PORTFOLIO

(1 인 개발 경험치 던전)

정승현

(https://github.com/JungSH95)

목차

1. 게임 소개

게임 이름 : 경험치 던전

게임 소개 : 간단한 조작 및 자동공격 시스템을 사용한, 로그라이크 게임

개발 환경 : unity 2018.4.8f1, GitHub, Visual Studio

사용 언어 : C#

개발 목표 : 1 인 개발 프로토 타입 제작 후 Google Play Store 출시,

유니티 엔진 사용 능력 향상

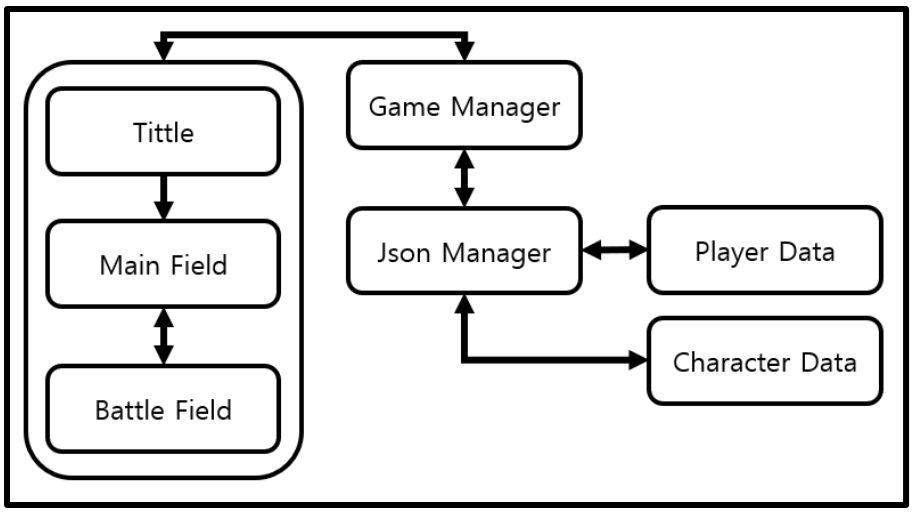
Google Play : https://play.google.com/store/apps/details?id=com.SHGames.ExpDungeon

GitHub : https://github.com/JungSH95/Senier-Project

플레이 영상 : https://youtu.be/hi88ptPsUEY

2. 전체 구성

씬 구성 : 씬은 크게 4가지로 타이틀, 마을, 전투, 튜토리얼로 구성되어 있습니다.



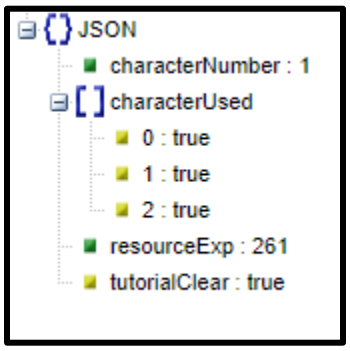
Game Manager 와 **Json Manager** 는 모든 씬에서 사용 필요성이 있으며, 자주 사용하기 때문에 **DontDestroyOnLoad()**를 사용하여 파괴시키지 않았으며, **싱글톤**을 사용하여 접근을 용이하게 하여 사용하기 편리하게 하였습니다.

```
public PlayerData playerData;
public OptionData optionData;

public List<CharacterBase> characterInfoList;
```

Game Manager 에서는 플레이어 데이터와 옵션 데이터, 캐릭터들의 정보를 리스트로 가지고 있습니다.

```
private void Start()
{
    JsonManager.Instance.PlayerDataLoad();
    JsonManager.Instance.OptionDataLoad();

    for (int i = 0; i < 3; i++)
        JsonManager.Instance.CharacterDataLoad(i);
}
```

Game Manager 에서 **JsonManager** 를 통해 플레이어, 옵션, 캐릭터의 정보를 불러옵니다.

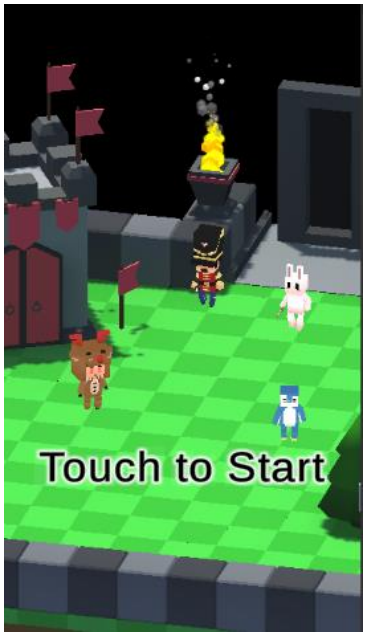


플레이어 데이터는 현재 선택한 캐릭터의 번호(characterNumber)를 가지고 있으며 현재 게임 내의 사용할 수 있는 캐릭터를 characterUsed 를 통해 저장했습니다. 각 캐릭터의 능력치를 올릴 수 있는 자원으로 경험치 자원이 있으며 튜토리얼 클리어 판단 유무를 저장합니다.

