구현이 필요한 주요 내용.

1. 좀비들이 '쌓인다'.
2. 쌓인 좀비들은 순환한다(아래쪽에 있는 좀비들은 뒤로 밀려나야한다).

'쌓인다' 는 물리적인 상호작용이라 콜라이더, 특히 2d게임이므로 2d 콜라이더를 사용해야했습니다.
최초에는 Box 콜라이더를 사용했지만 미끄러지는 효과를 조금 더 쉽게 보이기 위해서는 Circle 콜라이더가 적합할 것이라고 생각했습니다.

순환이라는 것을 조금 구분해서 구현해야 했습니다. 영상(링크)에서 위에 있는 좀비들이 아래에 있는 좀비를 뒤로 밀어내는 모양으로 순환이 진행됩니다. 좀비가 먼저 위에 올라가야 하고(점프) 아래에 깔린 좀비는 뒤로 밀려나야 합니다. 이 순환은 좀비가 플레이어를 공격하거나 플레이어에게 달려가는 동안 발생합니다. 특히 점프하는 행동은 좀비가 더 이상 앞으로 나아갈 수 없을 때(다른 좀비에 의해) 발생하는 행동으로 앞에 있는 좀비를 감지해야 합니다.

아래쪽에 있는 좀비가 뒤로 밀려나기 위해서는 자신을 위에서 누르는 좀비가 있어야 합니다. 이미 '쌓인다'에서 물리계산을 꽤나 많이 사용하고 있기에 이 감지를 트리거가 아닌 다른 방식으로 구현해야 한다고 생각했고 RayCast를 사용하기로 했습니다.

따라서 구현 순서는 다음과 같습니다.

1. 좀비는 플레이어를 향해서 이동한다.
2. 이동하는 좀비는 내 앞에 다른 좀비가 있는 경우 이를 감지하고 점프해야 한다.
3. 좀비는 자신의 머리 위에 다른 좀비가 올라올 경우 뒤로 밀려나야 한다.
4. 뒤로 밀려나는 것은 공격, 이동 중에 가능하다.

구현이 필요한 주요 내용.

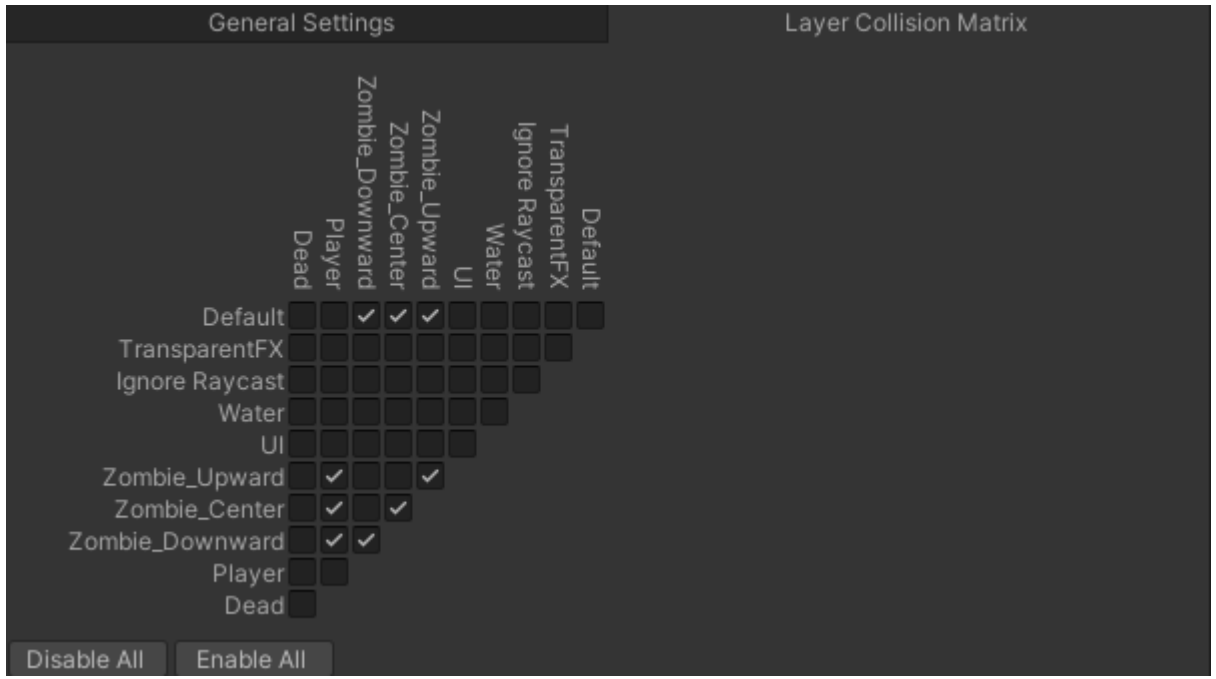
1페이지의 영상(링크)에서 좀비들은 자신만의 라인이 있는 것처럼 보입니다. 도로 이미지의 내부에서 각 좀비들만의 길이 있는 것처럼 보였습니다. 이 때 서로 다른 길의 좀비들과는 충돌하지 않는 것으로 보였고 충돌 레이어를 나눠 놓은 것으로 이해했습니다.

