

Unity Zombie 下

목차

1. Interface를 이용한 개체별 HP 관리 . . . 3p
2. HP UI . . . 14p
3. Effects & Sounds . . . 18p
4. Item . . . 27p
5. Item Spawn . . . 35p
6. Enemy Spawn . . . 43p
7. 포스트 프로세싱(Post Processing) . . . 52p

Interface를 이용한 개체별 HP 관리

▶ IDamageable(Interface)

- 데미지를 입을 수 있는 타입들이 공통적으로 가져야 하는 인터페이스

```
public interface IDamageable
{
    ///<summary> 데미지 크기(damage), 맞은 지점(hitPoint), 맞은 표면의 방향(hitNormal)</summary>
    void OnDamage(float damage, Vector3 hitPoint, Vector3 hitNormal);
}
```

▶ LivingEntity(Class)

- 체력을 가진 오브젝트가 공통으로 가지는 Class
- 체력의 회복, 피격, 죽음 처리를 한다
- 오브젝트 활성화 시에 해당 개체의 체력을 초기화 한다

```
public class LivingEntity : MonoBehaviour, IDamageable
{
    [SerializeField] protected float maxHealth = 100; // 최대 체력.
    public float health { get; protected set; } // 현재 체력.
    public bool isDead { get { return (0 >= health); } } // 죽음 상태 확인.
    public event Action OnDeath; // 사망 시 발동할 이벤트.
```

Interface를 이용한 개체별 HP 관리

```
protected virtual void OnEnable()
{
    health = maxHealth;
}

public virtual void OnDamage(float damage, Vector3 hitPoint, Vector3 hitNormal)
{
    if (isDead) return; // 이미 죽은 상태라면 더 이상 처리하지 않는다.

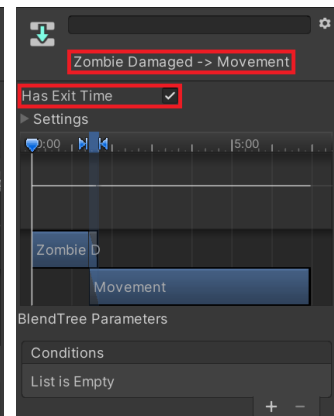
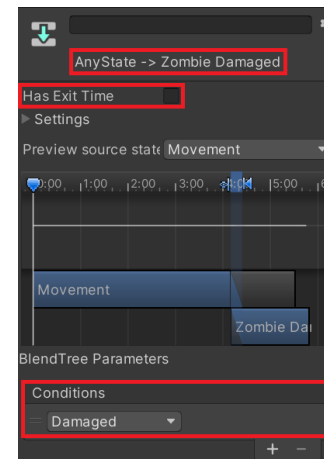
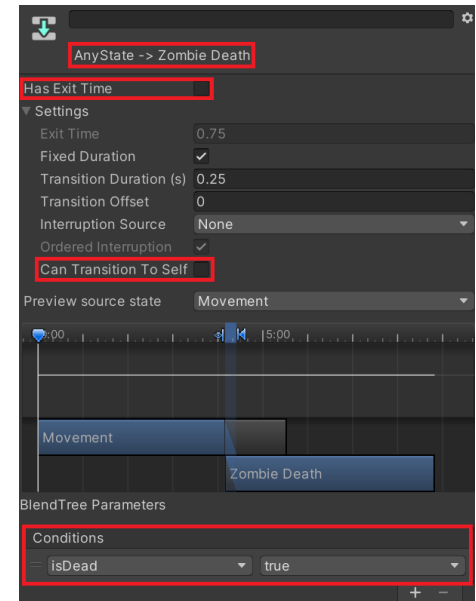
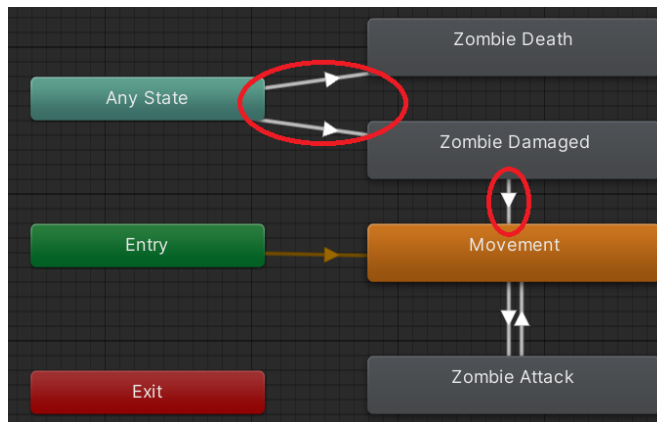
    health -= damage; // 데미지 만큼 체력 감소.
    if (isDead) Die(); // 데미지를 입어 체력이 0이하(사망 상태)라면 사망 이벤트 실행.
}

private void Die()
{
    if (null != OnDeath) OnDeath(); // 등록된 사망 이벤트 실행.
}

public virtual void RestoreHealth(float value)
{
    if (isDead) return; // 이미 죽은 상태에서는 체력을 회복할 수 없다.
    health = Mathf.Clamp(health + value, 0, maxHealth); // 체력은 최대치를 넘어 회복할 수 없다.
}
} // class LivingEntity
```

Interface를 이용한 개체별 HP 관리

- ▶ **Animator Controller(Zombie)**
 - **Zombie Death** Animation Clip **추가**
 - ▶ Any State→Zombie Death Conditions : **isDead(Bool:true)**
 - ▶ Any State→Zombie Death Has Exit Time : **false**
 - ▶ Any State→Zombie Death Setting/Can Transition To Self : **false**
 - **Zombie Damaged** Animation Clip **추가**
 - ▶ Any State→Zombie Damaged Conditions : **Damaged(Trigger)**
 - ▶ Any State→Zombie Damaged Has Exit Time : **false**
 - ▶ Zombie Damaged→Movement Has Exit Time : **true**



Interface를 이용한 개체별 HP 관리

▶ Enemy(Script)

- LivingEntity를 상속하여 피격 및 사망 처리
- **MonoBehaviour를 LivingEntity로 변경**

```
public class Enemy : LivingEntity
{
    private new Collider collider;

    private void Awake()
    {
        ...
        collider = GetComponent<Collider>(); // Collider의 종류를 신경쓰지 않는다.

        OnDeath += () =>
        {
            // 더 이상 피격 판정이 되지 않게 collider를 끈다.
            if (collider) collider.enabled = false;
            if (agent) agent.isStopped = true; // navigation 정지.
            if (anim) anim.SetBool("isDead", isDead); // Zombie Death 애니메이션 실행.
        };
    }
}
```

