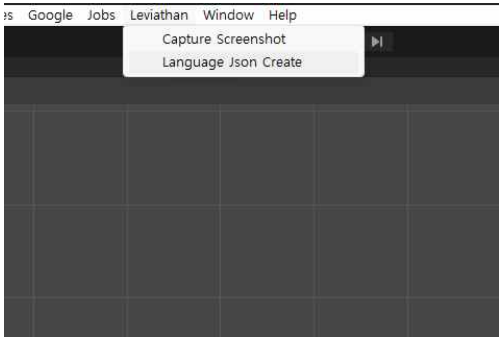
포트폴리오 설명문서

분류: 3인칭 슈팅

제목: 리바이어던

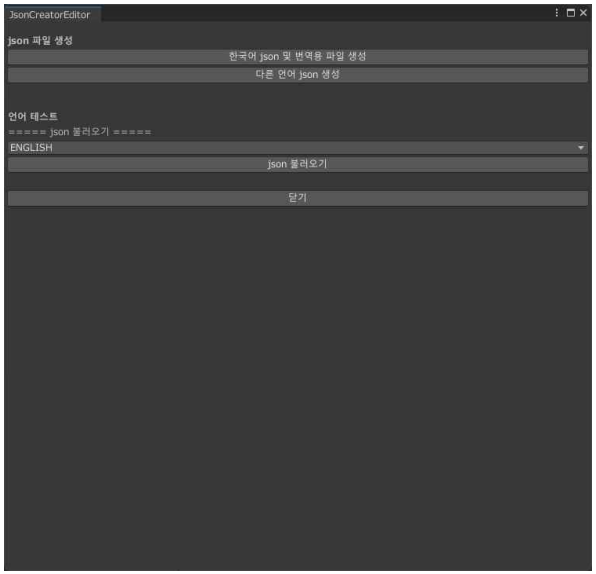
유니티 에디터 확장 기능



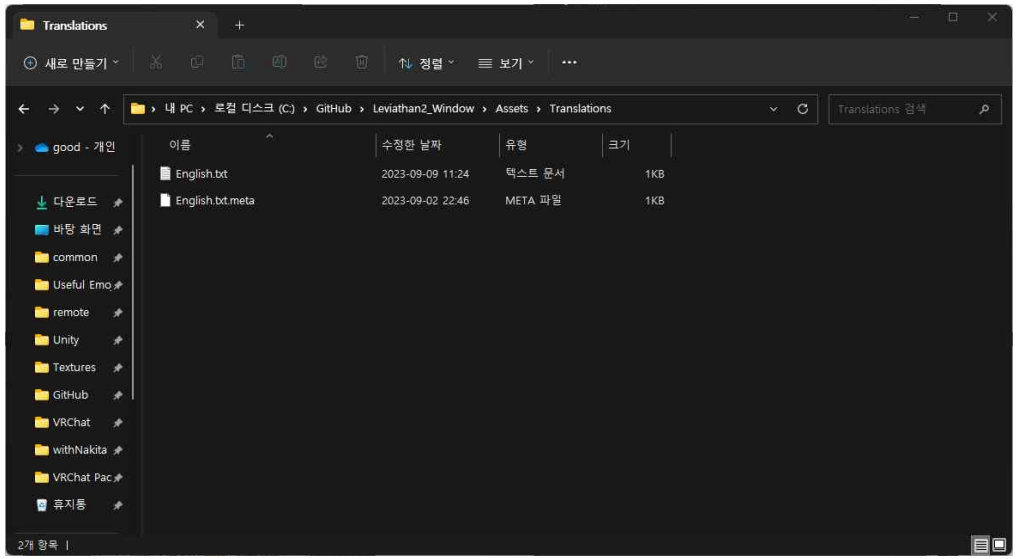
상단 메뉴에 새로 추가한 기능 사용 가능.

Capture Screenshot을 누르면 그 시점의 Game 화면을 Assets 폴더에 png 파일로 저장.

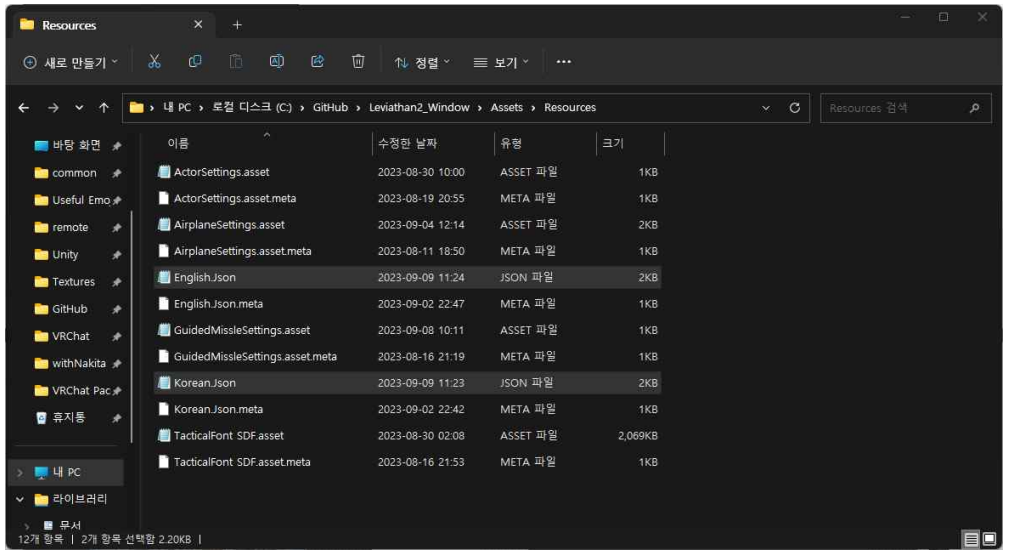
이 기능은 Actor의 프로파일 사진 촬영에 사용.



Language Json Create를 누르면 Json 파일 생성용 창을 소환하고 버튼을 눌러 파일 생성.



Translations 폴더의 텍스트 파일을 전부 읽어와서 Resources 폴더에 파일 이름은 그대로 json 형식으로 파일 생성.



Custom Shader 제작 - URP

Actor 셰이더

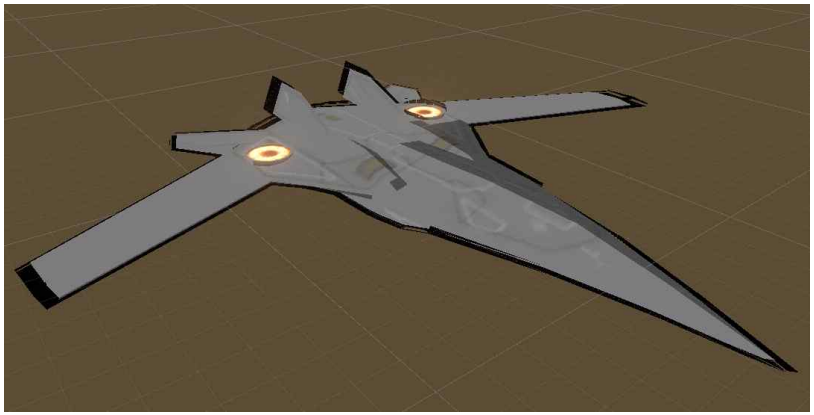
물체의 음영은 toon 셰이딩 느낌으로 제작. 그림자 부분은 에디터에서 그림자 색을 지정하여 텍스쳐 색상과 곱셈하여 표시.

Rim light를 toon 셰이더 느낌으로 주면 외곽선처럼 보일 수 있기 때문에 rim light는 광원을 등진 방향에서만 보이도록 제작.

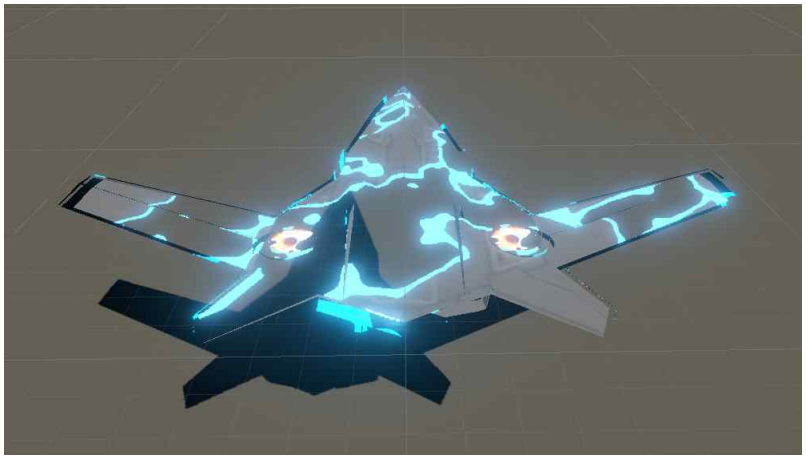
Local 좌표 기준으로 정점의 y 좌표값에 따라 dissolve 여부를 결정. 스크립트로 해당 메타리얼에 접근해서 dissolve 정도에 값을 주면, 그 값 이하의 y 좌표값에 해당하는 부분은 dissolve되어 투명화됨. 이때 투명과 불투명의 경계는 자연스럽게 보이기 위해 밝은 색깔로 표시.

