팀 바운딩 박스
게임 - Blocking

Project Cube

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

“큐브를 굴려서 적을 처치해나가는 퍼즐게임”이라는 컨셉트에서 시작된 프로젝트입니다. 각 방향에서 몬스터가 다가오고 플레이어의 영역에 다가오면 정해진 패턴에 따라 공격을 진행합니다. 몬스터가 다가오기 전에 또는 공격을 피해 몬스터를 모두 제거하는 것이 목표인 게임입니다.
만들래 10분 게임 공모전에 참여하기 위해 결성된 2인 팀이으로 약 한 달간 플레이가 가능한 수준의 프로토 타입을 만들어야 했기 때문에 핵심 매커니즘을 하이퍼 캐주얼 장르에 맞게 제한하고 쉬움 2개, 어려움 2개, 보스 1개의 레벨로 구성하여 제출하는 것을 목표로 했습니다.

게임 플레이 사진

목표

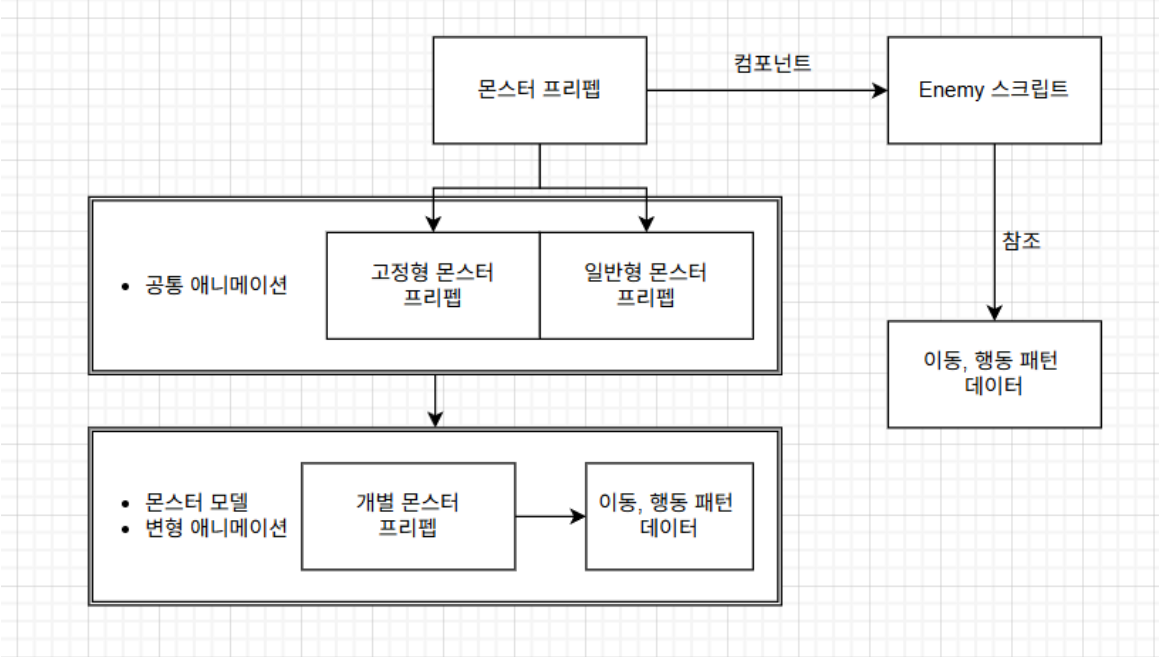
1. **제한된 시간에 맞춰 게임을 기획하고 개발**
2. **학교 과제를 넘어 실제로 게임을 만들어 불특정 대상들에게 게임을 공개해보는 경험**