Interface를 이용한 개체별 HP 관리

```
// 기존의 private를 protected로 변경 후 override 키워드 추가.
protected override void OnEnable()
{
    base.OnEnable(); // LivingEntity의 OnEnable() 호출.

    if (anim) anim.SetBool("isDead", isDead); // 사망 상태를 false. isDead=false.
    if (collider) collider.enabled = true; // 피격을 받을 수 있도록 collider를 활성화.
    ...
}

public override void OnDamage(float damage, Vector3 hitPoint, Vector3 hitNormal)
{
    base.OnDamage(damage, hitPoint, hitNormal);

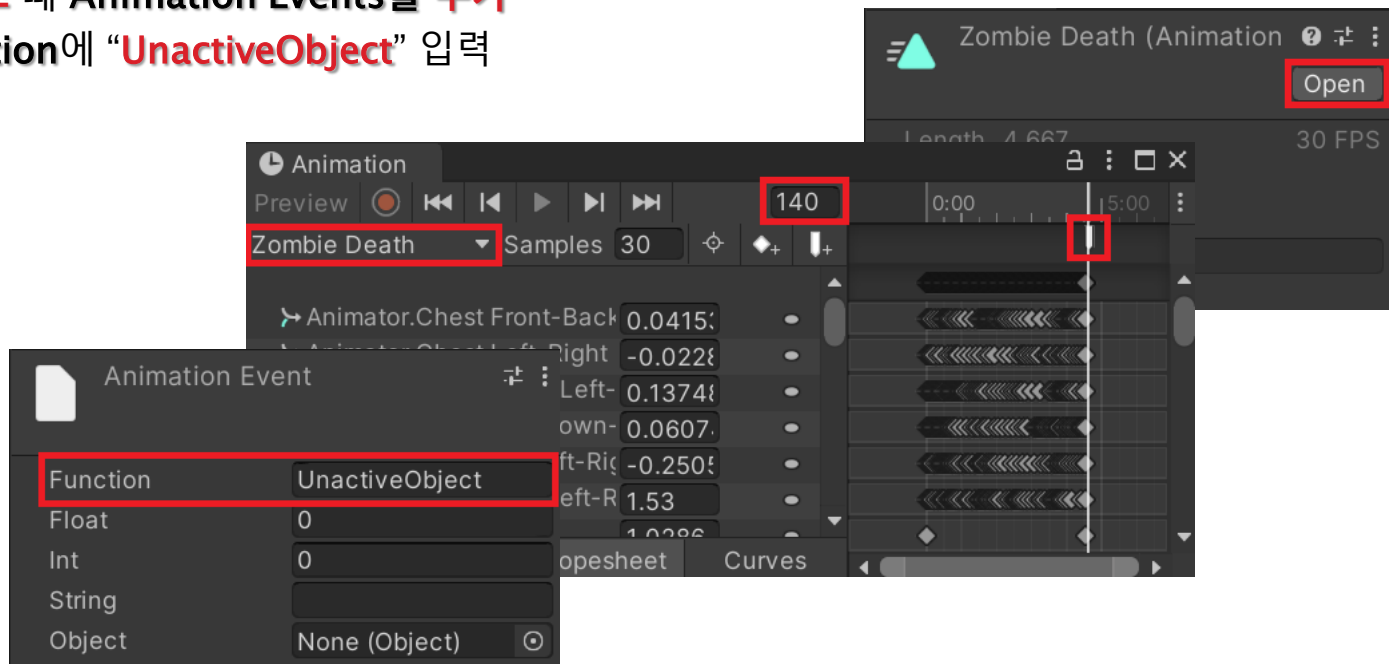
    if (anim && !isDead)
    {
        anim.SetTrigger("Damaged"); // 데미지를 입고 죽지 않았다면, 피격 애니메이션 실행.
    }
}

public void UnactiveObject() // Zombie Death 실행 후 호출하여 오브젝트를 비활성화 시킨다.
{
    gameObject.SetActive(false);
}
} // class Enemy
```

Interface를 이용한 개체별 HP 관리

▶ Zombie Death(Animation Clip)

- Animation Events를 추가, Enemy Script의 UnactiveObject()를 호출
- **Zombie Death Animation Clip**을 선택 하여 Inspector 창의 **[Open]** 버튼 선택
- **140초** 때 Animation Events를 **추가**
- Function에 “**UnactiveObject**” 입력



Interface를 이용한 개체별 HP 관리

▶ Gun(Script)

- '// TODO : Enemy(Zombie) Damageable Code 추가' 위치에 코드 추가
- 총이 가지고 있는 damage만큼 Enemy에게 피해를 입힌다

```
public class Gun : MonoBehaviour
{
    ...
    [SerializeField] private float damage = 25; // 무기의 공격력.

    private void Shot()
    {
        if (firePos)
        {
            ...
            if (Physics.Raycast(firePos.position, firePos.forward, out hit, hitRange))
            {
                // TODO : Enemy(Zombie) Damageable Code 추가
                IDamageable target = hit.collider.GetComponent<IDamageable>();
                if (null != target) target.OnDamage(damage, hit.point, hit.normal);
                ...
            }
            ...
        }
    }
}
```

Interface를 이용한 개체별 HP 관리

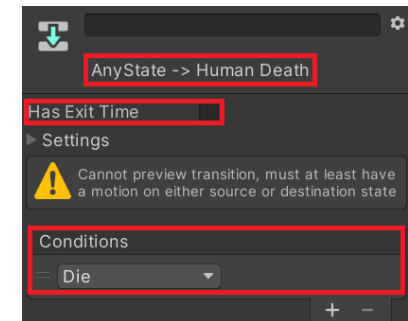
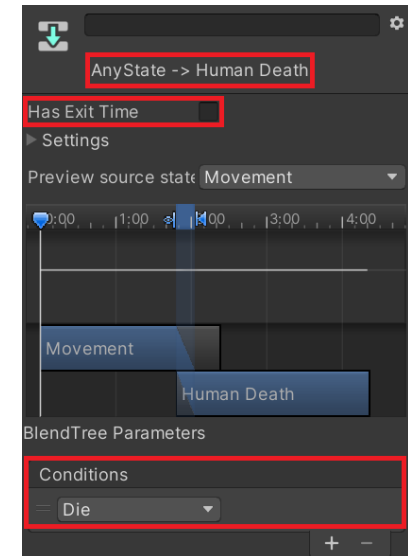
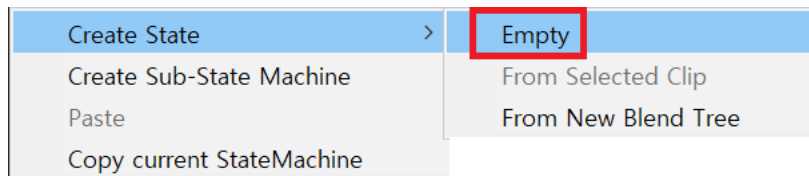
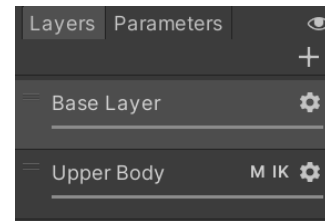
▶ Animator Controller(Player)

- Base Layer

- ▶ Human Death Animation Clip **추가**
- ▶ Any State→ Human Death Has Exit Time : **false**
- ▶ Any State→Human Death Conditions : **Die(Trigger)**

- Upper Layer

- ▶ Create State=>**Empty**(Human Death) **추가**
- ▶ Any State→Human Death Has Exit Time : **false**
- ▶ Any State→Human Death Conditions : **Die(Trigger)**



Interface를 이용한 개체별 HP 관리

▶ PlayerHealth(Script)

- C# PlayerHealth 스크립트 **생성**
- **Woman** 오브젝트에 **추가**
- 플레이어의 체력을 적용하여 사망 시 애니메이션 적용
- 플레이어 사망 시, 인풋을 제한하여 이동 및 총 발사를 막는다

```
public class PlayerHealth : LivingEntity
{
    private PlayerInput playerInput;
    private Animator anim;

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

        OnDeath += () =>
        {
            UIMgr.Instance.GameOver();
            if (playerInput) playerInput.enabled = false;
            if (anim) anim.SetTrigger("Die");
        };
    }
}
```

Interface를 이용한 개체별 HP 관리

▶ Enemy(Script)

- 범위 내의 대상이 살아있고 자신이 살아있는 경우, 확인하여 대상을 쫓아가 공격하도록 **변경**

```
public class Enemy : LivingEntity
{
    ...
    private float damage = 20f; // 공격력.
    private IEnumerator UpdatePath()
    {
        while (!isDead) { // true를 isDead로 변경, Enemy가 살아있을 경우 대상 찾기를 한다.
            if (agent) {
                ...
                if (null != targets && 0 < targets.Length) {
                    var livingEntity = targets[0].GetComponent<LivingEntity>(); // 추가.
                    if (livingEntity && !livingEntity.isDead) { // 대상이 존재하고 죽지 않았을 경우,
                        var targetPos = livingEntity.transform.position; // targets[0]을 livingEntity로 변경.
                        // 해당 Target을 향하여 이동, 일정 거리(stoppingDistance)만큼 다가갔을 경우, target을 향하여 바라보고,
                        ...
                        StartCoroutine(Attack(livingEntity)); // 공격을 시도한다. Transform을 LivingEntity로 변경.
                        ...
                    }
                } // if(null != targets && 0 < targets.Length)
                ...
            } // if(agent)
            ...
        } // while()
    } // UpdatePath()
}
```