우측 이미지는 '쌓여 있다'의 예시 이미지입니다.
2차원 평면에서 오른쪽 이미지와 같은 모습을 보이기 위해서 빈 공간이 없어야 합니다. 이 때문에 실제 좀비의 스프라이트 크기보다 더 작은 콜라이더가 필요했습니다.

구현 내용.

좀비의 스프라이트 보다 더 작은 크기의 Circle콜라이더 적용했습니다.
BoxPrefab의 Hpslider를 회전시켜 좀비의 체력바를 구현했습니다.

구현 내용.



충돌 레이어를 각각 다르게 설정하여 각 오브젝트간 충돌을 세분화.
좀비들은 생성 위치에 따라 자신만의 충돌 레이어를 갖게 되며 자신과 같은 충돌 레이어 및 플레이어와만 충돌하도록 설정했습니다. 좀비가 사망한 경우 레이어를 Dead로 바꿔 모든 충돌을 무시하게 합니다. 다른 좀비에 의해 넉백되면 레이어를 Default로 바꿔서 모든 좀비들과 상호작용하도록 해봤습니다.

구현 내용.

```
public class Zombie : MonoBehaviour, IDamageable
{
    [SerializeField] private Slider hpSlider;  HpSlider (Slider)
    [SerializeField] private TextMeshProUGUI dmgText;  Text (TMP) (TextMeshProUGUI)

    //데미지 표기 지속 시간
    private readonly WaitForSeconds _damageTextDelay = new (0.5f);

    //넉백에 의해 레이어변경 시, 변경 전 자신의 레이어
    private int _myLayerMask;

    //넉백 힘
    public readonly Vector2 PushVector = new Vector2(5f, 0f);

    // 자신의 위에 좀비가 있는지 없는지 확인할 때 사용되는 Ray의 시작 위치 보간
    public readonly Vector3 DetectRayPosGap = new Vector3(-0.3f, 0.9f, 0f);

    private const float Damage = 10f;
    private Player _player;
    Frequently called  8 usages
    public ZombieState CurrentState { get; private set; }

    /// <summary>
    /// 기본 상태, 공격 상태, 사망 상태, 점프 상태
    /// </summary>
    Frequently called  5 usages
    public ZombieRun RunState { get; private set; }
    4 usages
    public ZombieAttack AttackState { get; private set; }
    Frequently called  3 usages
    public ZombieDie DieState { get; private set; }
    Frequently called  2 usages
    public ZombieJump JumpState { get; private set; }

    public readonly int IsIdle = Animator.StringToHash( name: "IsIdle");
    public readonly int IsAttacking = Animator.StringToHash( name: "IsAttacking");
    public readonly int IsDead = Animator.StringToHash( name: "IsDead");
```

