올려두고 탁자의 아래 방향으로 카메라를 옮겨 같은 오브젝트를 바라보면 이를 관측할 수 없다. 가상의 오브젝트가 탁자의 평면에 가려졌기 때문이다. 이와 같이 현실과 가상의 오브젝트 간의 depth를 판단하여 물체의 가려짐을 사실적으로 표현하는 역할을 수행하는 것이 occlusion manager이다.
- 스마트폰의 후면 카메라로부터 얻은 depth image를 얻어 프로그램에서 사용할 수 있는 형태로 만들어야 한다.
예를 들어 ARCore에서 카메라를 통해 얻은 depth image는 카메라에 의해 가로로 길쭉한 형태이다. 그러나 이를 screen space에서 사용자에게 직접 보이는 화면과 매칭되게 하기 위해서는 depth image를 screen의 너비와 높이에 맞게 늘리고, 세로로 길쭉한 screen에 맞춰 회전시키는 작업이 수반될 필요가 있다.
- Depth Image는 말 그대로 image이기 때문에 2차원 형태의 좌표를 가진다. Depth image를 통해 depth가 급격히 변하는 지점의 Vector2 좌표를 구할 수 있는데, 이를 3차원인 현실에 맞춰 사용하기 위해서는 Vector2 좌표를 적절하게 Vector3 좌표로 변환하기 위한 방법이 필요하다.
Depth의 변화 지점을 찾는 로직을 간단히 표현하면 다음과 같다. OcclusionManager로부터 얻은 depth image를 통해 스크린에는 표시되지 않는 rawImage를 업데이트하고, red color로 init된

rawImage에서 인접한 픽셀들간의 red color 값을 비교하여 일정 값 이상이면 Vector2의 list 형태인 locations 배열 변수에 할당한다. 결과적으로 리턴되는 locations에는 카메라로부터 얻은 이미지로부터 depth가 급격히 바뀌는 지점들의 vector2 좌표들이 포함된다.

```
tw = texture.width; th = texture.height;
sw = Screen.width; sh = Screen.height;;
for (int i = 3; i < tw; i += 3)
{
    for (int j = 3; j < th; j += 3)
    {
        if (texture.GetPixel(i - 3, j).r - texture.GetPixel(i, j).r > 0.1f
        || texture.GetPixel(i, j - 3).r - texture.GetPixel(i, j).r > 0.1f)
        {
            locations.Add(new Vector2(i, j));
        }
    }
}
return locations;
```

- 몬스터가 공중이 아닌 바닥에서 걸어가며 Nexus를 향해 이동할 것이기 때문에, 몬스터는 AR plane 상에서 생성될 필요가 있다. 이를 고려하여 앞서 얻은 Screen 상에서의 Vector2 좌표에 raycast를 수행함으로써, 만약 해당 좌표에 AR Plane이 존재한다면 raycast된 world position의 Vector3 좌표를 구함으로써 앞선 Vector2 -> Vector 3 좌표의 변환 작업을 수행할 수 있다.
아래는 위의 설명을 간단히 구현한 코드이다. spawnLoc은 상술한 depth image를 screen에 맞춰 회전 및 확대하여 변환한 texture로부터 얻은 vector2 좌표이다. Texture는 앞선 locations의 원소 중 랜덤한 Vector2 원소를 선정하였다.
몬스터를 바닥에서 생성하기 위해 AR Plane과 부딪힌 경우에만 raycast가 수행되도록 하였다. 이를 판별하기 위해 PlaneWithinPolygon TrackableType를 사용하였다. Raycast로 리턴되는 Pose의 position은 Vector3 좌표이므로 자연스럽게 Vector2로부터 Vector3 좌표의 변환을 수행하였다.

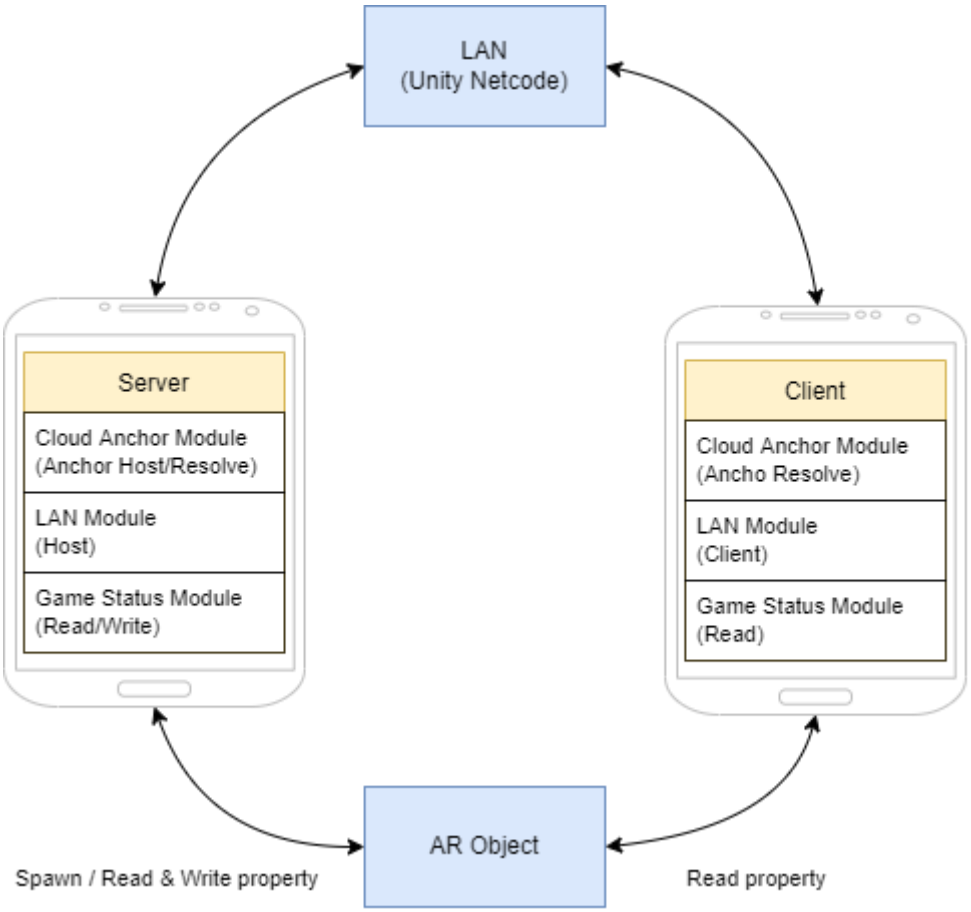
```
Vector2 spawnLoc = new Vector2(sw * (1 - textureLoc.y / th), textureLoc.x * (sh / tw));
List<ARRaycastHit> hitsList = new List<ARRaycastHit>();
if (raycastManager.Raycast(spawnLoc, hitsList, TrackableType.PlaneWithinPolygon))
{
        var h = hitsList[0].pose;
        return h.position;
}
```

