

# Unity를 활용한 하이브리드 RPG 게임 프로젝트

VOID

2025년 11월 07일

김기찬  
성명 송진혁  
임승태  
소속 컴퓨터공학전공  
학년 4  
20201536  
학번 20201531  
20201098  
지도교수 박정규 교수님(김기찬, 송진혁)  
이시진 교수님(임승태)

# 목 차

1.	작품개요	.....	04p
1.1	작품 개요	.....	04p
1.2	작품 선정 배경	.....	04p
1.3	유사 시스템 분석	.....	06p
1.4	구현 범위 및 목표	.....	07p
1.5	개발 환경	.....	09p
2.	관련 기술	.....	10p
2.1	Unity 2D 게임 개발	.....	10p
2.2	게임 디자인 패턴	.....	10p
2.3	데이터 지속성 기술	.....	11p
2.4	UI/UX 시스템	.....	12p
3.	구현내용	.....	13p
3.1	게임 시스템 아키텍처	.....	13p
3.1.1	전체 시스템 구조도	.....	13p
3.1.2	핵심 클래스 다이어그램	.....	13p
3.1.3	데이터베이스 스키마	.....	13p
3.2	주요 시스템 구현	.....	16p
3.2.1	플레이어 시스템	.....	16p
3.2.2	적 AI 시스템	.....	17p
3.2.3	보스 전투 시스템	.....	18p
3.2.4	씬 관리 및 포탈 시스템	.....	20p
3.2.5	세이브/로드 시스템	.....	21p
3.2.6	상점 시스템	.....	22p
3.2.7	인벤토리 시스템	.....	23p
3.2.8	미니게임 시스템	.....	24p
3.2.9	스테이지 3 수중 시스템	.....	24p
3.2.10	UI 시스템	.....	25p
3.3	시퀀스 다이어그램	.....	26p
3.3.1	게임 시작 플로우	.....	26p
3.3.2	전투 시퀀스	.....	27p
3.3.3	보스 전투 시퀀스	.....	28p
3.3.4	세이브/로드 시퀀스	.....	29p
3.3.5	씬 전환 시퀀스	.....	30p
3.4	주요 알고리즘	.....	32p

3.4.1	데미지 계산 알고리즘	.....	32p
	적 AI 상태 전환 알고리즘	.....	34p
	주사위 클래시 해결 알고리즘	.....	37p
4.	테스트 및 최적화	.....	39p
4.1	테스트 방법론	.....	39p
4.2	발견된 문제 및 해결	.....	39p
4.3	성능 최적화	.....	40p
5.	결론	.....	42p
5.1	프로젝트 성과	.....	42p
5.2	한계점 및 개선 사항	.....	43p
5.3	추후 과제	.....	43p
부록 I	졸업작품 사진	.....	45p
부록 II	졸업작품 포스터	.....	51p
부록 III	프로젝트 파일 구조	.....	52p
부록 IV	게임 플레이 가이드	.....	61p
부록 V	리소스 목록	.....	62p
부록 VI	참고 문헌	.....	63p

## 1. 작품개요

### 1.1 작품 개요

#### 1) 게임 장르 및 컨셉

본 프로젝트는 Unity 6 엔진을 사용하여 개발한 하이브리드 RPG 게임으로, 실시간 액션과 턴제 전략을 결합한 혁신적인 전투 시스템을 제공한다. 일반 맵에서는 RPG 방식의 실시간 전투를, 보스 맵에서는 전략적인 턴제 카드 전투를 경험할 수 있으며, 플레이어는 다양한 무기와 장비를 수집하고 성장하는 여정을 경험한다.

#### 2) 주요 특징

1. 하이브리드 전투 시스템: 일반 맵(실시간 RPG) + 보스 맵(턴제 전략)
2. 3가지 무기 시스템 (검, 창, 메이스)
3. 레벨업 및 스텝 강화 시스템
4. 카드 기반 주사위 전투 시스템 (보스전)
5. 다양한 스테이지와 미니게임
6. 세이브 / 로드 시스템 (3개 슬롯)

#### 3) 핵심 게임플레이 매커니즘

1. 탐험 모드 : 횡스크롤 플랫폼과 방식의 스테이지 탐험
2. 전투 시스템 : 실시간 액션 전투 (일반 적) + 턴 기반 카드 전투 (보스)
3. 성장 시스템 : 경험치, 레벨업, 스텝 업그레이드, 장비 강화
4. 미니게임 : 대장간 불꽃 수집 게임, 라이프 맵 미니게임

## 1.2 작품 선정 배경

### 1) 개발 동기

기존 로그라이크 및 턴제 RPG 게임들은 각각 고유한 매력을 가지고 있지만, 몇 가지 구조적 한계점을 지니고 있다. 본 프로젝트는 이러한 장르의 문제점을 분석하고, 하이브리드 방식을 통해 개선된 게임 경험을 제공하는 것을 목표로 한다.

### 2) 기존 장르의 문제점 분석

#### 2.1) 로그라이크 게임의 문제점

1. 성장 피드백 부족 : 도주 실패 시 대부분의 진행 상황이 초기화되어 플레이어의 노력이 무의미하게 느껴질 수 있다. 영구 성장 요소가 제한적이거나 추상적이어서 가시적인 성장 피드백이 약하다.
2. 운(랜덤성)에 과도한 의존: 아이템 드롭, 맵 구조, 적 배치 등 핵심 요소가 운에 의존하여, 실력보다 운이 승패를 좌우하는 경우가 빈번하다.

이는 플레이어의 좌절감을 증가시키고 게임의 공정성을 해친다.

## 2.2) 턴제 게임의 문제점

1. 느린 게임 템포 : 턴제 전투는 각 행동마다 대기 시간이 필요하여 전체적인 게임 진행 속도가 느려진다. 특히 전투가 반복될수록 지루함이 누적된다.
2. 반복되는 전투 패턴 : 적의 AI 패턴이 고정적이거나 단순하여, 플레이어가 패턴을 파악한 후에는 동일한 전략을 반복하게 되어 전투의 긴장감이 떨어진다.
3. 전투 외 콘텐츠 부족 : 턴제 게임은 전투 시스템에 집중하다 보니 탐험, 수집, 미니게임 등 전투 외 콘텐츠가 부족하여 게임의 다양성이 제한된다.

## 2.3) 하이브리드 RPG 제안

본 프로젝트는 위의 문제점들을 해결하기 위해 다음과 같은 하이브리드 시스템을 제안한다.

### 1. 이중 전투 시스템

- 일반 맵 : RPG 방식의 실시간 전투로 빠른 템포와 액션성 제공
- 보스 맵 : 턴제 전략 전투로 깊이 있는 전술적 사고 요구
- 두 시스템의 장점을 결합하여 다양한 플레이 경험 제공

### 2. 다중 성장 시스템

- 레벨업 성장 : 능력치 증가 + 부가 효과 획득으로 가시적 성장
- 재화↔경험치 전환 : 플레이어가 성장 방향을 능동적으로 선택 가능
- 영구적이고 누적되는 성장으로 로그라이크의 허무함 해소

### 3. 동적 보스 전투

- 동시 턴 진행 : 플레이어와 보스가 동시에 행동하여 전투 템포 향상
- 상황별 패턴 변화 : 보스가 전투 상황에 따라 패턴을 변경하여 예측 불가능성 증가
- 단순 반복 전략을 방지하고 매 전투마다 새로운 전략 필요

### 4. 통합 콘텐츠 설계

- 실시간 탐험, 턴제 보스전, 미니게임, 상점 시스템을 유기적으로 연결
- 다양한 플레이 스타일을 지원하여 게임의 재미 요소 극대화

## 2.4) 프로젝트 목적 및 의의

본 프로젝트의 목적은 Unity 게임 엔진에 대한 깊이 있는 이해, 하이브리드 게임 디자인 실험 및 검증, 게임 디자인 패턴 및 아키텍처 학습, 프로젝트 관리 및 버전 관리 경험 축적, 완성도 있는 작품 제작이다.

특히 기존 장르의 한계를 극복하는 혁신적인 시스템 설계를 통해 게임 디자인 역량을 증명하고자 한다.

### 1.3 유사 게임 분석

#### 1) 벤치마크 게임 분석

본 프로젝트는 다양한 장르의 성공적인 게임들을 벤치마킹하여 하이브리드 시스템을 설계했다.

#### 2) 실시간 RPG 참고 게임

- Hollow Knight(Team Cherry): 벽 슬라이드, 대시 메커니즘, 체력 시스템을 참고했으며, 정교한 플레이어 컨트롤과 반응성을 학습했다. 2D 횡스크롤 액션의 기본 프레임 워크로 활용했다.

- Dead Cells(Motion Twin): 무기 시스템, 적 AI 패턴, 레벨 디자인을 참고했고, 빠른 전투 템포와 피드백을 학습했다. 다양한 무기와 장비 수집 시스템에서 영감을 얻었다.

#### 3) 턴제 전략 참고 게임

- Library of Ruina(Project Moon): 주사위 기반 카드 전투 시스템을 참고하여 전략적 전투 메커니즘을 학습했다. 특히 동시 합 시스템과 빌딩 요소를 보스 전투에 적용했다.

- Slay the Spire(MegaCrit): 카드 선택과 자원 관리, 전투 중 전략적 의사결정 구조를 참고했다. 영구 성장 요소와 메타 진행 시스템에서 아이디어를 얻었다.

#### 4) 성장 시스템 참고 게임

- Hades(Supergiant Games): 도주 실패 후에도 누적되는 영구 성장 시스템, 재화를 통한 업그레이드, NPC와의 상호작용을 참고했다. 실패해도 의미 있는 진행을 제공하는 설계를 학습했다.

#### 5) 차별화 포인트

본 게임은 다음과 같은 차별화 요소를 가지고 있다.

##### 1. 하이브리드 전투 시스템

- 일반 맵과 보스 맵에서 완전히 다른 전투 방식 제공
- 실시간 액션(일반)과 턴제 전략(보스)의 유기적 결합
- 두 시스템이 독립적이 아닌 서로 영향을 주는 통합 설계

##### 2. 동시 턴 진행 방식

- 기존 턴제 게임과 달리 플레이어와 보스가 동시에 행동
- 빠른 전투 템포 유지하면서도 전략적 깊이 제공
- 상대의 선택을 예측하고 대응하는 심리전 요소 추가

### 3. 다중 성장 시스템

- 레벨 업 시 능력치 증가 + 부가 효과 동시 획득
- 재화↔경험치 상호 전환으로 플레이어 선택권 강화
- 영구적이고 누적되는 성장으로 장기적 목표 제공
- 로그라이크의 랜덤성을 배제하고 확정적 성장 보장

### 4. 적응형 보스 AI

- 보스가 전투 상황(HP, 턴 수, 플레이어 전략)에 따라 패턴 변경
- 고정된 패턴 암기를 방지하고 매 전투마다 새로운 전략 요구

### 5. 통합 콘텐츠 생태계

- 실시간 탐험, 턴제 보스전, 미니게임, 상점 시스템이 유기적으로 연결
- 각 콘텐츠에서 얻은 자원이 다른 콘텐츠에 영향

## 1.4 구현 범위 및 목표

### 1) 주요 구현 기능

플레이어 시스템은 이동, 점프, 대시, 벽 슬라이드 등의 기본 액션과 3종 무기 시스템, 체력/공격력/방어력/이동속도 관리, 레벨 및 경험치 시스템을 포함한다.

전투 시스템은 일반 적 AI, 특수 패턴을 가진 중간보스 AI, 주사위 기반 카드 전투 보스전, 무기별 데미지 계산 및 피격 시스템을 구현했다.

세이브/로드 시스템은 3개 슬롯 지원, JSON 기반 데이터 저장, 씬 전환 시 자동 저장, 플레이어 위치/스탯/인벤토리 저장 기능을 제공한다.