주요 작업

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

몬스터

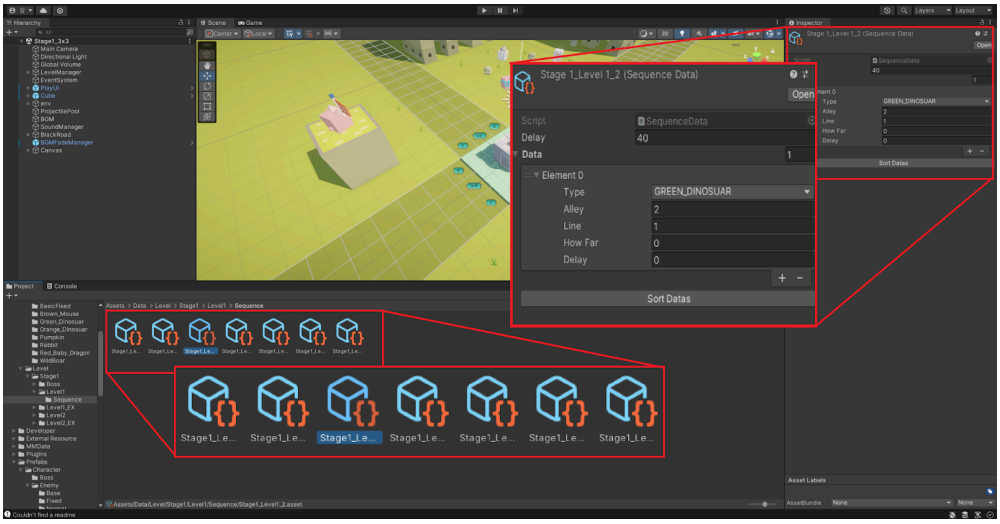
몬스터의 종류로는 **일반형, 플레이어의 공격에 넉백을 받지 않는 고정형, 보스** 3종류가 있습니다.
몬스터의 종류가 추가될 상황을 고려해 **일반형과 고정형 모두 같은 Enemy 스크립트로 동작**하지만 기존 프리펩을 상속하여 외형과 애니메이션을 다르게 설정할 수 있도록 구현하였고 **이동, 행동 패턴 데이터를 받아 다르게 동작**하게 구현되었습니다.



보스는 일반적인 몬스터와는 다르게 기믹을 수행할 수 있어야 하기 때문에 Enemy 스크립트가 아닌 **독립적인 스크립트를 따라 동작하게 구현**하였고, 차후에 보스 기획이 더 생겼을 때 추상화 하기로 하였습니다.

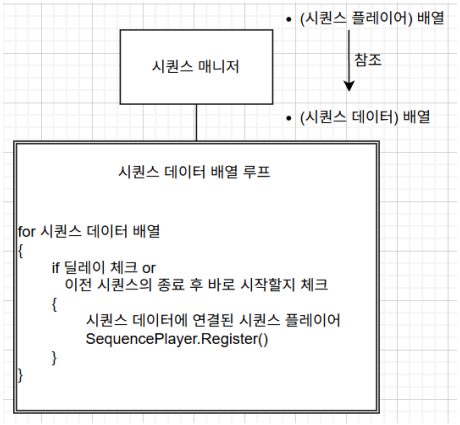
시퀀스 시스템

데이터를 기반으로 레벨이 구성되기 때문에 몬스터가 생성될 정보에 대한 데이터를 읽고 수행할 클래스를 설계하였습니다. **순차적으로 생성되는 몬스터의 정보를 시퀀스**라고 부르고 하나의 레벨 데이터에 담겨있는 **여러 시퀀스들의 정보에 따라 동시 또는 순차적으로 확인하여 몬스터를 생성**해주는 시스템 입니다.



