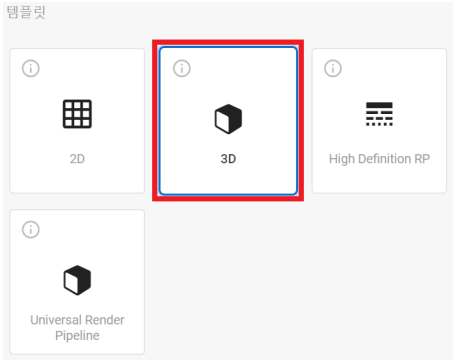
Unity Zombie 上

참고:레트로의 유니티 게임 프로그래밍 에센스

목차

프로젝트 준비



새 프로젝트 생성

- 새로 만드는 **프로젝트**의 템플릿을 **3D**로 선택

리소스 추가

- 프로젝트에 **Resources 폴더를 만든다**
- **예제소스 폴더 => Zombie 폴더**에 있는 **Scripts 폴더를 제외**한 **모든 폴더 및 파일**을 전부 **Resources 폴더로 복사**

Hierarchy 기본 설정

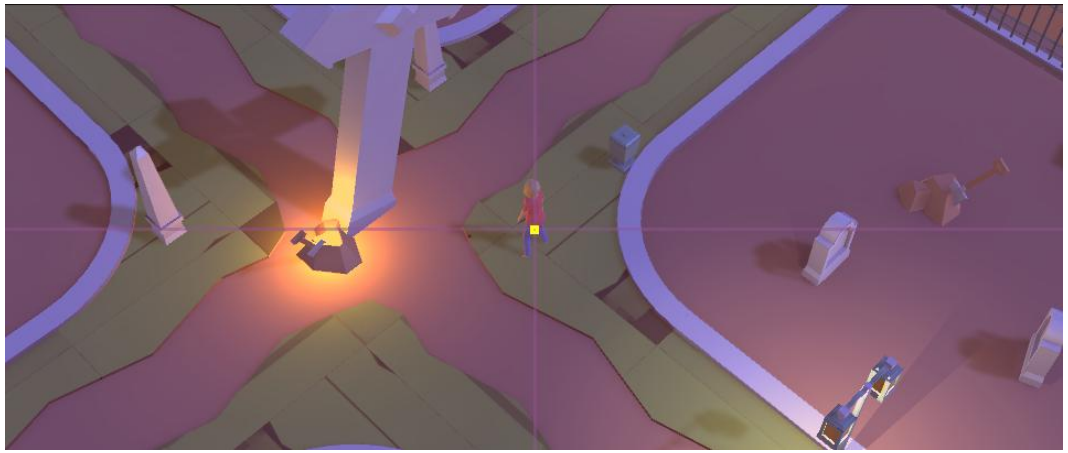
- **게임 개발 용어**로 레벨은 여러 의미가 있으며, **Map을 Level이라 칭한다**
- Prefabs=>Level Art를 Hierarchy에 등록
- Models=>Woman을 Hierarchy에 등록
- **Woman 오브젝트**에 **Capsule Collider 추가**
- Capsule Collider
  - Center : (0, 0.75, 0)
  - Radius : 0.25
  - Height : 1.45

Cinemachine

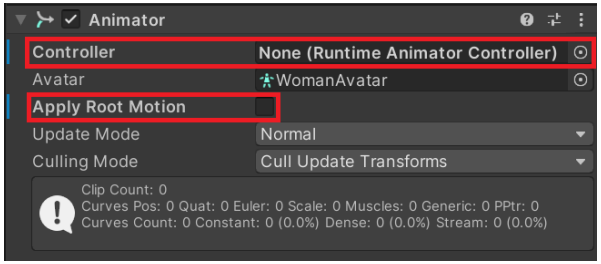
- **Package Manager**에서 **Cinemachine**을 **설치**

플레이어 이동

- **PlayerCamera(CinemachineVirtualCamera)**
- Cinemachin=>Create Virtual Camera
- Follow, Look At : Woman
- Lens
  - Field Of View : 20
- Body : Transposer
  - Binding Mode : World Space
  - Follow Offset : (-8, 16, -8)
  - X, Y, Z Damping : 0.1
- Aim : Composer
  - Tracked Object Offset : (0, 0.5, 0)
  - Lookahead Smoothing : 10
  - Horizontal, Vertical Damping : 0
  - Soft Zone Width, Height : 0



플레이어 이동

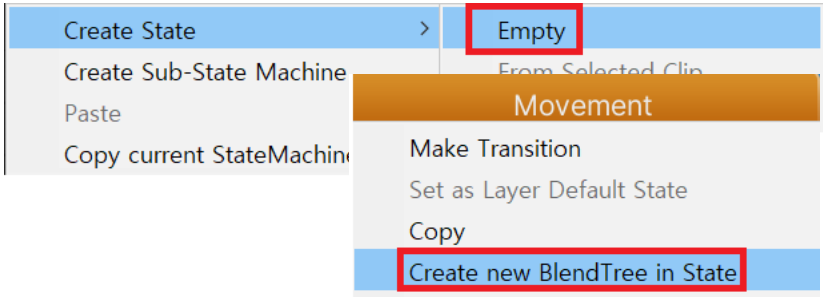


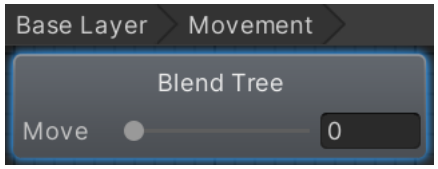
Animator

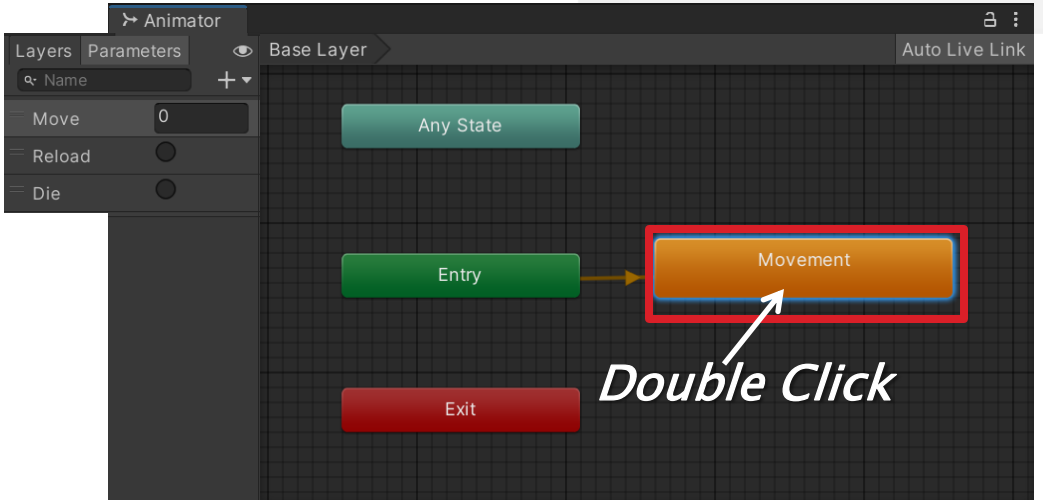
- **Woman**의 Animator에 **Animator Controller(Player)**를 **새로 만들어서 추가**
- **Apply Root Motion**(애니메이션에 있는 좌표이동, 회전을 오브젝트에 적용)의 **체크를 해제**

Animator Controller(Player)

- **Parameters**
  - Move(float) : 움직임 값을 받아서 애니메이션을 제어
  - Reload(Trigger) : 총알 재장전 애니메이션을 알리는 키
  - Die(Trigger) : 체력이 없을 경우 애니메이션을 알리는 키
- Create State=>Empty(Movement)
- **Movement**에 **Blend Tree**를 **추가**
- **Movement(State)**를 **더블 클릭**

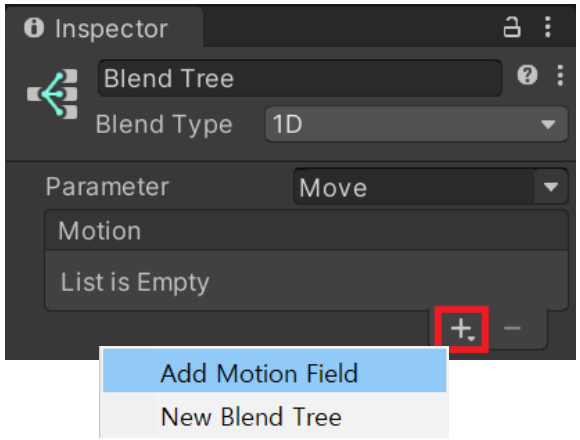


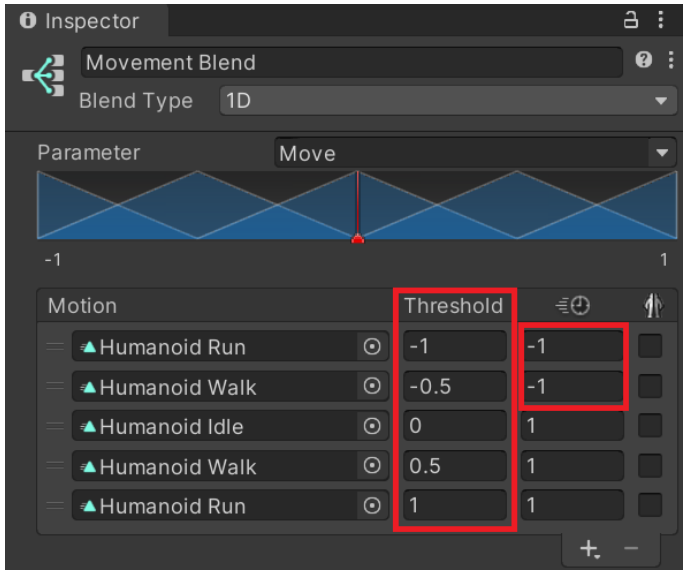




플레이어 이동

- **Movement(State) Blend Tree**
  - Blend Tree의 이름을 Movement Blend로 변경
  - **Motion List**에 **5개의 Motion을 추가**
  - **순서대로** Humanoid Run, Walk, Idle, Walk, Run **애니메이션 클립**을 **등록**
  - **Move(Parameter)**의 값을 **Input.GetAxis() 값을 참조**하여 쓰도록 구현을 하기 때문에 Parameter의 영향을 받는 **Threshold를 -1 ~ 1 범위의 값**으로 지정
  - **처음 두** Run, Walk **애니메이션의 속도(시계모양 아이콘)를 -1로 변경**
  - 애니메이션의 **속도가 음수**가 되면 **애니메이션은 역으로 재생**된다





