3D

# 포트폴리오 기술문서

- Escape Mention -

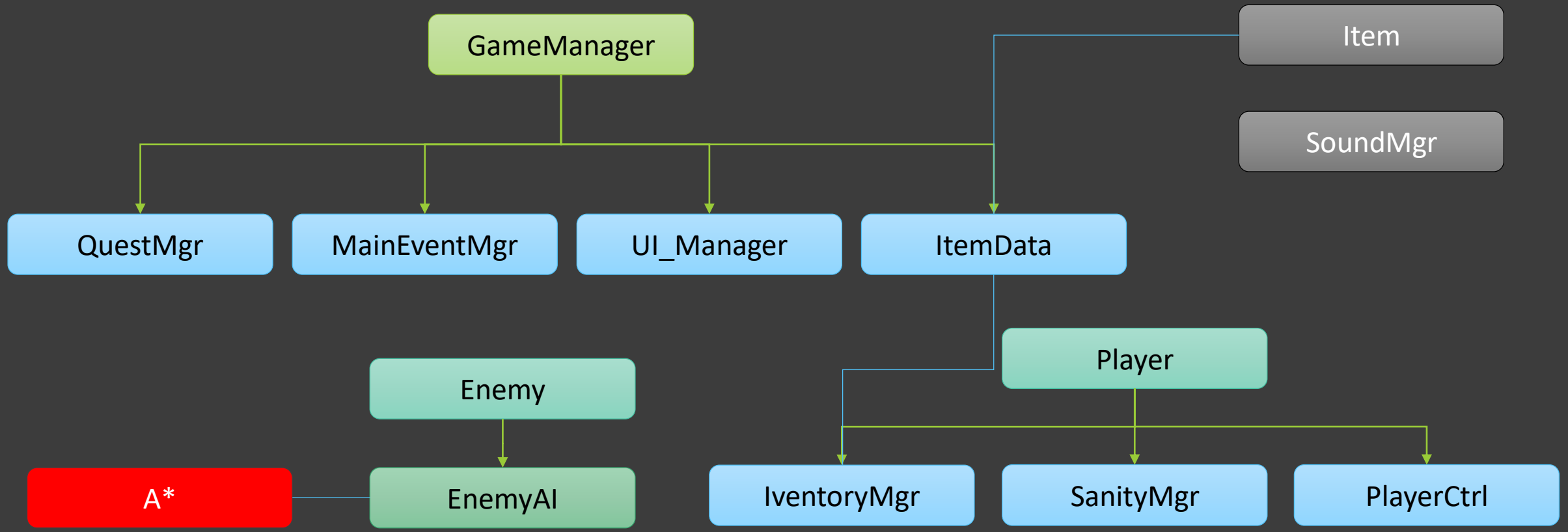
이승준

# < 목차 >

# < 게임 소개 >

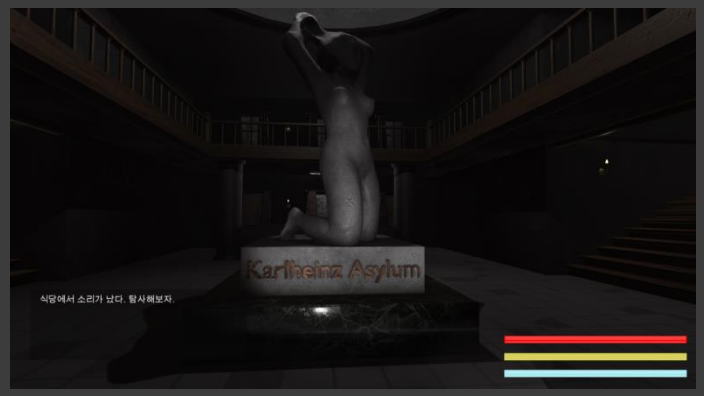
1. 장르 : 1인칭 서바이벌 호러
2. 인게임 목표 : 괴물들을 피해서 학교를 탈출한다.
3. 개발 엔진 : Unity
4. 개발 기간 : 2020/02/09 ~ 2020/02/29
5. 모티브 게임 : 아웃라스트, 암네시아
6. 영상 소개 : https://youtu.be/vCOS_shaH2o

# < 클래스 개요도>


GameManager
Item
SoundMgr
QuestMgr
MainEventMgr
UI_Manager
ItemData
Player
Enemy
A*
EnemyAI
IventoryMgr
SanityMgr
PlayerCtrl