3.2.4. 게임 플레이 환경 및 컨텐츠에 대한 요구사항

- Nexus의 체력을 하트 모양 UI를 통해 직관적으로 표현한다.
- 처치해야 하는 남은 몬스터의 수를 지속적으로 제공함으로써 사용자에게 목표를 부여한다.
- 멀티 플레이어 게임으로써 게임의 협동성을 높이기 위해 각 플레이어가 서로의 시야를 보완해야 하도록 한다. 몬스터가 항상 스크린 상에서 스폰되어 다가온다면 단순히 다가오는 몬스터를 쓰러뜨리는 형식밖에 되지 않으므로, 서로의 카메라가 바라보지 않는 곳에서 몬스터가 스폰될 필요가 있다. 이를 통해 사용자가 끊임없이 카메라를 움직여 스폰된 몬스터를 찾도록 유도한다.
- 너무 쉽거나 어려운 게임이 되지 않도록 게임의 난이도를 조정할 필요가 있다. 이를 위해서 던지는 공의 재생성 시간을 조절하거나, 몬스터의 스폰 주기 및 체력 등을 설정하여 게임 밸런스를 조정한다.
- 더욱 다양한 게임 플레이 경험을 제공하기 위해, 몬스터의 종류를 분화하고 그에 따른 특성을 부여할 필요가 있다. 예를 들어, 날아다니는 몬스터의 경우 AR Plane 상이 아닌 공중에서 스폰되게하고 이동속도를 더욱 빠르게 설정함으로써 그 형태에 맞는 특성을 가질 수 있다.
- 더욱 사실적인 게임 플레이 환경을 위해, 몬스터의 애니메이션을 적용할 필요가 있다. 몬스터의 이동 모션, 사망 시 쓰러지는 모션 등을 적용하여 게임성을 높인다.

3.3. 시스템 설계

3.3.1. 시스템 구성도



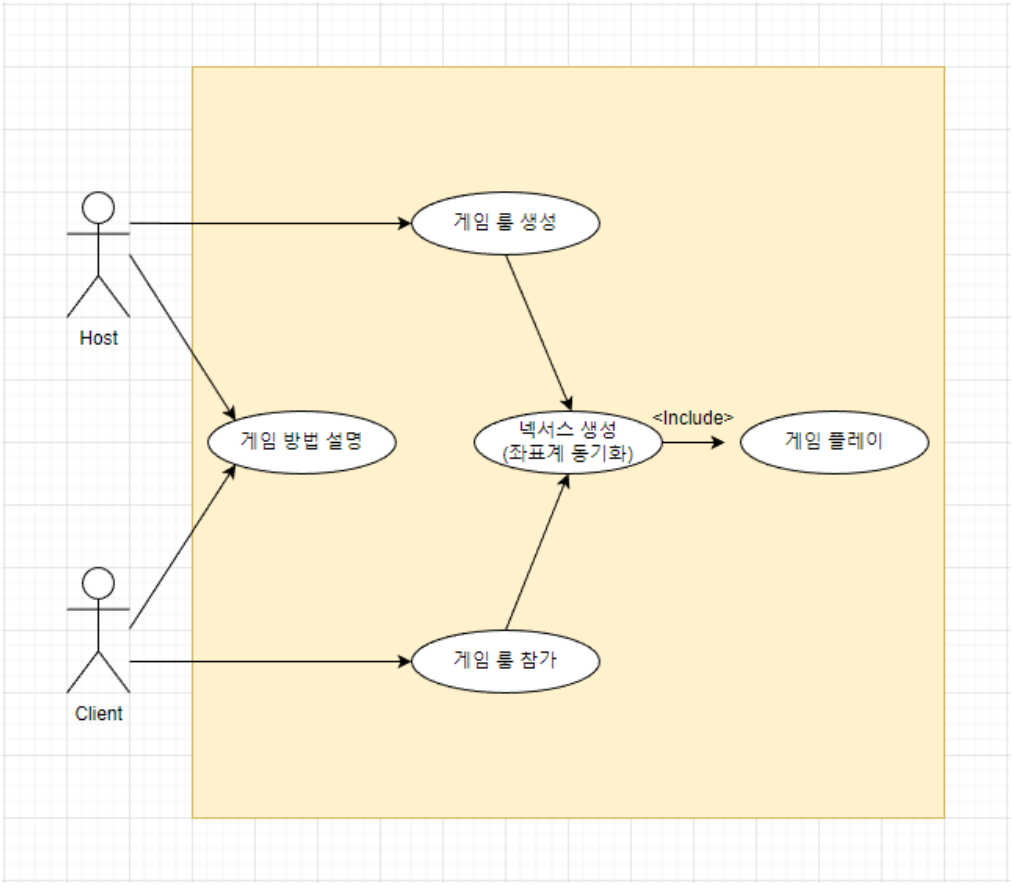
[그림 11] 시스템 구성도

시스템 구성도는 **[그림 11]**과 같다. 게임 내 통신은 클라이언트-서버 모델로 이루어진다. 서버 플레이어는 서버와 클라이언트를 동시에 수행하는 호스트(Host)의 역할로 게임에 참여한다. 각 디바이스에는 Cloud Anchor 를 관리하는 모듈, LAN 통신을 관리하는 모듈, 게임 상태를 관리하는 모듈이 존재한다. 서버 여부에 따라 모듈을 통해 수행할 수 있는 작업의 권한이 다르며 그 목록은 다음과 같다.

1. 서버에서만 Cloud Anchor 를 Host 할 수 있다.
2. 서버에서만 게임 상태 및 오브젝트 속성을 편집할 수 있다.
3. 서버에서만 AR Object 를 Spawn 할 수 있다.

3.3.2. UML Diagram 을 통한 시스템 모델링

3.3.2.1. Use Case Diagram



[그림 12] Use Case Diagram

본 게임의 Use Case Diagram 은 **[그림 12]**와 같다. Host 와 Client 는 모두 <Game Rule> 버튼을 통해 게임을 어떻게 플레이해야 하는지에 대한 정보를 얻을 수 있다.

Host 와 Client 는 Wifi 를 통해 무선 LAN 상에서 게임 룸을 접속하여 상호간 통신을 할 수 있다. 이를 위해서 Host 는 게임 룸을 생성하고, Client 는 생성된 게임 룸의 IP 주소를 입력함으로써 게임 룸에 접속한다. 서로 간의 통신 환경이 성공적으로 구축되었다면 두 디바이스에서 같은 증강현실 환경을 공유하기 위해 넥서스를 생성하고 resolve 하여 디바이스 간 좌표계를 동기화시킨다.