플레이어 이동

- **PlayerInput(Script)**
  - C# **PlayerInput** 스크립트 **생성**
  - **Woman 오브젝트**에 **추가**
  - Player의 **Input**을 **갱신** 처리
  - Input 결과를 PlayerMovement에서 사용
  - PlayerMovement에서 자동으로 당 스크립트를 추가하기 때문에 직접 추가할 필요는 없다

```
public class PlayerInput : MonoBehaviour
{
    [SerializeField] private string moveAxisName = "Vertical";
    [SerializeField] private string rotateAxisName = "Horizontal “;

    public float move { get; private set; }
    public float rotate { get; private set; }

    void Update()
    {
        move = Input.GetAxis(moveAxisName);
        rotate = Input.GetAxis(rotateAxisName);
    }
}
```

플레이어 이동

- **PlayerMovement(Script)**
  - C# **PlayerMovement** 스크립트 **생성**
  - **Woman 오브젝트**에 **추가**
  - Rigidbody와 PlayerInput을 자동으로 추가하도록 한다
  - PlayerInput 값을 참조
  - **Player**를 **이동, 회전** 한다

```
[RequireComponent(typeof(Rigidbody))]
[RequireComponent(typeof(PlayerInput))]
public class PlayerMovement : MonoBehaviour
{
    [SerializeField] private float speed = 4f;

    private PlayerInput playerInput;
    private Rigidbody rigid;

    private Animator anim;

    void Start()
    {
        playerInput = GetComponent<PlayerInput>();
        rigid = GetComponent<Rigidbody>();
        if (rigid) rigid.constraints = RigidbodyConstraints.FreezeRotation;

        anim = GetComponent<Animator>();
    }
```

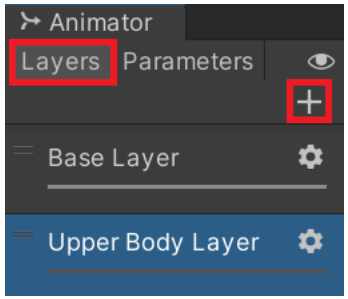
플레이어 이동

```
    private void Move()
    {
        if (playerInput && rigid)
        {
            Vector3 velocity = transform.forward * playerInput.move * speed;
            rigid.velocity = velocity; //또는, rigid.MovePosition(rigid.position + velocity * 0.01f);
        }
    }

    private void Rotate()
    {
        if (playerInput && rigid)
        {
            float angle = playerInput.rotate * speed;
            rigid.rotation *= Quaternion.Euler(0, angle, 0);
        }
    }

    void FixedUpdate()
    {
        Move();
        Rotate();

        if (anim && playerInput) anim.SetFloat("Move", playerInput.move);
    }
}
```

Gun Shoot, Reload

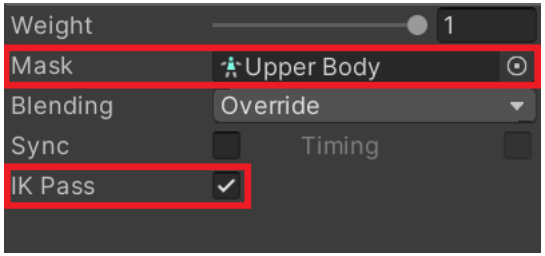
Layers

- **여러 상태**(애니메이션)를 **동시에 작동**하게 한다
- 애니메이션의 **동작이 겹치는 부분**이 있다면 **가장 아래의 Layer로 덮어씌운(Blend)다**
- **가중치(Weight)**가 높을 수록 **덮어씌우(Blend)는 정도**가 높아 진다
- **Avatar Mask**를 설정하여 **특정 부위의 애니메이션만을 적용**할 수 있다



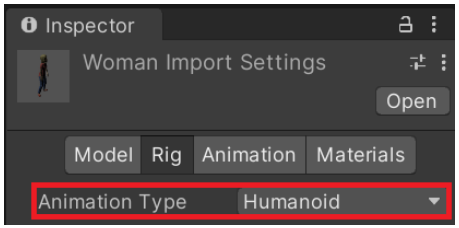
Upper Body Layer

- **새 Layer 추가**
- **Aim Idle 애니메이션 clip 등록**
- 설정(톱니바퀴)
  - Weight : 1
  - **새 Avatar Mask**(Upper Body)를 만들어 **Mask**에 **등록**
  - IK Pass : **true**



역운동학(Inverse Kinematics)

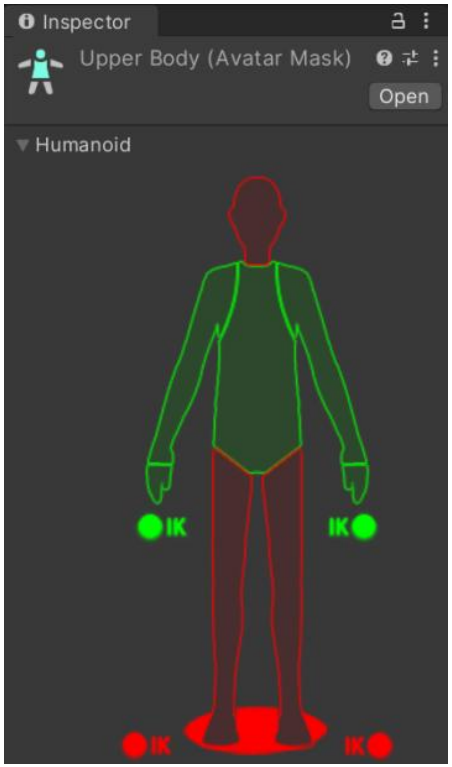
- **특정 위치를 중심으로 관절의 위치를 역으로 계산**
- **오브젝트의 위치**가 어디에 있든, 거기에 맞추어 **시선이나 팔 등의 위치를 결정**할 때 사용
- **Animation Type**이 **Humanoid에서만 사용 가능**

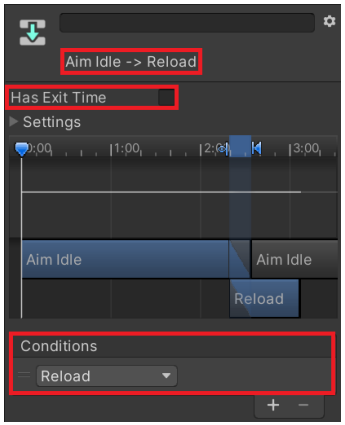


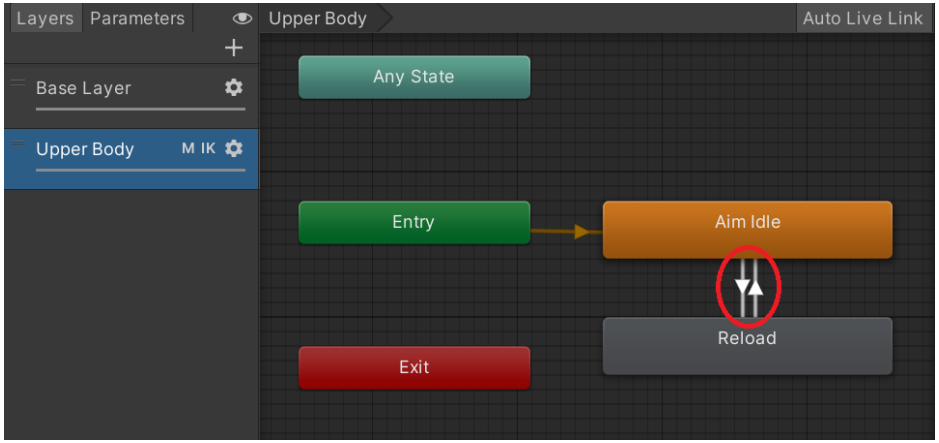
Gun Shoot, Reload

- **Avatar Mask(Upper Body)**
  - **두 팔**과 **상체**, **두 손** 그리고 **두 손의 IK**를 선택

- **Upper Body Layer**
  - **Reload 애니메이션 clip 등록**
  - **Aim Idle**과 **Reload** 서로 **Transition으로 연결**
  - **Aim Idle->Reload** Conditions : **Reload(Trigger)**
  - **Aim Idle->Reload** Has Exit Time : **false**



