Unity를 이용한 3D 눈싸움 게임 제작 및 PvP 플레이 구현

한국외국어대학교
공과대학
디지털정보공학과

박선우

2016年 10月

Unity를 이용한 3D 눈싸움 게임 제작 및 PvP 플레이 구현

위 논문을 학사학위 논문으로 제출합니다.
지도교수: Ho Yub Jung

2016年 10月

대학 : 한국외국어대학교
학과 : 디지털정보공학과
학번 : 201201270
이름 : 박선우

박선우의 학사학위 논문을 심사하여
합격으로 판정합니다.

심사위원: (인)

Unity를 이용한 3D 눈싸움 게임 제작 및 PvP 플레이 구현

요약문

Ⅰ. 서 론

1. 개발의 필요성

지금까지 게임 시장에 출시되었던 눈싸움 게임들은 2D 그래픽 디자인의 단순한 디펜스 형식의 게임이 대부분이다. 3D FPS 방식으로 눈싸움 게임을 만들어 눈싸움 게임이 가지고 있었던 기존의 진부하다는 느낌을 없애고, 박진감 넘치는 게임으로 인식을 바꾸고 싶다는 생각을 토대로 이 주제를 가지고 게임을 제작하게 되었다.

Ⅱ. 개발 목표

1) 두 종류의 컨트롤러를 이용하는 게임
2) TCP/IP 통신을 이용한 PvP 환경 제공
3) AI를 이용한 적 유닛의 움직임 구현

Ⅲ. 개발 내용 및 방법

1) 두 종류의 컨트롤러 적용

- 마우스와 키보드를 통해 동작하게 하는 함수를 구현하였음. 이후, Kinect Manager를 참고하여 Kinect 동작에 대한 이해를 바탕으로 Kinect를 통한 캐릭터의 움직임을 직접 구현하였음.

2) 통신을 이용한 다인 모드 플레이 구현

- Unity Relay Server를 이용한 패킷 전송 방식을 이용해 Host와 Client간의 통신을 원활하게 함.

3) AI를 이용한 적 유닛의 움직임 구현

- 기존의 A* 알고리즘을 참고한 Path-Finding 알고리즘을 구현해 장애물 플레이를 좀 더 현실적으로 구성하고, Escaping&Hiding 알고리즘을 이용해 몬스터의 움직임을 다양하게 만들어 게임의 박진감을 살리는데 목표를 둠.

Ⅳ. 개발 결과 및 문제점 해결

1. 개발 결과

1) 디자인 구현

- 캐릭터의 외형과 애니메이션을 직접 제작해 게임의 이질감을 없앰.
- 맵 오브젝트의 경우 Unity에서 제공하는 이미지를 수정해 직접 배치하여 사용함.

2) AI 개발

- Unity에서 기본적으로 제공하는 함수 중 하나인 NavMeshAgent 함수를 사용했을 경우, 이동 불가 지역에 들어가면 움직일 수 없는 현상 발생.

3) PvP 모드 개발

- Unity에 내장된 HLAPI(네트워킹 명령 세트)를 이용해 서버를 구현해 호스트와 유저들 간의 통신이 가능하게 하고, 유지 관리가 편리하도록 만듬.

2. 문제점 해결

1) Grid 방식 Path-Finding 알고리즘 개발

- 우선순위 큐를 이용하여 작업시간 단축
- 맵의 모양에 상관없이 자유롭게 적용 가능
- 직접 구현한 알고리즘이기 때문에 기능 추가가 용이함

2) Hiding 알고리즘 개발

- 적 유닛의 체력이 일정 수준 이하로 떨어졌을 때, 플레이어 캐릭터로부터 멀리 벗어나 숨은 뒤, 체력을 회복시키는 AI 개발

Ⅴ. 향후 개발 내용

1. 향후 개발 목표

- 컨트롤러를 더 추가하여 다양한 플랫폼에서 게임이 작동할 수 있도록 하며, 게임의 전체적인 스토리를 보완하여 컨텐츠 부족 문제를 해결한 뒤 게임 시장에 출시할 계획.

목차

그림 목차

1. 서론

1. 연구 주제

과거와는 다르게 현재 게임 시장은 끊임없는 진보와 발전을 통해 성장해 왔다. 우리나라 최초의 머드(MUD, Multi User Dialogue) 게임인 <단군의 땅>부터 시작하여 머드 게임에 그래픽을 추가한 머그(MUG, Multi User Graphic) 게임을 거친 뒤, MMORPG(Massive Multi-player Online Role Playing Game) 순으로 발전해 왔다.

중간 생략된 많은 게임의 종류가 있겠지만, 가장 많이 발전한 분야 중 하나는 바로 FPS(First-Person Shooter) 라고 할 수 있을 것이다. 플레이어의 시점, 자신이 사물을 보는 시점과 같은 화면에서 무기나 도구를 이용해 플레이하기 때문에 게임 속 캐릭터의 시점과 플레이어의 시점이 동일해야 하며, 보통 3D 방식으로 제작된다. 처음 FPS 게임이 시장에 등장했을 때에는 단순한 방식과 패턴을 가지는 온라인을 지원하지 않는 싱글 모드 게임들뿐이었지만 시간을 거듭할수록 발전해 <스페셜포스>, <서든어택> 등의 온라인 FPS 게임이 등장하였고, 수많은 FPS 게임들이 아직 두 게임의 아성을 넘지 못했다.

위의 두 게임은 실제 전쟁터를 방불케 한다. 그래서 일부 게임들은 천편일률적인 군사물과 차별을 두기 위해 마법을 사용하는 판타지 FPS, 미래의 무기를 사용하는 SF FPS 등을 표방하기도 했다. 하지만 해당 게임들은 시장에서 큰 성공을 거두지 못했고 이후, 상위권을 굳건히 지키고 있는 두 게임 내에서 하나의 모드로 출시가 되고 있는 실정이다.

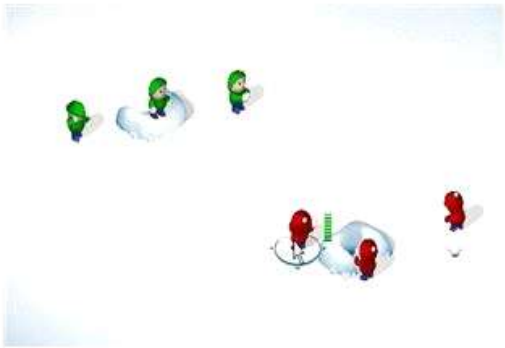
그래서, 기존의 전쟁을 주제로 한 FPS 게임을 만드는 것에 치중하기 보다는 다른 방식의 FPS 게임을 만들어 보고자 하였다. 기본적의 틀은 같다. '슈팅 게임' 이란 것은 무언가를 쏘는 게임이기 때문에 총을 쏘거나, 무언가를 던져 상대방이나 적 유닛을 맞추는 것에 기본 틀을 두는 것을 잊지 않았다.

그래서 생각해낸 방식이 바로 '눈싸움'이다. 눈을 던져 상대방을 맞추는 방식, 바로 그 방식이 슈팅 게임의 방식이며 눈싸움을 주제로 하여 많은 컨텐츠를 가진 게임을 만들 수 있기 때문에 본 연구를 통해 게임을 제작하게 되었다.

2. 개발의 필요성

1) 개발 동기

어렸을 때부터, 눈싸움을 주제로 한 다양한 게임들을 즐겨왔다. [그림 1]과 같은 단순한 디펜스 방식과 마우스를 클릭해 숨어있는 적을 맞추는 눈싸움 게임이나, [그림 2]의 테니스 코트 형식의 맵에서 간단한 좌우 컨트롤만 이용해 서로 눈덩이를 던져 맞추는 방식의 게임들이 현재까지 게임 시장에 출시된 눈싸움을 주제로 만들어진 게임들이다.



[그림 1] 마우스 컨트롤 방식 눈싸움 게임 **[그림 2] 키보드 컨트롤 방식 눈싸움 게임**

[그림 1], [그림 2]와 같은 단순한 방식의 플래시 게임이 아닌, 3D FPS 방식으로 게임을 만들어 기존의 눈싸움 게임이 가지고 있었던 진부한 방식을 개선하고 지루하다는 이미지를 벗겨내고 싶다는 생각을 토대로 게임을 제작하게 되었다.

2) 기존의 부작용 해결

기존의 FPS 게임들은 전 세계적으로 가장 인기가 높은 장르이지만, 호불호가 명백히 갈린다. 그 이유는 '폭력성' 때문이다. 기존의 전쟁을 토대로 한 FPS 게임 특성상 피와 살이 튀는 폭력적인 장면들을 사실적으로 묘사할 수밖에 없는데, 이러한 게임들과 어린 아이들을 차단할 수 있는 마땅히 뾰족한 방법이 없어 사회적인 측면에서 큰 문제점 중 하나로 불거지기도 하였다. 또한, 평소에 FPS 게임을 즐겨 했던 사람들 중에서 대규모 총기 사건의 주범들이 있었다는 점이 밝혀지면서, 해당 FPS 게임뿐만이 아니라 모든 폭력성을 가진 게임을 대상으로 한 사회적 질타를 피할 수가 없게 되었다. 따라서 눈싸움을 주제로 해 폭력성을 배제하고, 캐주얼한 분위기가 담겨있는 맵과 캐릭터 및 효과를 담아 남녀노소 누구나 눈살을 찌푸리지 않고 사회적으로 문제가 되지 않는 FPS 게임을 만들고자 하였다.

3. 개발 목표

1) 개발 목표

앞서 언급한 대로, 기존의 문제점이었던 '폭력성'을 배제하고, 진부한 방식의 게임이 아닌 박진감 넘치는 게임을 만들고자 다음과 같은 개발 목표를 설정하였다.

첫째, 두 종류의 컨트롤러를 이용해서 게임을 만들었다. 기본 설정은 마우스와 키보드를 이용한 조작법이지만, 최근 들어 VR과 모션 인식 센서를 이용한 FPS 게임이 많이 출시되고 있고 유저들의 평도 좋기 때문에 키넥트를 이용해 손만 움직여 플레이하는 게임이 아닌 자신의 모든 신체 부분을 이용해 플레이 할 수 있게 하였다.

