팀스파르타 과제 보고서

250304_이의환

1. 과제 수행 목차

1 : 움직이는 2D 배경 구현

2 : 몬스터 스포너 및 움직임 구현

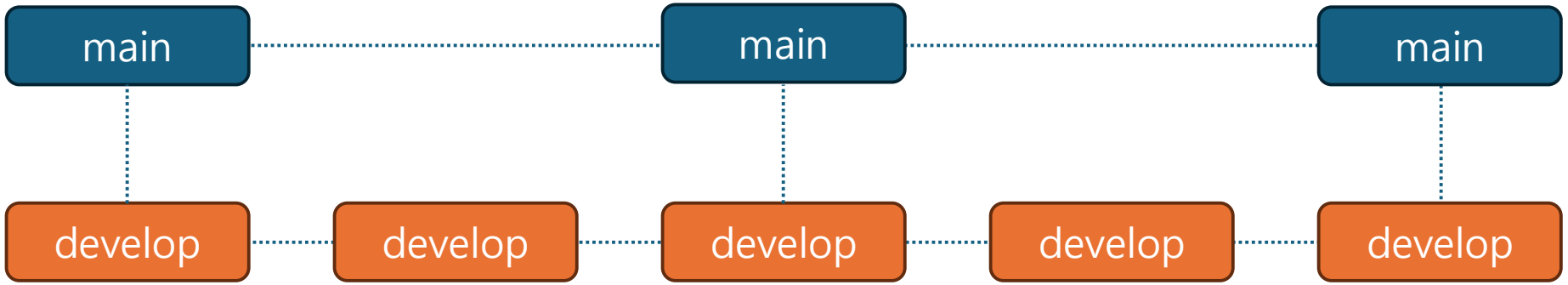
3 : Box 오브젝트 & 몬스터 충돌처리 구현

- 충돌처리: 박스 앞에 있을 시 공격 모션 추가
- 충돌처리: 충돌에 따른 애니메이션 스피드 감소

4 : 몬스터 움직임 알고리즘 구현

5 : 추가 구현

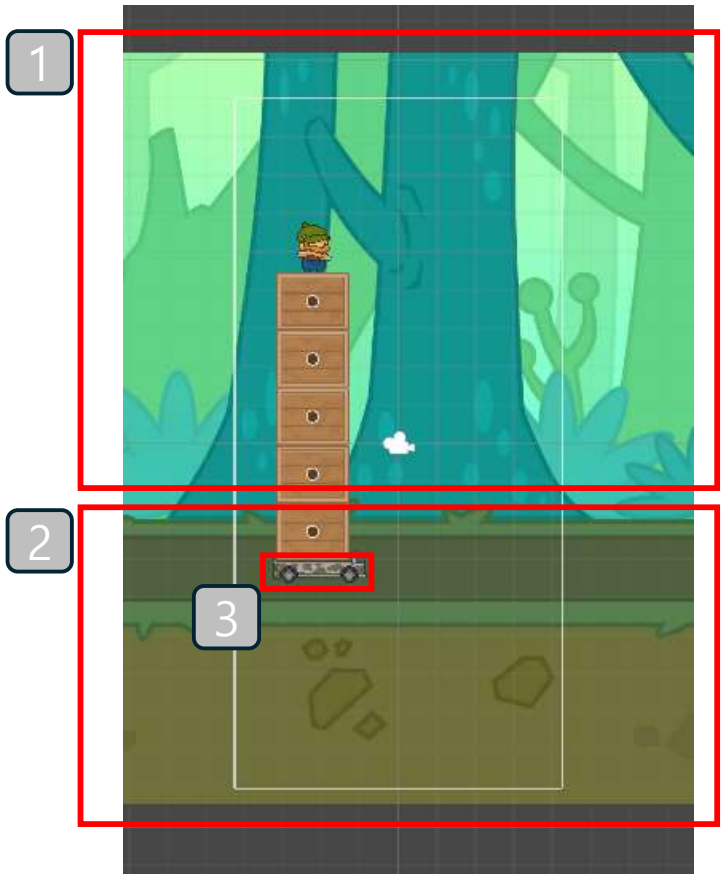
2. Git 형상관리



Git 형상 관리는 1일차에 main branch만 이용하다 develop branch만 따로 만들어
하루마다 과제 수행한 내용 버그 없이 해결하고 main에 합병하는 형식으로 관리하였습니다.

Unity 버전은 2022.3.42f1을 사용했습니다.

1. 움직이는 2D 배경 구현

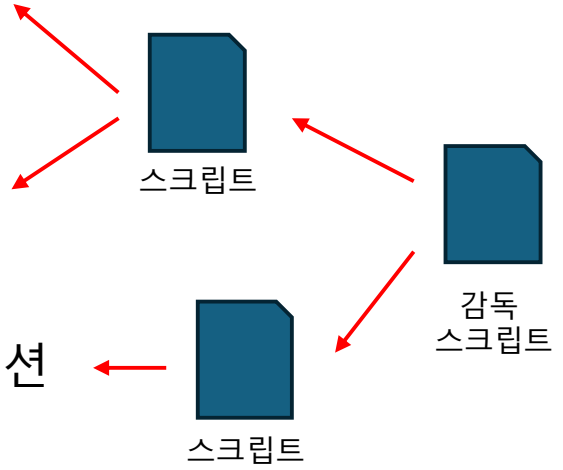


<움직이는 오브젝트 구현 구상도>

1 : 뒷 배경 나무 숲 이동 애니메이션

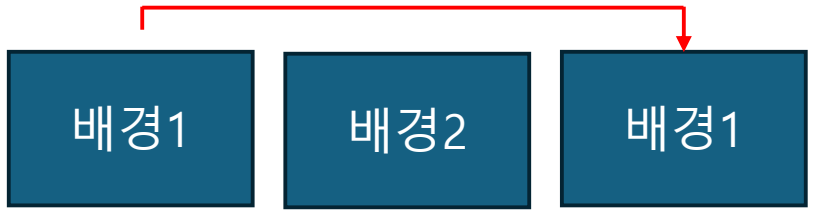
2 : 앞 배경 길 이동 애니메이션

3 : 길 이동에 따른 바퀴 회전 애니메이션



2025-03-05 / 1일차

1. 움직이는 2D 배경 구현



time

2가지 배경을 이용하여 시간이 지남에 따라 정해진 범위를 넘으면 뒤에 있는 배경이 앞으로 배치되게 하여

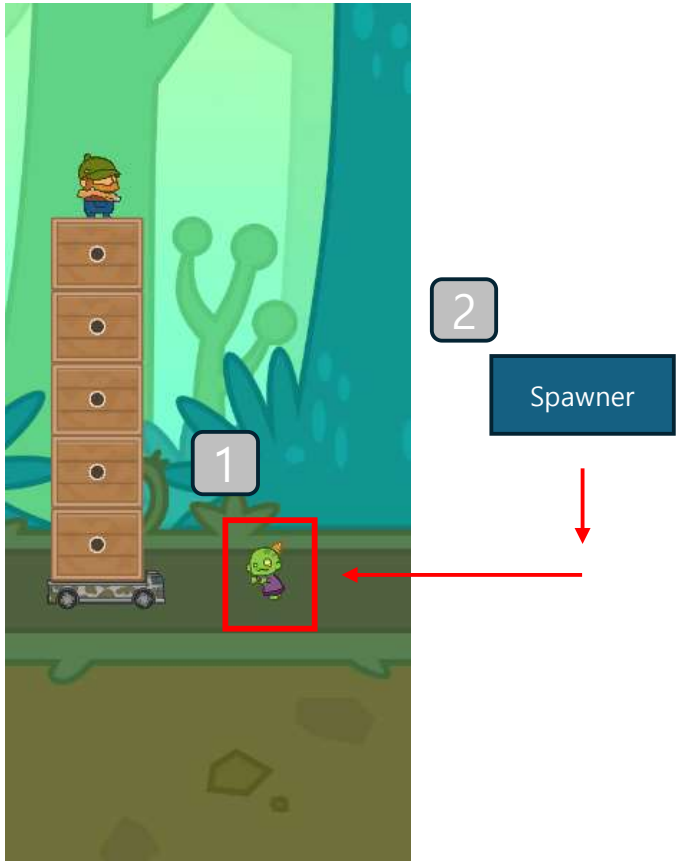
트럭 주체가 움직이기 보다 배경이 움직이게 구현하였습니다.
움직임에 따라 트럭 자체의 위치는 변하지 않아서
트럭 자체를 움직이기 보다 도로와 배경을 스피드 값을 따로 주어 역동적으로 애니메이션이 표현되도록 구현하였습니다.

```
// Update is called once per frame
Unity 메시지 | 참조 0개
void Update()
{
    // 설정 방향대로 배경 이동
    transform.position += moveDir * moveSpeed * Time.deltaTime;

    // 배경이 카메라 범위 내 벗어나면 재설정
    if(transform.position.x <= -scrollAmount)
    {
        Debug.Log(target.position);
        Debug.Log(moveDir * scrollAmount * 2);
        this.transform.position = target.position - (moveDir * scrollAmount * 2);
    }
}
```