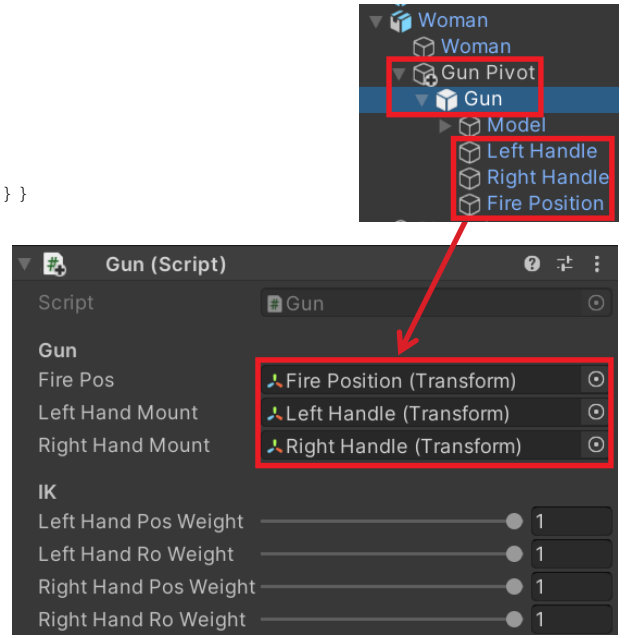




Gun Shoot, Reload

- **Gun(Script)**
  - **Woman**의 **Gun Pivot**에 **Prefabs 폴더**의 **Gun(GameObject)**을 **추가**
  - **Gun(GameObject)에 Gun(Script) 추가**
  - IK에 이용할 **관절이 위치할 좌표**를 지정

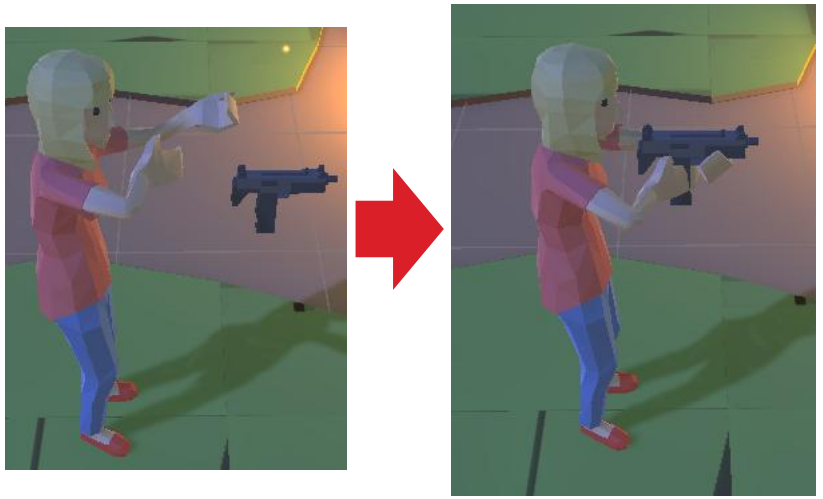
```
public class Gun : MonoBehaviour
{
    private Transform pivot;

    [Header("Gun")] // Inspector에 표시, 해당 Data들의 사용처를 알려준다.
    [SerializeField] private Transform firePos; // 총알 발사 위치.
    [SerializeField] private Transform leftHandMount; // 왼손 위치 지점.
    [SerializeField] private Transform rightHandMount; // 오른손 위치 지점.

    [Header("IK")] //[Range(a, b)] : 값의 범위를 제한(a~b).
    [SerializeField] [Range(0, 1)] private float leftHandPosWeight = 1f;
    [SerializeField] [Range(0, 1)] private float leftHandRoWeight = 1f;
    [SerializeField] [Range(0, 1)] private float rightHandPosWeight = 1f;
    [SerializeField] [Range(0, 1)] private float rightHandRoWeight = 1f;

    public Vector3 Pivot { get { return (pivot) ? pivot.position : Vector3.zero; } set { if (pivot) pivot.position = value; } }
    public Vector3 LeftHandMountPos { get { return (leftHandMount) ? leftHandMount.position : Vector3.zero; } }
    public Quaternion LeftHandMountRo { get { return (leftHandMount) ? leftHandMount.rotation : Quaternion.identity; } }
    public Vector3 RightHandMountPos { get { return (rightHandMount) ? rightHandMount.position : Vector3.zero; } }
    public Quaternion RightHandMountRo { get { return (rightHandMount) ? rightHandMount.rotation : Quaternion.identity; } }

    public float LeftHandPosWeight { get { return leftHandPosWeight; } }
    public float LeftHandRoWeight { get { return leftHandRoWeight; } }
    public float RightHandPosWeight { get { return rightHandPosWeight; } }
    public float RightHandRoWeight { get { return rightHandRoWeight; } }

    private void Awake()
    {
        pivot = transform.parent;
    }
}
```



Gun Shoot, Reload

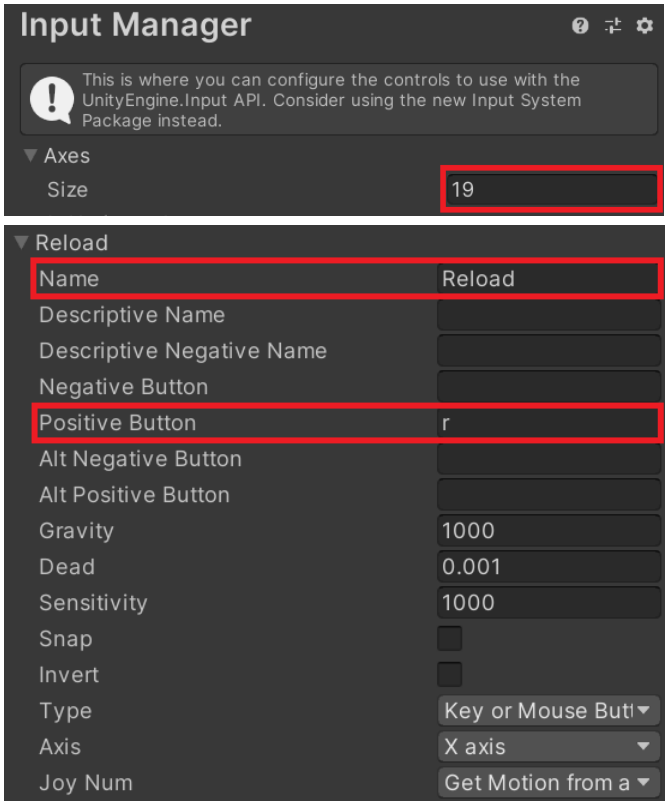
- **PlayerShooter(Script)**
  - C# **PlayerShooter** 스크립트 **생성**
  - **Woman 오브젝트**에 **추가**
  - Gun(Script)에서 **IK**에 사용할 값을 받아, **총과 플레이어의 손의 위치를 설정**

```
public class PlayerShooter : MonoBehaviour
{
    [SerializeField] private Gun gun;
    private Animator anim;

    private void Start()
    {
        anim = GetComponent<Animator>();
    }

    private void OnAnimatorIK(int layerIndex)
    {
        if (gun && anim)
        {
            // 해당 오브젝트(Gun)의 pivot을 해당 애니메이션(upper body)의 오른쪽 팔꿈치 위치로 이동.
            gun.Pivot = anim.GetIKHintPosition(AvatarIKHint.RightElbow);

            // 왼손의 position, rotation을 해당 오브젝트(Gun)의 왼쪽 손잡이 위치에 맞춘다.
            anim.SetIKPosition(AvatarIKGoal.LeftHand, gun.LeftHandMountPos);
            anim.SetIKRotation(AvatarIKGoal.LeftHand, gun.LeftHandMountRo);
            // 가중치(weight)를 추가하여 위치, 회전을 미세조정 한다.
            anim.SetIKPositionWeight(AvatarIKGoal.LeftHand, gun.LeftHandPosWeight);
            anim.SetIKRotationWeight(AvatarIKGoal.LeftHand, gun.LeftHandRoWeight);

            // 오른손의 position, rotation을 해당 오브젝트(Gun)의 오른쪽 손잡이 위치에 맞춘다.
            anim.SetIKPosition(AvatarIKGoal.RightHand, gun.RightHandMountPos);
            anim.SetIKRotation(AvatarIKGoal.RightHand, gun.RightHandMountRo);
            // 가중치(weight)를 추가하여 위치, 회전을 미세조정 한다.
            anim.SetIKPositionWeight(AvatarIKGoal.RightHand, gun.RightHandPosWeight);
            anim.SetIKRotationWeight(AvatarIKGoal.RightHand, gun.RightHandRoWeight);
        }
    }
}
```



Gun Shoot, Reload

- **Input Manager 추가**
  - Project Setting=>Input Manager
  - Size : 18=>19
  - **추가된, 가장 아래의 Axes 데이터 변경**
  - Name : **Reload**
  - Positive Button : **r**
  - Alt Positive Button : **공백**



Gun Shoot, Reload

- **PlayerInput(Script)**
  - 발사 버튼과 재장전 버튼 입력 처리 추가

```
public class PlayerInput : MonoBehaviour
{
    ...
    [SerializeField] private string fireBtnName = "Fire1";
    [SerializeField] private string reloadBtnName = "Reload";

    ...
    public bool fire { get; private set; }
    public bool reload { get; private set; }

    void Update()
    {
        ...
        fire = Input.GetButton(fireBtnName);
        reload = Input.GetButtonDown(reloadBtnName);
    }
}
```