# < 게임 플로우 차트 >


1 . 메뉴 씬




2. 게임 씬


2. 클리어 씬

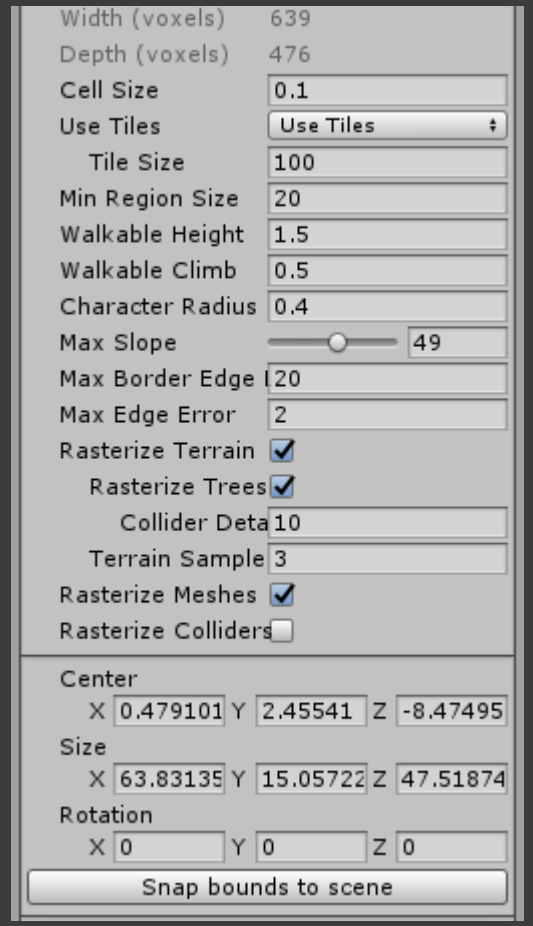
# < 개발 일정 >

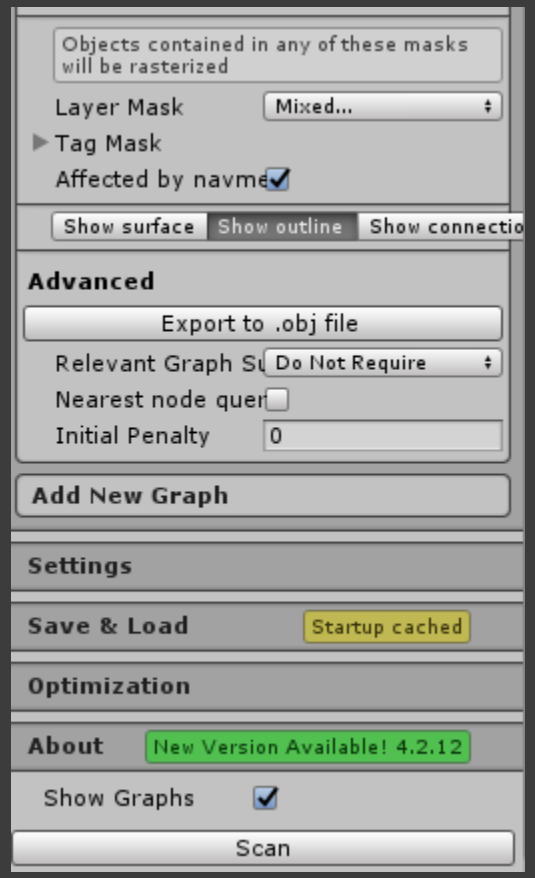
| 월 | 화 | 수 | 목 | 금 | 토 | 일 |
|---|---|---|---|---|---|---|
| 기획서 작성 및 사용할 에셋 찾기 | 1. 적 애니매이션 리소스 찾기<br>2.적 이동, 공격, 달리기 애니매이션 등 구현. | 1.맵 바닥, 벽, 문 깔기.(큐브)<br>2.내비게이션 메쉬를 이용한 적 순찰 시스템 구현. | 1. 플레이어 기초 이동, 달리기, 스테미나 구현<br>2. 체력, 스테미나 등 화면 UI | 플레이어 인벤토리 구현 | 1. 아이템 차트 만들기<br>2. 아이템 효과 구현 | 디버깅 및 작업 마무리 |
| 1. 플레이어 데쉬벨 차트 만들기<br>2. 데쉬벨로 플레이어 감지 및 추적 시스템 구현 | 데쉬벨로 플레이어 감지 및 추적 시스템 구현 | 패스 파인딩 에셋을 활용한 적 AI 강화 | 문열기, 아이템 찾기 등 사물 오브젝트와 상호 작용 구현 | 1. 1차 빌드 테스트<br><br>2. 디버깅 및 작업 마무리 | 플레이어 앉기 구현 및 애니메이션 구현 | 1. 정신력 시스템구현(시야 흐려짐, 달리기 제한 등의 패널티) |
| 게임 진행 가이드를 위한 퀘스트 시스템 구현 | 메인 이벤트 구현 | 1. 사운드 삽입<br>( BGM, 오브젝트 상호작용)<br>2. 3D UI카메라 구현 | 플레이어 특수 능력 구현 | 퀘스트와 메인 이벤트 보완 | 작업 마무리 및 빌드 테스트 | 작업 마무리 및 빌드 테스트 |

