팀 바운딩 박스

Project Cube

Game Project, Unity Project
2024.7 - 2024.8
https://mandlemandle.com/project/blocking/game

“큐브를 굴려서 적을 처치해나가는 퍼즐게임”이라는 컨셉트에서 시작된 프로젝트입니다.
만들래 10분 게임 공모전에 참여하기 위해 결성된 2인 팀이고 1개월이 남은 상황에서 시작하여 핵심 매커니즘을 하이퍼 캐주얼 장르에 맞게 제한하고 5개의 프로토타입 레벨로 구성하여 제출하는 것을 목표로 했습니다.
공모전 결과는 주최측과 유저 평가로 5점 만점에 4점 이상으로 만족스럽게 나왔지만 평가 수의 부족으로 160여개의 팀 중 33등으로 조금 아쉬운 결과로 마무리하게 되었습니다.

목표

1. **제한된 시간에 맞춰 게임을 기획하고 개발**
2. **학교 과제를 넘어 실제로 게임을 만들어 불특정 대상들에게 게임을 공개해보는 경험**

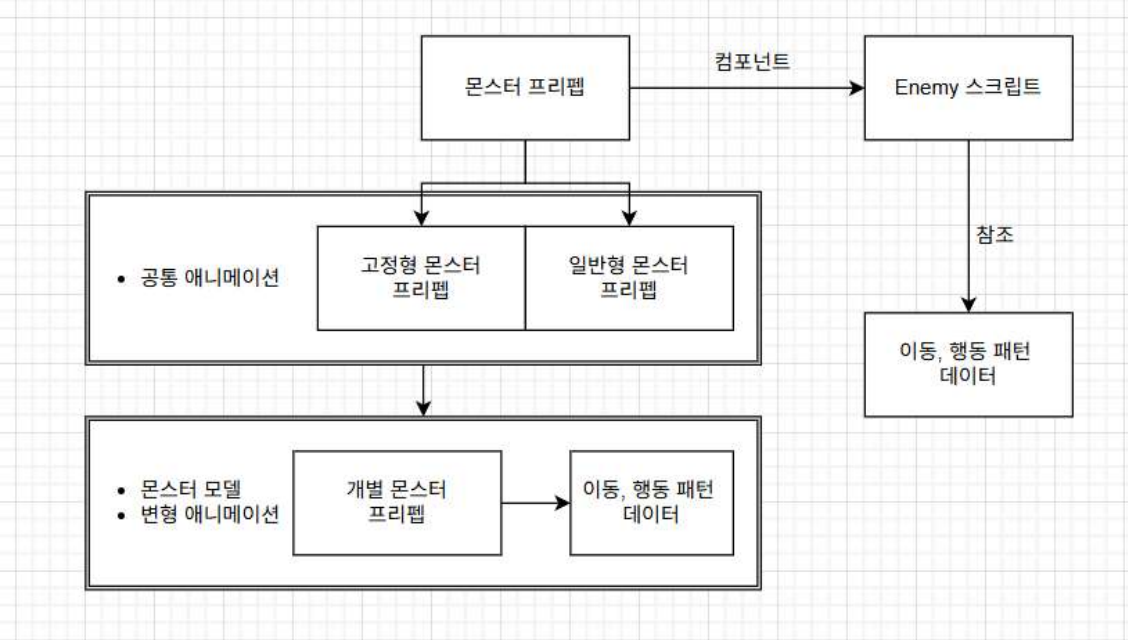
게임 플레이 사진

주요 기여

기획에도 참여하였지만 주로 **프로그래머의 역할**을 하였고 **몬스터와 핵심 게임 플레이 매커니즘**과 관련된 작업을 담당 하였습니다. 부가적으로 타이틀 씬과 크레딧, 히든 레벨과 보스전, 레벨 에디터 같은 부분도 작업하였습니다.
Unity 엔진, C#, 유니티 에셋 (Feel, Shapes, DoTween)이 사용되었습니다.