Gun Shoot, Reload

- **Gun(Script)**
  - 총을 쏘고 재장전을 할 때 호출될 함수를 미리 선언

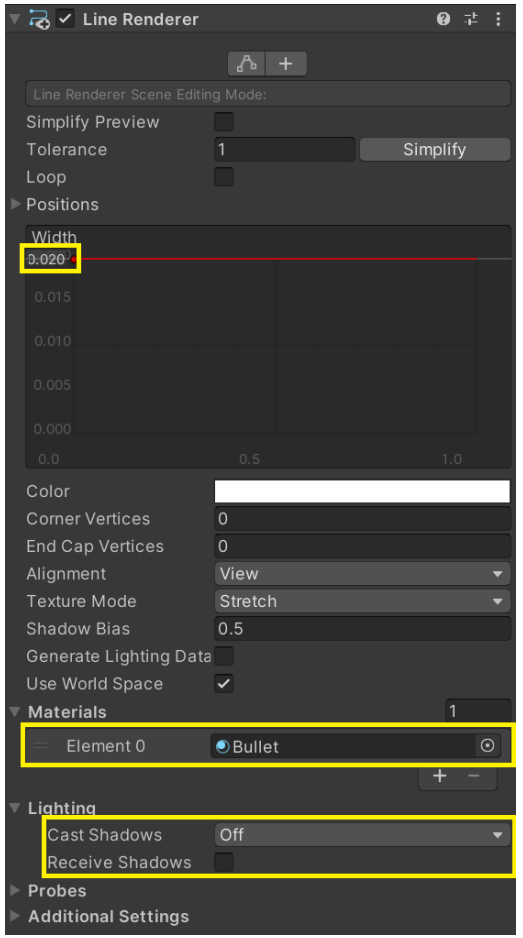
```
public class Gun : MonoBehaviour
{
    public void Fire() { }
    public bool Reload() { return true; }
}
```

- **PlayerShooter(Script)**
  - Input 이벤트 처리
  - 재장전 애니메이션 적용

```
public class PlayerShooter : MonoBehaviour
{
    private PlayerInput playerInput;
    private void Start()
    {
        ...
        playerInput = GetComponent<PlayerInput>();
    }
    private void Update()
    {
        if (playerInput && gun)
        {
            if (playerInput.fire) gun.Fire(); // 총알 발사.
            if (playerInput.reload && gun.Reload() && anim) anim.SetTrigger("Reload"); // 재장전 상태 확인 후, 재장전 애니메이션 재생.
        } // if
    } // Update()
} // class PlayerShooter
```

Gun Shoot, Reload

- **LineRenderer**
  - **Gun 오브젝트**에 **Line Renderer 추가**
  - **총알 궤적**을 그리기 위하여 사용
  - Width : 0.02
  - Cast Shadows : Off
  - Receive Shadows : false
  - Materials : 1
    - Element 0 : Materials 폴더의 **Bullet** Material을 **등록**



Gun Shoot, Reload

- **Gun(Script)**
  - 총 발사 처리
  - **발사 간격**을 설정하여 연사
  - 총알 재장전 처리

```
public enum State
{
    Ready, // 발사 준비됨
    Empty, // 탄창이 빔
    Reloading // 재장전 중
}
public class Gun : MonoBehaviour
{
    private State state = State.Ready; // 총의 현재 상태 정보.

    [Header("Gun 속성")]
    [SerializeField] private float hitRange = 50f; // 사정거리.
    [SerializeField] private float timeBetFire = 0.12f; // 총알 발사 간격.
    private float lastFireTime; // 총을 마지막으로 발사한 시점.
    private LineRenderer bulletLineRenderer; // 총알 궤적을 그리기 위한 도구.

    private readonly int magCapacity = 25; // 탄창 용량.
    public int ammoRemain { get; private set; } = 100; // 소지하고 있는 총알의 수.
    public int magAmmo { get; private set; } // 현재 탄창에 남아있는 총알의 수.

    [SerializeField] private float reloadTime = 0.9f; // 재장전 소요 시간.
```

Gun Shoot, Reload

```
private void Awake()
{
    …
    bulletLineRenderer = GetComponent<LineRenderer>();
    if (bulletLineRenderer)
    {
        bulletLineRenderer.positionCount = 2; // 선을 그리기 위한 두 점을 설정.
        bulletLineRenderer.enabled = false; // 총을 쏘기 전까지 궤적이 보이지 않도록 비활성화.
    }

    magAmmo = magCapacity; // 초기 탄창의 총알을 최대치로.
    state = State.Ready;
    lastFireTime = Time.time - timeBetFire;
}

public void Fire()
{
    if (state.Equals(State.Ready) && Time.time >= lastFireTime + timeBetFire) // 현재 총을 쏠 수 있는 상태인지 확인.
    {
        lastFireTime = Time.time; // 발사 시점 갱신.
        Shot(); // 발사 처리.
    }
}
```

Gun Shoot, Reload

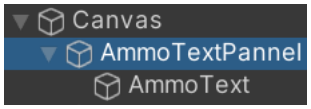
```
private void Shot()
{
    if (firePos)
    {
        RaycastHit hit; // Physics.Raycast()를 이용하여 충돌 지점 정보를 알아온다.
        Vector3 hitPos = firePos.position + firePos.forward * hitRange; // 총알이 맞은 위치를 저장, 최대 거리 위치를 기본 값으로 가진다.
        if (Physics.Raycast(firePos.position, firePos.forward, out hit, hitRange))
        {
            // TODO : Enemy(Zombie) Damageable Code 추가
            hitPos = hit.point; // 실제 총알이 맞은 지점으로 갱신.
        }
        StartCoroutine(ShotEffect(hitPos)); // 총알 발사 이펙트.

        magAmmo--; // 탄창의 총알을 감소.
        if (0 >= magAmmo) state = State.Empty; // 남은 총알 수 확인. 총알이 없다면, 총의 상태를 Empty로 변경.
    }
}
private IEnumerator ShotEffect(Vector3 hitPosition)
{
    if (bulletLineRenderer)
    {
        bulletLineRenderer.SetPosition(0, firePos.position); // 총알의 발사 지점에서,
        bulletLineRenderer.SetPosition(1, hitPosition); // 총알이 맞은 위치까지 선을 그린다.
        bulletLineRenderer.enabled = true; // 그림을 그리기위해 LineRenderer를 활성화 시킨다.
    }
    yield return new WaitForSeconds(0.03f); // 0.03초 동안 잠시 처리를 대기.
    if(bulletLineRenderer) bulletLineRenderer.enabled = false; // LineRenderer를 비활성화하여 총알 궤적을 지운다.
}
```

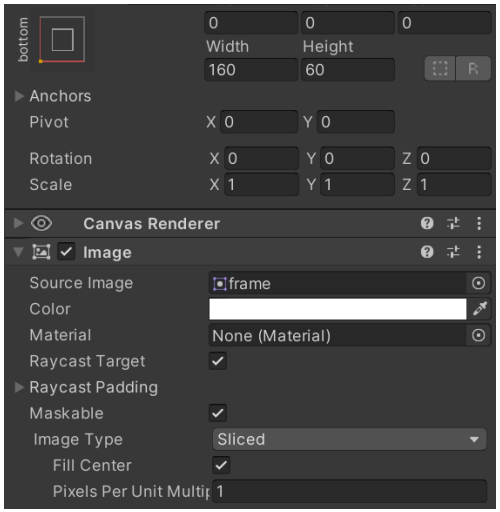
Gun Shoot, Reload

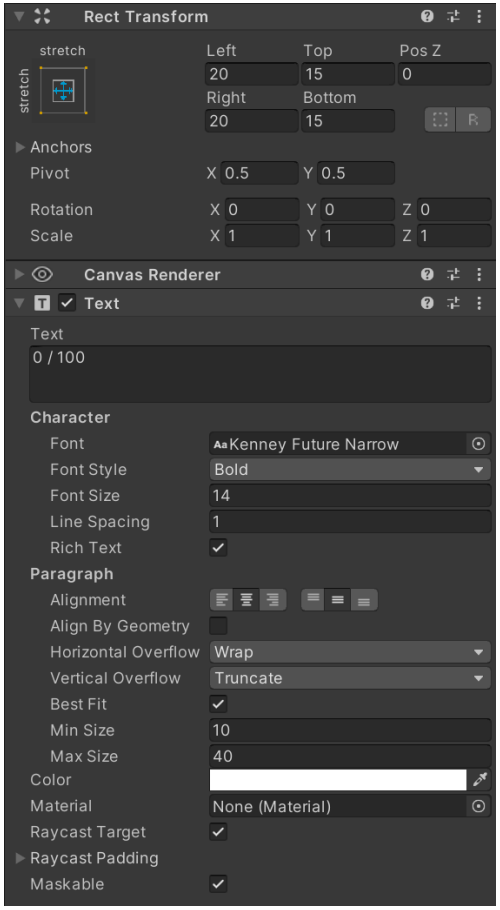
```
    public bool Reload()
    {
        // 재장전 중이거나 남은 탄약이 없거나,
        // 탄알이 가득차서 더이상 추가할 수 없는 경우는 재장전 불가능.
        if (state.Equals(State.Reloading) || 0 >= ammoRemain || magCapacity <= magAmmo) return false;

        StartCoroutine(ReloadRoutine()); // 재장전 상태로 변경.
        return true;
    }

    private IEnumerator ReloadRoutine()
    {
        state = State.Reloading; // 현재 상태를 재장전 중 상태로 전환.

        yield return new WaitForSeconds(reloadTime); // 재장전 소요 시간 만큼 처리를 쉬기.

        // 탄창에 채울 총알을 계산.
        // 소지하고 있는 총알과 탄창의 빈 공간 수를 확인하여 작은 값을 사용.
        int ammoTofill = Mathf.Min(magCapacity - magAmmo, ammoRemain);
        ammoRemain -= ammoTofill; // 소지하고 있는 총알의 수를 감소.
        magAmmo += ammoTofill; // 감소한 총알의 수만큼 탄창에 총알을 추가.

        state = State.Ready; // 총의 현재 상태를 발사 준비된 상태로 변경.
    }
} // public class Gun
```

