Mixmaster

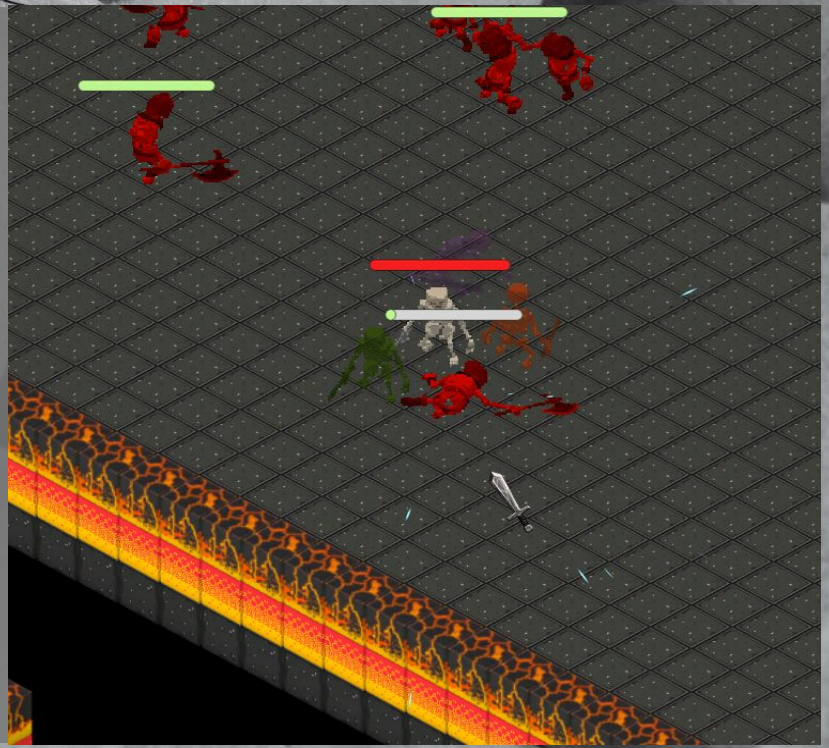
믹스마스터 모작

[게임 소개]

- 이 게임은 플레이어와 3마리의 몬스터(펫)이 한 팀을 이루어 전투하는 방식의 RPG 게임입니다.
- 필드에서 몬스터를 처치하면 일정 확률로 해당 몬스터를 자신의 팀원으로 영입할 수 있습니다.

[제작 기간]

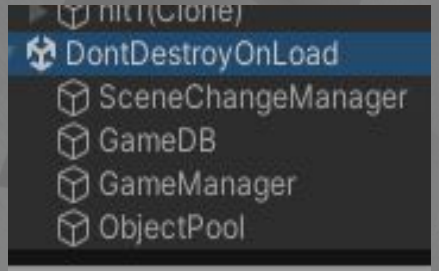
- 24.00.00~24.00.00(약2주)
- Unity 2D 개인 프로젝트



주요 기능

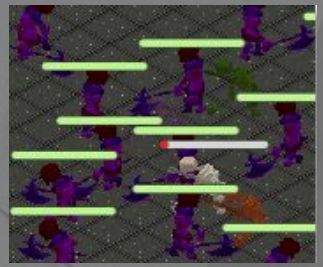
1. 인스턴스 관리

- Singleton<T>
  - dbmanager.cs
  - scenemanager.cs
  - gamemanager.cs
  - objectpool.cs



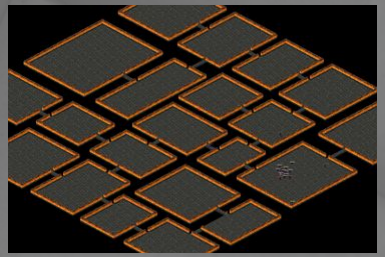
2. 오브젝트 관리

- 오브젝트풀링
  - Monster object
  - effet object



3. 랜덤맵 생성

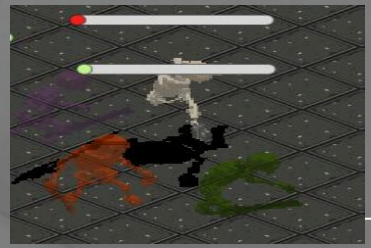
- BSP Algorithm
- Isometric TileMap
- Navmesh