Custom Shader 제작 - URP

비행기 셰이더



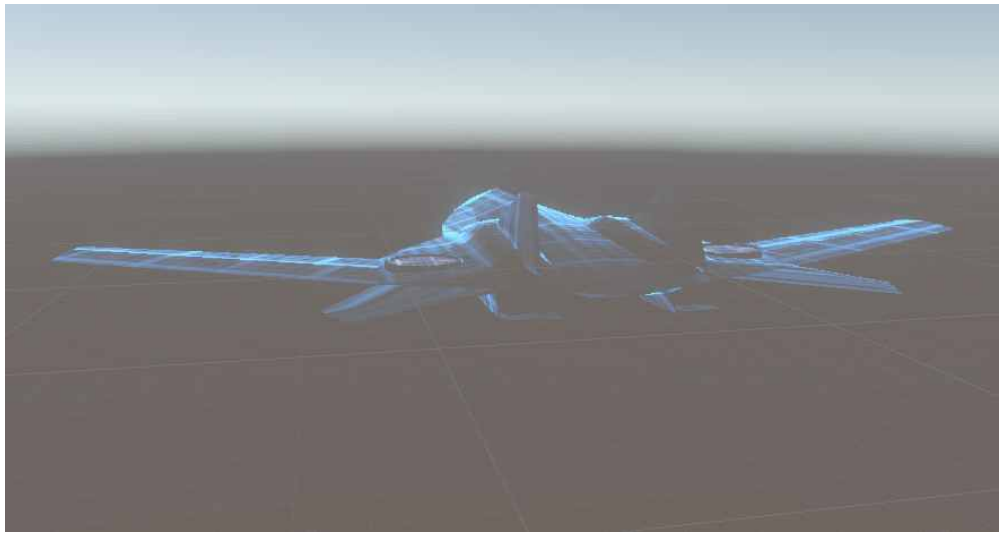
물체의 음영은 toon 셰이딩 느낌으로 제작. Actor와 마찬가지로 그림자 부분은 에디터에서 지정한 그림자 색과 텍스쳐 색상으로 표시.



Dissolve는 Actor와 다르게 노이즈 텍스쳐를 이용해서 적용.

이때 자연스러운 dissolve를 위해 외곽선도 dissolve 적용.

노이즈 텍스쳐에는 UV 애니메이션을 적용.



Dissolve로 투명화 시 격자무늬 텍스쳐에 rim light를 적용해서 투명화되었음을 표시.

Dissolve 시 투명과 불투명의 경계를 표시할 때, dissolve 정도로 주어진 값 이하에 해당하는 부분은 모두 밝은색을 주도록 했기 때문에, 격자무늬 텍스쳐를 알파 값으로 사용.

외곽선은 스크립트에서 dissolve 값을 주게 하기 위해 render feature를 사용하지 않고 비행기 메타리얼에 새로운 슬롯으로 추가.

외곽선을 구현하는 방법은 normal 방향으로의 정점 이동 방식과 local 좌표 기준으로 scale을 변환하는 방식을 혼합.

OutlineMode라는 프로퍼티를 만들고, 그 프로퍼티가 0에 가까울수록 normal 방향으로의 이동, 1에 가까울수록 scale 변환하는 방식에 더 가중치를 줌.

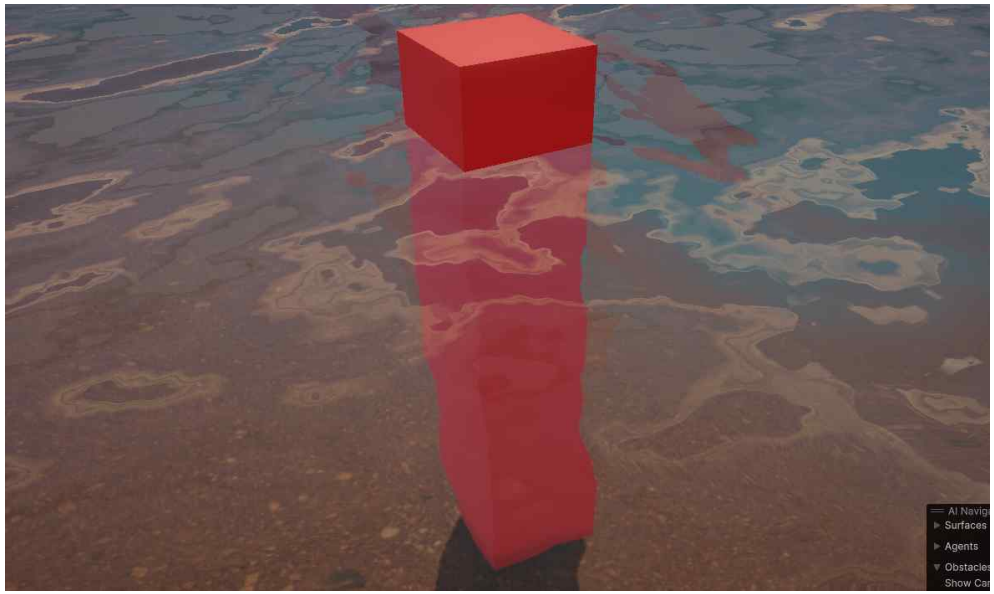
Custom Shader 제작 - URP

물 셰이더

자연스러운 물의 일렁임을 표현하기 위해 두 개의 normal 텍스쳐를 사용하여 UV 애니메이션 적용.

Skybox는 cubemap을 사용하고 있고, 셰이더그래프의 Sample Reflected Cubemap 노드에 사용 중인 cubemap과 normal을 적용하여 일렁이는 반사효과를 제작.

이때 반사효과는 rim light처럼 적용해서 측면에서 볼 때만 보이도록 하고, 수면을 수직으로 볼수록 투명해 보이도록 제작.



물이 투명해 보일 때는 굴절 효과를 나타내기 위해 셰이더그래프의 Scene Color 노드에서 UV 부분에 normal을 적용하여 표시.

굴절 정도는 수심이 깊을수록 커지게 하기 위해 Scene Color 노드에 normal을 적용하기 전에 수면 depth와 밑바닥 depth의 차이를 normal에 곱셈함.

Normal에 depth 값을 곱셈하여 굴절 효과를 구현하면 굴절이 이중으로 보이는 현상이 발생하기 때문에, depth 차이 계산을 두 번 하여 이중 굴절을 상쇄.

결과적으로 얕은 수면에서 적은 굴절, 깊은 수면에서 많은 굴절이 나타남.

또한, 수심이 깊을수록 투명하게 보이는 것이 아니라 물의 색상을 표시해서 점점 불투명해 보이도록 제작.

Custom Shader 제작 - URP

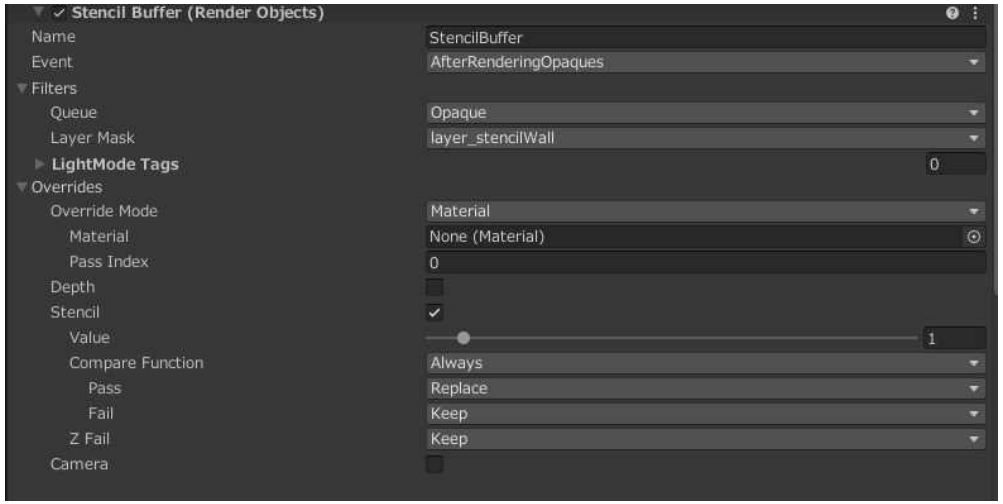
Stencil Mask

벽 뒤에 있는 물체를 표시하기 위해 stencil mask 적용.

