길 찾기(A*) 알고리즘을 응용한 유니티3D 네트워크 3인칭 슈팅게임 개발

Application of the A* algorithm, network, third-person shooter developed

한국외국어대학교
공과대학
디지털정보공학과

이현범

2016年 10月

길 찾기(A*) 알고리즘을 응용한 유니티3D 네트워크 3인칭 슈팅게임 개발

Application of the A* algorithm, network, third-person shooter developed

위 논문 학사학위 논문으로 제출합니다.

지도교수: Ho Yub Jung

2016年 10月

대학 : 한국외국어대학교

학과 : 디지털정보공학과

학번 : 201202868

이름 : 이현범

이현범의 학사학위 논문을 심사하여

합격으로 판정합니다.

심사위원: (인)

길 찾기(A*) 알고리즘을 응용한 네트워크 3인칭 슈팅게임 개발

Application of the A* algorithm, network, third-person shooter developed

요약

Abstract

게임AI는 게임에서 없어서는 안 될 필수 요소가 되었다. 그 중 길 찾기 알고리즘은 많은 게임들에서 사용되는 알고리즘이고, 보편적으로 사용되는 두 가지 방식이 존재한다. 또한 유니티에서는 길 찾기 알고리즘을 쉽게 제작할 수 있도록 제공해주는 기능이 있다. 하지만 실제 게임을 제작할 때, 이 기능에 문제점이 발생하고, 개발하는 게임마다 적용하기 힘든 점이 있다. 따라서 유니티에서 제공하는 기능을 사용하지 않고, 직접 길 찾기 알고리즘을 개발하여 사용하고, 게임AI를 개발하고, FSM을 사용하여 알고리즘을 구현하여 게임을 개발한다. 게임의 재미를 증가 시키기 위해서 네트워크를 이용해서 다수의 사용자가 게임을 하고 동시에 게임을 즐길 수 있도록 개발한다.

목차

1. 서론

1.1 개발 동기

게임 산업은 점차 고성능화 되는 PC, 콘솔 등의 플랫폼과 함께 많은 발전을 이룩해오고 있는 반면, 스마트폰이라는 새로운 플랫폼의 등장과 함께 새로운 영역으로 발돋움 하고 있다. 스마트폰 게임 중 유니티로 개발된 게임은 2016년 기준으로 2분기에 23만 개를 기록하고 계속된 성장세를 보이고 있다. 게임 설치 건수는 44억 건을 기록했다.

그림 [1]

그림[1]은 지금까지 유니티로 개발된 게임들 중 많은 인기를 가진 게임들이다. 이 게임들은 유니티를 이용해 쉽게 개발할 수 있고, 간단한 플레이를 요구하지만 엄청난 인기를 이끌고, 상당한 이익을 달성했다. 이처럼 많은 게임들이 유니티로 개발되었고, 유니티는 게임개발에 있어서 가장 좋은 도구 중 하나이다. 따라서 유니티를 이용하여 모바일과 PC에서 즐길 수 있는 게임을 개발하게 되었다.

게임 개발에는 디자인, 프로그래밍, 그래픽 등 많은 분야가 있다. 하지만 게임에서 가장 중요한 역할을 담당하고 게임의 재미를 증폭시킬 수 있는 분야는 게임AI이다. 예전 게임업계에서는 사용자 경험을 향상시키기 위해, 주로 게임의 그래픽을 발전시켜왔지만, 그에 따른 제작비의 상승과 같은 이유로 그래픽을 통해

다른 게임과 차별화 시키는 것이 점차 어려워지고 있다. 반면 최신 인공지능 기법을 적용하여 사용자에게 더 새롭고 차별화된 재미를 제공하는 것이 새로운 대안으로 관심을 모으고 있다. 게임에서 게임AI의 역할은 함께 경쟁할 수 있는 개체를 제공해 재미를 만들어내고 흥미를 유발하는 NPC(non-player characters)들을 제공해 게임에 현실감을 불어 넣는다.

기술적으로는 실제 인간과 근사하게 개발되고 있지만, 게임에서는 그 정도의 수준을 요구하진 않는다. 게임에서의 인공지능은 최대한 효율적이어야 한다. 게임AI를 구현하는 방법에서 여러 가지가 있지만, 그 중 유한 상태 기계(FSM, Finite State Machine)을 이용하여 단순한 형태의 인공지능 모델을 제작 한다.

유니티는 유연한 엔진으로 인공지능 패턴을 구현하는 다양한 방법을 제공한다. 그 중 일부는 쉽게 적용할 수 있는 것도 있지만, 개발되는 게임에 따라서 불필요한 조건이 포함된 경우도 있다.[4] 따라서 유니티를 최대한 활용하면서 불필요한 부분은 제거하고, 필요한 부분을 개발하여 게임을 개발한다.

1.2 개발 주제

그림 [2]

그림[2]처럼 3인칭 시점 게임인 TPS(Third-Person Shooter)를 유니티로 개발한다. 플레이어는 키보드와 키넥트 중 하나를 선택해서 플레이를 하게 된다. 플레이어가 상대하는 NPC를 개발하여 게임을 진행한다. NPC에는 게임AI를 적용하여 실시간으로 플레이어의 상태를 확인하면서 상황에 맞는 행동을 하고, 게임의 난이도를 정할 수 있게 개발한다.

길 찾기 알고리즘을 주축으로 하여 알고리즘을 개발한다. 유니티에서 제공되는 길 찾기 알고리즘을 이용하지 않고, 게임에 최적화된 알고리즘을 개발하여 사용한다. 네트워크를 통해서 여러 플레이어가 함께 플레이를 할 수 있도록 한다. 게임의 배경은 게임에서 흔히 발생하는 폭력성을 줄이기 위해 캐주얼한 분위기의 눈싸움으로 정하였다.

현재 개발된 게임은 PC에서만 구현이 되는 상태이고, 추후에 플레이 방법을 터치로 개발하여 스마트폰 게임으로 제작할 예정에 있다.

1.3 개발목표

본 게임은 게임AI와 두 가지 컨트롤러 그리고 네트워크가 주요 개발 부분이다. 게임에서 사용되는 맵과 지형지물 그리고 게임 객체들은 Unity3D와 그래픽 개발도구인 3DMAX를 이용하여 제작한다. 유니티에서 제공되는 게임AI를 사용하고, 필요한 게임AI는 개발하여 사용한다.

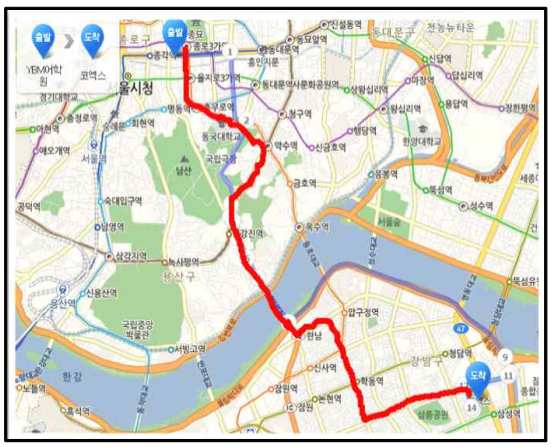

그림 [3]

그림 [4]

길 찾기 알고리즘은 현재 위치에서 목표까지의 최단거리를 구하는 알고리즘이다. 주로 그림[3]과 같은 내비게이션에 사용되는 알고리즘이지만, 게임에 적용되는 사례가 많이 있다. 그림[4]처럼 유명한게임인 스타크래프트에도 적용되어 있다. 길 찾기 알고리즘이 적용되어 객체들이 줄지어서 같은 지점으로 가는 모습을 흔히 볼 수 있다. 유니티에서는 길 찾기 알고리즘을 제공하지만, 최적화를 하는데 여러 가지 문제가 발생했다. 따라서 직접 게임에 최적화된 길 찾기 알고리즘을 개발하여 NPC에 적용한다. 한 객체에 여러 가지 알고리즘을 적용하고 사용하는 방법을 구상하고, 유한 상태 기계 알고리즘을 개발한다. 컨트롤러는 키보드와 키넥

트를 사용할 수 있도록 개발한다. 또한 유니티에서 제공하는 네트워크기술을 활용하여 여러 플레이어가 함께 게임을 진행할 수 있도록 개발한다.

2. 연구 배경

2.1 길 찾기 알고리즘

많은 게임에서 몬스터나 적들이 플레이어를 따라다니거나 장애물을 피해가며 특정 지점으로 이동하는 모습을 볼 수 있다. 예를 들어 실시간 전략 게임의 경우 일정 그룹을 선택한 후 특정 위치를 클릭해서 이동시키거나 적을 클릭해서 공격하도록 한다. 그러면 장애물과 충돌을 하지 않고 목적지에 도착하기 위해서는 경로를 탐색해야 한다. 이때, 사용되는 알고리즘이 길 찾기 알고리즘이다. 그 중 A*(에이스타) 길 찾기 알고리즘을 개발한다.