# < 게임 기능 소개>

1. 몬스터
2. 플레이어
3. 아이템과 인벤토리
4. 게임 시스템

# 1. 몬스터 - 길찾기

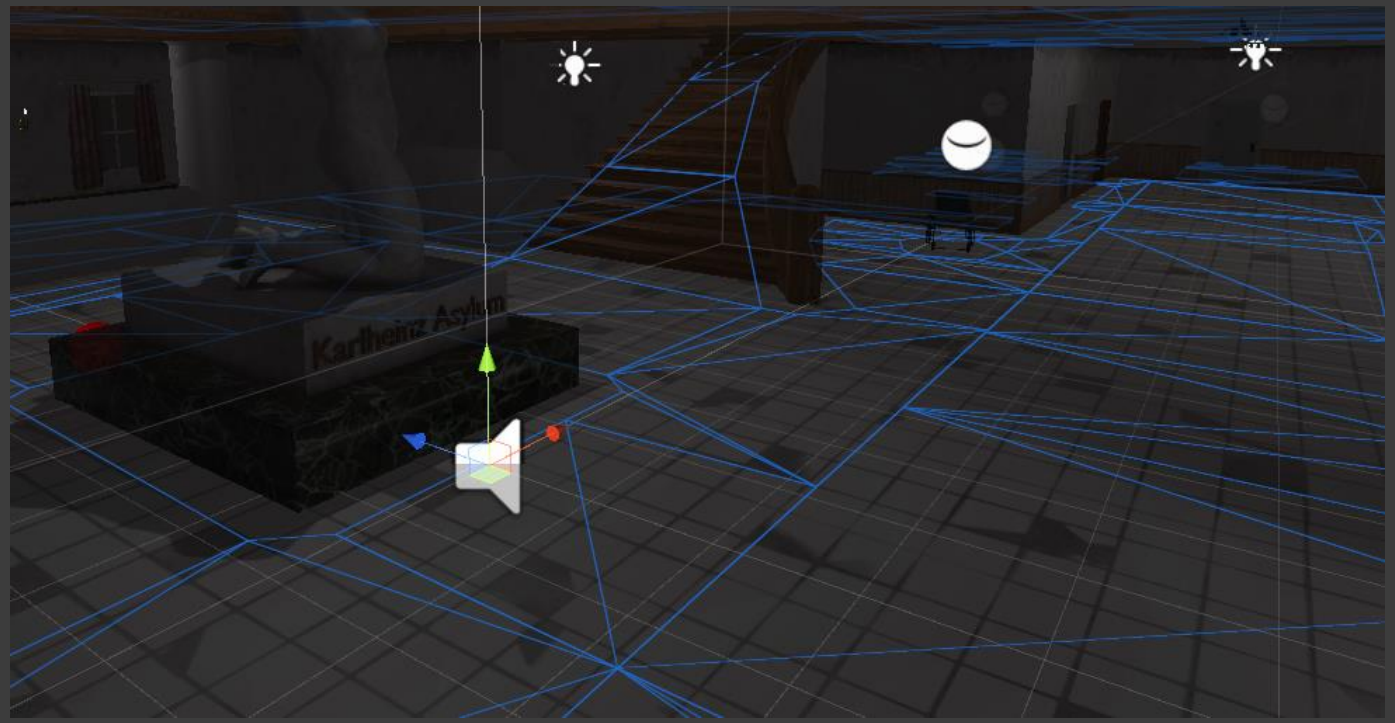


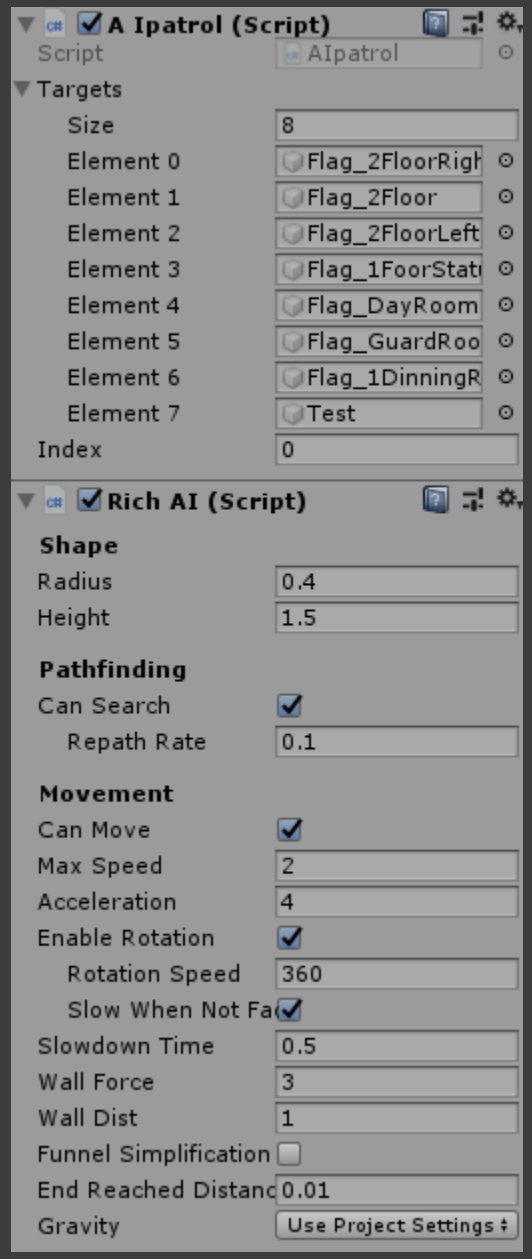

몬스터의 길찾기는 Unity 내장 내비게이션이 아닌 A* 기반으로 만들어지고 좀더 빠르고 향상된 성능을 제공하는 에셋을 활용했다.

길찾기는 기본적으로 Recast시스템 기반으로 구성되어 있다. 월드를 복셀화해 데이터들을 삼각형으로 표시하기 때문에 좀 더 부드럽고 빠른 경로 찾기를 할 수 있다.

# 1. 몬스터 - 길찾기



A* 시스템이 내장된 몬스터들은 파란색 선들의 영역 안에서 움직이게 되고 설정에 따라 실시간으로 최소 경로를 갱신하거나 길찾기가 시작된 시점에서 처음 계산된 경로를 따라 움직이게 된다.
이 길찾기 시스템은 콜라이더 사용을 권장하지 않고 있기 때문에 CharacterController컴포넌트를 사용해 충돌을 제어하였다.