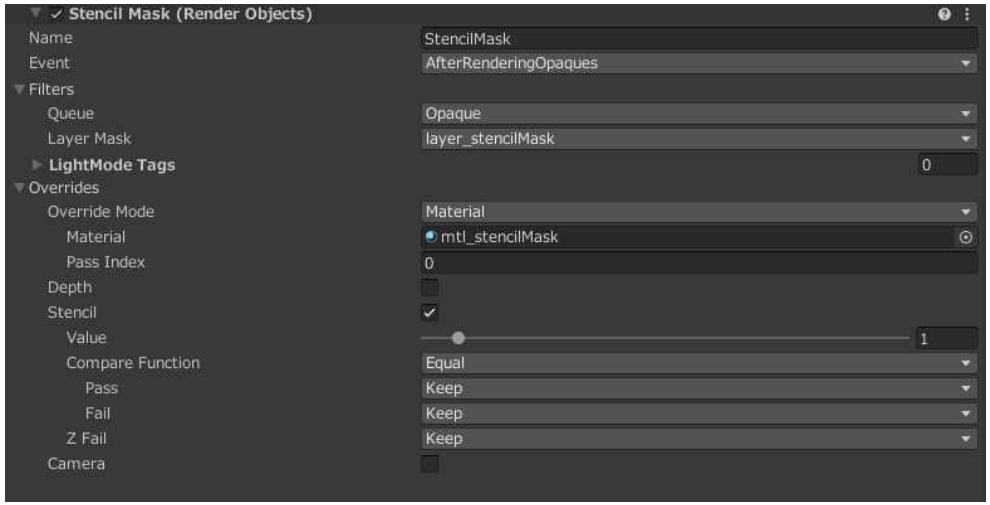
Stencil mask 셰이더는 텍스쳐에 UV 애니메이션을 적용.

Stencil mask로 표시할 대상은 적 오브젝트고, stencil mask가 표시될 위치는 건물 오브젝트.

따라서 적과 건물 오브젝트는 각각의 레이어를 지정하여 reder feature에서 각각의 동작을 설정.



불투명 오브젝트 렌더링 이후에 건물에 해당하는 레이어에는 렌더링 시 stencil 버퍼에 값을 대입.

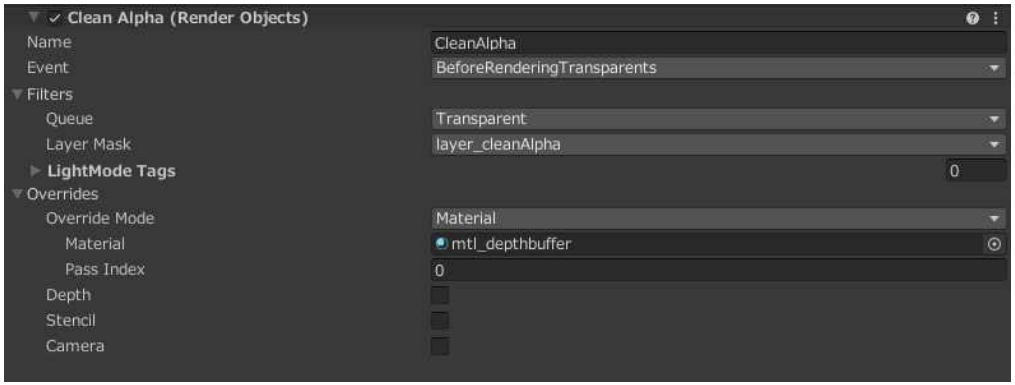


적에 해당하는 레이어는 stencil 버퍼의 값을 비교해서 stencil mask 메타리얼을 렌더링함.

이때 stencil mask 셰이더는 depth 버퍼 상관없이 항상 렌더링하도록 설정.

Custom Shader 제작 - URP

Transparent Depth Buffer

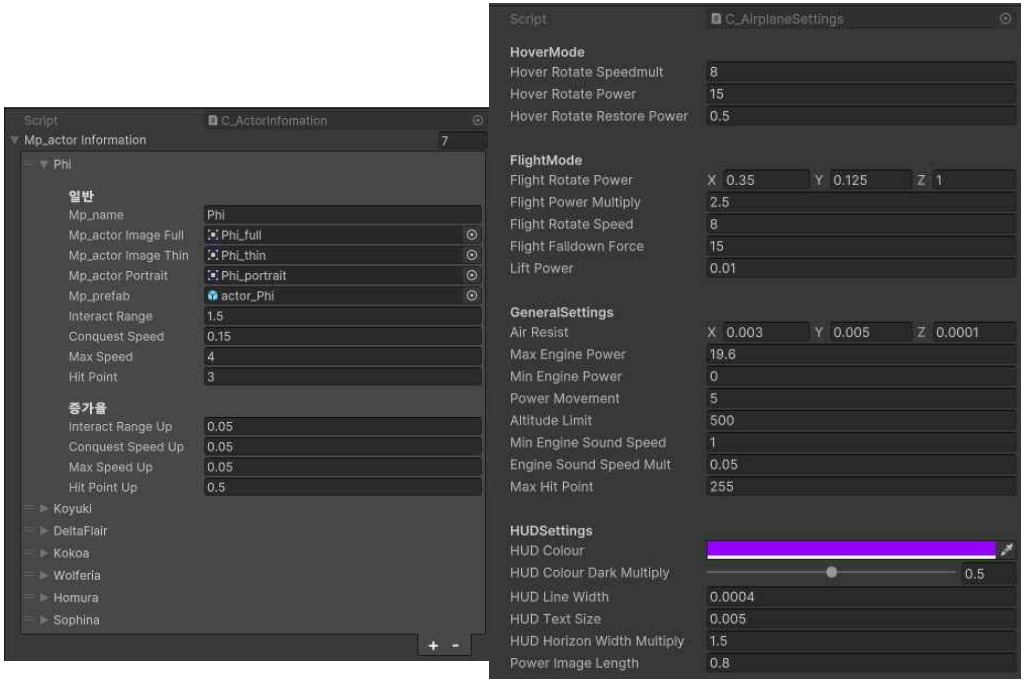


반투명 렌더링 시 z fighting 현상이 발생하는 것 때문에 depth 버퍼에 미리 값을 계산.

반투명을 렌더링하기 전에 해당 레이어에 특정 메타리얼을 적용해서 depth 버퍼를 미리 계산.

이때 이 메타리얼은 오직 depth 버퍼를 계산하기 위한 메타리얼이므로 Scene Color 노드를 이용해서 투명한 것처럼 렌더링.