둘째, TCP/IP 통신(Unity 내부 Network 기능)을 이용해 서로 경쟁할 수 있는 환경을 제공한다. 이 개발 부분이 게임의 박진감을 살려줄 수 있는 하나의 중요한 요소라고 생각된다. 기존의 눈싸움 게임들은 전부 1인칭 모드의 스테이지 방식으로 스토리를 전개해 나가는 방식이었다. 기존의 스테이지 방식의 1인칭 모드로 눈덩이를 맞힌 적의 수를 계산해 점수를 매겨 사용자 간 경쟁을 활성화시킬 수 있는 모드를 추가하는 방법 이외에, Unity 어플리케이션 내부에서 제공해 주는 Network API를 이용해 상대방과 일대일로 눈싸움을 하며 경쟁할 수 있는 모드를 제공해 소위 '재미없다'라는 평을 없앨 수 있다.

셋째, 1인칭 모드의 스테이지 방식은 진부하다는 평이 대부분이었으므로, 이를 개선하기 위한 적 유닛의 AI를 개발해 적용한다. 단순한 패턴, 기존의 짜여 진 틀에서만 움직이는 적 유닛을 상대하는 것은 게임의 재미를 반감시킬 수 있는 요소이다. 따라서, A* 알고리즘을 구현해 장애물 플레이를 더욱 더 현실적으로 보여지게 할 수 있으며, Hiding 알고리즘을 통해 적 유닛들의 움직임이 다양해지므로 게임성이 증가하는 효과를 얻을 수 있다.

세 가지 목표를 설정하고, 그에 맞추어 게임을 개발했기 때문에 유저들의 흥미를 유발할 수 있다. 이 세 가지 목표 이외에도 게임 내에 세부적으로 설정한 목표에 대해서는 본론의 개발 내용 부분에서 추가적으로 설명하려고 한다.

2) 주요 개발 도구

- Unity 3D

다양하고 조작이 간편하므로 좀 더 쉬운 컨텐츠 개발이 가능하다. 그리고 다양한 플랫폼을 지원하기 때문에 어떤 환경에서든 실행될 수 있는 게임 개발에 용이하며 앞서 언급한 조작의 편리성 덕분에 빠른 개발 속도와 완성도를 가질 수 있다. 쉬운 컨텐츠 개발 효율성이 높기 때문에 그로 인한 게임성을 증가시킬 수 있으며 디자인을 전공하거나 그림을 직접 그리지 않는 대부분의 개발자에게서 나타나는 디자인 및 모델링 부분의 부족한 점을 Unity Asset Store에서 필요한 리소스나 툴을 다운로드받아 자신의 게임에 적용시킴으로써 해결할 수 있다. 또한, 내장된 Network API가 있어 통신 연결에 용이해 PvP 모드를 구현하는 데에 있어 매우 편리하다.

- Kinect

마우스와 키보드를 이용하는 방식은 매우 단순하다. 손가락을 이용해 원하는 명령을 키를 눌러 입력하는 것이다. 하지만 Kinect에는 색을 인식할 수 있는 카메라와 유저의 3차원 동작을 이해하기 위한 깊이 측정기, 거리를 측정하기 위한 적외선 센서, 그리고 소리의 방향을 알아낼 수 있는 마이크 등의 기본적으로 장착된 부품들이 있어 유저가 어떤 동작을 할 때, 그 동작을 게임 캐릭터의 동작과 연결만 시켜준다면 유저들은 마우스와 키보드를 이용한 단순한 방식의 플레이보다는 Kinect를 이용한 박진감 넘치는 플레이를 더욱 더 선호할 것이다.

- 3DMAX

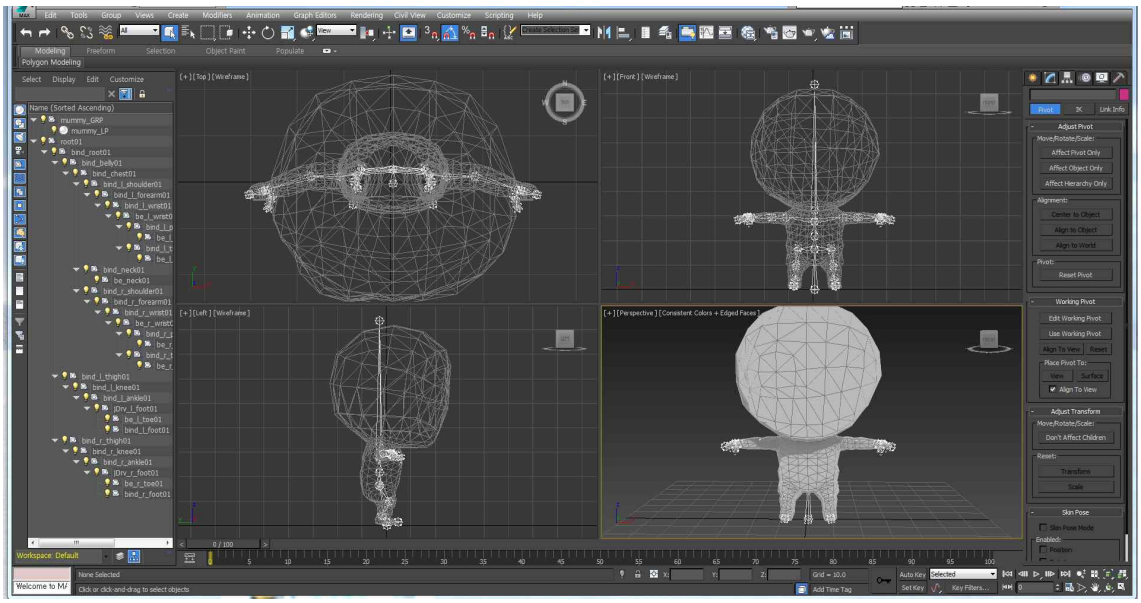
일반적인 프로그램에서는 그려내기 힘든 3차원 형태의 캐릭터의 외형이나 애니메이션을 구현하는 데에 사용한다. 해당 프로그램을 이용함으로써 캐주얼한 캐릭터의 모습과 애니메이션을 구현하였다.

2. 연구 및 개발 내용

1. 캐릭터 디자인

Unity를 이용한 게임이 한 분기에 23만개 정도 출시되고 있다. 그만큼 Unity 3D 프로그램을 이용해 제작한 게임이 시장에 많이 출시되어 있다는 뜻이고 그만큼 많은 사람들이 이 프로그램을 이용한다는 것이다. 그래서 Unity Asset Store가 활성화 되어있다. Unity Asset Store란 말 그대로 게임 제작 과정에 필요한 스크립트, 리소스, 툴 등 모든 것이 판매 혹은 유통되는 시장과 같은 개념이다. 이 스토어에서 게임 관련 툴을 다운로드 받아 사용하는 방법도 있으나, 새로운 FPS 게임을 개발하는 데에는 새로운 캐릭터가 필요하다고 판단되어 Unity Asset Store에서 다운로드받아 사용하지 않고 직접 제작하는 방법을 채택하였다.

1) 외형 제작



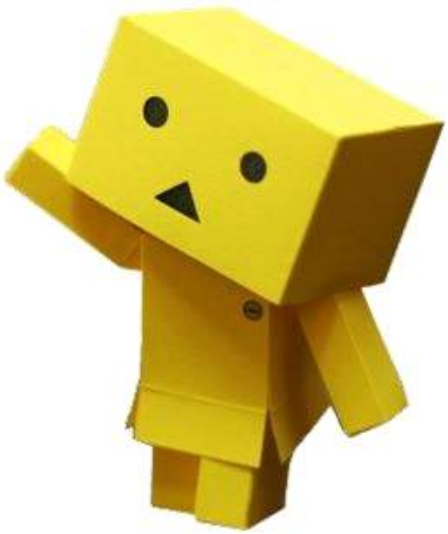
[그림 3] 3DMAX 프로그램을 이용한 캐릭터 외형 제작

캐릭터를 만들어 내기 위해서는 먼저 캐릭터가 보여질 모습, 즉 외형을 직접 만들어야 한다. 게임을 개발하는 데 사용하는 도구가 Unity 3D 프로그램이기 때문에 캐릭터도 3D로 만들어야 한다. 따라서 3DMAX 라는 프로그램을 사용해 캐릭터를 제작하였다.

3DMAX를 이용해 기본 폴리곤 방식으로 캐릭터 모델을 3차원으로 구현한다. 머리 부분의 구체를 먼저 그려낸 후, 원기둥을 납작하게 만들어 몸통으로 만든다. 나머지 팔과 다리 부분도 몸통과 마찬가지로 원기둥을 이용해 양 쪽 끝 부분을 사람의 형태로 보여지도록 넓이를 조정해 만들어 낸다. 각각의 부분을 모두 그려낸 후 사람의 형태에 맞게 붙여주면 캐릭터의 외형이 완성된다.

외형이 완성된 후, 그 위에 이미지를 붙여준다. 처음 게임 캐릭터를 디자인 했을 때에는 [그림 4]의 군인 모습으로 캐릭터를 디자인하였으나, FPS 게임이 가지고 있었던 '폭력성'을 해결하고자 하는 취지에 맞지 않아 캐릭터를 캐주얼한 모습으로 변경하게 되었다. 이후, [그림 5]와 같이 캐주얼한 모습의 캐릭터를 박스 형태로 제작하였으나 사람의 모습이 좀 더 드러나도록 하는 것이 좋다고 판단되어 [그림 3]의 캐릭터 형태로 최종 변경하였다.