성공적으로 좌표계가 동기화된 것을 시스템 내부적으로 확인하게 되면 이제 양 측의 디바이스가 모두 게임을 플레이할 준비가 되었다는 것을 의미하므로, 본격적인 게임을 플레이하면 된다.

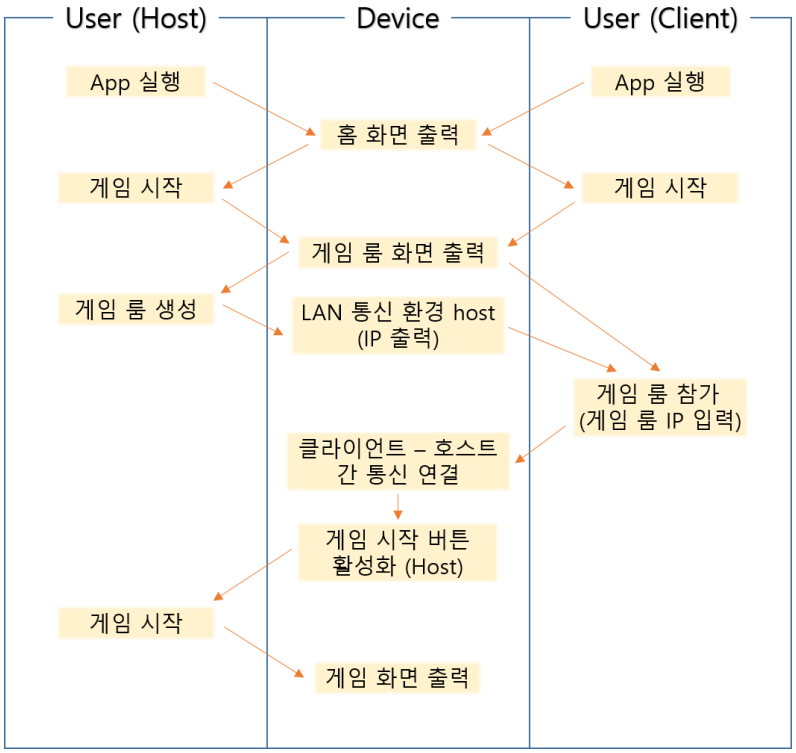
양 플레이어는 서로 협동하며 다가오는 몬스터에게 공을 투척하는 등의 행동을 통해 몬스터를 쓰러뜨리며 게임을 진행한다.

3.3.2.2. Class Diagram

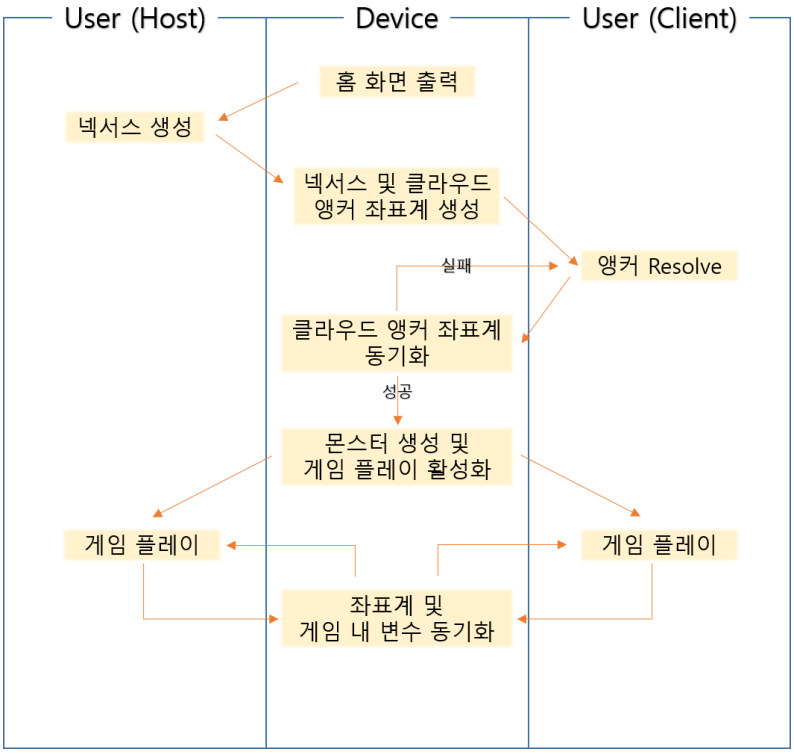
[그림 13] Class Diagram

[그림 13]에 나타난 5개의 클래스가 네트워킹 및 실제 위치 동기화, 게임 로직 수행의 역할을 수행한다. 먼저 LAN 클래스는 동일한 WIFI 상에서의 통신을 지원하는 역할을 한다. 게임이 시작하면 CloudAnchorManager 인스턴스를 생성하여 실제 위치 동기화 작업을 준비한다. 사용자가 앵커를 생성하고 Host 와 Resolve 를 수행하면 GameStatus 클래스에서 해당 정보를 참조하여 게임 상태를 변경한다. CloudAnchorManager 에서 몬스터를 생성하면 ARSyncObject 가 상대좌표와 상대회전각 정보를 관리하여 실제 위치 동기화를 수행한다. Monster 클래스는 몬스터의 정보를 관리하며, 발생한 이벤트에 따라 GameStatus 클래스를 참조하여 Alert 를 호출한다.

3.3.2.3. Activity Diagram



[그림 14] Activity Diagram (~게임 화면)