편리한 정보 수정

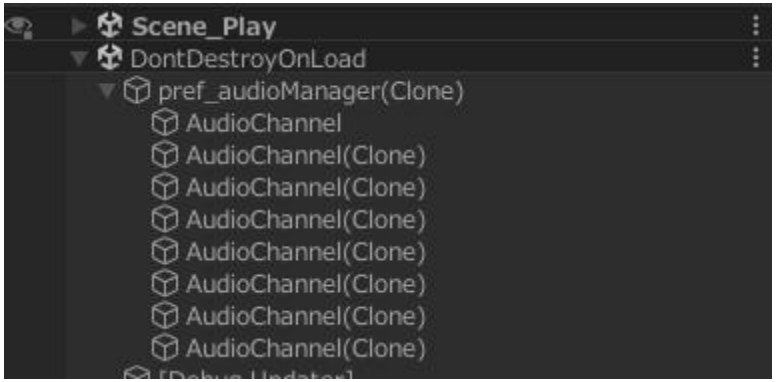


자주 수정할 가능성이 높은 정보는 ScriptableObject로 만들어 유니티 에디터에서 누구나 편리하게 수정 가능.

```
참조 68개
public static class C_Constants
{
    // 수학
    public const float DOUBLE_PI = Mathf.PI * 2.0f;

    // C_Actor 전용, Actor 이동 방향
    public const byte ACTOR_FORWARD = 0b00001;
    public const byte ACTOR_BACKWARD = 0b00010;
    public const byte ACTOR_LEFT = 0b00100;
    public const byte ACTOR_RIGHT = 0b01000;

    // C_Actor 전용, 점령 상태
    public const byte CONQUEST_STANDBY = 0;
    public const byte CONQUEST_START = 1;
    public const byte CONQUEST_PROGRESSING = 2;
    public const byte CONQUEST_CANCEL = 3;

    // C_Airplane 전용, 은폐 활성화
    public const byte STEALTH_ENABLE = 0b01;
    public const byte STEALTH_ANIMATION = 0b10;

    // C_StateHover 전용, 모드 전환 상태
    public const byte HOVER_STANDBY = 0;
    public const byte HOVER_UNAVAILABLE = 1;
    public const byte HOVER_ACTIVATE = 2;

    // C_Enemy 전용, 행동트리 상태
    public const byte ENEMY_BASICACTION = 0;
    public const byte ENEMY_HEAD_TO_ENEMY = 1;
    public const byte ENEMY_ATTACK = 2;

    // C_PlayerBase 전용, 메세지 상태
    public const float MESSAGETIME = 5.0f;
    public const byte PLAYER_HIT = 0B001;
    public const byte PLAYER_HALFHITPOINT = 0B010;
    public const byte PLAYER_LOWHITPOINT = 0B100;

    // C_ObjectPool 전용, 개수 제한
    public const byte LANDFORCELIMIT = 16;
    public const byte OCEANFORCELIMIT = 16;
    public const byte ALLYLIMIT = 16;

    // C_Message 전용, 메세지 상자
    public const float MESSAGEBOX_DELAY = 2.0f;
    public const float MESSAGEBOX_APPEARING_TIME = 0.5f;
    public const float MESSAGEBOX_SCALEMULT_Y = 1.0f / MESSAGEBOX_APPEARING_TIME;

    // C_Message 전용, 메세지 상태
    public const byte MESSAGE_STANDBY = 0;
    public const byte MESSAGE_SHOWING = 1;
    public const byte MESSAGE_DONE = 2;

    // 일반
    public const float DISTANCE_FADE = 500.0f;
    public const float CAMERA_ANGLE = 45.0f;
    public const byte NUM_OF_ACTOR_LIMIT = 3;
}
```

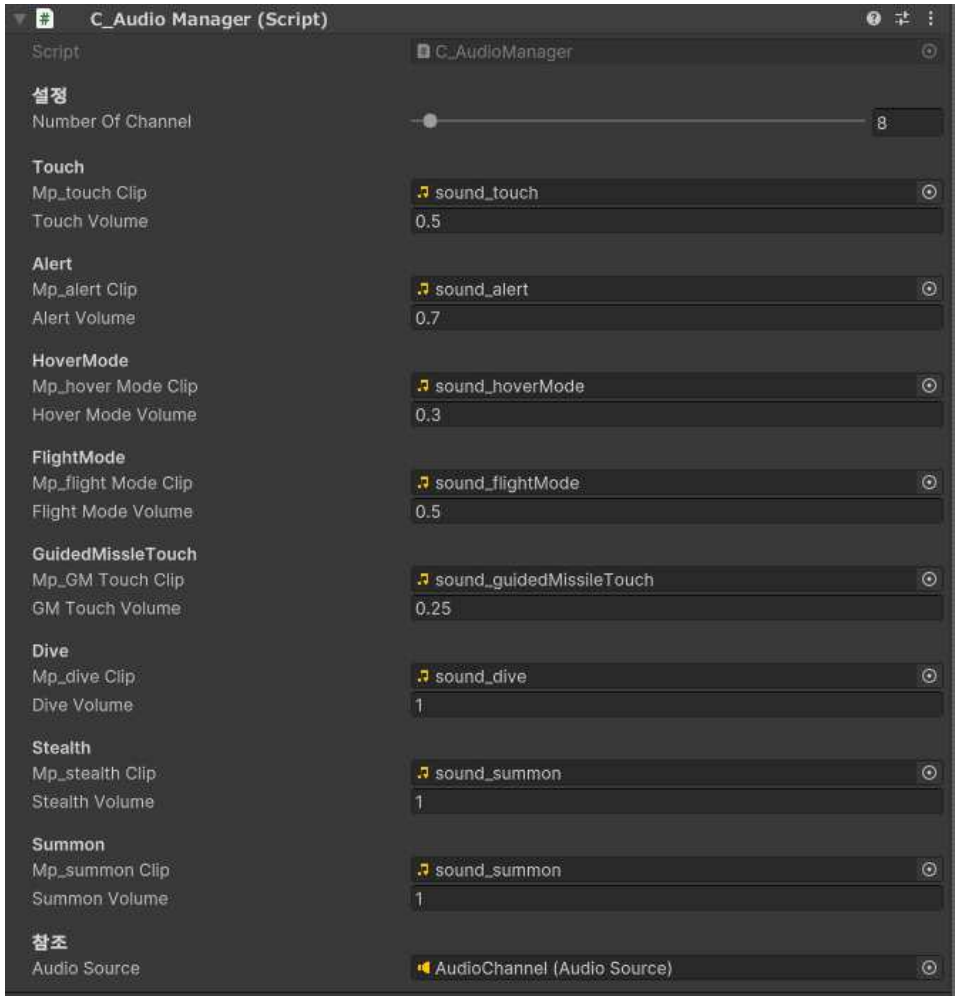
자주 쓰거나 중요한 상수를 모아둔 스크립트가 따로 존재함으로 관리가 용이.

다수의 오디오 채널



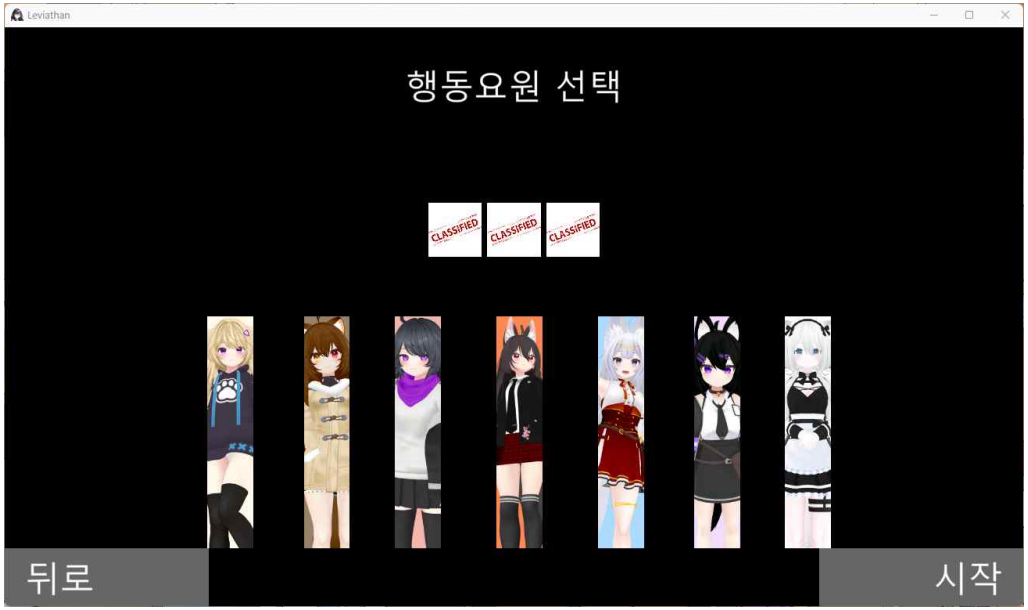
같은 소리를 연속으로 빠르게 재생하거나 소리 재생 중에 씬을 이동하는 경우 재생 중이던 소리가 끊어지는 현상이 발생하는데, 이를 방지하기 위해 여러 개의 소리 채널을 생성.

소리 재생을 위해 AudioManager에 요청하면 AudioManager는 각 채널에 순서대로 소리 클립을 할당하고 재생.



AudioManager는 Prefab으로 만들어져있고 인스펙터 창에서 소리 채널 개수 설정 가능.

시작 전

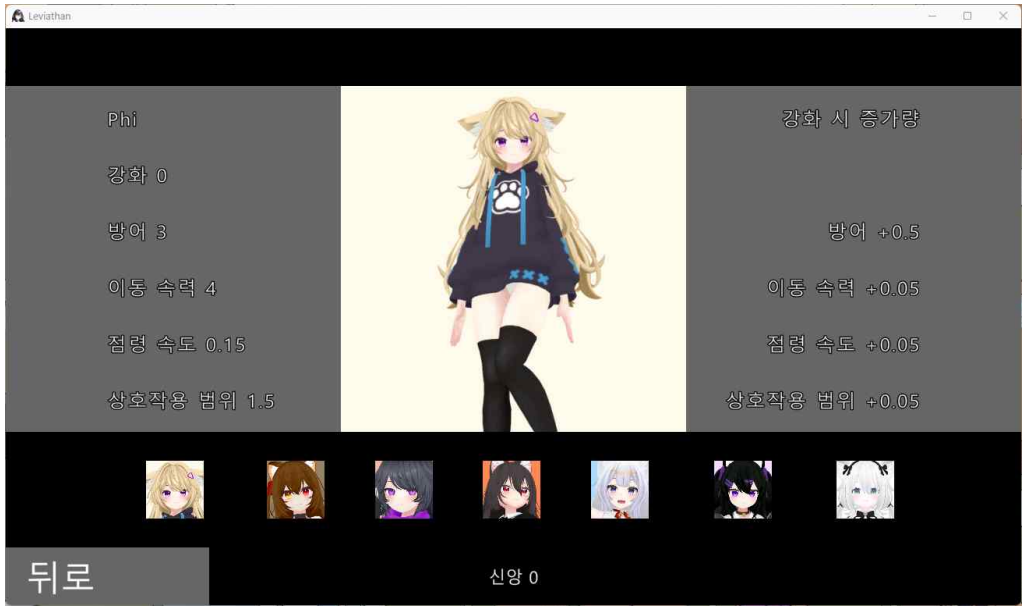


게임 시작 전에 요원을 최대 3명까지 선택.