Interface를 이용한 개체별 HP 관리

```
private IEnumerator Attack(LivingEntity target)
{
    if (agent && target)
    {
        var trTarget = target.transform; // target의 Transform.
        while (!isDead && !target.isDead) // true에서 (!isDead && !target.isDead)로 변경.
        {
            ...

            if (Vector3.Distance(trTarget.position, transform.position) > agent.stoppingDistance) break;
            // Player Damageable Code 추가.
            if (isDead || target.isDead) yield break;
            var hitNormal = transform.position - trTarget.position;
            target.OnDamage(damage, Vector3.zero, hitNormal);

            yield return new WaitForSeconds(1.2f);

            if (Vector3.Distance(trTarget.position, transform.position) > agent.stoppingDistance) break;
        }
    }
    if (!isDead) StartCoroutine(UpdatePath()); // if(!isDead) 조건 추가.
}
} // class Enemy
```

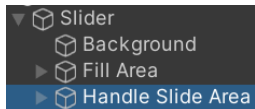
HP UI

▶ Slider

- UI=>Canvas를 **먼저 만든 후** 그 하위에 UI=>Slider를 **만든다**



- Slider의 **Handle Slide Area**를 **제거**

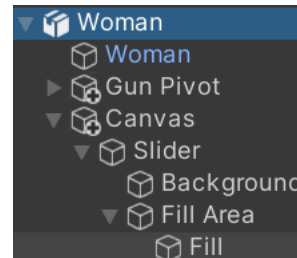


- Slider를 포함한 하위 자식들을 **모두 선택**

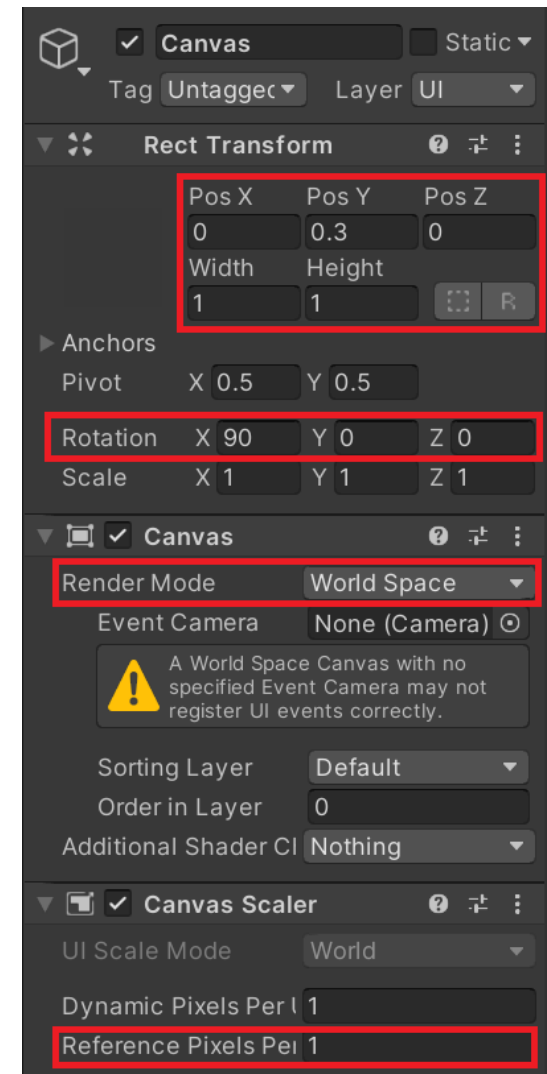
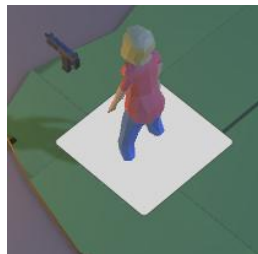


- **Anchor 창**에서 **Alt 키를 누른 상태**로 (stretch, stretch)를 선택

- **Canvas를 Woman의 하위에 추가**



- Render Mode : **World Space**
- Reference Pixels Per Unit : **1**
- Position : (0, 0.3, 0)
- Width/Height : (1, 1)
- Rotation : (90, 0, 0)

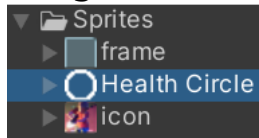


HP UI

- Background와 Fill을 선택



- Image의 Source Image를 Sprites/Health Circle로 변경



- Background

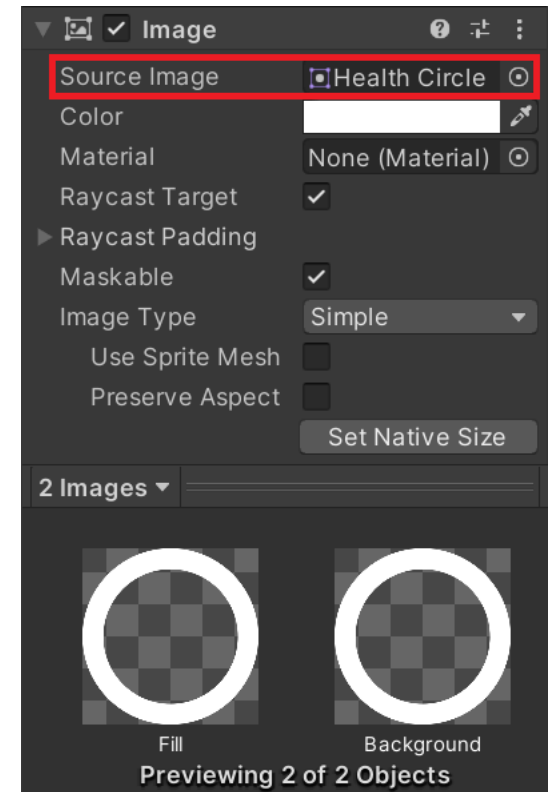
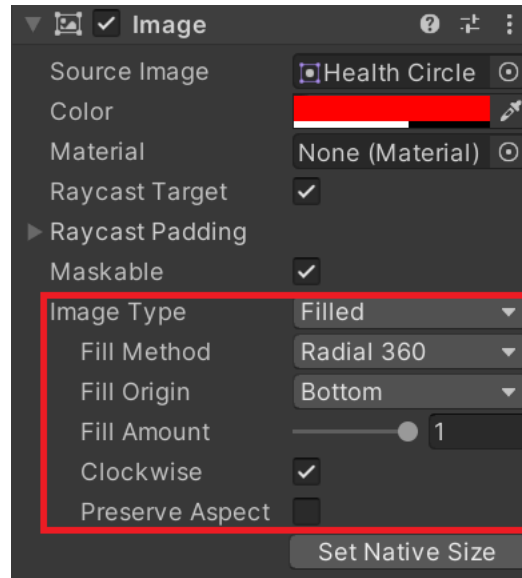
➤ Color : (255, 255, 255, 30)

- Fill

➤ Color : (255, 0, 0, 150)

➤ Image Type : Filled

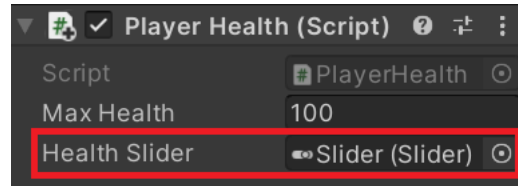
➤ Fill Method : Radial 360



HP UI

▶ PlayerHealth(Script)

- UI Slider를 Script로 제어
- 플레이어가 피해를 입거나 회복을 할 경우 Value 값을 변경하도록 한다



using UnityEngine.UI; // UI를 사용하기 위하여 추가.

public class PlayerHealth : LivingEntity

{
 [SerializeField] private Slider healthSlider; // 체력 표시를 할 UI Slider

protected override void OnEnable()

{

 ...

 if (healthSlider)

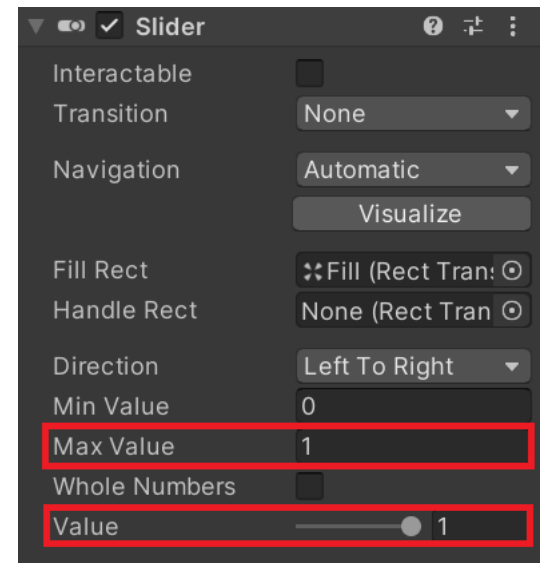
 {

 healthSlider.maxValue = maxHealth; // 최대 체력.