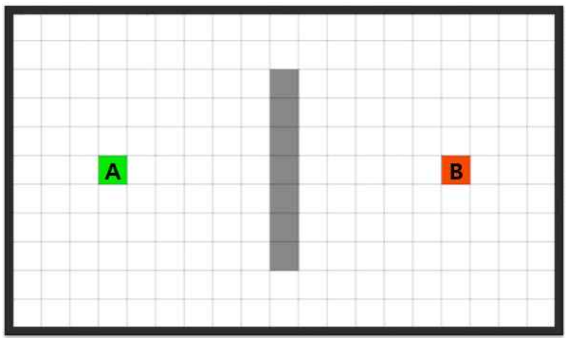

그림 [5]

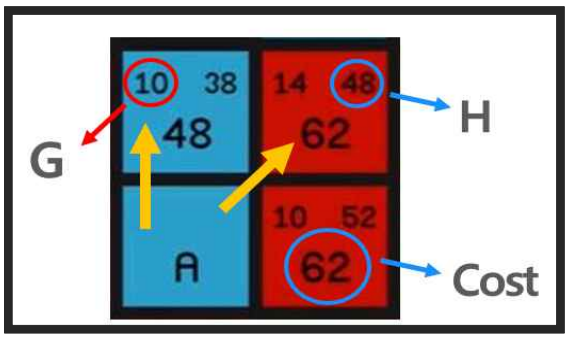

그림 [6]

그림[5]에서 A에서 B로 가는 경로를 찾기 위해서는 장애물의 위치 등과 같은 정보에 대해 알아야한다. 이를 위해서 전체 맵을 사각형 격자들(Grid)로 나눠서 사용한다. 전체 맵을 격자 형태로 구성하게 되면 탐색 지역을 좀 더 단순화 하고 계산하기가 용이해서 길 찾기에 큰 도움이 된다. 이런 방법을 통해서 검색지역을 단순한 2차원 배열로 만들 수 있게 된다. 탐색 지역에 상태는 갈수 있는 곳, 갈수 없는 곳, 시작 지점, 도착 지점으로 구성 된다.

최단거리를 구하는 방법은 그림[6]처럼 A의 주변의 타일에 대해서 계산을 하게 된다. 여기서 G, H, Cost의 값을 비용이라고 부른다. G는 시작점에서부터 해당 지점까지의 상대적인 비용이다. 가로, 세로 방향의 비용은 10이고, 대각선 방향의 비용은 14이다. 대각선 방향에 비용이 14인 이유는 격자가 정사각형이고, 이등

변 삼각형 길이의 비 $\sqrt{2}:1:1$에 의해서 $\sqrt{2}$(약 1.414) 가된다. 따라서 대각선 방향의 비용을 14로 정한다. H는 해당 지점에서 도착점까지의 예상비용이다. 이 비용을 계산할 때는 장애물을 무시하고 계산을 한다. Cost는 G의 비용과 H의 비용을 더한 비용이다. 이 Cost를 비교하여 최단경로를 구하게 된다. 각 값에 대해서 배열을 만들어 사용하게 된다.

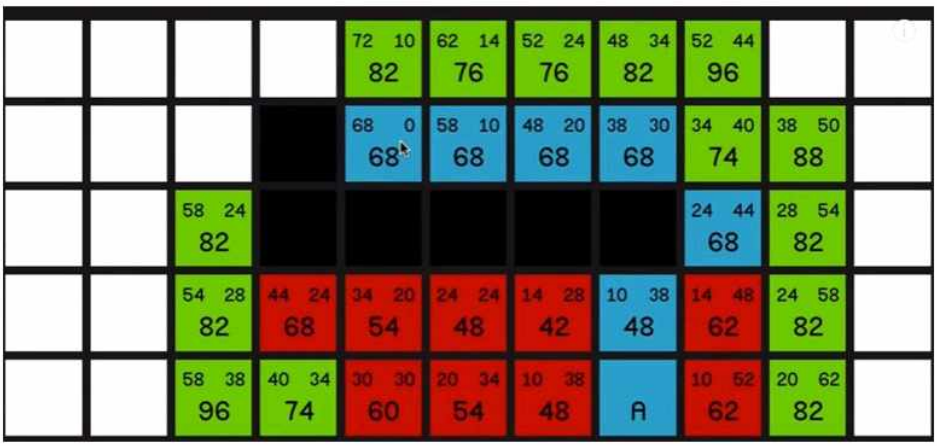

그림 [7]

그림[7]에서 A칸의 주변에서 Cost가 가장 작은 값은 48이다. 그 다음 Cost가 48인 칸의 주변 Cost를 구한다. 그 중 작은 값은 선택하고, 같은 방법으로 최종 목적지 까지를 구하게 된다. 그림에서 초록색 칸은 Cost를 비교했을 때, 도착점을 찾지 못한 경우이다. 따라서 그림에서 파란색 칸들이 최단거리를 나타낸다.

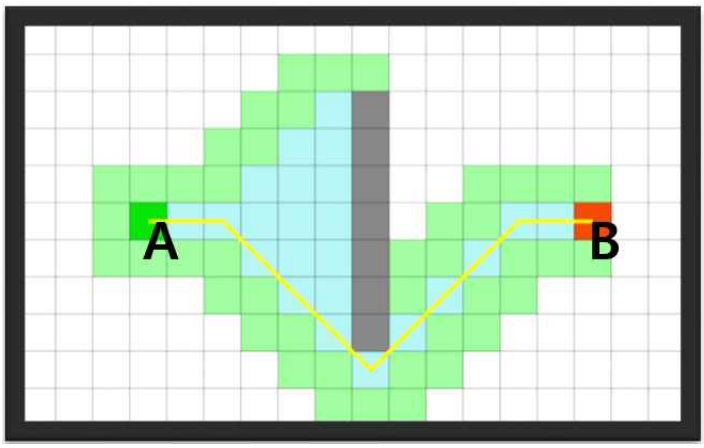

그림 [8]

그림[8]은 그림[5]에서 Grid방식을 적용하여 최단거리를 구한 그림이다. 이 방식을 이용하여 실시간 계산을 통해 적NPC가 이동할 최단거리를 구하게 된다.

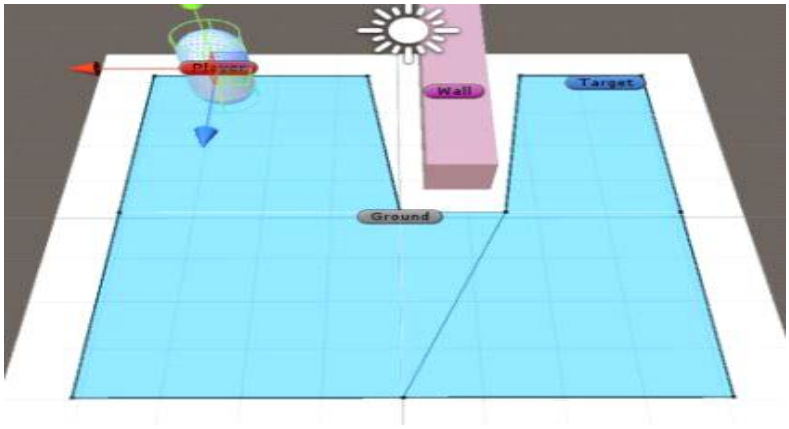

그림 [9]

A*는 길 찾기에서 중요한 패턴이지만 유니티에는 바로 사용할 수 있는 자동 내비게이션 메시 생성과 NavMesh 에이전트를 사용할 수 있다. 그림[9]은 내비게이션 메시를 생성한 모습이다. 하지만 NavMesh는 제공 되는 기능이기 때문에 모든 게임에 적용이 어려운 점이 있다. 따라서 A* 길 찾기 알고리즘을 직접 구현해서 사용해야 할 경우가 많이 있다.

내비게이션 메시는 사각형 격자(Grid) 방식이 아닌 웨이포인트(Waypoint) 방식을 사용한다. 웨이포인트 방식은 맵을 생성할 때, 특정 포인트를 미리 지정하는데 이 포인트를 웨이포인트라고 부르고, 이 웨이포인트를 이용하여 최단거리를 이동하는 방식이다.

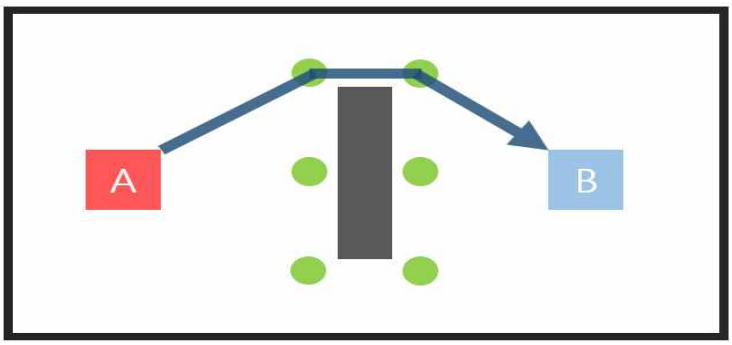

그림 [10]

그림[10]에서 초록색 점이 웨이포인트이다. 목적지 B의 방향으로 가장 가까운 포인트가 선택이 되고, 그 다음으로 가까운 포인트를 선택한다. 상당수의 게임이 연산 효율성을 높이기 위해 간단한 처리에는 웨이포인트 방식이 적극 활용되

고 있다. 하지만 이 방식을 사용하면 맵에 장애물이 추가될 때마다 웨이포인트를 갱신해야 한다. 이 알고리즘이 적용된 NPC는 각 노드를 직선으로 연결하는 경로를 따라 차례로 이동하면 목적지에 도달할 수 있다. 하지만 실제 게임을 제작할 때, 경로가 장애물에 가까이 있을 경우 장애물에 걸려서 움직이지 못하는 현상이 자주 발생한다. 이를 완화하기 위해서 경로를 곡선으로 구성한다고 해도 근본적인 문제가 해결되지는 않는다.