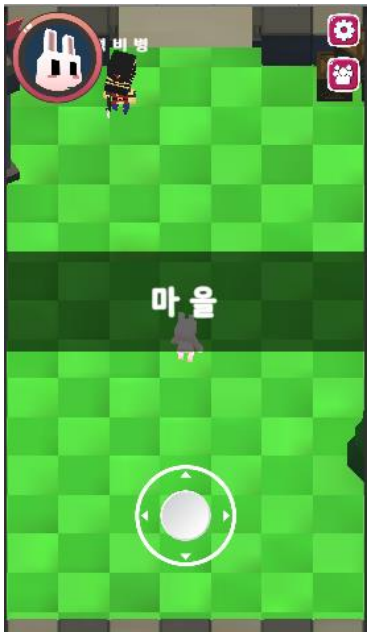
옵션 데이터는 배경음과 효과음의 On/Off 를 **Bool 타입**으로 저장합니다.

캐릭터 데이터는 능력치(근력, 민첩, 건강)이 있으며 무기의 종류와 해당 정보와 체력, 공격력, 공격속도, 이동속도를 가지고 있습니다. 공격력은 근력과 무기 레벨에 따라서 상승되며, 공격속도는 민첩, 체력은 건강에 의해서 상승합니다.

Json 을 선택한 이유로 유니티 내에서 **JsonUtility API** 를 제공하고, 플레이어 데이터와 캐릭터 데이터처럼 간단한 데이터를 저장할 수 있으며 직관적이므로 보기 편리하여 선택하였습니다.



타이틀 씬에서는 **Game Manager** 에서 **Json Manager** 를 통해 json 데이터(플레이어, 옵션, 캐릭터)를 불러옵니다. 이후 화면을 클릭(터치) 입력이 들어오게 되면 마을 씬으로 넘어가게 됩니다.

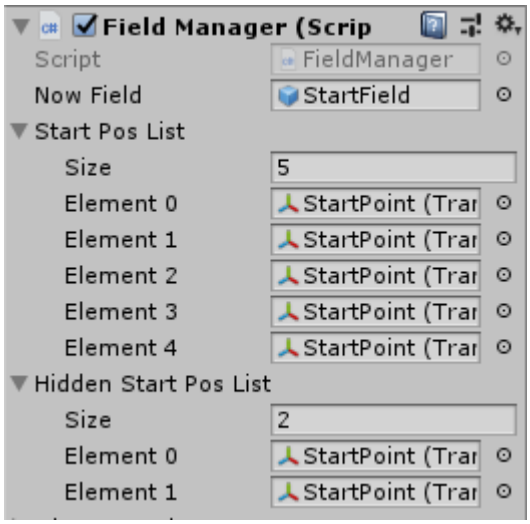


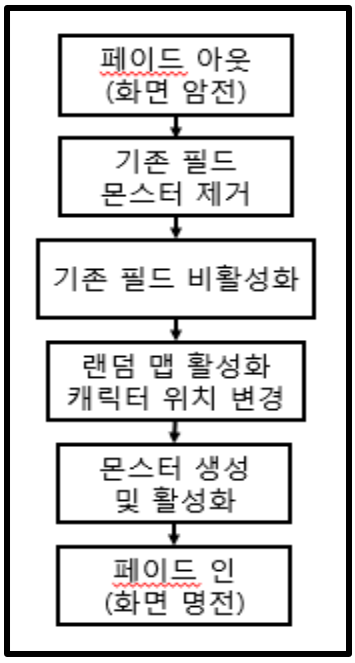
마을에서 우측 상단의 **옵션 버튼**을 누르면 배경음과 효과음을 On/Off 할 수 있고, 좌측 상단의 **캐릭터 버튼**을 클릭하면 캐릭터의 능력치 별 레벨을 볼 수 있으며, 경험치 자원을 통해 각 능력치의 레벨을 올릴 수 있습니다.