 healthSlider.value = health; // 현재 체력

 }

}



HP UI

```
public override void OnDamage(float damage, Vector3 hitPoint, Vector3 hitNormal)
{
    ...
    base.OnDamage(damage, hitPoint, hitNormal);
    healthSlider.value = health; // 데미지 적용 후, health 정보를 받아서 갱신.
}

// 체력 회복용 아이템을 사용 할 경우 호출.
// value 값 만큼 체력을 회복 시킨다.
public override void RestoreHealth(float value)
{
    base.RestoreHealth(value);
    healthSlider.value = health; // 회복 적용 후, health 정보를 받아서 갱신.
}
} // class PlayerHealth
```

Effects & Sounds

▶ Enemy

- Enemy 사망 시, **사망 효과음** 추가 / 피격 시, **피격 효과음과 이펙트**를 **추가**
- **AddComponent**를 이용하여 **AudioSource** 컴포넌트를 **추가**하여 사운드를 재생

```
public class Enemy : LivingEntity
{
    ...
    [SerializeField] private AudioClip deathSound; // 사망 효과음.
    [SerializeField] private AudioClip hitSound; // 피격 효과음.
    [SerializeField] private ParticleSystem hitEffect; // 피격 이펙트.
    private AudioSource audioSource; // 효과음을 출력하는데 사용.

    private void Awake()
    {
        ...
        // 현재 오브젝트에 AudioSource 컴포넌트 추가.
        audioSource = gameObject.AddComponent<AudioSource>();
        audioSource.playOnAwake = false; // 플레이 시, 사운드 실행되지 않도록 한다.
        OnDeath += () => {
            ...
            if (audioSource && deathSound) audioSource.PlayOneShot(deathSound); // 사망 효과음 1회 재생.
        };
    }
}
```

Effects & Sounds

```
public override void OnDamage(float damage, Vector3 hitPoint, Vector3 hitNormal)
{
    base.OnDamage(damage, hitPoint, hitNormal);

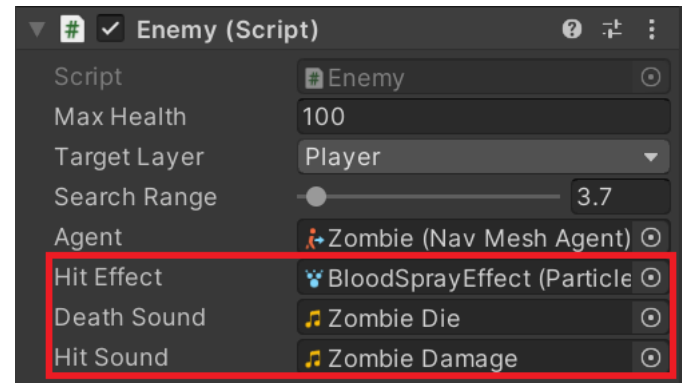
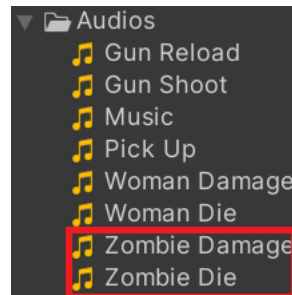
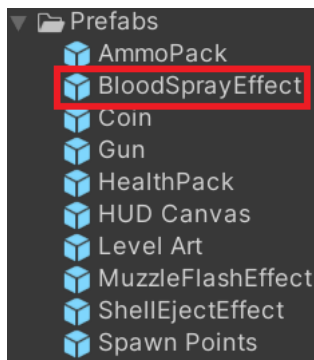
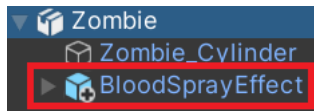
    if (anim && !isDead)
    {
        if (hitEffect)
        {
            var hitEffectTR = hitEffect.transform;
            hitEffectTR.position = hitPoint; // 이펙트를 피격 지점으로 이동.
            // 피격 당한 방향으로 회전.
            hitEffectTR.rotation = Quaternion.LookRotation(hitNormal);
            hitEffect.Play(); // 이펙트 재생.
        }

        // 피격 효과음 1회 재생.
        if (audioSource && hitSound) audioSource.PlayOneShot(hitSound);
        anim.SetTrigger("Damaged"); // 피격 애니메이션 실행.
    }
} // class Enemy
```

Effects & Sounds

▶ Zombie

- **Zombie GameObject**에 Prefabs/**BloodSprayEffect** **추가**
- Enemy Script의 **Hit Effect**에 추가한 **BloodSprayEffect**를 **등록**
- Enemy Script의 **Death Sound**와 **HitSound**에 각각 **Audios**의 **Zombie Die**와 **Zombie Damage**를 **등록**



Effects & Sounds

▶ PlayerHealth

- Player 사망 시, **사망 효과음** 추가 / 피격 시, **피격 효과음**과 **이펙트**를 **추가**
- **AddComponent**를 이용하여 **AudioSource** 컴포넌트를 **추가**하여 사운드를 재생

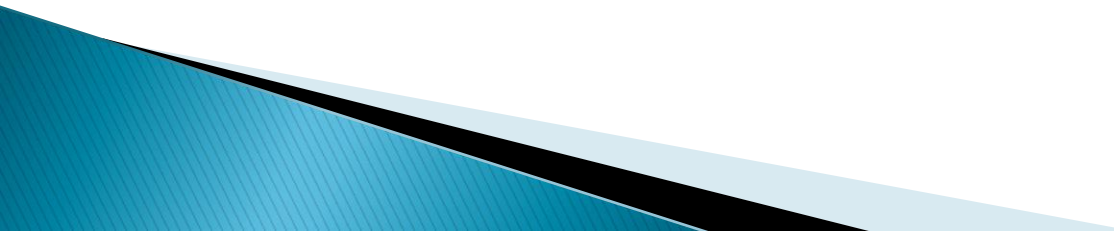
```
public class PlayerHealth : LivingEntity
{
    [SerializeField] private ParticleSystem hitEffect;
    [SerializeField] private AudioClip deathSound;
    [SerializeField] private AudioClip hitSound;
    private AudioSource audioSource;

    private void Awake()
    {
        ...
        audioSource = gameObject.AddComponent<AudioSource>();
        audioSource.playOnAwake = false;
        OnDeath += () =>
        {
            ...
            if (audioSource && deathSound) audioSource.PlayOneShot(deathSound);
        };
    }
}
```

Effects & Sounds

```
public override void OnDamage(float damage, Vector3 hitPoint, Vector3 hitNormal)
{
    if (!isDead && hitEffect)
    {
        hitEffect.transform.rotation = Quaternion.LookRotation(hitNormal);
        hitEffect.Play();

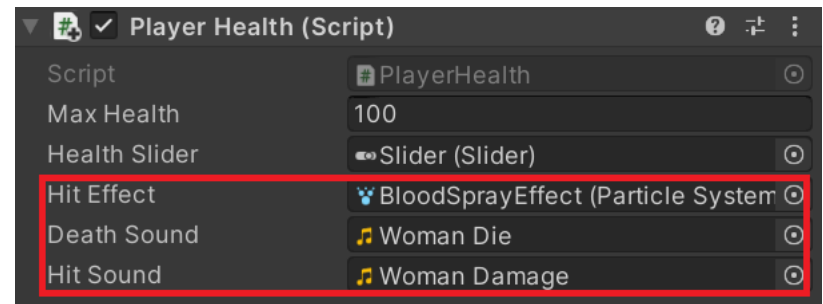
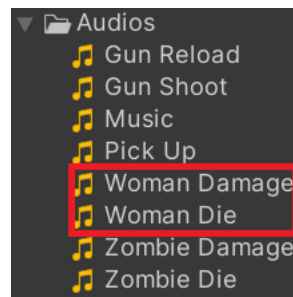
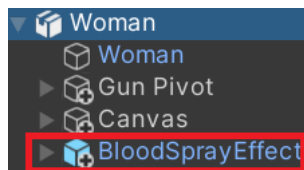
        if (audioSource && hitSound) audioSource.PlayOneShot(hitSound);
    }
    ...
}
} // class PlayerHealth
```