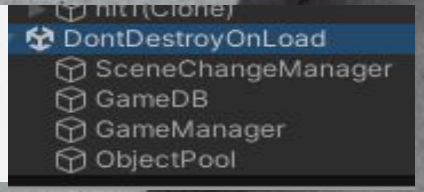
4. 상태 관리

- 상태패턴
  - idle
  - move
  - attack
  - chace

- Blend Trees
  - 8 directions Sprite Img

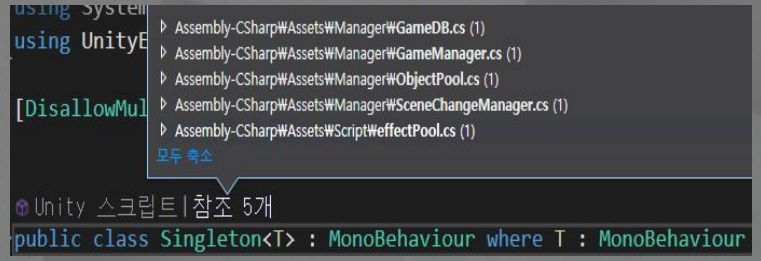


I. 인스턴스 관리 - Singleton<T>



싱글톤 패턴 구현 이유

- **GameDB.cs** : 몬스터, 플레이어정보와 같은 중요한 데이터는 씬이 변경되어도 유지 되어야 해서 싱글톤 사용.
- **SceneChange Manager.cs** : 변경 된 씬에서도 플레이어의 현재 위치를 알 수 있게 하기 위해 싱글톤 사용
- **ObjectPool** : 객체를 자주 생성하고 파괴하는 경우 가비지 컬렉터가 실행되어 성능 문제를 야기할 수 있기 때문에 싱글톤으로 몬스터 및 이펙트 오브젝트를 미리 생성해두고 필요 시 활성화/비활성화 하였습니다.

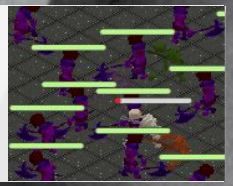


싱글톤 제네릭 클래스 구조 사용 이유

- 싱글톤 패턴을 필요로 하는 클래스마다 코드 작성하지 않게 하기 위해 제네릭 싱글톤 구조를 상속받게 하였습니다.

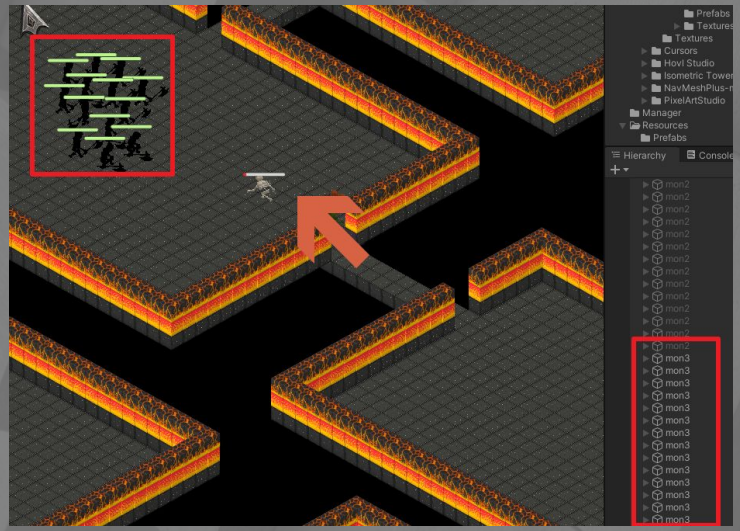
```
public class GameDB : Singleton<GameDB>
public class GameManager : Singleton<GameManager>
public class ObjectPool : Singleton<ObjectPool>
public class SceneChangeManager : Singleton<SceneChangeManager>
public class effectPool : Singleton<effectPool>
```