[그림 4] 기존의 군인 모습 캐릭터



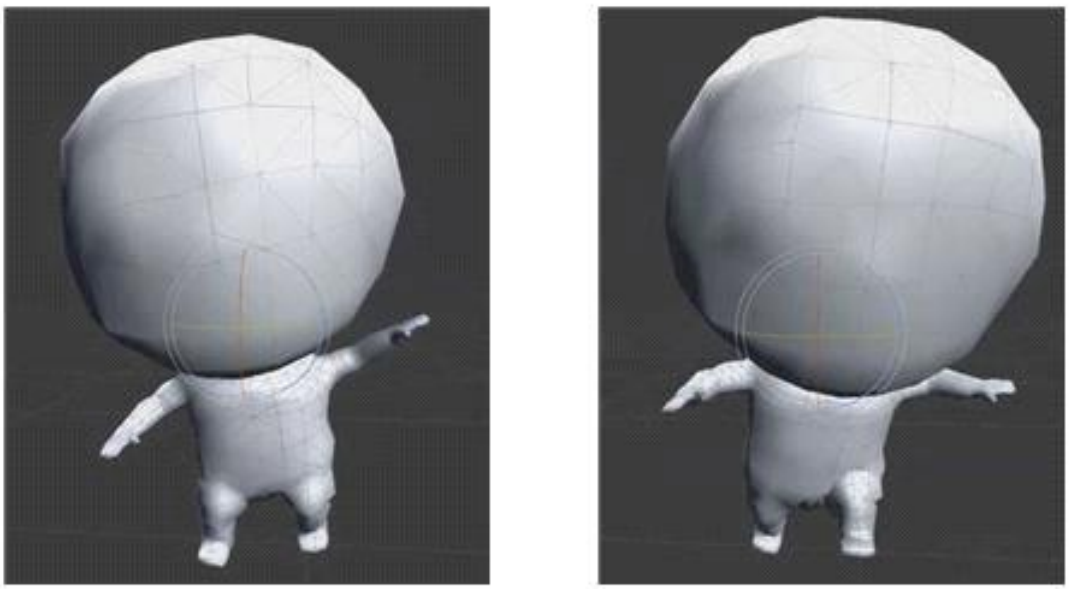
[그림 5] 캐주얼하게 변경한 캐릭터(예시)

2) 캐릭터 애니메이션 제작

캐릭터 모델링을 통해 캐릭터의 외형이 만들어졌으므로 캐릭터의 움직임을 애니메이션을 이용해 구현한다. 캐릭터 애니메이션은 외형 제작과 같이 3DMAX 프로그램을 이용하여 구현할 수 있다. 캐릭터의 애니메이션을 만들어 내는 방법은 보통 세 가지로 구성된다.

맨 처음 단계는 세팅이다. 세팅 과정이란 살(캐릭터 모델링을 통해 만들어진 캐릭터의 외형)에 뼈대를 붙이는 작업으로, 뼈들이 움직일 때 필요한 보조 뼈를 구성한다. 인체는 수많은 뼈로 이루어져 있으므로 실제 사람의 움직임을 구현할 때에는 캐릭터 내부에 많은 뼈대 구성이 필요하지만, 캐주얼한 캐릭터의 특성상 움직임이 제한되어 있고, 사람에 비해 비교적 간단한 움직임들이 일어나기 때문에 뼈대를 구성하기가 간편하다. 따라서, 중점적으로 움직이는 부분인 머리 부분과 팔다리 부분의 움직을 구현하기 위해 해당 부분에 넣을 뼈대를 만들었다.

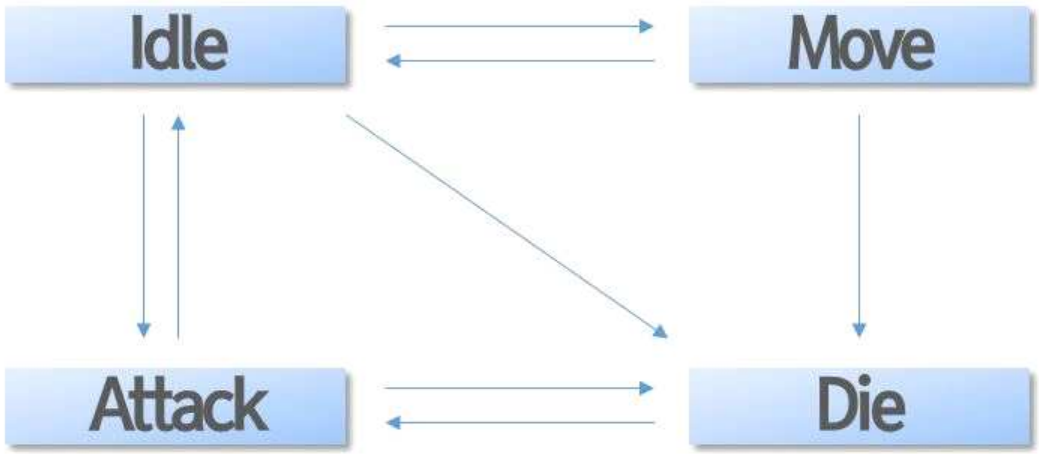
다음 단계는 리딩으로, 뼈와 살을 하나로 연결시켜 주는 작업이다. 이미 외형 제작 부분에서 살을 만들어 두었으므로, 뼈와 뼈가 이어지는 부분을 움직이는 부분인 머리와 팔 그리고 다리 부분에 연결점을 만들어 만든 뼈를 이어 붙이는 과정을 실행해 리딩 작업을 하였다.



[그림 6] 캐릭터 애니메이션 제작 시 프레임 단위로 움직이는 모습

마지막으로, 루프 애니메이션을 만드는 과정이다. 루프 애니메이션이란 처음과 끝이 연결되어 무한하게 반복하는 애니메이션을 말한다. 어떠한 한 동작과 다른 한 동작을 만들어 내는데, 반복적인 작업을 계속 해 줄 필요 없이 한 패턴의 동작을 만들어 낸 뒤 계속 반복시킨다. 루프를 만들어내기 위해 프레임 단위로 캐릭터의 모습을 변형시켜 저장하고, 이를 이어 붙여서 최종적인 캐릭터의 애니메이션을 만들어 냈다. [그림 6]은 프레임 단위로 이어붙이기 위해 캐릭터의 위치를 변경해주고 있는 모습이 담긴 그림이다.

캐릭터에는 FSM 차트를 통한 총 4가지의 상태가 있다. 아무것도 하지 않는 Idle 상태와, 움직이는 상태는 Move, 적을 만나 공격하는 모습을 보여주는 상태는 Attack, 그리고 체력이 전부 소진되어 캐릭터가 죽는 Die 상태가 있다.



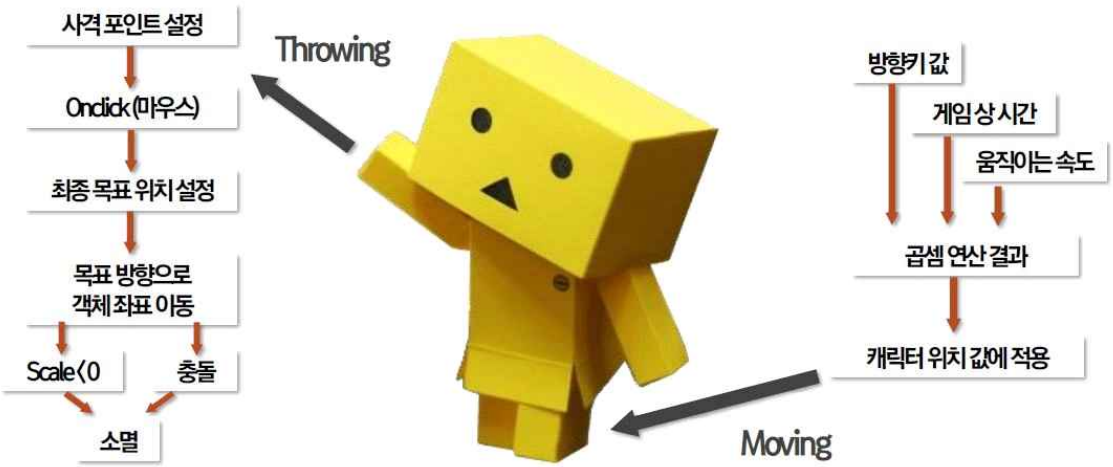
[그림 7] 캐릭터 움직임 FSM 차트

[그림 7]에서 양방향 화살표의 경우는 서로의 상태를 왔다갔다 할 수 있다는 뜻이다. 각각의 상태마다 각기 다른 애니메이션을 제작하여 구성하였으며, 움직이면서 공격을 하는 동작이 바로 이어지면 프레임이 끊기는 현상이 발생해 중간 동작에 Idle 상태를 거쳐 다음 동작으로 이어지게 함으로써 부자연스러운 현상을 해결하였다.

캐릭터 모델링 과정에 있어 적 유닛의 모델링도 직접 하였다. 그러나 적 유닛의 FSM 차트는 캐릭터의 FSM 차트에 비해 훨씬 복잡하게 이루어져 있으므로 AI 개발 부분에서 자세히 설명하려고 한다.

3) 캐릭터 객체 구현

[그림 7]의 FSM 차트를 통해, 기본적인 캐릭터 애니메이션을 구현하였다. 각각의 동작에 따르는 함수의 구성은 [그림 8]과 같다.



[그림 8] 캐릭터 객체 구현을 위한 함수의 구성

[그림 7]의 FSM 차트를 통해 보인 Attack의 경우는 Throwing이라는 함수로 구성되어 있다. 상대방의 좌표값을 받아와 사격 포인트를 설정한 후, OnClick함수를 통해 해당 좌표에 클릭 이벤트가 설정되면 최종적인 목표 지점의 위치가 설정이 된다. 이후 목표 방향으로 객체(눈)의 좌표를 이동시켜 목표 지점과의 충돌 시 객체를 소멸시키는 형태로 구현되어 있다.

[그림 8]의 Moving 이라는 함수는 [그림 7]의 Move 동작에 해당한다. 키보드를 통해 방향키 값 입력을 받아와 움직이는 위치에 따라 좌표를 변경한다. 그리고 게임 상 시간과 움직이는 속도값을 적용하여 단위 시간동안 움직일 수 있는 최대 거리를 계산하여, 방향키 값에 따르는 이동 좌표값을 곱해 주어 캐릭터의 위치 값에 적용해 캐릭터를 이동시킬 수 있다.

Die 상태는 체력이 모두 떨어졌을 때 발생하는 상태이므로, 캐릭터의 체력이 0이 되었을 때 상태 변경을 해 준다.

2. 맵 디자인

만들고자 하는 게임은 PvP(Player 간 경쟁)모드에서 사용할 맵과, 1인칭 모드에서 사용할 스테이지 맵 등 수많은 맵을 제작해야 했다. 맵을 만들기 위해서는 다양한 오브젝트가 필요했고, 직접 디자인해서 쓰는 경우 디자이너가 아니기 때문에 시간적 효율성이 매우 떨어졌고, Unity 3D 프로그램을 이용해 맵을 제작했기 때문에 Unity Asset Store에 있는 수정 및 재배포가 가능한 오브젝트를 가져와 수정하여 여러 개의 오브젝트를 만들어 두고 재배치하여 맵을 만들었다.