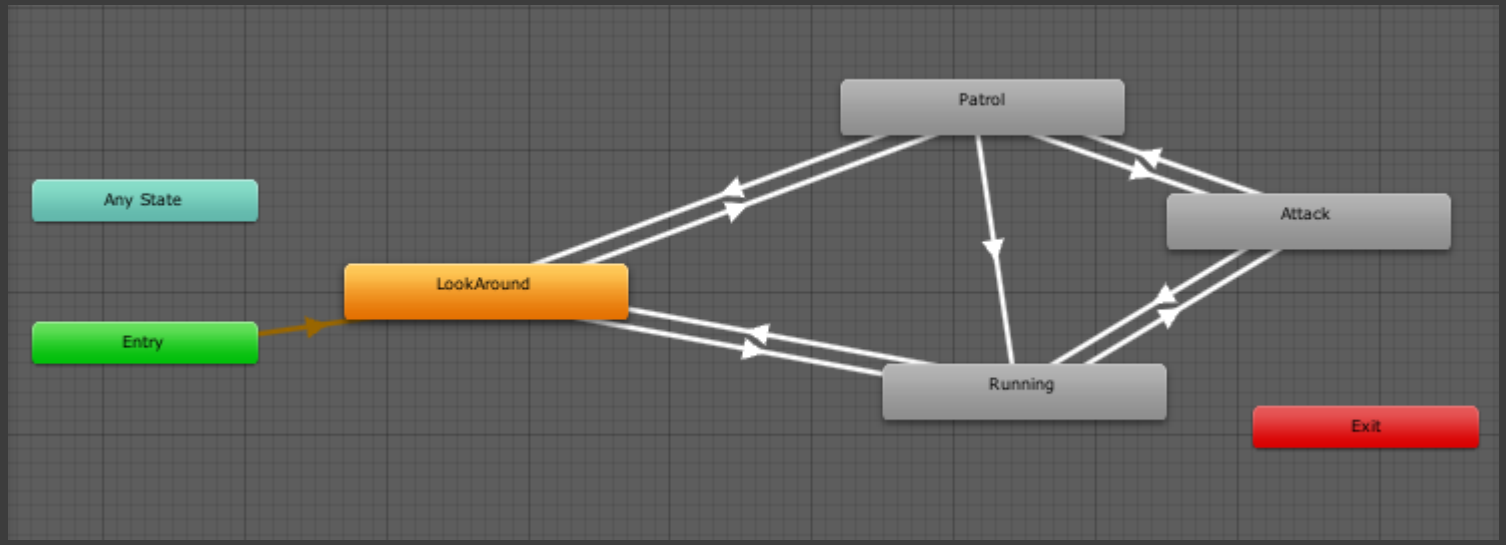



# 1. 몬스터 - 순찰

몬스터 움직임 제어는 에셋에서 제공하는 Rich AI 스크립트를 활용해서 제어했다. 하지만 연속적인 순찰 경로를 제어하는 기능은 제공하지 않기 때문에 AI Patrol이라는 스크립트를 만들었다.

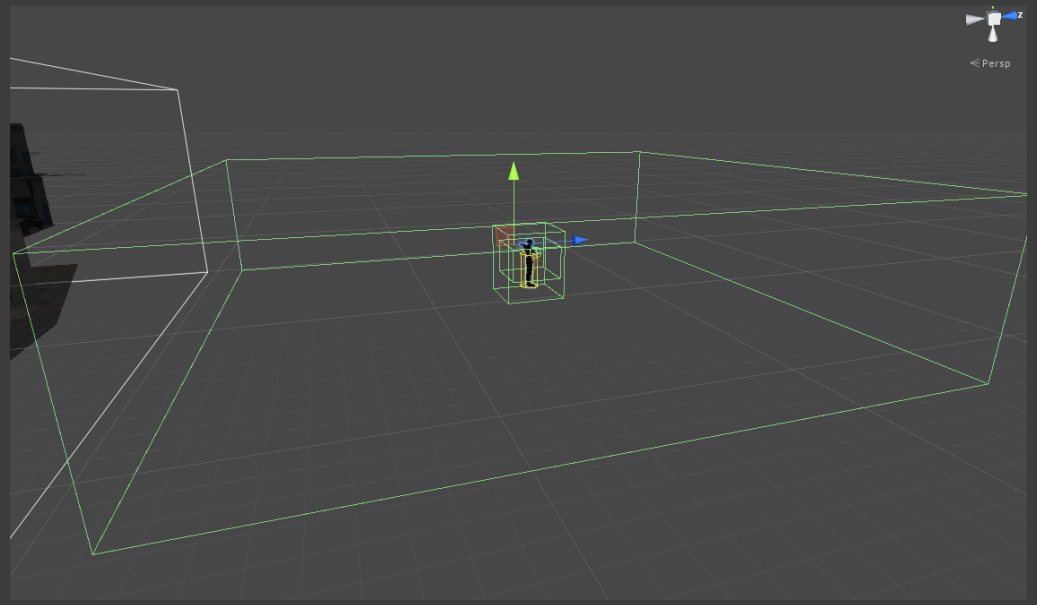
게임이 시작되면 몬스터는 Flag 태그를 검색해 목표지점들을 수집한다. 인덱스 번호에 따라 목표지점으로 순찰하고 Rich AI에서 목표 지점 도착 신호를 보내면 다음 목표로 이동한다. 만약 목표지점으로 가는 경로가 없다면 근처까지 이동한 뒤 다음 목표로 이동하게 된다.