Effects & Sounds

▶ Woman

- Woman GameObject에 Prefabs/BloodSprayEffect **추가**
- PlayerHealth Script의 **Hit Effect**에 추가한 **BloodSprayEffect**를 **등록**
- PlayerHealth Script의 **Death Sound**와 **HitSound**에 각각 **Audios**의 **Woman Die**와 **Woman Damage**를 **등록**



Effects & Sounds

▶ Gun(Script)

- 총 발사 시, 이펙트와 효과음 적용
- 탄창 재장전 시, 효과음 적용

```
public class Gun : MonoBehaviour
{
    [Header("SFX")]
    [SerializeField] private ParticleSystem muzzleFlashEffect; // 총구의 화염 효과.
    [SerializeField] private ParticleSystem shellEjectEffect; // 탄피 배출 효과.
    [SerializeField] private AudioClip shootSound; // 총 발포 효과음.
    [SerializeField] private AudioClip reloadSound; // 탄창 재장전 효과음.
    private AudioSource audioSource;

    private void Awake()
    {
        audioSource = gameObject.AddComponent<AudioSource>();
        audioSource.playOnAwake = false;
        ...
    }
}
```


Effects & Sounds

```
private IEnumerator ShotEffect(Vector3 hitPosition)
{
    if (muzzleFlashEffect) muzzleFlashEffect.Play();
    if (shellEjectEffect) shellEjectEffect.Play();
    if (audioSource && shootSound) audioSource.PlayOneShot(shootSound);
    ...
}
```

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

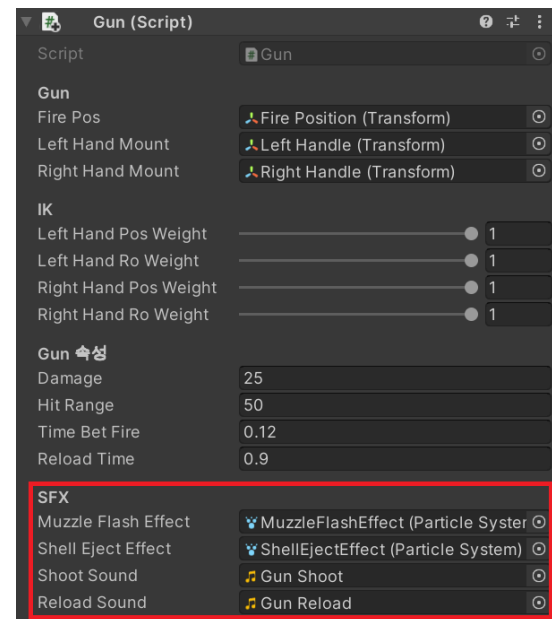
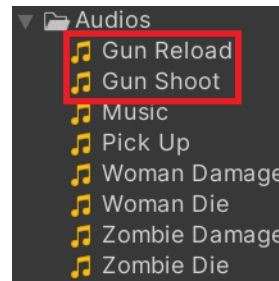
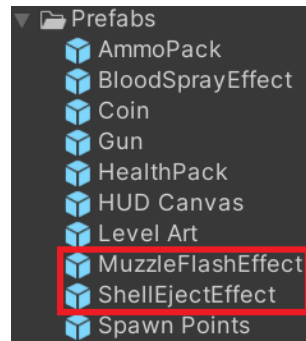
    if (audioSource && reloadSound) audioSource.PlayOneShot(reloadSound);

    ...
}
} // class Gun
```

Effects & Sounds

▶ Gun(GameObject)

- Woman이 지나고 있는 Gun(GameObject)에 Prefabs/ShellEjectEffect를 **추가**
- Gun(GameObject)의 Fire Position에 Prefabs/MuzzleFlashEffect를 **추가**
- Gun(Script)의 Muzzle Flash Effect와 Shell Eject Effect에 Gun(GameObject)에 추가한 MuzzleFlashEffect, ShellEjectEffect를 **등록**
- Gun(Script)의 Shoot Sound와 Reload Sound에 Audios/Gun Shoot, Audios/Gun Reload를 **등록**



Item

▶ GameMgr(Script)

- GameManager GameObject를 만들어 GameMgr 스크립트 **추가**
- Score 추가

```
public class GameMgr : MonoBehaviour
{
    private static GameMgr instance = null;
    public static GameMgr Instance
    {
        get
        {
            if (!instance)
            {
                instance = FindObjectOfType<GameMgr>();
                if(!instance) instance = new GameObject("GameMgr").AddComponent<GameMgr>();
                DontDestroyOnLoad(instance.gameObject);
            }

            return instance;
        }
    }
}
```

Item

```
private void Awake()
{
    // instance가 아닌 Instance 임을 주의!!
    if (this != Instance) Destroy(gameObject);
}

// 현재 달성한 점수.
private int score = 0;

public void AddScore(int value)
{
    score += value;
    // UI의 ScoreText를 갱신.
    UIMgr.Instance.UpdateScoreText(score);
}
} // class GameMgr
```

Item

► Gun(Script)

```
public class Gun : MonoBehaviour
{
    ...
    public void AddAmmo(int value)
    {
        ammoRemain += value;
    }
}
```

► PlayerShooter(Script)

```
public class PlayerShooter : MonoBehaviour
{
    ...
    public void AddAmmo(int value)
    {
        if (gun) gun.AddAmmo(value);
    }
}
```

Item

▶ BaseItem(Script)

- 사용 Item에 공통으로 적용되는 추상 클래스

```
abstract public class BaseItem : MonoBehaviour
{
    //해당 아이템을 습득, 사용가능 대상이 되는 레이어.
    [SerializeField] protected LayerMask targetLayer;
    [SerializeField] protected int value = 0; // 아이템 사용 시, 적용되는 값.

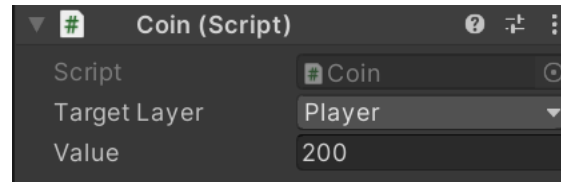
    virtual public bool Use(GameObject target)
    {
        // *.layer는 index 값을 가지고,
        // targetLayer는 bit shift(2^index)된 값을 가진다.
        if ((1 << target.layer).Equals(targetLayer))
        {
            gameObject.SetActive(false);
            return true;
        }
        return false;
    }

    public void SetPosition(Vector3 pos)
    {
        transform.position = pos;
        gameObject.SetActive(true);
    }
}
```

Item

▶ Coin(Script)

- C# 스크립트 **생성**
- Prefabs/**Coin.Prefab**에 **추가**
- LayerMask : Player
- Value : 200

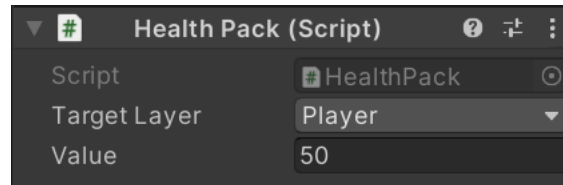


```
public class Coin : BaseItem
{
    public override bool Use(GameObject target)
    {
        if (base.Use(target))
        {
            GameMgr.Instance.AddScore(value);
            return true;
        }
        return false;
    }
}
```

Item

▶ HealthPack(Script)

- C# 스크립트 **생성**
- Prefabs/**HealthPack.Prefab**에 **추가**
- LayerMask : Player
- Value : 50

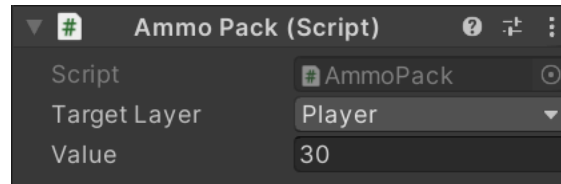