그림 [11]

그림[11]은 내비게이션 매쉬를 이용하여 움직이는 목표에 대한 최단거리를 구하고 그 길을 따라 움직이는 NPC의 모습인데, 내비게이션 매쉬가 생성될 때 일정 지역을 Bake(유니티에서 맵의 정보를 읽어오는 방법)를 하게 되는데, Bake가 되지 않은 부분인 이동 불가 지역에 들어가게 되면 그림처럼 NPC가 움직일 수 없게 된다. 유니티에서 제공하는 내비게이션 매쉬는 장점이 많지만, 실제 게임에서 이러한 버그가 자주 발생하고, 제공되는 기능이기 때문에 수정하기가 힘들다.

유니티에서 제공하는 내비게이션 매쉬는 위의 문제점을 해결하기 위해서 볼록한 폴리곤을 사용한다. 그리고 맵 전체의 정보를 얻어오게 된다. 하지만 모든 맵에 따른 내비게이션 매쉬를 생성하는 과정은 다소 복잡하고, 게임성능 개선에 불필요한 연산이 생기게 된다. 따라서 내비게이션 매쉬는 간단한 길 찾기 또는 작은 지역에 사용하면 개발하는데 시간이 많이 절약되지만 길 찾기 알고리즘이 실시간으로 적용되는 게임에서는 사용하는데 어려움이 있다.

2.2 유한 상태 기계 알고리즘

유한 상태 기계(FSM, Finite State Machine)는 주어지는 모든 시간에서 처해 있을 수 있는 유한개의 상태를 가지고 주어지는 입력에 따라 어떤 상태에서 다른 상태로 전환하는 것이다. 따라서 특정한 상태를 정의하기 위한 개념적인 모델이다. 보통 프로그램을 작성할 때, if문과 같은 조건문을 자주 사용하게 된다. 하지만 게임 프로그램은 여러 가지 상태가 존재하고, 그 상태를 if문으로 작성할 경우 계산에 많은 부담이 있고, 오류가 생길 경우 수정하는데 많은 어려움이 생긴다. 따라서 코드의 직관성과 유연성을 위해서 FSM을 사용하게 된다.

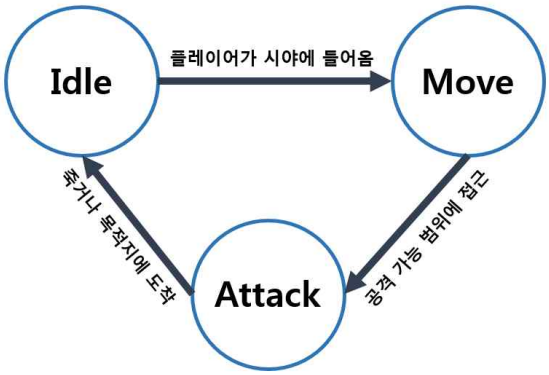

그림 [12]

그림[12]는 일반적인 슈팅 게임에서 NPC에 적용되는 간단한 FSM의 예시이다. 그림과 같이 기본 상태인 Idle에서 정해진 상황이 되면 Move상태로 전환된다. 그리고 Move상태에서 또 다른 상황이 되면 Attack상태로 전환한다. 이렇게 FSM으로 게임에 적용될 알고리즘을 만들면 발생할 수 있는 상황을 한눈에 볼 수 있고, 오류를 찾고 수정하는데 시간이 절약된다.

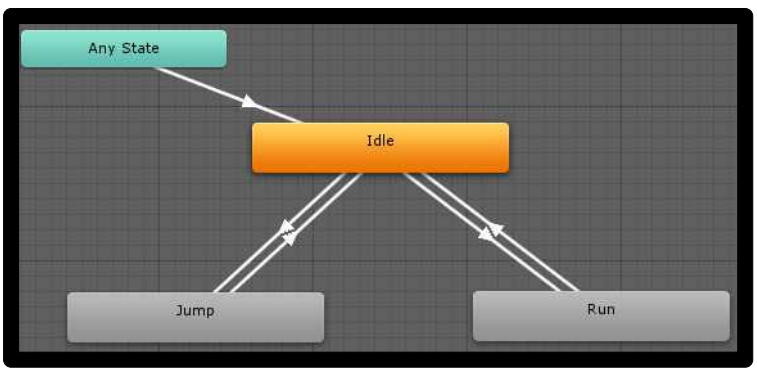

그림 [13]

그림 [13]은 유니티에서 FSM을 만드는 기능이 있다. 따라서 알고리즘을 개발하는데 많은 도움이 되고 오류와 수정에 용이하다.

2.3 게임 시나리오

그림 [14]

그림[14]는 지금 까지 개발 되었던 눈싸움 게임의 모습이다. 이러한 눈싸움 게임에서 영감을 얻고 TPS(Third-Person Shooter)형식의 눈싸움 게임을 개발하게 되었다. 플레이어는 주어진 캐릭터를 조작하고, 플레이어를 공격하는 적NPC를 제거하는 간단한 방식이다. 공격방법은 마우스 클릭 또는 키넥트의 정해진 동작을 취할 경우 플레이어의 캐릭터가 눈을 던지게 되고 이 눈을 맞은 적NPC는 체력이 줄어들고, 체력이 바닥 날 경우 사라지게 된다. 전체 게임은 각각의 스테이지로 나누어져 있다. 플레이어가 주어진 스테이지를 해결하면 그 다음 스테이지로 이동하게 된다. 적NPC에는 게임AI가 적용되어 있고, 난이도가 올라갈 수 록 AI의 성능이 올라가고, 적 객체의 개수가 늘어나게 된다.

게임은 크게 두 가지 모드가 있다. 첫 번째로 플레이어 혼자서 진행하는 1인 모드가 있다. 이 모드에서는 스테이지 마다 생성되는 적NPC의 수가 늘어나고, 게임AI의 성능을 조절하여 난이도가 상승하게 된다. 적NPC의 수는 3마리에서 시작하고 다음 스테이지로 넘어갈 때 2마리씩 추가가 된다. 또한 플레이어에게 다가가는 속도나 점차 늘어나고, 체력이 늘어난다.

그림 [15]

두 번째는 그림[15]처럼 네트워크를 통해서 다른 플레이어와 대결을 하거나 함께 스테이지를 해결하는 다(多)인 모드가 있다. 호스트가 방을 생성하게 되면 다른 클라이언트들이 호스트를 통해서 게임을 진행하게 된다. 이 모드에서는 1인 모드에 등장하는 적NPC는 1인 모드보다 더 크고 공격력이 높은 보스 몬스터가 나오게 된다.

게임 컨트롤러는 총 2가지가 있다. 기본적인 컨트롤은 키보드와 마우스가 있다. 키보드는 플레이어 객체의 이동과 공격을 입력 받는다. 마우스는 카메라의 이동과 플레이어 객체의 회전 입력 값을 받는다. 또한 키넥트를 이용한 플레이를 구현했다. 키넥트를 이용해 사용자의 팔의 동작을 인식하여 공격을 하고, 사용자의 허리와 어깨를 인식하여 플레이어 객체의 회전을 동작하도록 구현했다. 추후에는 스마트폰의 터치를 사용하여 게임을 진행 할 예정이다.

2.4 게임 객체 소개

- Player

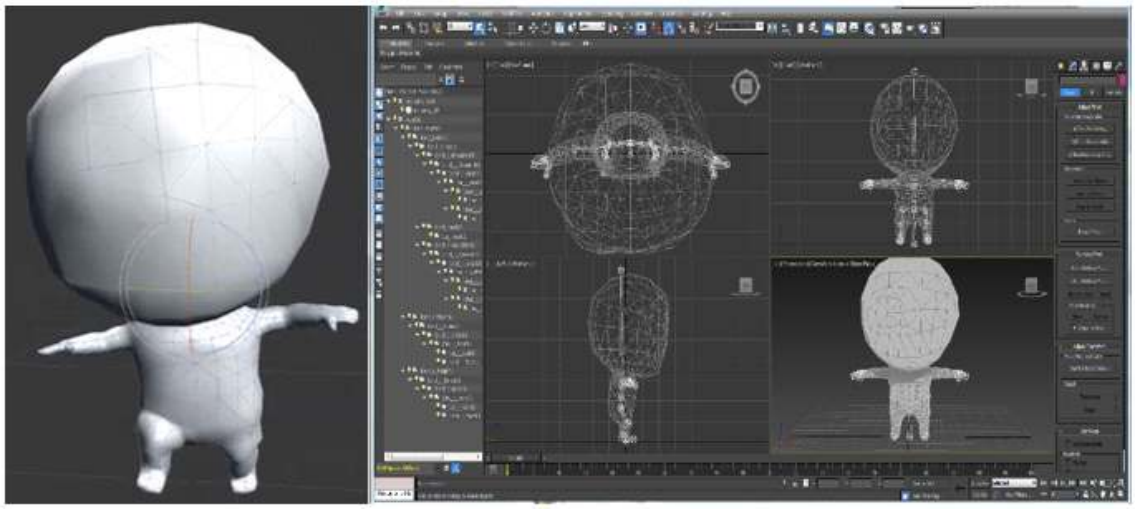

그림 [16]

그림[16]처럼 플레이어가 조종하는 Player객체는 3DMAX를 이용하여 제작되었고, 총 4가지 상태를 개발하였다.. 기본 상태인 Idle상태, 움직이는 상태인 Move상태, 공격하는 상태인 Attack상태 마지막으로 체력이 없어서 게임을 할 수 없는 상태인 Die상태가 있다. 이 4가지 상태에 대해서 각각의 애니메이션 효과가 적용되어 있다.