우측 상단에 옵션 버튼 아래의 버튼은 **캐릭터 도감 버튼**입니다. 클릭하면 현재 해금되어 있는 캐릭터에 한하여 해당 캐릭터의 모습과 클릭 시 해당 캐릭터의 정보를 확인할 수 있습니다. 해금되어 있는 캐릭터는 마을 지역에 해당 캐릭터 별 위치가 지정되어 있으며 그 위치에 있는 캐릭터가 활성화 되여 그 캐릭터를 클릭하여 **캐릭터 변경**이 가능합니다. NPC 를 클릭하면 해당 NPC 의 대사를 표시합니다. 위에 있는 문에 들어가면 **전투 씬**으로 이동합니다.

전투 씬에서는 랜덤으로 필드를 이동하여 전투를 진행합니다. 히든 스테이지를 클리어 시 상단에 해당 히든 스테이지의 캐릭터를 획득한 것을 확인할 수 있습니다.

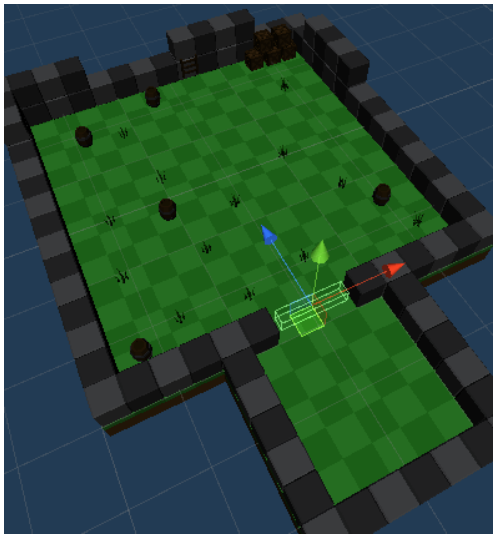
3. 전투 맵

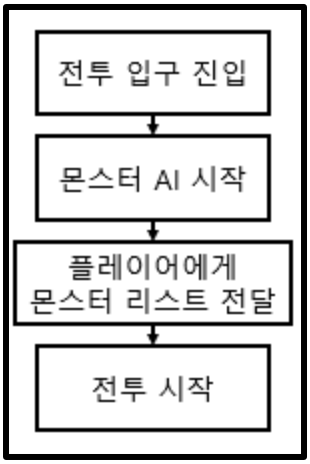




마을을 통해 **전투지역에 들어오게** 되면 처음 시작의 방에서 게임을 시작하게 됩니다. 전투 씬에서 Field Manager 를 통하여 다음 필드와 몬스터 재생성, 전투 시작을 진행합니다.

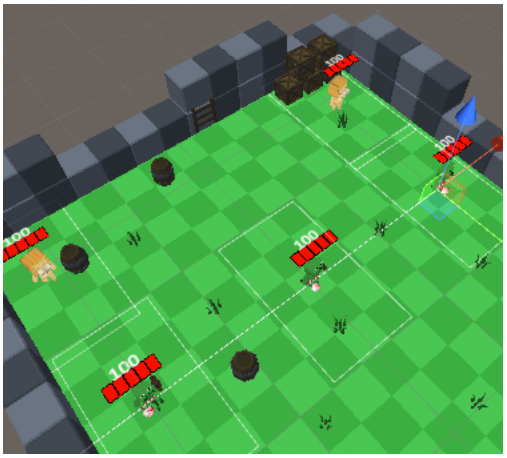
필드 매니저는 필드의 캐릭터 시작 지점(일반 필드 5 가지, 히든 필드 2 가지)들을 가지고 있습니다. 포탈을 통해서 다음 필드로 이동하게 되면 페이드 아웃한 후 기존 필드의 **몬스터를 제거(오브젝트 풀 반납)** 후 필드를 비활성화하고, 랜덤을 통해 StartPosList 에서 시작위치를 얻어와 캐릭터의 위치를 변경하고 필드를 활성화 한 뒤 몬스터 생성, 활성화 과정을 거쳐 페이드 인하게 됩니다.





전투 필드에 들어와 **입구**를 지나가게 되면 몬스터의 AI 가 시작하고, 플레이어에게 해당 몬스터 리스트를 전달합니다.