# 1. 몬스터 – 애니메이션과 AI




몬스터의 애니메이션 처리는 애니메이터를 활용하여 간단하게 처리하였다. Enemy AI라는 스크립트에서 애니메이터의 파라미터를 제어하고 코루틴을 활용해 유기적으로 상태가 변할 수 있도록 설정하였다.
기본 움직임은 Patrol상태에서 목표지점에 도착하면 Idle상태가 되고 몇 초 후에 다시 Patrol이 되는 반복구조이다. 적을 발견하면 적을 추격하고 목표를 잃으면 다시 Idle – Patrol을 반복한다.

# 1. 몬스터 – 적 감지



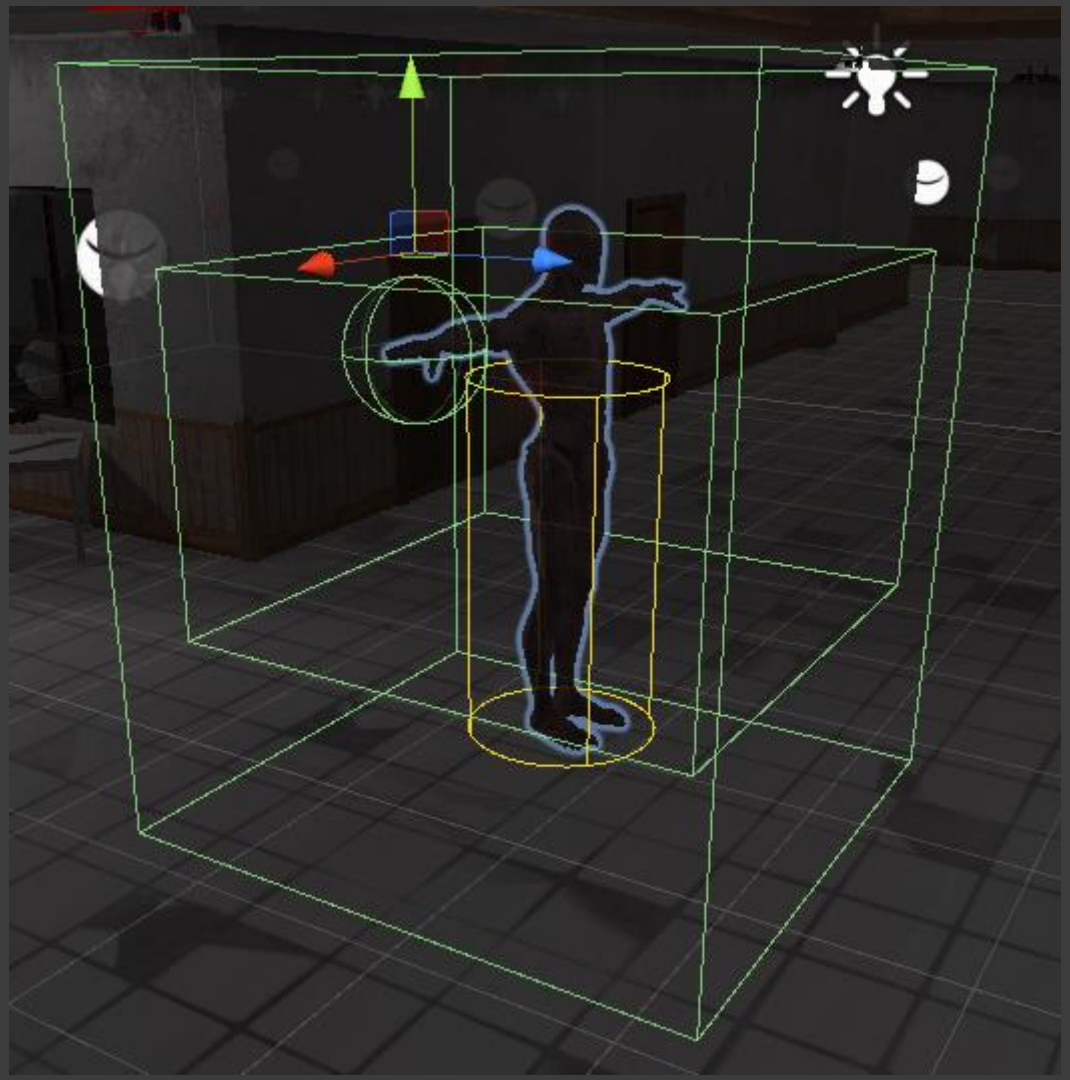

```
public void TraceTargetRun(Vector3 _v3)
{
    agent.maxSpeed = 5.5f;
    agent.destination = _v3;
    agent.SearchPath();
}

public void TraceTargetWalk(Vector3 _v3)
{
    agent.maxSpeed = 3.5f;
    agent.destination = _v3;
    agent.SearchPath();
}
```

몬스터는 Ear라고 하는 넓은 감지범위를 가지고 있다. Ear는 말 그대로 귀로써 플레이어의 움직임 소리를 감지하는 역할을 한다.

1. Ear를 통해 소리를 탐색한다.
2. 소리가 발견되면 해당 방향으로 레이캐스트를 발사해 벽 유무를 검사한다.
3. 벽이 없다면 소리수치를 그대로 받아들이고 벽이 있다면 1/10의 수치를 얻게 된다.
4. 소리가 일정 수치이하이면 적은 반응하지 않는다.
5. 일정 수치를 넘으면 값을 랜덤 함수에 넣어 확률적으로 소리에 반응한다. 소리 값이 커질수록 반응할 확률이 상승한다.
6. 소리에 반응하게 되면 소리 위치를 순찰지점으로 설정하고 소리 크기에 따라 걷거나 뛰거나 하는 추격 형태가 결정된다.

