프로젝트 최종보고서

(프로젝트 명 : 루카스 어드벤처)

32153033 이건희, 32154024 장현준

[목차]

1. 프로젝트 개요
    1.1 제안 배경
    1.2 기능 개요
    1.3 기대 효과
    1.4 협업 방식 및 개발 환경
2. 사용자 가이드
    2.1 제공 기능
    2.2 설치 및 사용 방법
3. 개발자 가이드
    3.1 주요 데이터 및 알고리즘 명세 (외부 API 명세 포함)
    3.2 소스 코드 구조(GitHub 링크 포함)
    3.3 빌드 및 테스트 방법
4. 개발 내용
5. 테스트 보고서
    5.1 테스트 항목 및 결과
    5.2 이슈 트래킹 로그
6. 문제점 및 개선 방향
7. 프로젝트 후기

1. 프로젝트 배경

1.1 제안 배경

- 게임 개발에 대한 흥미와 개발 과정에 대한 전체적인 이해하기 위함
- 새로운 장르의 개발에 대한 도전

1.2 기능 개요

- 로그라이크 RPG
- 게임을 진행하면서 점점 강력해지는 캐릭터를 조작하며 스테이지를 클리어하는 진행 방식
- 탑다운 뷰를 통한 종-횡 이동이 주를 이루는 조작
- 기본적인 인터페이스(플레이어 상태바, 소지품창, 옵션)를 지원하며 더 직관적인 상태를 파악할 수 있음
- 구현된 상점에서 몬스터를 처치하고 나온 재화를 사용하여 플레이어 캐릭터를 강하게 만드는 아이템을 구매할 수 있음
- 무기 아이템은 소모품으로서 사용되며 몬스터를 통해 획득할 수 있고 사용횟수가 지정됨.
- 습득되는 무기에 따라 플레이어는 매번 게임을 즐길때 마다 다양한 조합을 느낄 수 있음

1.3 기대 효과

- 2D 게임, Unity, C# 인터페이스 이해도 증가
- 게임 시스템의 작동 원리의 이해
- Github활용으로 인한 협업 능력의 향상

1.4 협업 방식 및 개발환경

게임 엔진 

프로젝트관리 

사용 언어 

문서 관리 

커뮤니케이션 

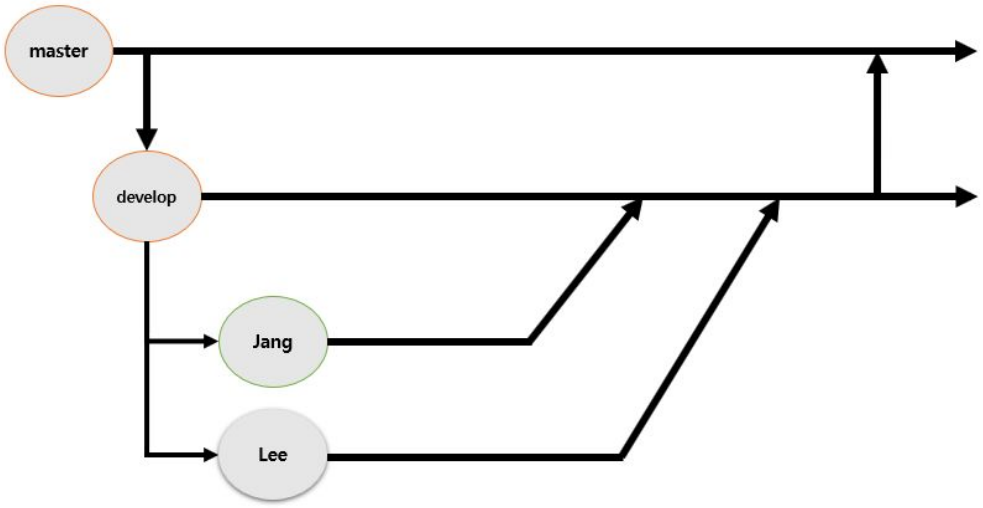
협업 방식

- 각자가 만든 요소들은 Github를 통한 프로젝트 공유
- 일단 위에 주어진 대로 작업을 처리하되, 필요에 의해 한 가지 작업을 동시에 진행할 수 있다
- 프로젝트 완성단계에서 서로 완성한 요소들을 적절히 조합하고 부족한 점을 보완 및 버그수정을 한다.
- 각 주차의 금~일요일에 zoom/카카오 보이스톡/ 대면을 통해 서로가 만든 요소를 합쳐서 테스트를 실행한다.
- 문서작업은 구글문서를 통해 다 같이 작업을 한다.

프로젝트 관리

- Github link : https://github.com/hyeonjun414/Lucas-Adventure

프로젝트는 깃허브를 통해 코드를 공유하며, 아래 이미지의 개발 형태를 갖추고 있다.



- master에서 develop branch로 나누어지고 develop에서 각자가 맡은 부분으로 나누어 각각 Jang과 Lee로 다시 나뉘게 된다.
- develop branch는 Jang과 Lee에 만든 부분들을 순서대로 병합시키면 완료되면 develop은 정상적인 가동이 가능하다고 판단하여 master에 완성된 게임 버전 1을 올리게 된다.
- 가능한 master에는 파일을 업로드 하지 않으며 모든 기능을 갖춘 develop branch만 올라가게 된다.

2. 사용자 가이드

2.1 제공 기능

캐릭터

Player	기능	설명
	Hp	Player 체력
	Mp	스킬을 사용하기 위한 에너지
	Exp	경험치
	공격	Player의 공격력 수치
	레벨	증가 할 때 마다 Player의 능력치가 상승
	방어	몬스터에게 공격을 받을 때 일부 데미지를 절감시켜주는 것
	스피드	Player가 달리는 속도
	Swap	무기는 바꿔주는 기능
	Inventory	Unique Item을 획득 시 저장하는 공간
	Weapon Slot	기본무기를 제외한 나머지 무기를 획득 시 저장되는 공간(최대 한도는 4개)

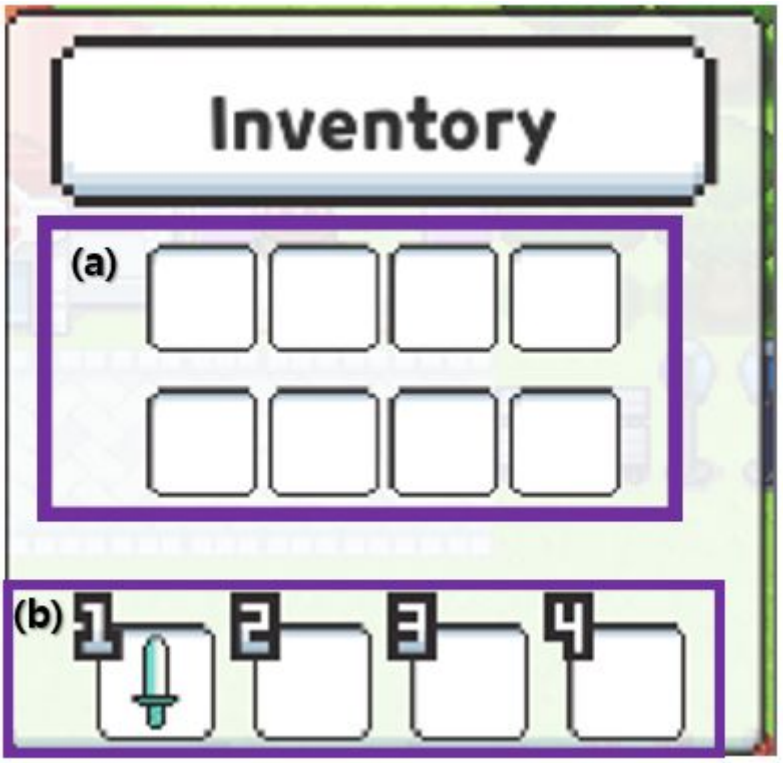
조작화면

	기능	설명
(a)	상태창	Player의 Hp, Mp, Coin, Exp, Level을 알려주는 창
(b)	위치	Player가 현재 있는 위치를 알려준다.
(c)	방향키	조이스틱 버튼으로 상하좌우로 Player를 움직인다.
(d)	Weapon slot	Player가 소유하고 있는 무기들을 저장하는 곳
(e)	무기교환	-S키는 Weapon slot의 무기를 교체해주는 키
	공격	-A키는 Player가 공격하는 키
(f)	상태차	Player의 공격력, 방어력, 스피드를 보여주는 곳
	인벤토리	Player가 소유하고 있는 아이템을 보여주는 곳
	메뉴	저장하기, 불러오기, 환경설정, 나가기

상태창

기능	설명
체력	Player의 체력
공격력	Player의 공격력
방어력	Player의 방어력
이동속도	Player의 이동속도

인벤토리



	기능	설명
(a)	Item slot	Unique Item을 구매/획득시 저장하는 곳
(b)	Weapon slot	무기를 획득시 저장하는 곳

메뉴

	기능	설명
(a)	저장하기	현상태의 게임을 저장한다
(b)	불러오기	저장된 데이터를 불러온다
(c)	나가기	시작화면으로 나간다
(d)	환경설정	환경설정으로 들어간다
(e)	메뉴 나가기	메뉴 창이 사라진다

환경설정

	기능	설명
(a)	효과음	무기 타격 효과음
(b)	배경음	맵의 배경음
(c)	음소거	-버튼을 클릭시 빈 박스의 경우는 음소거 -버튼을 클릭시 박스에 V가 있으면 소리가 나타난다
(d)	소리 크기 조절	소리 크기를 조절하는 장치
(e)	나가기	환경설정 창을 나간다

상점

상점	기능	설명
	상점 기능	Player가 필요한 아이템을 구매하는 곳
	특수 아이템	Unique 아이템을 구매하는 창
	포션 아이템	Portion 아이템을 구매하는 창

무기

무기	설명
	기본 무기
	360도 회전하면서 공격하는 대도
	긴 사거리로 적을 찌르는 창
	한방 데미지가 강하지만 공격속도가 느린 철퇴

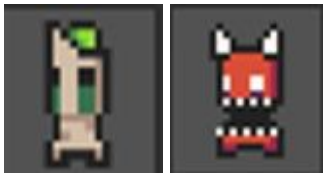
포션

포션 아이템	설명
	Hp를 회복시켜주는 포션
	Mp를 회복시켜주는 포션
	Exp를 증가시켜주는 포션

고유 아이템

고유 아이템	설명
	방어력 5를 증가시켜주는 갑옷
	기본 체력 게이지 100을 늘려주는 반지
	공격력 10을 증가시켜주는 장갑
	이동속도 2를 증가시켜주는 신발

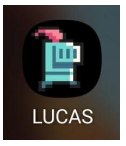
몬스터

근접 몬스터	
	Player가 범위안에 들어오면 Player를 추격해서 공격한다.
원거리 몬스터	
	Player가 범위안에 들어오면 총알을 발사해서 Player를 향해서 공격한다.
보스 몬스터	
	돌진, 몬스터 소환 및 근접 공격 위주로 공격하는 보스 몬스터
	돌진, 원거리 투사체 및 원거리 공격 위주로 공격하는 보스 몬스터

2.2 설치 및 사용방법

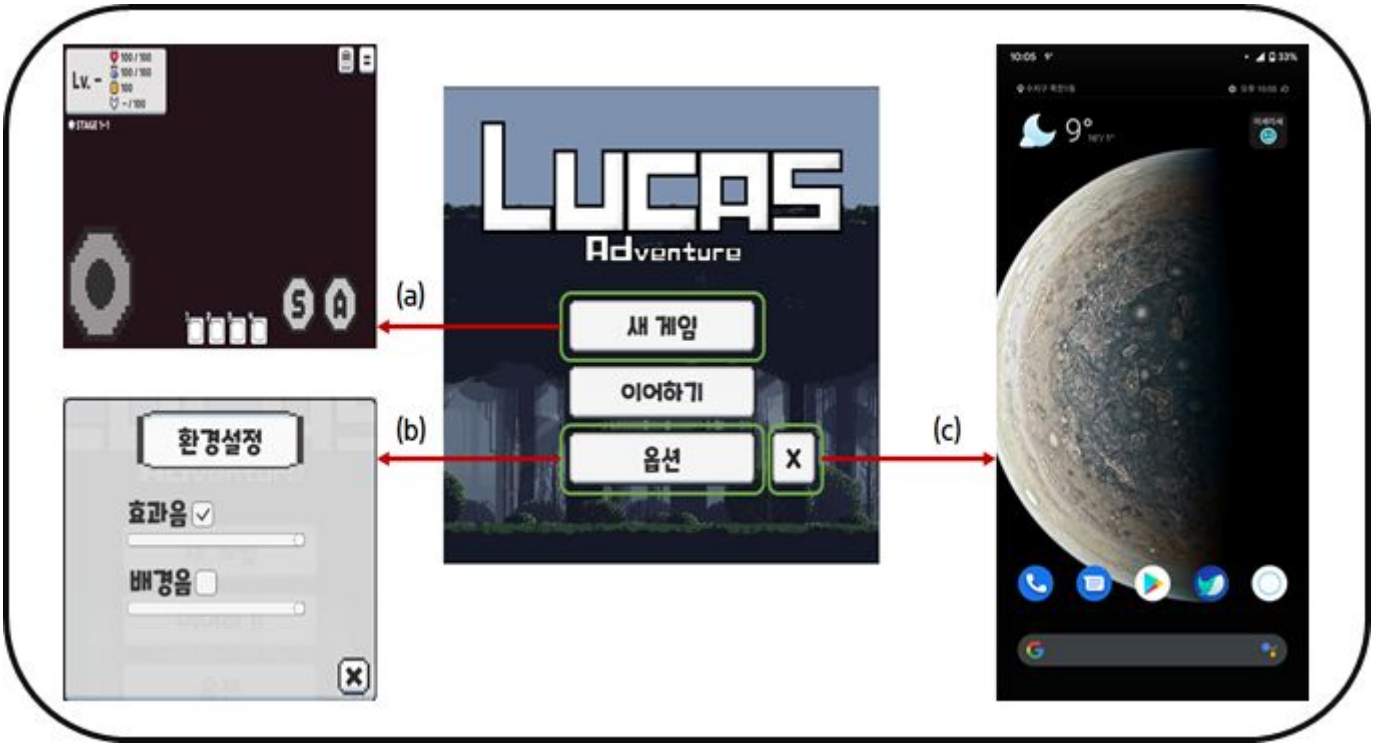
1. 링크에서 apk파일을 다운로드

   https://drive.google.com/drive/u/0/folders/1bc4nv2I0hiKiiRLDkDRXgsdXCCPjMphI



2. 게임 시작 버튼 클릭

환경설정(시작전에 설정, 게임 중에도 설정 가능)



3. 홈에서 시작하여 Area1으로 들어간다

4. Area1의 2개의 스테이지와 Boss스테이지를 클리어 한다.(단, Player가 게임 중간에 죽으면

Home으로 다시 돌아간다.)

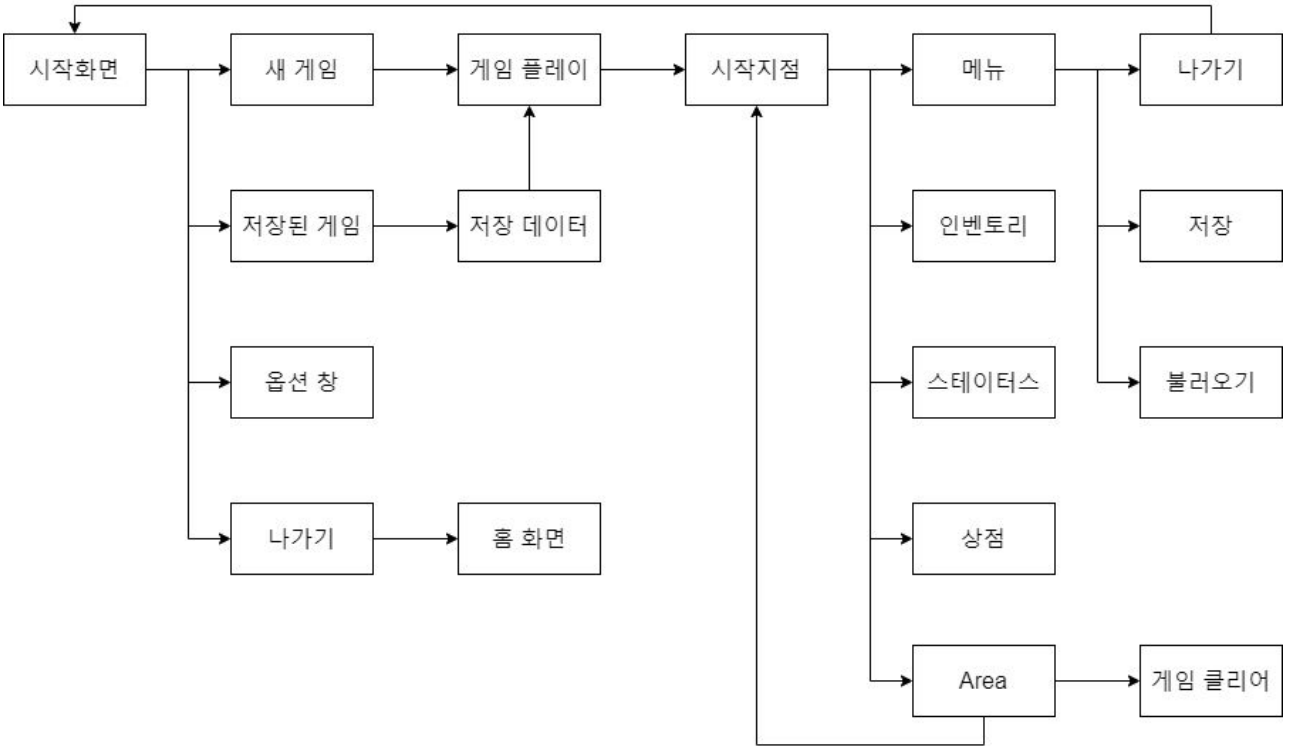
5. 홈으로 오면 필요에 따라 상점에서 개인 정비한다 .