Reference: https://www.youtube.com/watch?v=uyttyyB1HjI

2025-03-05 / 1일차

2. 몬스터 스포너 및 움직임 구현



<1차 몬스터 스포너 및 움직임 구상도>

1 : 몬스터 Prefab을 이용 전용 Controller로 이후 몬스터 AI 구현 ← 스크립트

2 : 몬스터 Prefab을 특정 시간마다 소환하는 스포너 ← 스크립트

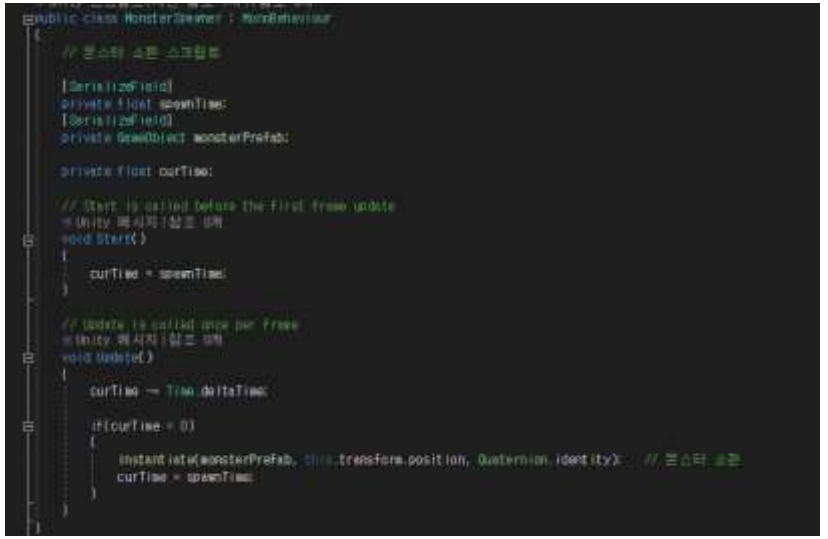
2025-03-05 / 1일차

2. 몬스터 스포너 및 움직임 구현

1차 구현

MonsterSpawner

1. 정해진 시간 대로 몬스터 소환



MonsterSpawner

MonsterController

1. 소환되면 왼쪽으로 이동

```
Unity 메시지 | 참조 0개
private void Update()
{
    // 소환될 시 기본적으로 옆으로 이동
    transform.position += moveDir * moveSpeed * Time.deltaTime;
}
```

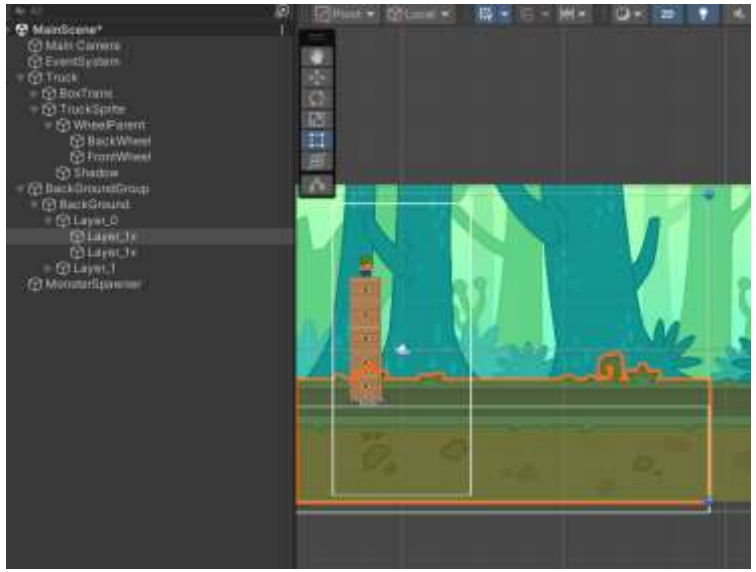
MonsterController

2025-03-05 / 1일차

2. 몬스터 스포너 및 움직임 구현

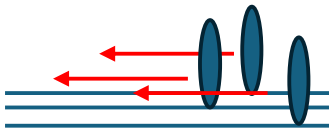
게임 안에서 **"몬스터의 이동 경로는 총 3가지"**가 있다는 것을 확인한 후 기존 1개의 길을 3가지 길로 변경하고 이후 이 경로에 따라 여러 상호작용을 구현해야겠다는 생각이다.

<3가지 길 경로 배치 구상도>



기존

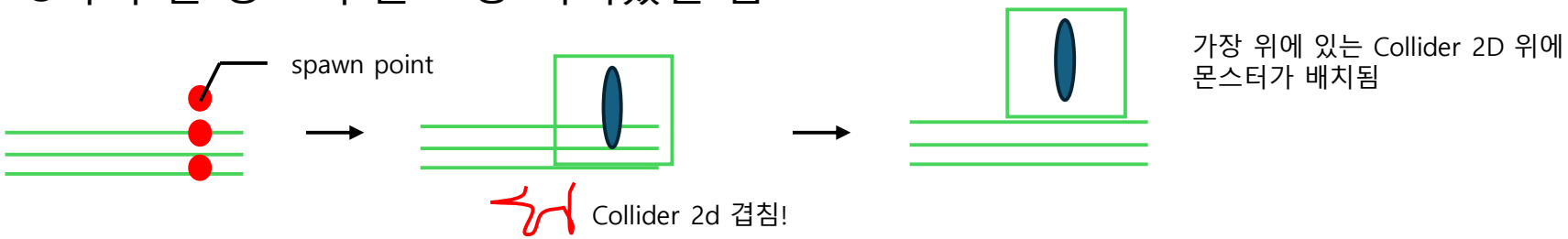
변경해야 할 점



2025-03-06 / 2일차

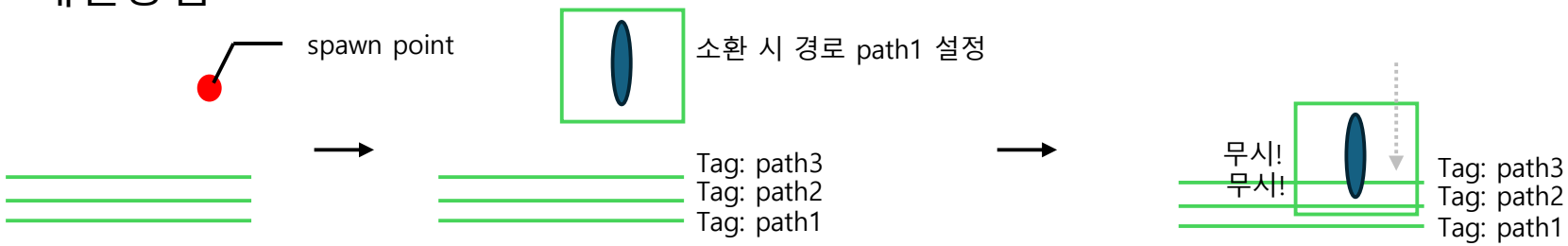
2. 몬스터 스포너 및 움직임 구현

<3가지 길 경로 구현 도중 어려웠던 점>



내가 원하는 그림은 **3가지 spawn point를 지정하여 Instantiate 함수를 실행하면 각각의 경로에서 움직**일 줄 알았으나
몬스터의 Collider와 3가지 경로의 Collider가 모두 겹쳐 결국 가장 위에 있는 경로로 몬스터가 배치되는 문제가 발생하였다.

➔ 해결방법



Unity의 **Physics2D.IgnoreCollision 함수를 사용하면 특정 Collider 객체의 충돌을 모두 무시**할 수 있다는 것을 알고
Spawn point를 한 가지만 설정하고 소환 시 3가지 경로 중 나머지 2개의 경로 collider는 모두 무시하도록 코드를 수정하였습니다.

2025-03-06 / 2일차

2. 몬스터 스포너 및 움직임 구현

객체 관리 자료구조는 배열과 리스트를 이용

우선 랜덤으로 경로를 설정하고
3가지 경로 Collider 배열을 담은
리스트를 이용 Monster 객체에게
너 남은 2가지 무시해라 적용

```
참조 1개
public void SetPath(int myPath, Collider2D myCollider, Collider2D[] ignorePath1, Collider2D[] ignorePath2)
{
    // 주어진 경로 지정
    spawnPath = myPath;

    // 정한 path 외 다른 path 충돌처리 무시
    IgnoreCollisionWithPaths(myCollider, ignorePath1);
    IgnoreCollisionWithPaths(myCollider, ignorePath2);
}

참조 2개
private void IgnoreCollisionWithPaths(Collider2D myCollider, Collider2D[] ignorePath)
{
    foreach (Collider2D pathCollider in ignorePath)
    {
        if (pathCollider != null && myCollider != null)
        {
            Physics2D.IgnoreCollision(myCollider, pathCollider);
        }
    }
}
```

MonsterController

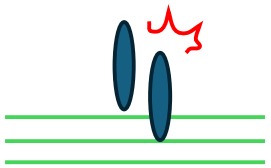
```
private void SetMonsterPath(GameObject monster)
{
    // 경로 3개 중 1개를 랜덤으로 선택
    int randomPathIndex = Random.Range(0, 3);

    Collider2D myCollider = monster.GetComponent<Collider2D>();
    Collider2D[] collider1 = null;
    Collider2D[] collider2 = null;

    Debug.Log("선택경로: " + randomPathIndex);

    // 경로 선택
    switch (randomPathIndex)
    {
        case 0:  // Path1 선택
            collider1 = pathColliders[0];
            collider2 = pathColliders[1];
            break;

        case 1:  // Path2 선택
            collider1 = pathColliders[1];
            collider2 = pathColliders[2];
            break;

        case 2:  // Path3 선택
            collider1 = pathColliders[2];
            collider2 = pathColliders[0];
            break;
    }

    // 몬스터에게 경로 전달
    MonsterController monsterController = monster.GetComponent<MonsterController>();
    monsterController.SetPath(randomPathIndex, myCollider, collider1, collider2);
}
```

MonsterSpanwer

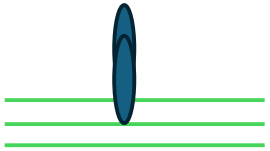
2025-03-06 / 2일차

3. Box 오브젝트 & 몬스터 충돌처리 구현

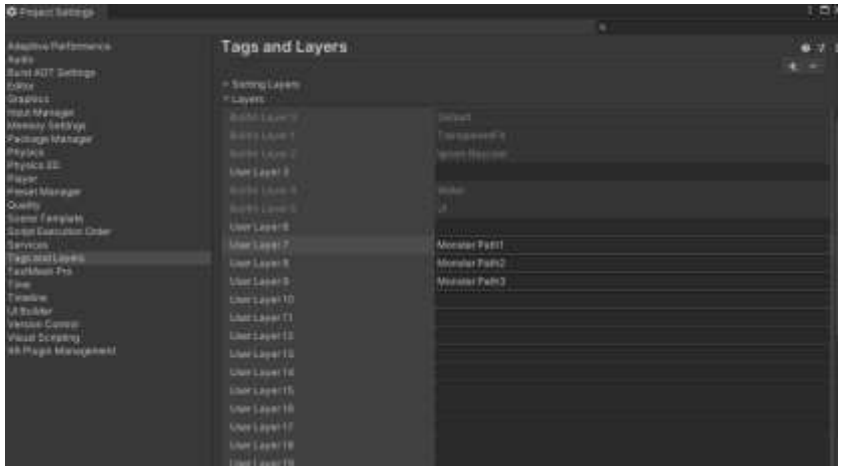
아직 3가지 경로로 몬스터 배치는 성공했으나 몬스터끼리 간의 충돌 처리는 해결하지 못함.