[그림 9] Unity Asset Store에서 제공하는 오브젝트를 수정 후 재배치해 만든 맵

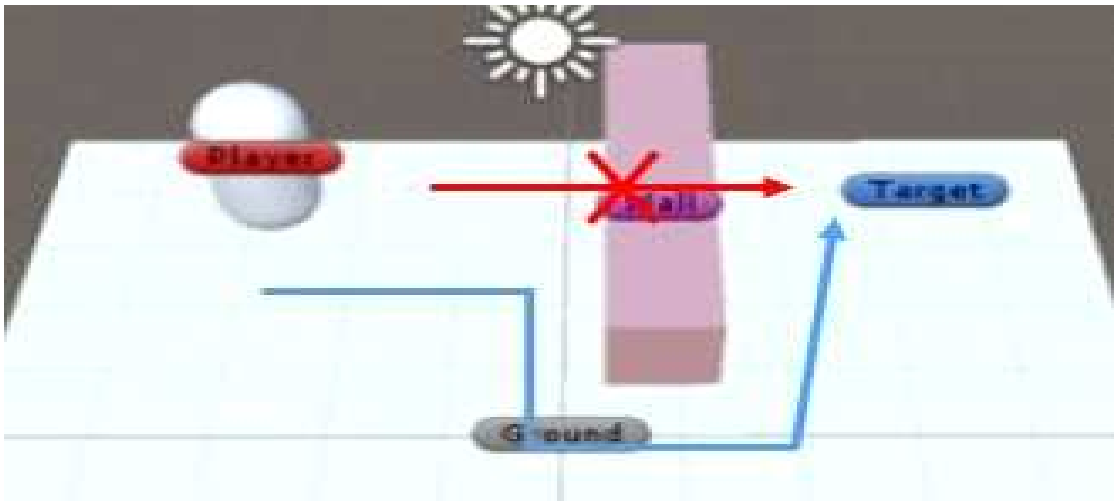
[그림 9]의 맵은 PvP용 맵으로 제작된 것이며, 1인칭 스테이지에 사용될 맵도 따로 여러 개를 제작하여 사용하였다. 상대방과 겨루기 위해서는 맵이 비교적 작아야 하며, 작은 맵 속 많은 장애물을 만들어 놓아 다양한 변수가 일어날 수 있도록 했으며, 스테이지 모드의 맵은 비교적 큰 맵과 큰 장애물 몇 가지를 이용해 물 흐르는 듯한 구성을 느낄 수 있도록 하였다.

3. AI 개발

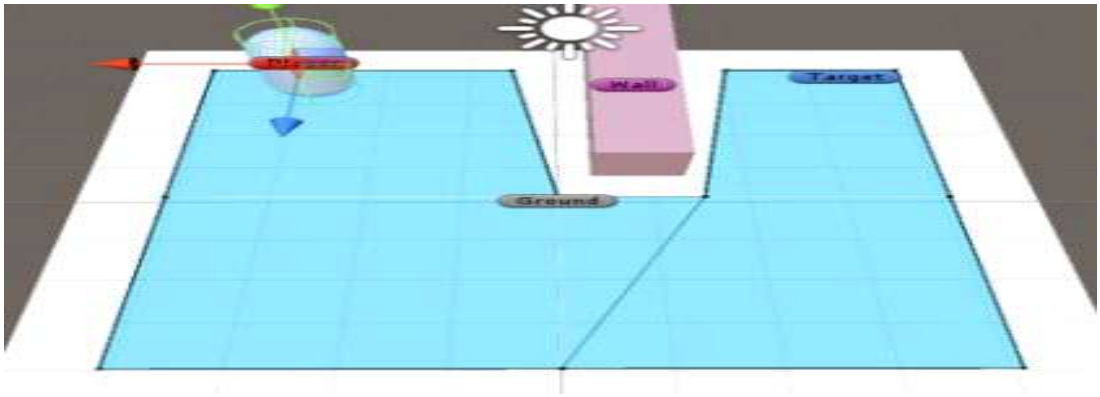
PvP 모드에서는 각각의 플레이어들과 경쟁하기 때문에 캐릭터의 움직임과, 통신 연결 시의 동기화 문제를 해결하면 되었다. 하지만, 1인칭 스테이지 모드에서는 적이 등장해 눈덩이를 맞춰 적을 없애면 점수가 오르는 방식이므로 적 유닛의 움직임이 단조롭지 않게 하는 것이 게임성을 증가시킬 수 있는 관건이 되었다.

1) 기존 알고리즘의 문제점

Unity 3D에서 제공해주는 다양한 함수 중, NavMeshAgent라는 길 찾기 알고리즘이 있다. 먼저, 이동할 플레이어와 피해갈 벽, 이동 목표의 타겟과 그라운드로 이루어진다. 기본적인 이동의 경우 [그림 10]의 빨간색 선을 따라 타겟 지점으로 이동하지만, 벽이 있어 이동하지 못한다. 그래서 벽에 막혀 이동하지 못하고 밀려나는 상황이 발생하는 데, NavMeshAgent 알고리즘을 사용하면 파란색 화살표 방향으로 이동할 수 있다.

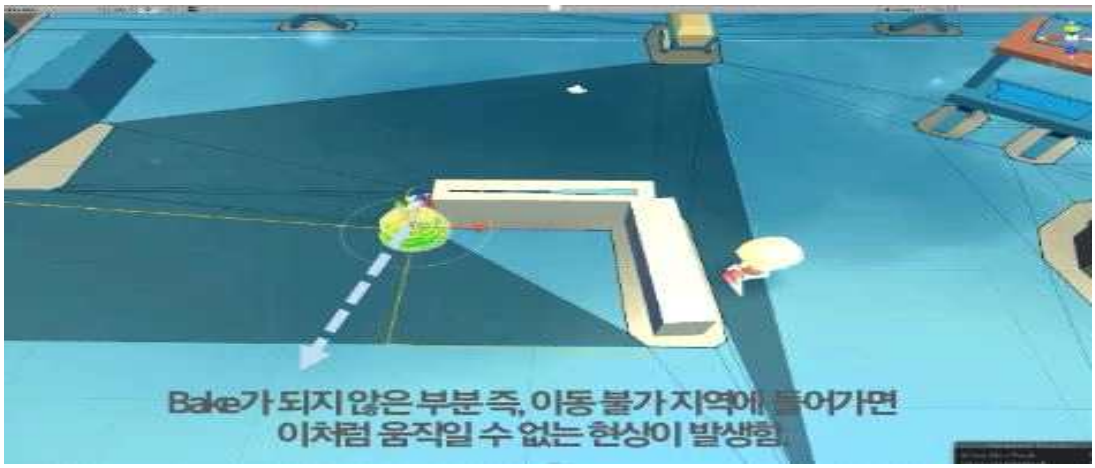


[그림 10] Unity에서 제공하는 NavMeshAgent



[그림 11] NavMeshAgent를 통해 Bake 한 부분의 모습

이후, 세팅(Bake) 과정을 통해 세팅된 부분에 들어오는 정보를 모두 받아 적용 시킬 수 있다. [그림 11]에서 그라운드 부분에 파란색으로 표현된 부분이 Bake된 부분이다. 바로 이 부분에서 문제점이 발생하게 되었다. Bake 된 부분이 벽의 경계선과 일치하기 않기 때문에, 벽과 부딪히는 상황이 발생했을 시 Bake 가 되지 않은 부분에 적 유닛이 걸리게 되면 제대로 움직이지 못하는 현상이 발생했다.



[그림 12] NavMeshAgent의 문제점

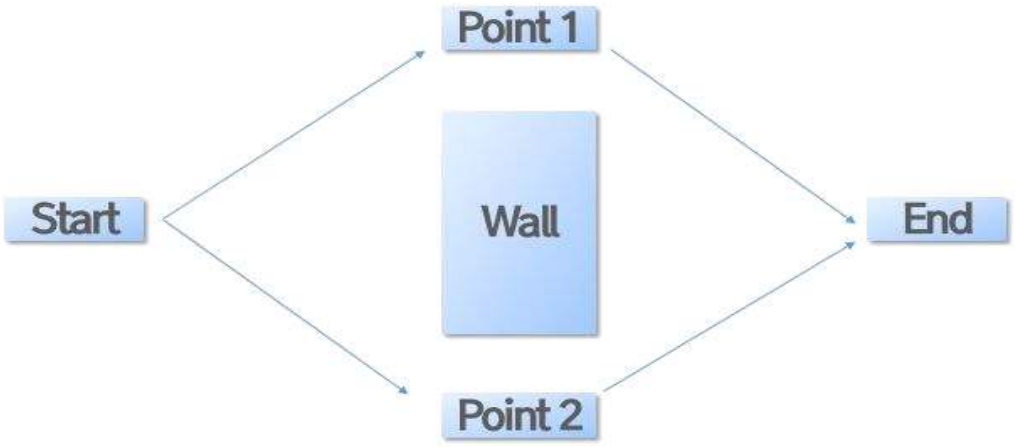
[그림 12] 에서 진한 파랑색으로 보이는 부분이 Bake가 된 부분이다. 적 유닛이 캐릭터를 쫓아가다가 벽 두개로 인해 막혀 지나가지 못하는 상황이다. Bake가 되지 않은 부분에서는 캐릭터를 잘 찾아가지만, Bake가 되지 않은 부분을 지날 시 이동 불가 상태가 되어 아무것도 하지 못한다.

이와 같은 문제점은 게임에 적용했을 때 매우 심각한 버그가 될 수 있다. 또한, 매끄럽지 못한 적의 움직임으로 인해 게임성이 반감된다. 맵 특성상 장애물이 매우 많고 맵이 그리 크지 않기 때문에, 맵의 각 오브젝트가 가지가 있는 Bake가 되지 않은 부분에서 적 유닛들이 이동 불가 상태로 머물러 있다면 적 유닛이 길 찾기 알고리즘을 제대로 실행하고 있지 못한다는 지표가 된다.