```
public class HealthPack : BaseItem
{
    public override bool Use(GameObject target)
    {
        if (base.Use(target))
        {
            var health = target.GetComponent<LivingEntity>();
            if (health) health.RestoreHealth(value);
            return true;
        }
        return false;
    }
}
```


Item

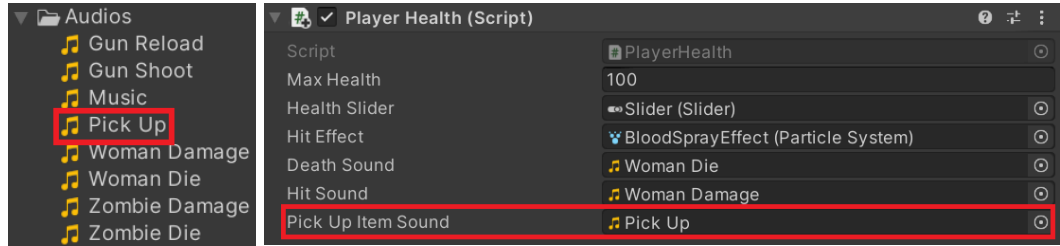
▶ AmmoPack(Script)

- C# 스크립트 **생성**
- Prefabs/**AmmoPack.Prefab**에 **추가**
- LayerMask : Player
- Value : 200



```
public class AmmoPack : BaseItem
{
    public override bool Use(GameObject target)
    {
        if (base.Use(target))
        {
            var shooter = target.GetComponent<PlayerShooter>();
            if (shooter) shooter.AddAmmo(value);
            return true;
        }
        return false;
    }
}
```

Item



▶ PlayerHealth(Script)

- 접촉(충돌) 대상이 아이템 이라면 Use() 함수(Method)를 호출
- 아이템 습득 효과음 출력
- 각 아이템의 Collider는 isTrigger:ture이기 때문에 OnTriggerEnter를 이용하여 접촉(충돌) 확인이 가능

```
public class PlayerHealth : LivingEntity
{
    [SerializeField] private AudioClip pickUpItemSound;
    ...
    private void OnTriggerEnter(Collider other)
    {
        // 모든 Item들은 추상 클래스 BaseItem을 상속 받기 때문에,
        // GetComponent를 이용하여 BaseItem을 찾아 *.Use() 함수를 호출하면,
        // 지금 접촉한 Item의 Use() 함수가 실행된다.
        var item = other.GetComponent<BaseItem>();
        if (item)
        {
            item.Use(gameObject);
            if (audioSource && pickUpItemSound) audioSource.PlayOneShot(pickUpItemSound);
        }
    }
}
```

Item Spawn

▶ Spawn(Script)

- 풀링을 이용한 오브젝트 스폰에 사용될 base class 작성
- 제네릭(Generic)을 이용하여 Prefab 배열을 구성하고 제네릭 T는 **MonoBehaviour** 형식

```
using UnityEngine.AI;
abstract public class Spawn<T> : MonoBehaviour where T : MonoBehaviour
{
    [SerializeField] protected T[] prefabs; // 생성할 리스트.
    [SerializeField] private float maxDistance = 5f; // 생성될 최대 반경.

    // pooling한 오브젝트 저장 공간.
    // random한 순서로 꺼낼 필요가 없을 경우 Queue를 사용.
    private List<T> pooling = new List<T>();
    public bool isEmptyPool { get { return !(0 < pooling.Count); } }

    abstract public void Updates();
    abstract protected void SpawnObj();
}
```

Item Spawn

```
protected void MakeObjPooling(int index) //오브젝트를 만들고 pooling 리스트에 추가한다.
{
    var select = Mathf.Min(prefabs.Length - 1, index);
    if (0 <= select)
    {
        var obj = Instantiate(prefabs[select]);
        if (obj)
        {
            pooling.Add(obj);
            obj.gameObject.SetActive(false);
        }
    }
}

// 시용이 끝난 오브젝트를 다시 pooling 리스트에 넣는다.
public void SetPooling(T obj)
{
    if (!pooling.Contains(obj)) pooling.Add(obj);
}
```

Item Spawn

```
private T GetObject(int index) // pooling 리스트의 index위치의 값을 제거하고 리턴.
{
    T obj = null;
    if (0 < pooling.Count)
    {
        obj = pooling[index];
        pooling.RemoveAt(index);
    }

    return obj;
}

protected T GetObjRandom() // pooling 리스트의 랜덤한 위치(index)의 값을 리턴.
{
    var select = Random.Range(0, pooling.Count);
    return GetObject(select);
}

protected T GetFirstObj() // pooling 리스트의 가장 첫 번째 값을 리턴.
{
    return GetObject(0);
}
```

Item Spawn

```
/// <summary>
/// NavMesh 위의 랜덤한 위치를 반환하는 메서드. <br/>
/// center를 중심으로 maxDistance 반경 안에서 랜덤한 위치를 찾는다.
/// </summary>
protected Vector3 GetRandomPointOnNavMesh(Vector3 center)
{
    // center를 중심으로 반지름이 maxDistance인 구 안에서의 랜덤한 위치 하나를 지정.
    // Random.insideUnitSphere : 반지름이 1인 구 안에서의 랜덤한 한 점을 반환.
    Vector3 randomPos = Random.insideUnitSphere * maxDistance + center;

    NavMeshHit navHit;
    // maxDistance 반경 안에서, randomPos에 가장 가까운 내비메시 위의 한 점을 찾는다.
    NavMesh.SamplePosition(randomPos, out navHit, maxDistance, NavMesh.AllAreas);

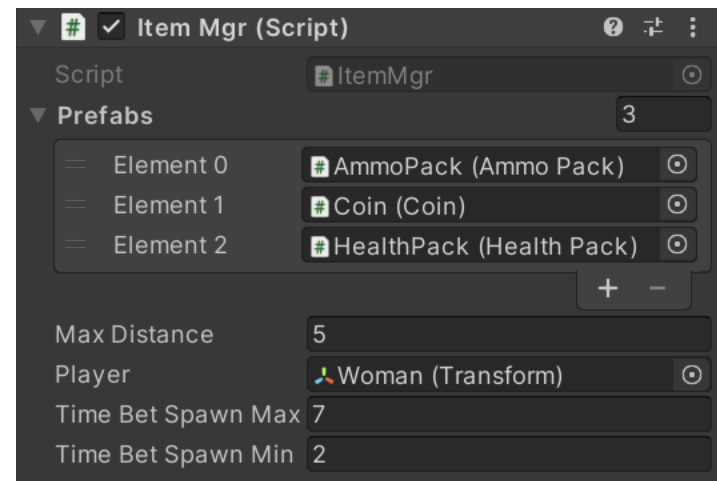
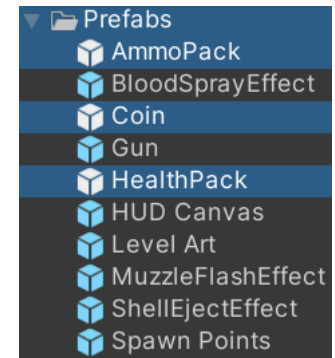
    return navHit.position;
}
} // abstract class Spawn
```

Item Spawn

▶ ItemMgr(Script)

- ItemMgr(GameObject) 생성하여 ItemMgr(Script) **추가**
- 플레이어(Woman)를 중심으로 **일정 반경** 안에 **아이템을 생성**
- 습득을 하지 않으면 **5초** 경과 후 **자동으로 제거**
- 생성 주기를 min~max 사이의 시간을 랜덤으로 구한다
- Prefabs/AmmoPack, Coin, HealthPack.prefab을 prefabs에 **등록**
- Player에 Woman(GameObject)를 **등록**

```
public class ItemMgr : Spawn<BaseItem>
{
    public static ItemMgr Instance { get; private set; }
    private void Awake()
    {
        if (!Instance)
        {
            Instance = this;
            return;
        }
        Destroy(gameObject);
    }
}
```



Item Spawn