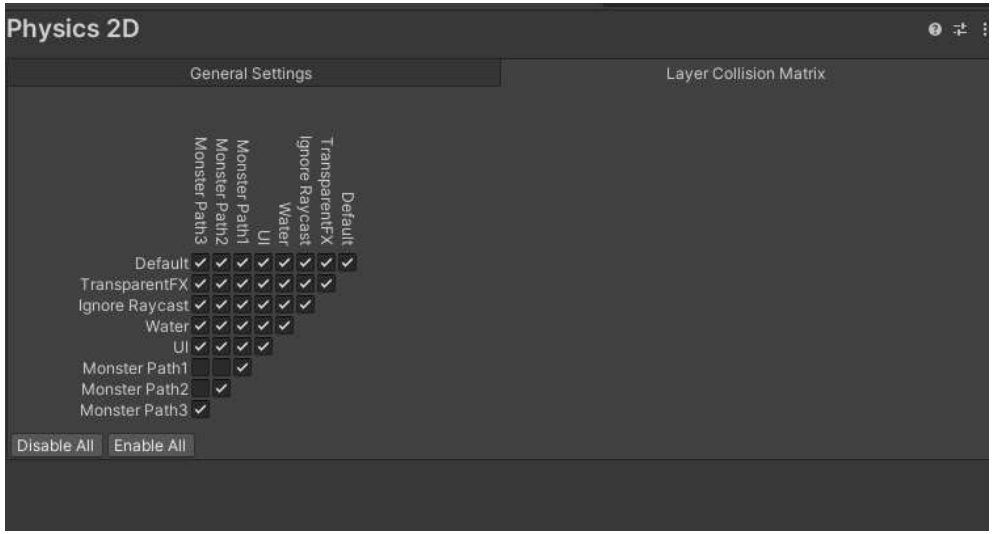


X



O





Project Setting->Layer 충돌처리를 이용하여 같은 경로의 몬스터 끼리 충돌처리를 하고 다른 경로의 몬스터는 건들지 않도록 설정!

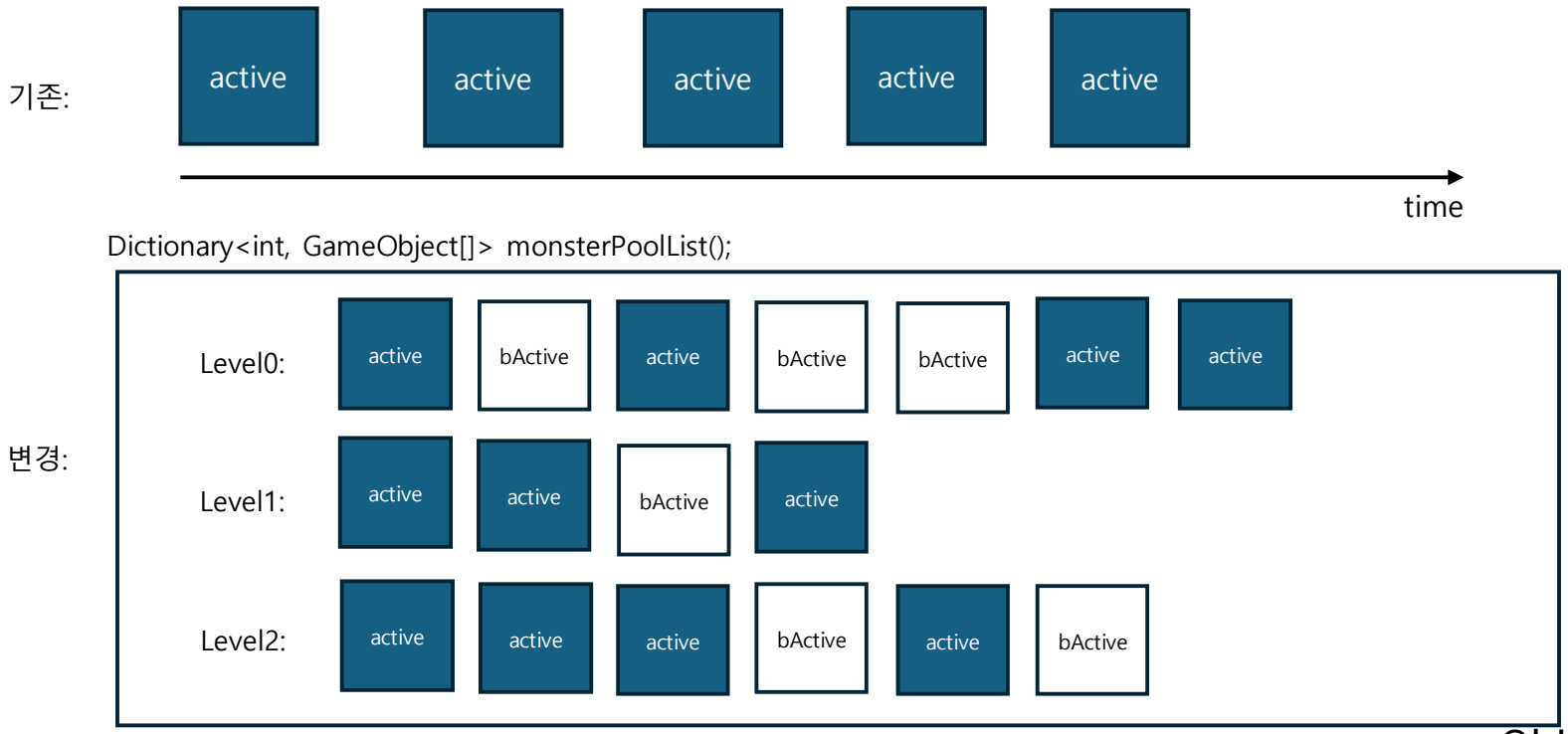
게임을 하면서 몬스터의 움직임을 보면 같은 경로의 몬스터만 점프하여 넘어가려고 하고 다른 경로에 있는 몬스터는 따로 충돌처리나 점프했을때 다른 경로의 몬스터 위로 올라가지 않는 것 같아서 이렇게 설정하였습니다

2025-03-06 / 2일차

2. 몬스터 스포너 및 움직임 구현

최적화 부분&리펙토링 구상도

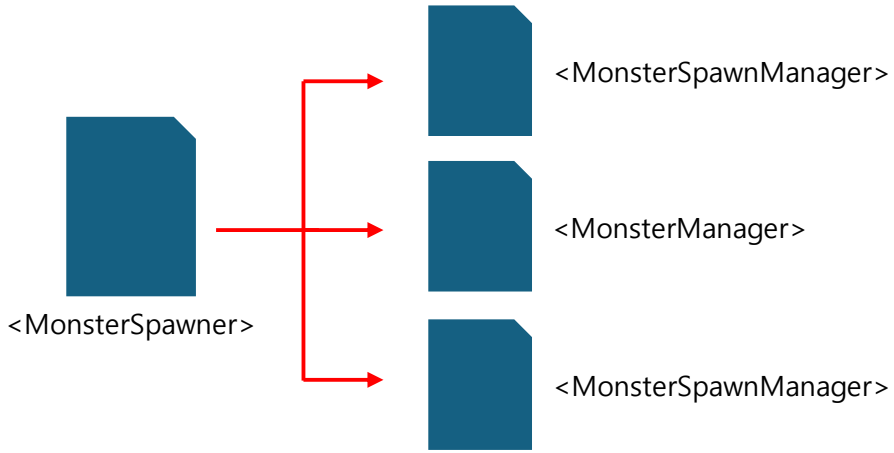
몬스터 소환에 있어서 **패키지 내 몬스터의 종류가 4가지나 있음을 확인하고 체계적으로 몬스터 오브젝트를 관리할 필요성**을 느꼈습니다..
또한 **객체를 생성(Instantiat) 삭제(Destory)로 관리하는 것이 아닌 활성화/비활성화로 관리하여 오브젝트 폴링 최적화**를 하려고 했습니다.
체계적으로 관리하기 위한 자료구조는 Dictionary를 이용했으며 레벨 별 몬스터를 만들어 사용하려 했습니다.