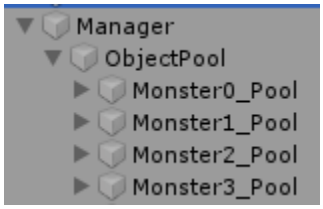
4. 몬스터 생성



```
public void SetSpawnTransform(GameObject fieldObj)
{
    spawnTransform = fieldObj.transform.Find("Spawn").transform;

    int nSize = spawnTransform.childCount;
    for(int i=0; i< nSize; i++)
        points.Add(spawnTransform.GetChild(i));

    isSpawnEnd = false;

    StartCoroutine(CreateMonster());
}
```

몬스터의 생성은 플레이어가 **필드를 이동하는 과정**에서 생성합니다. 다음 필드가 랜덤으로 정해지면 해당 필드를 Spawn Manager 에 넘겨주고, 해당 필드의 Spawn Object 를 찾아 **자식의 수**만큼 위치(Transform)을 **리스트**에 추가합니다.

```
IEnumerator CreateMonster()
{
    for(int monsterCount = 0; monsterCount < points.Count; monsterCount++)
    {
        GameObject newMonster = GetMonsterObject(points[monsterCount].tag);
        newMonster.transform.parent = points[monsterCount].transform;
        newMonster.transform.position = points[monsterCount].position;
        newMonster.transform.Find("Character").gameObject.
            transform.position = newMonster.transform.position;
        monsterList.Add(newMonster.transform.Find("Character").gameObject);
    }

    isSpawnEnd = true;
    isMonsterClear = false;
    yield return null;
}
```



몬스터를 생성할 위치와 수를 파악했으므로 몬스터의 타입을 **태그**로 확인하여 해당 몬스터를 **Object Pool** 을 통해 받아와 몬스터의 위치를 설정합니다.

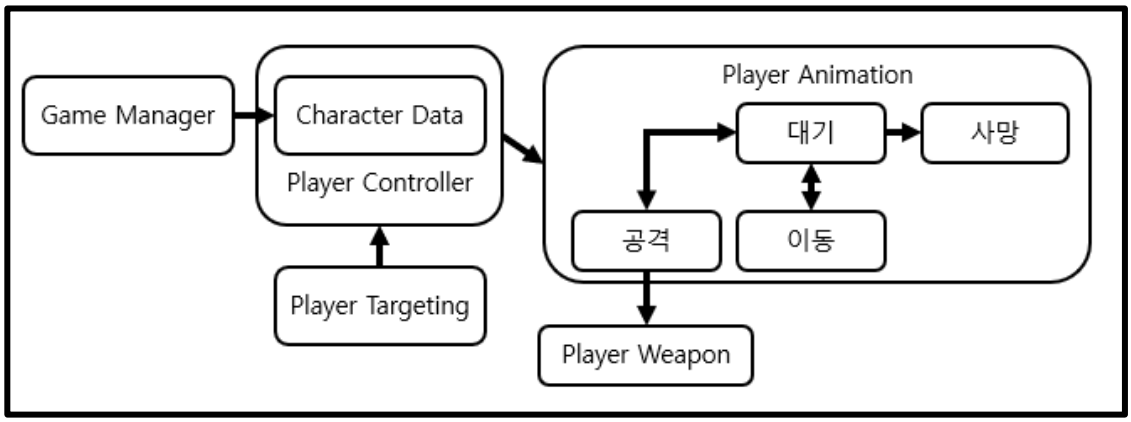
```
public GameObject GetMonsterObject(string type)
{
    switch(type)
    {
        case "MonsterType0":
            return ObjectPool.Instance.PopFromPool("Monster0");
        case "MonsterType1":
            return ObjectPool.Instance.PopFromPool("Monster1");
        case "MonsterType2":
            return ObjectPool.Instance.PopFromPool("Monster2");
        case "MonsterType3":
            return ObjectPool.Instance.PopFromPool("Monster3");
        default:
            return ObjectPool.Instance.PopFromPool("Monster0");
    }
}
```

몬스터의 경우 계속해서 재사용하므로 오브젝트 풀을 사용하여 추가적인 생성과 삭제에 대한 부담을 줄였습니다.

5. 플레이어

플레이어의 이동은 아래 Asset의 **Fixed Joystick**을 사용하여 **Player Controller**에서 처리하였습니다.

조이스틱 :https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631



플레이어는 Game Manager를 통해 Character Data를 Player Controller에 가져옵니다. Player Controller에는 이동, 사망, 충돌처리를 합니다.

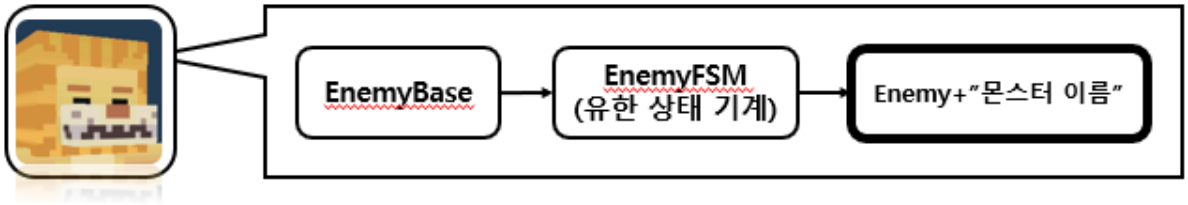
전투 시작과 함께 몬스터 리스트를 전달받으면 몬스터들에게 **Ray**를 쏴서 장애물에 안 걸리고 **태그가 Monster**인 것 중 가장 가까운 몬스터를 타겟으로 잡습니다.