UI

- **AmmoTextPannel(Image)**
  - UI=>Image
  - Anchor : (left, bottom)
  - Pivot : (0, 0)
  - Pos X, Pos Y, Width, Height : (0, 0, 160, 60)
  - Source Image : Sprites=>frame
  - Child : AmmoText(Text)
    - Anchor : (stretch, stretch)
    - Left, Top, Right, Bottom: (20, 15, 20, 15)
    - Font : Fonts=>Kenney Future Narrow
    - Font Style : Bold
    - Alignment : 가운데 정렬
    - Best Fit : true
    - Color : (255, 255, 255, 255)







UI

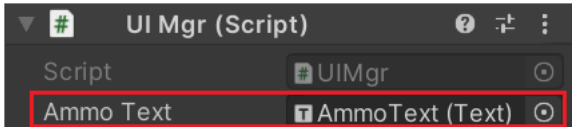
- **UIMgr(Script)**
  - Canvas에 UIMgr 스크립트 **적용**
  - 남은 총알의 수와 탄약의 수를 표시

```
using UnityEngine.UI;
public class UIMgr : MonoBehaviour
{
    public static UIMgr Instance { get; private set; }

    private void Awake()
   {
        if (null == Instance)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
            return;
        }
        Destroy(gameObject);
    }
```

UI

```
    [SerializeField] private Text ammoText; // 탄약 표시용 텍스트.

    public void UpdateAmmoText(int magAmmo, int remainAmmo)
    {
        //var strBuilder = new System.Text.StringBuilder();
        //strBuilder.Append(magAmmo);
        //strBuilder.Append(" / ");
        //strBuilder.Append(remainAmmo);
        //ammoText.text = strBuilder.ToString(); 또는,
        ammoText.text = string.Format("{0} / {1}", magAmmo, remainAmmo);
    }
} // class UIMgr
```

- ammoText에 AmmoText(Text) **연결**

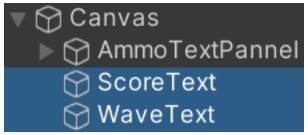


UI

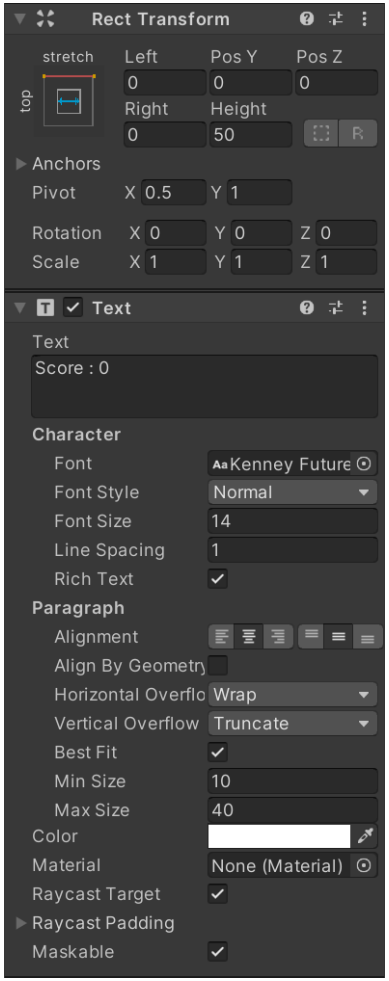
- **PlayerShooter(Script)**
  - 총알과 탄약의 수량을 갱신

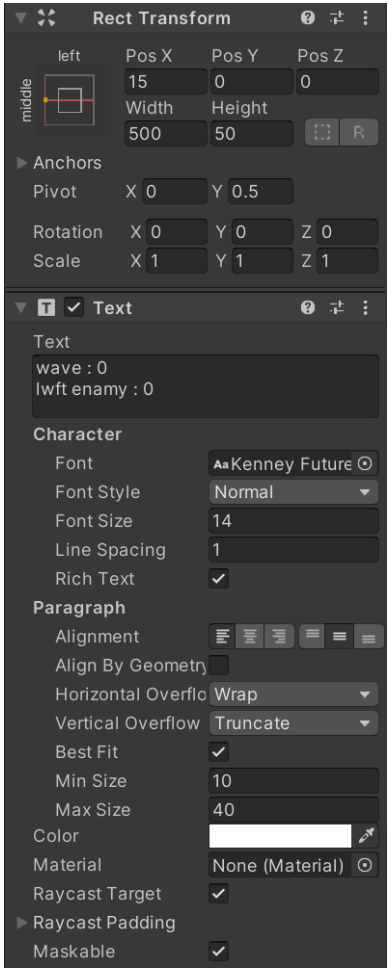
```
public class PlayerShooter : MonoBehaviour
{
   private void Update()
    {
        ...
        if (gun && UIMgr.Instance) UIMgr.Instance.UpdateAmmoText(gun.magAmmo, gun.ammoRemain);
    }
}
```

UI



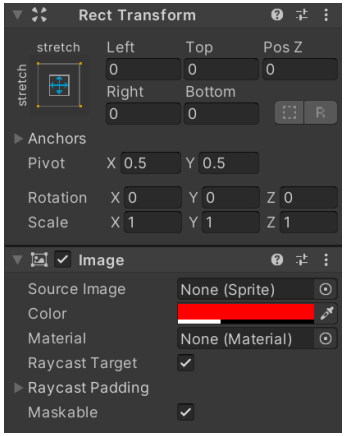
- **ScoreText(Text)**
  - UI=>Text
  - Anchor : (stretch, top)
  - Pivot : (0.5, 1)
  - Left, Pos Y, Right, Height : (0, 0, 0, 50)
  - Font : Fonts=>Kenney Future Narrow
  - Alignment : 가운데 정렬
  - Best Fit : true
  - Color : (255, 255, 255, 255)
- **WaveText(Text)**
  - UI=>Text
  - Anchor : (left, middle)
  - Pivot : (0, 0.5)
  - Pos X, Pos Y, Width, Height : (15, 0, 500, 50)
  - Font : Fonts=>Kenney Future Narrow
  - Alignment : 왼쪽, 가운데 정렬
  - Best Fit : true
  - Color : (255, 255, 255, 255)

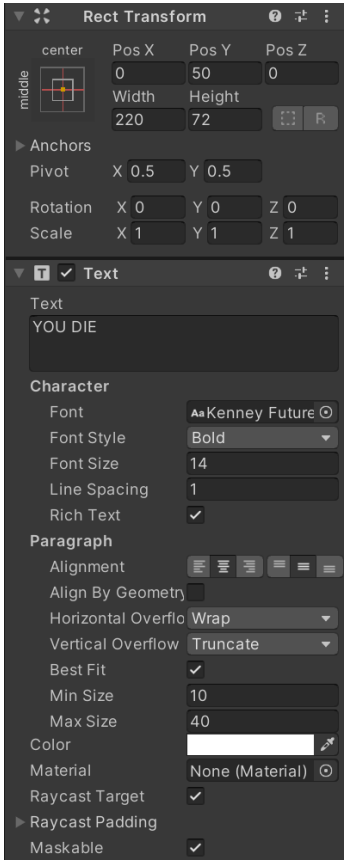


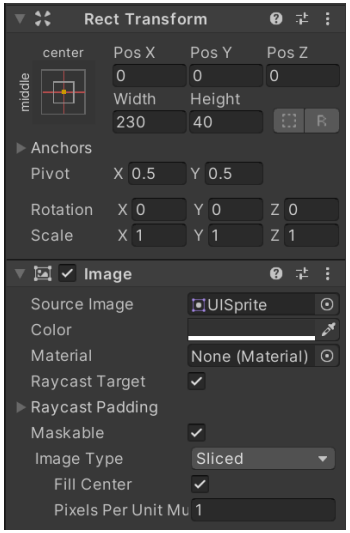


UI

- **GameOver(Image)**
  - UI=>Image
  - Anchor : (stretch, stretch)
  - Pivot : (0.5, 0.5)
  - Left, Top, Right, Bottom : (0, 0, 0, 0)
  - Source Image : None (Sprite)
  - Color : (255, 0, 0, 80)
  - Child : Text(UI=>Text)
    - Anchor : (center, middle)
    - Pos x, Po Y, Width, Height : (0, 50, 220, 72)
    - Text : YOU DIE
    - Font : Fonts=>Kenney Future Narrow
    - Font Style : Bold
    - Alignment : 가운데 정렬
    - Best Fit : true
    - Color : (255, 255, 255, 255)
  - Child : RestartButton(UI=>Button)
    - Anchor : (center, middle)
    - Pos x, Po Y, Width, Height : (0, 0, 230, 40)
    - Color : (50, 50, 50, 255)
    - Child : Text
      - Text : RESTART
      - Font : Fonts=>Kenney Future Narrow
      - Color : (255, 255, 255, 255)