2025-03-06 / 2일차

2. 몬스터 스포너 및 움직임 구현

최적화 부분&리펙토링 구상도



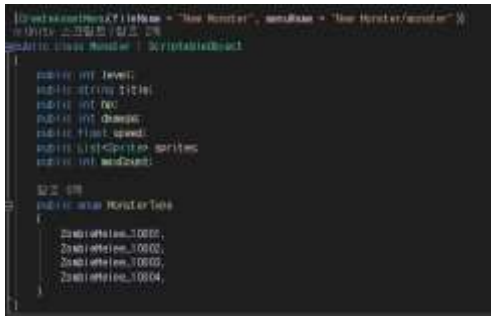
래퍼런스:
https://codeposting.tistory.com/entry/Unity-%EC%9C%A0%EB%8B%88%ED%8B%B0-%EC%8B%B1%EA%B8%80%ED%86%A4-%ED%99%9C%EC%9A%A9-singleton-DontDestroyOnLoad

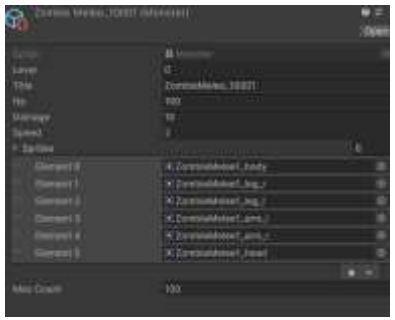
https://github.com/UihwanLee/2023_Progress_at_Play_digital_art_competition/blob/main/Partner/Assets/Scripts/Message.cs

우선 싱글톤 디자인패턴을 이용해
MonsterSpawner에 기능한 스폰, 경로배치 등을
Manager 이름을 부여하여 분리하였습니다.

한 스크립트(클래스) 내 수행하는 기능은
최대한 그 기능만 수행하는 역할만 할 수 있도록
생각하여 이렇게 분리하였습니다.

몬스터 정보를 담은 리소스 새로 생성





또한, 유니티 패키지 내에는 4가지 몬스터 밖에 없었지만
이후 몬스터가 늘어날 것을 고려하여 몬스터 정보를 담은
리소스를 ScriptableObject을 상속한 스크립트를 이용하여
생성하였습니다.

몬스터 정보 내에는 몬스터의 레벨, 이름, 체력, 데미지
이미지 리소스 등 많은 정보를 담았습니다.

2025-03-06 / 2일차

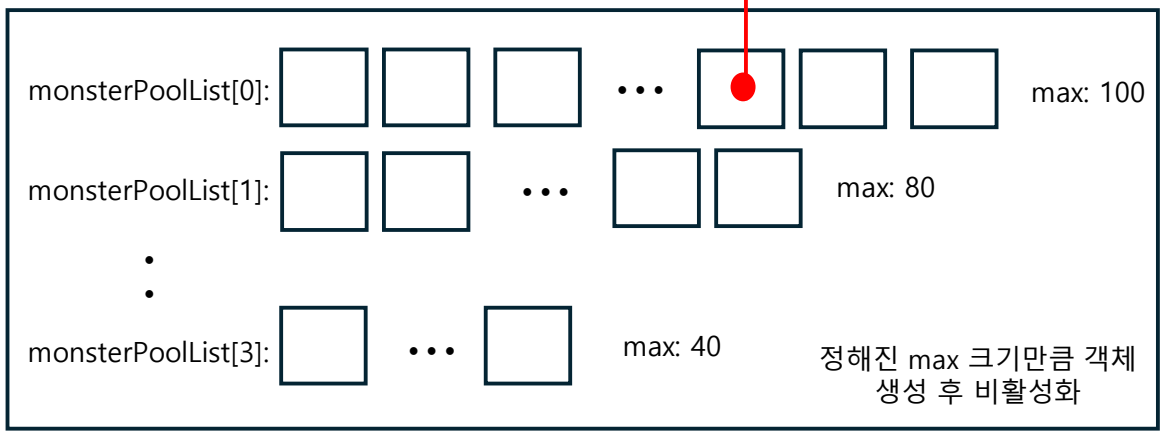
2. 몬스터 스포너 및 움직임 구현

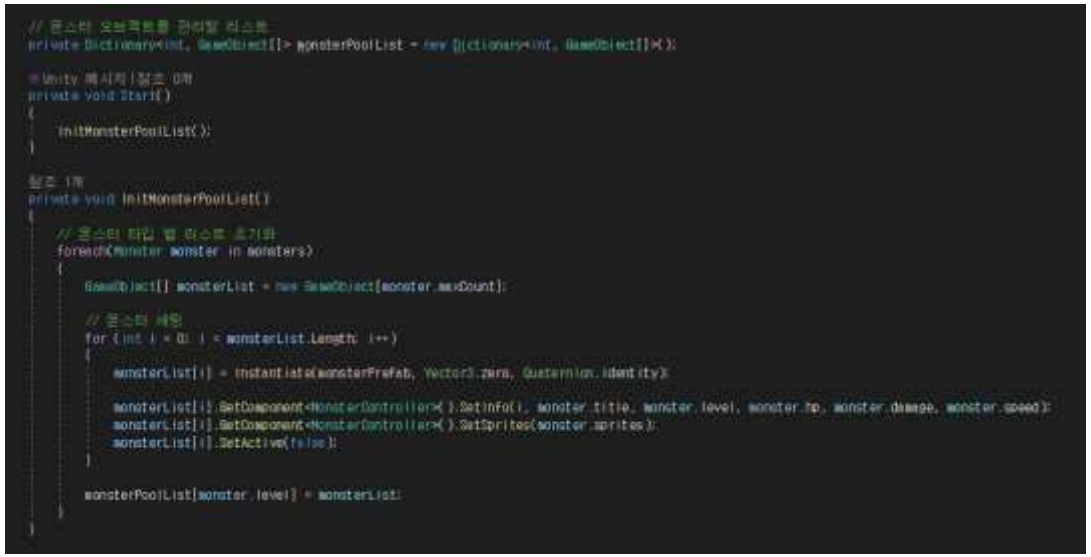
최적화 부분&리펙토링 구상도

Monster1: 최대 소환 수 100, level:0
Monster2: 최대 소환 수 80, level:1
..

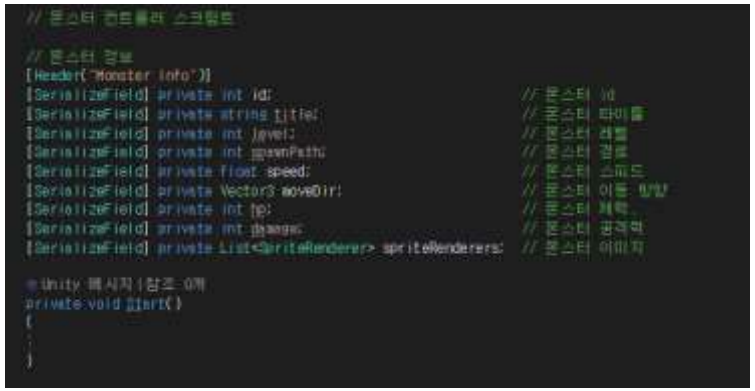
<MonsterSpawnManager>

리소스 내
담긴 정보
그대로 담는다





MonsterSpanwnManager

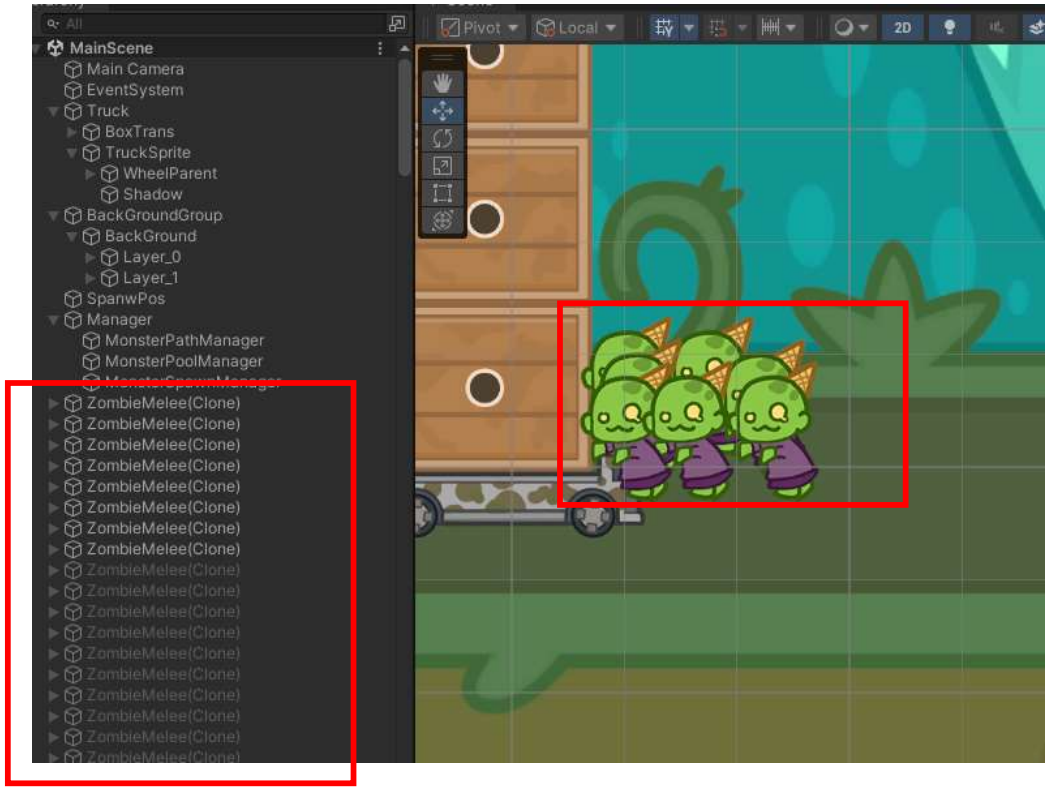


MonsterController

2025-03-06 / 2일차

2. 몬스터 스포너 및 움직임 구현

최적화 부분&리펙토링 구상도



게임 시작 시 몬스터 객체를 최대 지정 개수만큼 생성하고 시작

```
// 몬스터에게 경로 전달
MonsterController monsterController = monster.GetComponent<MonsterController>( );
monsterController.SetPath(path, myCollider, collider1, collider2);
monsterController.AddOrderLayer((4-path)*10);
```