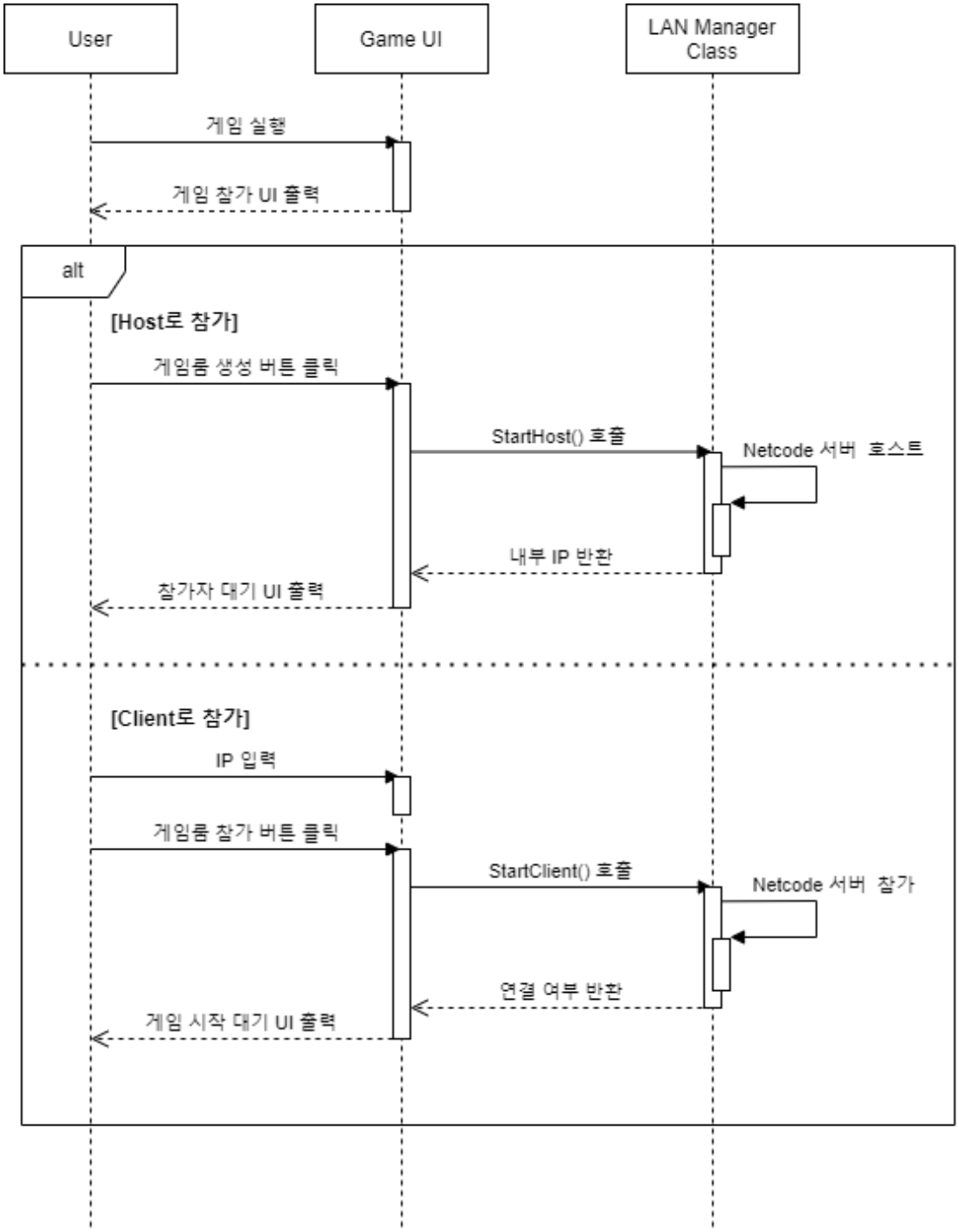
[그림 15] Activity Diagram (게임 화면)

Diagram 의 크기와 단계 별 특성에 따라 Activity Diagram 을 게임 화면의 출력 이전과 이후의 두 가지 diagram 으로 분할하였다.

[그림 14]의 Activity Diagram 에서 User 의 역할은 host 와 client 로 구분된다. 호스트 유저가 게임 룸을 생성하면 해당 디바이스는 서버 역할을 수행하게 되며, 게임 룸의 IP 를 출력한다. 출력된 IP 를 클라이언트 디바이스에서 입력함으로써 게임 룸에 참가할 수 있으며, 게임 룸 참가가 이루어지면 클라이언트와 호스트 간 통신이 활성화되며 호스트 유저에게 이를 알려준다.
동시에 게임 시작 버튼 역시 활성화되는데, 호스트 디바이스에서 이를 클릭하면 호스트와 클라이언트 양 측에 게임 화면이 출력되게 된다.

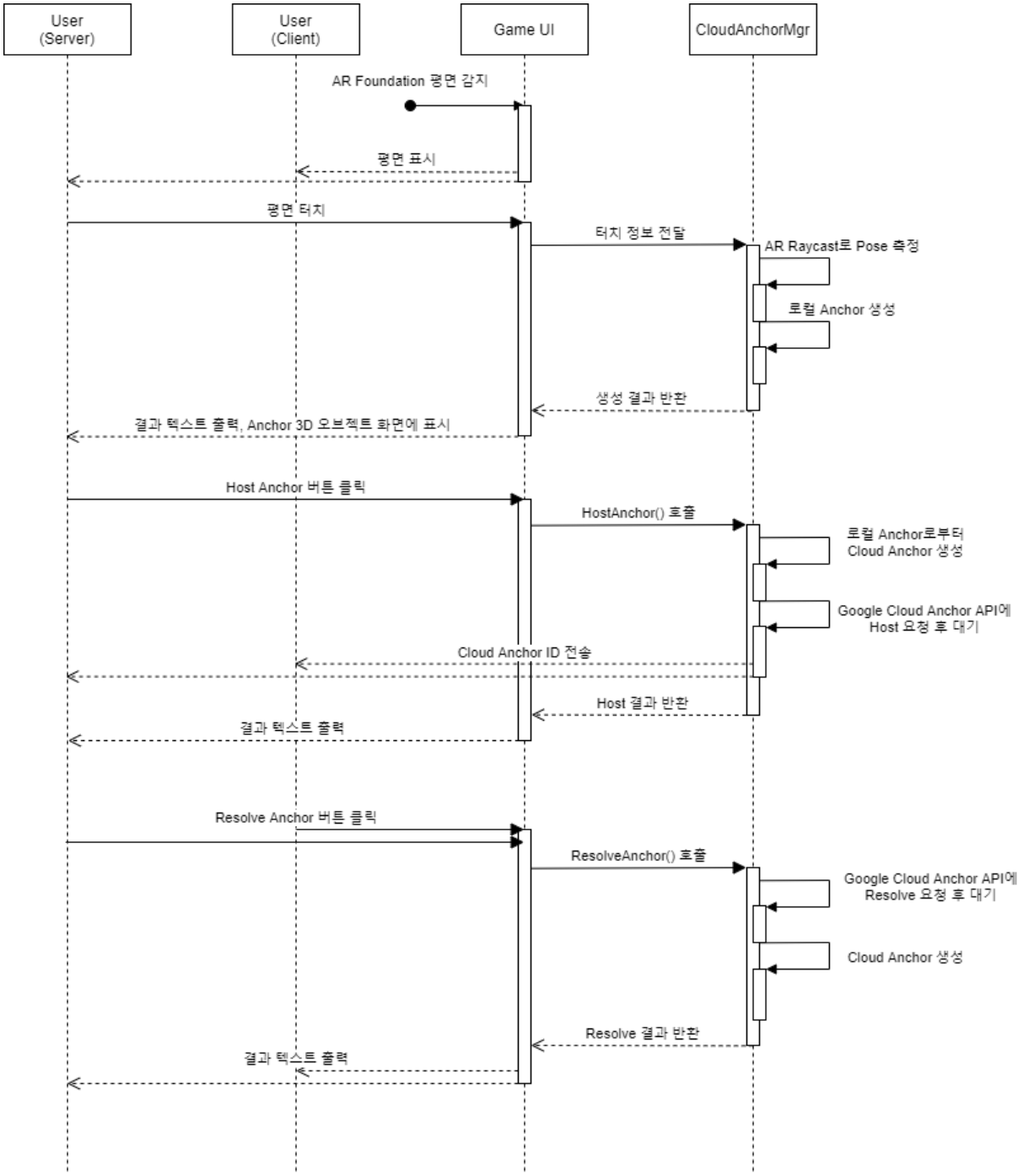
[그림 15]는 게임 화면에서의 Activity Diagram 이다. 호스트는 넥서스를 생성함으로써 클라우드 앵커 좌표를 초기화하고, 이를 클라이언트 디바이스에서 resolve 함으로써 서로간의 좌표계를 동기화시킨다. 동기화 작업이 성공적으로 진행되었다면, 게임 플레이 메커니즘이 활성화되며 본격적인 게임을 시작할 수 있다. 게임 플레이 환경에서도 오브젝트의 좌표계 및 게임 내 변수 등은 수시로, 혹은 필요에 따라 지속적으로 동기화된다.