2) A* 알고리즘

위와 같은 문제점 발생으로 인해, 만들고자 했던 게임과 Unity에서 제공해주는 길 찾기 알고리즘인 NavMeshAgent 함수는 어울리지 않는다는 결과를 얻었다. 그래서 게임에 어울리는 알고리즘을 직접 개발하게 되었다.

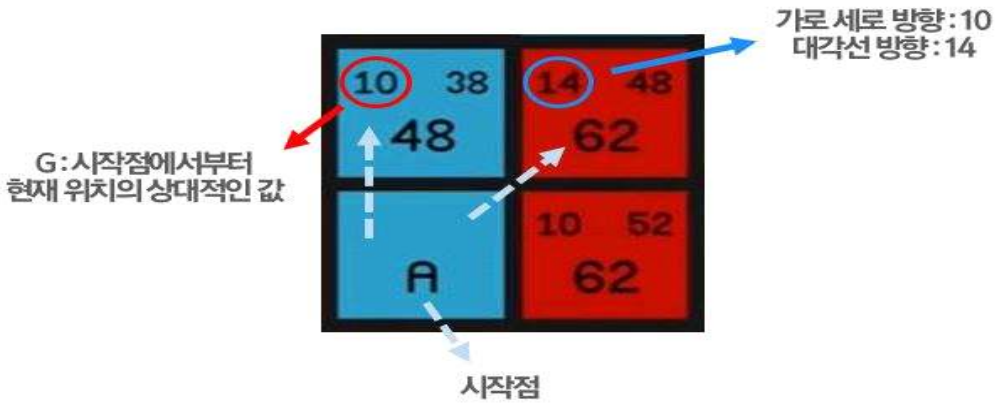
바로 A* 알고리즘을 이용해 길 찾기 알고리즘을 구현하였다. A* 알고리즘이란 주어진 출발 꼭짓점 부분에서부터 목표 꼭짓점까지 가는 최단 경로를 찾아내는 graph/tree 탐색 알고리즘 중 하나이다. 이 알고리즘은 각 꼭짓점 x에 대해 그 꼭짓점을 통과하는 최상의 경로를 추정하는 순위값인 "휴리스틱 추정 값 $h(x)$"를 매기는 방법을 쓴다. A* 알고리즘을 구현하는 방식에는 두 가지 방식이 있는데, Waypoint 방식으로 구현하는 방법과 Grid 방식으로 구현하는 방법이 있다.



[그림 13] Waypoint 방식

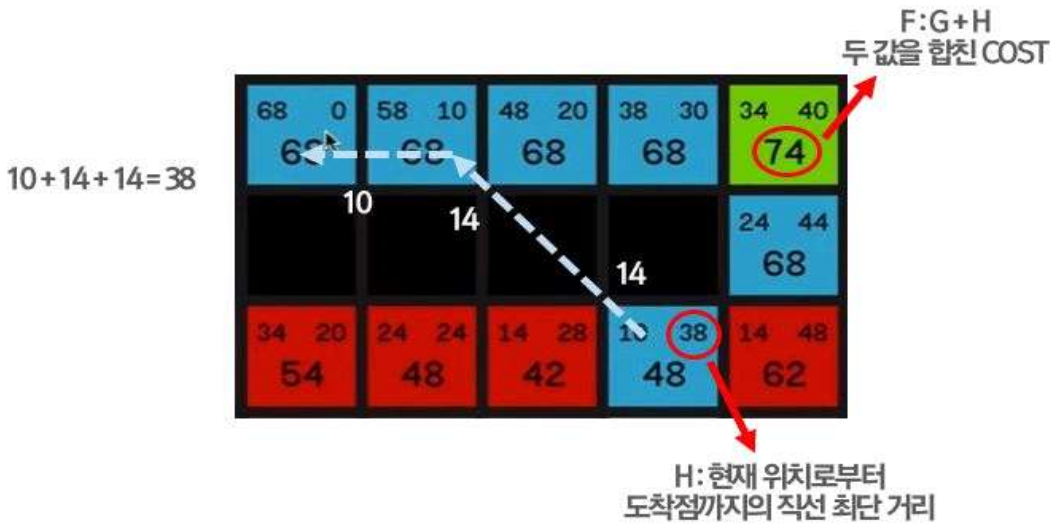
[그림 13]에서 간단하게 보여주고 있는 Waypoint 방식은 최종 도착점을 설정 한 후, 일정한 지점에 Waypoint를 설정해 해당 Waypoint를 따라 길을 찾는 방법이다. Waypoint 방식은 각 노드를 point 별로 하나하나 설정해 주는 방식이기 때문에 일일이 설정해주어야 한다는 번거로움과 캐릭터의 크기에 따라서 각각의 오브젝트에

플레이어와 적 유닛이 걸리는 일이 간혹 발생한다. 반면에 Grid 방식의 경우 맵 전체를 매핑하는 방식이기 때문에 플레이어나 적 유닛이 걸릴 일이 거의 없다. Grid 방식을 이용해 구현하는 것이 본 게임에 더욱 어울리는 방식이기 때문에 Grid 방식을 이용해 길 찾기 알고리즘을 구현하였다.



[그림 14] Grid 방식 설명 1

Grid 방식의 A* 알고리즘은 맵 전체를 격자의 형태로 만든 뒤(Grid Layout) 각 격자를 각각의 노드로 생각한다. 기존의 언급한 "휴리스틱 추정 값 $h(x)$"을 찾아내기 위해, 맨하탄 거리(Manhattan Length)라고 하는 간단한 방법을 이용해 만들어 냈다. [그림 14]에서 보여주는 G cost는 시작점에서부터의 현재 위치의 상대적인 값으로, 현재 타일까지의 이동 비용을 나타낸다.



[그림 15] Grid 방식 설명 2

[그림 15]에서 보여주고 있는 H cost는 현재 위치로부터 도착점까지의 직선 최단 거리를 나타내는 것으로, 현재 타일에서 목적 지점 타일까지의 이동 비용을 뜻한다. 동서남북 방향으로 이동 할 시에는 cost 10을 더해주고, 대각선 방향으로 이동할 때에는 cost 14를 주었다. F cost는 G cost와 H cost를 합친 값이다. [그림 15]에서, H cost가 38인 타일은 도착점까지 대각선 이동 두 번과 평행 이동 한 번을 통해 10+14+14 = 38의 결과를 보여주고 있다.

[그림 16] Grid 방식 설명 3

이러한 Grid를 이용해 길을 찾는 방법은 먼저, 주변의 타일들을 리스트에 넣고 정렬을 한다. cost가 가장 작은 타일들은 [그림 16]의 하늘색 타일들이다. 해당 타일에 도착하면, 다시 주변 타일들을 리스트에 넣어 정렬한다. 가끔씩 cost가 동일한 여러 타일이 나오는 경우가 있는데, 그런 경우에는 무작위로 한 타일을 선택하거나 가장 최근에 찾아낸 타일 하나를 선정하여 다시 리스트에 넣어준다. 그 후, 인접한 주변의 타일들을 다시 리스트에 넣고 정렬하는 과정을 반복한다. 이 과정을 목적지 타일에 도착할 때 까지 반복하면 전체 타일의 cost를 구해 낼 수 있다. 모든 타일이 정렬된 상태에서, 목적지부터 시작해 출발지까지 역추적을 통해 길을 다시 찾아가면 최단 경로를 찾아낼 수 있다.

이 과정을 프로그램 코드로 작성한다면, 다음과 같은 순서로 작성할 수 있다.

- 시작 지점에서 인접한 타일을 OpenList에 추가한다.
- 위 과정을 반복해 모든 인접한 타일을 OpenList에 추가한다.
- OpenList에서 가장 낮은 F cost를 가진 타일을 현재 노드로 설정한다.
- 해당 노드를 OpenList에서 꺼낸 뒤 ClosedList에 넣는다.
- 인접한 타일이 ClosedList에 있거나, 장애물이면 무시하고 넘어가고, OpenList에 없는 타일이라면 추가한 뒤 부모 노드를 현재 노드로 변경해 준 뒤 모든 cost를 계산한다.
- OpenList에 이미 있다면, G cost를 비교하여 더 작은 값을 가진 타일을 찾아 이동하고, 해당 타일을 부모 노드로 만든 뒤 모든 cost를 계산한다.
- 목표 노드를 OpenList에 넣었을 때, OpenList가 비어있다면, 최종 도착한 것이다.
- 최종 노드로부터 시작 노드까지를 역순으로 반환해 주어 최단거리를 찾는다.

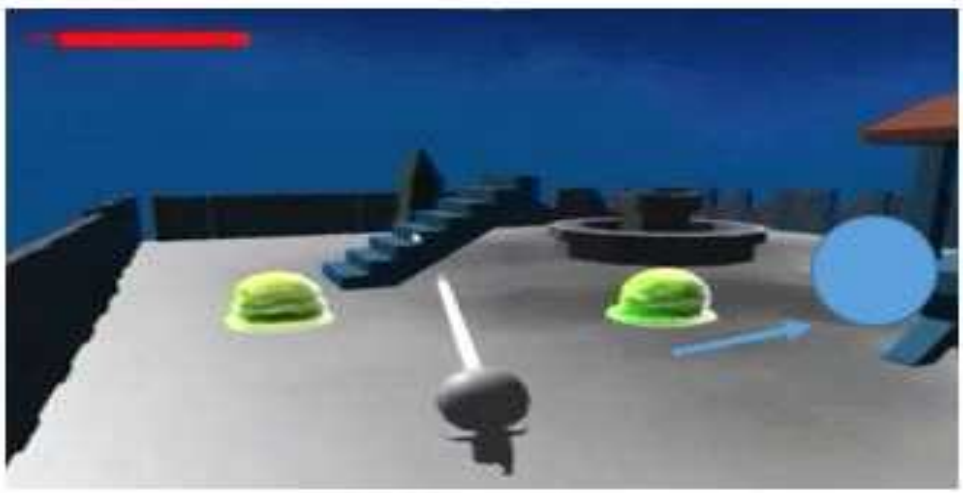
Grid 방식으로 구현한 A* 알고리즘을 사용할 시, 각각의 최종 cost를 우선순위 큐에 넣어 최초 정렬 후 계속 이용하기 때문에 평균적인 시간 복잡도가 $O(n)$이 되어 계산 시간이 빨라진다. 그래서, 다른 길 찾기 알고리즘과 비교했을 때, 장애물을 피해 가는 것이 수월하므로 개발하고자 한 게임에 매우 적합한 알고리즘이다. 또한, 맵의 모양에 영향을 받지 않는다. Grid 위에 장애물만 적용해 주면 어디서든 사용이 가능하다. 그리고 기존의 NavMeshAgent 함수의 경우, Unity 3D에서 기본적으로 제공을 해 주는 함수이기 때문에 소스 코드를 수정해서 사용하기가 매우 어렵다. 직접 구현한 알고리즘인 덕분에 여러 가지 추가 기능을 설정하기가 매우 용이하였다.