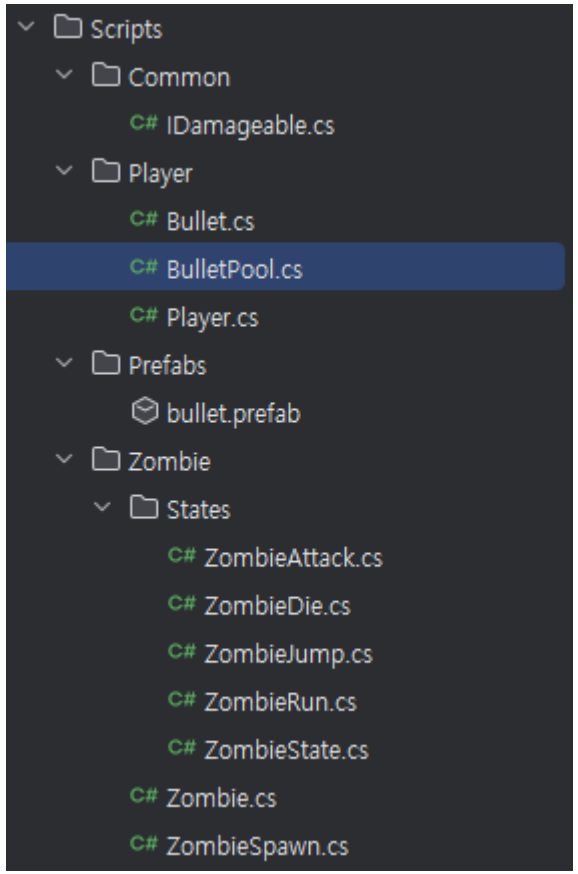
```
2 usages  More...
public Animator Animator { get; private set; }
Frequently called  7 usages
public Rigidbody2D Rigidbody2D{ get; private set; }
```

상태패턴을 적용하여 이동, 공격, 점프, 사망을 정의하고 각 상태에 따른 로직을 따로 구현했습니다.

공격 상태를 구현하고 애니메이션을 실행하는 도중 애니메이션 이벤트가 적용된 것을 확인했고 공격 상태에서 플레이어에게 데미지를 주는 것이 아닌 해당 이벤트를 활용하여 데미지를 주도록 변경했습니다.

전체 코드는 요청하신 대로, 깃 허브에 업로드하겠습니다.

구현 내용.



스크립트 폴더 구조는 다음과 같습니다.

사용된 주요 패턴은 오브젝트 풀, 싱글톤, 상태패턴입니다.

오브젝트 풀의 경우 Bullet과 좀비에 적용했고 싱글톤은 BulletPool에 적용했습니다. 상태패턴은 좀비의 구현에 적용했습니다.

Bullet을 오브젝트 풀로 구현했던 이유는 최초에는 직접 마우스의 왼쪽 버튼을 클릭하여 총을 발사하는 방식으로 구현했기 때문입니다. 이 후 영상을 다시 확인했을 때, 자동으로 발사하는 것 처럼 보여서 자동으로 발사하는 방식으로 수정했습니다.

간단한 소감

평소에 개인 프로젝트를 하면서 느낀 것은 사용되는 에셋들이 정말 중요하다는 것을 느꼈습니다.
게임 컨셉이나 기획에 맞는 에셋을 구하는 것은 매우 어려운 일이었기 때문입니다. 물론 에셋스토어나 AI사이트를 활용하는 것도 방법이지만 이는 한계가 명확했습니다.

이번 과제를 진행하면서 이 과제만을 위한 에셋이 제공되어 과제를 진행하는 동안 더 많은 즐거움을 느꼈습니다.

또 한, 주어진 에셋들의 세부 내용들(애니메이션 이벤트, 박스 프리팹에 설정된 HP슬라이드 바 등)이 정해져 있었고 이 틀 안에서 과제를 진행하는 경험도 좋았습니다. 개인 프로젝트였다면 제가 원하는 대로 진행해도 되었지만 이번 과제에서는 마치 '나를 활용해 달라' 라는 것처럼 느껴지는 요소들도 있었기 때문입니다.

어떤 기능을 구현하는 방식은 다양하지만 어떤 방식을 선호하는 지는 모든 사람들이 다를 것이라고 생각하고 있었습니다. 이 과제를 제공한 팀의 프로젝트 진행 간 구현 방식은 이렇다 라는 것을 간접적으로 느낀 것 같습니다.