MonsterPathManager

```
참조 1개
public void AddOrderLayer(int amount)
{
    foreach(SpriteRenderer render in spriteRenderers)
    {
        render.sortingOrder += amount;
    }
}
```

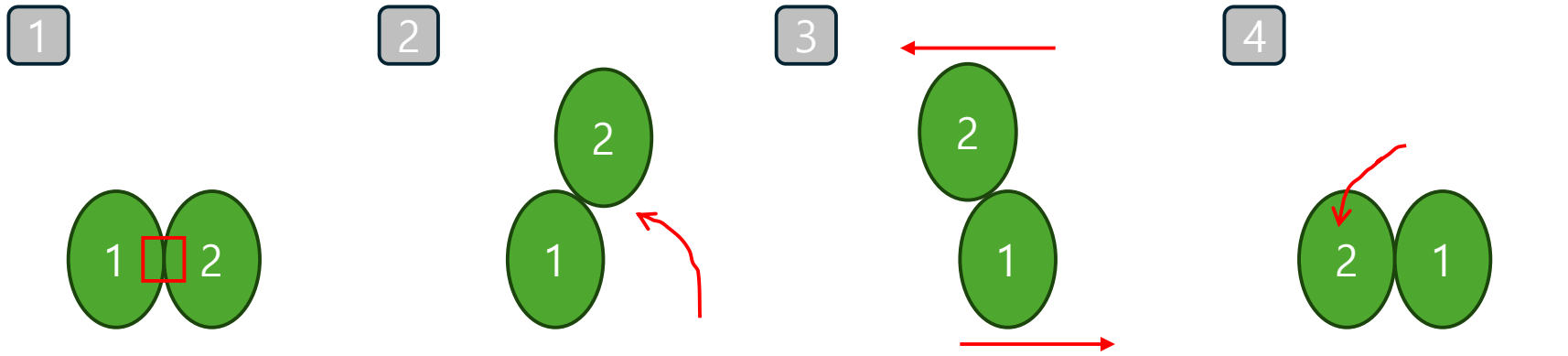
MonsterController

몬스터 소환 시 몬스터 오브젝트의 이미지 레이아웃 오더를 경로(0, 1, 2)에 따라 추가하여 경로 3가지에 맞는 오더 레이아웃 배치
(기존 이미지에서 머리가 가장 나와있어야해서 5로 설정함 만큼 더해서 이를 해결하려고 했습니다.)

2025-03-06 / 2일차

4. 몬스터 움직임 알고리즘 구현

몬스터들이 행동하는 구조가 어떻게 될까?



1. 충돌발생 2. 점프 3. 앞으로 가면서 밀려짐 4. 미끄러지면서 앞뒤가 바뀜

3가지 경로의 각각 독립적인 공간에서 몬스터들이 충돌할때 뒤에 있는 몬스터는 올라가서 앞으로 미끄려지는 형태이고 기존에 앞에 있던 몬스터는 위에 몬스터에 밀려나 뒤에 배치되는 구조로 파악했습니다.

이 구조를 만들기 위하여 여러가지 시뮬레이션을 진행하였고 여러가지 방법도 고안해보았습니다.

2025-03-07 / 3일차

4. 몬스터 움직임 알고리즘 구현

순환구조를 만들기 위한 정말 많은 여러가지 시도 : 충돌처리

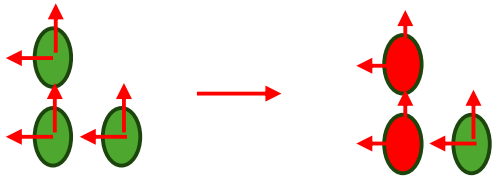
1 단순하게 OnCollisionStay2D를 이용한 충돌처리

-장점: 유니티에서 기본적으로 제공하는 충돌처리 함수로 인해 사용하기 편함
-단점: 충돌되는 객체에 대한 세세한 처리가 불가능하여 MonsterController 내 움직임 함수를 조정해야함

2 Physcis2D.OverlapCircle 함수를 이용 path Layer만 감지한 충돌처리

-장점: 정해진 원 안에 원하는 객체(상하좌우)만 뽑아내어 충돌처리 가능
-단점: 원하는 객체를 얻기 위한 조건문에서 코드가 지저분해질 수 있음

3 Physcis2D.RayCast 함수를 이용 path Layer만 감지한 충돌처리



-장점: Ray의 directio만 조정하여 앞과 위 객체만 선별 가능
-단점: 독립적인 객체에서 각각 2가지의 RayCast 호출로 인한 계산이 많아짐.

4. 몬스터 움직임 알고리즘 구현

순환구조를 만들기 위한 정말 많은 여러가지 시도 : 충돌처리

2 Physcis2D.OverlapCircle 함수를 이용 path Layer만 감지한 충돌처리

```
참조 0개
private void CheckAllCollision()
{
    Collider2D[] hitColliders = Physics2D.OverlapCircleAll(new Vector2(transform.position.x, transform.position.y), 0.5f, LayerMask.GetMask(layer));

    foreach(Collider2D collider in hitColliders)
    {
        if(collider.gameObject.layer != LayerMask.NameToLayer(layer))
        {
            float otherMonsterX = collider.transform.position.x;
            float otherMonsterY = collider.transform.position.y;
            float myX = transform.position.x;
            float myY = transform.position.y;

            if (otherMonsterY - myY > 1.0f) { }
            else if (myX - otherMonsterX > 0.5f) { }
            else return;
        }
    }
}
```

Physics2D 래퍼런스
https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Physics2D.html

3 Physcis2D.RayCast 함수를 이용 path Layer만 감지한 충돌처리

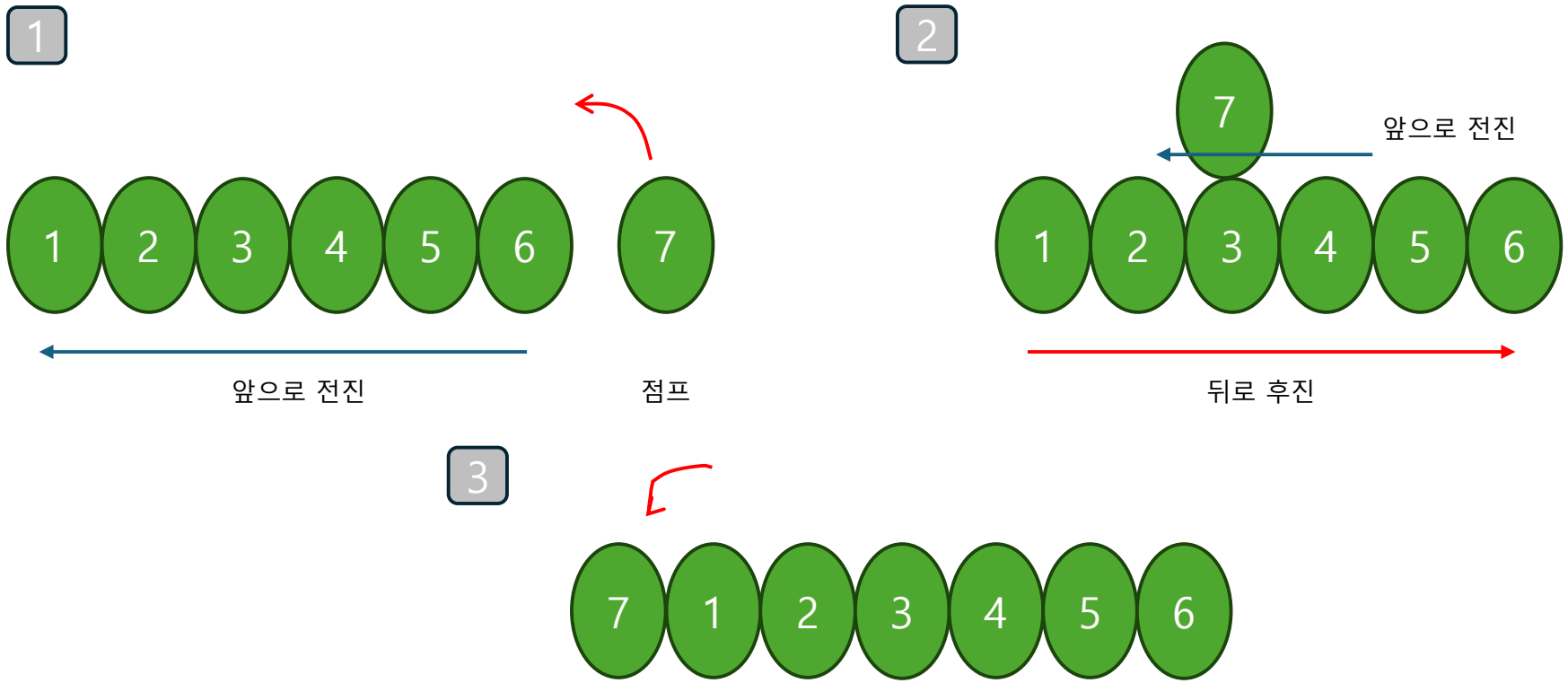
```
참조 0개
private void CheckAllCollision()
{
    Vector3 startPosition = new Vector3(transform.position.x, transform.position.y + 0.5f, transform.position.z);
    Debug.DrawRay(startPosition, new Vector3(-0.5f, 0.0f, 0.0f), new Color(1, 0, 0));

    CheckRayCastHit(startPosition, Vector3.left, 0.5f, LayerMask.GetMask(layer), 0);
    CheckRayCastHit(startPosition, Vector3.up, 0.5f, LayerMask.GetMask(layer), 1);
}


참조 2개
private void CheckRayCastHit(Vector3 _startPosition, Vector3 _direction, float _distance, int _layMask, int _condition)
{
    RaycastHit2D[] hitList = Physics2D.RaycastAll(_startPosition, _direction, _distance);

    foreach (RaycastHit2D hit in hitList)
    {
        if (hit.collider.gameObject != this.gameObject &&
            hit.collider.gameObject.layer == LayerMask.NameToLayer(layer))
        {
            if (_condition == 0 && isSliding == false)
            {
                isClimbing = true;
            }
            else if (_condition == 1)
            {
                isSliding = true;
            }
        }
    }
}
```

Physcis2D를 이용한 코드 모두 새로운 변수와 계산양이 많아져 코드가 난잡해서 두 방법 모두 포기했습니다.