각 시퀀스 마다 딜레이 또는 이전 시퀀스 종료 같은 실행 조건이 존재하고 시퀀스 내부에 몬스터 스폰에 대한 데이터 또한 존재하여 **시퀀스 실행과 몬스터 생성을 담당할 두 종류의 클래스가 필요**하다고 판단하였고, **SequenceManager에서 각 시퀀스의 딜레이와 이전 시퀀스의 종료 상태를 체크**하여 실행 시점을 확인하고 **SequencePlayer에 시퀀스를 등록,** 이후 **SequencePlayer 내부에서 독립적으로 몬스터 스폰 데이터를 확인하여 생성**하는 방식으로 설계하였습니다.

```
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();
    }
}
```



각 SequencePlayer는 서로와 관계 없이 몬스터를 생성해야 했기에 매니저에서는 플레이어를 실행, 제거만 담당하고 각 플레이어의 Play 함수는 **이벤트 시스템**을 활용해 **시퀀스가 시작되어야 할 시점에 턴 시작 이벤트를 구독**시켜 **각각의 시퀀스가 자체적으로 동작**하게 구현하였습니다.

오브젝트 풀 & 매니저

다양한 오브젝트들이 자주 생성, 소멸되는 구조였기 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.

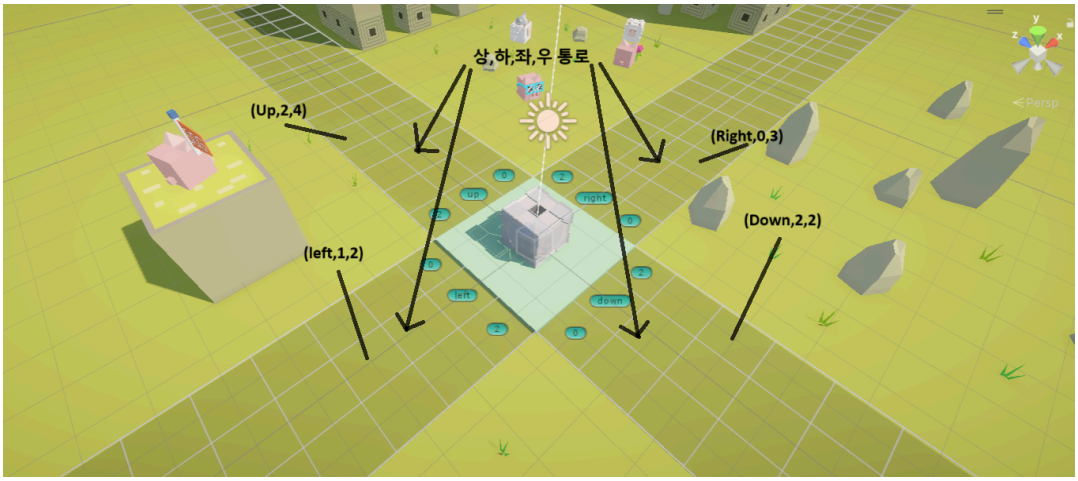
```
public U GeneratePool<U>(T key, U objPool) where U : ObjectPoolBase
{
    pools[key] = objPool;
    return pools[key] as U;
}
0 references
public void RemovePool(T key)
{
    if (DoesPoolExist(key) == false)
    {
        Debug.Log($"No matching pool found, Key: {key}");
        return;
    }