몬스터

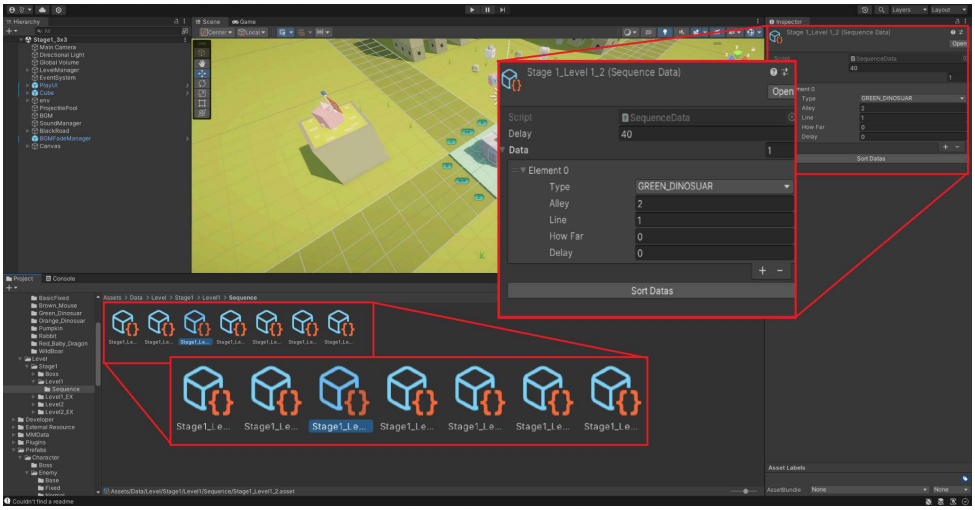
몬스터의 종류로는 **일반형, 플레이어의 공격에 넉백을 받지 않는 고정형, 보스** 3종류가 있습니다.
일반형과 고정형 모두 같은 Enemy 스크립트로 동작하지만 몬스터의 종류가 추가될 상황을 고려해 기존 프리펩을 상속하여 외형과 애니메이션을 다르게 설정할 수 있도록 구현하였고 **이동, 행동 패턴 데이터를 받아 다르게 동작**하게 구현되었습니다.



보스는 일반적인 몬스터와는 다르게 기믹을 수행할 수 있어야 하기 때문에 Enemy 스크립트가 아닌 **독립적인 스크립트를 따라 동작**합니다.
몬스터의 이동에 DoTween, 피격과 타격 및 각종 애니메이션을 위해 Feel, 이동 경로나 공격 범위 등의 시각적인 표현을 위해 Shapes가 사용되었습니다.

시퀀스 시스템

턴 단위로 여러 시퀀스를 시간차에 따라 실행시켜 몬스터를 생성하는 핵심 게임 진행을 담당하는 클래스입니다. **하나의 레벨에 여러개의 시퀀스**가 들어갈 수 있으며 **시퀀스는 몬스터의 등장에 대한 데이터** 입니다.



각각의 시퀀스는 동시에 관리되지만 딜레이나 이전 시퀀스의 종료 상태에 따라 실행되기 시작하면 독립적으로 몬스터를 생성해야 했기에 **SequenceManager에서는 각 시퀀스의 딜레이와 이전 시퀀스의 종료 상태를 체크**하며 **SequencePlayer를 시퀀스가 시작되어야 할 시점에 턴 이벤트를 구독**시켜 **각각의 시퀀스가 독립적으로 동작**하게 구현하였습니다. SequencePlayer는 이벤트 구독, 해지, 턴마다 실행될 함수로 구성되어 있고 **SequencePlayer가 종료 시점에 스스로 이벤트를 해지**하게 구현하였습니다.

```
class SequencePlayer
{
    2 references
    private int _currTurn = 0;
    5 references
    private int _currData = 0;
    5 references
    private SequenceData _sequence;
    1 reference
    public bool IsSequenceEnd => _currData >= _sequence.Data.Count;
    1 reference
    public SequencePlayer(SequenceData seq)
    {
        _sequence = seq;
    }
    0 references
    public void Register()
    {
        LevelManager.Instance.Subscribe(LEVEL_EVENT_TYPE.PLAYER_STARTMOVE, Play);
    }
    1 reference
    public void Unregister()
    {
        LevelManager.Instance.UnSubscribe(LEVEL_EVENT_TYPE.PLAYER_STARTMOVE, Play);
    }
    2 references
    public void Play()
    {
        int size = _sequence.Data.Count;
        for (; _currData < size; ++_currData)
        {
            if (_currTurn >= _sequence.Data[_currData].Delay)
                EnemyManager.Instance.SpawnEnemy(_sequence.Data[_currData]);
            else break;
        }
        _currTurn += 1;
        if (IsSequenceEnd == true)
            Unregister();
    }
}
```

오브젝트 풀 & 매니저

프로젝트 진행 중, 오브젝트 생성과 소멸의 퍼포먼스 문제는 직접적으로 확인되진 않았지만, **다양한 오브젝트들이 자주 생성, 소멸되는 구조**였기 때문에 향후 퍼포먼스 이슈에 대비하기 위해 오브젝트 풀링 시스템을 설계하고 구현했습니다.