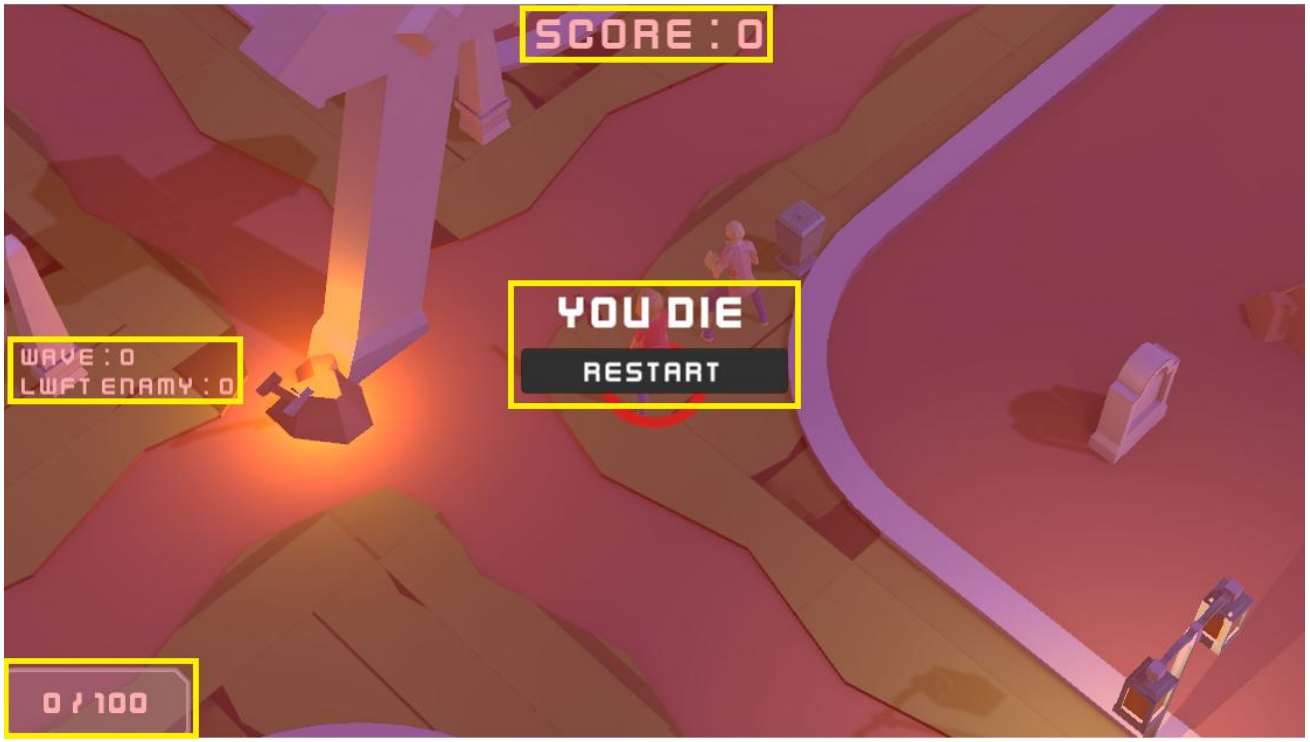






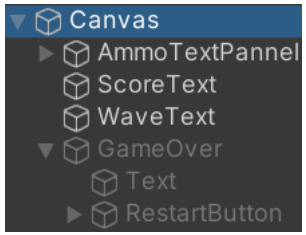
UI

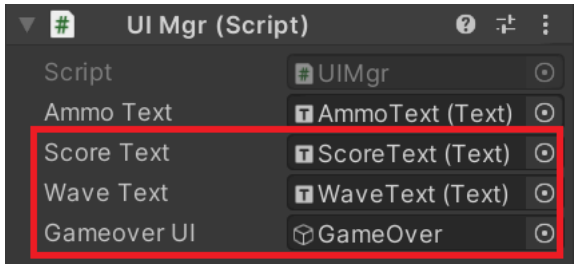
- 플레이 중에는 보이지 않도록 **GameOver**(Image)의 **오브젝트 비활성화**

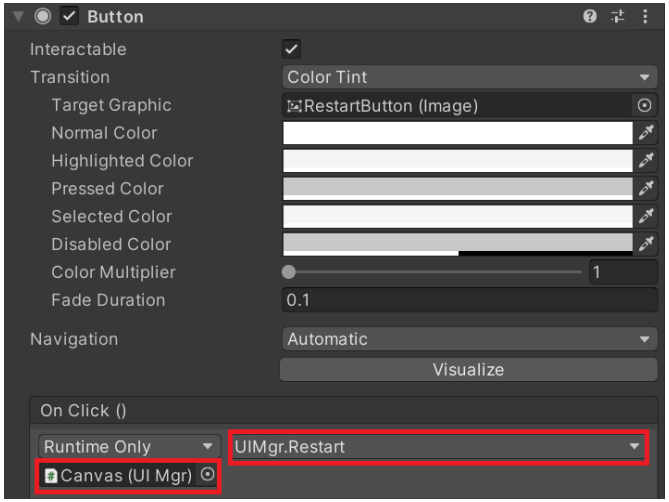


UI

- **UIMgr(Script)**
- Score, Wave Info, GameOver, Restart Button을 연결
- Restart Button의 OnClick Event를 적용





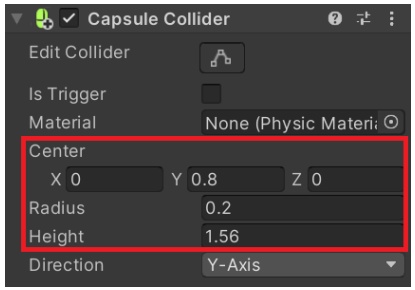



```
public class UIMgr : MonoBehaviour
{
    ...
    [SerializeField] private Text scoreText; // 점수 표시용 텍스트.
    [SerializeField] private Text waveText; // 적 웨이브 표시용 텍스트.
    [SerializeField] private GameObject gameoverUI; // 게임 오버시 활성화할 UI.
    public event System.Action RestartEvent;
```

UI

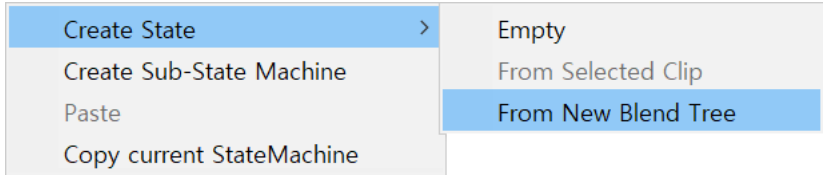
```
    public void UpdateScoreText(int newScore)
    {
        scoreText.text = string.Format("Score : {0}", newScore);
    }

    public void UpdateWaveText(int waves, int count)
    {
        waveText.text = string.Format("Wave : {0}₩nEnemy Left : {1}", waves, count);
    }

    public void GameOver()
    {
        gameoverUI.SetActive(true);
    }

    public void Restart()
    {
        if (null != RestartEvent) RestartEvent();
    }
} // class UIMgr
```

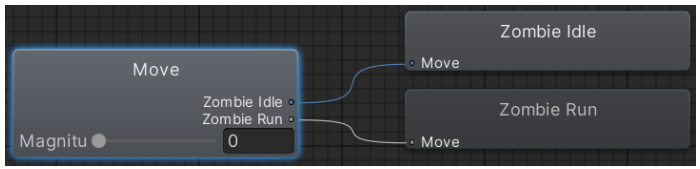
Enemy, AI

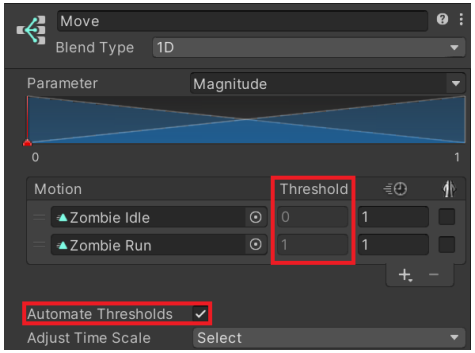


- **Enemy**
  - Models=>Zombie를 Hierarchy에 등록
  - **Zombie**의 Animator에 **Animator Controller(Zombie)**를 **새로 만들어서 추가**
  - **Capsule Collider 추가, Center(0, 0.8, 0), Radius(0.2), Height(1.56)**

- **Animator Controller(Zombie)**
  - **Parameters**
    - Magnitude(**float**) : 움직임 값을 받아서 애니메이션을 제어
    - Attack(**Trigger**) : 공격 애니메이션을 알리는 키
    - Damaged(**Trigger**) : 피격 애니메이션을 알리는 키
    - isDead(**Bool**) : 체력이 없을 경우 애니메이션을 알리는 키
  - Create State=>From New Blend Tree(Movement)
  - From New Blend Tree(Movement)에 Zombie Idle, Zombie Run 등록
  - Zombie Idle Threshold : 0
  - Zombie Run Threshold : 1