2025-03-07 / 3일차

4. 몬스터 움직임 알고리즘 구현

순환구조를 만들기 위한 정말 많은 여러가지 시도 : 충돌처리



결국 핵심은 몬스터 위에 몬스터가 이동할 경우 아래에 있는 몬스터는 밀리는 형태에서
위 맨 앞에 있는 몬스터가 1층 맨 앞으로 내려오는 구조를 최대한 구현해보려고 하였습니다.

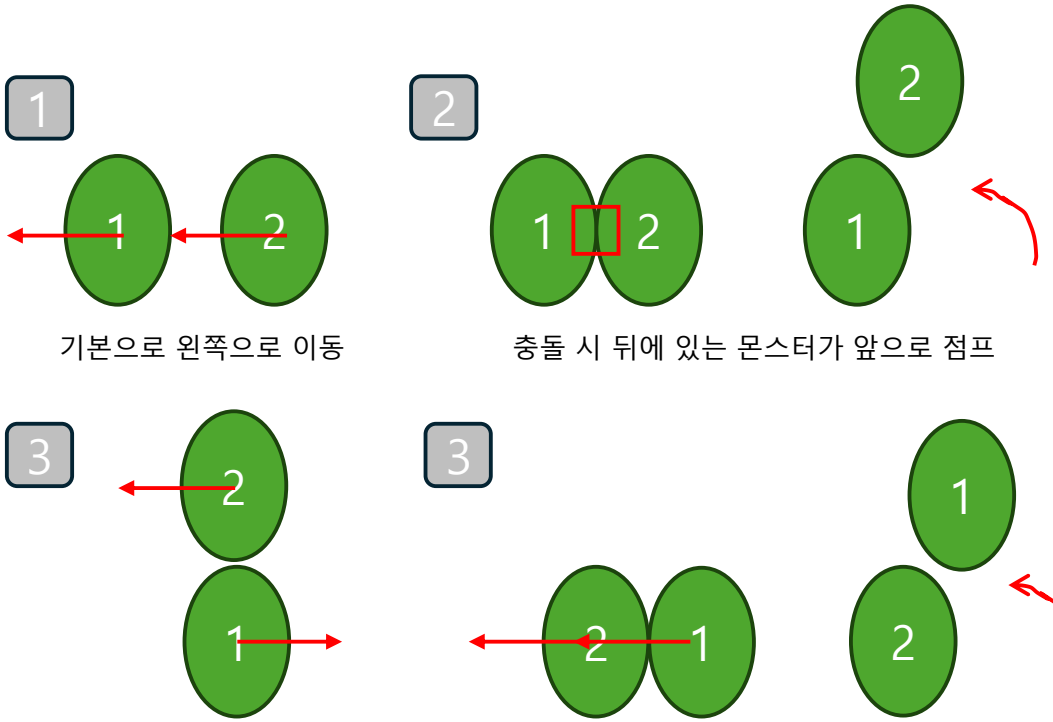
2025-03-07 / 3일차

4. 몬스터 움직임 알고리즘 구현

순환구조를 만들기 위한 정말 많은 여러가지 시도 : 충돌처리

```
참조 1개
void Moving()
{
    transform.position += moveDir * speed * Time.deltaTime;
}

// 미끄러지는 동작
참조 1개
void Sliding()
{
    rb.velocity = new Vector2(3.0f, rb.velocity.y);
}

// 올라가는 동작
참조 1개
void ClimbUp()
{
    StartCoroutine(ClimbUpCoroutine());
}

참조 1개
private IEnumerator ClimbUpCoroutine()
{
    isJumping = true;

    transform.position += moveDir * speed * Time.deltaTime;

    yield return new WaitForSeconds(0.5f); // 점프 유지 시간

    rb.velocity = new Vector2(rb.velocity.x, jumpForce);

    yield return new WaitForSeconds(0.5f); // 점프 유지 시간

    isClimbing = false;

    yield return new WaitForSeconds(3.0f); // 쿨타임

    isJumping = false; // 점프가 끝나면 다시 점프 가능
}
```

```
private void OnCollisionStay2D(Collision2D collision)
{
    if (collision.gameObject.layer == LayerMask.NameToLayer(layer))
    {
        int hitCondition = GetHitPosition(collision);

        switch (hitCondition)
        {
            case 0:
                isSliding = true;
                isClimbing = false;
                break;
            case 1:
                if (isJumping == false && isSliding == false)
                {
                    isClimbing = true;
                }
                isSliding = false;
                break;
            case -1:
                break;
            default:
                isSliding = false;
                isClimbing = false;
                break;
        }
    }
}

Unity 메시지 | 참조 0개
private void OnCollisionExit2D(Collision2D collision)
{
    if (collision.gameObject.layer == LayerMask.NameToLayer(layer))
    {
        if(collision.transform.position.y > transform.position.y)
        {
            isSliding = false;
        }
    }
}
```



기본으로 왼쪽으로 이동

충돌 시 뒤에 있는 몬스터가 앞으로 점프

2번 몬스터와 충돌되고 있을 시 1번 몬스터는 계속 밀려나가고 2번 몬스터는 기본 왼쪽 이동

2번 몬스터는 아래로 떨어지고 2 몬스터다 왼쪽 이동 이후 1번이 점프하면서 반복

2025-03-07 / 3일차

4. 몬스터 움직임 알고리즘 구현

순환구조 구현 후 아직 못다한 부분, 고쳐야할점

리펙토링

- **Hero**와 **Monster** 둘다 쓰려고 했던 **BehaviorTree** 스크립트 적용을 못함
- 충돌처리의 움직임 제어에 있어활용한 변수들에 대한 정리와 리펙토링 필요(하드 코딩함)

순환구조 움직임

- 1층에 있는 몬스터들이 더 밀려나야함 (위쪽에 있는 몬스터가 쌓일 수록 더 밀려나감)
- 점프는 맨 끝쪽에 있는 몬스터만 함

Behavior 래퍼런스: https://dodobug.tistory.com/17

```
참조 0개
private void InitBehaviorTree()
{
    root = new Selector();
    Sequence actionSequence = new Sequence();
    Sequence jumpSequence = new Sequence();
    Sequence attackSequence = new Sequence();
    Sequence chaseSequence = new Sequence();

    Condition isPlayerAround = new Condition(IsPlayerAround);
    Condition isMonsterAround = new Condition(IsMonsterAround);

    Action jumpAction = new Action(TryJumping);
    Action attackAction = new Action(Attack);
    Action chaseAction = new Action(Chase);

    root.AddChild(actionSequence);
    root.AddChild(chaseSequence);

    actionSequence.AddChild(jumpSequence);
    actionSequence.AddChild(attackSequence);
    chaseSequence.AddChild(chaseAction);

    jumpSequence.AddChild(isMonsterAround);
    jumpSequence.AddChild(jumpAction);
    attackSequence.AddChild(isPlayerAround);
    attackSequence.AddChild(attackAction);

    root.Run();
}
```

쓰려고 했던 **BehaviorTree**

2025-03-07 / 3일차

5. 추가구현

추가 구현할 기능들을 정리

1 : 몬스터들의 공격 기능

- 몬스터 공격 애니메이션 적용
- 몬스터 공격에 대해 박스/히어로 데미지 적용 및 여러 상호작용
- 몬스터 쌓임에 따른 트럭/배경 속도 감소

2 : 히어로 공격 기능

- 히어로의 포물선 공격 (인 게임 최대한 비슷하게 구현)
- 히어로 공격에 따른 몬스터 죽음

3 : 레벨 디자인 적용 (X)

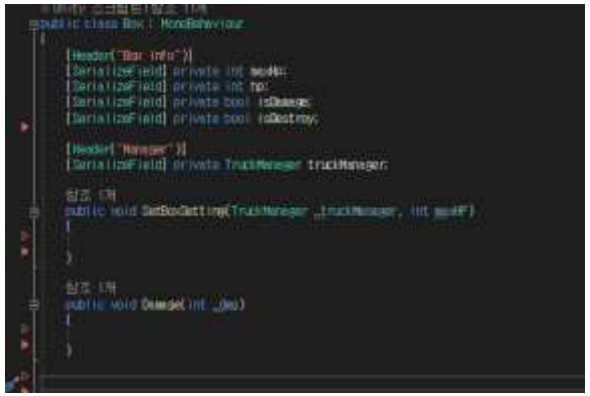
- 시간이 지남에 따라 나오는 몬스터 레벨 업

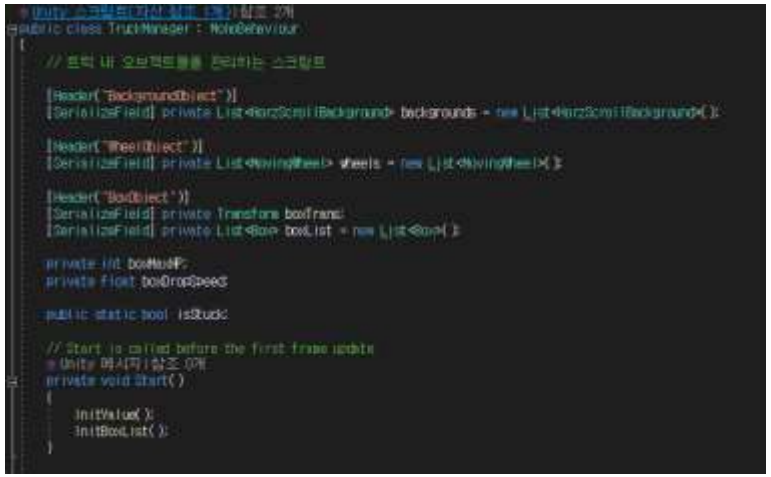
2025-03-08 / 4일차

5. 추가구현 : 몬스터 공격

몬스터와 상호작용할 Box와 Box를 관리할 BoxList

몬스터의 공격 기능을 만들기 앞서 몬스터 공격에 따라 무너져가는 박스 관리가 필요하기에 **Box** 스크립트와 그 **Box** 스크립트를 관리하는 스크립트가 필요했습니다.