요원의 이미지를 터치해서 목록에 추가하거나, 이미 추가한 요원의 경우 목록에서 제외할 수 있음.

목록의 이미지를 터치하면 해당 요원을 목록에서 제외.

요원을 목록에 추가 시 좌측부터 빈 공간에 추가하고, 게임 시작 시 목록에 추가된 순서를 그대로 사용.



각 요원은 다른 능력치를 보유.

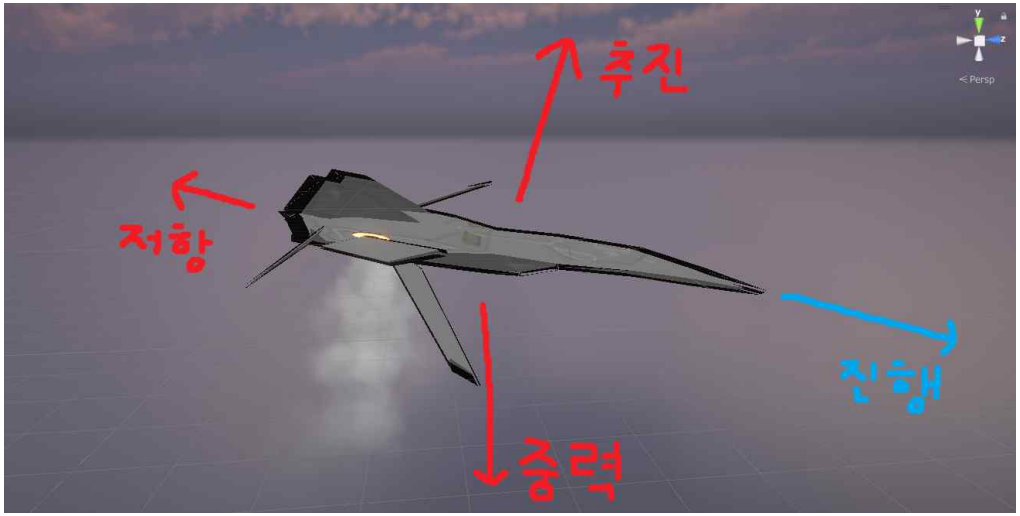
능력치에는 방어, 이동 속력, 거점 점령 속도, 상호작용 범위가 있음.

이러한 능력치는 ScriptableObject로 만들어졌고, 1회 강화 시 능력치가 증가하는 정도 또한 같은 ScriptableObject에 있음.

따라서 강화 횟수만 저장하면 총 능력치가 결정되므로, 나중에 밸런스 패치 등으로 수정하는 일이 발생하더라도 능력치가 훼손되는 일은 없음.

비행기의 움직임

플레이어가 조종할 비행기에는 수직이착륙모드와 비행모드가 있음.



수직이착륙 시, 기체의 위 방향으로 추진과 속력에 따른 저항의 합력에 의한 가속도에 기체의 회전을 곱한 것에
월드 좌표 기준 수직 방향으로의 중력 가속도를 합한 것으로 진행 방향과 속력을 계산.

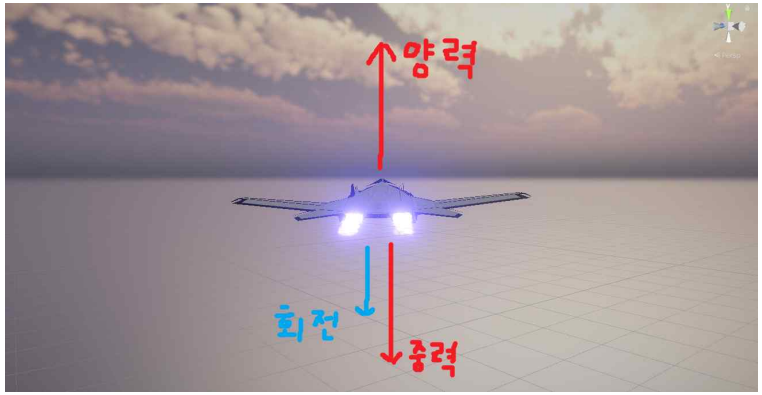
비행 시 추진 방향은 기체의 앞 방향이고 나머지는 모두 동일하게 계산.

저항 계산 시, 변수가 되는 것은 속력이고, 나머지 요소는 게임 제작 시에 계산하여 전후, 좌우, 상하 방향으로의 계산 값을 미리 설정해 둠.

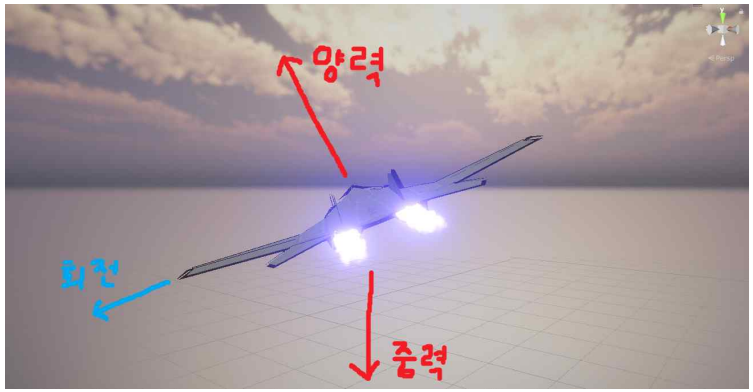
비행 시에는 양력을 적용.

현실에서 양력은 기체 위 방향으로 가속을 주는 것이나, 여기서는 단순하게 기체를 회전시키기만 하는 것으로 제작.

양력에 의한 회전은 날개의 상하 방향에만 영향을 줌.



기체가 바르게 비행 중일 때 속력이 작을수록 아래로 회전.



기체가 기울여서 비행 중일 때는 양력에 의한 상하 회전이 영향력이 적으므로 측면으로 회전하기 시작함.

비행 HUD

비행기 조종 시에 보이는 HUD는 수평선과 각도 등 실제 보이는 화면과 맞추기 위해 화면 크기에 따라 실시간으로 변하도록 제작.

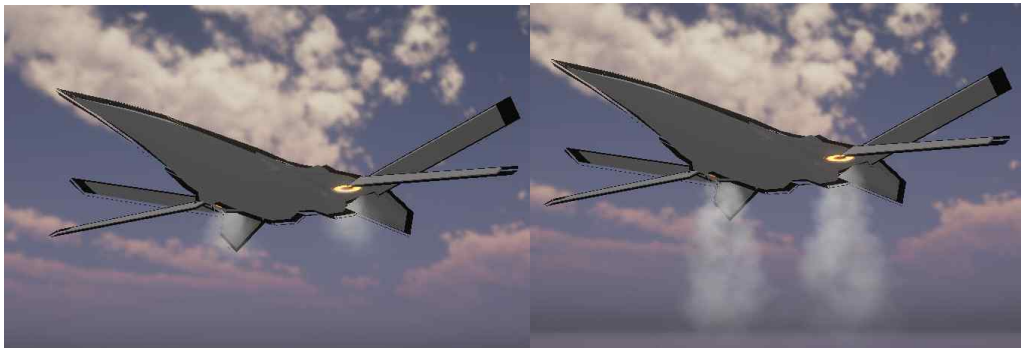
안드로이드 빌드 시에는 창 크기가 변할 일은 없기 때문에 #if 전처리기를 이용해서 윈도우 빌드 시에만 해당 기능이 빌드되도록 함.

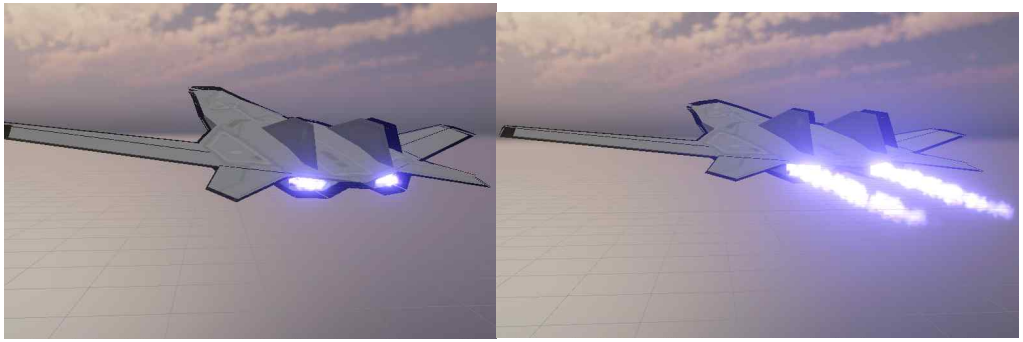
각도 표시 HUD는 기체의 진행 방향을 표시하는 아이콘을 중앙에 두도록 함.