3) Hiding 알고리즘

A* 알고리즘은 길 찾기 알고리즘으로 게임을 부드럽게 진행되도록 하며, 게임을 구성하는 데 있어 필수불가결한 알고리즘이었다. 반면에, Hiding 알고리즘은 게임성을 높이기 위한 알고리즘이다. 이와 같은 알고리즘을 구현해 게임에 추가시켰기 때문에 1인칭 스테이지 모드의 다양성이 증가된다.

[그림 17] A* 알고리즘을 통해 캐릭터를 따라오는 모습

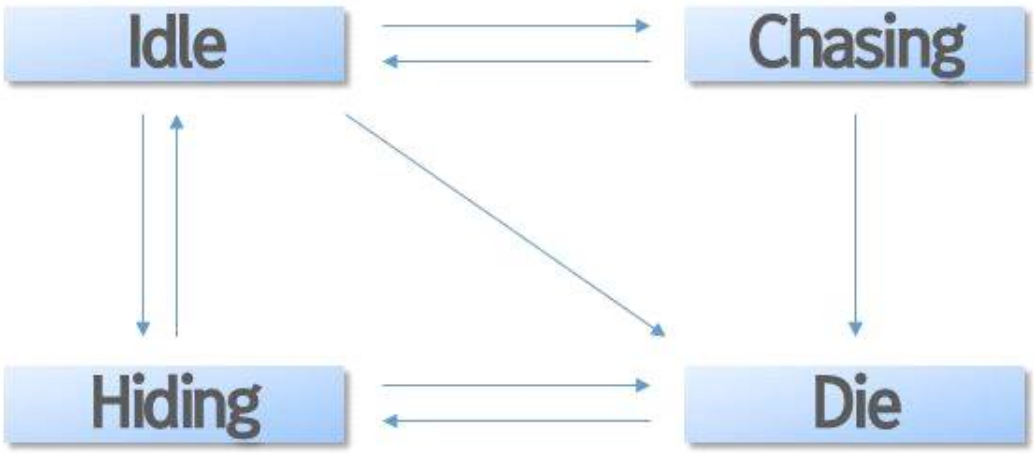
[그림 17]은 적 유닛이 나타났을 때, 적 유닛이 A* 알고리즘을 통해 캐릭터 유닛을 계속해서 쫓아오는 모습을 나타내고 있다. 그러나 플레이어가 눈덩이를 계속해서 던져 적 유닛의 체력을 일정 수준 이하로 만들 경우, Hiding 알고리즘이 작동해 적 유닛의 움직임을 바꾼다.



[그림 18] Hiding 알고리즘이 발동한 모습

[그림 18]에 보이는 파란색 원은 Hiding Point이다. 플레이어가 던진 눈덩이에 맞아 적 유닛의 체력이 일정 수준 이하로 떨어질 시, A* 알고리즘을 취소하고, Hiding 알고리즘을 동작시켜 지정된 Hiding Point로 빠르게 이동해 숨는다.

4) 적 유닛 AI 모델링



[그림 19] 적 유닛의 AI FSM 차트

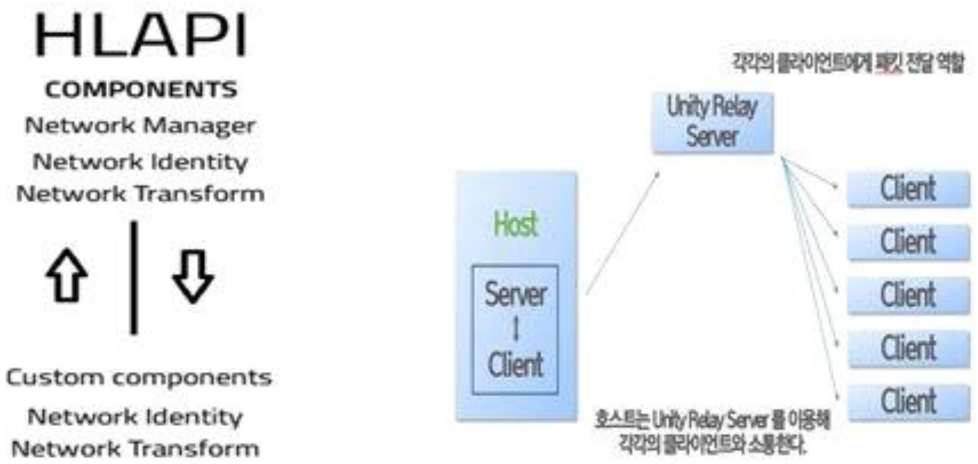
적 유닛의 AI FSM 차트의 경우, 플레이어 캐릭터의 FSM 차트와 비슷한 형태로 구성되어 있으나, 세부적인 내용에서 차이점을 보인다. 아무것도 하지 않는 상태인 Idle 상태는 적 유닛의 체력이 0 이상인 상태를 기본 설정으로 한다. 이후, 플레이어가 감지되면 플레이어 캐릭터의 위치를 최종 도착점으로 설정하여 Chasing을 한다. Chasing을 할 때 필요한 알고리즘이 A* 알고리즘이다. 초당 1회씩 플레이어 캐릭터의 위치를 요청하여 배열에 넣어 A* 알고리즘을 이용해 플레이어 캐릭터로 향하는 길을 찾게 된다.

플레이어 캐릭터의 위치를 찾아 최종 목적지에 도착하는 경우에는, 플레이어 캐릭터 객체와 적 유닛 객체 간의 충돌이 일어나게 된다. 해당 상태를 Collision으로 두어, 적 유닛의 중심점과 플레이어 캐릭터의 중심점이 거리보다 작아질 때로 설정한다. 이 경우에, 플레이어 캐릭터의 체력이 0 이상, 즉 플레이어 캐릭터가 Alive 상태라면 플레이어 캐릭터를 공격해 체력을 깎는다.

플레이어가 적 유닛을 발견하면 공격을 하는데, 적 유닛이 눈덩이에 맞아 체력이 50 이하로 떨어진다면, 이 때 Chasing 상태에서 Idle 상태로 전환 후 Hiding 상태로 변경해 플레이어 캐릭터에게서 멀리 떨어져 Hiding Point에 숨어 체력을 회복한 후 다시 Chasing을 시작한다. 적 유닛의 체력이 0이 되면 Die 상태가 되어 객체가 소멸한다.

4. TCP / IP 통신

이전 소주제로 설명한 AI 관련 부분은 캐릭터의 동작을 제외하면 PvP 플레이를 통한 경쟁 모드에서는 필요가 없다. 하지만, 적 유닛 대신 상대방과 게임을 플레이하기 위해서는 게임 내 통신이 구현되어야 한다. Unity 3D를 이용해 게임을 개발했으므로, Unity 3D 5.1 이상 버전에서 직접 제공하는 내부의 Network API인 HLAPI(High-Level Application Programming Interface)구조를 이용해 개발하였다.

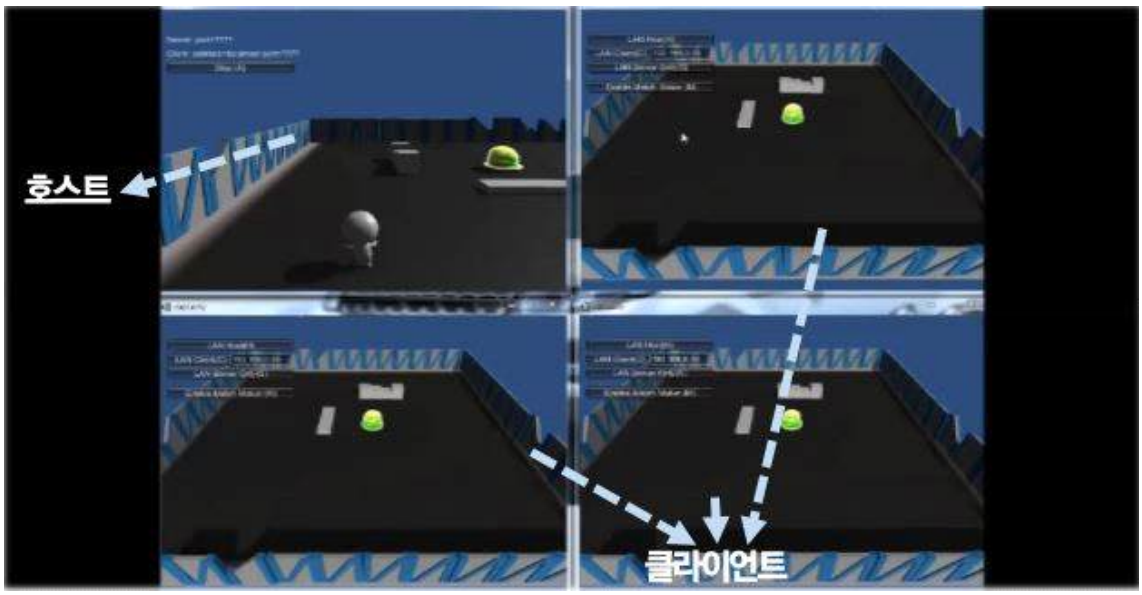


[그림 20] Unity Network의 구성

먼저, 서버는 Unity에서 제공하는 Unity Relay Server를 이용한다. 이 서버는 각각의 클라이언트에게 패킷을 대신 전달해주는 역할을 하고 있으며, 호스트가 Unity Relay Server를 이용해 각각의 클라이언트와 소통할 수 있게 된다.