3.3.2.4 Sequence Diagram



[그림 16] 게임 참가 Sequence Diagram

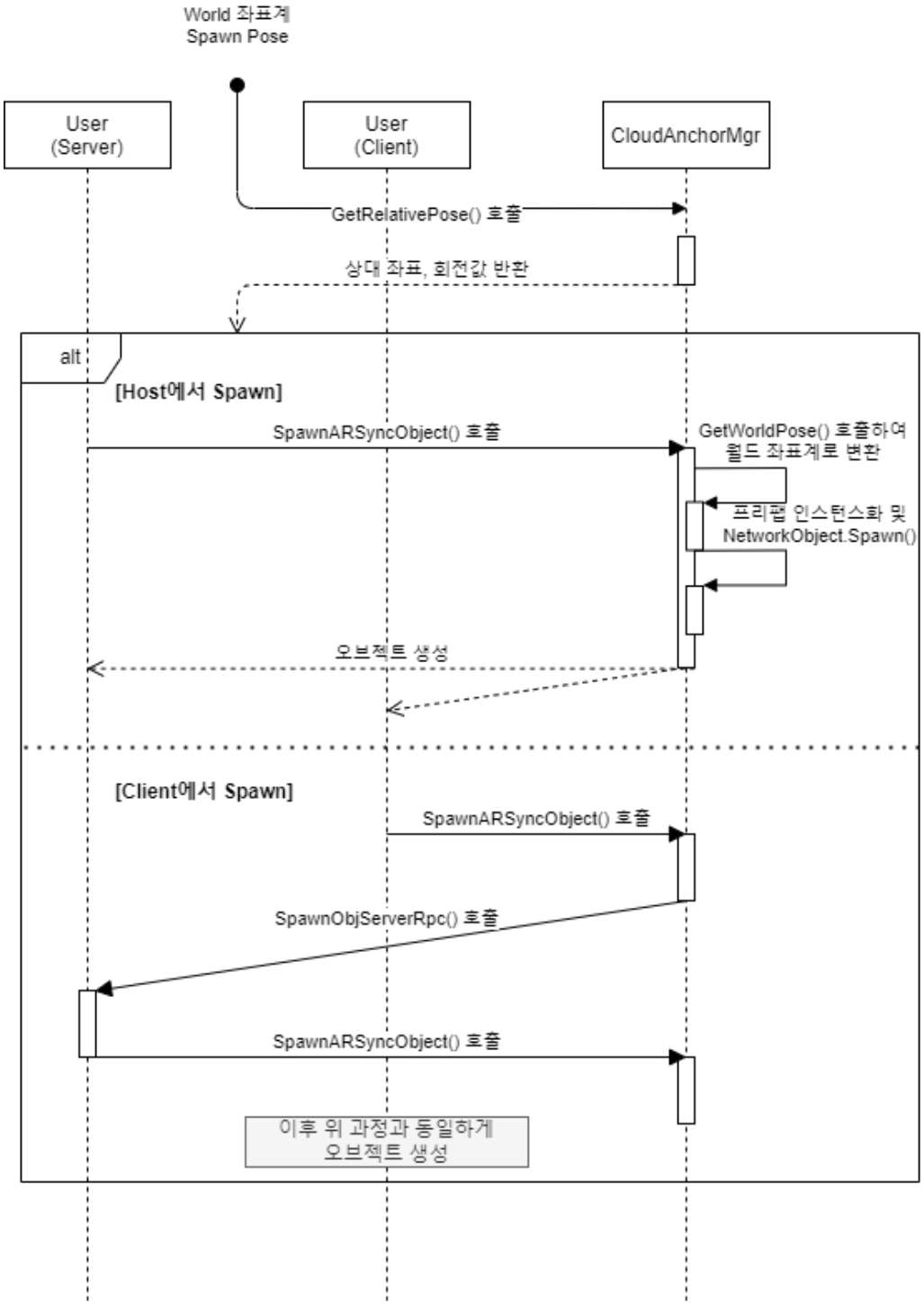
Host 로 게임에 참여할 경우에는 게임 룸 생성 버튼을 클릭하여 서버를 생성한 뒤 출력되는 IP 를 공유하여 게임을 진행한다. Client 로 게임에 참여할 경우에는 공유받은 IP 를 입력하고 게임 룸 참가 버튼을 클릭하여 게임을 진행한다.



[그림 17] Cloud Anchor Host & Resolve Sequence Diagram

앵커를 Host 하기 위해서는 먼저 로컬 앵커를 생성해야 한다. 로컬 앵커는 AR 화면 상의 평면을 터치하는 것으로 생성할 수 있으며 서버에서만 수행할 수 있다. 로컬 앵커를 생성한 후 Host 버튼을 클릭하면 Google Cloud Anchor API 에 Host 요청을 보내고, 성공적으로 Host 되었을 경우 모든 클라이언트에 Cloud Anchor Id 를 전송한다. Id 를 전달받은 후에는 Resolve 과정을 수행할 수

있다. Resolve 과정을 거치면 Id 로부터 Cloud Anchor 가 생성되며, 성공적으로 수행되면 현실에서 동일한 위치에 오브젝트를 배치할 준비를 모두 마친 것이다.