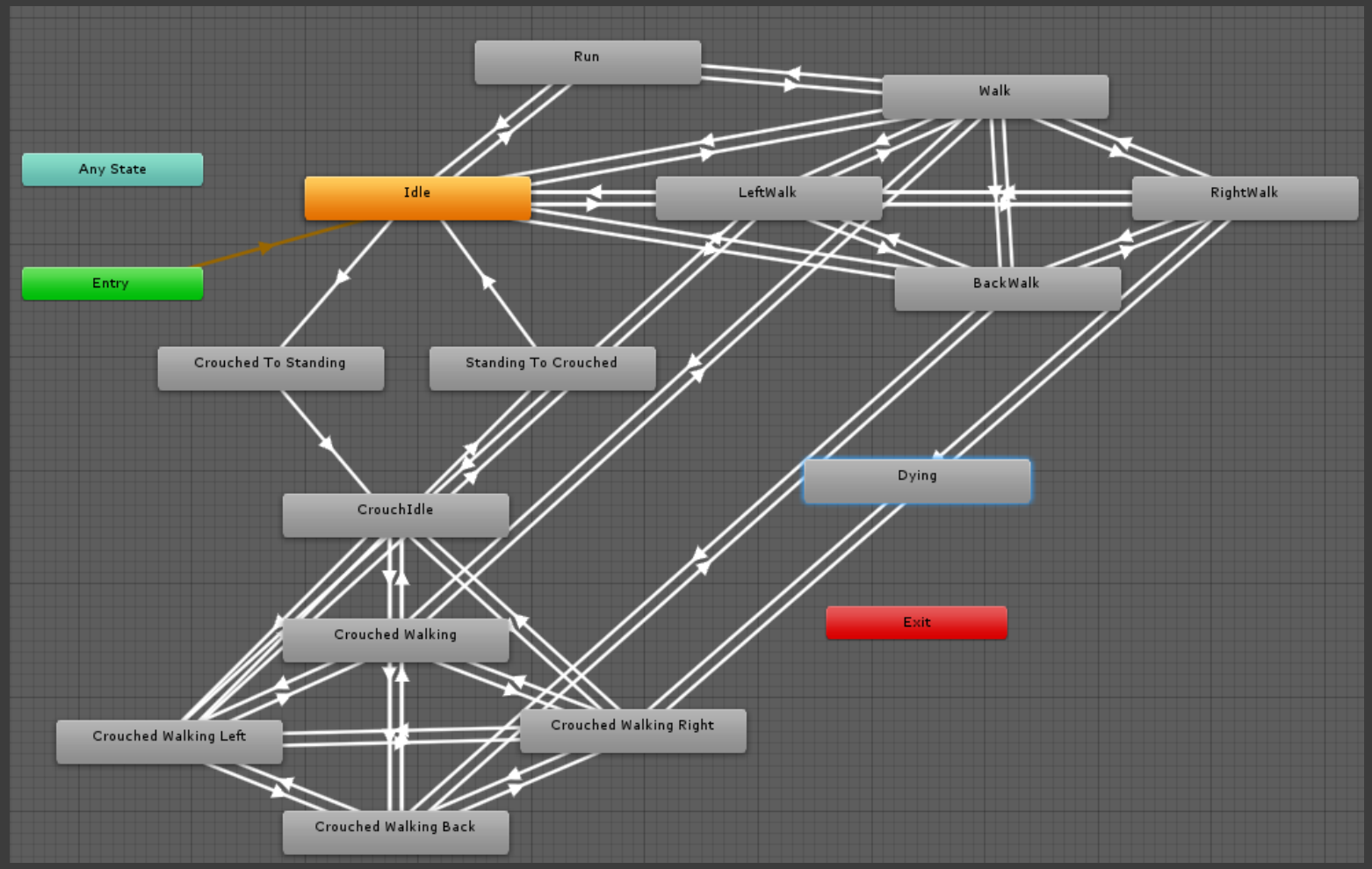
# 1. 몬스터 – 적 공격과 추격종료



적은 시야가 매우 제한되어 있어 플레이어를 직접적으로 감지하기는 어렵지만 직접 탐지 범위(바깥 박스)안에 들어오면 소리와 관계없이 탐지 당하고 공격범위(안 박스)에서는 공격 당하게 된다.

순찰 형태는 기본 Idle – Patrol과 적 추격Patrol이 있는데 적 추격 상태에서는 플레이어의 소리 위치를 지속적으로 탐색해 순찰 지점을 갱신한다.
하지만 마지막 갱신지점에 도착하고 더 이상 소리를 탐지하지 못하면 Idle – Patrol 상태로 돌아가 순찰을 재개한다.