- **Enemy**

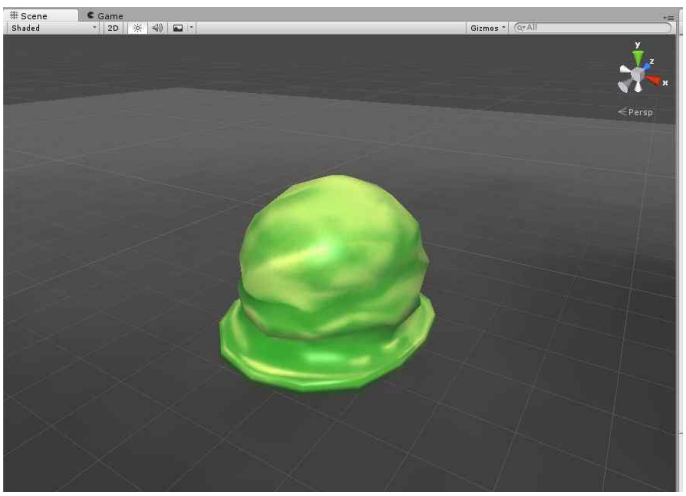

그림 [17]

그림[17]은 적NPC Enemy객체의 모델링 모습이다. Player객체와 마찬가지로

3DMAX를 이용하여 제작했다. Enemy객체는 Player객체 보다 FSM이 복잡하게 설계되어 있다. Enemy객체에는 Player객체와 같이 Idle, Move, Attack 상태가 있고, 추가적으로 Hiding, Recover, Die 상태가 있다. 각 상태는 A*의 값과 게임진행 상태에 따라 전환된다.

- Map

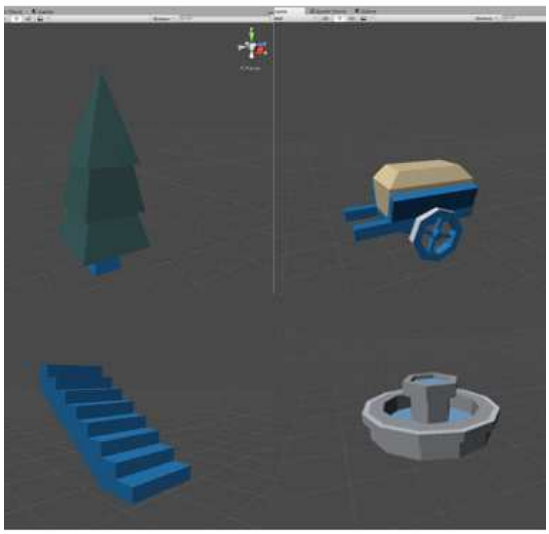

그림 [18]

그림[18]은 맵에 구성되는 객체들이다. Player와 Enemy객체와 같이 3DMAX를 이용하여 디자인 되었고, 게임의 분위기에 맡게 캐주얼한 분위기로 디자인 했다.

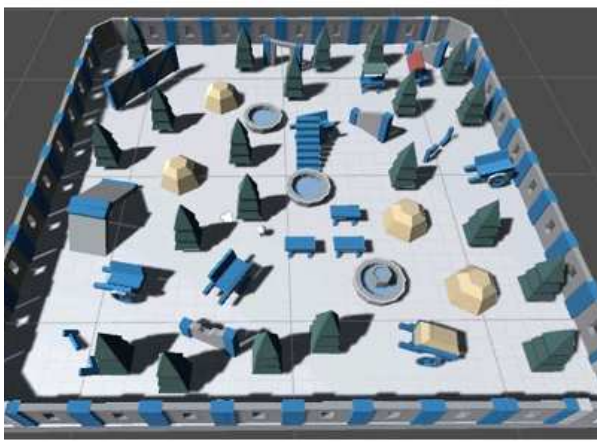

그림 [19]

그림[19]는 그림[18]의 객체들을 게임의 스토리와 난이도에 맡게 배치한 모습이다. Player는 각 객체를 이용하여 게임 플레이를 하도록 되어있다.

3. 연구 내용

3.1 Player 객체 알고리즘

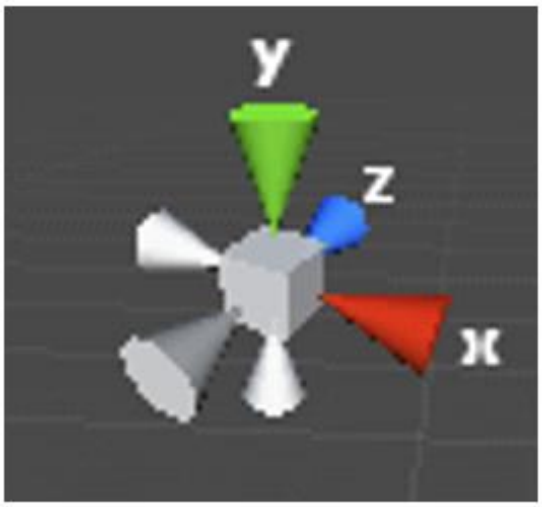

그림 [20]

Player객체는 플레이어가 직접 다루는 객체이기 때문에 따로 게임AI를 개발할 필요가 없었다. 따라서 기본적인 동작인 Move, Attack, Idle에 대한 함수가 개발되었다. 본 게임은 3D게임이고 유니티는 그림[20]처럼 X, Y, Z 3가지 좌표를 제공한다.

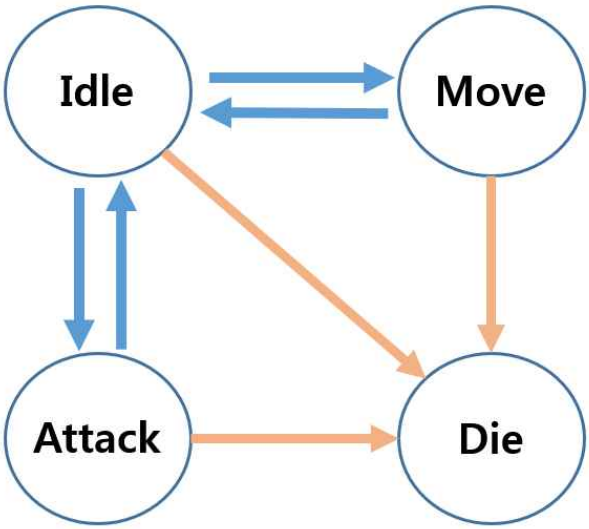

그림 [21]

그림 [21]은 Player객체에 FSM이다. Move에서 바로 Attack을 전환할 경우 움직임이 부자연스러운 문제가 있어서 중간에 Idle상태를 거쳐서 가도록 설계가

되었다. Die상태를 제외한 모든 상태에서 Player가 죽을 수 있기 때문에 모든 상태에서는 Player의 체력을 확인하고 체력이 0이하로 내려갈 경우 Die상태로 전환한다. Die상태로 전환되면 더 이상 플레이어는 Player객체를 조작할 수 없게 된다. Player객체의 FSM에는 하나의 계층이 존재한다. Idle상태와 Move상태 그리고 Attack상태는 같은 계층이고, Die상태는 앞의 3가지 상태보다 한 단계 위의 계층구조로 되어있다.

Move함수에는 캐릭터 이동, 카메라 추적, 캐릭터 회전 이 3가지를 다루는 함수이다. 캐릭터 이동은 키보드의 방향키를 입력받는다. 유니티에서 제공하는 함수임 Input.GetAxis("Horizontal")를 사용한다. 여기서 Horizontal은 좌, 우 값을 입력받는다. 위, 아래 값은 Vertical을 입력한다. Horizontal값을 h변수에 넣고, Vertical값은 v변수에 넣는다. Vector3 movement를 선언한 뒤, movement에 X좌표에는 h값을 Z좌표에는 v값을 넣는다. 게임에서 Player는 점프 동작을 하지 않기 때문에, Y좌표에 대한 값은 0f값으로 초기 값을 정한다. 그리고 정해놓은 속도 값 speed변수 값과 게임의 시간 값인 deltaTime을 movement의 초기 값과 곱한다.

마우스의 움직이는 방향에 따라서 Player 객체는 회전을 하게 된다. 유니티에서 마우스 좌표를 제공해준다. Input.mousePosition의 값을 가져오고, 그 값을 Player 객체의 rotate값에 넣어준다.