따라서 각도 표시 HUD가 진행 방향에 따라 좌우로 움직일 수 있음.

화면 크기가 변하더라도 각도는 올바르게 표시.

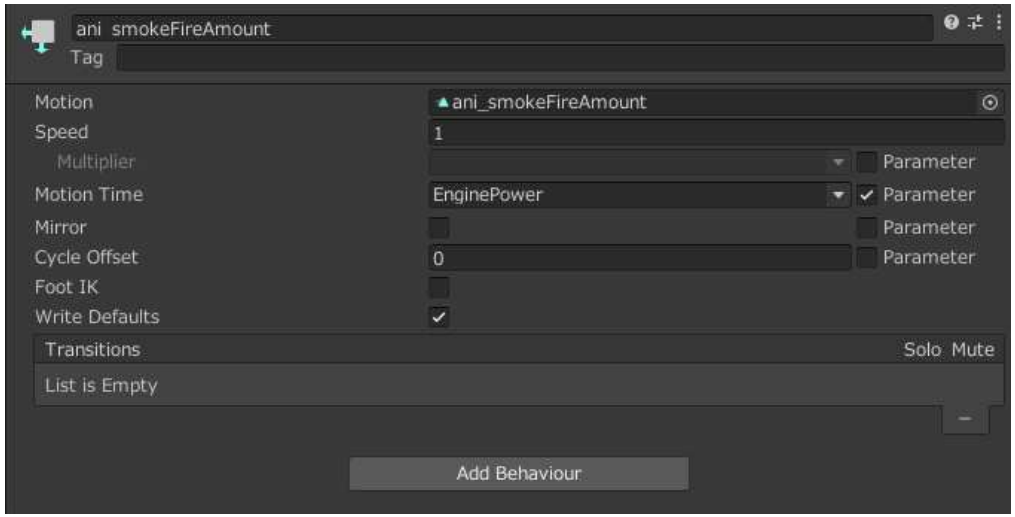
비행 효과





엔진에서 나오는 연기나 불꽃은 파티클로 제작.

애니메이션 클립을 만들어서 0초에서는 파티클 양을 최소, 1초에서는 파티클 양을 최대로 설정.



Animator controller에서는 이 애니메이션 클립에 motion time을 적용.

스크립트에서는 이 animator에서 motion time에 사용할 변수에 값을 전달.

가이드미사일 화면

비행기가 수직이착륙 모드일 때 가이드미사일 화면에 들어올 수 있음.

이 화면에 들어오면 post process의 colour adjustment가 수정되고, 화면에서 노이즈 이미지가 무한 스크롤해서 카메라같은 느낌을 형성.

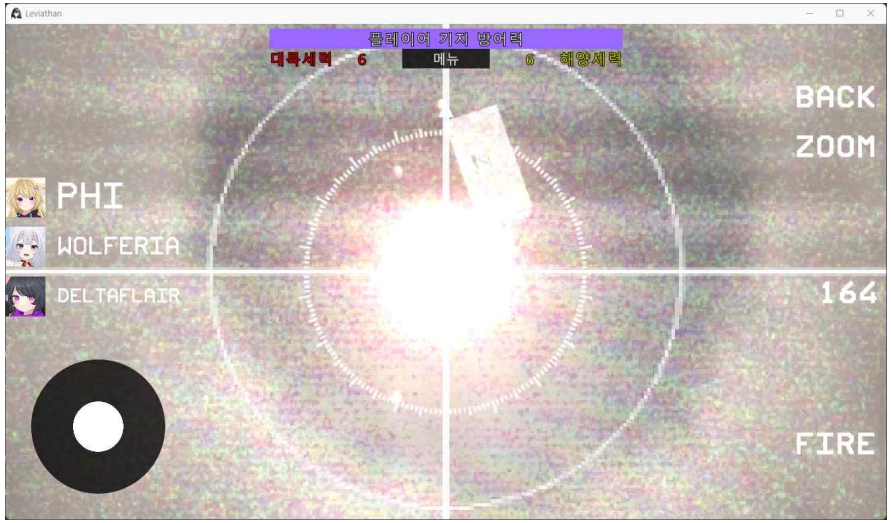
플레이어는 카메라는 보안카메라처럼 움직이고 원하는 방향으로 미사일을 발사.

좌측에는 게임 시작 전 선택했던 행동요원을 배치할 버튼이 존재.

이 버튼을 터치하면 Physics 클래스의 Raycast를 통해 화면 정중앙에 보이는 위치에 해당 Actor(행동요원)를 배치.

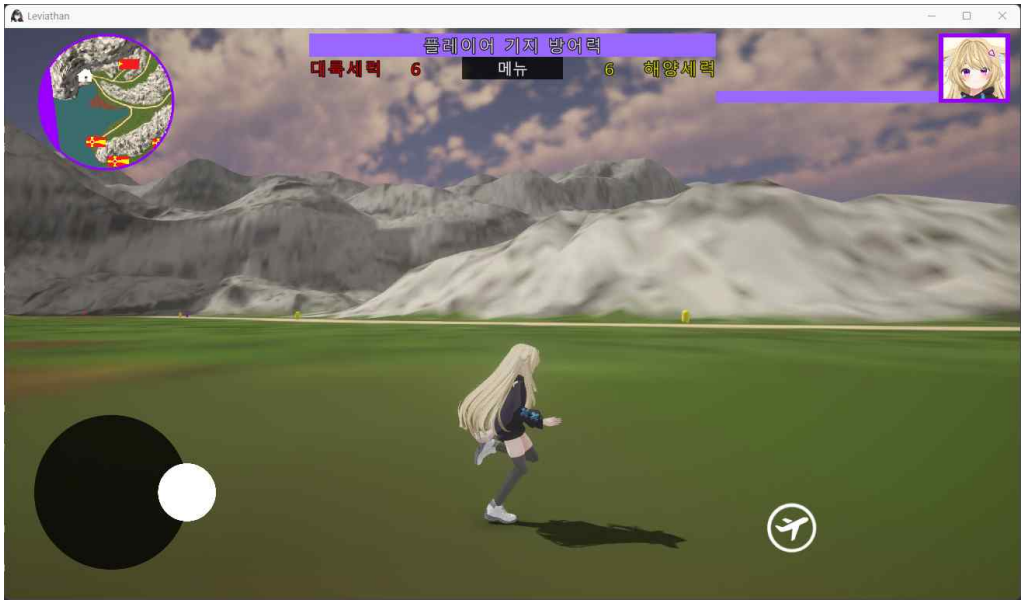
미사일이 발사되는 중에도 방향 조종 가능.

미사일 폭발은 주변 적을 제거.



폭발 효과는 프로그램 실행하는 동안 나타났다 사라지기를 반복하기 때문에 ObjectPool을 사용.

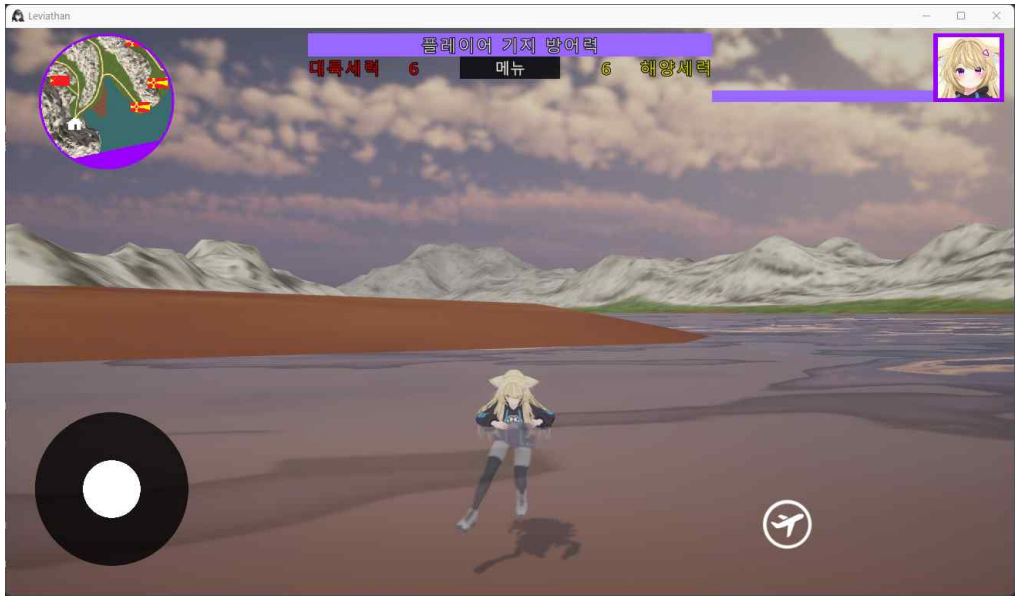
Actor의 움직임



Actor의 움직임 애니메이션은 서있기, 걷기, 뛰기가 있고, 이는 모두 BlenderTree로 연결되어 있음.