6. Area1과 동일하게 Area2를 클리어 하면 된다

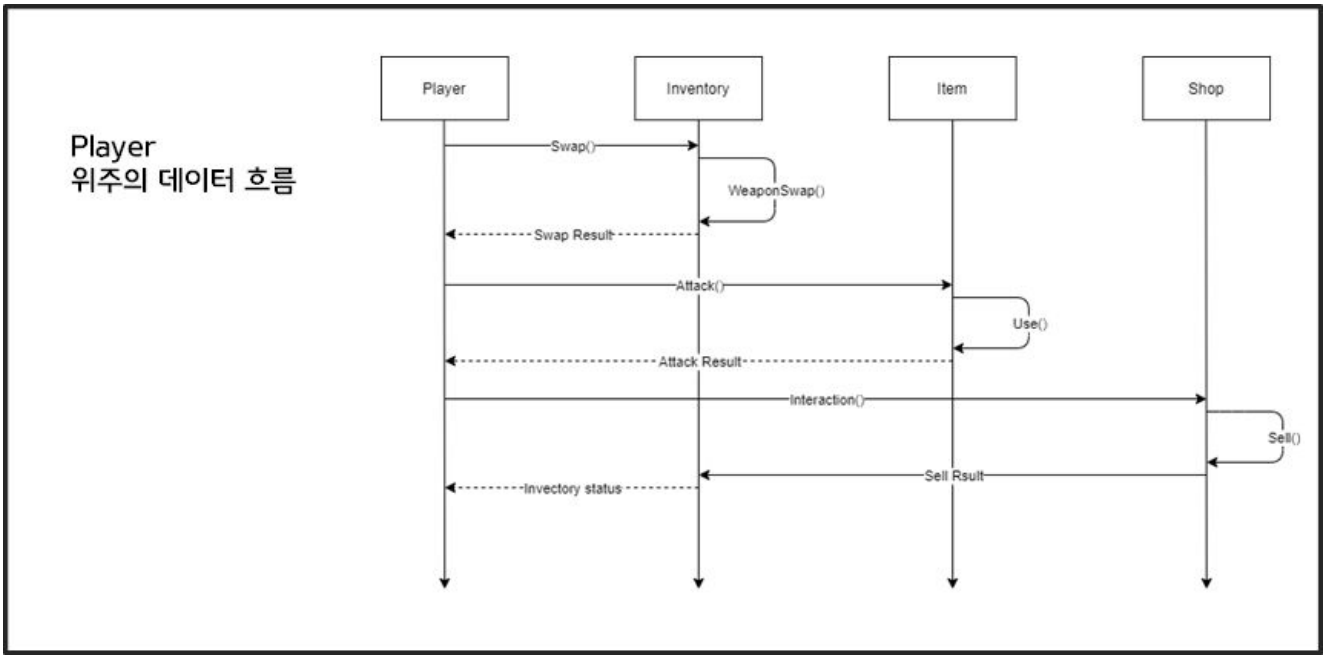
7. Area2를 클리어하고 홈으로 들어오면 게임 끝


Lv.9
212 / 280
100 / 100
260
157 / 214
STAGE home
Game Clear

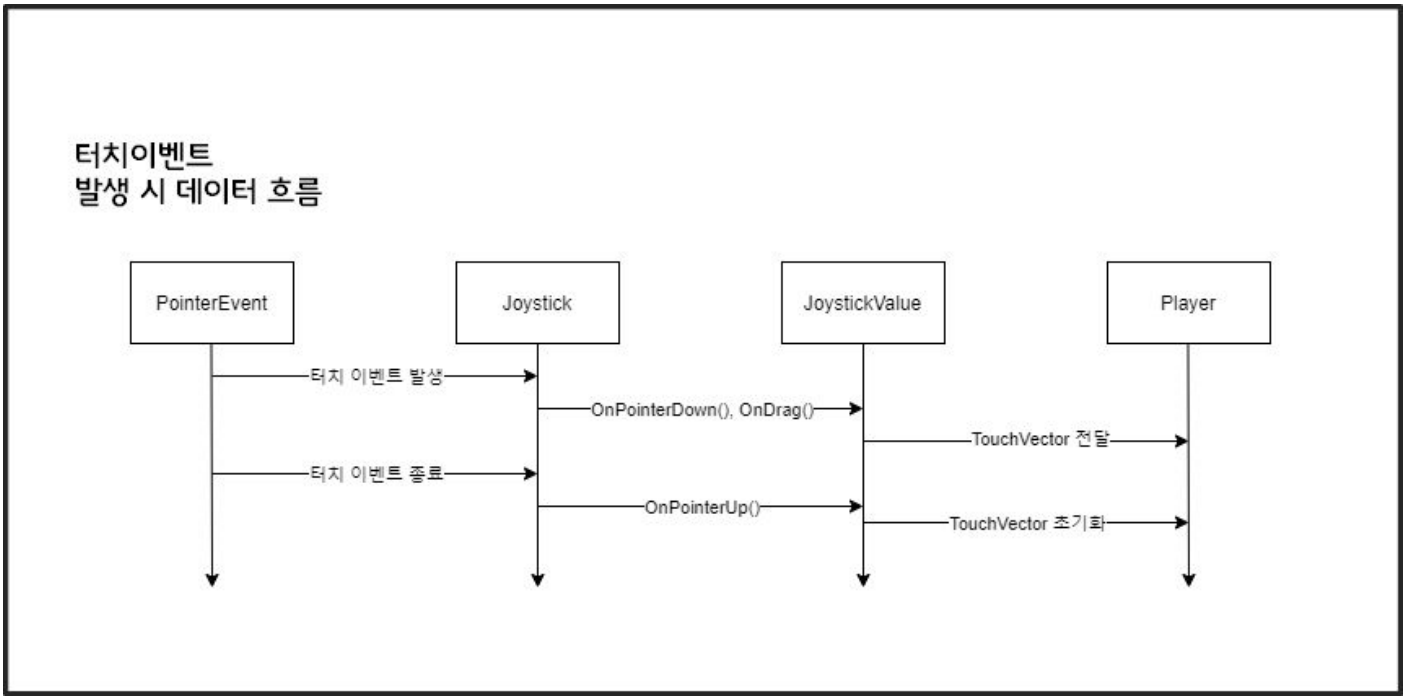
- **게임 진행 흐름도**


시작화면
새 게임
게임 플레이
시작지점
메뉴
나가기
저장된 게임
저장 데이터
인벤토리
저장
옵션 창
스테이터스
불러오기
나가기
홈 화면
상점
Area
게임 클리어

- 플레이어 무기 사용과 상점 이용 시나리오



- 터치 이벤트 발생 시나리오



3. 개발자 가이드

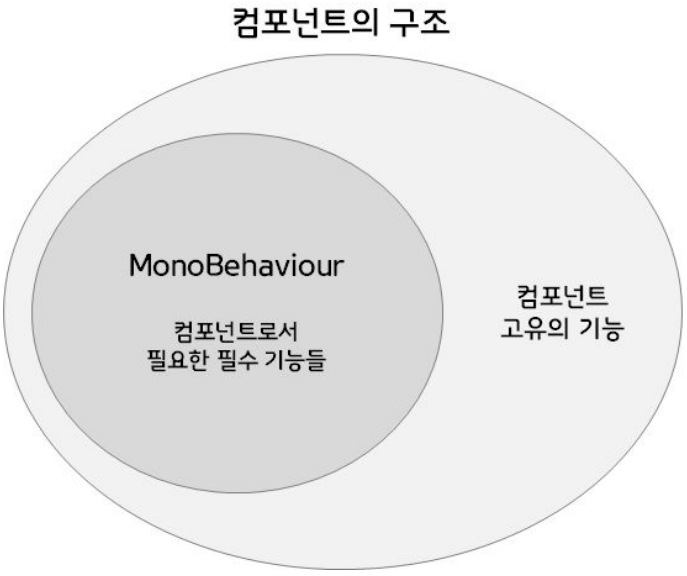
3.1 주요 데이터 및 알고리즘 명세

1) Unity 주요 아키텍처

유니티는 기본적으로 이미 완성된 API와 컴포넌트 등을 재사용하여 사용자가 원하는대로 게임 로직을 짤 수 있도록 도와주는 역할을 한다.

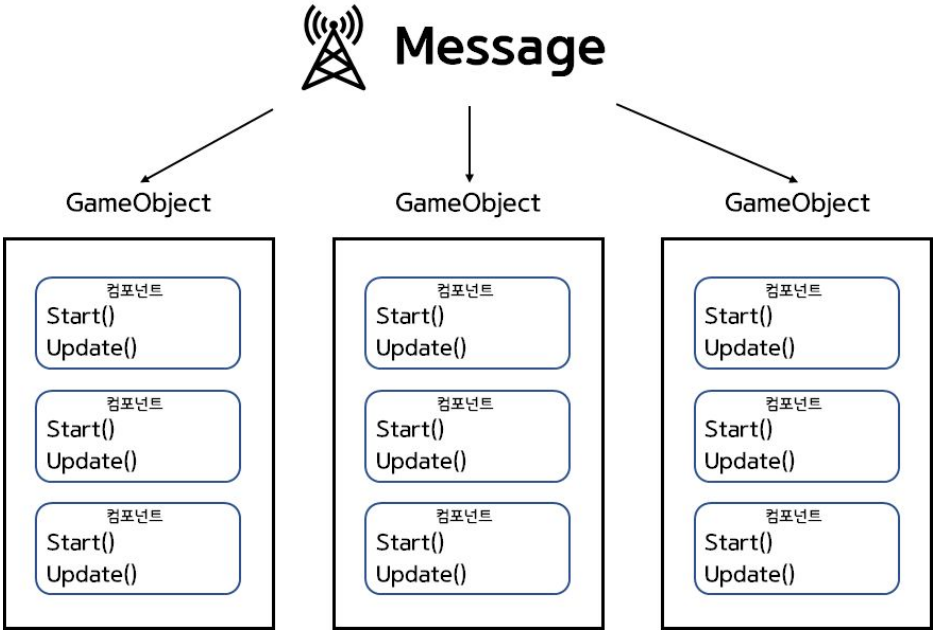
유니티가 정상적으로 게임 로직으로서 작동하는데에 MonoBehaviour와 Message System이 가장 중요한 역할을 한다.

- **MonoBehaviour**



유니티에서 기본적으로 지원하는 최상위에 위치한 클래스 중 하나이자 사용자가 생성하는 스크립트를 포함해서 모든 컴포넌트를 구성하는 클래스가 상속하는 클래스이다. MonoBehaviour 클래스와 상속되는 클래스들은 컴포넌트로서 게임 오브젝트에게 추가될 수 있게 되고, 그 이후로 유니티의 통제를 받게된다. 그리고 유니티 이벤트 메시지를 통해 감지할 수 있고 조작 가능하게 된다.

- **Message System**



유니티의 메시지 시스템은 3가지 특징을 가진다.

1) 보내는 쪽은 누가 받는지 신경쓰지 않는다.
2) 받는 쪽은 누가 보냈는지 신경 쓰지 않는다.
3) 메시지에 명시된 기능을 가지고 있으면 실행, 없으면 무시한다.

메시지는 브로드캐스팅 방식을 통해 지금 실행되는 씬에 존재하는 모든 오브젝트에게 무차별적으로 전부 전달되게 된다. 그렇게 되면 게임 오브젝트 내부의 컴포넌트에게 전달되어 해당 기능들을 실행하게 되는 것이다.

주로 게임오브젝트가 생성될때 필요한 초기작업을 실행하는 부분인 Start(), 지속적으로 현재 상태를 갱신하며 상호작용을 하도록해주는 Update(), 객체가 충돌했을때 특정 기능을 실행시키는 OnTrigger계열등 게임 로직의 작동을 직접적으로 관여하는 부분이 메시지 시스템이다.

메시지와 브로드 캐스팅은 오브젝트 간의 복잡한 참조 관계를 끊고 라이프사이클을 스스로 관리할 수 있도록 하는 역할을 한다.

2) 주요 기술

- 유니티 내부 API

유니티에는 기본적으로 UnityEngine에서 제공하는 수많은 API가 있지만 그 외에 따로 선언하여 사용한 API와 중요 기능을 담당한 API위주로 설명 하겠다.

1) UnityEngine.EventSystems

EventSystems는 사용자가 모바일 환경에서 터치를 통한 조작을 할 수 있도록 도와주는 API이다. 이벤트 시스템은 키보드, 마우스, 터치, 커스텀 입력등 입력 기반의 어플리케이션 오브젝트에 이벤트를 전송하는 방법으로, 이벤트 전송에 함께 작용하는 일부 컴포넌트로 구성된다.

이벤트 시스템의 주요 역할은 다음과 같다.

- 어떤 게임 오브젝트를 선택할지 관리
- 어떤 입력 모듈을 사용할지 관리
- 필요에 따라 레이캐스팅 관리
- 필요에 따라 모든 입력 모듈 업데이트

여기서 레이캐스팅이란 레이캐스터를 통해 수행하는 작업을 뜻하며, 레이캐스터는 이벤트가 발생하는 포인터를 가리키는 것에 사용된다. 목표로 하고 있는 플랫폼인 모바일은 포인팅 디바이스이기 때문에 그 위에 무엇이 있는지 산출하고, 씬 내부과 상호작용할 수 있도록하는 API이다.

2) UnityEngine.SceneManagement

SceneManagement는 개발자가 직접 제작한 Scene들을 오갈 수 있도록 만들어 주는 API이다. 씬 매니지먼트는 유니티 에디터에 등록된 씬에 한해서 접근할 수 있도록하며, 씬 매지니먼트에서 지원하는 SceneManager를 통해 씬에 접근하여 불러올 수 있다. 유니티에서 씬 이동은 기존 씬의 파괴와 다음 씬의 초기화이기 때문에 씬을 이동할 경우 유지시켜야하는 오브젝트가 있다면 스크립트 레벨에서 처리를 해야한다.

3) PlayerPrefs

PlayerPrefs는 유니티에서 제공해주는 데이터 관리 클래스이다. 해당 클래스는 int, float, string, bool 타입의 변수를 저장하고 로드하는 기능을 제공한다. 하지만 기본적으로 제공하는 위의 4가지 타입을 제외하고 사용자가 만든 클래스나 컴포넌트의 상태같은 것은 저장을 지원하지 않기 때문에 사용하기에 따라 적절한 변환이 필요하다. PlayerPrefs를 사용하면 어플리케이션을 종료시키더라도 해당파일은 데이터에 존재하기 때문에 이어서 할 수 있기 때문에 해당 프로젝트의 저장/불러오기 기능, 옵션의 상태 저장 등에 사용되었다.

- **유니티 외부 API**

개발에 있어서 거의 모든 기능은 유니티에서 지원하는 내부 API들로 구성했지만, 일부 오브젝트의 이동부분을 조금 더 효과적으로, 편리하게 표현하기위한 Dotween API를 사용하였다.

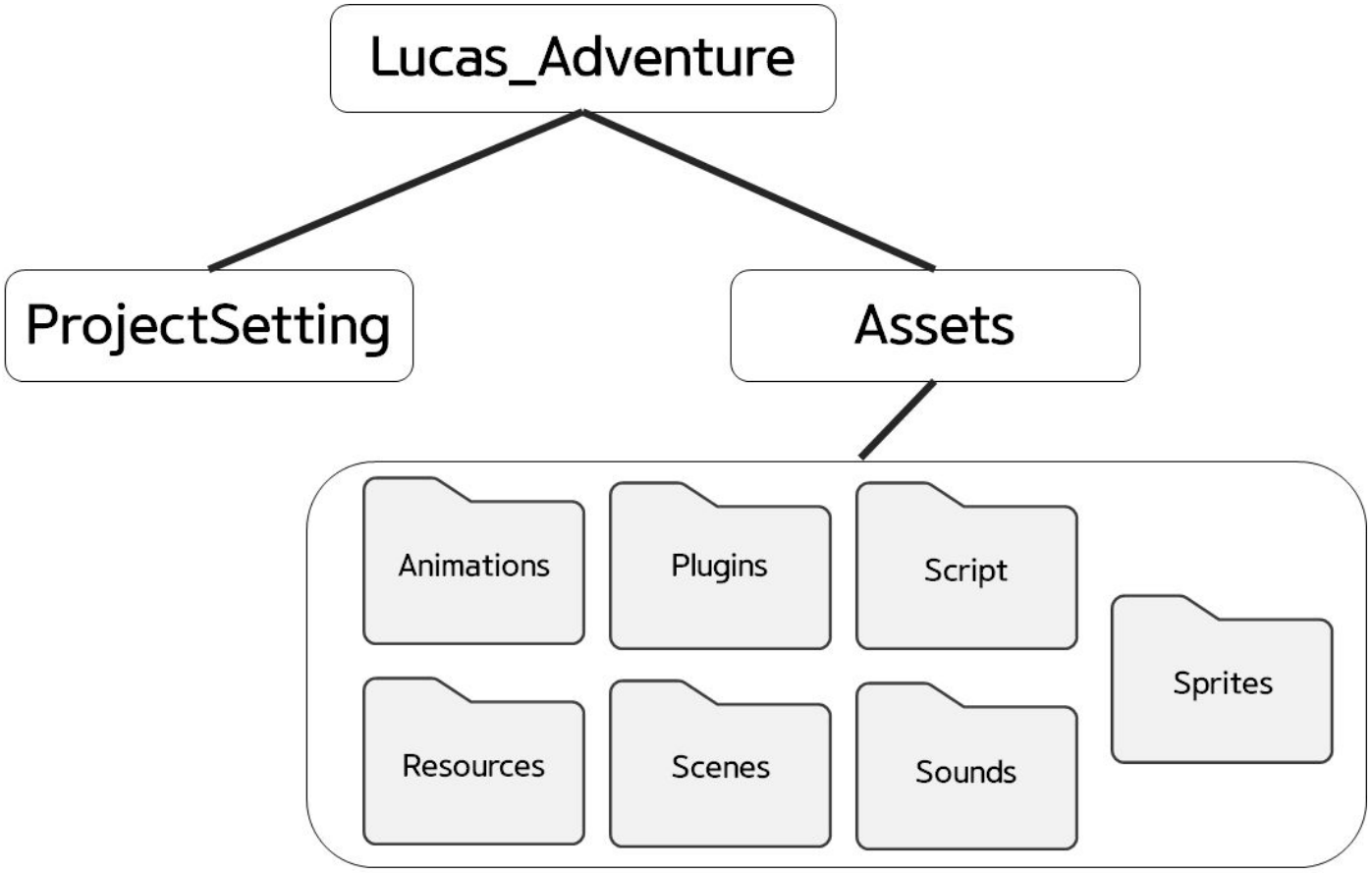
1) **Dotween**

닷트윈은 유니티 상에서 구현하는 트위닝을 조금 더 간단하고 효과적으로 표현하기 위해 사용되었다. 트위닝이란 오브젝트의 시간당 변화를 의미하기도 하는데, 기존에는 스크립트를 통해 직접 제어를 했어야했다. 하지만 닷트윈을 사용하면 스크립트의 제어 부분을 따로 미리 구현해 줌으로써 사용자가 함수를 호출하여 값을 변경해주는 것만으로 간편하게 트위닝을 제어할 수 있게 된다.

3.2 소스코드 구조

GitHub Link : https://github.com/hyeonjun414/Lucas-Adventure

1) 소스코드 디렉토리 설명



Lucas_Adventure 내부에 직접 개발하며 사용한 두 개의 디렉토리가 존재한다. Packages 디렉토리도 존재하지만 유니티에 의해 자동 생성되는 폴더이므로 제외했다.

1.1) Project Setting

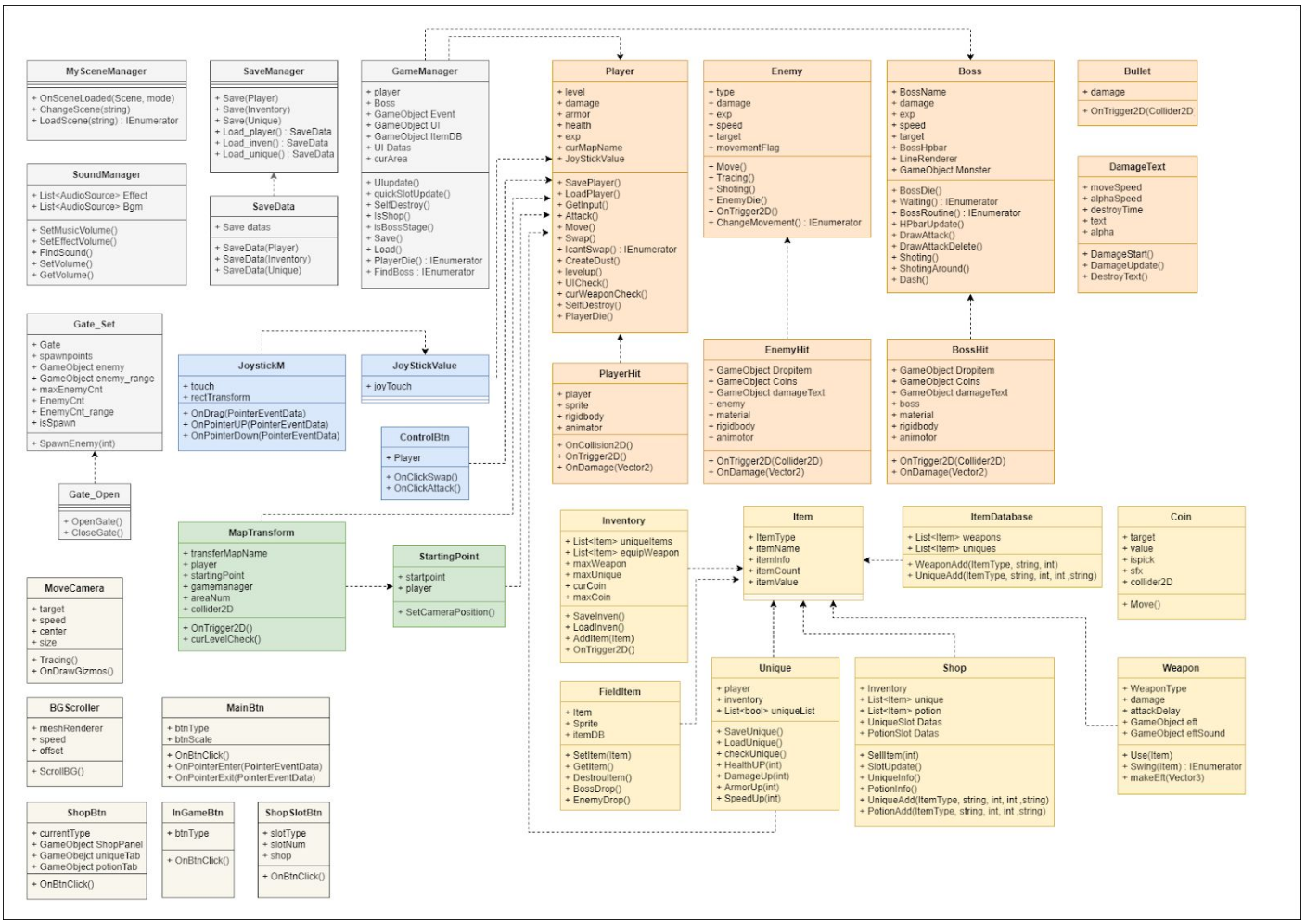
유니티 에디터를 사용하면서 변경되는 각종 설정 파일들이 모여있는 디렉토리이다. 모바일 환경으로 만들기 위해 변경하는 빌드 정보, 각 오브젝트에 적용되는 태그 정보, 개발자에 맞춰 설정이 가능한 에디터 설정 등에 대한 설정 파일이 있다.