  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

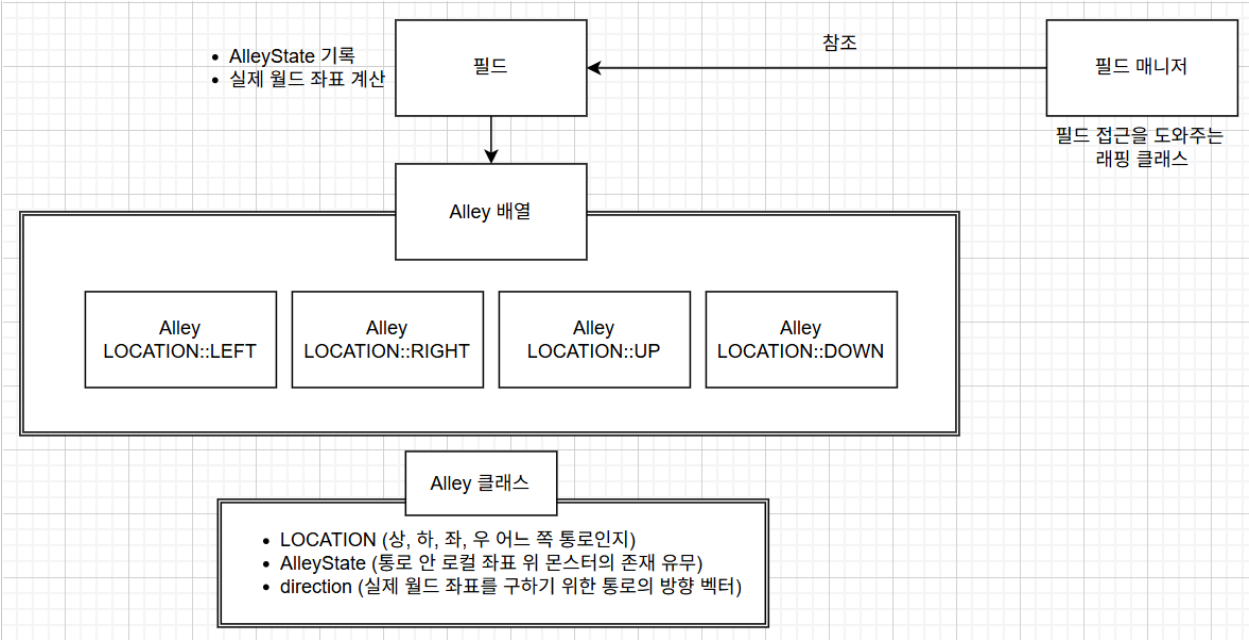
유연한 제어와 커스터마이징을 통한 확장성에 중심을 두고 설계하였으나 **비슷한 오브젝트라도 다른 초기화를 적용해주고 싶다면 새롭게 상속**해서 만들어야 한다는 점이 아쉬웠습니다. 오브젝트 풀을 **생성할 시점에 초기화 매커니즘을 설정**해 줄 수 있다면 더 나은 오브젝트 풀이 될 것 같습니다.

필드

몬스터가 이동할 지점에 대한 정확한 정보나 같은 방향에서 오는 **몬스터는 이동 또는 피격을 통해 위치가 변경 될 때 서로 충돌 같은 상호작용이 존재**했기 때문에 같은 방향에서 오는 **(같은 통로에 속한) 몬스터의 정보**를 확인할 방법이 필요했습니다. 따라서 몬스터가 소환되고 이동할 상,하,좌,우 통로, 통로 내부의 가상의 그리드 위에 존재할 타일, 타일에 존재할 몬스터를 가르킬 배열을 소유하고 있는 클래스를 만들었고, 필드 매니저를 통해 로컬 그리드 좌표의 타일에 접근 하여 위치와 현 타일의 상태를 확인할 수 있게 구현하였습니다.



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



마무리

제한된 시간 안에 기획–개발–배포 전 과정을 직접 경험하였고, 레벨과 몬스터를 데이터 기반 구조로 설계하여 여러 번 데이터를 구현, 수정하며 데이터 구조 설계의 중요성을 체감했습니다. 또한, 오브젝트 풀을 구현하며 초기화 버그, 이벤트 시스템과의 연계, 몬스터 상호작용을 구현하며 생긴 결합 문제들을 경험하며 시스템 간의 연계 구조에 대해 고민해 보며 유지보수가 가능한 설계의 중요성을 다시 생각해 볼 수 있던 기회가 되었습니다. 이후 프로젝트에서는 풀 초기화 전략 같은 시스템의 역할을 확실히 파악하고, 결합을 느슨하게 만들어 줄 시스템에 집중해 유지보수에 더욱 이점을 가질 수 있게 발전시키고자 합니다.