따라서 움직이지 않을 때는 서있기, 천천히 움직일 때는 걷기, 빠르게 움직일수록 뛰기 애니메이션을 더 크게 볼 수 있음.

카메라를 회전시키기 위해서는 화면을 드래그하거나 위 사진처럼 Actor를 화면의 측면으로 움직여서 카메라를 회전.



Actor가 물에 닿으면 수영 애니메이션을 동작.

Actor가 육지에 있을 때 물 감지를 하고, 물에 있을 때는 육지 감지를 실시.

Actor가 물에 있는 동안에는 Actor의 y 위치는 물의 y위치와 동일.

걷기, 수영할 때의 소리 효과는 해당 애니메이션의 AnimationEvent로 등록하여 애니메이션 동작에 맞춰 소리 재생.

Actor의 동작

Actor는 적 근처에서 던지기 공격을 함.

던지기 공격은 포물선 궤적을 그리는데, 수평 방향으로는 등속운동, 수직 방향으로는 등가속도운동을 하기 때문에 궤적은 이차함수 그래프의 모습으로 나타남.

$$y = ax^2 + bx + C \qquad Y_1 = aX_1^2 + bX_1 + Y_0$$

$$Y_0 = C \qquad -b = \frac{aX_1^2 + Y_0 - Y_1}{X_1}$$

$$= aX_1 + \frac{Y_0 - Y_1}{X_1}$$

$$\frac{dy}{dx} = 2ax + b$$

$$\frac{d^2y}{dx^2} = 2a$$

위는 시작 지점을 0으로 가정했을 때 목적 지점까지 이어지는 이차함수 그래프를 구할 공식.

여기서 수직 방향으로의 이동을 중력이 작용했을 때와 일치시키기 위해서는 수평 방향으로의 이동 속력에 따라 최고차항의 계수가 변화해야 함.

최고차항계수 a는 항상 음수여야 하기 때문에 수평 방향으로의 이동 속력이 증가할수록 a는 더 작은 수가 되어야 함.

이 수치는 유니티 에디터에서 설정 가능.

적 거점이 상호작용 범위 안에 있을 때는 거점 정령 가능.

점령 시 적 세력은 거점을 잃고 그곳에서 아군 유닛이 생성됨.

조종 대상 및 상태 전환

플레이어가 조종할 수 있는 것은 비행기, 가이드미사일, Actor가 있음.

각 조종 대상은 상태패턴을 적용하여 상태 클래스로 제작했고, C_PlayManager가 상태기계 역할을 함.

C_PlayManager가 상태 클래스의 Update, FixedUpdate 함수를 호출하기도 하나, 동작 순서를 정할 필요가 있는 경우 대리자를 통해 호출 순서를 정함.

대리자를 사용하는 이유는 상태 클래스가 아닌 것에도 동작 순서가 필요하기 때문.

C_Airplane은 C_PlayManager의 상태 클래스이면서 동시에 비행기 내부에서 상태기계가 됨.

비행 모드로는 수직이착륙 상태와 비행 상태, 은폐 모드로는 은폐 상태와 미은폐 상태가 있음.

비행 모드는 마찬가지로 상태 클래스를 제작하여 Update와 FixedUpdate의 동작을 정의했으나, 은폐 모드는 그보다 훨씬 짧고 간단한 동작이기 때문에 switch문을 사용하는 것으로 충분함.

C_GuidedMissile은 C_PlayManager의 상태 클래스면서 동시에 가이드미사일 내에서는 상태기계가 됨.

미사일 발사 상태와 발사 대기 상태가 있는데, 필요한 동작은 C_GuidedMissile의 멤버 함수로 정의했고, 그렇게 정의된 함수는 switch 문에서 호출하는 것으로 충분함.

C_Actor는 C_PlayManager의 상태 클래스면서 동시에 Actor에서는 상태기계가 됨.

소환 중 상태, 대기 중 상태, 거점 점령 중 상태, 복귀 중 상태가 있음.

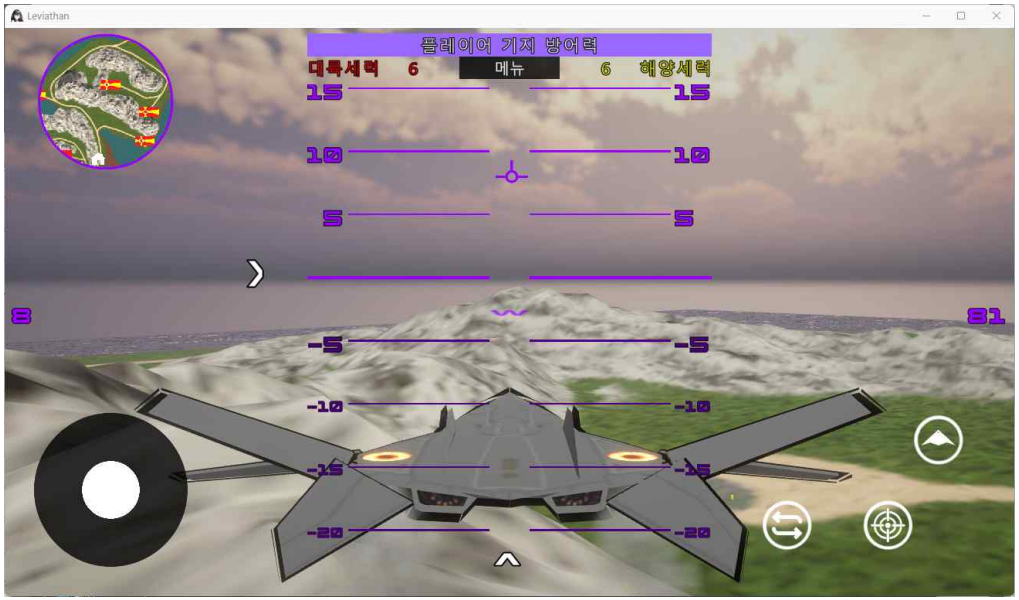
C_Airplane 만큼 복잡하지는 않으나 C_GuidedMissile 만큼 간단하지는 않음.

따라서 위 상태에 대한 상태 클래스를 제작하거나 switch 문으로 간단가게 구현하는 방법을 사용하지 않고, 대신 대리자를 이용함.

대리자의 배열을 생성하고 각 상태에 맞는 인덱스에 해당 동작을 등록.

그러면 상태에 따라 올바른 인덱스에 접근해서 동작을 수행.

UI 버튼과 UI 조이스틱



화면 우측 하단에 있는 UI 버튼은 안드로이드에서 터치, 윈도우에서는 마우스 클릭 혹은 키보드 단축키로 동작.

안드로이드와 윈도우에서의 동작은 #if와 #elif 전처리기를 사용해서 빌드 시 각각 다르게 빌드되도록 제작.

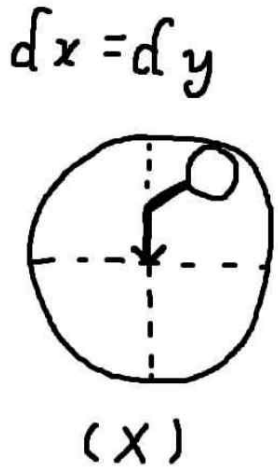
화면 좌측 하단에 있는 UI 조이스틱은 기본적으로 안드로이드 플랫폼을 위한 것이나, 윈도우에서도 마우스 드래그로 사용 가능.

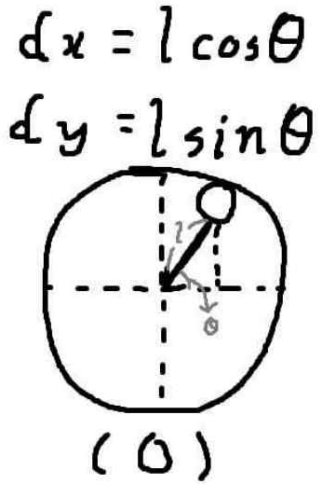
윈도우에서 키보드 방향키 사용하면 그 방향대로 UI 조이스틱이 움직임.

조이스틱에서 손을 놨을 때 조이스틱은 자동으로 원위치로 돌아감.