1.2) Assets

유니티를 통해 개발하면서 직접 관여하여 스크립트 파일을 생성하고 외부의 그래픽이나 사운드 같은 에셋 요소를 포함하는 중요한 디렉토리이다. 하위 디렉토리는 개발자가 개발하며 분류를 용이하기 위해 용도에 맞게 분류할 수 있다.

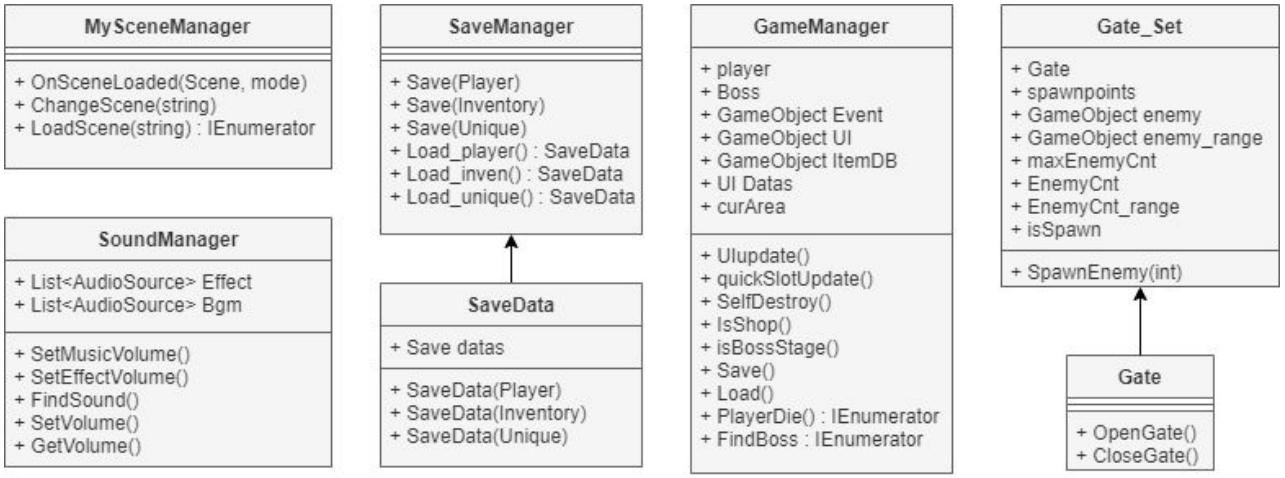
- **Animations** : 움직일 수 있는 오브젝트에 대한 애니메이션 정보가 들어있다. 애니메이션은 이미지 단위로 구성되며 프레임에 맞게 작동하는 정보가 담겨있다. 애니메이션을 실행하는 Animation Controller와 이미지 정보가 담겨있는 Animation 파일로 나눠진다.
- **Plugins** : 개발하면서 사용되는 유니티에서 기본적으로 제공안하는 기능이나 글씨체 같은 외부 요소의 사용을 분류하기 위해 만들어진 디렉토리이다. 외부 API의 정보도 존재한다.
- **Scripts** : 오브젝트 또는 게임 환경에 적용되는 스크립트 파일을 모아놓은 디렉토리이다. 개발자의 의도에 따라 용도에 맞는 하위 디렉토리를 나누어 스크립트를 관리하는 것이 용이하다.
- **Resources** : 유니티에는 프리팹이라는 개념이 존재한다. 씬에 만들어 놓은 오브젝트를 일정한 형태로 저장하여 재사용 가능하도록 만들 수 있는 것이 바로 프리팹이다. 이렇게 생성된 오브젝트 파일을 모아놓은 곳이 리소스 디렉토리이다. 개발은 하면서 프리팹으로 만들어놓는 형태를 가진 오브젝트로는 아이템, 몬스터 등이 존재한다.
- **Scenes** : 오브젝트를 통해 구성되는 화면을 Scene이라고 한다. 이러한 Scene을 관리하기 위해 만들어진 디렉토리이다.
- **Sounds** : 게임에서 상호작용을 통해 재생되는 효과음이나, 일정하게 Scene에서 흘러나오는 배경음의 오디오소스를 모아놓은 디렉토리이다.
- **Sprites** : 해당 프로젝트는 3d가 아닌 2d를 기반으로 제작되었다. 2d 기반의 그래픽을 표현하기 위해서는 이미지 파일이 필요한데 이러한 이미지 파일을 Sprite라고 부른다. 게임의 화면 형성을 위해 꼭 필요한 요소이다. 하위 디렉토리로 캐릭터, 배경 등으로 나누어 이미지간의 특징을 분류 할 수도 있다.

2) 프로젝트 클래스 다이어그램 구조



해당 프로젝트의 클래스는 크게 System, ActiveObject, MapTransform, Control, Item, UI로 나뉘어져 있다.

- System



게임의 흐름과 사용 환경을 조절하는 클래스로 사운드, 저장/불러오기등의 기능이 있다.

1) GameManager

게임매니저는 전체적인 게임의 흐름을 담당하고 있고, 각 씬마다 존재하는 오브젝트 간의 연결을 도와 주는 역할을 한다.

- PlayerDie() : 플레이어 캐릭터가 사망한 경우 실행된다. 죽는 시점 까지 진행된 플레이어의 정보를 초기화 시키고 시작 지점으로 돌아간다. (처음부터 다시 시작)
- isBossStage() : 현재 플레이어가 존재하는 씬이 보스스테이지라고 감지되면 씬에 존재하는 보스 정보를 게임 매니저에 추가시키는 기능을 한다.
- Save() : 게임 매니저에 연결되어있는 플레이어 정보와 instance를 통해 접근 가능한 Inventory와 Unique 정보를 바이너리 직렬화를 통해 정보를 저장한다.
- Load() : Save에 의해 저장된 바이너리 파일을 불러와 저장된 파일로 각 오브젝트의 정보를 새롭게 덮어 씌운다.

2) SoundManager

사운드 매니저는 사운드 옵션 구현을 위해 만들어 졌다. 옵션UI의 슬라이더와 토글 버튼에 연결되어 모든 오디오 소스에 대한 볼륨값 조절과 음소거 유무를 결정한다.

- FindSound() : 씬의 이동이 생길때 마다 실행되면 기존의 오디오 소스를 초기화하고 새로운 오디오 소스를 불러온 후에 태그를 비교하여 효과음과 배경음을 구분하여 저장한다.
- SetMusicVolume(), SetMusicVolume() : UI와 연결된 슬라이더와 토글 버튼의 값을 받아와 현재 배경음에 들어가있는 오디오 소스의 음량과 음소거 유무를 결정한다.
- SetVolume(): 현재 설정되어 있는 값을 메모리상에 저장한다. 매 Update마다 실시간으로 실행된다.
- GetVolume(): SetVolume에 의해 저장된 값을 불러와 초기값을 저장된 값으로 바꿔준다. 새로운 사운드 매니저가 생성될때 실행된다.

3) MySceneManager

씬 매니저는 씬과 씬 사이의 이동이 발생할 때 씬 이동 처리와 씬 간의 이동이 발생할때의 페이드아웃과 페이드인 처리를 담당한다.

- OnSceneLoaded(Scene, mode) : 씬 로드가 이루어진 이후 실행되는 기능으로 이동된 씬에서 페이드 인 효과를 준다.
- ChangeScene(string) : 이동되는 씬 이름(string)을 받아 LoadScene 코루틴을 실행하는 역할을 한다.
- LoadScene(string) : ChangeScene(string)에서 실행되며 string 정보를 이어받아 비동기식으로 다음 씬을 불러온다. 비동기식으로 불러옴으로 페이드 아웃과 페이드 인 작동 시간을 확보한다.

4) SaveManager, SaveData

SaveManager는 실질적인 바이너리 직렬화를 통한 데이터 저장을 담당한다. 클래스 내의 모든 함수는 각각 저장을 필요로 하는 클래스에서 호출되어 사용된다. 불러오기를 할때에는 파일에서 읽어온 데이터를 각각의 클래스에 맞는 SaveData 형으로 반환한다.

SaveData 클래스는 저장 기능을 지원하는 각각의 클래스에서 저장되어야하는 변수를 모아놓은 클래스이다. 데이터의 관리를 좀더 수월하게 만들어준다. 각 SaveData 생성자는 사용되는 클래스 타입을 변수로 받아 해당 객체의 현재 변수를 저장하는 역할을 한다.

- Save(class) : 바이너리 직렬화를 담당하는 부분으로 바이너리 형식으로 특정 경로에 파일을 만들고 생성된 SaveData파일을 직렬화 방식으로 입력하고 파일을 저장한다.
- Load_class() : 경로에 저장되어 있던 파일을 열고 파일이 존재하면 SaveData 형에 파일 정보를 입력하여 데이터 파일을 반환한다.

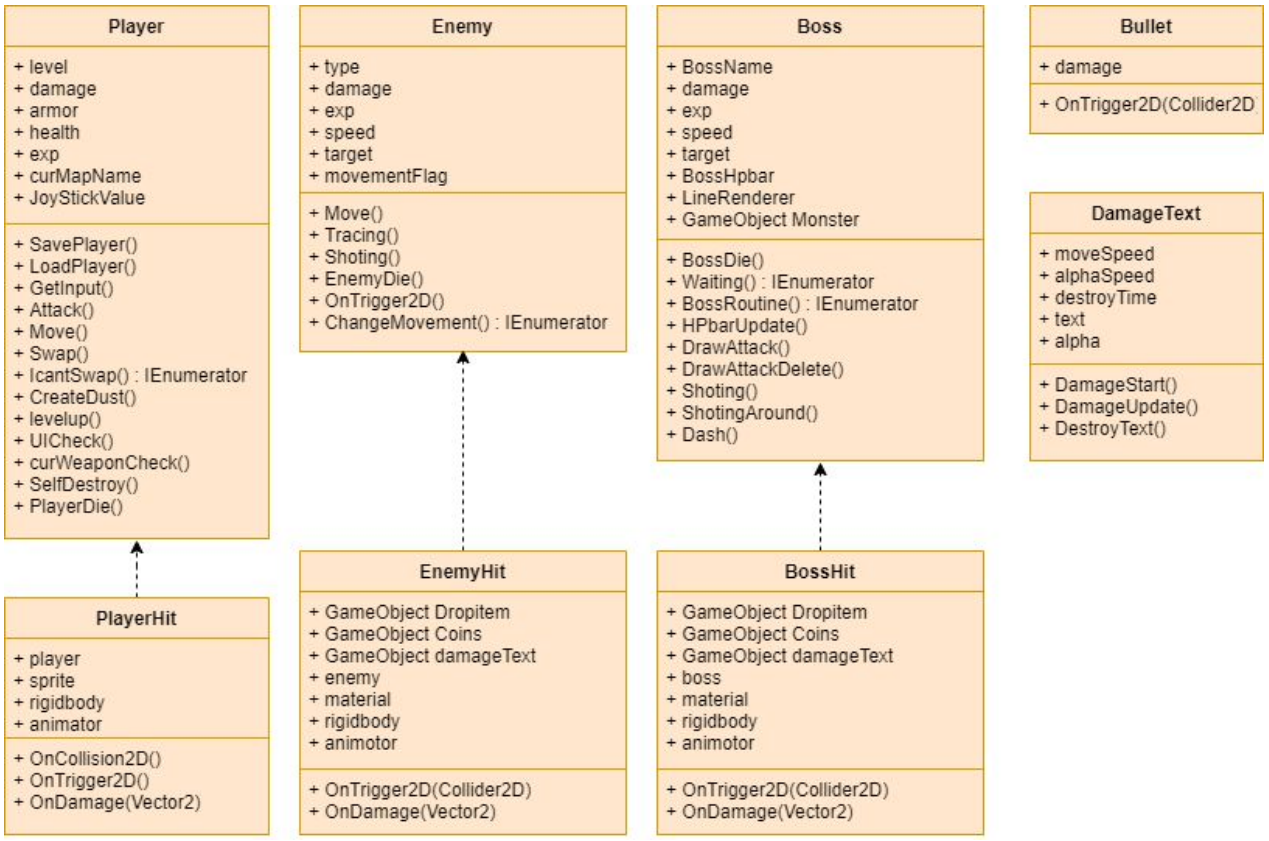
5) Gate_Open, Gate_Set

Gate_Set은 Gate_Open 클래스를 관리하면서 몬스터의 생성과 스테이지 클리어 제약 조건을 판단한다. 각 스테이지 마다 GameObject형태로 배치되며 미리 생성된 몬스터 Prefab 파일을 등록하여 몬스터 생성에 사용한다.

Gate_Open은 스테이지의 클리어 제약조건이 해제되면 Gate_Set에 의해 함수가 실행되며 맵이동을 방해하는 장애물의 활성화/비활성화를 설정하는 클래스이다.

- SpawnEnemy(int) : 설정된 스폰포인트 위치의 인덱스를 변수로 받아 해당 위치에 몬스터를 생성한다. 생성마다 EnemyCnt를 증가시키고 미리 지정된 몬스터 오브젝트를 생성한다. 생성된 몬스터는 제거될때 Gate_Set을 참조하여 EnemyCnt를 줄이고 0이 되었을때 Gate_Open의 함수를 호출해 게이트를 열고 닫는다.

- **ActiveObject**



ActiveObject는 객체중심적인 이동처리가 가능한 오브젝트를 담당하는 클래스 집합이다.

1) Player(PlayerHit)

Player 클래스는 게임을 플레이하는 플레이어가 조작하는 캐릭터를 담당하는 클래스로, 이동-공격-스왑 등 플레이어가 입력을 받아 사용되는 클래스이다.

PlayerHit 클래스는 캐릭터의 피격을 담당하는 클래스로 충돌에 의한 피격 처리와 피격으로 발생하는 밀려남 효과, 일시적인 캐릭터 색 변경 등 추가적인 처리를 해준다.

2) Enemy, Boss(EnemyHit, BossHit)

Enemy 클래스와 Boss 클래스는 비슷한 부분이 상당 수 존재한다. 각각 일반몬스터의 조작과 보스몬스터의 조작을 담당한다. 플레이어의 위치를 파악하는 Tracing을 통해 근접몬스터는 다가가 공격을 할 수 있고, 원거리몬스터는 발사체를 발사하여 공격한다.

보스몬스터는 추적을 하여 근접하여 공격하는 것이 아닌 특유의 패턴을 가지고 플레이어 위치정보를 참고하여 공격을 실행한다.

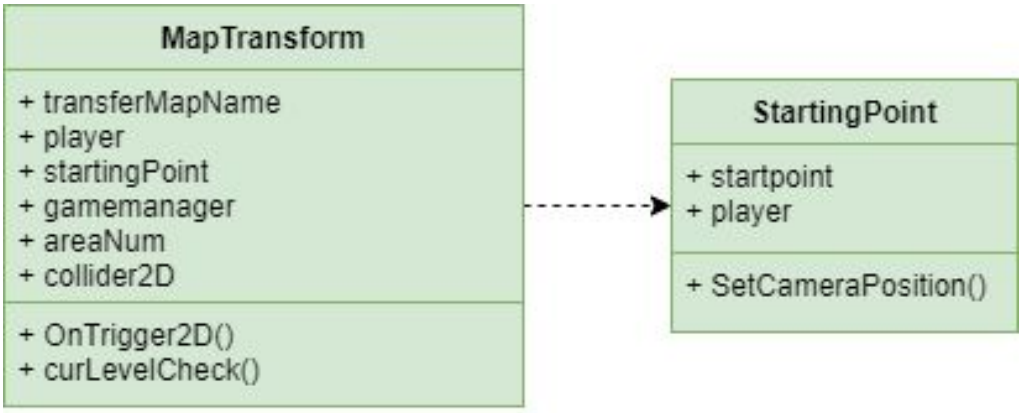
EnemyHit 클래스와 BossHit 클래스는 각각의 피격처리를 하는 클래스이다. PlayerHit 클래스와 다르게 아이템드랍 부분이 추가된다. Get_Set 클래스를 인스턴스로 접근해 EnemyCnt의 조절해도 관여한다.

3) Bullet, DamageText

Bullet 클래스는 발사체에 대한 정보를 담고 있는 클래스이다. 발사체의 공격력 정보가 들어있고, 탄환의 기준으로 한 충동처리를 실행한다. 특정한 오브젝트와 충돌하면 사라지도록 만들어져있다.

DamageText 클래스는 캐릭터가 몬스터에게 무기를 통해 공격을 가했을때의 공격력 정보를 시각화해주는 클래스이다.

- MapTransform



인 게임 내에서 캐릭터와의 상호작용으로 MySceneManager를 사용해 현재 씬을 정해진 씬으로 불러오는 역할을 한다.

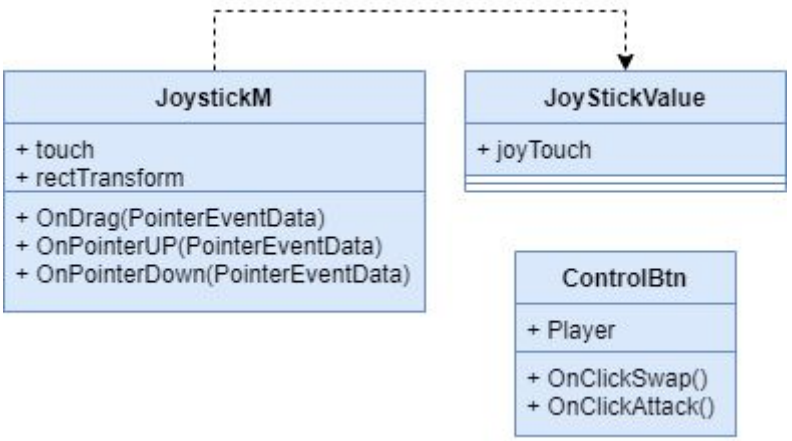
1) MapTransform

씬과 씬 사이에서 충돌 이벤트를 통해 게임의 현재 스테이지 정보를 갱신하고 transferMapName과 일치하는 새로운 씬을 불러오는기능을 한다. 맵과 맵사이를 이어주는 포탈 오브젝트에 삽입되며, 특히 home 씬에 존재하는 포탈 오브젝트는 GameManager의 현재 진행되는 에어리어 정보를 참조하여 활성/비활성 유무를 판단한다.