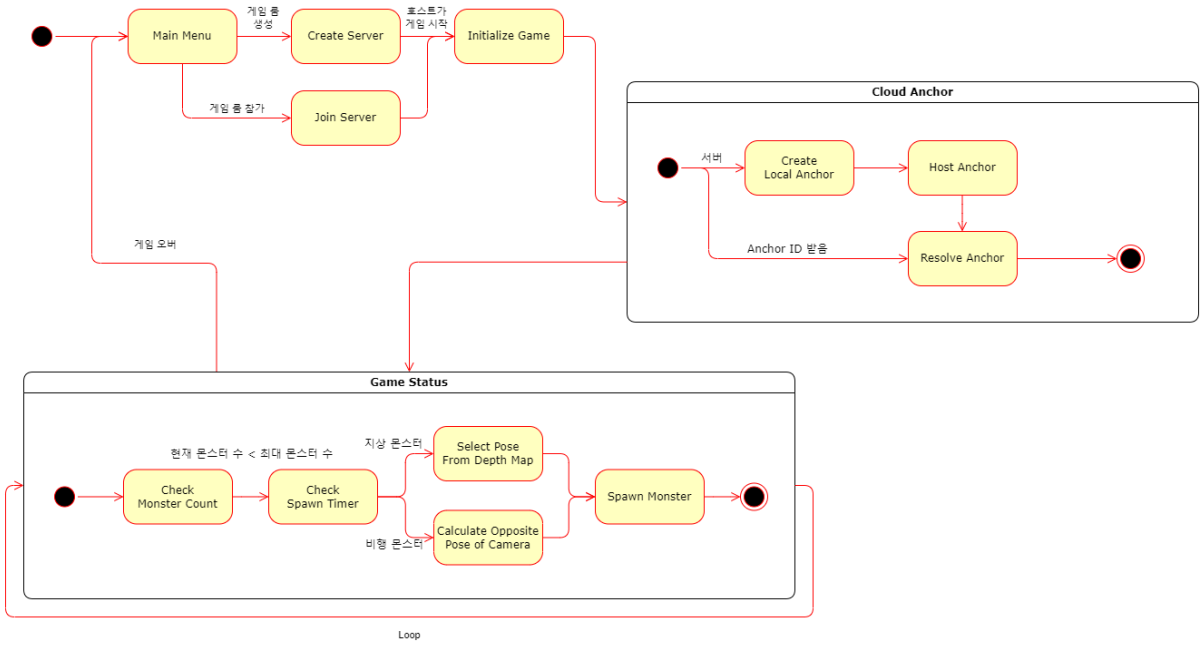


[그림 18] Spawn Object Sequence Diagram

증강현실 환경에서는 각 클라이언트마다 월드 좌표계가 다르기 때문에 오브젝트를 다룰 때 클라우드 앵커를 기준으로한 상대 좌표계로 정보를 주고받는다. 서버(호스트)에서 오브젝트를

spawn 할 때는 직접 프리팹을 인스턴스화 하여 로컬에 먼저 생성한 뒤 NetworkObject.Spawn()을 호출하여 클라이언트에 생성명령을 내린다. 클라이언트에서 오브젝트를 spawn 할 때는 서버에 RPC(Remote Procedure Call)를 보내 서버에서 spawn 할 때와 동일한 과정을 수행한다.

3.3.2.5 State Diagram



[그림 19] State Diagram

전체 게임의 진행 과정을 State Diagram 으로 작성했다. Cloud Anchor 를 생성하는 과정과 Game Status 를 관리하는 과정을 Sub State 로 구분하여 표현했다. 호스트 플레이어가 서버를 생성하고 다른 플레이어가 서버에 참가하면 게임 초기화가 이루어지고 Cloud Anchor Sub State 로 전이된다. Cloud Anchor Host/Resolve 프로세스가 마무리되면 메인 게임 로직을 수행하는 Game Status Sub State 로 전이된다. 해당 Sub State 는 게임오버 될 때까지 반복되며, 게임 내 변수들을 확인하여 게임 지속 여부를 결정하고 몬스터 생성 로직을 수행한다.

3.4. 구현

GitHub: https://github.com/KHU-Capstone-pARty/pARty

구현된 프로젝트 파일은 위 GitHub 링크에 업로드 되어 있다. 주요 파일들의 경로는 다음과 같다.

소스코드 경로	역할
Assets/AR/BallManager.cs	몬스터를 처치하기 위한 Ball 의 조작
Assets/AR/SpawnLocManager.cs	Depth Image 로부터 Spawn 위치 추출
Assets/Multiplayer/Scripts/ARSyncObject.cs	AR 멀티플레이 환경에서 오브젝트 동기화
Assets/Multiplayer/Scripts/AvatarCtrl.cs	플레이어 아바타 제어
Assets/Multiplayer/Scripts/CloudAnchorMgr.cs	Cloud Anchor 제어 및 동기화 관련 유틸리티
Assets/Multiplayer/Scripts/GameStatusMgr.cs	게임 상태 관리
Assets/Multiplayer/Scripts/LANMgr.cs	LAN 멀티플레이 관리
Assets/Multiplayer/Scripts/LightEstimation.cs	AR 환경에서의 광원 추정
Assets/Multiplayer/Scripts/MonsterCtrl.cs	몬스터 제어

Scene 경로	역할
Assets/Scenes/GameMenu.unity	게임 메인 메뉴 화면
Assets/Multiplayer/MultiTestScene.unity	AR 컨텐츠 플레이 화면

시스템 설계에서 계획된 내용들은 모두 위 소스코드 상에서 구현되었으며, 자세한 내용은 GitHub 에서 확인할 수 있다.

4. 프로젝트 결과

4.1. 연구 결과

본 프로젝트에서 제시하는 핵심적인 기능은 크게 두 가지가 있다. 하나는 클라우드 앵커를 통해 현실에서 동일한 위치에 오브젝트를 동기화하는 기능이고, 다른 하나는 Depth Map 으로부터 자연스러운 몬스터 생성 위치를 결정하는 기능이다. 이 절에서는 두 가지 핵심 기능이 요구사항에 맞게 구현되었는지 평가하고 기존 연구와의 비교를 수행한다.

4.1.1. AR Object 동기화 기술

[그림 20] 서버 디바이스 시점 (좌), 클라이언트 디바이스 시점 (우)

증강현실 환경에서는 각 디바이스마다 가진 World 좌표계가 일치하지 않는다. 따라서 증강현실에서 멀티플레이를 구현하기 위해서는 좌표계를 일치할 수 있는 방법이 필요하다. 기존의 멀티플레이를 지원하는 증강현실 게임에서는 GPS 를 통해 대략적인 위치만을 동기화하거나, 동기화하지 않아도 문제가 없는 종류의 게임을 개발하는 식으로 간접적인 해결방법을 사용한다.

본 프로젝트에서는 Cloud Anchor 를 기준으로 하는 상대 좌표계로 통신을 수행함으로써 동기화 문제를 해결한다. **[그림 20]**에서 보이는 것처럼 서로 다른 시점에서 바라보더라도 동일한 위치에 오브젝트들이 생성되는 것을 확인할 수 있다. 사진에서 성 모양 오브젝트가 Cloud Anchor 를 나타내는 오브젝트이며 그 주위로 몬스터들이 생성되어 동기화된 상태로 공격하는 모습을 볼 수 있다. (캡처 시간 차이로 인해 일부 몬스터들은 동기화되지 않은 것처럼 보인다.)