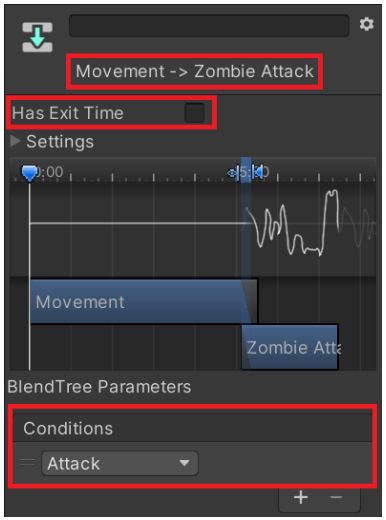


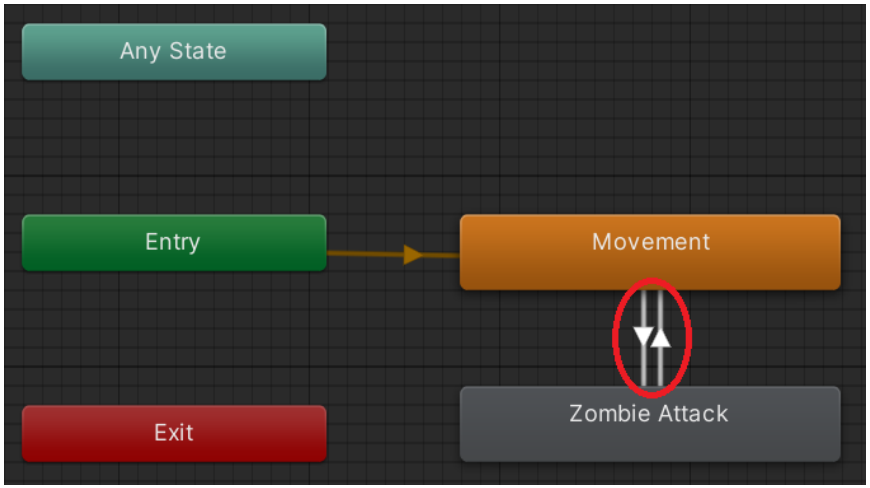




Enemy, AI

- **Zombie Attack** Animation Clip **추가**
- **Movement->Zombie Attack** Conditions : **Attack(Trigger)**
- **Movement->Zombie Attack** Has Exit Time : **false**
- **Zombie Attack->Movement** Has Exit Time : **true**

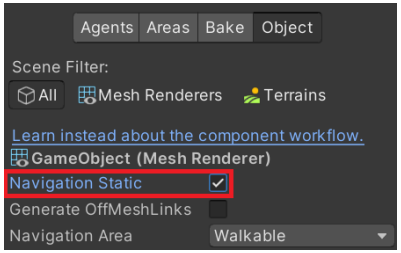


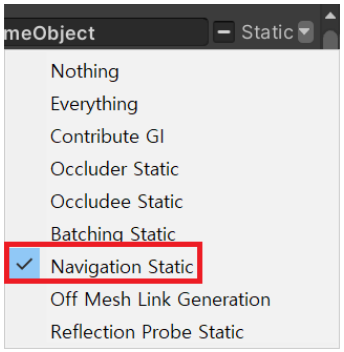


Enemy, AI

Navigation

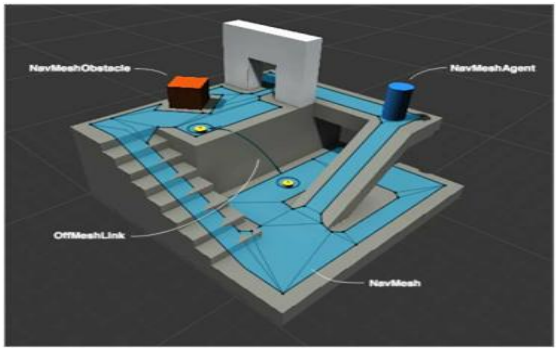
- A* 알고리즘을 **기반**으로 만들어진, **길 찾기 시스템**
- Window=>AI=>Navigation
- Agent : Navigation을 이용하여 **이동하는 대상**
- NeviMesh : Agent가 **이동하는 경로**
- Obtacle : Agent가 가는 길을 방해하는 **장애물**





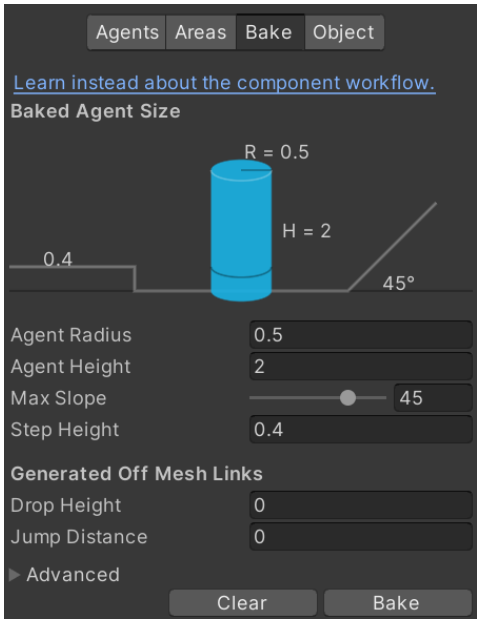
Object 탭

- All : Inspector 창에 모든 오브젝트 표시
- Mesh Renderers : Inspector 창에 Mesh Renderer를 지닌 모든 오브젝트를 표시
- Terrains : Inspector 창에 Terrain 형식의 모든 오브젝트 표시
- Navigation Static : **선택된 모든 오브젝트로 NevMesh를 만든다**, GameObject에 있는 **Static에서도 선택 가능**
- Generate Off Mesh Links : 끊어진 길을 이어 이동이 가능하게 한다
- Navigation Area : 선택한 오브젝트에 Areas 탭에서 설정한 지형의 속성을 설정

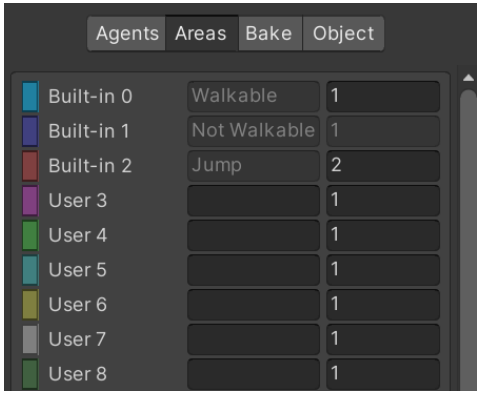


Enemy, AI

- **Bake 탭**
  - Clear : NaviMesh를 지운다
  - Bake : 설정한 값을 참고하여 **NeviMesh를 만든다**
  - Agent Radius : Agent의 크기(반지름), **지나갈 수 있는 길의 폭(너비)**
  - Agent Height : Agent의 높이, **지나갈 수 있는 터널 등의 높이**
  - Max Slope : **이동 가능한 최대 경사각**
  - Step Height : **올라갈 수 있는 계단의 높이**

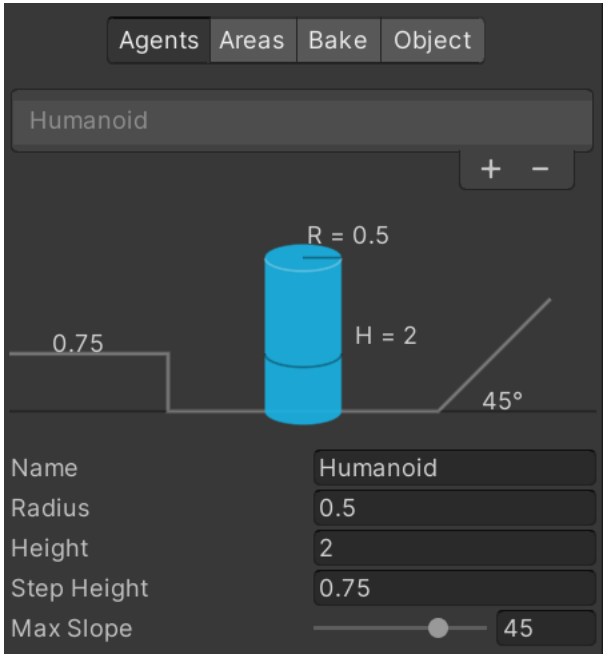


- **Areas 탭**
  - 해당 지형을 표시하는 색
  - 해당 지형의 이름
  - **해당 지형을 이동하는데 필요한 비용(cost)**



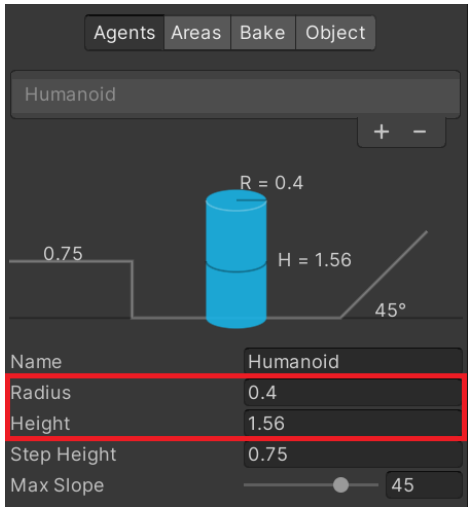
Enemy, AI

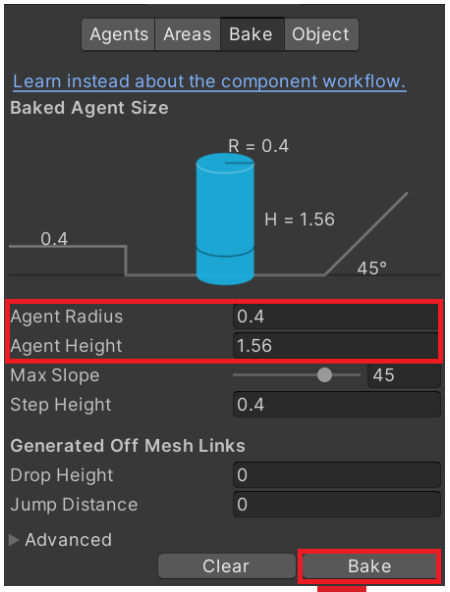
- **Agents 탭**
  - **실제 Agent의 이동에 사용**
  - 여러 Agent를 만들어 어떤 Agent는 지나갈 수 있는 지형도 다른 타입의 Agent는 지나가지 못하고 돌아가서 지나가는 등의 처리가 가능
  - Name : 해당 **Agnet의 Type이름**
  - Radius : **Agent의 너비**
  - Height : **Agent의 높이**
  - Step Height : Agent가 **이동 가능한 계단의 높이**(걸음 높이)
  - Max Slope : Agent가 **이동 가능한 언덕의 최대 기울기**