2) StartingPoint

캐릭터가 다음 맵으로 이동할 때 캐릭터가 생성되는 위치를 가리키는 오브젝트를 형성하는 클래스이다. 현재 캐릭터의 위치정보를 StartingPoint 오브젝트의 위치 정보로 교체하고 SetCameraPosition()을 사용하여 새로운 씬에 존재하는 카메라의 위치를 캐릭터에 맞춰서 위치를 변경해주는 역할을 한다.

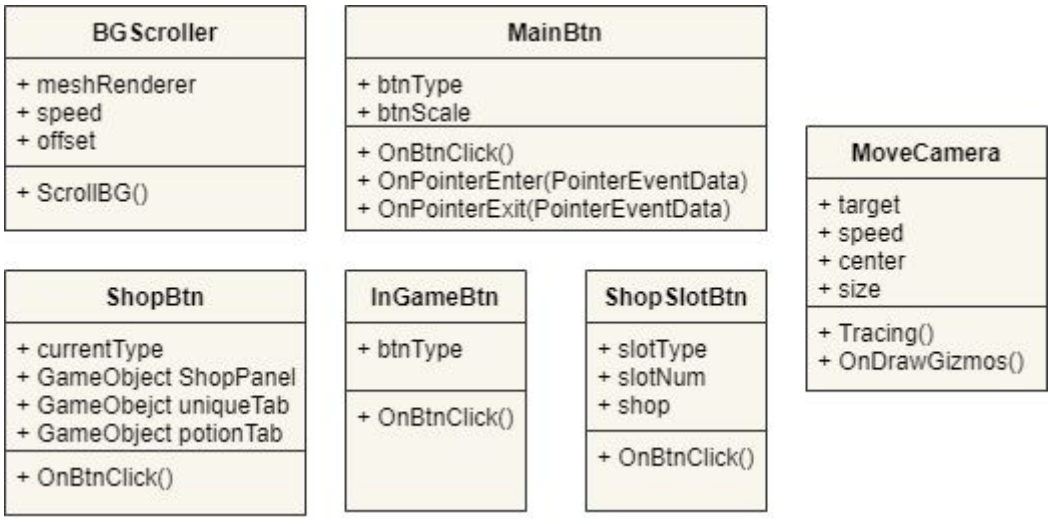
- Control



모바일 환경을 지원하기위해 터치 기반의 조작을 담당하는 클래스를 만들어 주었다. JoystickM 클래스 내부에서 해당 오브젝트 좌표와 실제로 터치되는 좌표를 계산하여 벡터 값을 만들어주고 값을 JoystickValue를 통해 플레이어 오브젝트에 전달한다. 전달된 벡터 값은 Player의 move() 함수를 통해 이동 기능을 수행한다.

ControlBtn 클래스도 모바일 환경을 지원하기 위한 버튼 인터페이스에 사용되는 클래스이다. 각 역할에 맞는 버튼을 클릭하면 연결되어 있는 함수를 실행하여 공격, 교체 기능을 실행하게 된다.

- UI



UI 관련 클래스들은 게임 내부에서 플레이어와의 상호작용을 통해 실행되는 기능에 대한 인터페이스의 연동 또는 오브젝트의 변화 시각적으로 보이는 효과를 처리하는 부분을 담당한다.

1) Btn (버튼 관련 클래스들)

직접 사용자가 터치를 통해 접근하는 버튼 오브젝트에 대한 기능의 추가와 버튼이 사용되는 씬의 종류에 따라 기능을 소분류 해주었다.

MainBtn 클래스는 시작화면에 대한 버튼 이벤트, InGameBtn 클래스는 인 게임 화면에 대한 버튼 이벤트 그리고 ShopBtn 클래스와 ShopSlotBtn 클래스는 인터페이스와 상점 객체의 연결과 상점 내부 인터페이스의 조작을 다룬다.

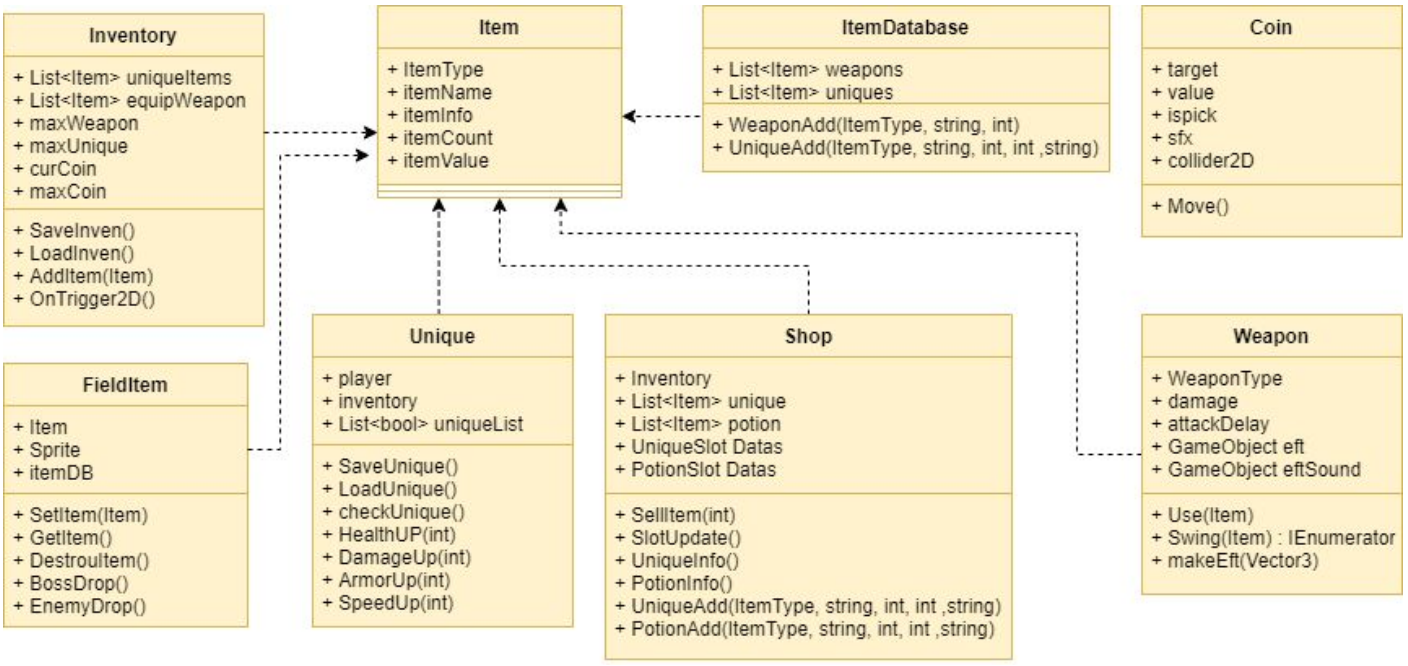
2) BGScroller

BGScroller 클래스는 백그라운드 스크롤러로 시작화면에서 배경에 대한 효과를 담당한다. 시간의 지남에 따라 연속되는 이미지를 횡이동시키는 기능을 한다.

3) MoveCamera

화면을 비추는 카메라 오브젝트에 대한 추가적인 스크립트로 캐릭터를 추적하는 역할을 하여 화면이 캐릭터를 중심으로 구성되도록 한다. 카메라의 이동 범위를 설정해줘서 화면 밖으로 과하게 넘어가지 않도록 하였다.

- Item



게임 내에서 사용되는 무기 아이템이나 고유 아이템, 아이템정보를 관리하는 데이터베이스, 아이템의 판매를 하는 상점들의 기능을 지원하는 클래스이다.

1) **Item** : 기본적인 아이템을 구성하는 클래스로 아이템 종류, 아이템 이름 등의 기본정보를 담고있다.
2) **Weapon** : 무기 오브젝트 형성에 사용되는 클래스로 기본적으로 근접무기의 사용이 구현 되어있다. 플레이어에서 무기 오브젝트를 참고하여 Use()를 호출하면 함수 내부에서 무기 타입을 체크하여 타입에 맞는 함수를 통해 공격을 사용한다.
3) **Inventory** : 상점이나 몬스터 사냥을 통해 얻은 아이템과 코인을 저장하는 클래스로 플레이어의 하위 오브젝트에 위치한다. 플레이어는 Inventory 클래스를 참조하여 무기 정보에 접근할 수 있다.

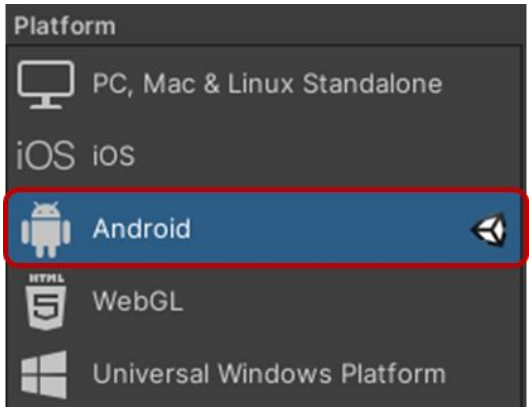
아이템의 추가는 드랍된 FieldItem과 충돌이벤트가 발생하면 AddItem(Item)을 통해 아이템 정보를 받아와 타입에 맞게 리스트에 추가하는 방식이다. 바이너리 직렬화를 통한 저장/불러오기를 지원한다. 무기 아이템 리스트, 고유 아이템 리스트, 현재 소지중인 코인 정보를 가지고 있다.

4) **FieldItem** : ItemDatabase를 참조하여 랜덤한 아이템을 생성한다. FieldItem을 생성하는 주체가 일반 몬스터인지 보스몬스터인지에 따라 무기아이템 생성과, 고유아이템 생성으로 나뉜다.
5) **ItemDatabase** : 게임에서 사용되는 모든 아이템 정보가 담겨있는 클래스이다. 두 개의 리스트를 통해 무기아이템과 고유아이템 정보를 만들어낸다.
6) **Unique** : 플레이어가 소지중인 고유아이템을 체크하여 일치하는 기능을 플레이어 상태에 추가하는 역할을 한다. Unique 클래스 내부의 함수로는 적용되는 고유 아이템에 대한 기능별 함수가 존재한다.
7) **Shop** : 플레이어가 가지고 있는 재화를 통해 고유 아이템과 포션 아이템을 구매할 수 있는 상점의 기능을 지원하는 클래스이다. 포션 아이템에 대한 효과는 Shop 클래스 내부에서 플레이어의 상태를 확인하고 조건에 충족하면 실행되는 방식으로 이루어져 있다.
8) **Coin** : 상점에서 물건을 구입하는데 사용되는 재화로, 몬스터 사냥이 드랍되는 요소 중 하나이다. 코인 오브젝트의 움직임을 자연스럽게 하기위해 Dotween을 통해 트위닝 기술이 적용되었다.

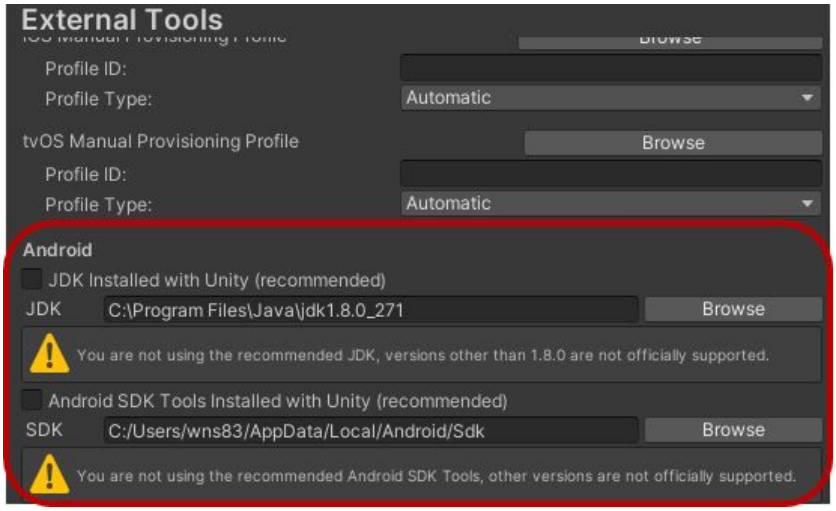
3.3 빌드 방법

개발환경으로 사용한 Unity는 기본적으로 빌드 부분에 있어서 자동화가 되어있는 부분이 많다. 개발자는 자동화 되어 있는 부분에서 프로그램 명, 프로그램 아이콘 등 일부 항목에 대해서만 수정을 하면 나머지 빌드 작업은 Unity에서 처리해준다. 모바일 환경으로의 빌드를 위해서 아래의 절차를 진행한다.

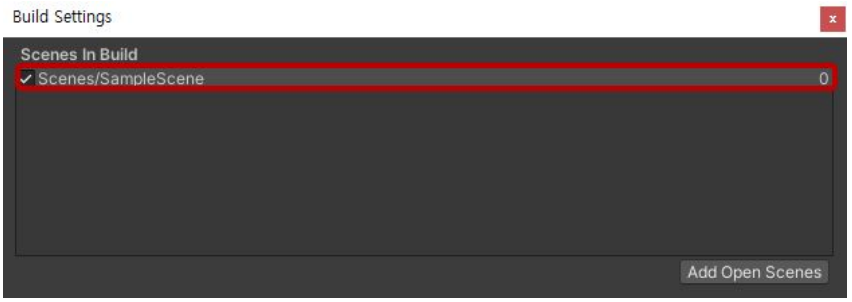
1) 사용중인 개발 툴인 Unity의 Build Setting에서 Mobile Platform으로 전환 한다.



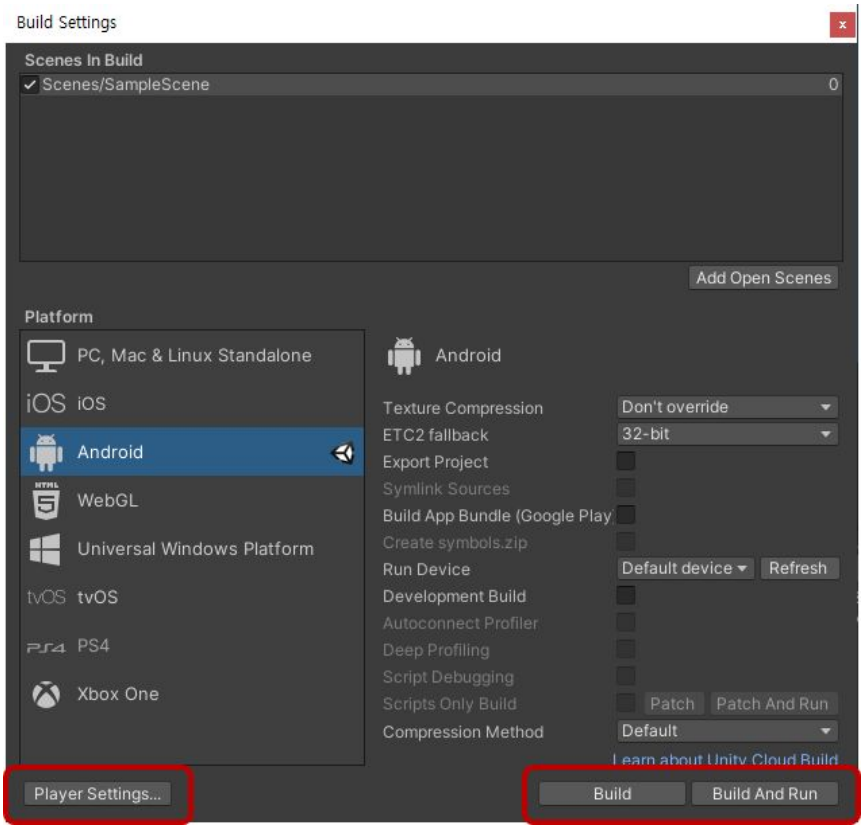
2) Preferences의 External Tools에서 모바일 환경 구성에 필요한 JDK, SDK를 버전에 맞게 설치해준다.



3) **Build Setting에 실행에 사용되는 Scene을 추가해준다.**

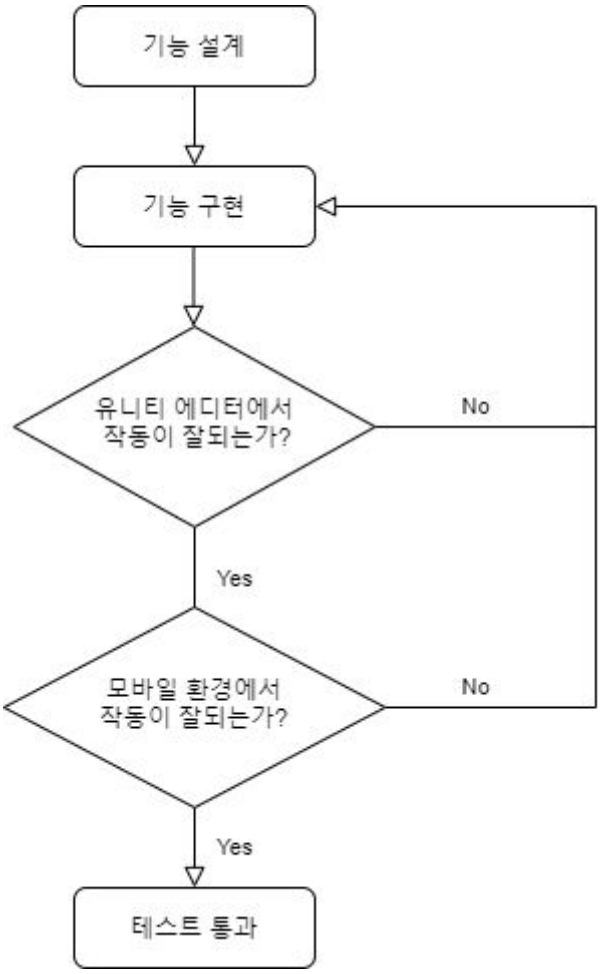


4) **Player Setting에서 어플의 이름, 아이콘등을 수정해준뒤 Build 또는 Build and Run 버튼을 클릭한다.**



3.4 테스트 방법

개발중인 기능이 정상적으로 작동되는지 알아보기 위해 단계 별로 테스트를 진행한다.



1) **기능 설계** : 게임에 적용하고자하는 기능의 전체적인 구조를 만든다.
2) **기능 구현** : 설계한 내용을 바탕으로 작동가능한 코드를 작성하여 테스트를 시작한다.
3) **유니티 에디터 테스트** : 유니티 에디터 상에서 구현한 기능에 대한 테스트를 진행한다. 오류 또는 버그 발견시 기능 구현 단계로 돌아가 코드를 수정한다.
4) **모바일 테스트** : 유니티 에디터에서 모바일 환경에 맞춰 빌드한 후에 모바일에서 기능의 테스트를 진행한다. 유니티 에디터의 결과와 일치하면 해당 기능에 대한 테스트가 끝난다. 오류 또는 버그 발견시 기능 구현 단계로 돌아가 코드를 수정한다.