# 2. 플레이어 - 애니메이션

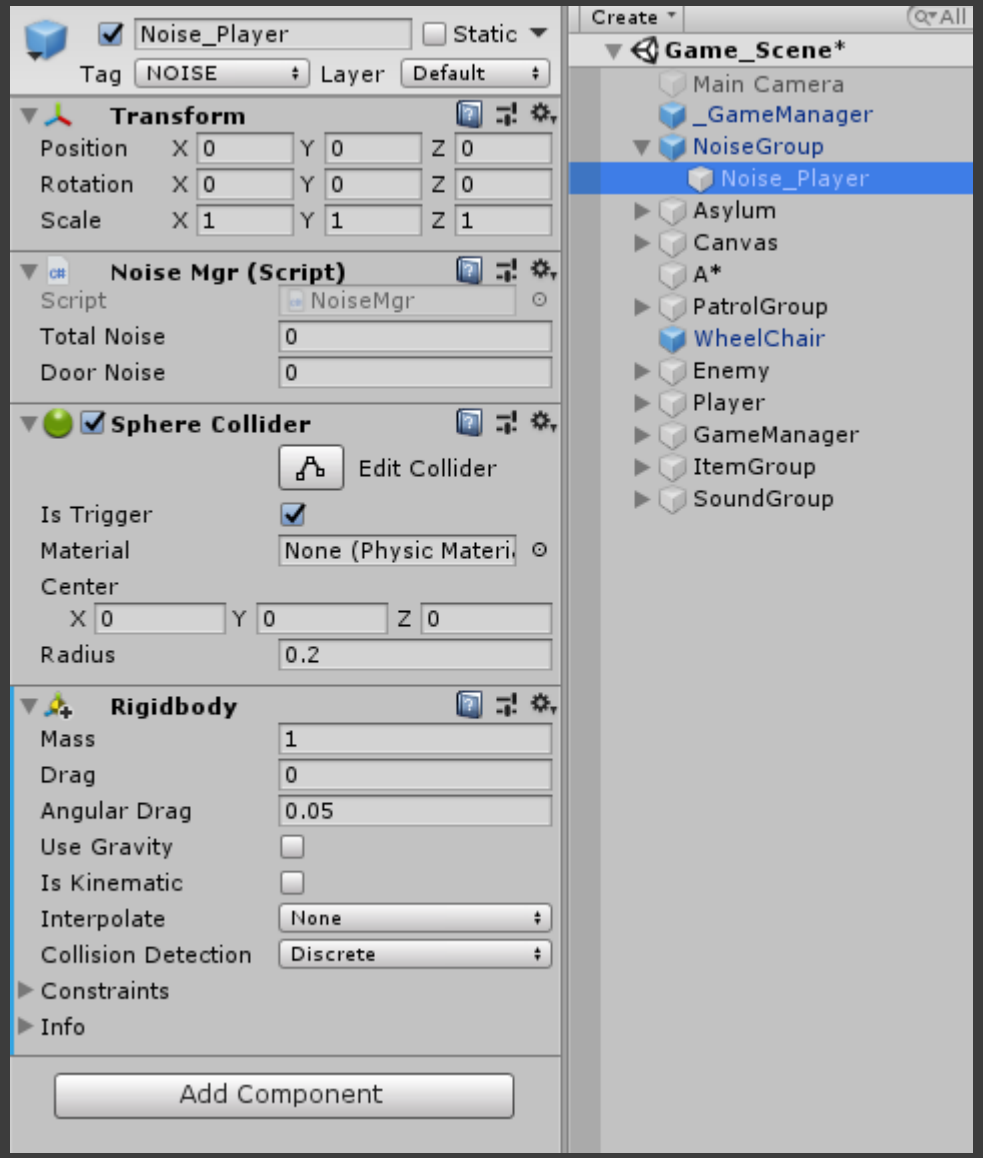



플레이어의 움직임은 3가지로 구분된다.

1. 걷기
2. 뛰기
3. 앉은 상태로 걷기

걷기 상태에서는 어떤 상태로도 전환이 가능하다.
뛰는 상태에서는 앉기상태로 전환이 불가능하다.
걷기와 앉은 상태에서 걷기는 언제든지 전환이 가능하다.

# 2. 플레이어 - 소리




플레이어의 움직임은 소리를 발생 시킨다. Noise_Player는 플레이어를 따라 다니며 플레이어의 소리를 수집해 소리를 방출한다.

<소리 탐지의 예>

플레이어의 기본소리(심작박동) 10 + 걷기(100) + 벽유무 + 거리에 따라 소리 밸류 감소

한 값을 적이 탐지한다.

# 2. 플레이어 – 스테미나와 정신력


달리기를 하게 되면 스테미나가 사용되어 그림의 노란색 바가 감소한다. 스테미나가 없으면 걷기만 가능하고 시간이 지나면 회복된다.

정신력은 하늘색 bar로 표시되고 적이 근처에 있거나 특수 능력을 사용하면 감소한다. 정신력 수치가 낮아지면 패닉 상태가 되어 그림처럼 시야가 흐려진다.

시야 흐려짐은 블러를 사용해 처리했다.
관련 에셋 : https://assetstore.unity.com/packages/vfx/shaders/fullscreen-camera-effects/fast-mobile-camera-motion-blur-with-lwrp-support-140084

패닉상태에서의 초조함을 간단하게 표현하기 위해 Idle 상태의 애니메이션 속도를 조절하여 Idle상태에서의 불필요한 움직임을 더 자주 느낄 수 있게 했다.

# 2. 플레이어 – 감지


특정 키를 누르면 지속적으로 정신력이 감소하는 대신 플레이어의 소리범위를 볼 수 있다.
해당 시스템은 Sonar Shader를 수정해서 만들었다.
기존의 Sonar는 물체가 부딪히는 충격력을 감지해 충격파를 만들어 낸다. 하지만 이 게임에서는 플레이어의 소리를 0.3초 마다 가져와 충격파를 생성하도록 수정해서 사용하였다.