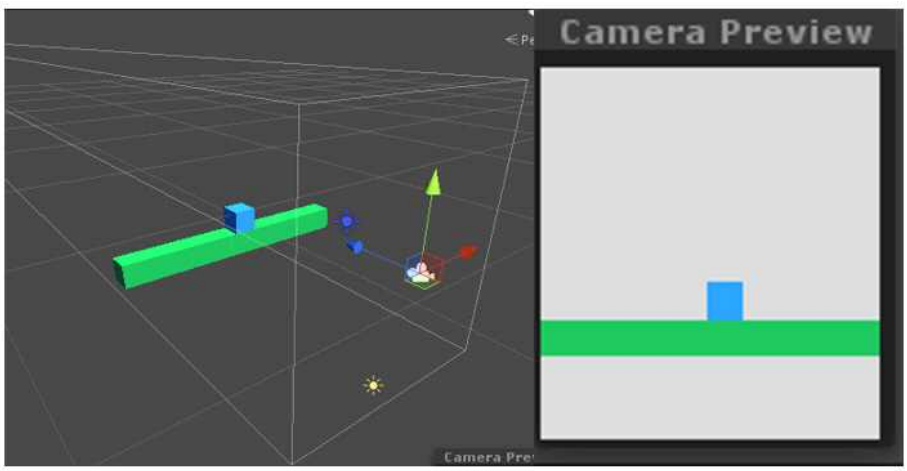

그림 [22]

그림[22]처럼 게임이 실행되면 게임 화면을 보여주는 객체가 Camera객체이다. 이 객체는 따로 개발하지 않고, Player 객체 안에 포함하여 제작하였다. Player

객체가 부모 객체이고, Camera객체가 자식 객체로 구성되어 있다. 따라서 Player의 X, Y, Z값에 변화에 따라서 Camera객체는 똑같이 이동하게 된다.

Attack함수는 마우스 클릭이 되었을 때, 총알 객체를 정해진 위치에서 생성하고, 목표지점을 설정하여 총알객체를 움직이는 함수이다. Player객체에서는 총알이 생성되는 위치 값을 가지고 있고, Player의 좌표에 따라서 값이 계속 바뀐다. 또한 Attack함수는 Enemy의 충돌체크 함수에게 총알의 위치 값을 주기적으로 전달한다.

3.2 Enemy 객체 알고리즘

Enemy객체에는 여러 가지 함수가 들어있고, 이 게임의 주 개발 객체이다. 우선 기본적인 동작은 Player와 비슷하게 Idle, Move, Attack함수가 있고, 게임AI를 담당하는 Hiding과 Recover함수가 있고, Move함수 안에는 Chasing함수가 포함되어 있다.

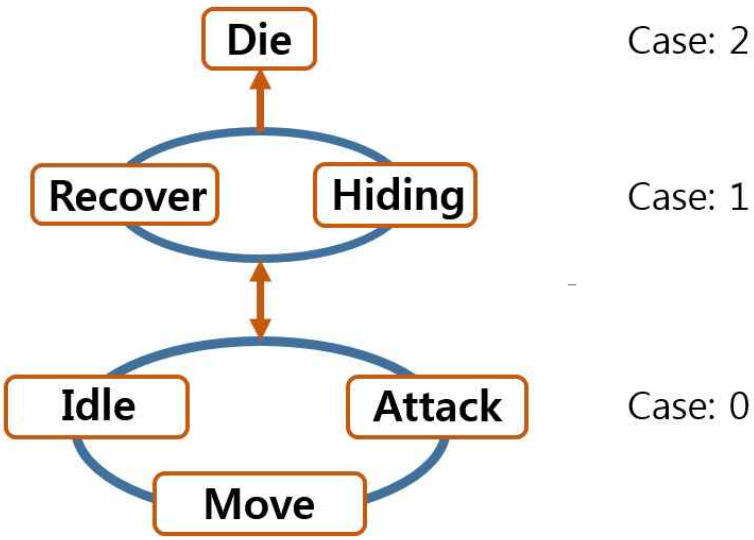

그림 [23]

그림[23]은 Enemy객체의 FSM이다. Player객체보다 더 많은 계층으로 이루어져 있다. Enemy의 상태와 게임 상황에 맞춰서 Case가 구분되어 있다. Case 0의 계층과 Case 2의 계층은 주기적으로 전환 될 수 있는 반면에 Case 2의 계층으로 들어가면 다른 계층으로 넘어갈 수 없는 구조로 설계되어 있다.

Case를 구분하는 Check함수가 있다. 이 함수는 Enemy의 체력상태에 따라서 Enemy의 Case값을 정하게 된다. Enemy의 체력이 원래 체력보다 50% 이하로 내려가게 되면 하던 움직임을 멈추고, Hiding함수를 시작한다. Hiding함수의 조건에 맞게 Enemy가 동작을 하면, 다음으로 Recover함수가 진행되고 Enemy의 체력이 일정량 증가한다. 여기서 체력이 증가하는 양은 게임의 난이도에 따라 다르게 설정되어 있다.

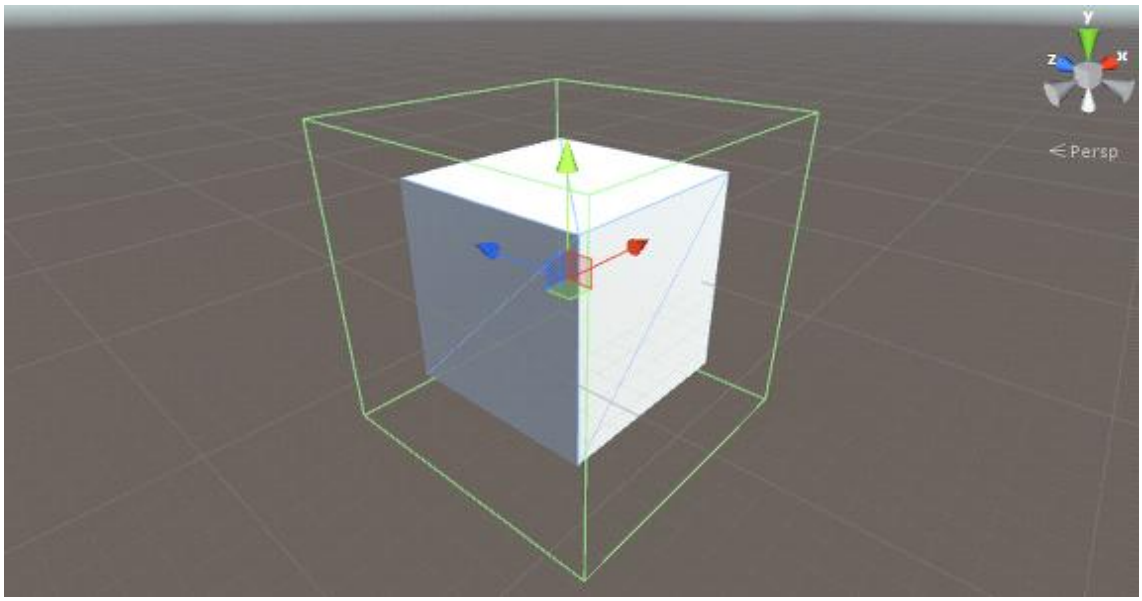

그림 [24]

그림[24]에서처럼 유니티에서 각 객체들은 Collider가 존재한다. Enemy와 Player객체는 보이지 않는 Collider로 둘러져 있다. 유니티에서는 이 Collider을 이용하여 각 객체의 충돌을 체크하게 된다. 유니티에서 충돌체크는 Collision과 Triger 2가지 방법이 존재하고, Collision을 사용했다. Enemy객체와 Player객체가 충동될 경우 Enemy객체는 Attack함수를 실행한다. Attack함수는 Player객체의 체력 값을 받아오고 그 값을 Enemy의 공격력만큼 낮추게 된다.

Enemy객체의 함수 중에서 가장 중요한 함수는 Move함수이다. Move함수에는 길 찾기 알고리즘이 들어있다. Enemy객체에 사용된 길 찾기 알고리즘은 맵을 격자 형태로 나눈 Grid방식으로 개발되었다. 길 찾기 알고리즘에서 얻어온 값을 GetPath함수에서 Enemy가 사용할 수 있는 값으로 구분하여 전달한다. Move함수는 GetPath로부터 받은 값을 확인한다. GetPath로부터 받은 값을 Path라고 하면, Path의 값이 Player객체의 위치에서 Enemy객체의 위치까지의 거리 값보다 클 경

우 Move함수는 Path를 따라서 Enemy객체를 이동한다. 반대로 Path의 값이 작을 경우 충돌체크 함수로 넘어가게 된다.

길 찾기 알고리즘은 Enemy의 상태에 따라서 목적지가 다르게 설정된다. 기본 상태에서 목적지는 Player가 된다. 하지만 Hiding상태와 Recover상태에서는 Player가 아닌 다른 지점을 목적지로 설정하게 된다. 각각의 Enemy들은 생성되는 위치가 다 다르게 설정 되어있고, Hiding상태가 되면 이 지점으로 가게 된다. Hiding상태에서 목적지에 도착하게 되면 Recover상태가 되고, 일정 체력을 채우게 된다. 이 상태에서 Player에게 공격을 받으면 다시 Hiding상태가 되고 랜덤으로 숨을 목적지를 설정하게 된다.

3.3 Unity3D Network

다多인모드에서는 유니티 네트워크를 사용하여 여러 플레이어가 동시에 게임을 진행하도록 했다. 유니티에서는 멀티 플레이 게임을 만들 수 있게 네트워크 기능을 제공한다. 그 중 HLAPI(High Level API)를 사용하였다. 이 기능은 "저레벨"의 구현에 대해서는 기본적으로 처리를 해주기 때문에 개발하는데 시간을 많이 절약할 수 있다.