4. 개발 내용

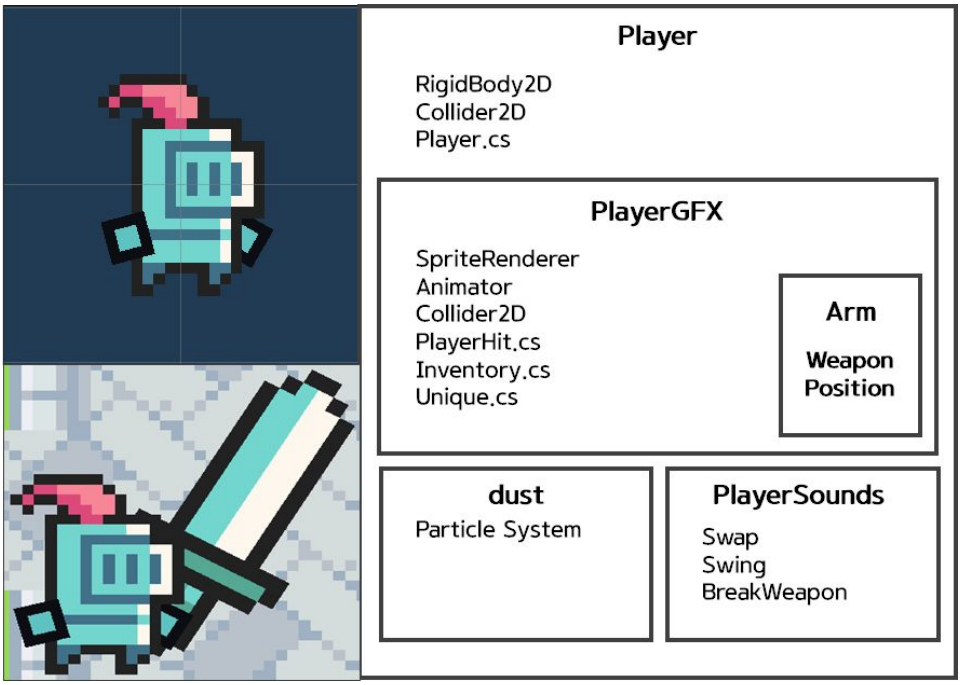
4.1 시작화면

모바일의 홈화면을 통해 어플리케이션을 실행하게 되면 위와 같은 화면이 표시된다. 타이틀을 제외하고 4개 버튼은 각각 개별의 기능이 존재한다.

- 새 게임 : 게임시작 버튼 중 하나로 저장된 데이터를 포함시키지 않은 상태로 home씬을 구성하여 불러온다.
- 이어하기 : 게임시작 버튼 중 하나로 저장된 데이터를 포함시킨 상태로 이전의 진행정보를 반영하여 home씬을 구성하여 불러온다.
- 옵션 : 옵션 창을 활성화 시키며 옵션 창에는 효과음과 배경음의 음량을 조절할 수 있는 슬라이더와 음소거를 할수 있는 토글 버튼이 각각 존재하고, 해당 설정들은 SoundManager를 통해 관리된다.
- X(앱 종료 버튼) : X버튼을 누르면 현재 실행되고 있는 어플리케이션이 종료된다.

4.2 플레이어와 몬스터

4.2.1 플레이어



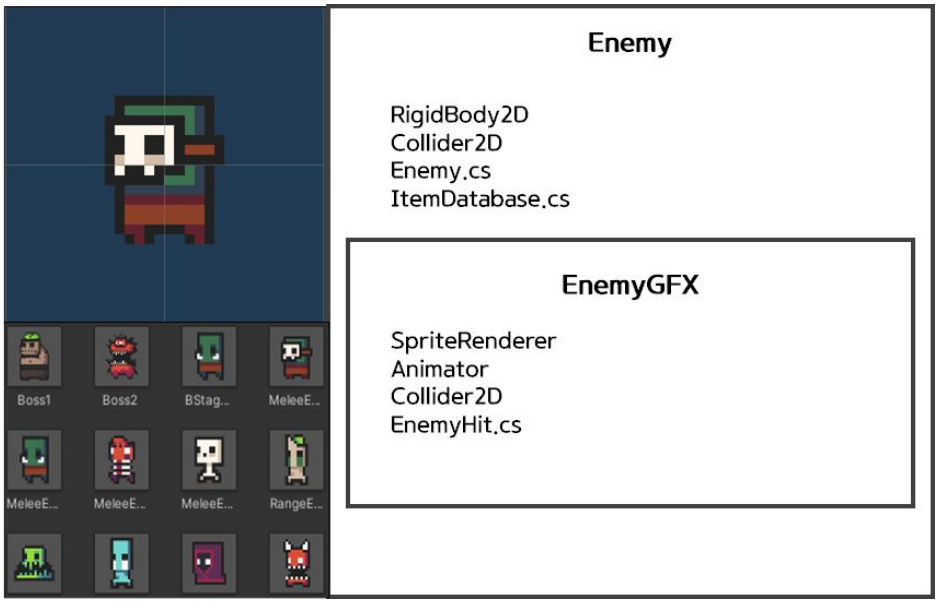
사용자가 조작하는 플레이어 캐릭터의 계층 구조이다. 사실 기능을 수행하는 기준으로 나누어보면 크게 Player와 PlayerGFX로 나뉘게 된다. 두 오브젝트의 기능을 나누면 아래와 같다.

- Player : 공격, 이동, 상호작용, 무기 장착, 무기 교체, 무기에 따른 애니메이션 결정, 레벨업
- PlayerGFX : 애니메이션 실행, 고유아이템 적용, 인벤토리 기능(습득 및 저장, 상점이용, 코인), 피격, 사망

dust와 PlayerSounds는 Player의 상태에 따라 상황에 맞게 실행된다. Arm 오브젝트는 Player가 PlayerGFX의 Inventory에 접근하여 현재 무기의 정보를 갱신하면 그에 맞도록 리소스 폴더에서 일치하는 무기 오브젝트를 불러와 Arm 오브젝트의 하위에 위치시켜 무기를 시각적으로 기능적으로 활성화시키기 위한 목적으로 사용되었다.

4.2.2 몬스터

4.2.2.1 일반 몬스터

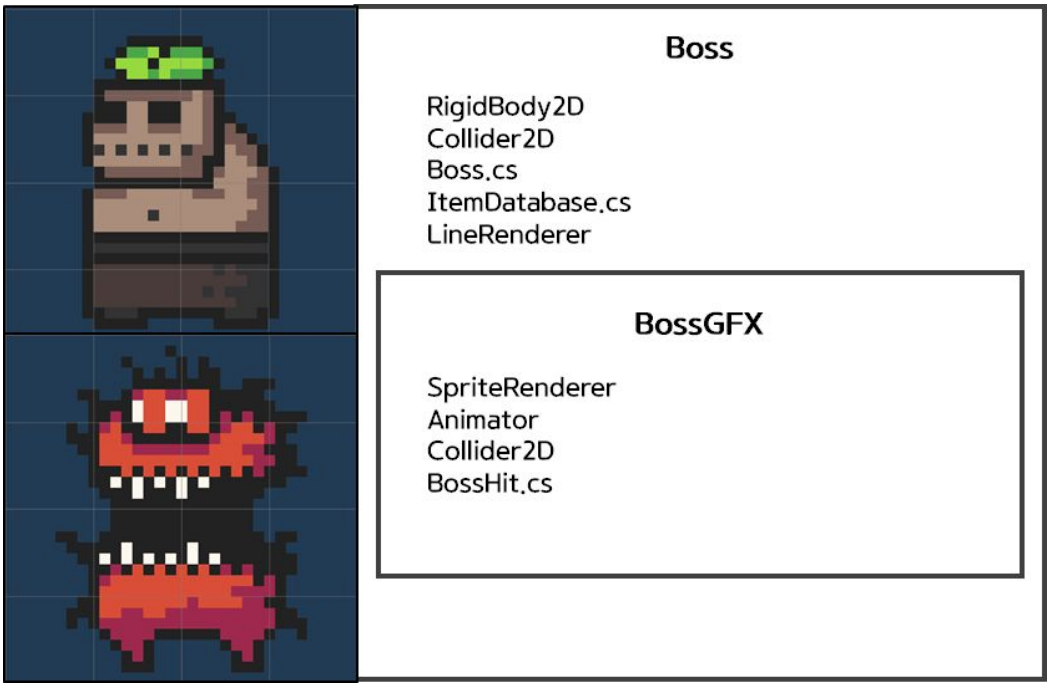


이 게임에서 몬스터는 근거리 몬스터, 원거리 몬스터, 소환형 몬스터로 크게 3종류로 나뉜다.

- 근거리 몬스터 : 플레이어를 추적하여 달라붙으며 방해 및 공격을 한다.
- 원거리 몬스터 : 플레이어를 타겟팅하여 탄환을 발사한다. 이동을 불가하며 맵에 고정 상태로 존재한다.
- 소환형 몬스터 : 보스 몬스터가 소환하는 몬스터이며 경험치나 드랍되는 재화, 아이템이 존재하지 않는다.

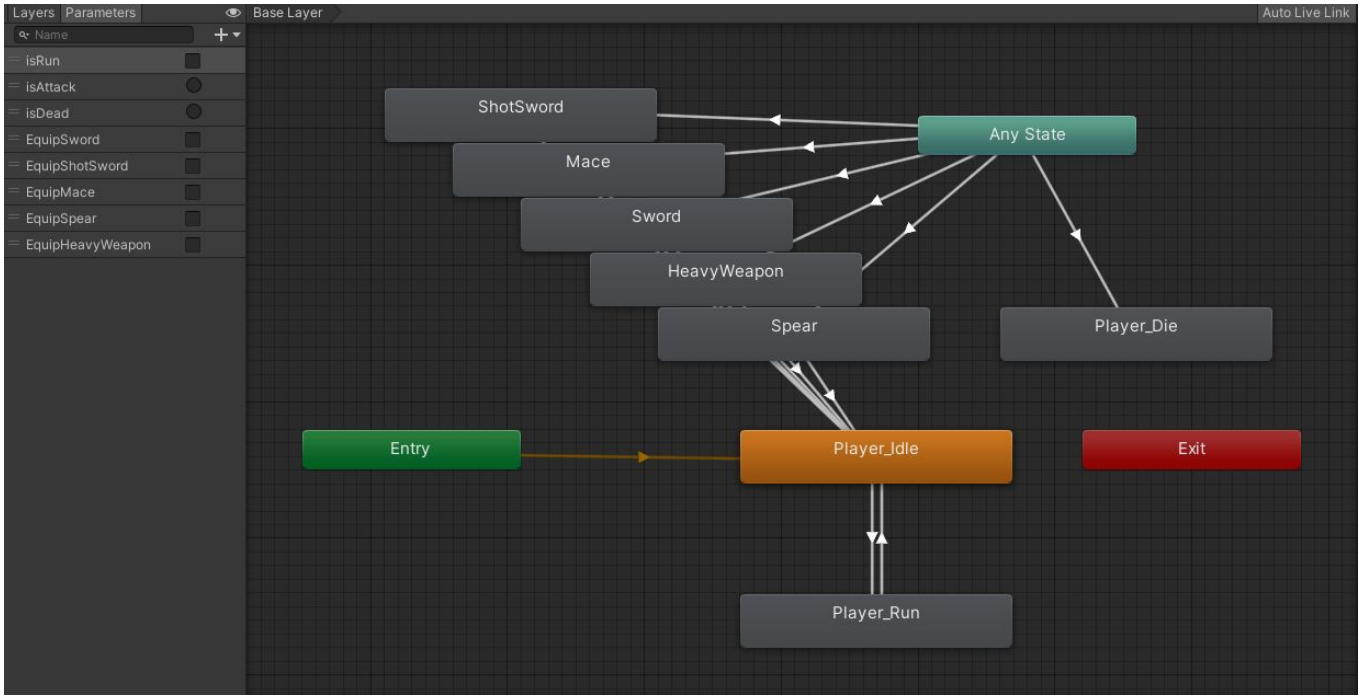
각 오브젝트의 기능을 살펴보자면, Enemy는 공격, 이동, 추적, 애니메이션제어의 기능을 가지고 있고, EnemyGFX는 애니메이션 실행, 피격, 사망 시 이벤트(플레이어 경험치 상승, 아이템 드랍, 코인 드랍)으로 나누어져 있다.

4.2.2.2 보스 몬스터



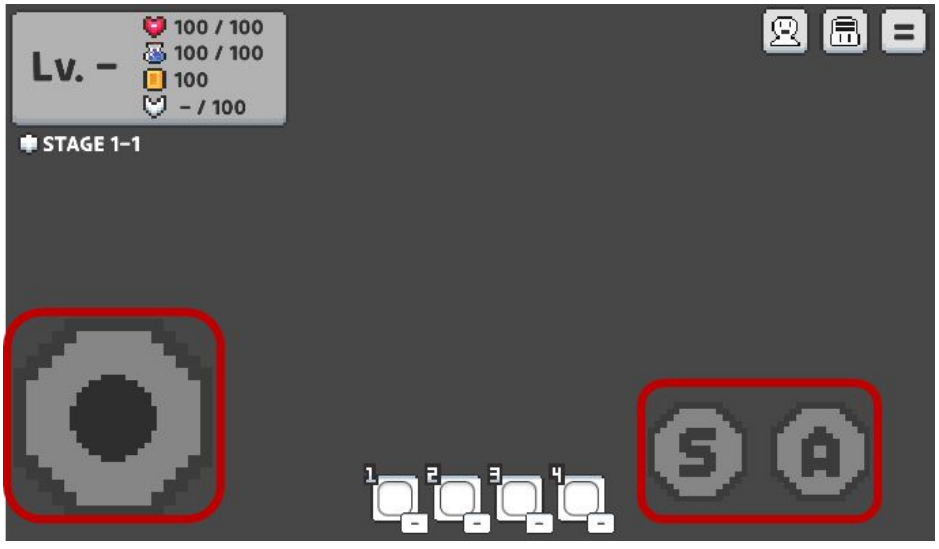
보스 몬스터는 일반 몬스터와 대부분 비슷한 구조를 가지고 있지만, Boss.cs에 각 보스의 특유의 패턴을 담고있는 보스 루틴이 존재한다. LineRenderer를 사용하여 보스 루틴에서 실행되는 공격에 대한 공격 방향을 시각적으로 표기하였다. 그리고 또 다른 차이점은 일반 몬스터와 다르게 무기 아이템이 아닌 장비아이템을 드랍한다.

4.2.3 애니메이션



애니메이션의 전체적인 흐름을 보이기 위한 Player오브젝트의 Animator를 가져왔다. 애니메이션은 컴포넌트가 생성되어 실행되면 Entry에서 Player_idle로 들어가 반복재생을 한다. 애니메이터는 지속적으로 파라미터를 체크하며 파라미터의 변경에 따라 실시간으로 조건에 맞는 애니메이션을 실행시킨다. 원래대로라면 각 애니메이션들은 모두 화살표로 이어져있어야 흐름이 끊기지 않고 실행되지만 Any State를 통해 화살표가 이어져있지않더라도 파라미터 조건만 일치하면 해당 애니메이션을 실행시킨다.

4.2.4 이동 및 공격



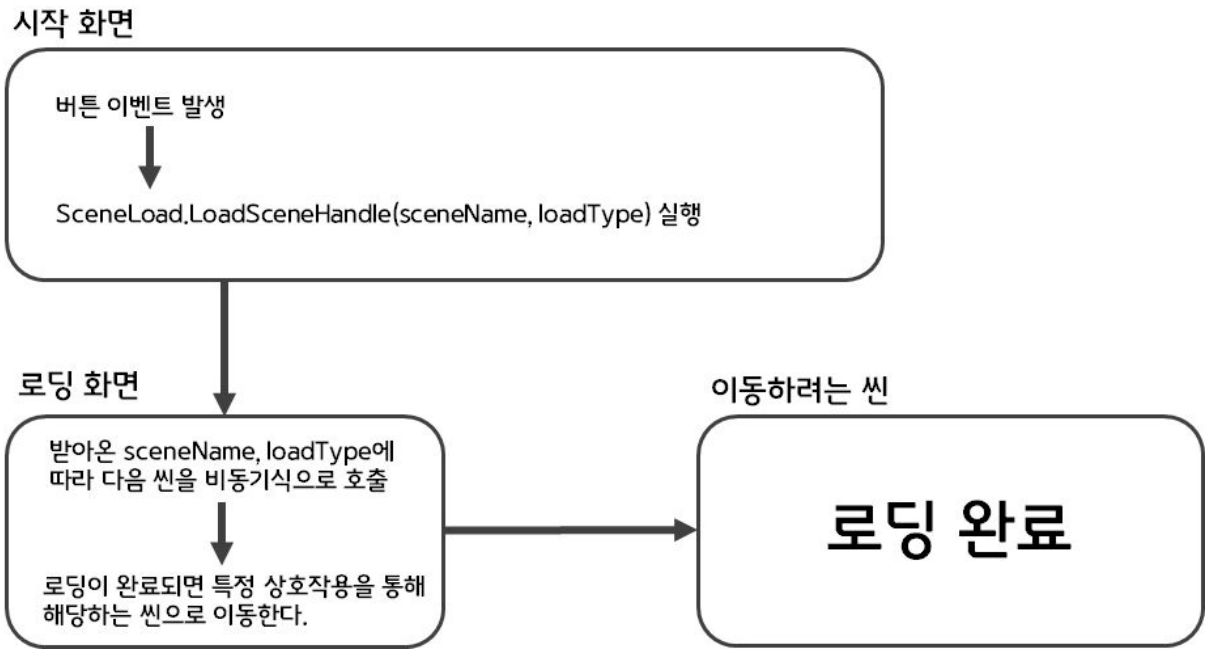
목표로 하는 플랫폼은 모바일 환경이기 때문에 터치를 통해 기능이 실행되야한다. 그러기 위해서 이벤트 시스템을 통한 터치 이벤트를 사용하여 버튼과 조이스틱을 통해 플레이어의 조작을 하도록 하였다.

조이스틱 : 해당 오브젝트는 터치를 통해 이동이 가능하며 오브젝트의 중심에서 벗어나는 정도를 계산하여 벡터값을 얻어와 플레이어에게 전달하여 이동하고자하는 방향을 알려준다.

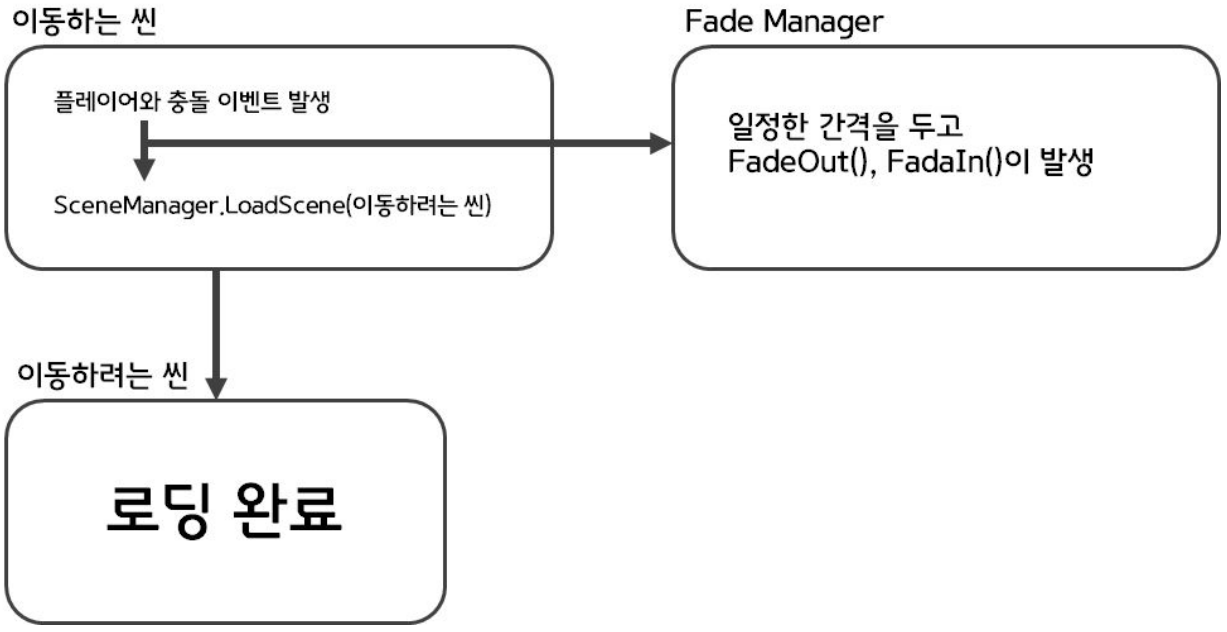
버튼 : 각 버튼은 교체 기능과, 공격 기능이 연결되어있으며 터치 시 기능이 실행된다.