인벤토리 및 상점 시스템은 아이템 획득 및 관리, 포션 사용을 통한 체력 회복, 랜덤 아이템 판매와 새로고침 기능을 가진 상점, 스탯 업그레이드 아이템을 포함한다.

미니게임 시스템은 골드, XP, 공격력 증가를 보상으로 주는 불꽃 수집 게임과 라이프 맵 미니게임을 구현했다.

UI/UX 시스템은 메인 메뉴, 무기 선택, 세이브 슬롯 선택 화면, 체력과 스탯을 표시하는 HUD, 스탯 선택이 가능한 레벨 업 UI, 일시 정지 메뉴, 데미지 텍스트 등을 포함한다.

씬 관리 시스템은 확인 UI를 포함한 포털 시스템, 플레이어 스폰 위치 관리, 미니게임 복귀 시스템, DontDestroyOnLoad 싱글 톤 관리를 구현했다.

스테이지 별 특수 시스템으로 Stage 3에는 수영 메커니즘, 산소 게이지, 산소 회복 구역을 구현했다.

## 2) 무기 시스템 비교

무기 시스템 비교

무기	기본 데미지	공격 범위	공격 속도	데미지 배수	특수 효과
검 (Sword)	10	1.5f	빠름	1.0x	대쉬 시 무적
창 (Lance)	8	2.5f	보통	0.9x	대쉬 공격
메이스 (Mace)	15	1.2f	느림	1.2x	넉백 효과

## 3) 개발 목표 및 완성도 기준

모든 핵심 시스템 구현 완료, 메인 메뉴부터 게임 종료까지 전체 플로우 완성, 3개 이상의 스테이지 및 보스전 구현 및 세이브/로드 시스템 안정성 확보, 버그 없는 플레이 경험 제공을 목표로 하였다.

## 4) 시스템별 구현 완성도

시스템	계획 기능 수	구현 기능 수	완성도	비고
플레이어 시스템	8	8	100%	이동, 점프, 대쉬, 벽 슬라이드, 3종 무기, 스텟 관리
적 AI 시스템	6	6	100%	일반 몬스터, 중간보스 2종, 보스, 스포너
보스 전투 시스템	5	5	100%	카드 선택, 주사위, 애니메이션, AI, 승패 처리
세이브/로드 시스템	4	4	100%	3슬롯, JSON, 자동저장, 데이터 복원
상점 시스템	4	4	100%	구매, 새로운 침, 랜덤 아이템, 효과 적용
인벤토리 시스템	3	3	100%	획득, 사용, 드래그 앤 드롭
미니게임 시스템	2	2	100%	대장간, 라이프맵
UI 시스템	7	7	100%	메뉴, HUD, 레벨업, 인벤토리, 일시정지 등
씬 관리 시스템	4	4	100%	포털, 스판, 복귀, DontDestroyOnLoad
Stage3 수중 시스템	3	3	100%	수영, 산소, 환경 요소
전체	46	46	100%	모든 계획 기능 구현 완료

## 1.5 개발 환경

### 1) 소프트웨어 환경

게임 엔진은 Unity 6를 사용했고, 개발 언어는 C#을 사용했다.

IDE는 Visual Studio Code를 사용, 버전 관리는 Git과 GitHub를 활용했다.

타겟 플랫폼은 Windows PC이다.

### 2) 하드웨어 환경

운영체제는 Windows 11을 사용했으며, Windows PC에서 개발을 진행했다.

### 3) 주요 사용 Unity 패키지

Unity 2D Pixel Perfect, Unity Timeline, Unity Input System, TextMeshPro를 사용했다. 그래픽은 Unity Sprite Editor와 외부 에셋을 활용했고, 사운드는 Unity Audio Source와 Audio Clip을 사용, 애니메이션 부분은 UnityAnimator와 AnimationController를 사용했다.

## 2. 관련 기술

### 2.1 Unity 2D 게임 개발

#### 1) Unity 2D 엔진 개요

본 프로젝트에서는 Unity 6를 사용했다. Unity 2D의 주요 특징으로는 2D Renderer 및 Sprite 시스템, 2D Physics, Tilemap 시스템, 2D 애니메이션 시스템, Sorting Layer 및 Order in Layer를 통한 렌더링 순서 관리가 있다.

#### 2) Rigidbody2D 물리 시스템

본 프로젝트에서는 Unity 6의 물리 API를 사용한다. linearVelocity를 사용하여 이동, 점프, 대시 등을 구현, 중력은 Gravity Scale을 통해 구현했다. 충돌 감지는 OnCollisionEnter2D와 OnTriggerEnter2D를 사용했다.

Body Type은 Dynamic을 통해 적과 플레이어 간에 구분을 짓도록 구현, 이동 플랫폼에는 Kinematic을 사용했다. Constraints는 2D에서 회전을 방지하기 위해 Freeze Rotation Z를 설정했다. Collision Detection은 빠른 움직임 감지를 위해 Continuous로 설정했다.

#### 3) 2D 애니메이션 시스템

Animator Controller를 사용하여 상태 기반 애니메이션을 구현하였다.

애니메이션 파라미터는 Float 타입으로 이동 속도, Bool 타입으로는 중력, 무기, 대쉬, 벽 짚기, Trigger 타입으로는 피격, 사망, 넥백 시스템을 구현했다.

애니메이션 이벤트는 EnableAttackHitbox()로 공격을 활성화하였으며, DisableAttackHitbox()로 공격을 비활성화하며, PlayAttackSound()로 사운드를 재생한다. 애니메이션 전환은 Has Exit Time을 false로 설정하여 즉시 전환하고, Transition Duration은 0.1초로 설정하여 부드러운 전환을 구현했다. Interruption Source는 Current State로 설정하여 현재 상태에서 중단 가능하도록 했다.

### 2.2 게임 디자인 패턴

#### 1) 싱글 톤 패턴 (Singleton Pattern)

게임 전역에서 하나의 인스턴스만 존재해야 하는 매니저 클래스에 싱글톤 패턴을 적용했다.

PlayerController : 플레이어 제어

GameManager : 게임 상태 및 세이브/로드

Inventory : 인벤토리 관리,

ShopManager : 상점 시스템,

BossGameManager : 보스 전투 상태 관리를 담당한다.

SaveManager : 세이브 파일 관리

싱글 톤 패턴 구현 시 DontDestroyOnLoad로 씬 전환 시에도 유지되도록 했으며, 메인 메뉴 복귀 시 수동으로 정리하여 메모리 누수를 방지했다.

## 2) 옵저버 패턴 (Observer Pattern)

이벤트 기반 시스템으로 UI 업데이트, 게임 상태 변경 등에 옵저버 패턴을 활용했다. 인벤토리 변경 시 UI 업데이트, 플레이어 스탯 변경 시 HUD 업데이트, 적 사망 시 아이템 드롭, 대화 완료 시 미니게임 시작 등에 적용했다.

## 2.3 데이터 지속성 기술

### 1) JSON 직렬화 (Unity JsonUtility)

플레이어 데이터를 JSON 형식으로 세이브/로드한다. SaveData 클래스는 플레이어 스탯, 인벤토리 아이템, 씬 정보, 메타데이터를 포함한다.

### 2) ScriptableObject 기반 데이터 관리

로드 가능한 게임 데이터를 ScriptableObject로 관리한다.

PotionItemData : 포션의 이름, 아이콘, 회복량, 설명 텍스트를 저장

ShopItemData : 상점 아이템의 이름, 아이콘, 가격, 타입, 효과를 저장

CombatPage : 카드의 이름, 코스트, 쿨 타임, 주사위 값을 저장

### 3) DontDestroyOnLoad 씬 간 데이터 유지

씬이 전환되어도 특정 오브젝트를 유지하는 기술이다.

PlayerController, GameManager, SaveManager, Inventory, UICanvas에 적용했다.

### 4) 정적 클래스를 통한 크로스 씬 데이터 공유

씬 간에 간단한 데이터를 전달할 때 정적 변수를 사용한다.

GameData : 선택된 무기,

PortalReturnData : 포탈 복귀 정보

PortalController : 사용된 포탈을 추적

StatueInteraction : 이전 씬 이름 저장

BlacksmithMinigameManager : 시간 정지 플래그

DontDestroyOnLoadManager : 메인 메뉴 복귀 플래그.

## 2.4 UI/UX 시스템

### 1) Unity UI (Canvas, UI 이벤트 시스템)

Canvas 시스템을 사용하여 모든 UI를 구현했다.

Canvas 설정은 Render Mode의 설정으로 카메라를 독립적으로 만들었고, Canvas Scaler의 설정으로 해상도에 대응하도록 했다. Reference Resolution은 1920x1080으로, Match는 0.5로 설정하여 Width/Height 균형을 맞췄다.

### 2) UI 계층 구조 (Main UI Canvas)

- HUD Panel : 체력, 스탯
- Inventory Panel : 인벤토리 시스템
- Level Up Panel : 레벨 업 시스템
- Pause Menu Panel : 일시정지 시스템

### 3) 인벤토리 드래그 앤 드롭

인벤토리에서 아이템을 직접 드래그하여 사용하는 시스템으로 구현했다. 인터페이스를 사용하여 드래그 앤 드롭 이벤트를 처리한다.

### 4) 동적 UI 업데이트

플레이어 스탯 변경 시 UI를 실시간으로 업데이트한다. 옵저버 패턴을 활용하여 PlayerStats에서 경험치가 추가되면 UpdateStatsUI()를 호출하고, PlayerController를 통해 UpdateAllStatsUI()를 호출하여, StatsUIManager에서 모든 UI를 업데이트한다.

### 5) 레벨 업 시스템 UI

레벨 업 시 스탯을 선택할 수 있는 UI를 표시한다. 경험치가 최대치에 도달하면 레벨 업이 발생하고, 시간을 정지한다. 레벨 업 패널을 표시하면 플레이어가 공격력, 방어력, 체력, 이동 속도 중 하나를 선택한다. 선택이 적용 되고 패널이 닫히면 시간을 재개한다.

미니게임 중 레벨 업 처리는 특별하게 구현했다. AncientBlacksmith 또는 LifeHeartMap 씬에서는 레벨 업 UI를 즉시 표시하지 않고 isLevelUpPending 플래그를 true로 설정한다. 미니게임 종료 후 ShowPendingLevelUpPanel()을 호출하여 대기 중이던 레벨 업 UI를 표시한다.

Time.timeScale 관리는 0으로 설정하면 게임이 일시 정지하도록 설정했다. 1로 설정하면 게임이 정상적으로 작동한다. Time.unscaledDeltaTime은 timeScale의 영향을 받지 않는 시간으로 UI 애니메이션에 사용한다.