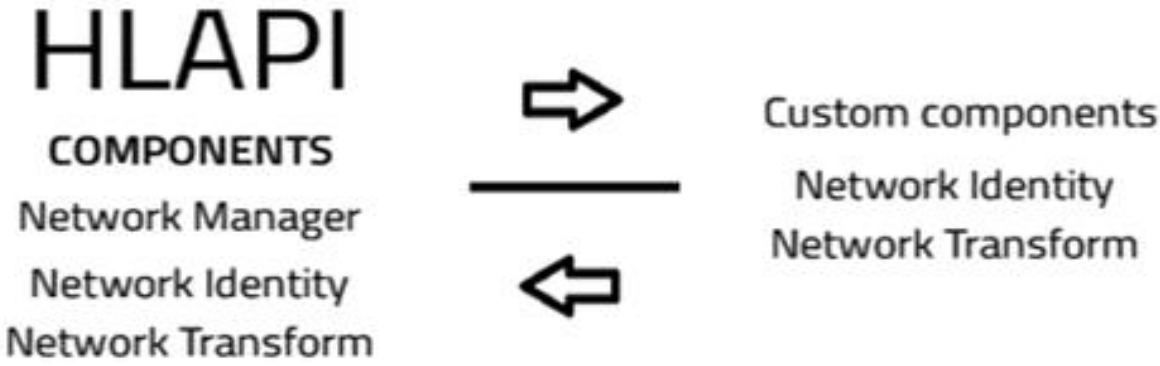

그림 [25]

그림[25]는 HLAPI를 나타내는 그림이다. Network Manager는 HLAPI에서 제공하는 클래스들을 이용해서 네트워킹 시스템을 편리하게 관리하기 위한 기능을 제공하고, 서버와 클라이언트를 최종적으로 관리한다. Network Identity는 네트워킹 시스템의 객체로 추가하는 컴포넌트이고, 오직 서버만이 객체를 생성할 수 있다. 새로운 플레이어가 접속할 경우, Identity를 이용하여 새로운 Player객체를 생

성하게 된다. Network Transform은 네트워크에 연결된 객체들을 동기화 하는 컴포넌트이다. 새로 생성된 Player의 위치 정보와 상태에 대한 정보를 서버에 전송하게 된다. 서버에 접속된 클라이언트 중에는 클라이언트 호스트가 존재한다. 이 호스트가 게임을 진행하게 된다. 따라서 클라이언트에서 서버로 네트워크 명령을 보내게 된다.

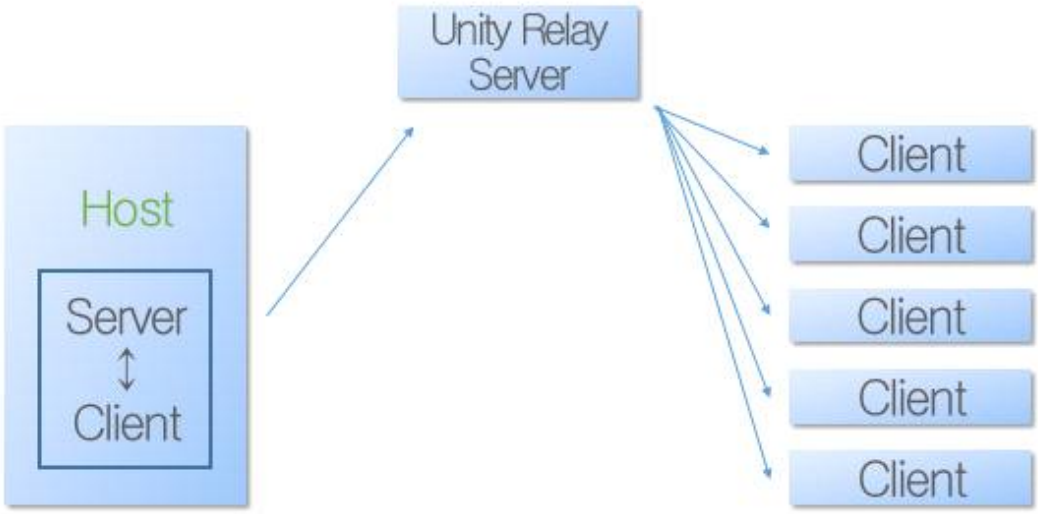

그림 [26]

그림 [26]은 유니티 네트워킹을 시각적으로 표현한 그림이다. 그림에서 보이듯 유니티 네트워킹 시스템에서 게임은 하나의 서버와 여러 클라이언트를 가진다. 전용 서버를 따로 만들지 않기 때문에, 클라이언트 중 하나를 서버로 설정합니다. 이 클라이언트를 호스트라고 한다. 호스트는 서버이자 동일한 프로세스 안의 클라이언트이다. 호스트는 LocalClient라는 특수한 클라이언트가 된다. 다른 클라이언트들은 RemoteClient가 된다. 동일한 프로세스 내에 있기 때문에 호스트는 직접적인 함수 호출 및 메시지 큐를 통하여 로컬서버와 통신한다. 다른 클라이언트들은 정기적인 네트워크 커넥션에 의해서 서보와 통신하게 된다. 여기서 호스트와 다른 클라이언트의 코드를 동일하게 하여, 한 종류의 클라이언트만 생각할 수 있도록 되어 있다.

네트워크 시스템에서 플레이어 오브젝트들은 특별한 존재로 간주된다. 게임을 플레이하는 각 사람에게는 플레이어 오브젝트가 있고, 관련된 명령은 그 오브젝트에 전송되게 된다. 자신이 아닌 다른 사람의 플레이어 오브젝트에 명령을 호

출할 수 없고 자기 자신에 대해서만 가능하다. 플레이어가 추가되고 연결이 이루어지면, 그 플레이어 오브젝트는 해당 클라이언트들의 호스트가 된다.

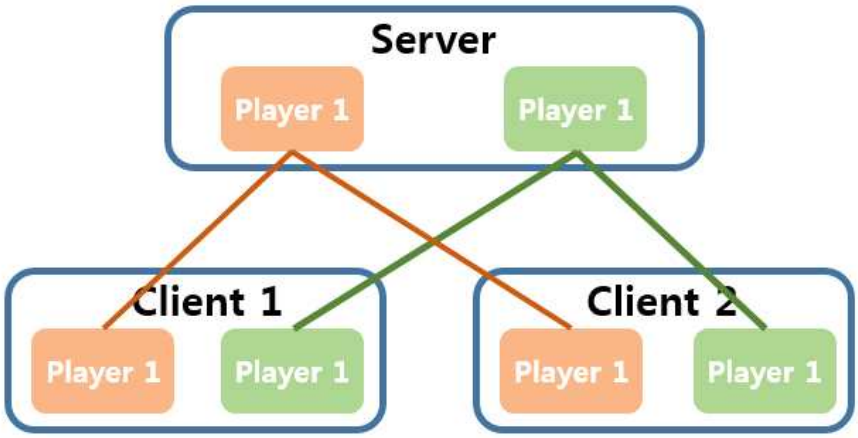

그림 [27]

그림[27]은 두 클라이언트와 그들의 로컬 플레이어를 나타낸 그림이다. isLocalPlayer 플래그 세트가 있는데 이것은 자신의 플레이어 오브젝트만이 키보드 입력처리, 카메라 설치 조작 등 자신의 플레이어에서만 이루어져야 하는 클라이언트 측의 처리를 하는 데에 사용할 수 있다. 그리고 플레이어 오브젝트는 로컬 권한을 가질 수 있게 된다.

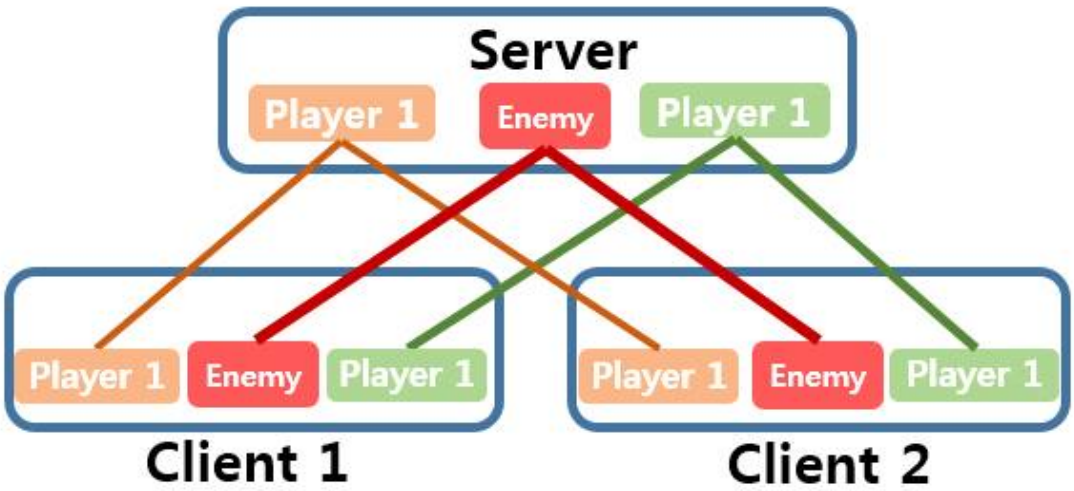

그림 [28]

그림[28]은 서버간의 Enemy객체를 클라이언트간의 공유하는 그림이다. Enemy객체는 플레이어가 없는 오브젝트이기 때문에 권한은 서버에 속하게 된다. 서버는 모든 클라이언트들에게 Enemy에 대한 정보를 보내주게 된다. 따라서 각 클라이언트에 동일하게 움직이는 Enemy를 볼 수 있게 된다.