이때 원위치로 순간이동하는 것이 아니라 서서히 자리로 돌아감.

돌아가는 방향은 정확히 원위치를 보는 방향으로 하기 위해 조이스틱의 x, y 좌표에 대해 아래와 같은 속력을 가짐.

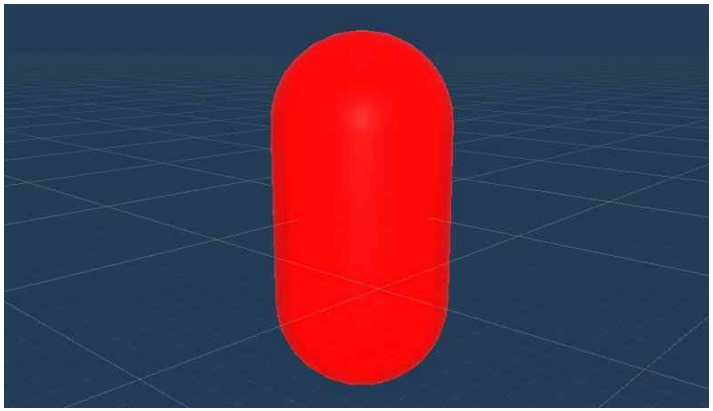




UI 조이스틱은 C_Joystick 클래스를 통해 작동하고, 이 클래스를 참조하면 정규화된 x, y 좌표와 조이스틱의 거리를 얻을 수 있음.

조이스틱은 prefab으로 만들어두었고 필요할 때 언제든 그 prefab을 가져다 사용할 수 있음.

적 개체

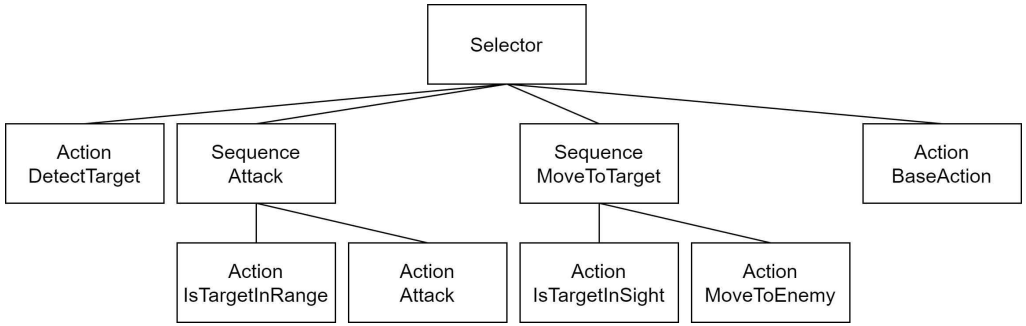


적 개체는 아직까지 전용 모델이 없고 유니티 에디터에서 제공하는 capsule을 사용.

적은 거점 근처를 순찰하는 개체와 플레이어의 기지로 이동하는 개체가 존재.

적의 행동은 행동트리로 구현.

공통 동작은 부모 클래스가 익명함수로 정의하고 개별 동작은 자식 클래스가 추상화 함수를 정의하는 것으로 함.



실행 순서는 좌측에서 우측 순서.

DetectTarget은 일정 범위 내 목표가 있으면 그 목표를 참조하고, 없으면 참조 값을 null로 한 후 Running 반환.

IsTargetInRange는 목표가 공격 범위 안에 있으면 True, 아니면 False 반환.

IsTargetInSight는 목표의 참조가 null이 아니면 True, null이면 False 반환.

여기까지의 Action 노드의 동작은 익명함수로 정의하고, BaseAction에 해당하는 동작은 추상화 함수를 호출하는 것으로 해서 자식 클래스에서 정의된 동작을 수행.

BaseAction에서 순찰용 개체는 거점 주변을 순찰하고, 공격용 개체는 플레이어의 기지로 이동.

실제 동작을 수행하는 Action 노드는 기본적으로 그 개체의 NavMesh의 목적지를 설정해서 개체를 이동시키는데, 목적지 설정에는 많은 연산이 들어가기 때문에 해당 노드로 상태가 변경되는 시점에만 새로운 목적지를 설정하도록 제작.

ObjectPool

적 생성, 폭발 효과 같은 것은 나타났다가 사라지기를 반복하기 때문에 ObjectPool을 적용.

C_ObjectPool은 필요한 prefab을 참조하고 있고 누군가가 어떤 오브젝트를 원할 때 그 prefab의 복사본을 반환.

복사본은 스택에 저장되어있고, 누군가가 오브젝트를 요청할 때 스택에 남은 오브젝트가 없으면 새로 복사본을 만들어서 반환.

여러 종류의 복사본을 저장하기 위해 스택의 배열을 만들어 각 인덱스별로 다른 종류의 복사본을 저장.

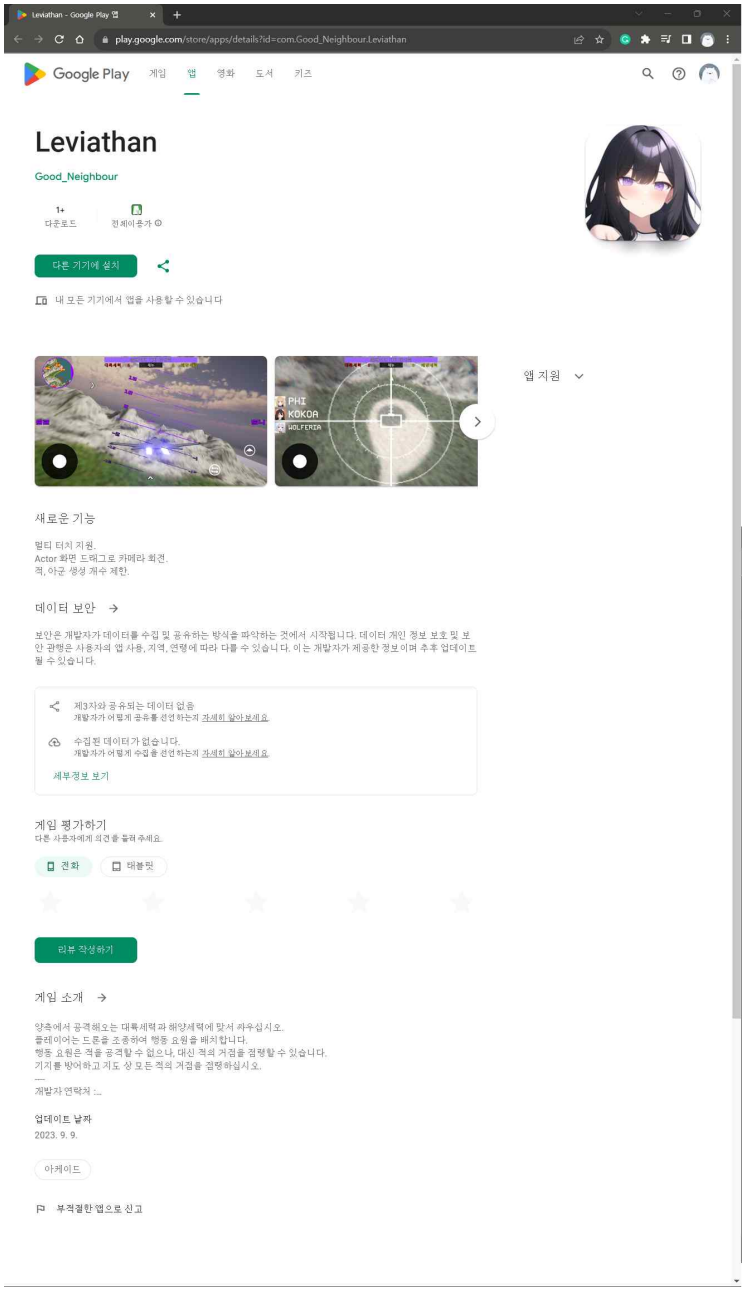
C_ObjectPool이 하는 역할은 오직 누군가가 오브젝트를 요청했을 때 복사본을 반환하는 것과 반납받은 오브젝트를 스택에 저장하는 것.

C_ObjectPool에서 오브젝트를 가져올 때와 반납할 때의 동작은 전적으로 호출한 측에서 담당.

C_ObjectPool이 복사한 복사본의 수에 제한을 두어서 복사본 수다 제한에 도달하면 null을 반환.

호출한 측에서는 null 체크를 통해 동작을 달리 함.

출시 및 디버깅



다양한 사람들에게 게임 테스트를 요청할 목적으로 Google Play Store에 게임 출시.

이를 통해 테스트 요청 시 앱이 안전하다는 것을 증명할 수 있고, 무언가 업데이트가 있을 때마다 일일이 업데이트 버전을 보내줄 필요 없이 Google Play Store에서 업데이트 가능.