### 3. 구현 내용

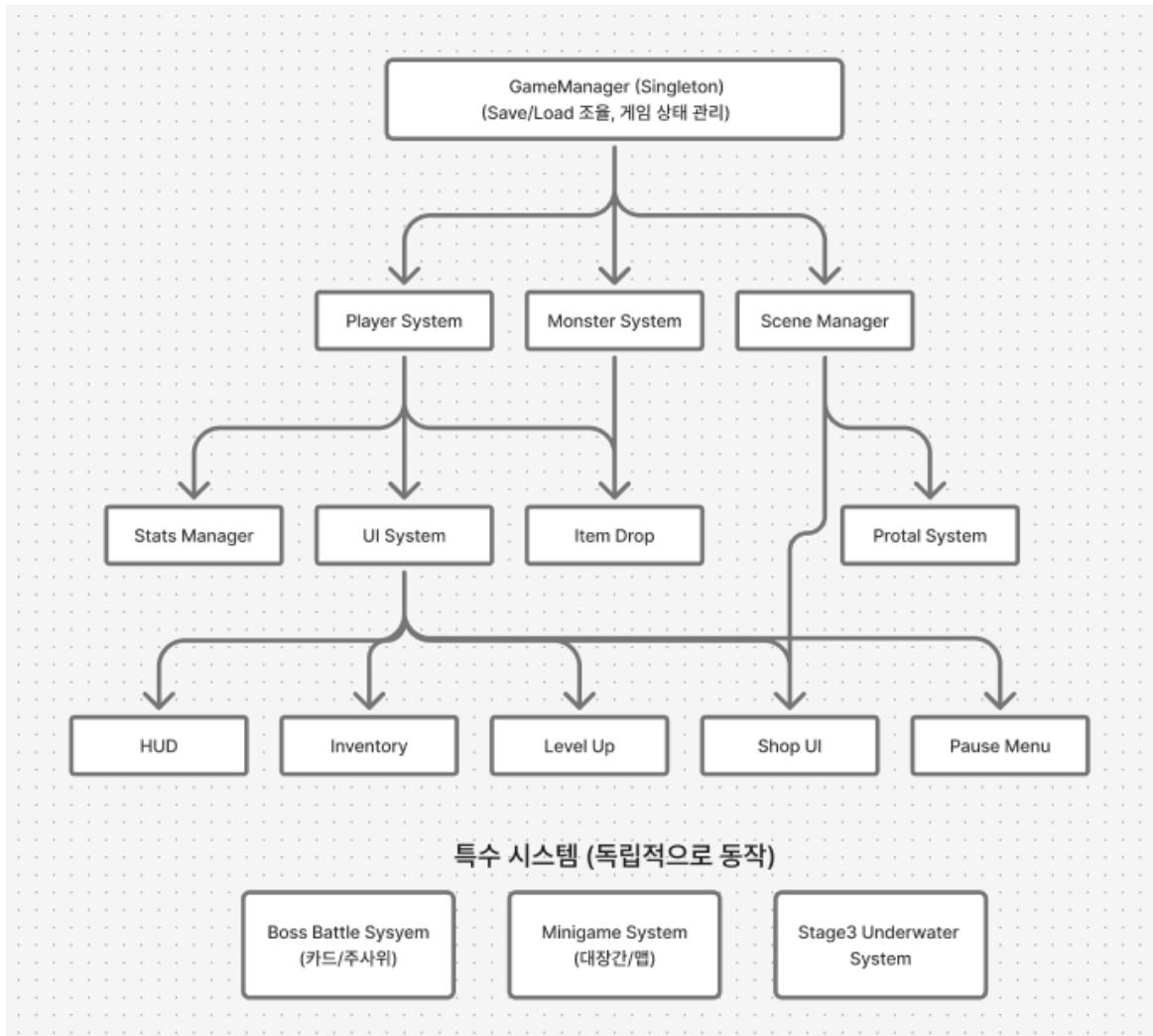
#### 3.1 게임 시스템 아키텍처

##### 1) 전체 시스템 구조

본 프로젝트는 11개의 주요 시스템으로 구성된다.

최상위는 GameManager이며 Save/Load 조율과 게임 상태 관리를 담당한다. GameManager 아래는 Player System, Monster System, Scene Manager가 있다. Player System은 Stats Manager와 UI System과 연결되며, Monster System은 Item Drop과 연결되고, Scene Manager는 Portal System과 연결된다.

UI System 아래는 HUD, Inventory, Level Up UI, Shop UI, Pause Menu가 있다. 특수 시스템으로는 Boss Battle System, Minigame System, Save System 그리고 Stage3 Underwater System이 있다.



## 2) 데이터 흐름

PlayerInput은 PlayerController로 전달되고, PlayerController는 Rigidbody2D를 통해 물리 처리를 하고 Animator를 통해 애니메이션을 처리한다.

PlayerController는 PlayerStats와 연결되어 LevelUpUIManager를 업데이트하고, PlayerHealth와 연결되어 HUD UI를 업데이트한다.

PlayerHealth는 DamageCalculation을 수행하고, 이는 MonsterHealth에 전달 되어 ItemDrop으로 이어진다. MonsterHealth는 Death Animation을 실행하고 플레이어에게 경험치를 지급한다.

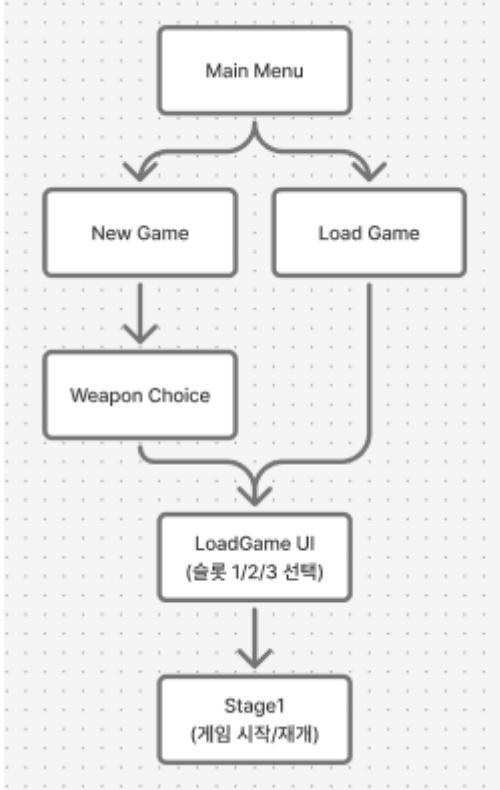
## 3) 씬 전환 구조

Main(메인 메뉴)에서 New Game와 Load Game으로 나뉜다.

New Game : Weapon(무기 선택)→LoadGame(슬롯 선택)→Stage1(게임 시작)

Load Game : LoadGame(슬롯 선택) → [저장된 씬](게임 재개)

Stage Flow는 다음과 같다. 각 스테이지에서 포탈을 통해 상점 씬으로 이동을 하고 각 상점 씬에서 포탈을 통해 보스 씬으로 이동한다.



## 4) 핵심 클래스 구조

PlayerController는 싱글톤이며 DontDestroyOnLoad를 사용하여 멤버 변수로 Instance, rb(Rigidbody2D), animator(Animator), hasSword, hasLance, hasMace 등을 가지고 있으며, 메서드는 Move(), Jump(), Dash(), Attack(), TakeDamage

(), RecalculateStats(), UpdateAllStatsUI() 등을 제공한다.

PlayerController는 PlayerStats, PlayerHealth, CharacterStats 컴포넌트를 요구하는데 PlayerStats는 level, currentXP, money, bonusAttackPower, bonusDefensePower, bonusMoveSpeed 등의 변수와 AddXP(), LevelUp(), AddMoney() 등의 메서드를 가진다. PlayerHealth는 maxHP, currentHP, defense 등의 변수와 TakeDamage(), Heal(), Die() 등의 메서드를 가지며, 피격 시에는 Invincibility Frames를 적용한다. CharacterStats는 deck, light, hp 등의 변수와 UsePage(), DrawCard() 등의 메서드를 가지며, LevelUpUI와 연결된다.

적 AI 시스템 클래스 구조는 MonsterController라는 일반 몬스터 AI의 베이스 클래스로, currentState(EnemyState), target(Transform), detectionRange, attackRange, moveSpeed 등의 변수와 UpdateState(), Patrol(), Chase(), Attack(), CheckGround(), CheckWall() 등의 메서드를 가진다.

Stage1MidBossAI는 MonsterController를 상속받아 CastSpell(), SummonSpell() 메서드를 추가한다. 마찬가지로 Stage2MidBossAI는 MonsterController를 상속받아 Teleport(), TeleportTo() 메서드를 추가한다.

MonsterHealth는 currentHP, maxHP, defense 등의 변수와 TakeDamage(), Die(), DieRoutine() 등의 메서드를 가진다. MonsterHealth는 ItemDrop과 연결되어 아이템 드롭을 처리한다.

### 5) 데이터베이스 스키마 (저장 데이터 구조)

SaveData는 JSON 형식으로 저장된다. 주요 필드는 슬롯 번호, 빈 슬롯 여부, 저장 시간, 플레이 시간, 플레이어 레벨, 현재 경험치, 스탯, 돈, 무기 상태, 아이템 목록, 현재 씬, 플레이어 위치를 포함한다.

PlayerStats 데이터 구조는 레벨 시스템은 레벨, 현재 경험치를 포함한다. 재화는 골드이다. 메서드는 AddXP(경험치 추가), AddMoney(돈 추가), SpendMoney(돈 사용), LevelUp(레벨업)이 있다.

Inventory 데이터 구조는 인벤토리 아이템 목록, 인벤토리 크기, 아이템 변경 콜백, 아이템 추가, 아이템 제거, 인벤토리 가득 참 여부가 있다.

Auto-Save는 씬 전환 시 OnSceneLoaded 이벤트 발생시 Main, LoadGame, Weapon, HowToPlay, Setting 씬을 제외한 씬에서, 0.5초 지연 후 자동으로 저장이 실행된다.

## 3.2 주요 시스템 구현

### 3.2.1 플레이어 시스템

#### 1) 이동 및 점프 메커니즘

플레이어의 기본 이동과 점프는 Rigidbody2D를 직접 제어하여 구현했다. 이동은 기본 속도와 보너스 속도를 합산한 후에 적용한다. 이동 속도에 따라 애니메이터의 Speed 파라미터를 업데이트하고, 이동 방향에 따라 스프라이트를 좌우 반전시킨다.

점프는 K 키 입력과 isGrounded 체크를 통해 구현한다. 점프 조건이 만족되면 점프 애니메이션을 실행한다.

#### 2) 대시 및 벽 슬라이드

대시는 코루틴을 사용하여 짧은 시간 동안 빠르게 이동하도록 구현했다. L 키 입력을 통해 대시 코루틴을 시작한다. 대시 중에는 중력을 0으로 설정하고, 캐릭터가 바라보는 방향으로 이동한다. 대쉬 후에 중력을 복원 한다. 대시 중에는 Dash 애니메이션을 실행한다.

벽 슬라이드는 다음 조건이 만족 될 때 실행된다. 벽에 닿아있고 땅에 닿아있지 않고, 하강 중일 때 벽 슬라이드가 활성화된다. 벽 슬라이드 중에는 하강 속도를 제한하고 Wall 애니메이션을 실행한다.

#### 3) 무기 시스템 (검, 창, 헤이스)

각 무기는 고유한 공격 패턴과 데미지 계산을 가지고 있다.

검은 추가 공격력 10과 사거리 1.5f, 대쉬 시 무적을 가진다. 공격 범위 내의 적을 감지하여 각 적에게 데미지를 입힌다.

창은 추가 공격력 8, 사거리 2.5f, 대쉬 시 공격 판정을 가진다. 전방 범위 공격을 공격 방향의 모든 적을 감지 한다. 검보다 데미지가 낮지만 긴 사거리로 안전한 거리를 유지할 수 있다.

헤이스는 추가 공격력 15, 사거리 1.2f, 대쉬 시 넉백 효과를 가진다. 높은 데미지와 함께 넉백 효과를 제공한다.

공격 실행은 J 키 입력을 통해 시작한다. Attack 애니메이션을 실행한다.

#### 4) 체력 및 스텝 관리

PlayerHealth는 maxHP 100, currentHP(시작 시 maxHP로 초기화) 그리고 defense 0을 가진다.

무적 상태이면 데미지를 무시한다. 데미지는 Max(1, damage - defense)로 계산하여 최소 1 데미지를 보장한다.

사망 시 Death 애니메이션을 실행하고 테드 씬으로 이동하며 게임 오버 처리한다.

## 5) 레벨업 시스템

PlayerStats는 level 1로 시작하며, 경험치를 통해 레벨 업을 한다. 레벨 업 후에도 남은 경험치가 있으면 계속 LevelUp()을 호출한다(while 루프).

레벨 업을 할 시 게임 시간이 멈추고 스탯 선택 창이 나타난다. 선택지에 IncreaseAttack은 공격력을 5 증가시킨다. IncreaseDefense는 방어력을 3 증가시킨다. IncreaseHealth 체력을 20 증가시킨다. IncreaseSpeed는 이동 속도를 0.5f 증가시킨다.