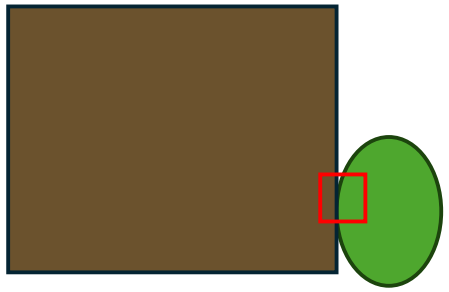


또한 공격 애니메이션을 적용하는 도중 에러가 발생하여 이부분도 래퍼런스를 참고하여 해결했습니다.

애니메이션 오류 제거 래퍼런스: https://pinaeong.tistory.com/45

2025-03-08 / 4일차

5. 추가구현 : 몬스터 공격



1. 충돌발생

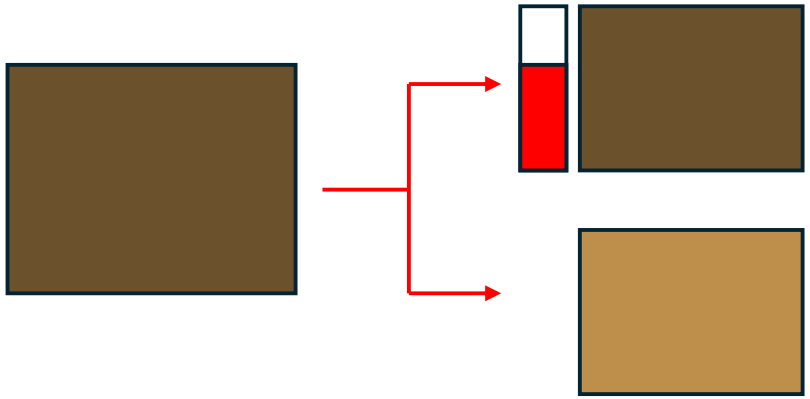
```
private void TryAttack(GameObject _boxObj)
{
    Box box = _boxObj.GetComponent<Box>();

    // 공격이 가능할 시 공격
    if(!isAttacking && box.isDestroy() == false)
    {
        Attack(box);
    }
}
```

2. 충돌한 박스에 데미지 적용

```
public void Damage(int _dmg)
{
    // hp가 0 이하 시 파괴
    if(hp - _dmg <= 0)
    {
        hp = 0;
        Destroy();
        return;
    }

    // 공격 받는 모션
    hpPannel.SetActive(true);

    // 데미지 적용
    hp -= _dmg;
    StartCoroutine(DamageEffect());

    // Hp 슬라이더 업데이트
    UpdateHpSlider();
}
```

```
private void Attack(Box _box)
{
    StartCoroutine(AttackCoroutine(_box));
}

private IEnumerator AttackCoroutine(Box _box)
{
    isAttacking = true;

    // 데미지 적용
    _box.Damage(damage);
    _box.isDamage = true;

    // 공격 애니메이션 실행
    anim.SetBool("isAttacking", isAttacking);

    yield return new WaitForSeconds(0.5f);

    isAttacking = false;
    _box.isDamage = false;
    anim.SetBool("isAttacking", isAttacking);
}
```

3. 몬스터는 공격 모션을 박스는 타격 타격이펙트 적용



타격 이펙트1: Hp바 활성화

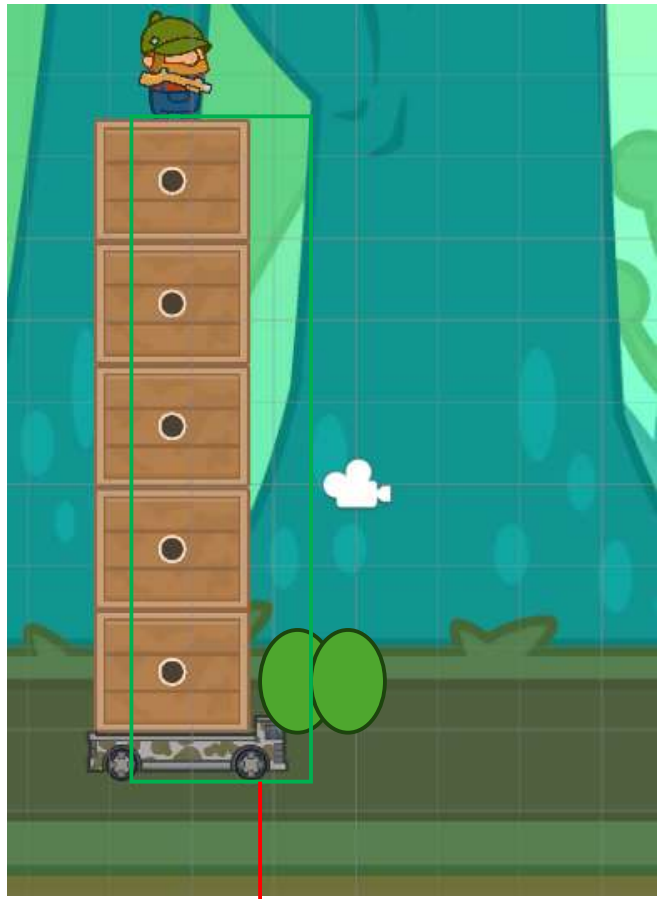
타격 이펙트2: 색상 변화

```
private void UpdateHpSlider()
{
    if (hpBar != null)
    {
        Slider hpSlider = hpBar.GetComponentInChildren<Slider>();
        hpSlider.value = (float)hp / maxHp; // HP 값을 0~1 범위로 변환
    }
}
```

```
private IEnumerator DamageEffect()
{
    if(boxSpriteRender != null)
    {
        Color damageColor;
        if (ColorUtility.TryParseHtmlString("#FF6B6B", out damageColor))
        {
            boxSpriteRender.color = damageColor;
            yield return new WaitForSeconds(0.1f);
            boxSpriteRender.color = Color.white;
        }
    }
}
```

2025-03-08 / 4일차

5. 추가구현 : 몬스터 공격



IsTrigger 켜져 있는 새로운 BoxCollision

또한 몬스터가 앞에 쌓일 수록 기존 2D 배경과 트럭의 바퀴 속도를 감속시켰습니다.

```
Unity 메시지 | 참조 0개
private void OnTriggerStay2D(Collider2D collision)
{
    if (collision.gameObject.tag == TagData.TAG_MONSTER)
    {
        // 몬스터와 충돌 시 배경을 서서히 멈춘다
        foreach (MovingWheel wheel in wheels)
        {
            wheel.isStuck = true;
        }

        foreach (HorzScrollBackground bg in backgrounds)
        {
            bg.isStuck = true;
        }
    }
}

Unity 메시지 | 참조 0개
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject.tag == TagData.TAG_MONSTER)
    {
        // 몬스터와 충돌 시 배경을 서서히 멈춘다
        foreach (MovingWheel wheel in wheels)
        {
            wheel.isStuck = false;
        }

        foreach (HorzScrollBackground bg in backgrounds)
        {
            bg.isStuck = false;
        }
    }
}
```

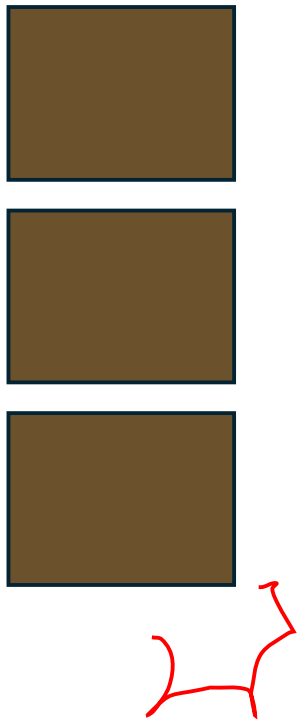
```
참조 1개
private void CheckStuck()
{
    if (isStuck)
    {
        // 공격받고 있다면 서서히 감속
        moveSpeed = Mathf.Lerp(moveSpeed, 0.0f, moveRate * Time.deltaTime);
    }
    else
    {
        // 공격받지 않는다면 서서히 가속
        moveSpeed = Mathf.Lerp(moveSpeed, maxMoveSpeed, moveRate * Time.deltaTime);
    }
}
```

2025-03-08 / 4일차

5. 추가구현 : 몬스터 공격

밑에 있는 박스가 파괴당하면 destroy하고 위에 있는 박스들은 차례대로 내려오게 설정하였습니다.