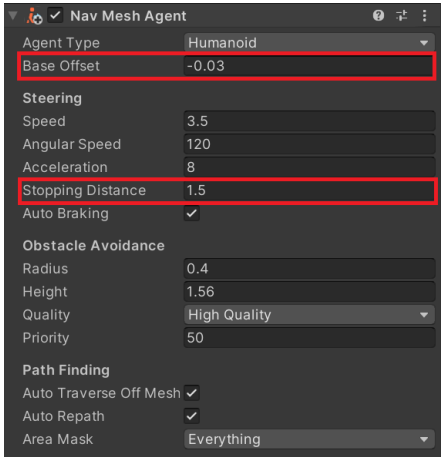


Enemy, AI

- **Navigation : Agents, Bake**
  - Agent Radius : 0.4
  - Agent Height : 1.56
- **Zombie(GameObject)**
  - **Nav Mesh Agent 컴포넌트 추가**
  - Base Offset : -0.03
  - Stopping Distance : 1.5



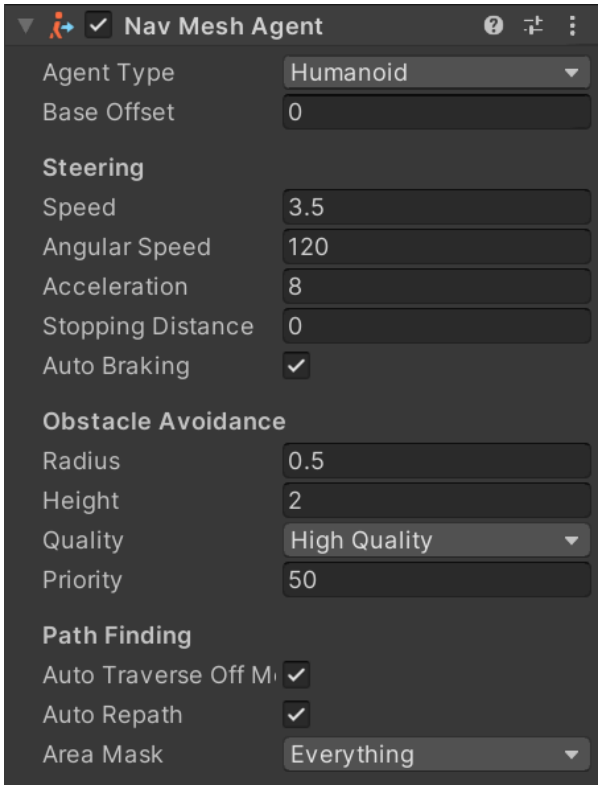




Enemy, AI

Nav Mesh Agent

- Agent Type : Navigation의 **Agent탭에서 설정**한 Agent Type
- Base Offset : 길 찾기를 할 때 사용되는 **충돌 실린더의 위치**
- Speed : Agent가 움직이는 **속도**
- Angular Speed : Agent가 **회전하는 속도**(Degree/sec)
- Acceleration : **가속도**
- Stopping Distance : 목적지보다 **설정한 값만큼 떨어진 지점에서 정지**
- Auto Braking : Agent가 **목적지 도착전에 감속**할지를 지정
- Radius, Height : **다른 Agent 또는 NavMeshObtacle과의 충돌 영역**
- Quality : 다른 Agent 또는 NavMeshObtacle과의 **회피 품질**
- Priority : **길 찾기의 우선 순위**, 우선 순위가 높은 Agent는
  우선 순위가 낮은 낮은 Agent를 무시하고 밀고
  지나가 버린다
- Auto Traverse Off Mesh Link : **Off Mesh Link를 이용**
- Auto Repath : **Nav Mesh가 변경될 경우 길을 다시 찾는다**
- Area Mask : Navigation의 Areas탭에서 설정한 Area 정보를
  참고하여 해당 **Agent가 지나갈수 있는 길**을 정한다



Enemy, AI

- **Enemy(Script)**
  - **C# Enemy** 스크립트 **생성**
  - **Zombie 오브젝트**에 **추가**
  - **NevMeshAgent를 이용**하여 **Player(target)을 따라가게**하고, **공격 거리에 들어가면 공격**
  - **탐색 범위를 설정**하여 Enemy가 Player를 **인식할 수 있는 범위를 제어**

```
public class Enemy : MonoBehaviour
{
    [SerializeField] private LayerMask targetLayer;
    [SerializeField] [Range(0, 100)] private float searchRange = 20;

    [SerializeField] private NavMeshAgent agent;
    private Animator anim;
```

Enemy, AI

```
// Gizmo를 이용하여 target을 찾는 가시 범위를 볼수 있다.
private void OnDrawGizmosSelected()
{
    Gizmos.DrawWireSphere(transform.position, searchRange);
}

private void Awake()
{
    anim = GetComponent<Animator>();
}

private void OnEnable()
{
    // 오브젝트가 활성화 될 경우(Respawn), target을 찾아 이동.
    if (agent) agent.isStopped = false;
    StartCoroutine(UpdatePath());
}
```

Enemy, AI

```
private IEnumerator UpdatePath()
{
    while (true)
    {
        if (agent)
        {
            var targets = Physics.OverlapSphere(transform.position, searchRange, targetLayer); // 설정한 탐색 범위 내에 Target(Player)이 있는 지 확인.
            if (null != targets && 0 < targets.Length)
            {
                var targetPos = targets[0].transform.position;
                agent.SetDestination(targetPos); // 해당 Target을 향하여 이동.
                if (Vector3.Distance(targetPos, transform.position) <= agent.stoppingDistance) //일정 거리(stoppingDistance)만큼 다가갔을 경우,
                {
                    targetPos.y = transform.position.y;
                    var dir = (targetPos - transform.position).normalized;
                    transform.rotation = Quaternion.LookRotation(dir); //target을 향하여 바라보고,
                    StartCoroutine(Attack(targets[0].transform)); // 공격을 시도한다.
                    yield break;
                }
            }
            // Enemy가 움직이는 속도(velocity)의 크기(magnitude)를 이용하여, 움직이는 애니메이션 처리를 한다.
            if (anim) anim.SetFloat("Magnitude", agent.velocity.magnitude);
        } // if(agent)
        yield return new WaitForSeconds(0.04f);
    } // While()
} // UpdatePath()
```

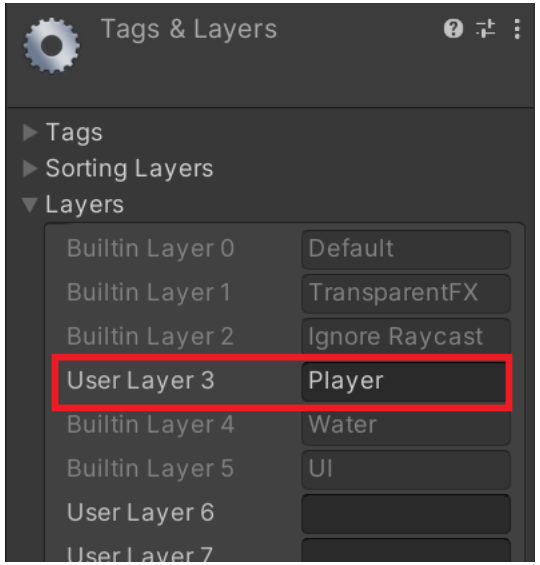
Enemy, AI

```
    private IEnumerator Attack(Transform target)
    {
        if (agent)
        {
            while (true)
            {
                // 공격 모션 실행.
                if (anim) anim.SetTrigger("Attack");
                yield return new WaitForSeconds(1.1f);

                // 피격 판정 타이밍에 target이 유효한 거리에 있는지 확인.
                if (Vector3.Distance(target.position, transform.position) > agent.stoppingDistance) break;

                // TODO : Player Damageable Code 추가.

                yield return new WaitForSeconds(1.2f);

                // 모션 종료 후, target이 유효한 거리에 있는지 확인.
                if (Vector3.Distance(target.position, transform.position) > agent.stoppingDistance) break;
            }
        }
        // target과의 거리가 벌어진다면 다시 target을 쫓아 간다.
        StartCoroutine(UpdatePath());
    } // Attack()
} // class Enemy
```

Enemy, AI

Layer

- 원하는 순서에 **'Player' 레이어 등록**



Woman

- **Woman(GameObject)**의 **Layer**를 **'Player'**로 **변경**
- Change Layer 창에서 **[No, this object only]**를 선택

Zombie

- Enemy Script의 **Target Layer**를 **'Player'**로 **변경**
- Search Range : 3.7
- **Agnet**에 **Zombie의 Agent**를 **등록**