AddMoney는 골드를 증가시키고 SpendMoney는 골드가 아이템의 가격 이상 이면 골드를 감소시킨다.

## 6) 씬 로드 시 플레이어 복원

OnSceneLoaded는 다음과 같이 동작한다. 플레이어 스폰 오브젝트의 위치에서 생성이 되고 자동으로 저장이 실행된다.

자동 저장은 0.5초 대기 후, 현재 씬 이름을 가져온다.

### 3.2.2 적 AI 시스템

#### 1) 일반 몬스터 AI (순찰, 추적, 공격)

일반 적 AI는 상태 머신 패턴을 사용하여 구현했다. 대기, 이동, 추적, 공격, 피격, 사망 상태를 가진다.

일반 몬스터는 사망 하면 시간이 지나면 다시 생성하도록 설정하였다.

UpdateState는 플레이어와의 거리를 계산하고, 현재 상태에 따라 대기 상태에서 추적 상태로 변환 그후 추적 상태에서는 거리가 멀어지면 대기 상태로 공격 사거리 이내에 플레이어가 들어오면 공격 상태로 전환한다.

#### 2) AttackHitbox 구현

EnemyAttackHitbox는 AttackHitbox 오브젝트에 부착된다.

OnTriggerEnter2D는 충돌한 오브젝트의 레이어가 playerLayer와 일치하는지 체크 한다. 일치하면 공격을 통해 데미지를 입힌다.

#### 3) 스테이지 1 중간보스 (근접 + 마법)

Stage1 중간보스는 기본 AI를 확장하여 마법 공격 패턴을 추가했다.

크기를 일반 몬스터보다 훨씬 거대하게 설정을 하고 더 많은 스탯과 처리 시 더 많은 보상을 얻을 수 있다.

상태 전환은 일반 몬스터와 코드를 공유하여 일치한다.

SummonSpell은 함수를 통하여 SpellAttack 컴포넌트를 가져와 플레이어를 타겟으로 설정하고 특별한 공격을 실행한다.

#### 4) SpellAttack.cs (투사체)

SpellAttack에 SetTarget은 target을 설정하고, 플레이어 방향을 계산하여 스펠의 회전을 direction에 맞춰 공격을 하고 애니메이션이 실행된다.

OnTriggerEnter2D를 통해 충돌한 오브젝트의 태그가 플레이어 일때 데미지를 입힌다.

#### 5) 스테이지 2 중간보스 (텔레포트)

Stage2 중간보스는 플레이어와의 거리가 멀어지면 텔레포트하여 접근한다. 텔레포트는 설정 한 거리 이상 멀어지면 실행된다. 플레이어와의 거리를 계산하고, 거리가 공격 사거리 이내이면 공격을 실행한다.

MidBossAnimationEvents.cs (Sprite Child 패턴)

MidBossAnimationEvents는 Sprite 자식 오브젝트에 부착하여 애니메이션 이벤트를 부모 AI에 전달한다. 스테이지1과 2의 중간보스가 이에 해당한다.

#### 6) 체력 및 피격 시스템

MonsterHealth는 기본 값으로 체력 100, 방어력 0을 가진다. 그리고 각각 처치 시 얻을 수 있는 경험치와 일정 확률로 아이템을 드롭한다.

몬스터가 사망하면 1초 후 사망 애니메이션을 재생한다. itemDrop이 존재하면 DropItems()를 호출하여 아이템과 골드를 드롭한다.

#### 7) 아이템 드롭 시스템

ItemDrop은 몬스터를 처치 시에 아이템을 얻을 수 있게 하는 코드이다. 드롭 수량은 랜덤 값이다.

### 3.2.3 보스 전투 시스템 (주사위 기반 카드 전투)

보스 전투 시스템은 일반 전투와 달리 턴 기반 카드 전투 방식을 채택했다. Library of Ruina라는 게임의 주사위 시스템에서 영감을 받아 구현했다.

#### 1) 시스템 개요

보스 전투는 BossGameManager 싱글 톤이 게임 상태를 관리하며, Battle 상태에서는 PlayerController의 이동을 비활성화한다.

#### 2) 핵심 컴포넌트

BattleController는 전투의 주요 흐름을 관리한다. 플레이어가 최대 3장의 카드를 선택하면 보스도 랜덤하게 카드를 선택한다. 선택을 완료한 후에 실행을 하면 캐릭터들이 충돌 포인트로 이동한다. 각 합을 진행 할 때마다 주사위를 굴려 결과를 해결하고, 모든 합이 끝나면 캐릭터들은 각자의 홈 포지션으로 돌아간다. 카드의 쿨타임이 적용되고 새로운 턴이 시작된다.

CharacterStats 컴포넌트는 플레이어와 보스에게 적용한다. CombatPage 카드들의 텍, 카드를 사용하기 위한 자원, 현재 HP와 최대 HP를 관리한다. 카드 쿨타임 시스템은 카드를 사용한 후 일정 턴이 지난 후에 사용할 수 있도록 제한한다. 보스가 사용하는 텍은 공개 메커니즘을 통해 플레이어가 보스의 카드를 확인할 수 있다.

CombatPage는 카드 데이터인 카드 이름과 코스트 비용, 쿨타임, 주사위 배열(공격 또는 방어 주사위)을 포함한다.

CombatDice는 공격과 방어 타입을 가지며, 최소값과 최대값 사이에서 랜덤하게 주사위를 굴린다.

ClashManager는 정적 유ти리티 클래스로 플레이어와 보스의 주사위를 굴리고 합산한다. 주사위 값을 비교하여 패자에게 데미지를 적용한다.

### 3) 전투 흐름

전투는 다음과 같은 순서로 진행된다.

첫째, 플레이어가 카드를 선택하고 Space 바로 확인한다.

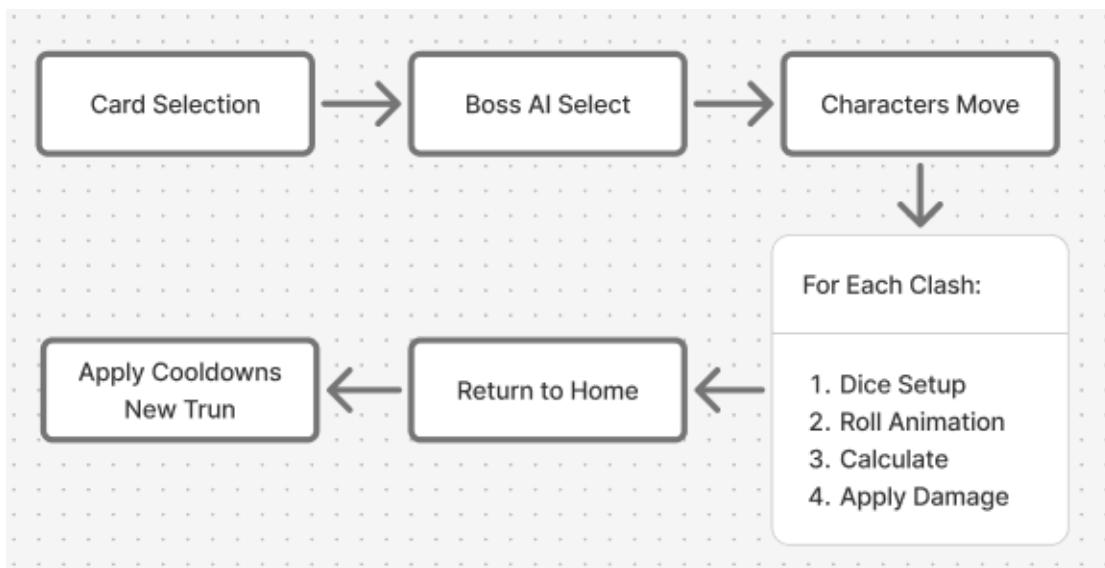
둘째, 보스가 사용 가능한 코스트 기반으로 랜덤하게 카드를 선택한다.

셋째, 충돌 위치로 이동한다.

넷째, 각 주사위 애니메이션을 실행하고 데미지를 해결한다.

다섯째, 캐릭터들이 홈 포지션으로 돌아간다.

여섯째, 카드 쿨다운이 적용되고 새로운 턴이 시작된다.



#### 4) 주사위 애니메이션 시스템

DiceVisual 컴포넌트는 각 주사위의 UI 표현을 담당한다. 숫자 사이클링과 회전으로 룰 애니메이션을 구현했고, 공격은 빨간색, 방어는 파란색으로 타입 별 색상을 적용했다. Win/Loss/Draw 상황에 따라 깜빡임, 확대, 흔들림 애니메이션을 실행한다. 주사위 굴림과 결과에 사운드 이펙트를 추가했다.

DiceAnimationManager는 합 애니메이션을 총괄한다. playerDiceContainer와 bossDiceContainer에 주사위 비주얼을 생성한다. 딜레이를 두고 주사위를 순차적으로 굴리는 애니메이션을 실행한다. 합산 결과와 합의 결과를 표시하고, ClashManager의 데미지 계산 로직을 사용하여 데미지를 적용한다. 주사위 개수가 다를 때 일방적인 공격을 처리한다.

중요한 점은 DiceAnimationManager가 BattleController에 반드시 할당되어야 애니메이션이 작동한다는 것이다. 할당되지 않으면 애니메이션 작동 없이 전투가 진행된다.

#### 5) 데미지 계산

ClashManager와 DiceAnimationManager는 동일한 데미지 계산을 사용한다. 공격 주사위가 이기면 데미지 계산을 하고 방어 주사위가 합에서 승리하면 (승자의 주사위 값 - 패자의 주사위 값)으로 계산한다. 두 시스템의 공식이 일치해야 일관된 게임플레이를 보장할 수 있다.

#### 6) 주요 메커니즘

PlayerController는 BossGameManager가 Battle일 때만 실행이 가능하다. 보스 텍 보기 는 플레이어 카드와 보스 공개 카드 사이를 전환한다. 승리 시 10초 지연 후 씬을 이동한다.

##### 3.2.4 씬 관리 및 포털 시스템

씬 전환과 포털 시스템은 게임의 비선형적인 탐험을 가능하게 한다.

###### 1) 포탈 시스템

PortalController는 씬 전환을 확인 UI와 함께 처리한다. 플레이어가 포탈에 진입하면 확인 패널을 표시하고, 플레이어의 선택에 따라 씬을 로드하거나 취소한다. 포탈은 사용 후 자동으로 파괴된다.

Ancient/Life 맵에서는 특수 포털이 사용된다. 일반 포털과 유사하지만 미니게임 전용으로 특화되어 있다. 정적 변수에 이전 씬 이름을 저장하여 미니게임 종료 후 돌아갈 수 있게 한다.

PortalReturnManager는 미니게임이나 특수 씬을 떠난 후 플레이어의 복귀 위치를 관리한다.

## 2) 데이터 지속성 패턴

포털 시스템은 정적 데이터 클래스를 사용하여 씬 간 데이터를 유지한다.

PortalReturnData는 복귀 위치와 씬 정보를 저장하는 정적 클래스다.

GameData.SelectedWeapon은 무기 선택을 저장한다.

## 3) 플레이어 스폰 시스템

PlayerSpawnPoint는 씬에 배치되는 빈 GameObject로 "PlayerSpawnPoint" 태그를 가진다. PlayerController는 씬이 로드되면 PlayerSpawnPoint를 찾아 플레이어를 해당 위치로 이동시킨다. 포털은 returnSpawnPoint를 참조하여 플레이어가 돌아올 위치를 지정한다.

## 4) 씬 전환 플로우

일반적인 씬 전환은 플레이어가 포털에 진입하면 확인 UI가 표시된다. 플레이어가 확인하면 PortalReturnData에 현재 위치를 저장, GameManager가 자동 저장을 수행한다. SceneManager로 새 씬을 로드한다. 새로운 씬에서 PlayerController가 호출되고, PlayerSpawnPoint 위치로 플레이어가 이동한다. 스탯이 재계산되고 UI가 업데이트된다.