4.3 씬 이동

4.3.1 시작화면 -> 인게임 씬 이동



4.3.2 스테이지->스테이지 씬 이동



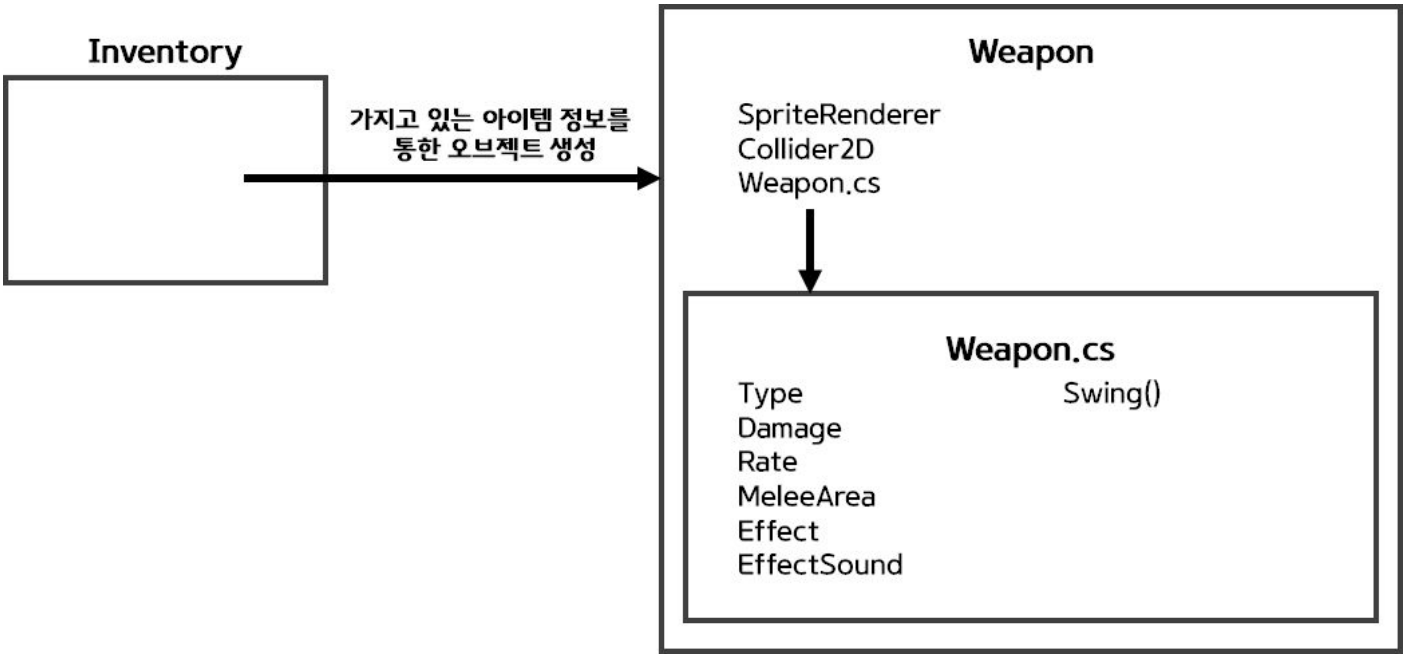
4.3.3 시작지점(home) 포탈 제약 조건

게임의 시작지점에는 각 에어리어에 진입할 수 있는 포탈이 존재한다. 진입 포탈마다 AreaNum이 부여되어 있어서 GameManager의 clearArea에 일치하는 포탈을 제외하고 모두 비활성화 된다.

각 에어리어의 보스를 클리어하게되면 clearArea를 하나 증가시킴으로서 1에서 2가 되면 에어리어1은 포탈이 닫히고 에어리어2의 포탈이 열리게 된다.

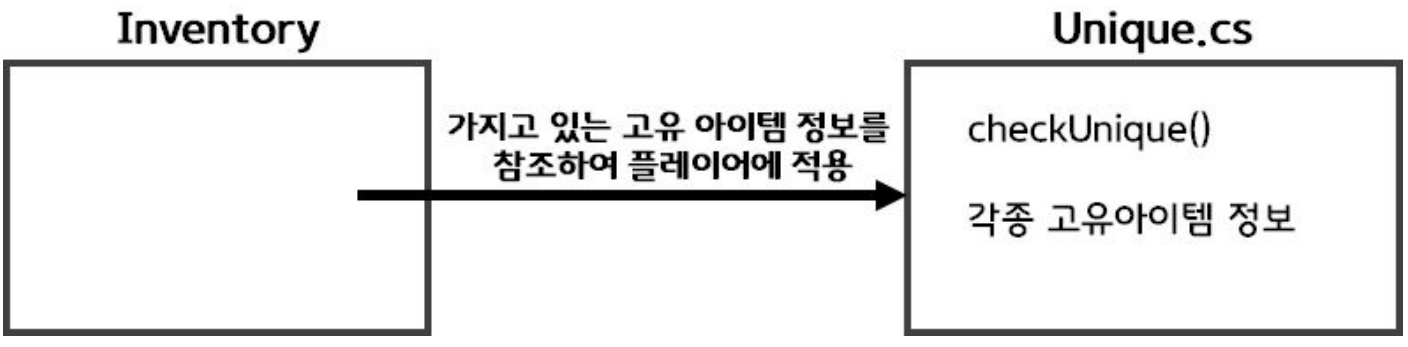
4.4 아이템

4.4.1 무기 시스템



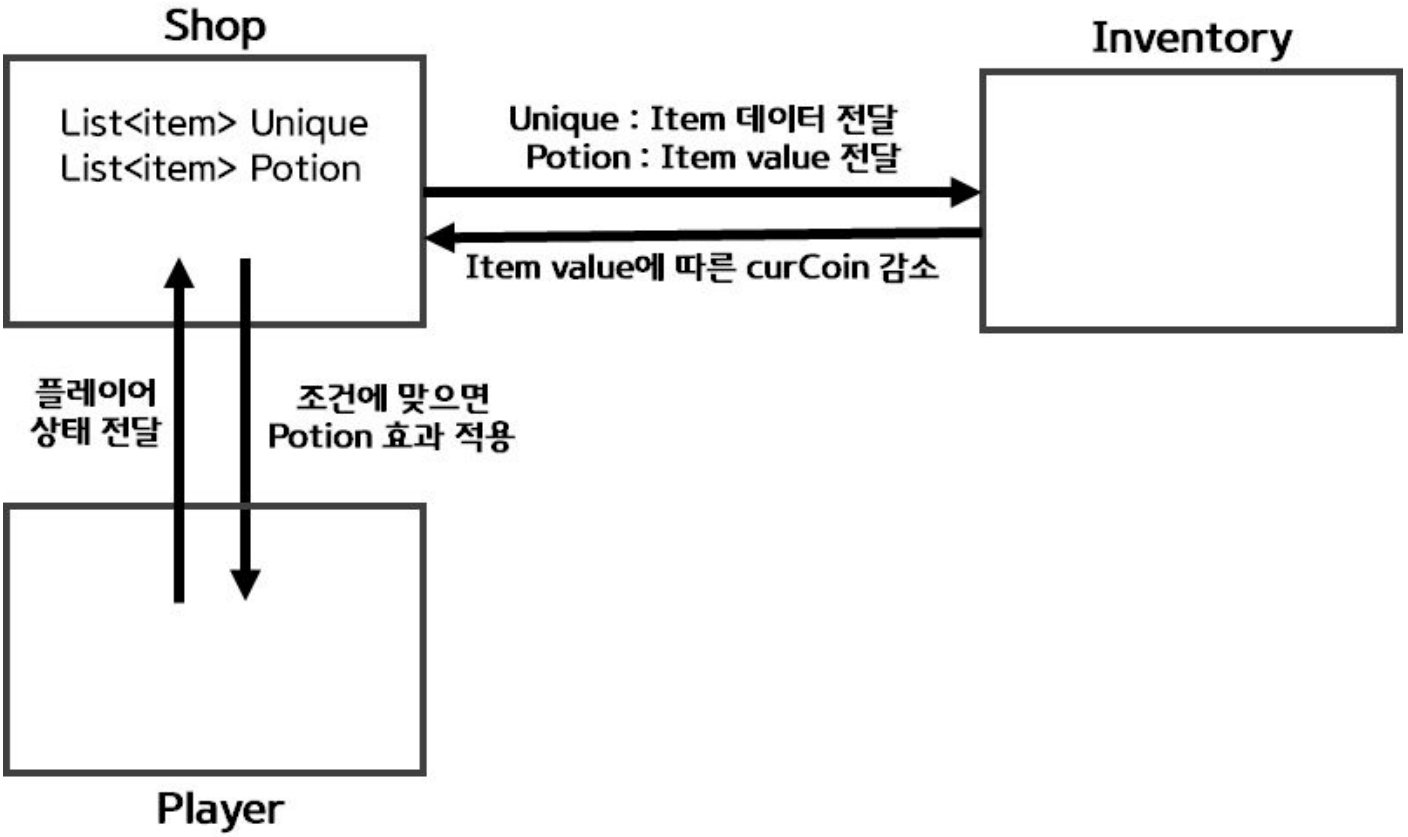
이 게임에서 무기는 사용횟수가 정해지며, 사용횟수를 전부 사용하면 무기는 파괴되는 소모품으로서 사용된다. Weapon.cs에 저장된 정보는 제외한 사용횟수등의 정보는 Inventory에 있는 Item 정보에 존재한다. 무기는 근접 무기이며, 무기의 세부 종류에 따라 공격 애니메이션이 결정된다.

4.4.2 고유 아이템



인벤토리에서 고유아이템 리스트를 받아와 checkUnique()를 통해 이미 가지고 있는 아이템이 아니라면 해당 기능을 적용시킨다. Unique.cs내부에 고유 아이템 기능에 대한 정보가 담겨져 있다.

4.5 상점 구현

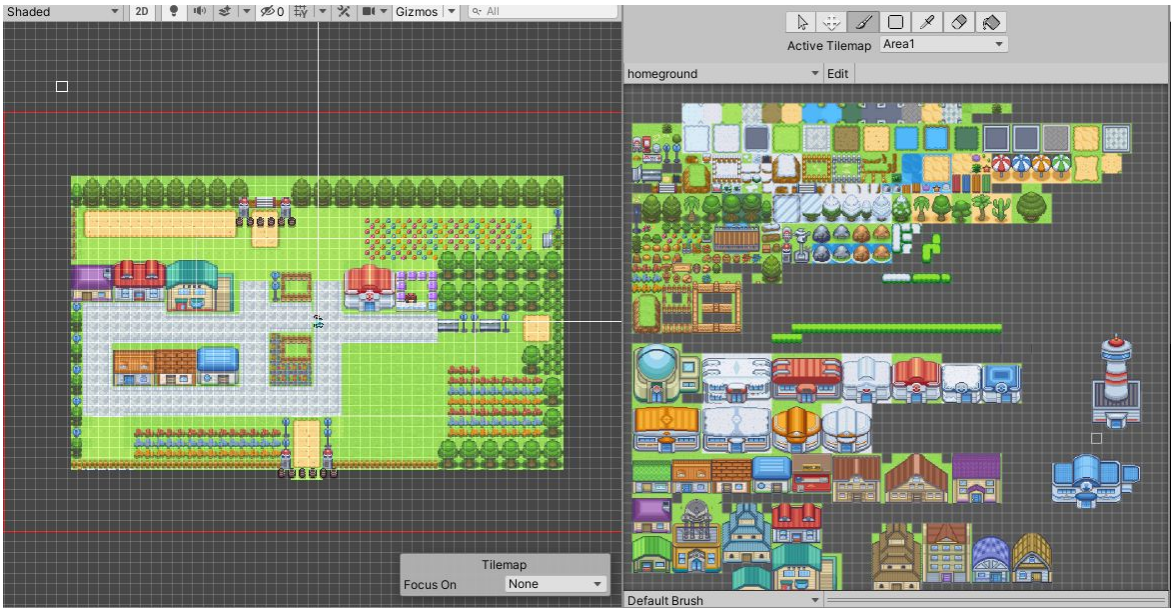


Shop을 통해 플레이어가 현재 Inventory를 통해 소지하고있는 Coin을 통해 고유 아이템을 구매하거나 포션 아이템을 구매할 수 있다. Shop은 일반적으로 구매 기능만 지원하며 특수 탭과 포션 탭을 통해 각 아이템 정보에 접근 할 수 있다. 그리고 슬롯을 클릭하여 구매를 희망하는 아이템 정보를 확인후 구매 버튼을 눌러 아이템은 얻는 방식이다.

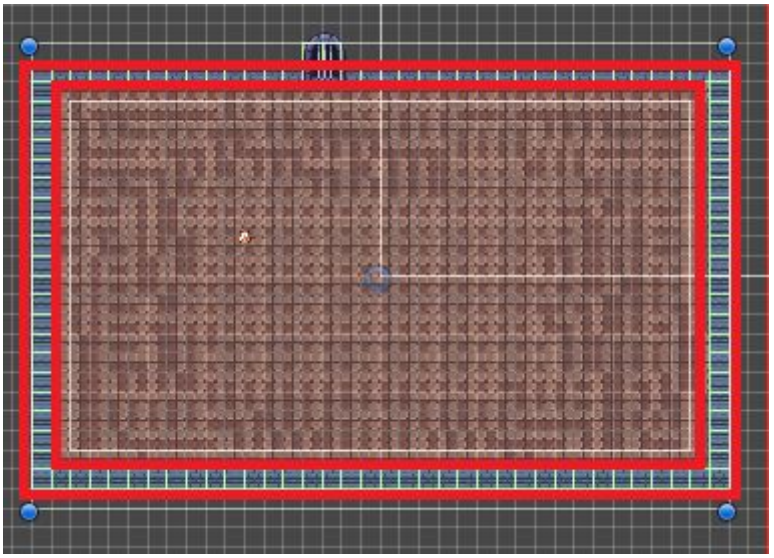
- 고유 아이템 구매 : 구매 버튼을 클릭하면 Shop 객체는 Inventory의 소지재화를 확인하여 선택한 고유 아이템이 소지재화보다 많으면 구매를 취소하고 이벤트를 취소한다. 소지재화가 더 많으면 인벤토리의 uniqueitem 리스트에 해당 Item 정보를 전달하여 플레이어가 Unique 스크립트를 통해 아이템 정보를 갱신할 수 있도록 해준다.
- 포션 아이템 구매 : 외부의 스크립트를 통해 기능을 구현하는 다른 아이템들과는 다르게 포션 아이템은 Shop 객체 내부에서 모든 기능을 지원한다. Shop객체는 플레이어와 연결되어 있어 언제든 플레이어의 상태를 확인 할 수 있기 때문에 구매한 포션 아이템에 따라 기능이 실행될때 조건에 맞는 경우 구매 과정이 발생하게 된다.

4.6 맵 - (클리어 조건, 보상, 몬스터 생성)

4.6.1 맵제작



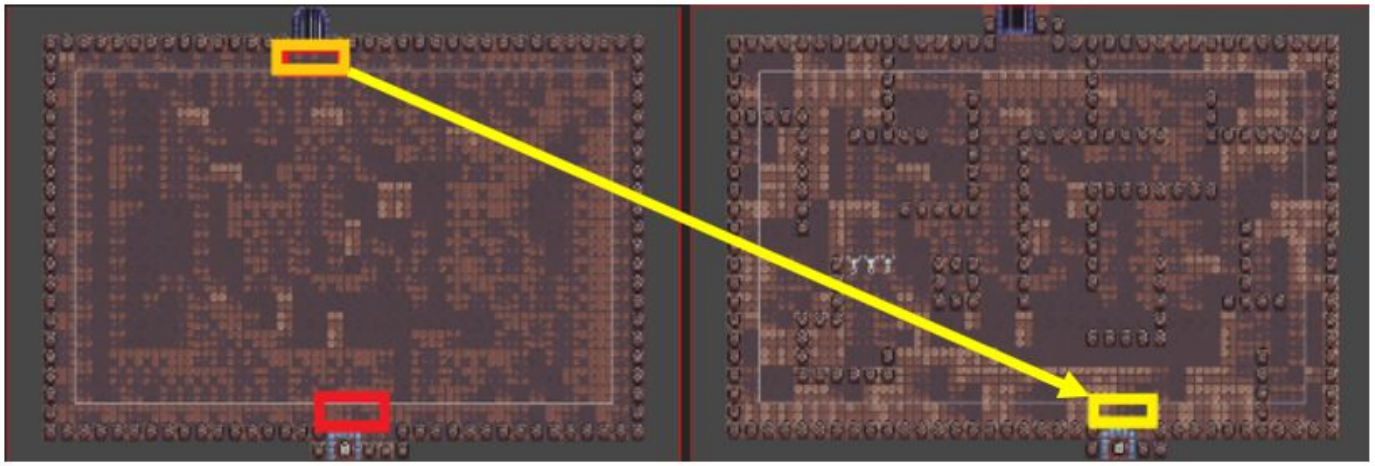
외부에서 가져온 에셋파일에서 이미지를 추출해 타일 팔레트를 만들고 타일 팔레트를 활용해 개발자가 원하는 방식으로 타일을 생성해 제작한다.



- **background :** 작은 빨간 박스영역이 Player가 밟고 지나갈 수 있는 모든 영역
- **colid :** 큰 빨간 박스와 작은 빨간 박스사이에 있는 테두리는 Player가 맵 밖으로 나갈 수 없게 만든 영역 또는 장애물

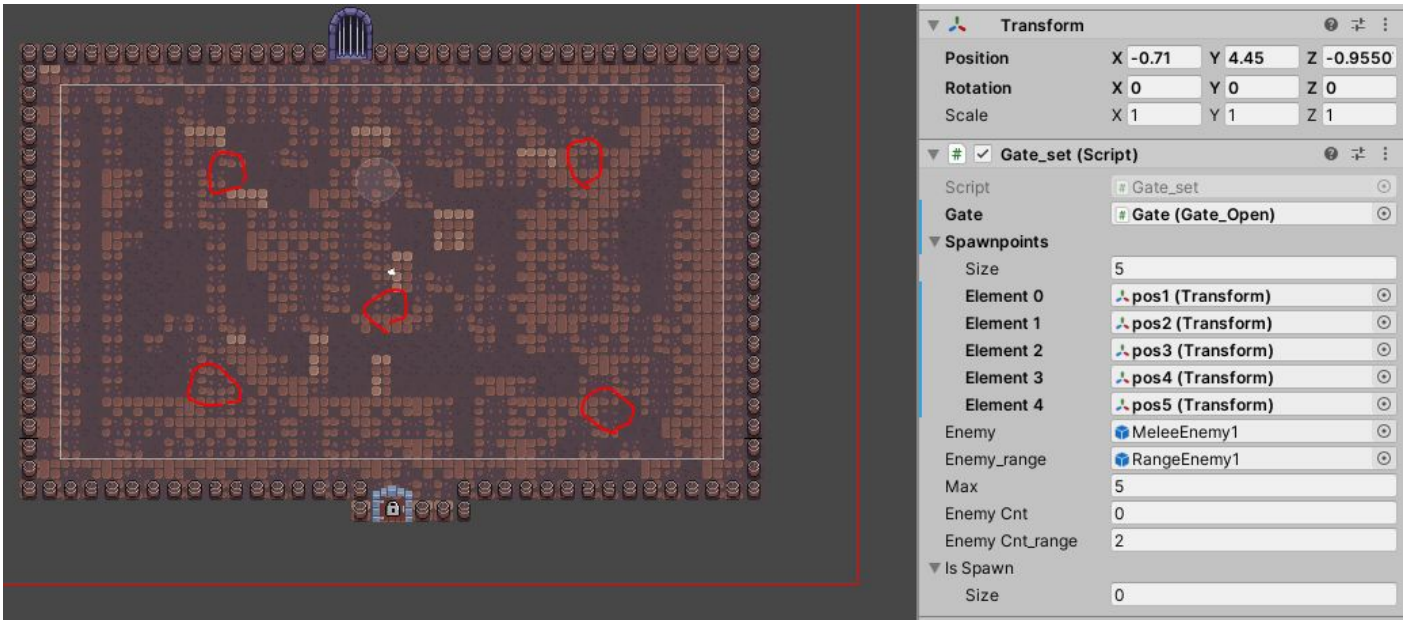
4.6.2 맵 이동

기능: Player가 다른 맵으로 이동할 수 있도록 한다.