ObjectPoolBase는 공통적으로 필요한 초기 사이즈, 확장될 크기 값, 소환할 프리펩 같은 데이터와 오브젝트 지급, 반환의 기능을 할 가상 함수로 구성 되어 있습니다.

```
public ObjectPoolBase(int InitialPoolSize, int sizeOfExpansion, GameObject prefab)
{
    info = new PoolInfo();
    info.InitialPoolSize = InitialPoolSize;
    info.SizeOfExpansion = sizeOfExpansion;
    info.Prefab = prefab;
}

2 references
virtual public GameObject TakeObject() { return null; }
2 references
virtual public void ReturnObject(GameObject obj) { }
```

ObjectPoolBase를 상속 하여 풀 데이터 구조와 오브젝트 초기화 같은 내부 동작을 커스텀 할 수 있게 설계하였고, ObjectPoolManager를 통하여 오브젝트 풀의 생성, 소멸과 오브젝트의 지급 반환을 조작할 수 있게 하였습니다.

```
public U GeneratePool<U>(T key, U objPool) where U : ObjectPoolBase
{
    pools[key] = objPool;
    return pools[key] as U;
}
0 references
public void RemovePool(T key)
{
    if (DoesPoolExist(key) == false)
    {
        Debug.Log($"No matching pool found, Key: {key}");
        return;
    }
    pools.Remove(key);
}
```

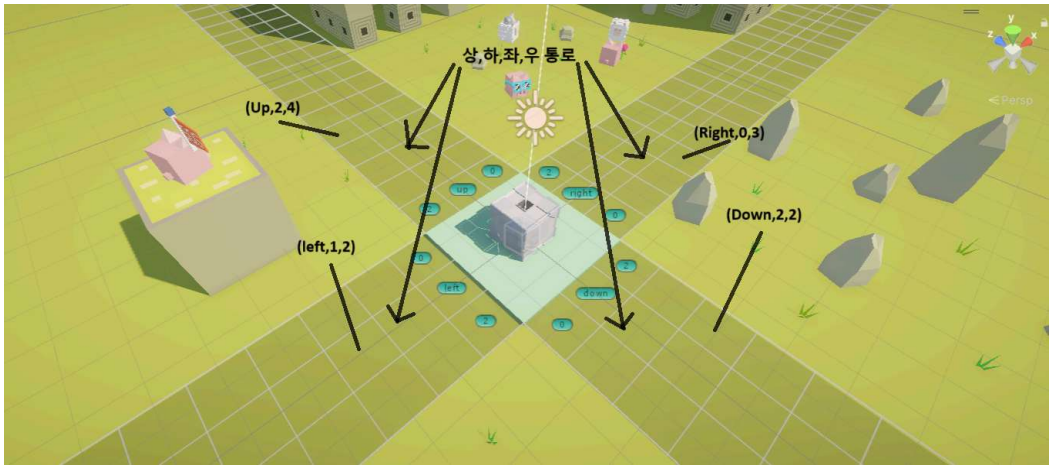
```
public GameObject TakeObject(T key)
{
    return GetPool(key)?.TakeObject();
}
0 references
public void ReturnObject(T key, GameObject obj)
{
    GetPool(key)?.ReturnObject(obj);
}
```

유연한 제어와 커스터마이징을 통한 확장성에 중심을 두고 설계하였으나 **비슷한 오브젝트라도 다른 초기화를 적용해주고 싶다면 새롭게 상속**해서 만들어야 한다는 점이 아쉬웠습니다. 오브젝트 풀을 **생성할 시점에 초기화 매커니즘을 설정**해 줄 수 있다면 더 나은 오브젝트 풀이 될 것 같습니다.