```
[SerializeField] private Transform player; // 플레이어 위치를 확인할 Transform.
```

```
[SerializeField] private float timeBetSpawnMax = 7f; // 생성 최대 시간 간격.
```

```
[SerializeField] private float timeBetSpawnMin = 2f; // 생성 최소 시간 간격 .
```

```
private float timeBetSpawn; // 실제 생성 간격.
```

```
private void Start()
```

```
{
```

```
    // 아이템 생성 시간 구하기.
```

```
    timeBetSpawn = Random.Range(timeBetSpawnMin, timeBetSpawnMax);
```

```
    for (int i = 0; prefabs.Length > i; i++)
```

```
    {
```

```
        MakeObjPooling(i);
```

```
    }
```

```
}
```

```
public override void Updates()
```

```
{
```

```
    if (0 >= (timeBetSpawn -= Time.deltaTime) && player)
```

```
    {
```

```
        // 생성 시간 재 설정.
```

```
        timeBetSpawn = Random.Range(timeBetSpawnMin, timeBetSpawnMax);
```

```
        SpawnObj();
```

```
    }
```

```
}
```


Item Spawn

```
protected override void SpawnObj()
{
    if (isEmptyPool)
    {
        var select = Random.Range(0, prefabs.Length);
        MakeObjPooling(select);
    }
    var item = GetObjRandom();
    Vector3 spawnPosition = GetRandomPointOnNavMesh(player.position);
    spawnPosition += Vector3.up * 0.5f; // 바닥에서 0.5만큼 위로 올린다.
    item.SetPosition(spawnPosition);
    StartCoroutine(LateInactive(item.gameObject));
}

private IEnumerator LateInactive(GameObject obj)
{
    var timer = 5f;
    while (0 < (timer -= Time.deltaTime))
    {
        if (!obj.activeSelf) yield break;
        yield return null;
    }
    obj.SetActive(false);
}
} // class ItemMgr
```

Item Spawn

▶ BaseItem(Script)

- 아이템이 비활성화(사라지거나 습득 시) 하면 **pooling 리스트에 다시 추가**

```
abstract public class BaseItem : MonoBehaviour
{
    virtual protected void OnDisable()
    {
        if(ItemMgr.Instance) ItemMgr.Instance.SetPooling(this);
    }
}
```

▶ GameMgr(Script)

- ItemMgr의 Updates()를 GameMgr의 Update()에서 **호출**

```
public class GameMgr : MonoBehaviour
{
    private void Update()
    {
        if(ItemMgr.Instance) ItemMgr.Instance.Updates();
    }
}
```

Enemy Spawn

▶ Enemy(Script)

- **Zombie에 Renderer를 등록**
- Enemy Spawn 시에 해당 **Enemy의 정보를 Setup**한다
- Enemy의 **능력치에 따라 색을 변경**

```
public class Enemy : LivingEntity
```

```
{
```

```
    [SerializeField] private Renderer enemyRenderer;
```

```
    public void Setup(float damage, float maxHealth, float speed, Color color, Vector3 pos)
```

```
{
```

```
        this.damage = damage;
```

```
        this.maxHealth = maxHealth;
```

```
        if(agent) agent.speed = speed;
```

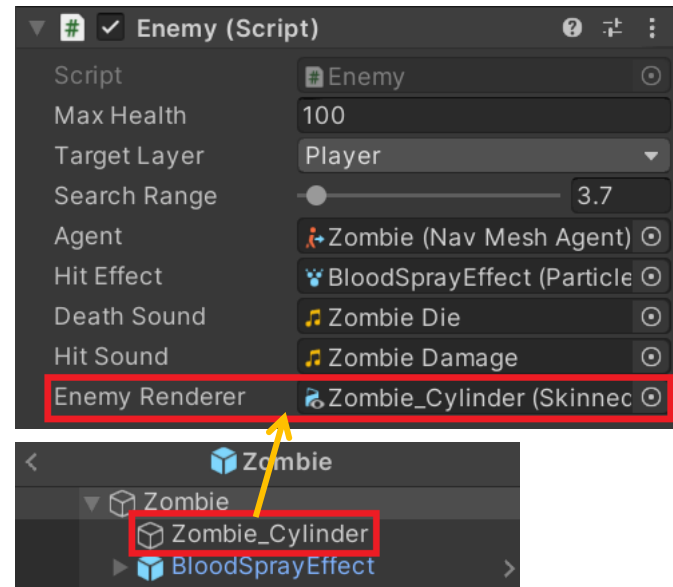
```
        if (enemyRenderer) enemyRenderer.material.color = color;
```

```
        transform.position = pos;
```

```
        gameObject.SetActive(true);
```

```
}
```

```
}
```



Enemy Spawn

▶ GameMgr(Script)

- Wave 정보와 Enemy의 Spawn 수를 설정

```
public class GameMgr : MonoBehaviour
{
    [SerializeField] private int maxWave = 10; // 최대 Wave 카운트.
    public int wave { get; private set; } = 0; // 현재 Wave 카운트.
    // 이번 Wave에 출현할 Enemy의 수.
    public int enemySpawnCount { get { return Mathf.RoundToInt(wave * 1.5f); } }

    public bool NextWave()
    {
        if (maxWave > wave)
        {
            wave++;
            return true;
        }

        return false;
    }
}
```

Enemy Spawn

▶ EnemyMgr(Script)

- Enemy의 공격력, 생명력, 이동 속도 등을 랜덤하게 뽑아 배치

```
public class EnemyMgr : Spawn<Enemy>
{
    public static EnemyMgr Instance { get; private set; }

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

Enemy Spawn

```
[SerializeField] private float damageMax = 40f; // 최대 공격력.  
[SerializeField] private float damageMin = 20f; // 최소 공격력.  
  
[SerializeField] private float healthMax = 200f; // 최대 체력.  
[SerializeField] private float healthMin = 100f; // 최소 체력.  
  
[SerializeField] private float speedMax = 3f; // 최대 속도.  
[SerializeField] private float speedMin = 1f; // 최소 속도.  
  
[SerializeField] private Color strongColor = Color.red; // 강한 적 AI가 가지게 될 피부색.  
  
private int spawnCount = 0; // 필드에 존재하는 Enemy의 수.  
  
private void Start()  
{  
    for (int i = 0; 50 > i; i++)  
    {  
        MakeObjPooling(0);  
    }  
}  
  
public override void Updates()  
{  
    if (0 >= spawnCount)  
    {  
        SpawnObj();  
    }  
}
```

Enemy Spawn

```
private void SetupEnemy(float intensity)
{
    // pooling list에 사용가능한 오브젝트가 없다면 추가로 생성.
    if (isEmptyPool) MakeObjPooling(0);

    var enemy = GetFirstObj();
    if (enemy)
    {
        // 필드에 존재하는 Enemy의 수를 확인하기 위하여 사용.
        spawnCount++;

        // Lerp(a, b, t):최소(a), 최대(b) 값 범위의 값을 t값(0~1)을 참조하여 보간.
        var damage = Mathf.Lerp(damageMin, damageMax, intensity);
        var health = Mathf.Lerp(healthMin, healthMax, intensity);
        var speed = Mathf.Lerp(speedMin, speedMax, intensity);
        var color = Color.Lerp(Color.white, strongColor, intensity);
        var pos = GetRandomPointOnNavMesh(Vector3.zero);
        enemy.Setup(damage, health, speed, color, pos);
    }
}
```

Enemy Spawn

```
public void DecreaseSpawnCount()
{
    // spawnCount를 감소하고 UI정보를 갱신한다.
    UIManager.Instance.UpdateWaveText(GameMgr.Instance.wave, --spawnCount);
}

protected override void SpawnObj()
{
    // wave 생성이 가능한지 확인.
    if (GameMgr.Instance.NextWave())
    {
        // 현재 wave에 생성되는 enemy의 수를 확인하여 생성.
        var count = GameMgr.Instance.enemySpawnCount;
        for (int i = 0; count > i; i++) SetupEnemy(Random.value);

        // 현재 wave UI정보를 갱신.
        UIManager.Instance.UpdateWaveText(GameMgr.Instance.wave, spawnCount);
    }
}

} // class EnemyMgr
```


Enemy Spawn

▶ Enemy(Script)