미니게임 복귀 플로우는 PortalReturnManager에 현재 씬을 저장한 후에 미니게임 씬으로 전환한다. 미니게임이 종료되면 원래 씬으로 돌아간다.

### 3.2.5 세이브/로드 시스템

세이브/로드 시스템은 JSON 기반의 3슬롯 저장 방식을 사용한다.

## 1) 시스템 아키텍처

SaveManager는 파일 I/O만 담당하는 지속성 계층이다.

SaveGame()은 SaveData를 JSON으로 직렬화하여 파일에 쓴다.

LoadGame()은 파일에서 JSON을 읽어 SaveData로 역직렬화한다.

DeleteSave()는 슬롯의 저장 파일을 삭제한다.

GetAllSaveSlots()는 3개 슬롯의 SaveData 배열을 반환한다.

GameManager는 비즈니스 로직을 담당하는 게임 상태 조율 계층이다.

현 게임 상태를 SaveData로 저장하고 SaveData를 복원하고 씬을 로드한다.

SaveData는 직렬화 가능한 데이터 클래스로, 플레이어 스탯, 인벤토리, 씬 정보, 메타데이터를 포함한다.

## 2) 저장 프로세스

플레이어가 저장을 요청하거나 자동 저장이 트리거되면 GameManager가 호출되고, PlayerController, PlayerStats, PlayerHealth, Inventory에서 데이터를

수집한다. SaveData 객체를 생성하고 모든 필드를 채운다. SaveManager가 SaveData를 JSON으로 직렬화하여 데이터를 저장한다

### 3) 로드 프로세스

플레이어가 슬롯을 선택하여 로드하면 LoadGameUI에서 GameManager가 SaveManager로 SaveData를 로드한다. SaveData가 비어있으면 데이터가 없기 때문에 에러 처리한다. PlayerController, PlayerStats, PlayerHealth를 찾아 SaveData의 값을 적용한다. Inventory의 아이템을 복원한다. SceneManager로 저장된 씬을 로드한다. PlayerController에서 saveData로 플레이어 위치를 설정한다.

### 4) 자동 저장 시스템

자동 저장은 씬 전환 시 OnSceneLoaded 이벤트가 발생하고 0.5초 지연 후 실행된다. Main, LoadGame, Weapon, HowToPlay, Setting 씬은 제외된다. 플레이어 위치는 PlayerSpawnPoint 위치로 저장된다.

### 5) 슬롯 관리

3개의 슬롯이 독립적으로 관리된다. SaveSlotButton UI는 슬롯 번호, 저장 정보(레벨, 씬, 시간), 빈 슬롯 텍스트를 표시한다. 선택/삭제 액션을 처리한다.

#### 3.2.6 상점 시스템

상점 시스템은 랜덤화 된 인벤토리와 새로고침 메커니즘을 가진 동적 아이템 구매 시스템이다.

### 1) 시스템 구성

ShopManager가 상점 시스템을 관리한다. 제시되는 아이템 풀에서 4개의 아이템을 중복 없이 랜덤 선택한다. PlayerStats와 통합하여 구매를 검증하고 처리한다. 새로고침마다 증가하는 동적 새로고침 비용을 구현했다. 구매, 새로고침, 에러에 대한 사운드 이펙트를 제공한다.

ShopItemData는 아이템의 종류를 나타낸다. ApplyEffect() 메서드가 스탯 보너스를 적용하거나 인벤토리에 포션을 추가한다. 아이템 이름, 아이콘, 가격, 타입, 효과값, 아이콘 스케일을 설정할 수 있다. 포션 타입의 경우 PotionItemData 참조를 포함한다.

ShopPedestal은 개별 아이템 표시대로 구매 상호작용을 처리한다. 상점 새로고침 기능을 위한 특수 표시대가 있다. ExchangePedestal은 돈이 아닌 아이템 교환을 위한 특수 표시대다.

## 2) 아이템 효과

각 아이템 타입은 체력 혹은 경험치를 올려주는 포션과 플레이어의 스탯(공격력, 방어력, 체력, 이동속도)을 상승시키는 아이템으로 구분 되어있다.

## 3) 구매 프로세스

플레이어가 아이템을 선택하면 ShopManager가 PlayerStats로 골드를 확인하고 차감하면서 검증한다. 검증 실패 시 에러를 재생하고 리턴한다. 검증 성공 시 구매 사운드를 재생하고 표시대에서 아이템을 제거한다.

## 4) 새로고침 메커니즘

플레이어가 새로고침 표시대를 활성화하면 다음과 같이 진행된다. 현재 새로고침 비용을 계산한다. PlayerStats로 비용을 검증하고 차감한다. 검증 실패 시 에러를 재생한다. 검증 성공 시 기존 전시된 아이템을 모두 제거, 중복 없이 4개의 새 아이템을 랜덤으로 선택한다. 새 아이템을 표시대에 할당하고 refreshCount를 증가시킨다. 새로고침 사운드를 재생한다.

### 3.2.7 인벤토리 시스템

인벤토리 시스템은 아이템 관리와 사용을 담당한다.

#### 1) 시스템 구조

Inventory 싱글 톤이 인벤토리 데이터를 관리한다. 인벤토리의 공간을 20으로 최대 용량을 제한한다. OnItemChanged를 통해 UI 업데이트 콜백을 제공한다.

InventoryUI는 인벤토리 패널과 슬롯을 관리한다. 아이템을 획득 할 시에 자동으로 UI를 업데이트한다. 슬롯을 관리하고 아이템 아이콘과 수량을 표시한다.

InventorySlot은 개별 슬롯 UI를 구현한다. 드래그 앤 드롭을 지원하여 드래그 시작 시 슬롯에서 아이템을 다른 슬롯으로 이동할 수 있고 클릭을 통해 아이템을 사용할 수 있다.

#### 2) 아이템 획득

ItemPickup은 플레이어가 아이템과 충돌 시 플레이어와 아이템의 충돌을 감지하고, Inventory를 호출하여 아이템을 추가하고 오브젝트를 파괴한다. 인벤토리가 가득 차면 에러 메시지를 표시한다.

Coin은 화폐 핵심을 처리한다. PlayerStats를 호출하여 돈을 추가하고, 코인 오브젝트를 파괴한다

### 3) 아이템 사용

플레이어가 클릭을 하여 포션을 사용하면 InventorySlot에서 플레이어에게 드롭한다. PlayerHealth를 호출하여 체력 혹은 경험치를 얻는다. Inventory를 호출하여 사용한 아이템을 인벤토리에서 제거한다.

#### 3.2.8 미니게임 시스템

미니게임 시스템은 메인 게임플레이와 연결된 미니게임을 제공한다.

##### 1) 미니게임 (AncientBlacksmith)

BlacksmithMinigameManager 싱글 톤이 대장간 미니게임을 관리한다. 게임 시작 전 DialogueController를 통해 대화를 진행한다. 점수 기반 보상 시스템을 CalculateAndGrantRewards()에 구현했다. 보상은 골드와 경험치, 추가 공격력으로 구성된다.

미니게임이 시작되면 불꽃 수집 게임을 진행한다. 미니게임이 끝날 시에 점수를 계산하고 보상을 지급한다. 그 후에 이전 씬으로 복귀한다.

보상 계산은 점수 구간별로 다르다. 높은 점수일수록 더 많은 골드, 경험치, 공격력 보너스를 받는다. 보상은 PlayerStats에 직접 적용된다.

##### 2) 라이프맵 미니게임 (LifeHeartMap)

LifeGameManager 싱글 톤이 LifeHeartMap 씬을 관리한다.

##### 3) 미니게임 공통 메커니즘

미니게임 중 레벨업이 발생하면 PlayerStats에서 레벨업 UI를 연기시킨다. 미니게임 종료 후 원래 씬으로 돌아가면 ShowPendingLevelUpPanel()이 호출되어 대기 중인 레벨업 UI를 표시한다.

#### 3.2.9 스테이지 3 수중 시스템

Stage3는 독특한 수영 메커니즘과 산소 관리를 특징으로 하는 씬이다.

##### 1) 시스템 관리

Stage3Manager가 씬 매니저로서 스테이지마다 특정 컴포넌트를 활성화/비활성화한다. OnEnable에서 PlayerSwimming과 PlayerOxygen을 활성화한다. OnDestroy에서 두 컴포넌트를 비활성화하여 다른 씬의 간섭을 방지한다. 이 패턴을 통해 플레이어 GameObject가 스테이지 별 스크립트를 충돌 없이 운반할 수 있다.

##### 2) 수영 메커니즘

PlayerSwimming은 표준 이동을 수영 메커니즘으로 대체한다. 일반적으로 플레이어에 부착되어 있지만 Stage3까지 비활성화된다. 수중에서는 중력이 감소하고 8방향 이동이 가능하다. 수직 이동을 지원한다.

### 3) 산소 시스템

PlayerOxygen은 시간에 따라 감소하는 산소 게이지를 관리한다.

Stage3에서만 활성화하는 컴포넌트로 산소가 다 떨어지면 플레이어가 지속적으로 데미지를 받는다. oxygenSlider UI를 참조하여 현재 산소량을 표시한다. Stage3을 떠날 때 oxygenSlider는 비활성화되도록 한다.

OxygenZone은 플레이어가 들어가면 산소를 회복하는 영역이다. 방울에서 지속적으로 산소를 회복한다.

### 4) 수중 환경 요소

Stage3의 수집 가능한 아이템으로 진주가 있다. PearlDisplayUI는 진주 수집 개수를 표시하는 UI다. 총 진주 갯수와 수집한 진주 갯수를 표시한다. GiantClam은 상호작용 가능한 조개 몬스터이다.

ClamMonsterController는 수중 조개 몬스터 전용 AI다. MonsterController와 유사하지만 수중 이동에 특화되어 있다.

### 5) 통합 메커니즘

플레이어는 Stage3Manager가 찾을 수 있도록 "Player" 태그를 가져야 한다. PlayerOxygen은 oxygenSlider UI를 참조하는데, Stage3을 떠날 때 비활성화 되어야 한다. 스테이지 별 로직 컴포넌트는 기본적으로 비활성화되어 있고, Stage3Manager가 씬 시작 시 활성화하고 씬 종료 시 비활성화한다.

## 3.2.10 UI 시스템

UI 시스템은 플레이어와 게임의 모든 상호작용을 관리한다.

### 1) 메인 메뉴 시스템

MainMenuController는 Main 씬을 제어한다.

New Game 선택 : Weapon 씬으로 전환

Load Game 선택 : LoadGame 씬으로 전환.

메인 메뉴 복귀 : DontDestroyOnLoad 오브젝트를 정리 .

LoadGameUI는 3개 슬롯을 가진 세이브 슬롯 선택 UI다. 레벨과 씬, 시간을 세이브 슬롯에 표시한다. 기존 세이브에 대한 덮어쓰기 확인을 제공한다. SaveManager 및 GameManager와 통합된다.

### 2) HUD 시스템

StatsUIManager는 플레이어 스탯을 실시간으로 표시한다. PlayerController의 UpdateAllStatsUI()를 통해 업데이트된다. 체력 바, 경험치 바, 레벨, 공격력, 방어력, 이동 속도, 돈을 표시한다. CurrencyUI는 플레이어의 현재 돈을 표시한다.

### 3) 레벨업 UI

LevelUpUIManager는 레벨 업 패널과 스탯 업그레이드 옵션을 표시한다. 패널이 표시되는 동안 시간을 정지한다. 공격력, 방어력, 체력, 이동 속도 증가 버튼을 제공한다. 선택 시 PlayerStats의 해당 메서드를 호출, 패널을 닫는다.

### 4) 인벤토리 UI

InventoryUI는 인벤토리 패널과 슬롯을 관리한다. 아이템 아이콘, 이름, 수량을 표시한다.

InventorySlot은 드래그 앤 드롭을 지원하는 개별 슬롯이다. 플레이어가 아이템을 사용 할 시 아이템을 사용한다.