HLAPI는 Unity 게임의 멀티플레이 기능을 구축하기 위해 필요한 시스템이다. 이 API는 저레벨 실시간 커뮤니케이션층의 상위층으로 구성되어 있으며, 멀티플레이에 필요한 수많은 작업들을 처리하는 데에 이용된다.

HLAPI는 UnityEngine.Networking 안에 있다. 사용하기 용이함과 상호연결성이 보장되는 개발에 초점이 맞춰져 있으며, 개발하고자 하는 게임의 통신 부분에서 가장 중요시 여기는 '상태 동기화' 기능에 유용한 도움을 제공해 준다.



[그림 21] 4인 모드 플레이 모습

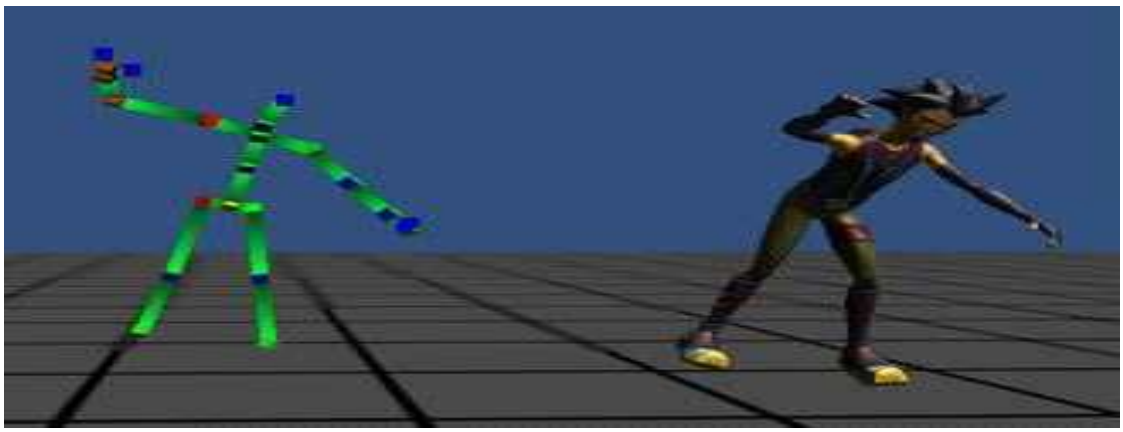
[그림 21]와 같이 호스트가 방을 설정한 뒤, 각 클라이언트들이 호스트의 IP를 이용해 게임에 접속해 멀티플레이를 할 수 있는 기본적인 틀이 완성된 모습이다. HLAPI에서 제공하는 함수 중 Network Transform 기능을 이용해 서버와 클라이언트 간의 동기화 문제를 해결할 수 있었다.

이 상태 동기화는 멀티플레이에 있어 가장 중요한 요소라고 할 수 있다. 상태 동기화가 되지 않으면 호스트의 움직임이 다른 클라이언트에게 보여지지 않거나, 수초 전의 동작이 클라이언트에게 늦게 보이는 등 여러가지 문제점이 발생해 정상적인 멀티플레이를 할 수 없으며 이 플레이 과정에서 개발 시 원했던 모습의 동작이 아닌 오류성 동작이 많이 일어나게 되기 때문이다.

5. Kinect 플레이 구현

앞서 언급했듯이, 최근에는 VR 기기를 연동하여 모션 인식 센서가 달린 장치를 이용한 FPS 게임들이 많은 인기를 끌고 있다. 위에서 플레이하는 방식은 모두 키보드와 마우스를 통한 일반적인 FPS의 모습이다. 따라서 Kinect를 이용한 모션 인식을 구현해 게임 플레이 시의 박진감을 향상시켰다.

Unity 3D로 제작한 게임 중, Kinect를 이용해 만든 재활치료 게임이 이미 시장에 많이 나와있다. Unity Asset Store에서 학생을 대상으로 MS에서 무료로 제공하는 Kinect v2 with MS SDK를 다운로드받아 추가해 Unity 3D 프로그램 내에서 직접 Kinect를 연동하여 사용하였다.



[그림 22] Kinect SDK 연동 시 기본적으로 제공하는 뼈대 및 캐릭터

Kinect에 관련된 함수는 Kinect 사에서 제공하는 Kinect Manager라는 인터페이스 안에 담겨있다. 게임에 연동하기 위해 각각의 함수를 원하는 동작에 맞추어 재구성하였다. [그림 22]에 기본적으로 제공하는 뼈대의 경우 해당 캐릭터에 맞춰 뼈대가 구성되어 있으므로, 본 게임의 캐릭터에 맞게 뼈대를 재구성하였다. 플레이어 캐릭터가 눈덩이를 던지는 동작을 구현하기 위해 손을 위에서 아래로 내리는 동작을 이용했으며, 깊이가 인식되므로 손을 얼마나 내리느냐에 따라 눈덩이가 발사되는 강도를 조절했다. 손의 기본 위치에서 손이 내려가면 내려갈수록 눈덩이가 가속도가 붙어 발사된다. 보통 화면에 비춰지는 유저의 모습이 상반신 정도이므로, 상하좌우 움직임은 키보드 입력으로 대체하였다.

3. 연구 및 개발 결과

1. 싱글 플레이 구현

[그림 23] 게임 로딩 화면

[그림 23]은 최종적인 게임의 시작 화면의 모습이다. 맨 위의 버튼은 1인칭 스테이지 모드를 실행시키는 버튼이며, 두 번째 버튼은 PvP 플레이를 실행시키기 위한 버튼, 마지막 버튼은 게임을 종료시키는 버튼이다.

[그림 24] 싱글 플레이 스테이지 1 화면

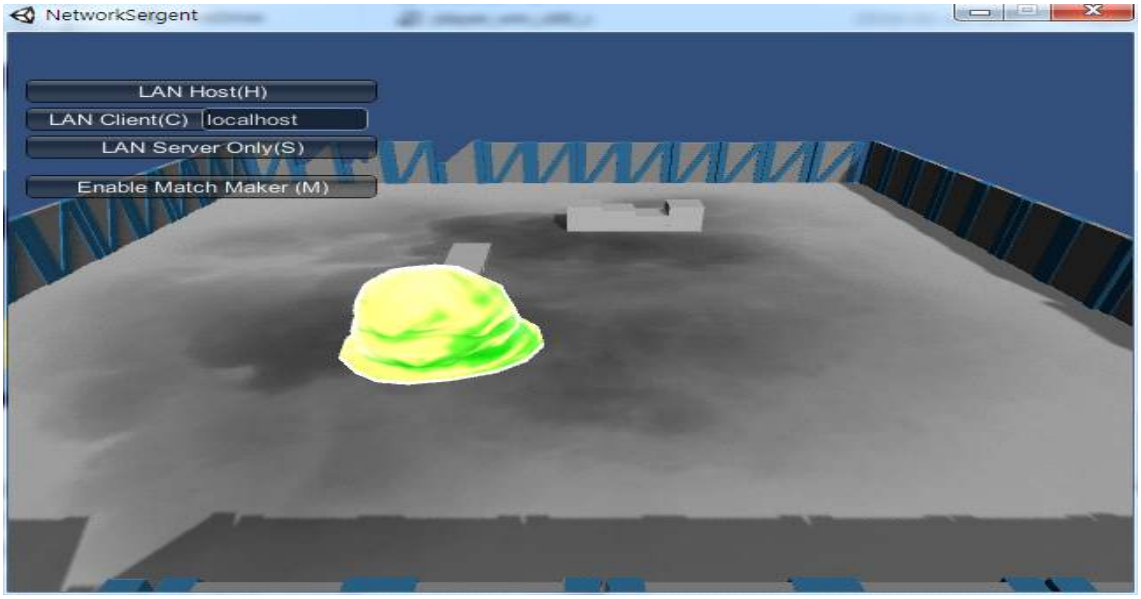
싱글 플레이 버튼을 클릭하면, [그림 24]와 같이 게임이 실행된다. 각 스테이지 별로 적 유닛의 위치가 정해져 있으며, 적용된 AI를 통해 움직이게 된다. 스테이지 클리어 조건을 모두 충족시키면, [그림 25]처럼 다음 스테이지로 넘어갈 수 있다.

[그림 25] 싱글 플레이 스테이지 2 화면

현재 보여지는 싱글 플레이는 3 스테이지까지 구현이 되어있으며, 3 스테이지까지 클리어하면 게임 플레이 시간을 카운트하여 점수를 매기는 방식으로 다른 사람들과 경쟁할 수 있도록 했다. 점수를 매기는 동시에, 게임이 끝나면 다시 [그림 24]의 로딩 화면으로 돌아가게 된다.

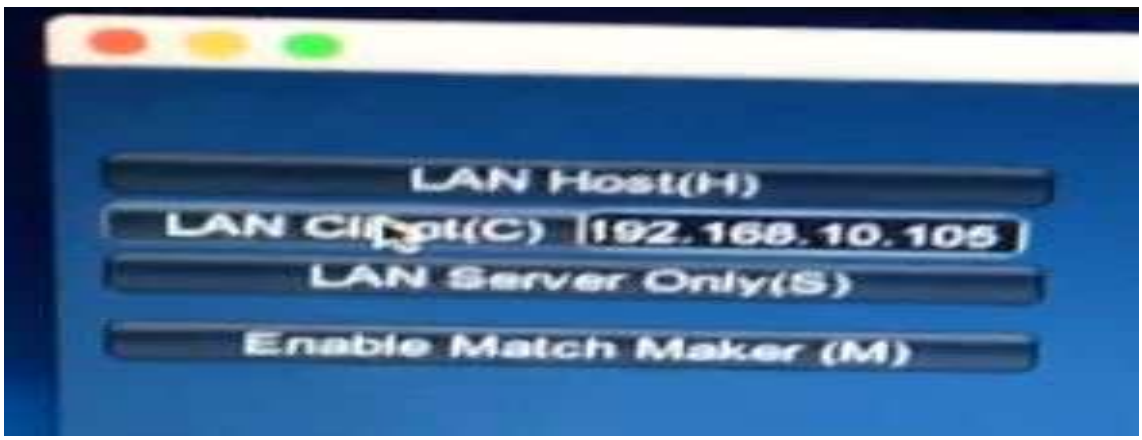
2. 멀티 플레이 구현

[그림 23]의 두 번째 버튼을 클릭하면, 멀티 플레이 게임을 실행할 수 있다.