3.4 Kinect 컨트롤러

키넥트는 가상현실게임을 구현할 때 자주 사용되는 컨트롤러이다. 게임 이외의 재활치료 등 많은 곳에서 응용되고 있다. 게임의 재미와 다양성을 위하여 키넥트를 이용해서 Player를 조작할 수 있도록 하였다.

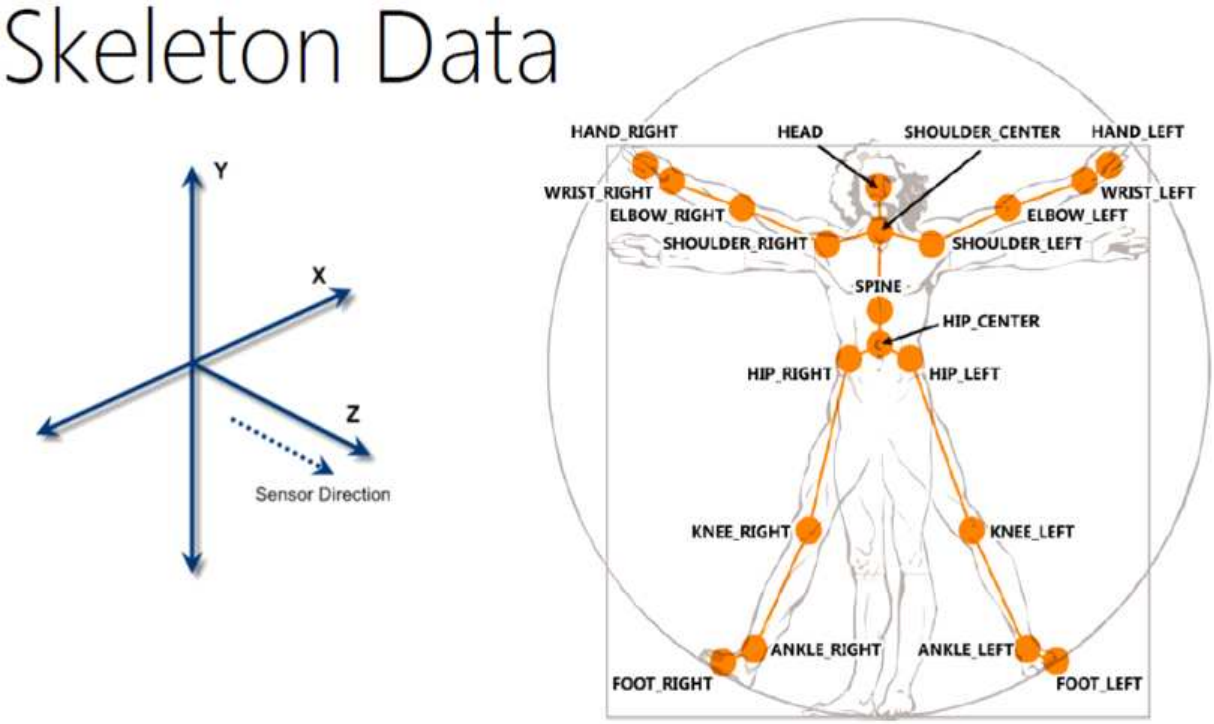

그림 [29]

그림[29]처럼 키넥트는 유니티와 같이 3가지 좌표를 제공한다. 또한 사람의 형체를 인식하고 각 관절 부분에 값을 제공한다. Player를 동작하는데 필요한 관절은 어깨, 팔꿈치, 허리, 손 좌표이다. 키보드와 마우스 컨트롤러에서는 마우스가 Player의 회전을 담당했고, 키넥트에서는 허리좌표가 마우스를 대신한다. Player객체 안에 회전 객체를 하나 생성한다. 그 다음 키넥트에서 제공하는 허리좌표를 회전 객체에 입력한다. 이미 Player객체에 Move함수에서 마우스좌표를 받아서 회전하는 부분이 있고, 이곳에 마우스 좌표대신 허리 좌표를 입력한다.

Attack함수에서는 마우스의 클릭을 입력받아서 Player가 공격을 했다. 키넥트에서 제공하는 좌표중 손, 팔꿈치, 어깨를 이용하여 마우스의 동작을 대신했다. 게임의 시나리오는 캐릭터가 눈을 던지기 때문에 플레이어가 눈을

던지는 동작을 취하도록 유도한다. 하지만 눈을 던지는 모습은 항상 일치하기가 힘들다.

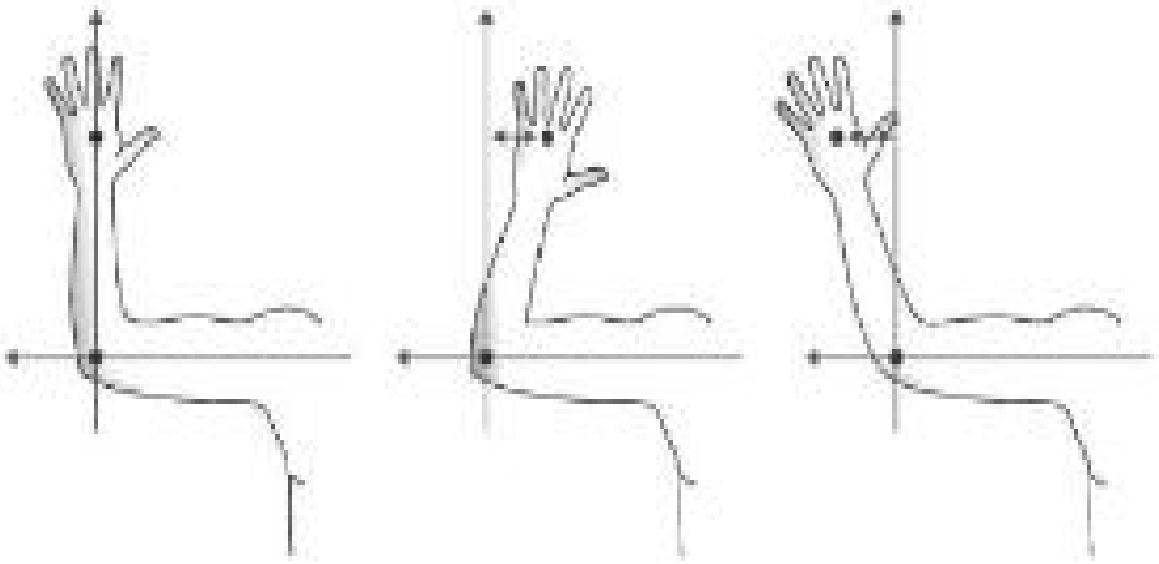

그림 [30]

따라서 그림[30]처럼 어깨좌표를 중심으로 손과 팔꿈치의 좌표가 어깨보다 위에 있으면 던지기 전 상태로 간주하고, 어깨보다 아래로 내려갈 경우 던지는 동작으로 인식한다. 정해진 동작에 맞게 사용자의 동작이 인식되면 Player객체에 Attack함수에 Ture값을 보내주게 되고 Player객체는 공격을 실행하게 된다.

4. 기능 및 구현

게임은 크게 두 가지 모드로 진행이 된다. 하나는 플레이어 한명이 스테이지를 해결하는 1인 모드가 있고, 네트워크를 통해 다수의 플레이어가 동시에 게임을 진행하고 경쟁하는 네트워크 모드가 있다.

4.1 1인 모드

1인 모드는 플레이어 혼자서 게임을 진행하는 모드이다. 게임은 스테이지로 나누어져 있고, 스테이지가 올라갈 수 록 Enemy의 객체수가 늘어나고, 플레이어를 따라가는 속도가 증가하게 된다.

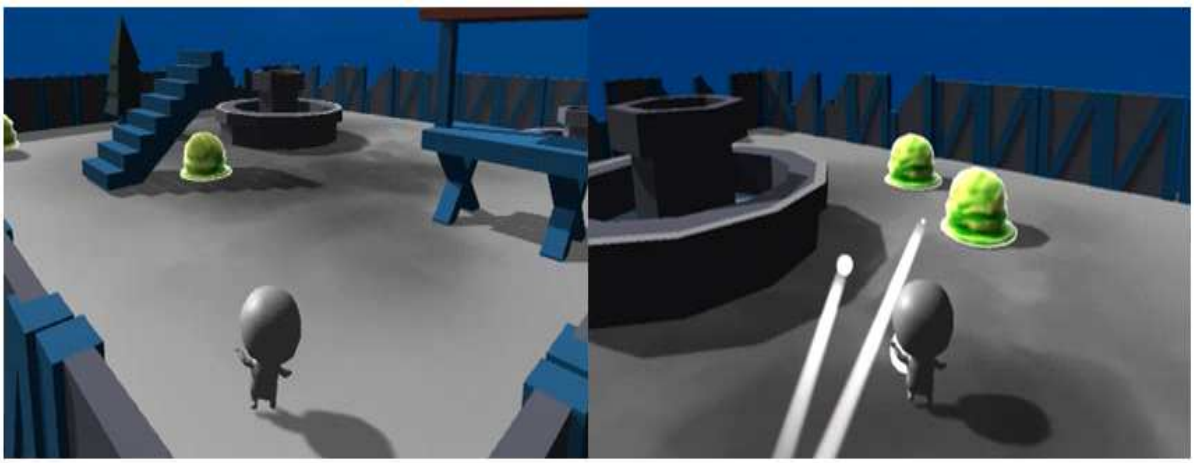

그림 [31]

그림[31]은 1인 모드에서 첫 번째 스테이지의 모습이다. 또한 마우스클릭으로 눈을 던져 Enemy에게 공격을 하는 모습이다. 첫 번째 스테이지에서는 Enemy 객체의 수가 적게 출현한다.