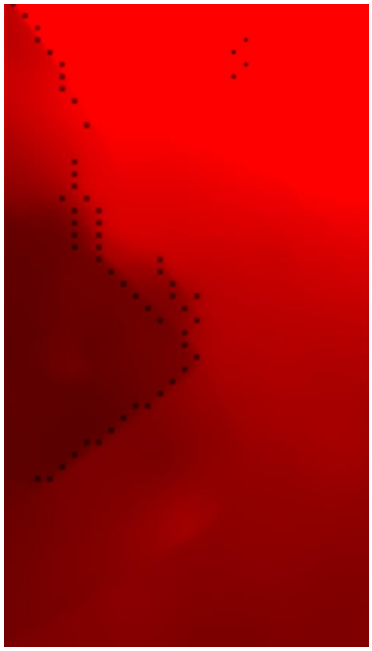
동기화는 오브젝트의 World 좌표계를 Cloud Anchor 로 뷰 변환(View Transform)을 수행하여 상대 좌표를 계산하는 것으로 이루어진다.

$$v' = RTv$$

클라우드 앵커 좌표계의 위치 벡터 v'는 World 좌표계의 위치 벡터 v에 앵커 방향으로의 회전행렬 R과 원점으로의 평행이동 행렬 T를 곱해주어 구할 수 있다. LAN 을 통해 전달받은 상대 좌표계 정보는 다시 World 좌표계로 동일하게 뷰 변환을 수행하여 실제로 생성될 위치를 결정한다. Unity 상에서는 Transform.TransformPose()와 Transform.InverseTransformPose() 함수를 통해 동기화 로직을 구현하였다.

4.1.2. Depth Map 을 활용한 스폰 위치 선정

Unity AR Foundation 에서 지원하는 ARCore Depth API 를 활용하여 현재 카메라로부터의 Depth Map 을 가져온다. Depth Map 에서 depth 가 급격히 바뀌는 지점 중 랜덤으로 하나의 지점을 몬스터 스폰 위치로 결정한다.



[그림 21] Depth 가 급격히 바뀌는 지점 시각화

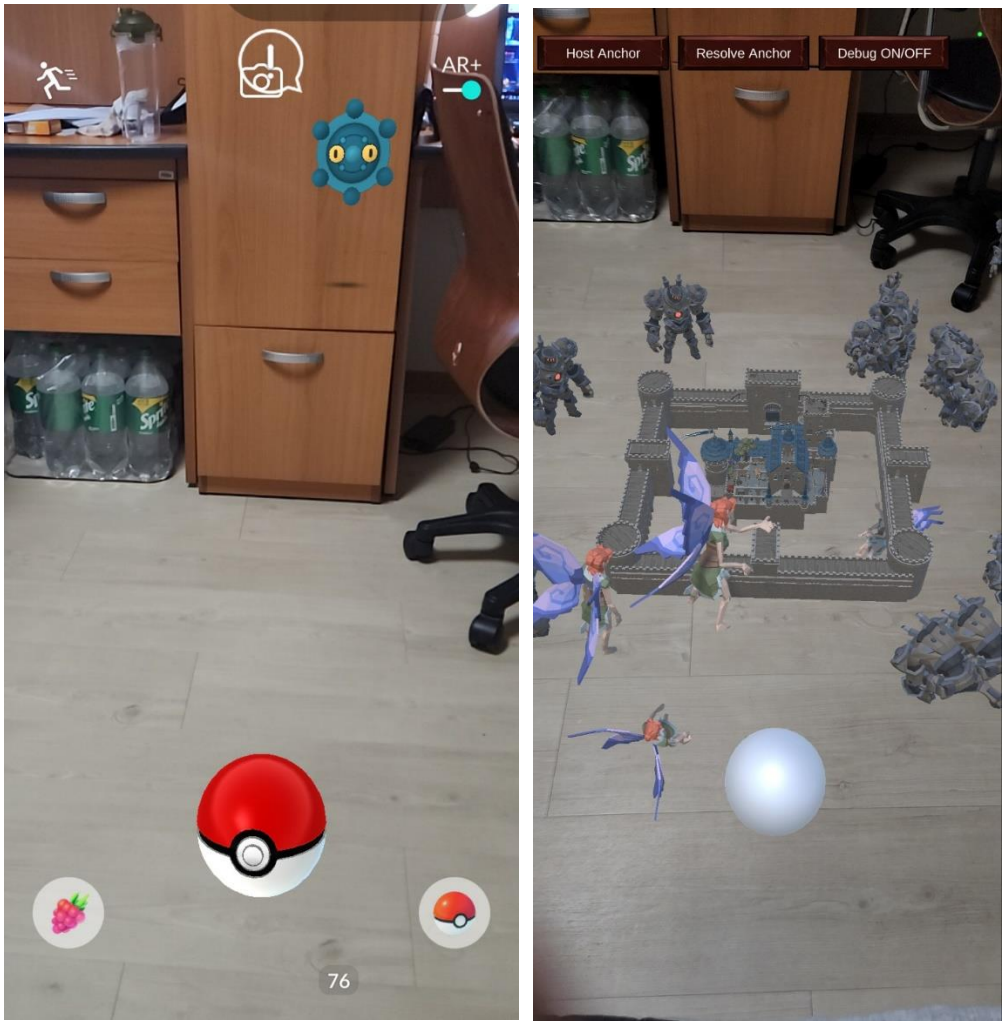
걸어 다니는 몬스터의 경우 공중에서 생성될 시 게임의 배경인 현실과 동떨어져 보이기 때문에 현실 환경에서의 평면 위에서 생성되도록 구현하였다. 따라서 걸어 다니는 몬스터는 현실 환경에서의 물체와 배경 사이의 경계면에 해당하는 평면에서 생성되는 것을 확인할 수 있다.

그러나 몬스터의 스폰 위치 선정에 Depth Map 만을 활용한다면, 플레이어의 시야 내에서만 몬스터가 스폰할 수밖에 없다. 이러한 로직은 게임을 단순하게 만들고, 게임의 난이도가 매우

쉬워진다는 문제가 있다. 따라서 날아 다니는 몬스터는 플레이어의 시야 밖에서 생성되며 이때 Depth Map 은 활용되지 않고, 플레이어가 바라보는 방향을 y 축 기준 180° 회전한 방향에서 랜덤한 위치에 몬스터를 생성한다.