Tooltip은 아이템에 마우스를 올리면 표시되는 툴팁이다. TooltipManager가 툴팁 표시를 관리한다. 아이템 이름, 설명, 효과를 보여준다.

### 5) 일시정지 메뉴

PauseMenuUI는 ESC 키로 호출되는 일시정지 메뉴다. Time.timeScale로 일시정지/재개를 관리한다. 메인 메뉴로 돌아갈 때 싱글 톤을 정리한다. 재개, 설정, 메인 메뉴 버튼을 제공한다.

### 6) 지속성 관리

DontDestroyOnLoadManager는 씬의 오브젝트 지속성을 관리한다. instanceId 기반 Dictionary로 인스턴스를 추적한다. 같은 instanceId를 가진 오브젝트를 방지한다. 메인 메뉴 복귀 시 지속성을 비활성화한다.

## 3.3 시퀀스 다이어그램

### 3.3.1 게임 시작 플로우

#### 게임 시작부터 게임플레이까지의 시퀀스

Player -> MainMenuController: New Game 선택

MainMenuController -> SceneManager: LoadScene("Weapon")

Player -> WeaponChoice: 무기 선택 (검/창/메이스)

WeaponChoice -> GameManager: selectedWeapon 설정, isNewGame = true

WeaponChoice -> SceneManager: LoadScene("LoadGame")

Player -> LoadGameUI: 슬롯 선택

LoadGameUI -> GameManager: SelectSaveSlot(slotNumber)

LoadGameUI -> GameManager: 기존 세이브 확인

GameManager -> SaveManager: LoadGame(slotNumber)

SaveManager -> GameManager: SaveData 반환

GameManager -> SceneManager: LoadScene("Stage1")  
SceneManager -> PlayerController: OnSceneLoaded()  
PlayerController -> PlayerSpawnPoint: 위치 조회  
PlayerController: transform.position 설정  
PlayerController -> PlayerStats: RecalculateStats()  
PlayerController -> StatsUIManager: UpdateAllStatsUI()  
PlayerController -> GameManager: PerformAutoSave()  
New Game의 경우 선택된 무기와 기본 스탯으로 새 게임을 시작한다.  
Load Game의 경우 SaveData를 복원하여 이전 진행 상황에서 재개한다.

### 3.3.2 전투 시퀀스

#### 플레이어가 적을 공격하는 일반적인 전투 시퀀스

Player: J 키 입력  
PlayerController: AttackRoutine() 코루틴 시작  
PlayerController: isAttacking = true  
PlayerController -> Animator: SetTrigger("Attack")  
Animator -> PlayerController: AttackHit() 이벤트 호출  
PlayerController -> SwordAttack: Attack(attackType)  
SwordAttack: OverlapCircleAll로 적 감지  
SwordAttack -> MonsterHealth: TakeDamage(damage, attackType, "Sword")  
MonsterHealth: 데미지 계산 (무기 배수, 공격 타입 배수)  
MonsterHealth -> DamageText: ShowDamageText(finalDamage)  
MonsterHealth -> Animator: SetTrigger("isHurt")  
MonsterHealth: currentHP 체크  
MonsterHealth: if HP <= 0, Die()  
MonsterHealth -> MonsterController: enabled = false  
MonsterHealth -> PlayerStats: AddXP(xpValue)  
MonsterHealth: DieRoutine() 코루틴 시작  
MonsterHealth: 1초 대기 (사망 애니메이션)  
MonsterHealth -> MidBossController: OnBossDeath() (중간보스인 경우)  
MonsterHealth -> ItemDrop: DropItems()  
ItemDrop: 각 DropItem 확률 체크  
ItemDrop: Instantiate(itemPrefab) 랜덤 위치  
ItemDrop -> Rigidbody2D: AddForce (튀어오름 효과)

MonsterHealth: Destroy(gameObject)  
PlayerController: attackDuration 대기  
PlayerController: isAttacking = false  
적이 사망하면 경험치 지급, 아이템 드롭, 이벤트 트리거로 처리된다.

### 3.3.3 보스 전투 시퀀스

#### 턴 기반 카드 전투 보스전의 시퀀스

Player: BattleTrigger 진입  
BattleTrigger -> BossGameManager: SetState(Battle)  
BossGameManager -> PlayerController: 이동 비활성화  
BattleController: 카드 선택 UI 표시  
Player: 카드 선택 (최대 3장)  
Player: Space 바로 확인  
BattleController -> CharacterStats(Boss): AI 카드 선택  
BattleController: StartClashPhase()  
BattleController -> CameraController: ZoomIn()  
BattleController -> CharacterVisuals: 클래시 위치로 이동  
BattleController -> DiceAnimationManager: SetupDiceVisuals()  
DiceAnimationManager: playerDiceContainer에 주사위 UI 생성  
DiceAnimationManager: bossDiceContainer에 주사위 UI 생성  
DiceAnimationManager: AnimateClashSequence()  
DiceAnimationManager -> DiceVisual: AnimateRoll() (각 주사위 순차적)  
DiceVisual: 숫자 사이클링 + 회전 애니메이션  
DiceVisual: 최종 값 표시  
DiceAnimationManager: 합산 표시 (플레이어 vs 보스)  
DiceAnimationManager: 클래시 결과 계산  
DiceAnimationManager -> DiceVisual: Win/Loss/Draw 애니메이션  
DiceAnimationManager: ApplyClashDamage()  
DiceAnimationManager -> CharacterStats: TakeDamage()  
DiceAnimationManager: 패배한 주사위만 파괴  
DiceAnimationManager: 다음 클래시 (주사위 개수만큼 반복)  
BattleController -> CharacterVisuals: 홈 포지션으로 복귀  
BattleController -> CameraController: ZoomOut()  
BattleController -> CharacterStats: 카드 쿨다운 적용

BattleController: 다음 턴 시작  
보스나 플레이어의 HP가 0이 되면 Victory() 또는 Defeat()가 호출되어 전투가 종료된다.

### 3.3.4 세이브/로드 시퀀스

#### 1) 자동 저장 시퀀스

SceneManager: 씬 로드 완료

PlayerController: OnSceneLoaded() 이벤트 수신

PlayerController: AutoSaveAfterDelay() 코루틴 시작

AutoSaveAfterDelay: 0.5초 대기

AutoSaveAfterDelay: 씬 이름 체크 (제외 씬인지 확인)

AutoSaveAfterDelay: currentSaveSlot > 0 체크

AutoSaveAfterDelay -> GameManager: PerformAutoSave()

GameManager -> GameManager: SaveCurrentGame()

GameManager -> PlayerController: 데이터 수집

GameManager -> PlayerStats: 데이터 수집

GameManager -> PlayerHealth: 데이터 수집

GameManager -> Inventory: 데이터 수집

GameManager: SaveData 객체 생성 및 채우기

GameManager: currentScene, playerPosition 설정

GameManager -> SaveData: UpdateSaveTime()

GameManager -> SaveManager: SaveGame(saveData, currentSaveSlot)

SaveManager: JsonUtility.ToJson(saveData)

SaveManager: File.WriteAllText(path, json)

#### 2) 로드 시퀀스

Player -> LoadGameUI: 슬롯 선택

LoadGameUI -> GameManager: LoadGameFromSlot(slotNumber)

GameManager: currentSaveSlot = slotNumber

GameManager -> SaveManager: LoadGame(slotNumber)

SaveManager: File.ReadAllText(path)

SaveManager: JsonUtility.FromJson<SaveData>(json)

SaveManager -> GameManager: SaveData 반환

GameManager: isEmpty 체크

GameManager -> GameManager: ApplySaveData(saveData)  
GameManager: PlayerController 찾기  
GameManager -> PlayerController: hasSword/hasLance/hasMace 설정  
GameManager -> PlayerStats: level, currentXP, money 등 설정  
GameManager -> PlayerHealth: maxHP, currentHP, defense 설정  
GameManager -> Inventory: 아이템 복원  
GameManager -> SceneManager: LoadScene(saveData.currentScene)  
SceneManager -> PlayerController: OnSceneLoaded()  
PlayerController: transform.position = saveData.playerPosition  
PlayerController: RecalculateStats()  
PlayerController: UpdateAllStatsUI()

### 3.3.5 씬 전환 시퀀스

#### 1) 포털을 통한 씬 전환 시퀀스

Player: 포털 트리거 진입  
PortalController: OnTriggerEnter2D()  
PortalController: confirmationPanel.SetActive(true)  
Player: 확인 버튼 클릭  
PortalController: OnConfirm()  
PortalController: usedPortalIDs.Add(portalID)  
PortalController -> PortalReturnData: 현재 위치 저장  
PortalReturnData: hasReturnInfo = true  
PortalReturnData: returnPosition = transform.position  
PortalReturnData: previousSceneName = currentScene  
PortalController -> GameManager: PerformAutoSave()  
GameManager: SaveCurrentGame()  
PortalController -> SceneManager: LoadScene(sceneToLoad)  
SceneManager: 새 씬 로드  
PlayerController: OnSceneLoaded()  
PlayerController: PlayerSpawnPoint 찾기  
PlayerController: transform.position = spawnPoint.position  
PlayerController: RecalculateStats()  
PlayerController: UpdateAllStatsUI()  
PlayerController: AutoSaveAfterDelay()

## 2) 미니게임에서 복귀하는 시퀀스

Player: 미니게임 완료

BlacksmithMinigameManager: EndGame()

BlacksmithMinigameManager: CalculateAndGrantRewards()

BlacksmithMinigameManager ->

PlayerStats: AddMoney(), AddXP(), bonusAttackPower 증가

BlacksmithMinigameManager: isGamePausedByManager = true

BlacksmithMinigameManager -> StatueInteraction: previousSceneName 참조

BlacksmithMinigameManager ->

SceneManager: LoadScene(previousSceneName)

SceneManager: 원래 쓴 로드

PlayerController: OnSceneLoaded()

PlayerController -> PortalReturnManager: hasReturnInfo 체크

PortalReturnManager: returnPosition 반환

PlayerController: transform.position = returnPosition

PlayerController -> PlayerStats: ShowPendingLevelUpPanel()

PlayerStats: isLevelUpPending = false

PlayerStats -> LevelUpUIManager: ShowPanel()

## 3.4 주요 알고리즘

### 3.4.1 데미지 계산 알고리즘

#### 1) 일반 적의 데미지 계산 알고리즘

입력:

- baseDamage: 기본 공격 데미지
- defense: 적의 방어력
- weaponType: "Sword", "Lance", "Mace" 중 하나
- attackType: 1 (약공격), 2 (강공격)

변수:

- weaponMultiplier: 무기 타입에 따른 배수
  - Sword: 1.0
  - Lance: 0.9
  - Mace: 1.2
- attackMultiplier: 공격 타입에 따른 배수
  - attackType == 1: 1.0 (약공격)
  - attackType >= 2: 1.5 (강공격)

계산:

##### 1. weaponMultiplier 결정

```
if weaponType == "Sword": weaponMultiplier = 1.0
else if weaponType == "Lance": weaponMultiplier = 0.9
else if weaponType == "Mace": weaponMultiplier = 1.2
```

##### 2. attackMultiplier 결정

```
if attackType == 1: attackMultiplier = 1.0
else: attackMultiplier = 1.5
```

##### 3. 최종 데미지 계산

```
finalDamage = RoundToInt((baseDamage - defense) × weaponMultiplier
× attackMultiplier)
```

##### 4. 최소 데미지 보장

```
finalDamage = Max(1, finalDamage)
```

출력: finalDamage

## 2) 보스 전투의 주사위 기반 데미지 계산 알고리즘

입력:

- playerPages: 플레이어가 선택한 카드 배열
- bossPages: 보스가 선택한 카드 배열
- playerAttackPower: 플레이어 공격력
- playerDefensePower: 플레이어 방어력
- bossAttackPower: 보스 공격력
- bossDefensePower: 보스 방어력