- Enemy가 죽으면 **score**를 증가하고 남은 **enemy** 수를 갱신한다
- 해당 **오브젝트**가 **비활성화** 되면 **enemy pooling list**에 돌려주어 **재사용** 한다

```
public class Enemy : LivingEntity
{
    private void Awake()
    {
        ...
        OnDeath += () =>
        {
            ...
            GameMgr.Instance.AddScore(100); // enemy 처치 시, 100 score 상승.
            EnemyMgr.Instance.DecreaseSpawnCount(); // enemy 처치 시, Spawn Count 감소.
        };
    }

    private void OnDisable()
    {
        if(EnemyMgr.Instance) EnemyMgr.Instance.SetPooling(this);
    }
}
```

Enemy Spawn

▶ GameMgr(Script)

- EnemyMgr의 Updates()를 GameMgr의 Update()에서 호출

```
public class GameMgr : MonoBehaviour
{
    private void Update()
    {
        if(ItemMgr.Instance) ItemMgr.Instance.Updates();
        if(EnemyMgr.Instance) EnemyMgr.Instance.Updates();
    }
}
```

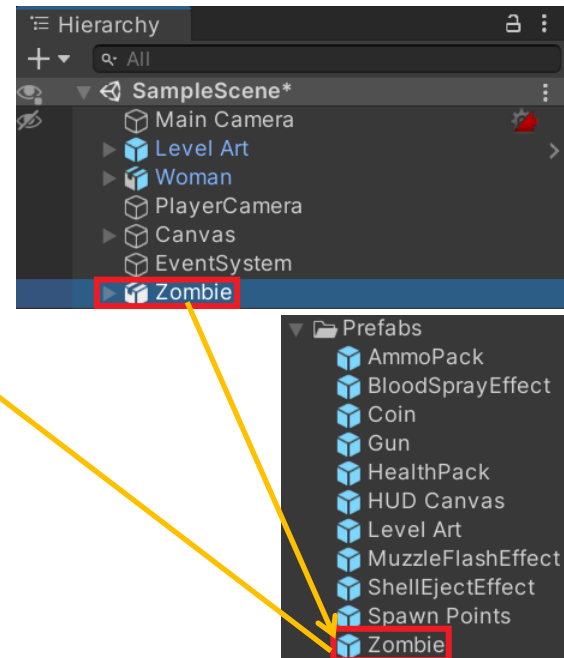
Enemy Spawn

▶ Zombie(GameObject)

- Hierarchy에 있는 **Zombie(GameObject)**를 Prefabs 폴더에 추가하여 **Prefab으로 만든다**

▶ EnemyMgr

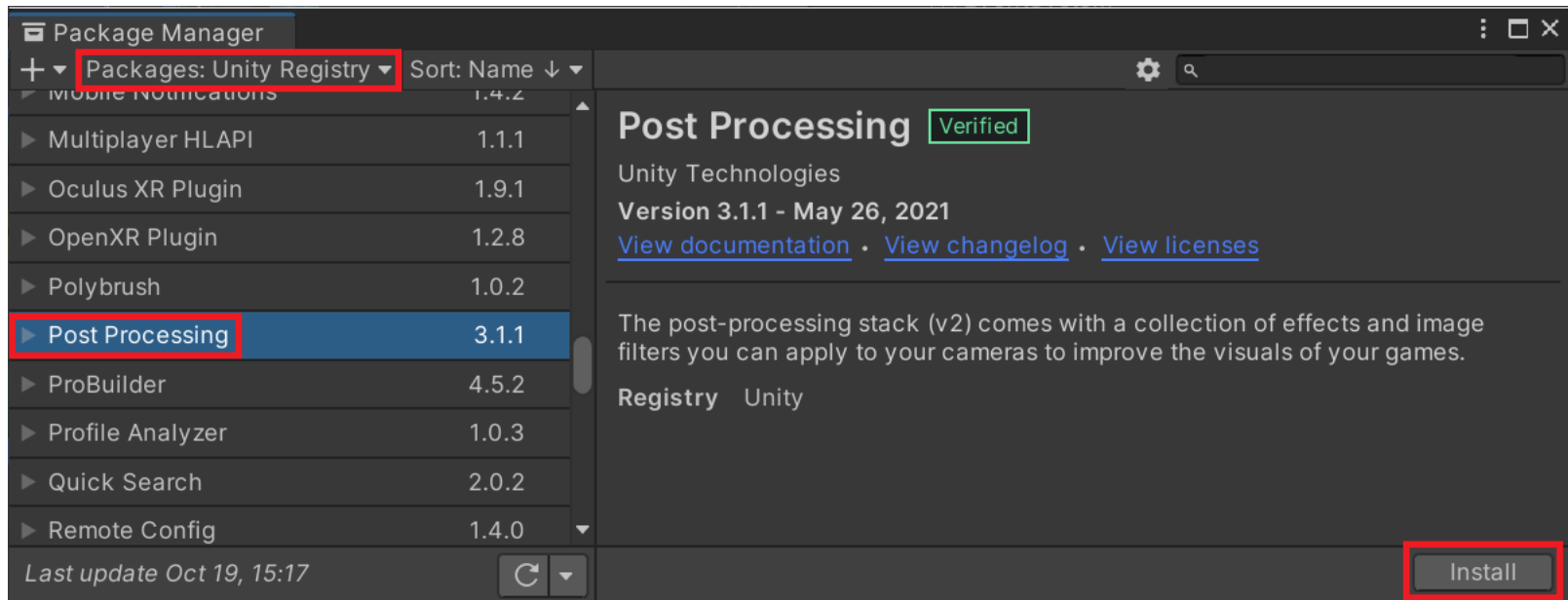
- EnemyMgr(GameObject)를 **생성**하고 EnemyMgr(Script) **추가**
- Prefabs/Zombie.prefab을 Prefabs에 **등록**



포스트 프로세싱(Post Processing)

▶ Post Processing

- 최종 렌더링 시에 추가 효과를 더하여 화면을 출력한다
- Package Manager에서 Post Processing 설치



포스트 프로세싱(Post Processing)

▶ Layer

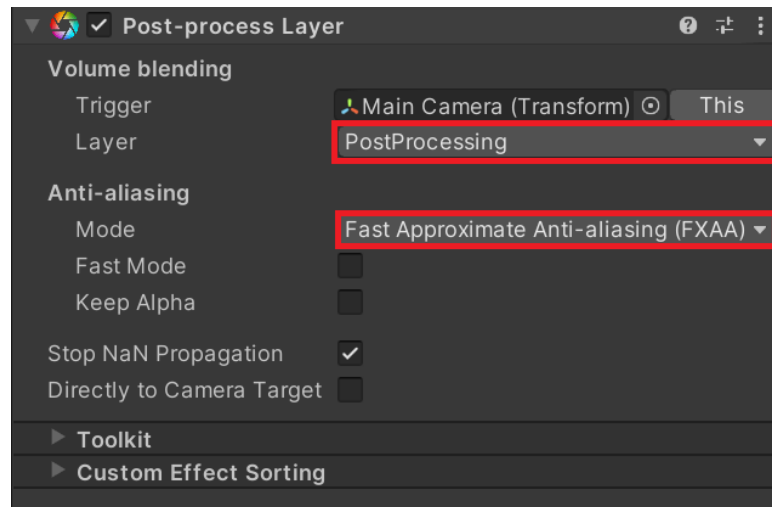
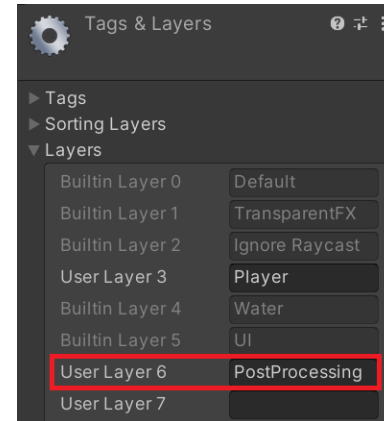
- PostProcessing **추가**

▶ Camera

- **메인 카메라에** Post-process Layer **Component 추가**

▶ Post-process Layer

- Trigger : Main Camera
- Layer : PostProcessing
- Anti-aliasing Mode : FXAA



포스트 프로세싱(Post Processing)

▶ Post-process Volume

- 새 오브젝트 생성, Post-process Volume 컴포넌트 **추가**
- Is Global : true
- Profile : Post-Process Profile/Global Profile