관련 에셋 :
https://assetstore.unity.com/packages/vfx/shaders/simple-sonar-shader-102734

# 2. 플레이어 – 오브젝트 상호작용



플레이어의 눈에는 카메라가 장착되어 지속적으로 전방에 레이캐스트를 발사한다. 캐스트가 충돌하면 해당 오브젝트의 레이어를 검사하고 그에 따라 오브젝트를 들거나, 획득하거나, 문을 열거나 하는 상호작용이 가능하다.

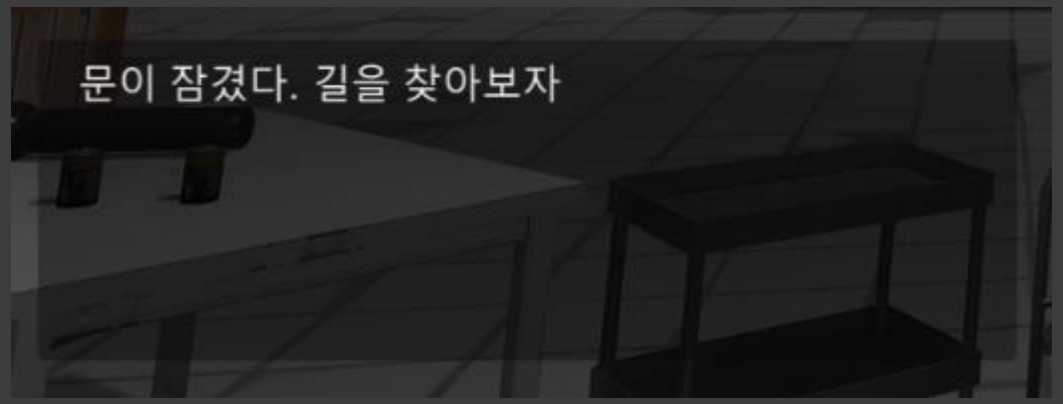
# 3. 아이템과 인벤토리 - 인벤토리

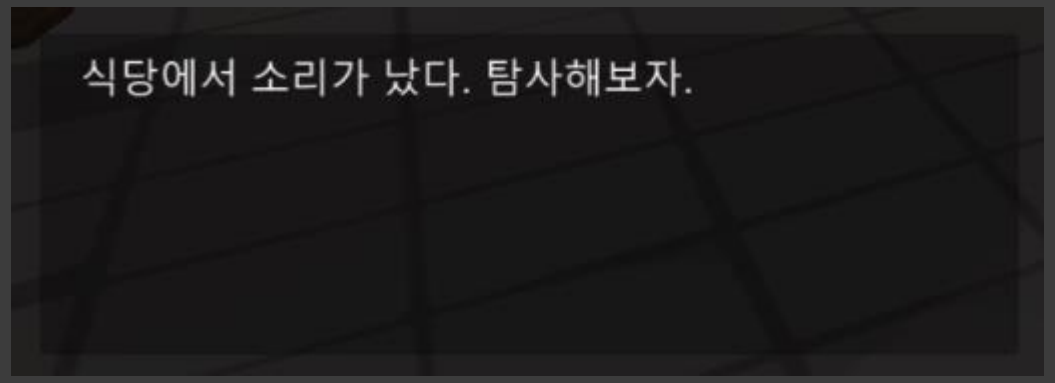

아이템을 획득하면 가방을 열어 해당 아이템들의 정보를 볼 수 있다. 더블 클릭을 하면 사용된다.

그림 오른쪽에는 클릭된 아이템이 회전을 하면서 정보를 보여준다.

UI에서 3D오브젝트 회전을 표현하기 위해 UI카메라를 만들고 Depth기능을 이용해 인벤토리를 열었을 때 같이 표시되도록 만들었다.

# 4. 게임 시스템 - 퀘스트




플레이어의 게임 진행을 보조하는 시스템으로 다음 진행을 위한 약간의 힌트를 준다.

특정 아이템을 가지고 있거나 위치에 도착하면 다음으로 진행되도록 설계했다.

# 4. 게임 시스템 – 메인 이벤트

단조로운 게임 플레이를 지양하기 위한 장치.

특정 조건을 만족하면 자동으로 진행되며 플레이어는 간단한 퍼즐을 풀어 클리어해야 한다.

# < 개발 이슈와 해결 >

1. 기존의 적 AI를 유니티 내부의 NaviMesh를 사용하였으나 맵이 복잡해지면서(복층, 문 열리고 닫힘) 동선이 꼬이는 버그
   : 더 향상된 기능을 가진 에셋을 활용하여 코드로 처리하기 힘들었던 복잡한 움직임을 간단하게 구현

2. 빌드단계에서는 Static 오브젝트에 접근하지 못해 A* 그래프 작성이 되지 않는 문제
   : 에디터에서 미리 그래프를 작성 후 저장하고 빌드 시 로드함으로서 해결, 그래프 작성 시간이 줄어 들어 로딩 시간이 단축되는 일석이조의 효과

3. 플레이어의 애니메이션 처리 문제
   : 앉기가 추가되면서 기존 애니메이터가 너무 복잡해졌기 때문에 블렌드 트리를 활용하려고 했다. 하지만 기존의 함수들을 전면 수정해야 되는 문제가 생겨 사용법만 익히고 적용하지 못해 아쉬웠음