그림 [32]

그림[32]는 세 번째 스테이지의 모습이고, 첫 번째 스테이지 보다 더 많은 Enemy가 출현 하는걸 볼 수 있다. 그림에서 Enemy들 중 Player에게 다가오는 객체도 있지만, 체력이 일정 감소하여 Hiding이 실행되어 도망치는 Enemy의 모습도 보인다.

4.2 네트워크 모드

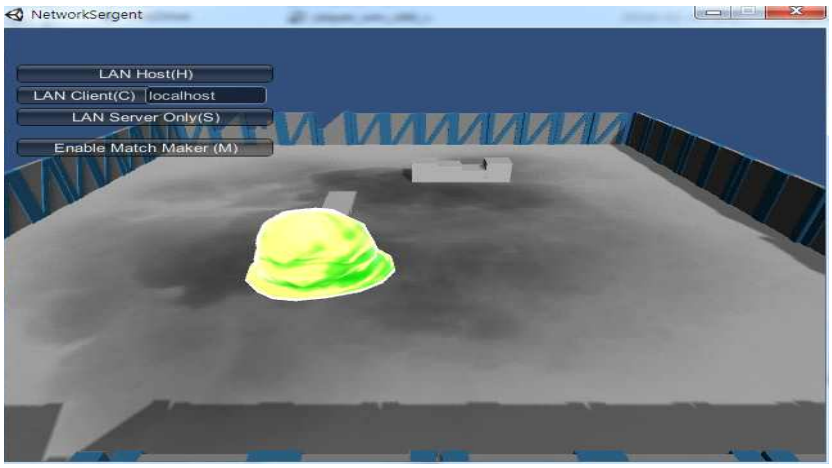

그림 [33]

그림[33]은 클라이언트가 호스트가 되기 전 상태이고, 왼쪽 상단에 버튼 중에서 LAN Host를 클릭하면 해당 클라이언트가 호스트가 되고 방이 개설된다. 호스트가 정해지면 다른 클라이언트들이 호스트를 통해서 서버에 접속이 가능하게 된다.

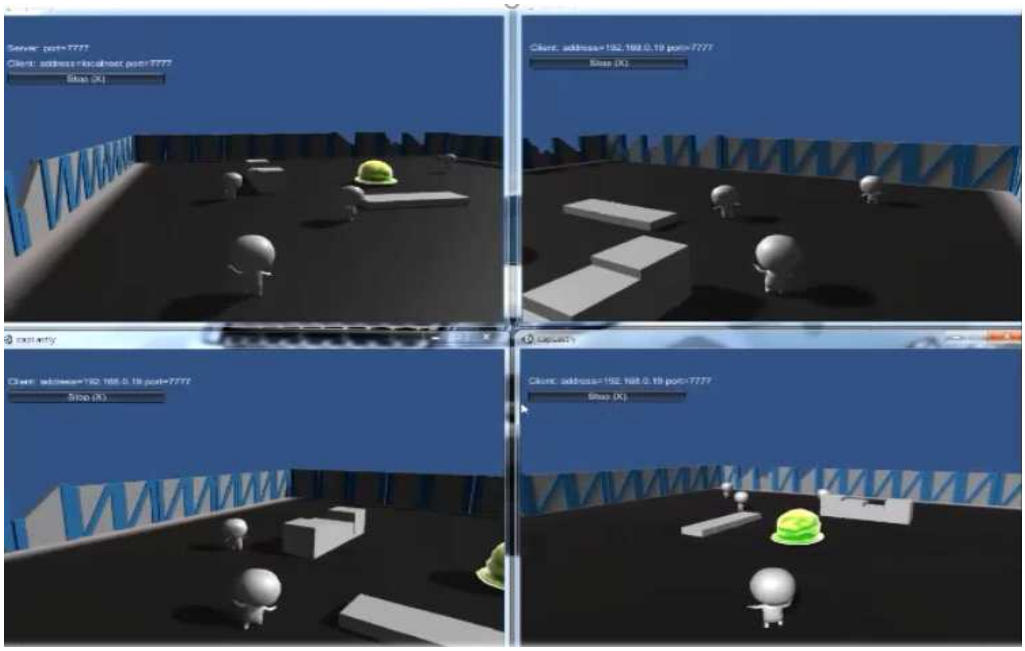

그림 [34]

그림[34]는 호스트가 개설한 방에 3명의 클라이언트들이 접속한 모습이다. 총 4명의 플레이어가 동시에 접속한 모습이고, 한 Enemy객체를 서로 공유해서 각 화면에 Enemy의 위치가 동일하게 보인다.

4.3 키넥트 컨트롤러

그림 [35]

그림[35]는 키넥트를 이용한 컨트롤 화면이다. 1인 모드와 다인 모드와 다르게 카메라 시점이 약간 내려가 있고, 볼록한 느낌을 갖는다.

그림 [36]

그림[36]에서처럼 손을 어깨 위에서 아래로 내리는 동작을 취하면 Player가 공격을 하고, 허리를 회전하면 Player객체도 똑같이 좌우로 회전을 한다.

5. 문제점 및 해결 방법

개발을 하면서 여러 가지 문제점이 발생했었다. 첫 번째는 길 찾기 알고리즘이 실행되고, 주어진 Path에 맞춰서 Enemy가 이동을 하는 과정에서 대각선방향을 이동하지 못하고 ㄴ자 모양으로 이동을 했었다. 이 문제를 해결하기 위해서 Grid방식 길 찾기 알고리즘에서 대각선의 Cost를 따로 계산하여 총 Cost를 구했다. 따라서 전보다 최단거리의 값이 더 작아지고 계산이 빨라졌다.

두 번째는 Player가 움직이면서 던지는 동작을 취할 때, 자연스럽지 못한 움직임을 보여줬다. 이 문제를 해결하기 위해서 움직이는 도중 Attack이 실행되면 중간에 연결 애니메이션을 만들어서 더욱 자연스러운 동작이 이루어지도록 했다.

마지막으로는 네트워크 모드를 개발할 때, 서로 다른 인터넷을 접속하고 있는 경우에는 서버를 통한 접근이 어렵다는 점이였고, 이 문제는 추후에 네트워크에 대한 공부를 더 하고, 새로운 방식을 이용하여 해결할 예정에 있다.

6. 향후 연구방향

지금까지 개발된 게임은 게임AI와 네트워크 등 게임개발에 중점이 되었다. 추후에는 캐릭터의 디자인과 맵 디자인을 수정하여 기획한 게임 분위기에 더 맞출 예정이다. 또한 Player의 모습도 여러 가지 형태로 제작하여 네트워크 모드에서 플레이어마다 다른 모습의 캐릭터를 조작하도록 할 예정이다.

Enemy의 종류가 현재는 한가지 이지만, 멀리서 공격을 하는 Enemy와 공중에서 날아다니는 Enemy등을 개발하여 게임의 재미 요소에 집중할 예정이다. 따라서 새로운 Enemy에 맞는 게임AI를 새로 개발하고, 길 찾기 알고리즘을 여러 가지 방식으로 개발할 예정이다.

게임개발의 최종 목표는 스마트폰게임으로 출시하는 것이다. 키보드와 마우스 컨트롤러를 스마트폰 터치로 동작할 방법을 구상하고, 상용화할 수 있는 완성도 있는 게임으로 개발할 예정에 있다.

<참고문헌>

[1] Kocca(한국콘텐츠진흥원), White Paper on Korea Games, 2011년 발행

[2] ZDNetKorea, 유니티 2분기 모바일 게임 산업 백과 2016 공개(기사), 남우혁, 2016.07.82

[3] 정보 과학회지, 게임인공지능 최신 연구 동향, 박현수(세종대)외 1명, 2013.07 발행

[4] 유니티 게임 AI프로그래밍2/e, 레이 바레라 외 3명, 에이콘(출판사)

[5] 실시간 전략 게임 환경에서 퍼지이론을 적용한 지능형 학습 알고리즘, 김건형(한세대) 외 1명

[6] http://jujubong.tistory.com/59, 게임과 AI, 2016.10.10.

[7] https://unity3d.com/kr/learn/tutorials, 유니티 제공 자습서, 2016.10.15.

[8] https://docs.unity3d.com/kr/current/Manual/UNet.html, 멀티플레이어와 네트워킹, 2010.10.20.

[9] 유니티3D 게임 스크립트, 카일 다우스트, 에이콘(출판사), 2015.05.29.

[10] Game Programing Patterns, Robert Nystrom, 2014.11.02.

[11] http://gameprogrammingpatterns.com/contents.html, 게임프로그래밍 패턴, 2016.10.22