- **Maptranform(A의 주황박스):** Player가 주황박스와 충돌시 A맵에서 B맵으로 이동하게 함
- **StartingPoint(A의 빨간박스, B의 노란박스):** Player가 다른 맵으로 이동할때 이동한 맵에 시작하는 위치로 좌표를 설정하게 만듬. 그래서 A맵에서 B맵으로 이동할 때 Player은 B맵의 노란박스에서 시작하게 된다.

4.6.3 몬스터 생성

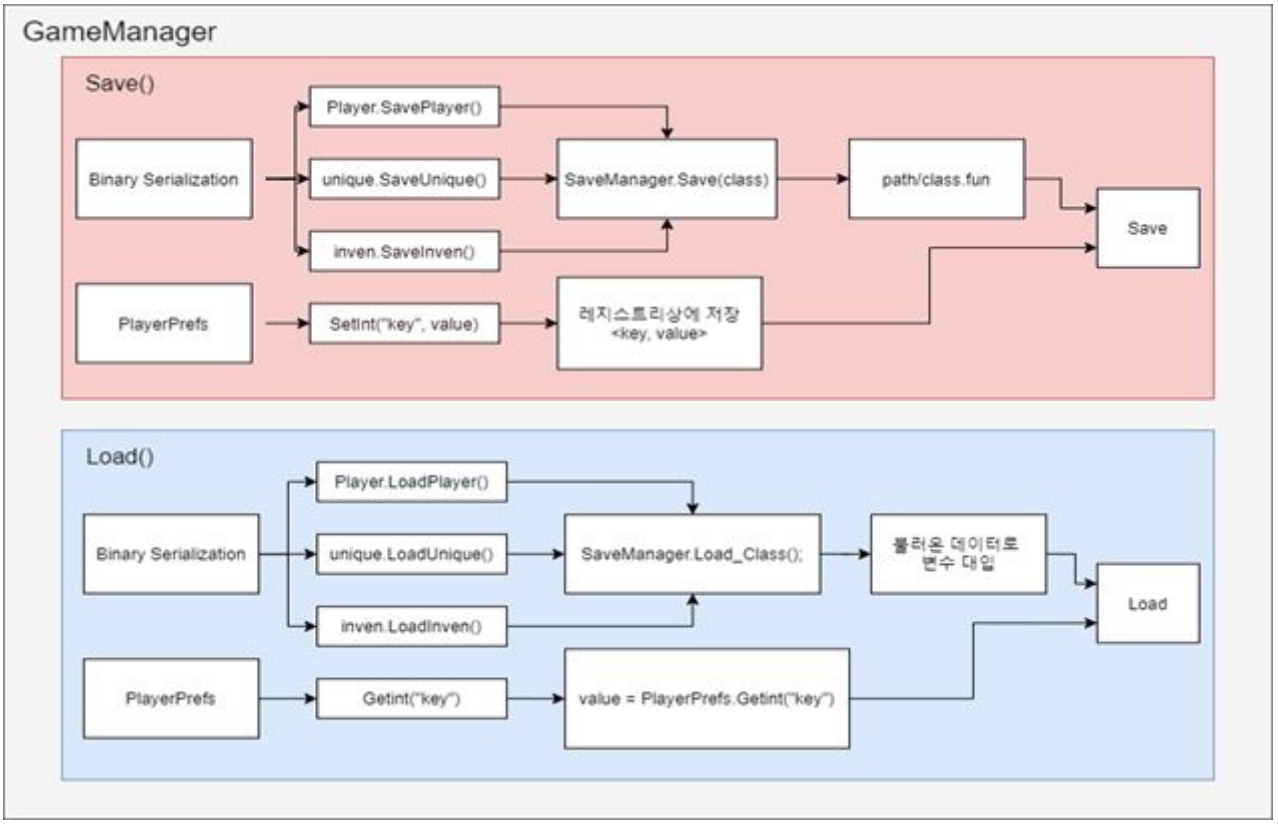


- **Gate :** Enemy Cnt가 0이 됐을 경우 Gate가 사라지게 설정
- **Spawnpoints.size:** 몬스터를 생성할 총 위치 값을 설정
- **Element:** 몬스터가 생성되는 위치
- **Enemy:** 근접 공격 몬스터 생성
- **Enemy_range:** 원거리 공격 몬스터 생성
- **Max:** 몬스터가 생성할 수 있는 최대의 수
- **Enemy Cnt:** 맵에서 생성된 몬스터 수를 나타나며, Player가 몬스터를 잡으면 1씩 값이 줄어듬

- **Enemy Cnt_range:** Max 범위 안에서 원거리 공격 몬스터 수를 미리 정해서 게임 실행시 미리 정한 수 만큼 나오게 설정하는 기능

4.7 저장 / 불러오기

저장과 불러오기는 두가지 방식으로 사용된다. 바이너리 직렬화 방식과 PlayerPrefs 클래스를 활용한 방식이다.



- Save()와 Load() 함수는 GameManager 내부에 존재하는 함수로 다른 오브젝트를 참조하여 저장이 필요한 객체에 대해 한번에 저장/불러오기 처리를 수행한다.
- **바이너리 직렬화** : 저장데이터를 파일의 형식으로 만들어 읽고 쓰는 것으로 오브젝트 정보의 갱신을 할 수 있다.
- **PlayerPrefs** : 유니티에서 제공하는 클래스이며 <key : value> 쌍으로 간단한 데이터의 저장을 구현할 수 있다.

5. 테스트 보고서

5.1 테스트 내용

[1차 테스트]

테스트 목록	테스트 내용	성공 판정 기준
1	아이템 수집, 몬스터 타격	-아이템을 수집시 인벤토리에 저장 -몬스터를 타격시 몬스터의 hp가 떨어져서 최종적으로 죽게 되면 아이템이 나오고 플레이어가 획득 가능
2	맵과 플레이어 연동	-플레이어가 맵 안에서만 움직이는지 확인 -플레이어가 사물이나 벽과 충돌시 더 이상 이동이 되지 않는지 확인
3	몬스터 활동 자동화	-몬스터가 스스로 움직이는지 확인 -몬스터 활동범위안에 플레이어가 오면 플레이어를 공격하는지 확인.
4	맵 불러오기	-각각의 에어리어를 실행시 2개의 스테이지와 정해진 보스 스테이지가 나오는지 확인

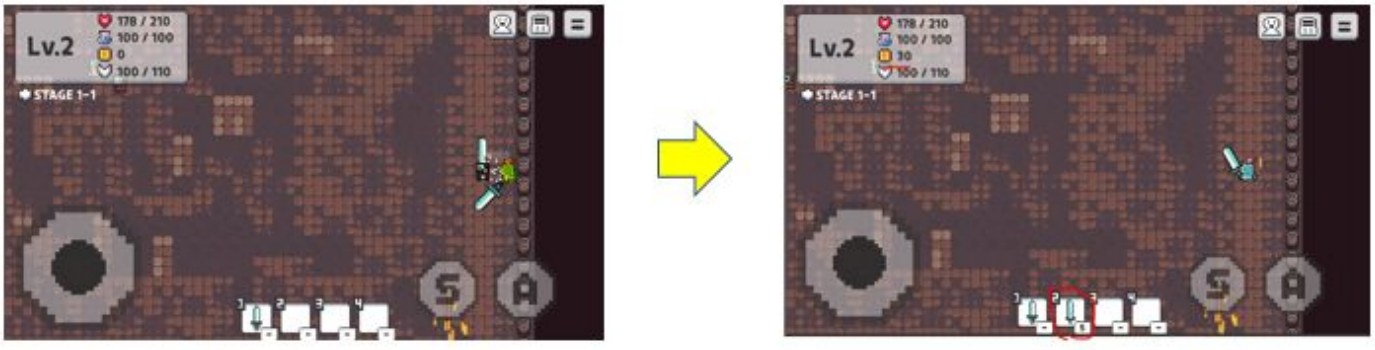
[2차 테스트]

테스트 목록	테스트 내용	성공 판정 기준
5	레벨업으로 인한 플레이어 능력치 향상	-플레이어가 레벨업시 능력치가 변화되는지 확인
6	보스 공격 패턴	-보스가 플레이어를 추적하여 공격 패턴을 수행하는 지 확인 - 패턴이 정해진 로직에 따라 움직이는지 확인
7	모바일 환경 적용	-폰에서 게임이 진행되는지 확인 -가상 패드와 가상 버튼으로 게임과

		상호작용이 뜻대로 되는지 확인
8	저장/불러오기	-저장한 후 게임을 종료하고 다시 실행시 제대로 불러오는 지 확인
9	상점에서 아이템 구매시 인벤토리 창에 이동 확인	-상점에서 구매된 아이템이 인벤토리에 들어가는 지 확인 -아이템 구매 시 일정한 양의 코인이 사용되는지 확인
10	background-music 작동	-무기를 사용시 타격음이 제대로 나오는지 확인 -스테이지에 맞는 bgm이 나오는지 확인 -스테이지를 클리어시 bgm이 나오는지 확인

5.2 테스트 결과

1) 아이템 수집, 몬스터 타격

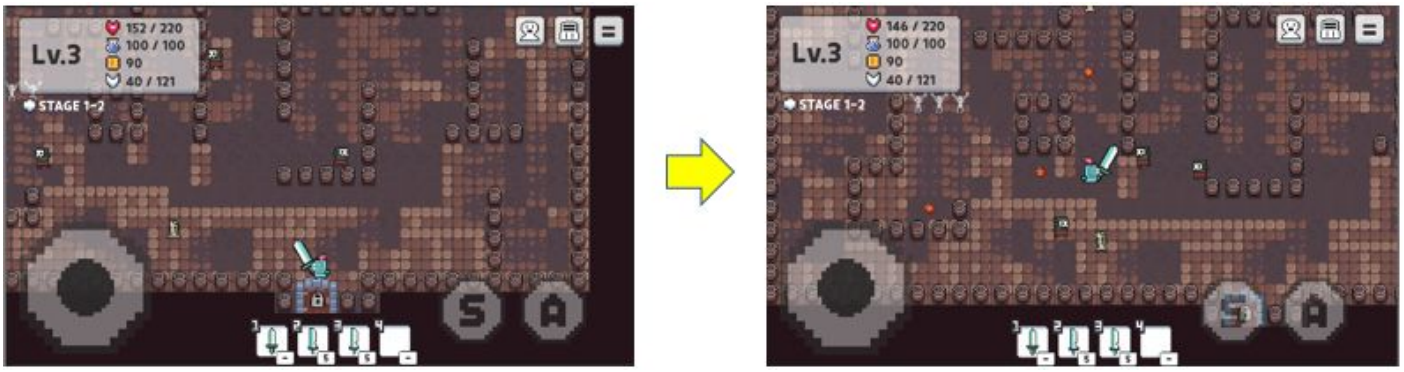


- 몬스터 타격을 통해 코인과 무기 아이템의 드랍을 확인 하였고, 근처에 다가가면 아이템이 인벤토리에 추가되는 것을 확인 할 수 있다.

2) 맵과 플레이어 연동

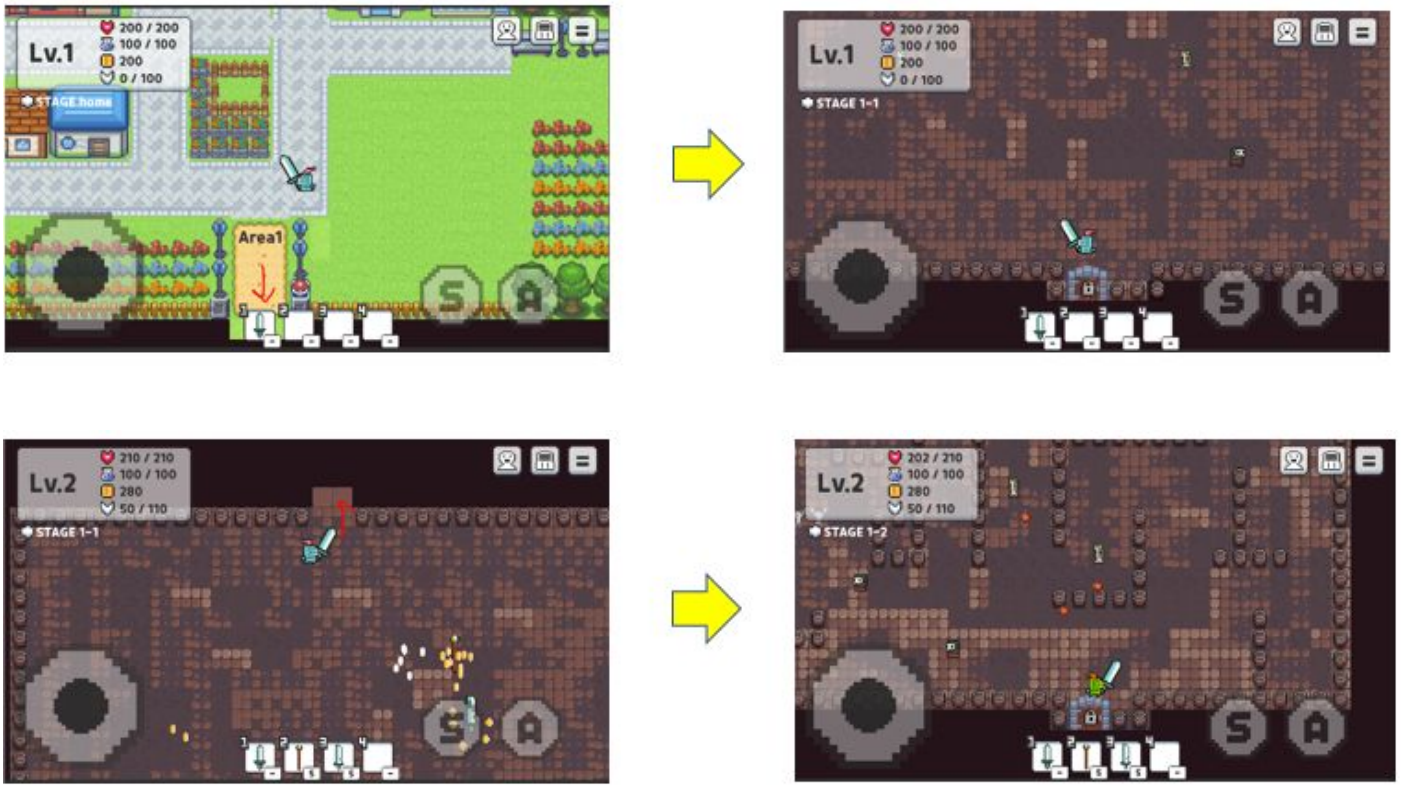
- 구성된 맵을 플레이어 캐릭터가 조작을 통해 이동 가능하고 맵의 외곽에 존재하는 콜라이더로 인해 맵 바깥으로 나가지지 않는다. 그리고 플레이어의 이동에 따라 카메라가 정상적으로 추적하여 캐릭터가 화면의 중심에 위치시켰다.

3) 몬스터 활동 자동화



- 생성된 몬스터는 시간의 흐름에 따라 자율적으로 좌우 이동을 하고 근처에 플레이어가 존재하면 추적하여 공격 또는 이동을 수행하는 것을 확인할 수 있었다.

4) 맵 이동



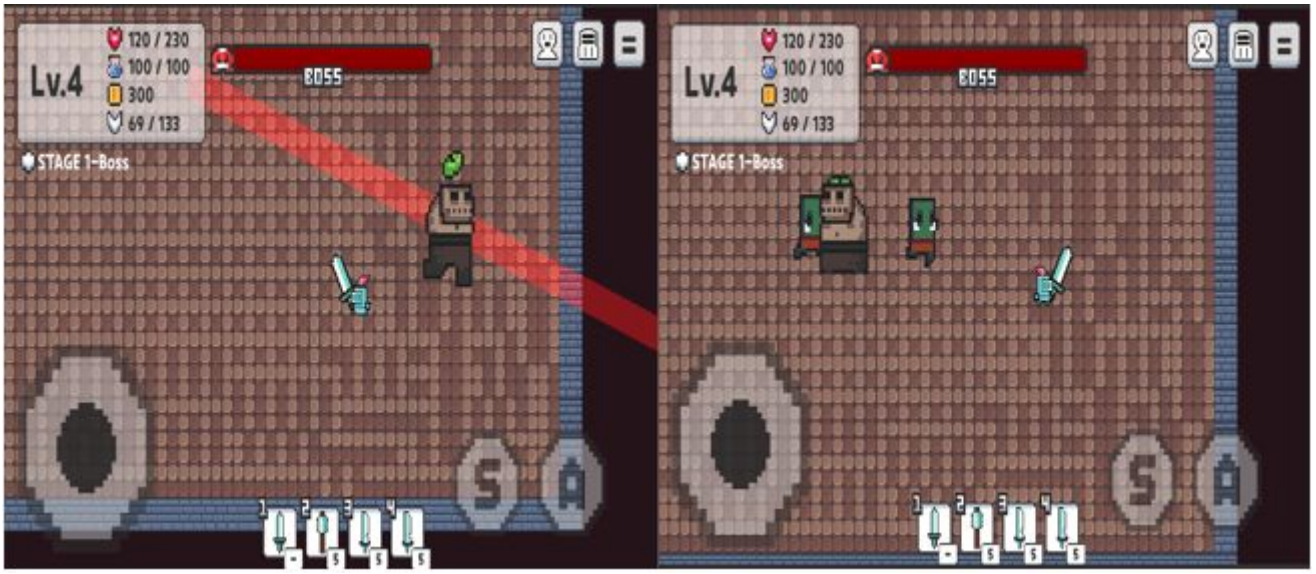
- 플레이어가 포탈을 사용하면 다음 맵을 불러오고 플레이어의 위치를 다음 스타트포인트에 위치시키고 카메라의 위치도 초기화 시킴으로써 맵 이동을 정상적으로 수행하였다.

5) 레벨업으로 인한 플레이어 능력치 향상

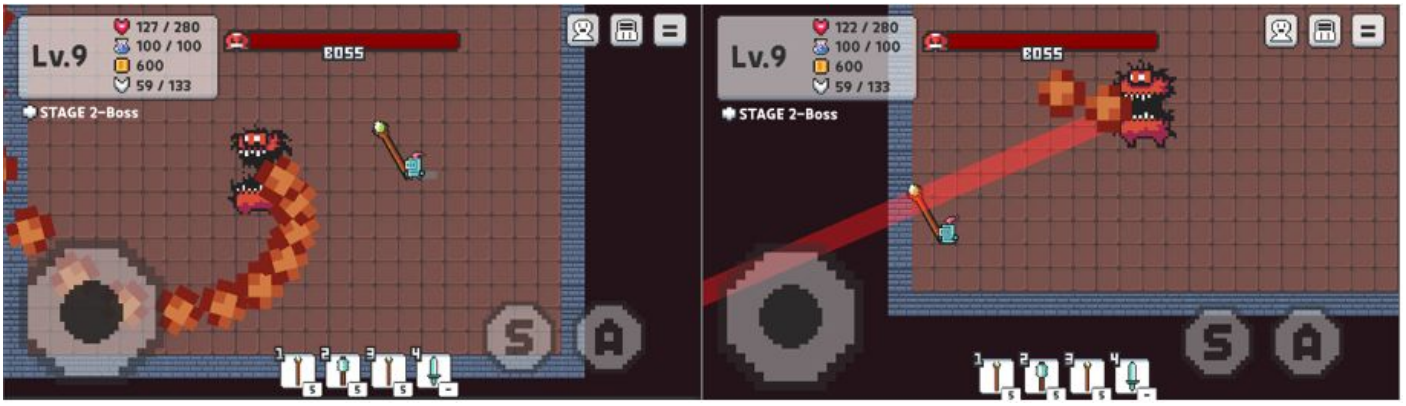
- 플레이어 캐릭터의 레벨이 증가함에 따라 체력, 공격력, 방어력의 수치가 설정한 고정값대로 정상적으로 상승함을 확인할 수 있었다.