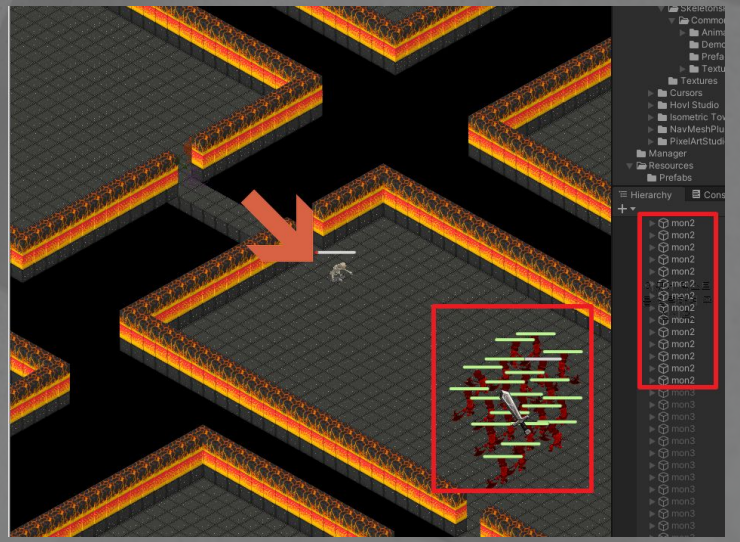
2. 오브젝트 관리



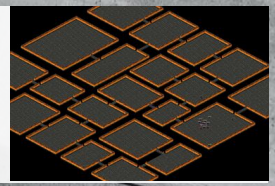
Object Pooling

- 프레임 드랍 현상을 최소화하기 위해, 몬스터 오브젝트를 비활성화 상태로 미리 생성하였습니다.
- 구역별로 등장하는 몬스터가 다르므로, 몬스터 프리팹 이름을 키로 사용하여 **Dictionary** 자료 구조를 통해 접근하도록 하였습니다.
- 플레이어가 위치한 구역에서만 몬스터 오브젝트가 활성화되도록 설정하였습니다.
- 몬스터를 처치하면 오브젝트를 재사용할 수 있도록 반환하게 하였습니다.





3-I. 랜덤맵 생성



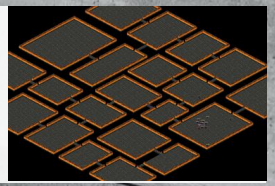
1. 분할 2. 연결 3. 가공 4.길생성

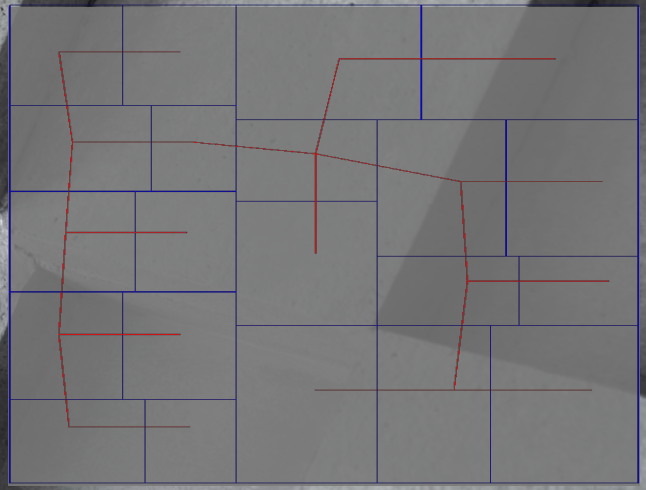
- 특정 조건이 충족될 때까지 함수를 재귀적으로 호출하면서 이진트리 구조로 노드를 **left, right** 변수에 담기도록 설계하였습니다.

```
public void divideMap()
{
    if (data.width <= 70 && data.height <= 70)
        return;
    RectInt leftRect;
    RectInt rightRect;
    if (data.width >= data.height)...
    else...
    left = new MapNode(leftRect);
    right = new MapNode(rightRect);


    left.divideMap();
    right.divideMap();
}
```

3-2. 랜덤맵 생성



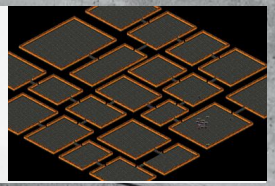


1. 분할 | **2. 연결** | 3. 가공 | 4.길생성

- **left,right** 변수의 모든 노드 쌍을 검사하고 최단 거리를 찾아 가까운 방끼리 연결하였습니다.

```
foreach (var lNode in leftList)
{
    foreach (var rNode in rightList)
    {
        float tempGap = CalculateDistance(lNode.data, rNode.data);
        if (gap > tempGap)
        {
            node1 = lNode;
            node2 = rNode;
            gap = tempGap;
        }
    }
}
```