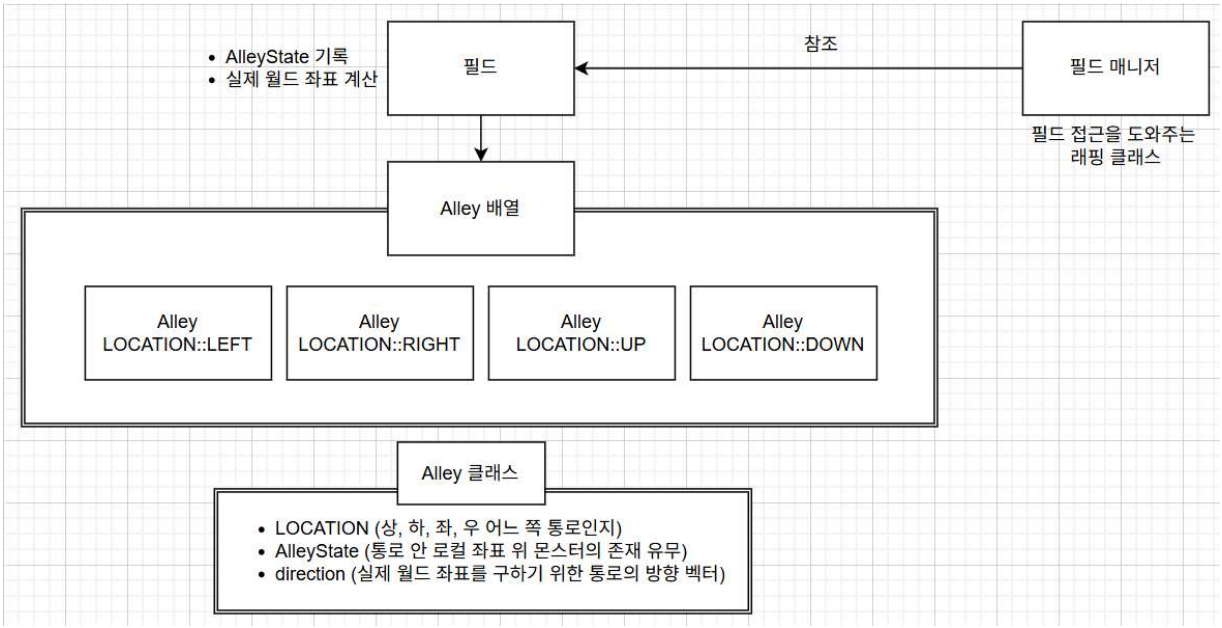
필드

몬스터의 스폰과 이동에 있어 방향을 기준으로 움직이는 것 보다 **로컬 그리드 좌표를 기반으로 실 위치를 계산해 처리**하는 것이 **무대의 위치가 바뀌어도 문제가 생기지 않고 정확도 면에서도 더 나을 것이라 판단**하여 설계된 클래스입니다.

필드는 몬스터가 소환되고 이동할 상,하,좌,우 통로, 통로 위의 타일을 소유하고 있는 클래스입니다. 필드 매니저를 통해 로컬 그리드 좌표의 타일에 접근 할 수 있게 구현되었습니다.



몬스터들의 이동 방식과 피격, 공격이 서로의 위치에 따라 딜레이 되거나 바뀌어야 했기에 필드 안의 각 **통로(Alley 클래스)에서 타일 위 몬스터의 유무를 기록**해두고 몬스터의 행동 시점에 필드 매니저를 통해 행동에 변화를 줄 수 있게 하였습니다.



마무리

약 한달간의 개발기간동안 프로젝트 진행도를 끌어올리기 위해 매일같이 프로젝트에 몰두했습니다. **기획부터 개발, 사용자 평가를 통한 개선**까지 전체 파이프라인을 경험, 불특정 유저들에게 게임을 공개한다는 불안감과 기한을 맞춰야 한다는 압박감 그리고 지인들의 피드백을 통해 발견되는 기획의 부족한 점 까지 학교에서 진행했던 프로젝트들과는 느껴지는 무게와 감상이 정말 많이 달랐습니다. 이 경험을 통해 **유니티를 활용한 게임 개발, 최적화, 실제 유저를 통한 시스템 개선과 난이도 조정 등 실제 게임 개발에 대한 감각**을 얻게된 값진 경험이었습니다.