작 합 처리:

```
for i = 0 to Min(playerPages.Length, bossPages.Length) - 1:  
    playerPage = playerPages[i]  
    bossPage = bossPages[i]  
    // 주사위 굴리기  
    playerRollTotal = 0  
    for each dice in playerPage.dices:  
        roll = Random.Range(dice.minValue, dice maxValue + 1)  
        playerRollTotal += roll  
    bossRollTotal = 0  
    for each dice in bossPage.dices:  
        roll = Random.Range(dice.minValue, dice maxValue + 1)  
        bossRollTotal += roll  
    // 탑 우위 계산  
    playerAdvantage = (playerPage.dices[0].type == Attack && bossPage.dices[0].type == Defense)  
    bossAdvantage = (bossPage.dices[0].type == Attack && playerPage.dices[0].type == Defense)  
    // 데미지 해결  
    if playerRollTotal > bossRollTotal:  
        if playerAdvantage:  
            damage = Max(1, (playerAttackPower + playerRollTotal) - bossDefensePower - bossRollTotal)  
            boss.TakeDamage(damage)  
        else:  
            counterDamage = playerRollTotal - bossRollTotal
```

```

        boss.TakeDamage(counterDamage)
    else if bossRollTotal > playerRollTotal:
        if bossAdvantage:
            damage = Max(1, (bossAttackPower + bossRollTotal) - player
DefensePower - playerRollTotal)
            player.TakeDamage(damage)
        else:
            counterDamage = bossRollTotal - playerRollTotal
            player.TakeDamage(counterDamage)
    일방적인 공격 (주사위 개수가 다를 때):
    if playerPages.Length > bossPages.Length:
        for i = bossPages.Length to playerPages.Length - 1:
            playerPage = playerPages[i]
            playerRollTotal = 주사위 굴리기 합산
            damage = Max(1, (playerAttackPower + playerRollTotal) - bossDef
ensePower)
            boss.TakeDamage(damage)
    else if bossPages.Length > playerPages.Length:
        (보스도 동일하게 처리)
        이 알고리즘은 ClashManager와 DiceAnimationManager 모두에서 동일하게
구현되어 일관성을 보장한다.

```

### 3.4.2 적 AI 상태 전환 알고리즘

#### 1) 적 AI의 상태 머신 알고리즘

**입력:**

- currentState: 현재 AI 상태
- playerPosition: 플레이어 위치
- enemyPosition: 적 위치
- detectionRange: 감지 범위 (예: 5f)
- attackRange: 공격 범위 (예: 1.5f)

## 매 프레임 실행:

### 1. 거리 계산

```
distance = Vector2.Distance(enemyPosition, playerPosition)
```

### 2. 상태별 전환 규칙

```
switch currentState:
```

```
    case Patrol:
```

```
        if distance <= detectionRange:
```

```
            ChangeState(Chase)
```

```
    case Chase:
```

```
        if distance > detectionRange:
```

```
            ChangeState(Patrol)
```

```
        else if distance <= attackRange && !isAttacking:
```

```
            ChangeState(Attack)
```

```
    case Attack:
```

```
        if distance > attackRange:
```

```
            ChangeState(Chase)
```

```
    case Hurt:
```

```
        // 피격 애니메이션 완료 후 자동으로 Chase 또는 Patrol로 복귀
```

```
    case Dead:
```

```
        // 더 이상 상태 전환 없음
```

### 3. 상태별 실행

```
switch currentState:
```

```
    case Patrol:
```

```
        Patrol() 실행
```

```
        - currentTarget 방향으로 이동
```

```
        - 목표 도달 시 currentTarget 전환
```

```
        - 벽/낭떠러지 감지 시 currentTarget 전환
```

```
    case Chase:
```

```
        Chase() 실행
```

```
        - playerPosition 방향으로 이동
```

```
        - 스프라이트 방향 조정
```

```
    case Attack:
```

```
        if !isAttacking:
```

```
            AttackRoutine() 코루틴 시작
```

- isAttacking = true
- 정지
- 공격 애니메이션
- AttackHitbox 활성화 (애니메이션 이벤트)
- attackCooldown 대기
- AttackHitbox 비활성화
- isAttacking = false

## 2) Stage2 중간보스의 텔레포트 알고리즘

### 입력:

- playerPosition: 플레이어 위치
- enemyPosition: 적 위치
- teleportRange: 텔레포트 트리거 거리 (예: 6f)
- teleportMinX, teleportMaxX: 텔레포트 가능 X 범위
- teleportY: 텔레포트 Y 좌표

### 매 프레임 실행:

1. 거리 계산  
distance = Vector2.Distance(enemyPosition, playerPosition)
2. 텔레포트 조건 체크  
if distance > teleportRange && canTeleport:  
    TeleportRoutine() 코루틴 시작

### TeleportRoutine():

1. canTeleport = false
2. 정지 (linearVelocity = Vector2.zero)
3. FadeOut() 실행 (0.5초 동안 alpha 1→0)
4. 새 위치 계산:  
    randomX = Random.Range(teleportMinX, teleportMaxX)  
    while Abs(randomX - playerPosition.x) < 2f:  
        randomX = Random.Range(teleportMinX, teleportMaxX)  
        newPosition = (randomX, teleportY)
5. transform.position = newPosition
6. FadeIn() 실행 (0.5초 동안 alpha 0→1)
7. teleportCooldown 대기 (예: 3초)
8. canTeleport = true

### 3.4.3 경로 탐색 알고리즘

#### 1) 순찰 경로 알고리즘

입력:

- patrolPointA: 순찰 지점 A
- patrolPointB: 순찰 지점 B
- currentTarget: 현재 목표 지점
- moveSpeed: 이동 속도

매 프레임 실행:

1. 방향 계산

```
direction = Normalize(currentTarget.position - enemyPosition)
```

2. 이동

```
linearVelocity.x = direction.x * moveSpeed
```

3. 스프라이트 방향 조정

```
if direction.x > 0:
```

```
    transform.localScale = (1, 1, 1)
```

```
else:
```

```
    transform.localScale = (-1, 1, 1)
```

4. 목표 도달 체크

```
if Distance(enemyPosition, currentTarget.position) < 0.5f:
```

```
    if currentTarget == patrolPointA:
```

```
        currentTarget = patrolPointB
```

```
    else:
```

```
        currentTarget = patrolPointA
```

5. 장애물 체크

```
hasWall = OverlapCircle(wallCheck.position, checkRadius, groundLayer)
```

```
hasGround = OverlapCircle(groundCheck.position, checkRadius, groundLayer)
```

```
if hasWall || !hasGround:
```

```
    if currentTarget == patrolPointA:
```

```
        currentTarget = patrolPointB
```

```
    else:
```

```
        currentTarget = patrolPointA
```

## 2) 추적 알고리즘

입력:

- playerPosition: 플레이어 위치
- enemyPosition: 적 위치
- chaseSpeed: 추적 속도

매 프레임 실행:

1. 방향 계산

```
direction = Normalize(playerPosition - enemyPosition)
```

2. 이동

```
linearVelocity.x = direction.x * chaseSpeed
```

3. 스프라이트 방향 조정

```
if direction.x > 0:
```

```
    transform.localScale = (1, 1, 1)
```

```
else:
```

```
    transform.localScale = (-1, 1, 1)
```

## 4. 테스트

### 4.1 테스트 방법론

#### 1) Unity Play 모드 테스트

Unity Editor의 Play 모드를 사용하여 실시간으로 게임을 테스트했다. Inspector 창에서 변수를 실시간으로 조정하며 즉각적인 피드백을 받았다. Console 창에서 Debug.Log()를 통해 디버깅 정보를 확인했다. Scene 뷰와 Game 뷰를 동시에 사용하여 오브젝트 위치를 시각적으로 확인했다.

#### 2) 씬별 개별 테스트

씬을 독립적으로 테스트하여 각 씬의 기능을 검증했다. 각 Stage를 Project 창에서 직접 열어 스테이지 별 메커니즘을 테스트했다. Boss1, Boss2, Boss3 씬을 열어 보스 전투 시스템을 테스트했다. Shop 씬을 열어 상점 시스템을 테스트했다. AncientBlacksmith, LifeHeartMap 씬을 열어 각각의 미니게임을 테스트했다.

#### 3) 통합 테스트

전체 게임 플로우를 처음부터 끝까지 플레이하여 통합 테스트를 수행했다. Main 메뉴에서 시작하여 무기 선택, 세이브 슬롯 선택, 게임플레이, 세이브/로드, 씬 전환, 보스 전투, 미니게임까지 모든 시스템을 연결하여 테스트했다.

#### 4) Canvas 설정 검증

UI 요소들이 각 Canvas 이름에 기댐으로 올바른 이름이 설정되었는지 확인했다. "DamageTextCanvas", "ConfirmationPanel" 등의 이름을 검증했다.

### 4.2 발견된 문제 및 해결

#### 문제 해결 이력 요약

문제	발생 원인	해결 방법	소요 시간	심각도
싱글톤 중복 생성	DontDestroyOnLoad 중복	instanceId Dictionary 패턴	2시간	높음
주사위 애니메이션 미작동	Inspector 미활당	null 체크 + 풀백 로직	1시간	중간
미니게임 시간 재개	자동 timeScale 설정	정적 플래그 추가	1.5시간	중간
세이브 데이터 불일치	PlayerSpawnPoint 부재	스폰 시스템 구현	2시간	높음
포털 재사용	포털 삭제 실패	usedPortalIDs 리스트	1시간	중간
레벨업 UI 중복 표시	미니게임 중 레벨업	지연 표시 시스템	1.5시간	낮음
애니메이션 이벤트 미전달	Sprite Child 구조	이벤트 브릿지 스크립트	2~3시간	높음

### 1) 주사위 애니메이션 미작동

**문제:** 보스 전투에서 주사위 애니메이션이 표시되지 않아 전투가 즉시 해결되는 문제가 발생했다.

**해결:** BattleController Inspector에 DiceAnimationManager가 ClashManager에 있는 데미지 계산 방식을 불러오게 만들어 두 코드의 충돌을 해결하였다.

### 2) 세이브 데이터 불일치

**문제:** 씬 전환 후 플레이어 위치가 잘못된 곳에 저장되거나 복원되는 문제가 발생했다.

**해결:** PlayerSpawnPoint 시스템을 구현하여 씬마다 명확한 스폰 위치를 정의했다. 자동 저장 시 PlayerSpawnPoint 위치를 저장하도록 수정했다. OnSceneLoaded()에서 PlayerSpawnPoint를 우선 찾고, 없으면 SaveData의 playerPosition을 사용하도록 했다.

### 3) 애니메이션 이벤트 전달 실패

**문제:** 스프라이트가 자식 오브젝트에 있는 중간보스의 경우 애니메이션 이벤트가 부모 AI 스크립트에 전달되지 않아 공격이 작동하지 않았다.

**해결:** 중간보스 전용 코드를 구현하여 스프라이트 자식 오브젝트에 부착한다. GetComponentInParent로 부모의 AI 스크립트를 찾아 애니메이션 이벤트를 전달한다. Stage1MidBossAI와 Stage2MidBossAI 모두 지원한다. useSpriteChild 플래그와 spriteTransform 참조를 AI 스크립트에 추가하여 Sprite Child 패턴을 선택적으로 사용할 수 있게 했다.

## 4.3 성능 최적화

### 1) 물리 최적화

Rigidbody2D의 Sleeping Mode를 Start Awake로 설정하여 정적 오브젝트는 물리 시뮬레이션에서 제외했다. Continuous Collision Detection(연속 충돌 감지)를 빠르게 움직이는 오브젝트(플레이어, 발사체)에만 적용하고 나머지는 Discrete로 설정했다.

### 2) UI 최적화

Canvas를 여러 개로 분리하여 일부 UI만 업데이트할 때 전체 Canvas를 리빌드하지 않도록 했다. 정적 UI 요소는 별도의 Canvas로 분리했다. 동적 UI 요소(HUD, 인벤토리)는 변경이 발생할 때만 업데이트 하도록 했다. Image의 Raycast Target을 불필요하면 비활성화하여 처리 비용을 줄였다.