3-3. 랜덤맵 생성

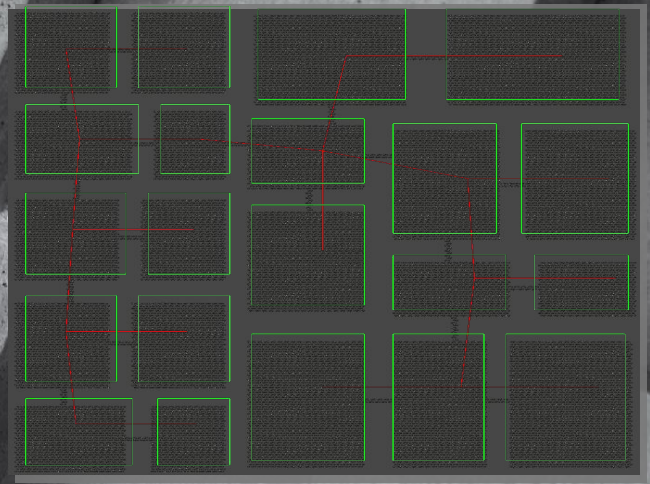


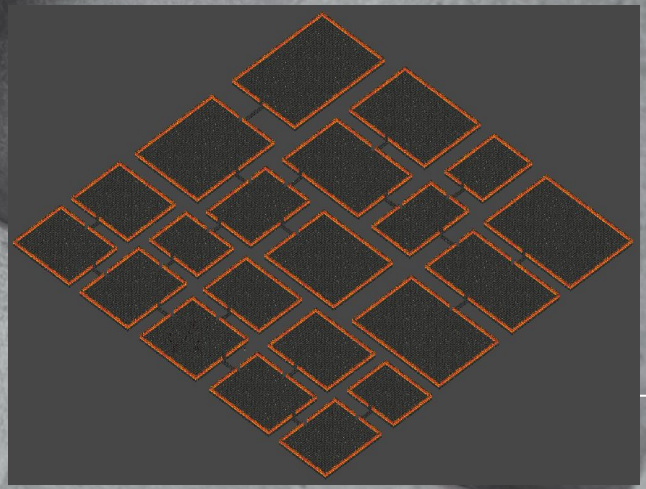
1. 분할 2. 연결 **3. 가공** 4.길생성

– 맵이 붙어있기 때문에 방 크기를 **80%**로 줄인 후 타일맵을 방 사이즈에 맞게 찍어냈습니다.

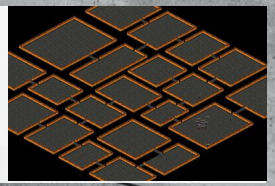
- 맵을 연결하는 다리가 대각선이 아닌 일자로 연결되도록 각 맵의 중심 좌표를 이용하여 이어주는 작업을 진행하였습니다.

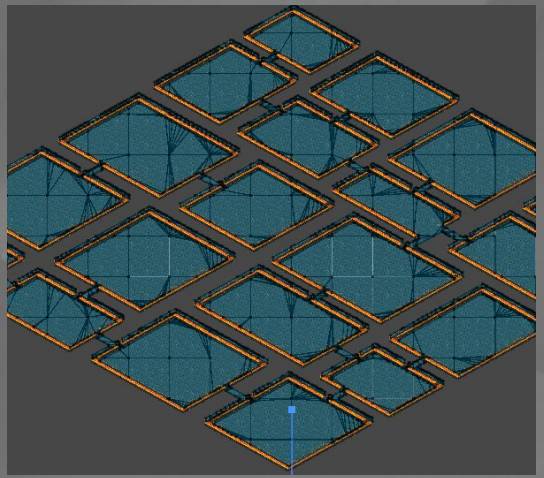
- **isometric** 타일을 사용하여 타일맵을 구성하였습니다.





3-4. 랜덤맵 생성





1. 분할 2. 연결 3. 가공 **4.길생성**

– 각 맵의 테두리 안쪽과 방과 방을 연결하는 다리만 플레이어와 몬스터가 이동할 수 있도록 **Navmesh**의 **Bake** 기능을 이용하여 타일맵을 구웠습니다.

4.상태관리

- 추후, 상태가 다양하게 추가될 수 있기 때문에 클래스 단위로 관리 되면서 쉽게 전환될 수 있도록 상태 인터페이스와 상태머신 클래스를 구현하였습니다.

```
// 상태 인터페이스 정의
참조 6개
public interface IState<T>
{
    참조 6개
    void Enter(T t);
    참조 6개
    void Update(T t);
    참조 6개
    void Exit(T t);
}

// 상태 머신 클래스 정의, 현재 상태를 추적하고 상태를 변경하는 데 사용
참조 2개
public class StateMachine<T>...

Unity 스크립트(자산 참조 6개) | 참조 26개
public class MonsterController ...
참조 2개
public class IdleState ...
참조 2개
public class MoveState ...
참조 1개
public class AttackState ...
참조 3개
public class DeadState : IState<MonsterController>
{
    참조 2개
    public void Enter(MonsterController monster)...

    참조 2개
    public void Update(MonsterController monster)...

    참조 2개
    public void Exit(MonsterController monster)...
}
```