타겟이 존재하고, 이동중이 아닐 경우 플레이어는 공격 애니메이션을 통해 공격 투사체를 발사합니다.

플레이어의 투사체는 정면으로 발사합니다.

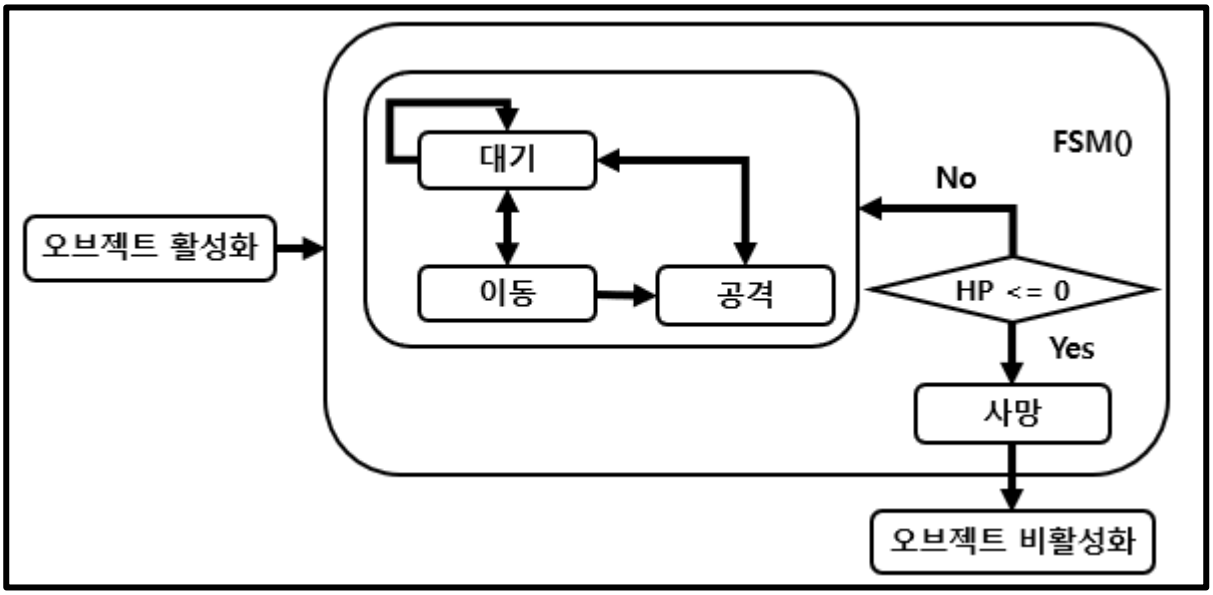
```
public void Shoot(float dmg = 1f)
{
    rigidbody = GetComponent<Rigidbody>();
    rigidbody.velocity = transform.forward * 5f;
    damage = dmg;

    // 삭제 안될 경우 방지
    StartCoroutine(CoWeaponDestroy());
}
```

6. 몬스터



몬스터는 Enemy + "몬스터 이름"으로 된 스크립트를 가지고 있습니다. 해당 스크립트는 Enemy FSM 을 상속받으며 Enemy FSM 는 Enemy Base 를 상속받습니다.

Enemy Base 에서는 플레이어의 무기 투사체와 충돌처리를 진행하며, 몬스터의 정보(체력, 공격력, 사거리, 공격 쿨타임, 이동속도)가 있습니다.



몬스터의 유한 상태 기계(FSM)은 대기, 이동, 공격, 사망의 상태를 가지고 있습니다. 몬스터의 이동은 유니티 내의 Nav Mesh Agent 를 사용하여 플레이어가 전투 지역에 진입을 하게 되면 FSM 이 시작하여 플레이어를 추적하게 하였습니다.

```
protected virtual IEnumerator FSM()
{
    yield return null;

    InitMonster();

    while(true)
    {
        if (currentHp <= 0)
            currentState = State.Dead;

        yield return StartCoroutine(currentState.ToString());
    }
}
```

FSM() 코루틴을 사용하여 현재 해당하는 상태에 맞는 코루틴을 실행하고, 만약 체력이 0 이하의 경우 상태를 사망으로 변경하여 사망 코루틴을 실행합니다.

감사합니다.