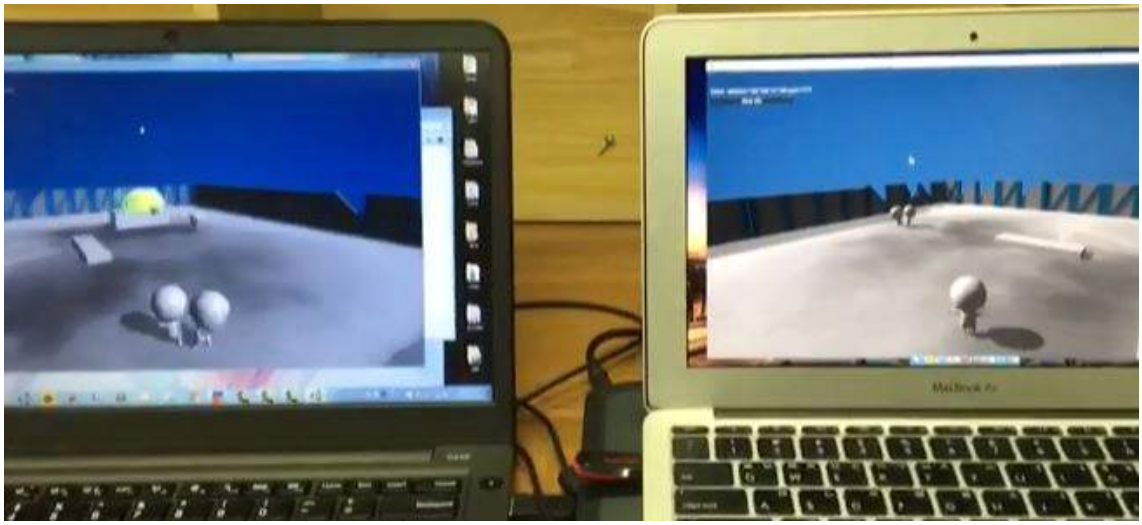


[그림 26] 멀티 플레이 실행 화면

버튼 클릭 시, [그림 26]과 같은 멀티 플레이 실행 화면이 나타난다. 제일 먼저 접속한 유저는 LAN Host 버튼을 눌러 방을 개설한다. 다른 유저들은 이미 방이 개설되었으므로 클라이언트가 되고, [그림 27]에서 보이는 것 처럼 두 번째 칸의 LAN Client의 입력 창에 Host의 IP 주소를 입력한 후 클릭하면 Host가 개설한 게임방에 들어갈 수 있게 된다.

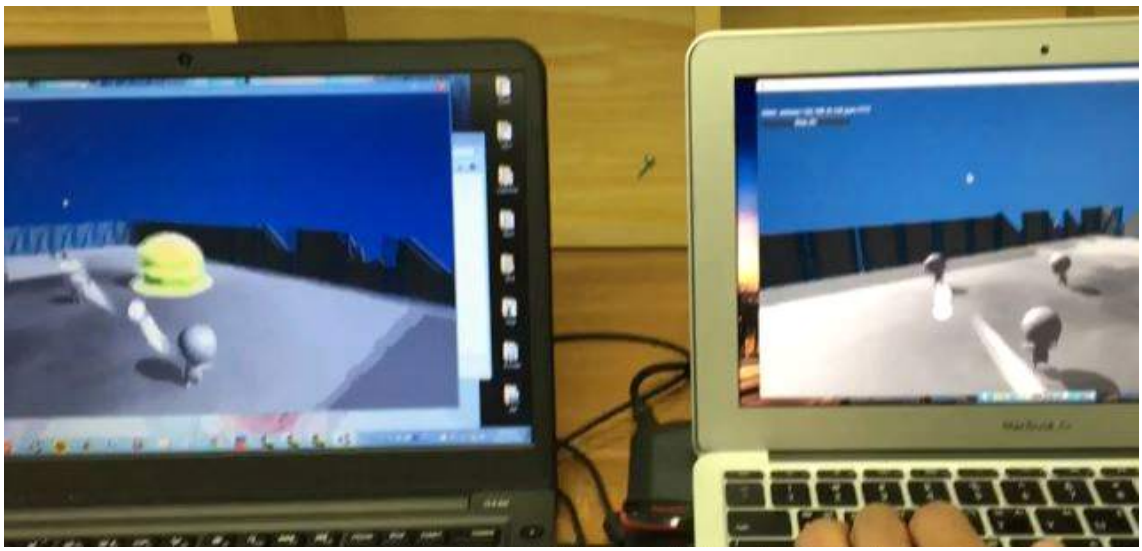


[그림 27] Client의 Host 방 입장 화면



[그림 28] 여러 유저가 접속한 모습

[그림 28]는 총 3명의 유저가 한 방에 접속한 모습을 보여주고 있다. 맵에 일정하게 움직이는 보스 몬스터가 있고, 보스 유닛을 공격해 해치우고 나면 상대방과 겨루는 모드가 시작된다. [그림 29]의 왼쪽 컴퓨터 화면과 오른쪽 컴퓨터 화면을 보면 세 유저가 각각 눈덩이를 던지는 모습이 동시에 보인다. 즉, 동기화된 화면을 보고 있다는 것을 알 수 있다.



[그림 29] 상대방에게 눈덩이를 던지는 모습

3. Kinect 플레이 구현

[그림 30] Kinect를 이용한 플레이

Kinect를 이용해 손이 위에서 아래로 내려가는 동작을 통해 마우스를 사용하지 않고도 플레이어 캐릭터가 눈덩이를 발사하도록 구현하였다. [그림 30]의 왼쪽 화면은 손을 위에서 아래로 내리는 동작을 하고 있는 모습을 담았고, 오른쪽 화면은 컴퓨터에 보이는 게임 화면을 크게 확대해 붙인 것이다. 마우스를 사용하지 않고 키넥트를 이용해 타겟을 맞추는 것은 즉, 에임의 정확성이 필요하다. 그러나 키넥트를 이용해 발사 타점 변환을 정확히 하는 것은 세밀한 조작이 없는 이상 매우 어렵다.

그래서 에임을 화면 중앙에 배치 한 뒤, 에임 부근에 큰 원 모양의 팬을 설정하여 맞추려는 적 유닛이 해당 팬 안에 위치하고 있으면 데미지를 입는 방식을 채택해 난이도를 낮췄다.

[그림 30]의 오른쪽 화면에 보이는 Kinect 뼈대는 사용자의 동작을 확인하기 위해 넣은 것으로 사용자의 동작을 입력받아 아주 미세한 입력 시간 오차를 가지고 똑같이 움직인다. 플레이어 캐릭터와의 연동이 잘 되고 있는지를 확인하기 위해 Kinect 플레이 시 화면에 뼈대를 적용하였다.

3. 향후 연구

1. 향후 개발 내용

1) 디자인 부분

기존의 게임은 자체적으로 만든 캐릭터와 맵 등을 이용해 구현하였다. 그로 인해 디자인 측면에서 봤을 때 조잡해 보이는 경우가 있다. Unity Asset Store에서 제공하는 다른 세련된 디자인의 맵과 오브젝트를 다운로드받아 적용시키고, 충분한 시간을 가지고 캐릭터를 디자인해 전체적인 게임 분위기를 업그레이드 할 예정.

2) 스토리 추가

흥행하는 게임의 대부분의 공통점은 그 게임만의 특이한 스토리가 있다는 점이다. 플레이어 캐릭터에 의미를 부여하기 위해 왜 플레이어 캐릭터가 이 게임 내에서 눈싸움을 하고 있는지, 그리고 각 스테이지마다 스토리 전개에 맞는 컨셉트를 적용해 게임을 플레이하는 데에 있어 지루함을 없앨 예정.

3) 부가기능 추가

현재까지 개발된 내용에는 기본적인 눈덩이를 던지는 기능만이 구현되어 있다. 앞으로 더 추가하려는 기능은 상대방 또는 적 유닛의 위치를 알려주는 미니맵을 구현하고 눈덩이를 제외하고, 미니맵을 이용해 적 유닛을 향해 공중에서 날릴 수 있는 눈사태 미사일, 상대방을 속박하기 위한 지뢰 등의 많은 종류의 무기 추가, 고정된 장애물이 아닌 움직이는 장애물, 플레이어가 빠지면 게임이 끝나게 되는 크레바스 등 게임성을 증가시킬 수 있는 여러 요소를 삽입할 예정.

위에 언급한 3가지 이외에도 Kinect 플레이 부분에 있어서 반응속도를 올리는 법과 좀 더 다양한 컨트롤러를 이용해 플레이 할 수 있는 기능들을 추가하고, 핸드폰 터치 방식으로 동작할 수 있도록 안드로이드를 접목시켜 Google Play Store 등 여러 어플리케이션 시장에 출시할 계획이다.

<참고 문헌>

1) '만들면서 배우는 유니티 게임 프로그래밍' - 송용성

2) '키넥트 프로그래밍' - 자렛 웹, 제임스 애쉴리

3) Kinect Manager – 키넥트 프로그램 자체 제공

4) 'C# 코딩의 기술' - 가와마타 아키라

5) 'C# 코딩의 기술 실전편' - 가와마타 아키라

6) opentutorials.org – 통신 부분 참고 2016.09.21

7) tutorialspoint.com – Kinect, 통신 부분 참고 2016.09.22

8) '3D 캐릭터 모델링의 정석' - Chris Legaspi

9) https://docs.unity3d.com/kr/current/Manual/UNetUsingHLAPI.html

- Unity 3D 통신 부분 기본 제공 매뉴얼 2016.10.15

10) 위키백과 (A* 알고리즘 관련 내용) 2016.10.04.

11) http://cafe.naver.com/gogoomas/33317 - A* 알고리즘 관련 2016.10.06.

12) http://blog.naver.com/jeonys1231/220809452069 - Unity 3D 관련 2016.09.27.

13) 'Unity로 배우는 실전 게임 개발' - 박승제

14) http://blog.naver.com/monkey5255/220823143237 - 서버 관련 2016.10.02.

15) http://blog.naver.com/gngh0101/220826114409 - 서버 관련 2016.10.02