6) 보스 공격 패턴

- 보스 1



- 보스2

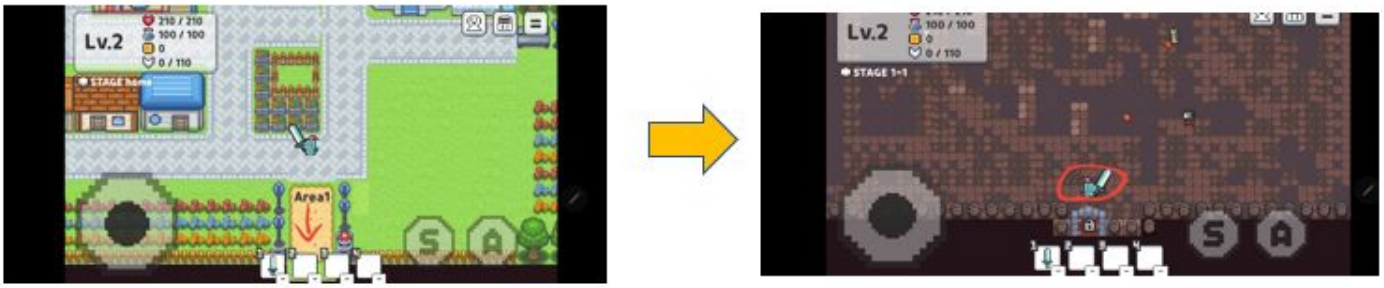


- 보스1, 보스2는 각각 고유의 패턴이 존재한다. 보스 1은 근접공격+소환공격, 보스2는 근접공격+원거리공격+전체공격 패턴이 있다. 테스트 결과 모든 패턴이 정상적으로 수행됨을 확인할 수 있었다.

7) 모바일 환경 적용

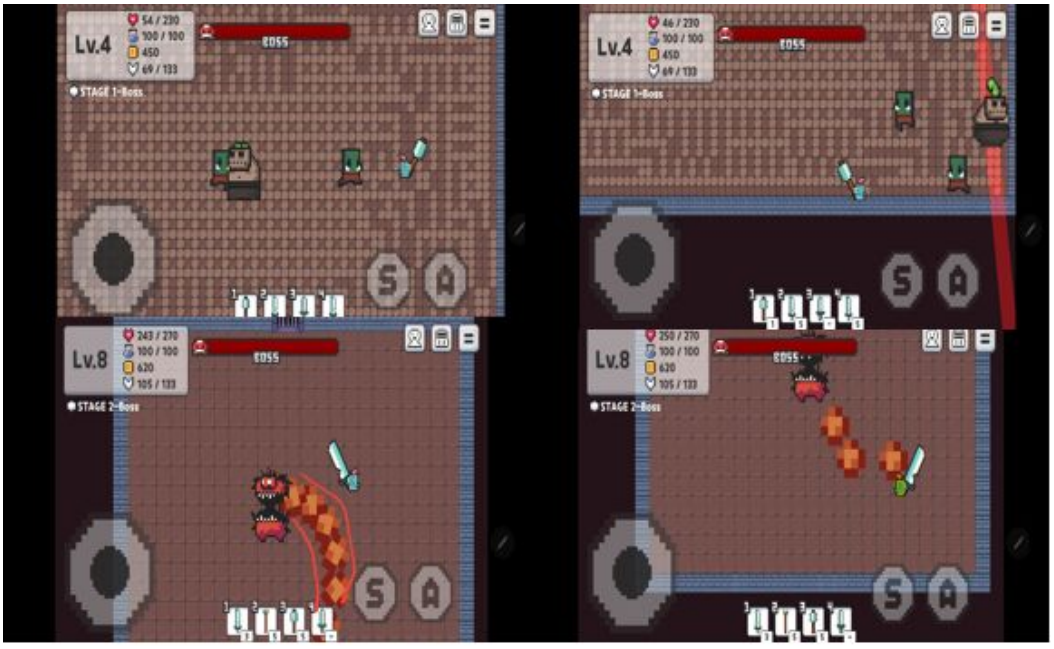
- 상점구매

- 맵 이동



- 저장/불러오기

- 보스 패턴



- 레벨 상태 변화

- 테스트를 완료한 항목들을 대상으로 모바일 환경에서 다시한번 구동하여 결과를 확인하였다. 모바일 환경에서 기존의 테스트 결과와 동일한 결과를 보였으며 정상적으로 작동함을 확인하였다.
-

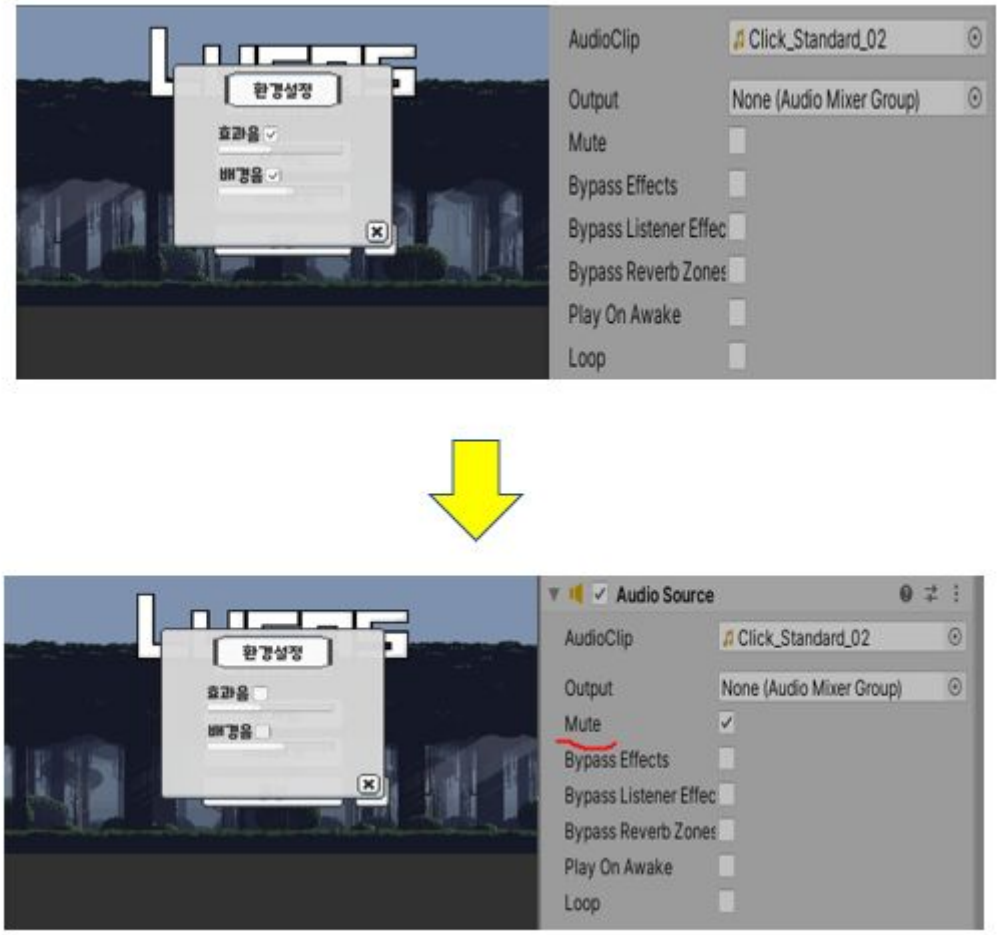
8) 저장/불러오기

- 게임의 초기 상태에서 저장된 데이터의 불러오기를 실행 하였다. 실행 결과 저장된 데이터를 기반으로한 플레이어 상태, 현재 클리어 해야할 에어리어 정보, 인벤토리 아이템 정보등 성공적으로 데이터가 갱신됨을 확인 할 수 있었다.

9) 상점에서 아이템 구매시 인벤토리 창에 이동 확인

- 상점을 통해 고유 아이템을 구매할 때 인벤토리의 코인 차감이 발생하고 인벤토리에 구매한 아이템 정보가 정상적으로 들어옴을 확인 할 수 있었다.

10) background-music 작동



- 씬에 존재하는 오디오 소스를 관리하는 SoundManager가 UI와 연동되어 볼륨의 조절, 항목 별 음소거를 확인할 수 있었다.

5.2 이슈 트래킹 로그

1) Collider2D 동시 사용 이슈

- **개요 :** 플레이어 캐릭터의 상호작용 콜라이더와 피격 콜라이더를 사용할때 한 콜라이더가 정상적으로 작동하지 않음.
- **문제 원인 :** 스크립트에서 충돌 이벤트 처리를 할때 순서 상 더 상위에 존재하는 콜라이더 하나만 우선적으로 점유하고 나머지 콜라이더는 처리 충돌 이벤트에서 제외된다.
- **해결 방법 :** 오브젝트를 상위 오브젝트와 하위 오브젝트의 형태, 즉 계층적 구조로 만들어 상호작용 콜라이더의 기능과 피격 콜라이더의 기능을 나누어 구현하는 것으로 두가지 기능 모두 정상적으로 처리하였다.

2) Player 정보 초기화

- **개요** : 플레이어 캐릭터가 한 에어리어를 클리어한 후 시작지점으로 돌아 왔을때 캐릭터의 레벨, 아이템, 경험치 등 누적된 정보들이 게임을 처음 시작한 상태로 초기화 됨.
- **문제 원인** : 플레이어 캐릭터는 씬이 이동할 때 오브젝트가 파괴되지 않도록 DontDestroyOnLoad 속성을 부여해준다. 시작지점에는 플레이어 캐릭터 오브젝트나 게임매니저 오브젝트가 이미 존재하는데, DonDestroyOnLoad 속성이 부여된 다른 캐릭터나 게임매니저가 씬을 넘어오게 되는 부분에서 문제가 발생한다. 결과적으로 한 씬에 두 세트의 게임 요소가 존재하여 씬에 원래 존재했던 초기파일을 먼저 읽어들이는 부분에서 초기화 라고 생각 된것이다.
- **해결 방법** : 오브젝트의 생성 단계에서 스크립트의 static을 사용한 생성여부를 판단하는 변수를 생성해주어 이미 생성된 기록이 있다면 바로 오브젝트를 파괴하여 한 게임에 하나의 오브젝트만 존재하도록 하는 싱글톤 기법으로 문제를 해결하였다.

3) 사운드 매니저 컨트롤

- **개요** : 시작화면 옵션에서 사운드 볼륨을 설정하고 게임을 시작할때 초기에 설정한 볼륨 값이 게임 내부에서 적용되지 않는다.
- **문제 원인** : 시작화면에서 게임 화면으로 넘어가는 부분에서 SoundManager는 파괴된 이후 재 생성된다. 이때 값을 저장하는 부분이 존재하지 않기 때문에 내부에서 재생성될때 초기 값으로 저장되지 않는 것이다.
- **해결 방법** : PlayerPrefs 클래스를 사용한 간단한 저장을 활용하였다. 업데이트 주기마다 SetVolume() 함수를 통하여 변수를 저장하고 새로운 SoundManager가 생성될때 GetVolume() 함수를 통하여 저장된 변수를 키값으로 불러와 볼륨 값을 정해준다. 그렇게 하면 업데이트를 통해 지속적으로 저장된 값이 새로만들어진 SoundManager 오브젝트에 적용되서 설정 값이 유지된다.

4) 세이브 이슈

- **개요** : 바이너리 직렬화 방식을 이용하여 Player, Inventory, Unique의 상태정보를 저장하려는 과정에서 정상적으로 저장이 안되고 오류문구 출력
- **문제 원인** : 바이너리 직렬화 방식의 저장은 int, float, string등 기본형의 저장에는 간편하지만 image, sound 같은 데이터는 저장을 지원하지 않는다. 모든 클래스에서 사용되는 Item 클래스가 이미지 데이터를 포함하고 있기 때문에 오류가 발생하는 것이다.
- **해결 방법** : Item 클래스의 Image 데이터 파일을 ItemName을 통한 이미지 매칭으로 재설계하는 것으로 해결하였다. string 값을 통한 이미지로 대체하였으므로 바이너리 직렬화를 통한 저장도 정상적으로 지원하는 것을 확인 할 수 있었다.

5) 보스 돌진 벽 뚫는 버그

- **개요** : 보스가 플레이어를 향해 대쉬 공격을 가하고, 맵의 벽에 충돌할 때 특정 확률로 벽을 뚫고 맵 바깥으로 나가는 버그 발생.
- **문제 원인** : 게임의 물리현상을 담당하는 RigidBody 컴포넌트에는 Collision Detection이라는 옵션이 존재한다. 여기서 설정 값을 Discrete로 해주게 되면 성능에는 좋지만 충돌을 검출하는 간격이 길어져 충돌 검출하는 시간보다 물체를 지나가는 속도가 더 빠르게 되면 충돌체를 지나가는 문제가 생긴다.
- **해결 방법** : 문제를 해결하기 위해서는 충돌체의 크기를 늘리는 방법과 Collision Detection의 또 다른 설정 값인 Continuous로 설정해 주면 된다. 여기서는 Continuous로 설정해주는 것으로 해결했다. Continuous는 프레임 단위로 충돌을 확인하여 처리하는 방식인데 처리하는데 들어가는 자원의 양은 증가하지만 확실한 결과를 보여준다.

6. 문제점 및 개선 방안

1) 조작감

문제점

터치를 통한 조이스틱의 조작이 민감한 문제가 있다. 그리고 실질적인 좌표상 이동은 문제가 없지만 캐릭터를 표현하는 이미지가 좌우, 2개만 존재하다보니 상하 조작에 있어서 표현이 부족하고 좌우로 상하를 표현하려다 보니 조작감이 불편하게 느껴졌다.

개선 방안

조이스틱의 민감도를 낮추고, 이동에 표현되는 이미지를 늘려야한다. 현재 이 프로젝트에서는 그래픽을 이루는 이미지를 외부에서 가져오므로 직접적인 수정이 힘들다. 그래서 이 문제를 해결하기 위해서는 4~8방향을 지원하는 에셋을 구해 캐릭터의 디자인을 변경하는 해야한다. 표현되는 이미지가 많아 질수록 캐릭터의 이동을 자연스럽게 표현 될 듯하다.

2) 스토리

문제점

개발을 진행하다보니 예상했던 흐름대로 개발이 되지 않고, 많은 오류와 수정으로 인해 개발이 늦춰지게 되면서 RPG를 구성하는 중요한 요소라고 생각되는 스토리가 미구현되었다. 스토리를 통해 플레이어가 좀 더 게임에 녹아들 수 있고, 즐기게 할 수 있는 요소라고 생각되었는데 이러한 스토리가 제외되어 게임의 완성도가 떨어지고 매력이 없는 듯 보인다.

개선 방안

추가적인 보완을 통해 스토리의 추가를 고려해야한다. 스토리는 단순히 게임의 배경뿐만아니라 카메라 무빙을 통한 화면의 연출, 대사를 통한 퀘스트 전달같은 요소가 추가될 수 있는 것이기 때문에 스토리의 추가로 플레이어가 즐길 수 있는 컨텐츠가 늘어나게 된다.

7. 프로젝트 후기

32153033 이건희

전에 아두이노/라즈베리 파이 실습, 웹 사이트 만든 경험을 제외한 새로운 영역에 도전하고 싶었습니다. 이후 게임 제작 영상을 보고 게임을 제작해보겠다는 마음을 먹게 되었습니다. 게임 제작 영상을 시청한 당시에 영상대로 따라하기만 하면 그렇게 어렵지 않을 것이라고 생각을 했습니다. 그러나 막상 게임을 제작을 하면서 생각치 않던 어려움들이 많았습니다. 게임제작 경험이 없었기 때문에 다른 사람들이 한 것을 보고 따라하고 참조하기도 했지만, 따라한 것이 오류가 발생했습니다. 게다가 각각의 모듈대로는 이상이 없었으나, 합치는 과정에서 모듈끼리 충돌했습니다. 지나고 생각하면 게임제작 경험이 없었기 때문입니다. 처음에는 기능들을 만들고 합치는 방식으로 반복이라고 생각했습니다. 마치 뼈대를 만들지 않고 살부터 붙이는 것입니다. 즉, 뼈대를 먼저 설정하고 그 다음으로 살을 붙이는 것입니다. 그래도 이번 프로젝트를 수행하면서 게임 프로그램 뿐만 아니라 프로그램을 만들 때 과정을 머리속에 어떻게 그릴 수 있는지 알게 됐습니다. 다만 아쉬운 점은 제작기간에 더 완성도가 높은 장르를 선택하지 못한 것입니다. 한 학기 동안 같이 제작한 동료에게 감사하고 다음번에는 완성도가 높은 작품을 만들기 위해 기량을 닦을 것입니다.

32154024 장현준

지금까지 게임개발에 관심은 있었지만 크게 행동으로는 옮기지 못해 동영상을 통해 부분적으로 따라만들어본 경험 밖에 없었던 저로서는 큰 도전이었다고 생각합니다. 그려져 있는 가이드 없이 처음부터 끝까지 개발에 개입하여 기본적인 게임 개발의 틀을 완성했다는 것에 큰 의미가 있는 프로젝트였습니다.

하지만 처음하는 도전이었기에 부족한 부분도 많았고 선택에 있어서 아쉬운 부분도 있었습니다. 특히 개발의 방향을 확실히 하는 것과 설계단계의 중요성을 많이 느꼈습니다. 개발을 하다보니 생각했던 방향과 다르게 진행이 되는 부분도 존재했고, 기존에 짜놓았던 기능의 코드가 추후에 추가되는 기능으로 인해 재설계를 해야하는 경우도 있었습니다. 그렇기에 다음번에 개발을 하게 될때운 더더욱 기능과 기능, 코드와 코드와의 관계를 확실히 하고 설계를 단단히 해두어야겠다고 생각하였습니다.

이 프로젝트를 하면서 물론 RPG장르를 형태를 갖추고 만들어가는 것도 재미있었지만, 기간을 고려했을때 RPG라는 장르가 적합했는지에 대해 조금 아쉬운 느낌이 들었습니다. 기본적인 시스템과 게임의 흐름은 구현했지만, 프로젝트 기간이 끝나갔을때 간신히 틀만 잡아놓은 느낌이어서 추가적인 게임의 색체를 주지 못한것이 아쉬웠습니다. 게임의 스토리나 스토리에 들어가는 카메라 무빙을 통한 연출같은 것도 해보고

싶었지만 집중했던 시간이 조금 부족했던 것 같습니다. RPG는 게임의 특징적으로 오브젝트간의 상호작용도 많고 물리적인 이벤트 처리도 많았기 때문에 기반을 닦는데 시간이 오래걸립니다. 그래서 만약 2D 플랫포머 점프 액션 게임이나, 퍼즐 게임 종류로 진행했다면 어떤 방식으로 진행되었을까 생각하고는 합니다. 하지만 RPG 만들어 보는 것에 로망이 있었고, 지금 배운 것을 기반으로 다음에는 좀더 나은 게임을 만들 수 있을 것이라고 생각합니다.

한학기동안 맡은 역할을 충분히 수행한 팀원에게도 정말로 감사하고, 이렇게 게임 개발이라는 생각치 못한 프로젝트로 한학기 즐겁게 개발할 수 있었던 것 같습니다.