4.2. 성능평가

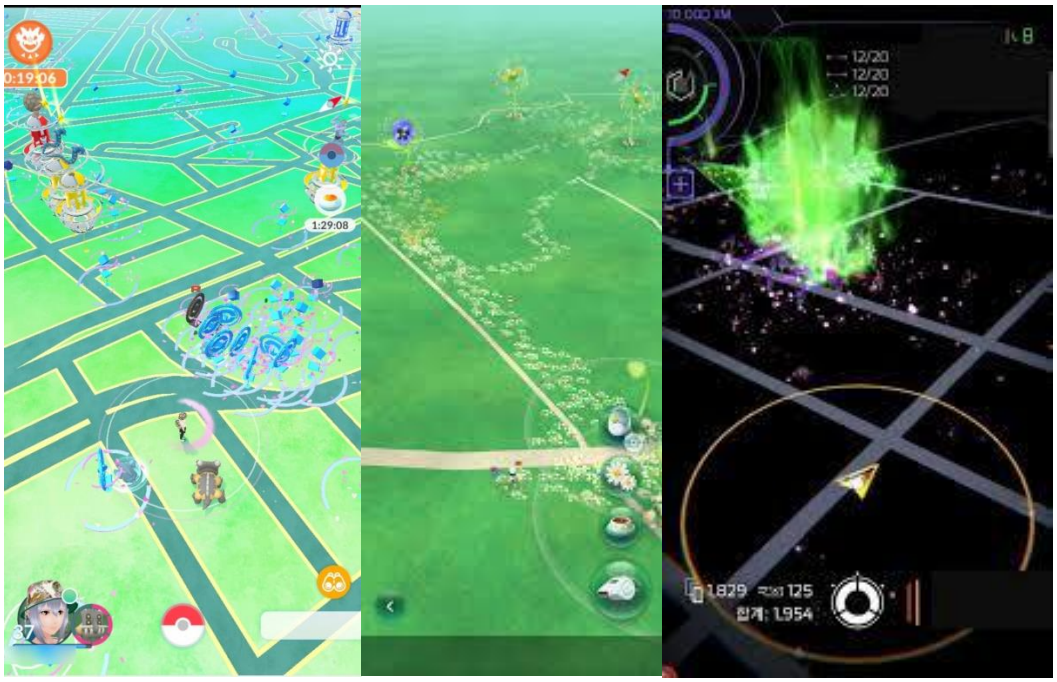
Google Play 상위 증강현실 게임과의 비교를 통해 연구 목표로 삼았던 GPS 기반 증강현실 게임들과의 차별성을 평가하고자 한다.



[그림 22] 포켓몬 고 (좌), AR Defense (우)

대표적인 GPS 기반 AR 게임인 '포켓몬 고' 와의 비교를 통해 현실과의 상호작용, 멀티플레이 경험 제공의 두 가지 측면에서 평가를 실시한다. 먼저 현실과의 상호작용면에서 포켓몬 고는 주변 환경을 배경으로만 활용한다. 포켓몬이 현실의 평면과 상호작용하거나 고정된 위치에

존재하지 않고 배경에 떠 있는 정도의 사용자 경험을 제공한다. 본 프로젝트에서 개발한 AR Defense 는 몬스터들이 현실의 평면을 걸어다니고 고정된 위치에 주요 오브젝트가 존재한다.



[그림 23] GPS 기반 AR 게임에서의 멀티플레이
(순서대로 '포켓몬 고', '피크민 블룸', '인그레스')

'포켓몬 고'뿐만 아니라 대부분의 GPS 기반 증강현실 게임들이 **[그림 23]**과 같은 지도 상에서의 멀티플레이를 제공한다. 특정 랜드마크에서만 다른 플레이어들과 상호작용을 할 수 있는 구조로 설계 되어있다. 이처럼 기존 게임이 수십 미터의 오차 범위로 멀티플레이 경험을 제공한다면, 본 프로젝트의 AR Defense 에서는 **[그림 20]**에서 보이듯이 몇 센티미터 정도의 오차 범위로 멀티플레이 경험을 제공한다. 또한 실시간으로 동기화되는 기능을 더함으로써 근거리에서 더 복잡한 상호작용을 정밀하게 제공할 수 있다.

5. 결론

5.1. 기대효과

첫째로 증강현실의 목표인 현실 환경에 가상 환경이 결합되어 가상 데이터가 증대된 현실을 사용자에게 제공한다. 플레이어는 현실 환경의 지형지물을 토대로 생성된 가상의 몬스터가 넥서스를 향해 다가오는 모습을 볼 수 있다. 기존에 흥행했던 GPS 기반의 증강현실 게임인 '포켓몬 고'와 이를 모방한 여러 게임들 뿐만 아니라, 본 프로젝트와 동일한 장르인 시중에 출시된 AR 디펜스 게임에서도 경험할 수 없는 현실 환경과 가상 환경의 상호작용을 방 안에서 느낄 수 있다.

둘째로 멀티플레이를 지원하기 때문에 두 플레이어가 함께 협동하며 게임을 즐길 수 있다. 협동 플레이는 비협동 플레이에 비해 플레이어의 게임 몰입도와 흥미를 모두 향상시킨다. 따라서 두 플레이어 사이 동기화된 몬스터의 위치와 협동하여 몬스터를 처치하는 플레이는 사용자에게 실제로 방 안에서 생성되는 몬스터와 싸우고 있다는 경험을 불러올 것이다.

5.2. 추후 연구 방향

현재는 함께하는 플레이어가 던지는 공이 스크린에 표시되지 않는 등의 미구현 기능이나 버그가 존재하는데, 이를 보완하여 게임의 몰입감에 방해가 되는 요소를 제거한다.

한정된 프로젝트 기간으로 인해 구현하지 못한 부가 컨텐츠를 추가한다. 부가 컨텐츠 중 하나인 스킬 시스템은 게임머니를 소모하여 스킬을 사용할 수 있으며 적절한 타이밍에 적절한 위치에 스킬을 사용하는 전략적인 플레이를 통해 게임의 완성도를 높인다. 플레이어는 몬스터를 처치할 때마다 몬스터 종류에 상응하는 게임머니를 획득할 수 있다.

더하여 공이 실제 지형의 평면에 부딪쳐 튕기면서 적을 처치할 수 있는 기능을 구현한다. 이는 기존에 의도했던 실제 환경과 가상 환경의 상호작용이라는 증강현실의 목표에 더 근접할 수 있도록 한다.

마지막으로 안드로이드 디바이스들만 플레이할 수 있는 문제를 해결하기 위해 여러 플랫폼에서 작동할 수 있도록 한다. 특히 ARCore 와 유사한 기능을 지원하는 프레임워크인 ARKit 를 통해 iOS 디바이스를 지원한다면 안드로이드-안드로이드, iOS-안드로이드와 같이 서로 다른 플랫폼끼리 함께 게임을 즐길 수 있을 것이다.

6. 참고문헌

[1] Lakshmi Sushma Daggubati(2016), Effect of cooperation on players' immersion and enjoyment

[2] ARCore 개발자 문서: https://developers.google.com/ar/develop

[3] ARKit: https://developer.apple.com/kr/augmented-reality/arkit/