1



2

```
참조 1개
private void Destroy()
{
    if(truckManager == null)
    {
        Debug.LogError("There is no truckManager");
        return;
    }

    isDestroy = true;

    truckManager.OnBoxDestroyed(this);

    Destroy(gameObject);
}
```

Box의 Destory 함수에서 TruckManager에게 Box가 파괴됨을 알림. TruckManager는 위에 있는 box들에게 순차적으로 내려오라고 수행

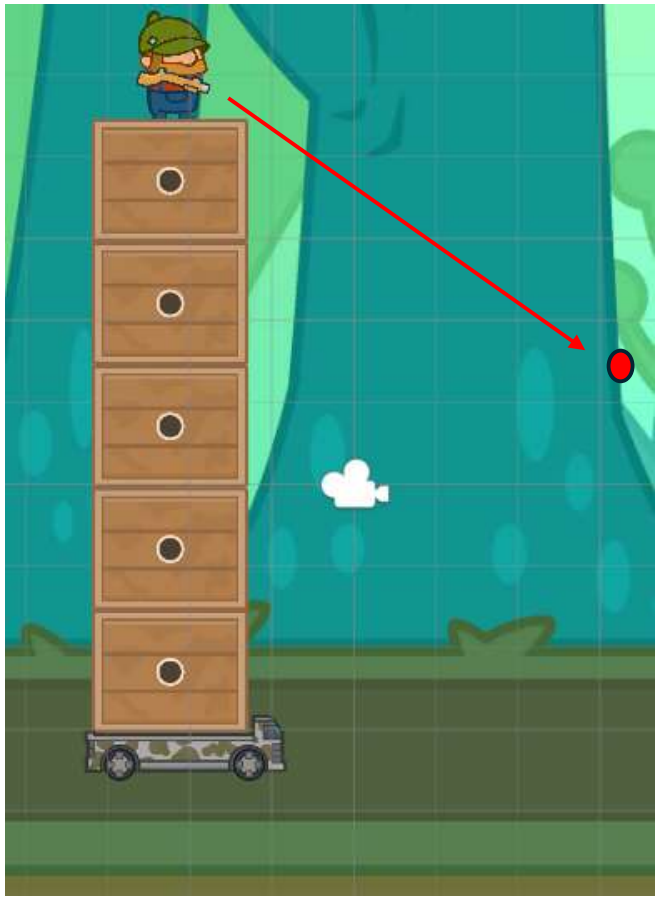
3

```
참조 1개
public void Drop(Vector3 targetPosition, float dropSpeed)
{
    StartCoroutine(DropCoroutine(targetPosition, dropSpeed));
}

참조 1개
IEnumerator DropCoroutine(Vector3 targetPosition, float dropSpeed)
{
    this.transform.position = Vector3.Lerp(this.transform.position, targetPosition, dropSpeed * Time.deltaTime);
    yield return new WaitForSeconds(0.1f);
    this.gameObject.transform.position = targetPosition; // 최종 위치 보정
}
```

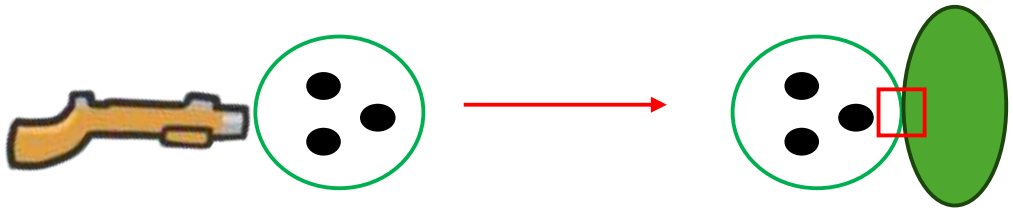
명령 받은 box는 drop 코루틴을 실행하며 동적으로 내려오기 수행

2025-03-08 / 4일차

5. 추가구현 : 히어로 공격 기능



히어로 공격은 마우스 클릭 이벤트에 대한 정보를 받아와서 하위 오브젝트인 Gun이 그 방향대로 향하게 했습니다.



또 Gun은 Bullet을 특정시간 지속 생성하고 그 Bullet은 CircleCollider를 가지고 있어 몬스터와 충돌 시 몬스터 또한 히어로의 데미지 만큼 데미지 받게 하였습니다.

2025-03-08 / 4일차

5. 추가구현 : 히어로 공격 기능

```
참조 1개
IEnumerator FireBulletsCoroutine()
{
    isFiring = true;

    yield return new WaitForSeconds(1.0f);

    // 총알 발사
    Fire();

    // 발사 간격만큼 대기
    yield return new WaitForSeconds(fireRate);

    isFiring = false;
}

참조 1개
private void Fire()
{
    GameObject bullet = GetBulletFromPool();

    if(bullet == null) { Debug.Log("총알이 없습니다!"); return; }

    if (bullet != null)
    {
        bullet.SetActive(true);

        // 위치 조정
        bullet.transform.position = gunMuzzle.position;
        bullet.transform.rotation = transform.rotation;

        // 발사
        Bullet bulletScript = bullet.GetComponent<Bullet>();
        bulletScript.Fire(gunMuzzle.right);
    }
}
```

GunController에서 발사할 수 있는 체크하고 발사

```
참조 1개
public void Fire(Vector2 direction)
{
    Vector2 moveDirection = direction.normalized;
    rb.velocity = moveDirection * bulletSpeed;
}
```

```
private void Update()
{
    if(CheckOutOfScreen())
    {
        // 화면 밖으로 나가면 비활성화
        gunController.ReturnBulletToPool(gameObject);
    }
}

참조 1개
bool CheckOutOfScreen()
{
    // 화면 밖으로 나갔는지 체크
    Vector3 position = transform.position;

    if (position.y > topBoundary || position.y < bottomBoundary || position.x > rightBoundary || position.x < leftBoundary)
    {
        return true;
    }

    return false;
}
```

Bullet은 발사 후 밖으로 나갔는지 체크하고 화면 밖으로 나가면 BulletPool에서 비활성화후 이후 재활용할 수 있도록 했습니다.

```
Unity 메시지 | 참조 0개
private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.gameObject.tag == TagData.TAG_BULLET)
    {
        // 총알 맞을 시 데미지를 입는다.
        Damage();
    }
}
```

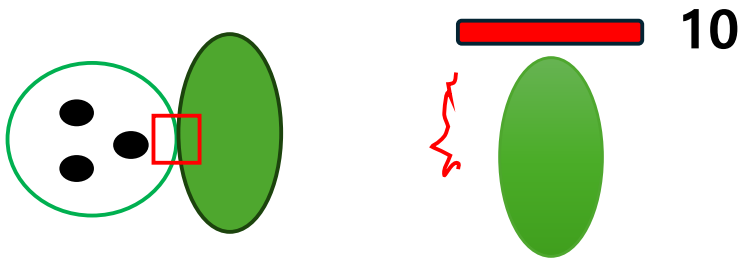
```
참조 1개
private void Dead()
{
    isDead = true;

    StartCoroutine(DeadCoroutine());
}

참조 1개
private IEnumerator DeadCoroutine()
{
    // 죽는 애니메이션 실행
    anim.SetBool("IsIdle", false);
    anim.SetBool("IsDead", isDead);

    yield return new WaitForSeconds(2.0f);

    // PoolManager에게 죽었다고 알림
    MonsterPoolManager.Instance.ReturnMonsterToPool(this.level, this.id);
}
```

몬스터는 Trigger를 통해 한번만 맞았는지 체크하고 hp가 0이하면 죽게(비활성화) 처리하였습니다.

2025-03-08 / 4일차

5. 추가구현 : 히어로 공격 기능

좀비 타격 이펙트 : 데미지 표시




```
참조 1개
IEnumerator DamageShow(int _dmg)
{
    ui_Damage.SetActive(true);
    ui_Damage_Text.text = _dmg.ToString();
    yield return StartCoroutine(FadeText(ui_Damage_Text, 0, 1, 0.5f));
    yield return StartCoroutine(FadeText(ui_Damage_Text, 1, 0, 0.5f));
    ui_Damage.SetActive(false);
}

참조 2개
private IEnumerator FadeText(TextMeshProUGUI text, float startAlpha, float endAlpha, float duration)
{
    float elapsedTime = 0f;
    while (elapsedTime < duration)
    {
        elapsedTime += Time.deltaTime;
        float alpha = Mathf.Lerp(startAlpha, endAlpha, elapsedTime / duration);
        textColor.a = alpha;
        text.color = textColor;
        yield return null;
    }
    textColor.a = endAlpha;
    text.color = textColor;
}
```

```
참조 1개
private void UpdateHpSlider()
{
    if (hpBar != null)
    {
        Slider hpSlider = hpBar.GetComponentInChildren<Slider>();
        hpSlider.value = (float)hp / maxHp; // HP 값을 0~1 범위로 변환
    }
}
참조 1개
```

Bullet 맞은 이펙트
1. Fade 코루틴을 통한 데미지 표기
2. 남은 hp바로 표시하였습니다.
3. 색상 변화

로 구현하였습니다.

2025-03-08 / 4일차

6. 정리

과제 후기 및 소감

회사에 합격하는 목적으로 과제를 수행한적은 처음이라 **과제를 하는 과정을 어떻게 전달해드릴지 고민과정**이 많았습니다..
이전 펨트론 현장실습에서 **실무경험을 배우면서 팀장님이 저한테 해준 조언으로는 ppt나 노트로 저의 개발 과정을 적으면서 무엇보다 디자인 신경쓰지 말고 어려웠던 점이나 코드를 작성하는 도중 래퍼런스를 포함하여 있는 그대로 보여달라**라고 하신적이 있습니다.

따라서 과제 수행하면서 어떤 기능을 구현할 때 저의 개발 구상가 그 과정에서 어려운 점이나 해결방법 등을 최대한 시각화하며 보여주고자 하였습니다.

제 생각 기준으로 본 과제였던 몬스터의 순환 구조 움직임 구현이 가장 어려웠으며 완벽히 구현하지 못하고 부족한 부분도 있다고 생각합니다. 부족한 부분이 무엇인지 회사 입장에서는 제대로 전달하는 것이 중요하기 때문에 최대한 제 코드 부분이 어떻게 구상되고 개발되었는지 입사 후에도 열심히 소통하는 개발자가 되겠습니다! 감사합니다!