### 3) 씬 관리 최적화

씬 로드 시 SceneManager.LoadSceneAsync()를 고려했으나 로딩 화면이 불필요한 짧은 로딩 시간으로 동기 로드를 유지했다. 사용하지 않는 에셋은 Resources.UnloadUnusedAssets()로 언로드했다. 큰 씬은 여러 작은 씬으로 분할하여 로딩 시간을 분산했다.

## 5. 결론

### 5.1 프로젝트 성과

#### 1) 구현 완료 기능 요약

본 프로젝트는 계획한 모든 핵심 기능을 성공적으로 구현했다. 플레이어 시스템은 이동, 점프, 대시, 벽 슬라이드 등의 정교한 컨트롤과 3종 무기 시스템, 레벨업 시스템을 포함한다. 전투 시스템은 일반 적 AI와 중간보스 AI, 주사위 카드 전투 보스전을 구현하여 다양한 전투 경험을 제공한다. 세이브/로드 시스템은 3개 슬롯과 JSON 기반 저장, 자동 저장 기능을 제공한다. 인벤토리 및 상점 시스템은 드래그 앤 드롭, 랜덤 아이템, 새로고침 기능을 포함한다. 미니게임 시스템은 대장간과 라이프 맵 미니게임을 보상과 함께 제공한다. UI 시스템은 메인 메뉴, HUD, 레벨 업 UI, 일시 정지 메뉴 등 완전한 사용자 인터페이스를 구현했다. 씬 관리 시스템은 포털, 플레이어 스폰, 미니게임 복귀 기능을 포함한다. 스테이지별 특수 시스템으로 Stage3 수중 시스템을 구현했다.

#### 2) 개발 과정에서 습득한 기술

Unity 2D 게임 개발 전반에 대한 깊이 있는 이해를 습득했다.

Rigidbody2D와 Collider2D를 활용한 물리 시스템 구현 능력을 키웠다.

Animator Controller와 애니메이션 이벤트 활용 능력을 개발했다.

싱글톤, 옵저버, 상태 머신 등의 게임 디자인 패턴을 실전에 적용했다.

JSON 직렬화, ScriptableObject, DontDestroyOnLoad 등의 데이터 지속성 기술을 습득했다.

Canvas와 EventSystem을 활용한 UI/UX 구현 능력을 개발했다.

코루틴을 활용한 비동기 프로그래밍 경험을 쌓았다.

Git과 GitHub를 활용한 버전 관리 경험을 습득했다.

디버깅 및 문제 해결 능력을 향상시켰다.

#### 3) 프로젝트 목표 달성을

모든 핵심 시스템 구현 완료 목표를 100% 달성했다. 메인 메뉴부터 게임 종료까지 전체 플로우를 완성했다. 3개 이상의 스테이지(Stage1, Stage 2, Stage3)와 보스전을 구현했다. 세이브/로드 시스템의 안정성을 확보했고 모든 게임 데이터를 정확히 저장/복원한다. 주요 버그를 모두 수정하여 안정적인 플레이 경험을 제공한다.

## 5.2 한계점 및 개선 사항

### 1) 현재 시스템의 한계

적 AI가 단순한 패턴만 사용하여 예측 가능한 행동을 보인다. 고급 경로 탐색(A\* 알고리즘 등)이 없어 장애물 회피가 제한적이다. 보스 전투가 2~3 종류로 제한되어 반복 플레이 시 지루할 수 있다. 난이도 조절 시스템이 없어 초보자와 숙련자 모두에게 적합하지 않을 수 있다. 스토리와 컷신이 부족하여 몰입도가 낮다. 멀티플레이 기능이 없어 싱글 플레이만 가능하다. 모바일 플랫폼을 지원하지 않아 PC에서만 플레이 가능하다.

### 2) 발견된 버그 및 미해결 과제

특정 상황에서 플레이어가 벽에 끼이는 현상이 가끔 발생한다. 프레임 드롭이 심한 환경에서 물리 계산이 부정확해질 수 있다. 매우 빠른 속도로 씬을 전환하면 DontDestroyOnLoad 오브젝트가 완전히 정리되지 않을 수 있다. 일부 UI 요소가 해상도 변경 시 잘못 배치될 수 있다.

### 3) 시간 부족으로 미구현된 기능

추가 무기 시스템(활, 마법 지팡이 등)을 구현하지 못했다. 스킬 트리와 다양한 스킬을 구현하지 못했다. 업적 시스템과 도전 과제를 구현하지 못했다. 사운드와 음악이 제한적이다. 상세한 튜토리얼과 도움말 시스템이 부족하다. 엔딩 시네마틱과 크레딧 화면을 구현하지 못했다.

## 5.3 추후 과제

### 1) 추가 스테이지 및 보스

2개 이상의 추가 스테이지를 개발하여 게임 플레이 시간을 연장할 수 있다. 각 스테이지에 고유한 테마와 메커니즘을 부여할 수 있다(예: 용암 스테이지, 얼음 스테이지). 2~3개의 추가 보스를 개발하여 각 보스마다 독특한 패턴과 전략을 제공할 수 있다. 히든 보스와 챌린지 모드를 추가할 수 있다.

### 2) 멀티플레이 기능

로컬 협동 플레이(2인)를 구현하여 친구와 함께 플레이할 수 있다. 온라인 멀티플레이를 구현하여 전 세계 플레이어와 협력하거나 대전할 수 있다. PvP 모드를 추가하여 플레이어 간 대전이 가능하게 할 수 있다.

### 3) 모바일 포팅

터치 컨트롤을 구현하여 모바일 기기에서 플레이할 수 있게 할 수 있다. UI를 모바일 해상도에 맞게 조정할 수 있다. 모바일 성능 최적화를 수행할 수 있다(배터리 소모 감소, 프레임 안정화). iOS와 Android 빌드를 제작할 수 있다.

#### **4) 난이도 조절 시스템**

Normal, Hard 난이도를 구현할 수 있다. 난이도에 따라 적의 체력, 데미지, 개체 수를 조정할 수 있다. 하드 모드는 추가 보상을 제공할 수 있다. 뉴 게임 플러스 모드를 추가하여 스탯을 유지하며 더 어려운 게임을 플레이할 수 있다.

#### **5) 추가 무기 및 스킬 시스템**

활, 마법 지팡이, 도끼 등 추가 무기를 구현할 수 있다. 각 무기마다 고유한 스킬 트리를 제공할 수 있다. 특수 스킬(궁극기)을 추가할 수 있다.

#### **6) 스토리 컷 씬 추가**

인트로 컷 씬으로 게임 배경을 설명할 수 있다. 스테이지 간 스토리 컷 씬으로 몰입도를 높일 수 있다. 보스 등장 컷 씬으로 긴장감을 조성할 수 있다. 엔딩 컷 씬과 크레딧을 추가할 수 있다. 대화 시스템을 확장하여 NPC와의 상호작용을 강화할 수 있다.

#### **7) 사운드 및 음악 개선**

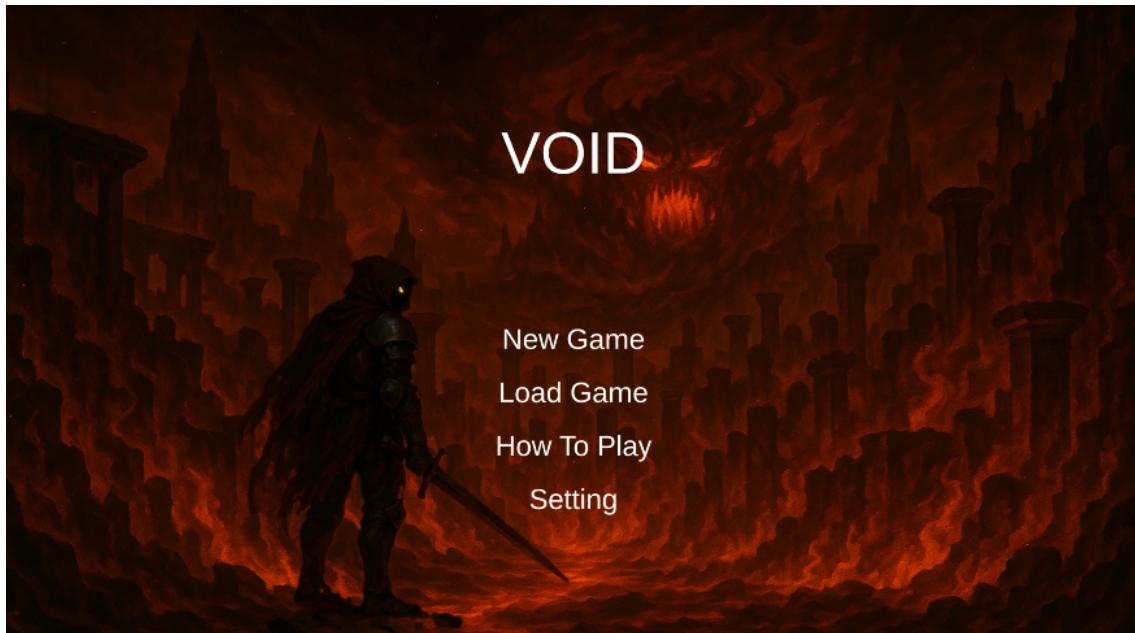
각 스테이지별 BGM을 작곡하거나 구매할 수 있다. 보스전 전용 음악을 추가할 수 있다. 다양한 효과음(공격, 피격, 걷기, 점프 등)을 고품질로 제작할 수 있다. 음량 조절과 음소거 옵션을 개선할 수 있다.

#### **8) 추가 기능**

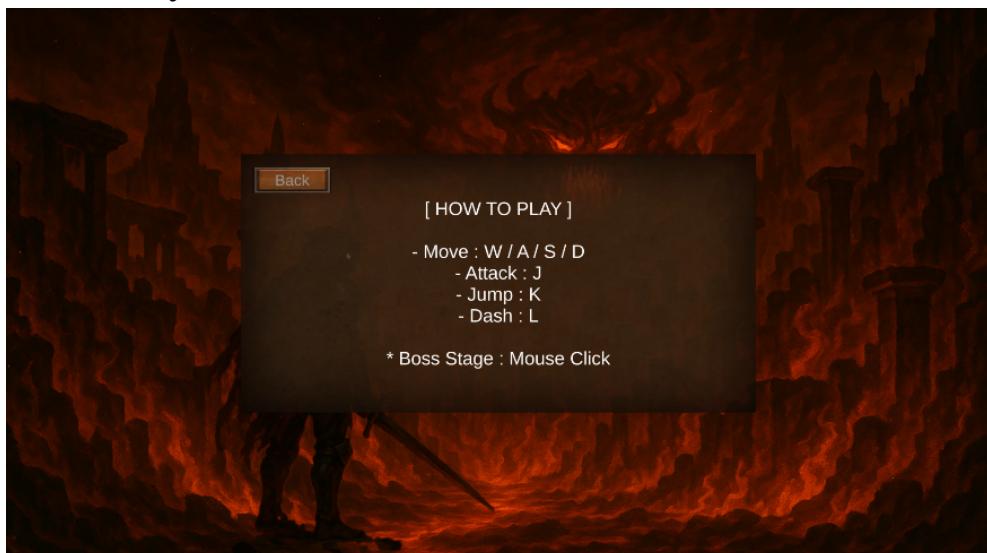
업적 시스템과 Steam 업적 연동을 구현할 수 있다. 리더보드를 추가하여 플레이어 간 경쟁을 유도할 수 있다. 데일리 챌린지와 주간 이벤트를 구현할 수 있다. 치장 시스템(캐릭터 외형, 무기 외형)을 추가할 수 있다. 상세한 통계 화면(플레이 시간, 처치 수, 사망 횟수 등)을 제공할 수 있다.

## 부록 I. 졸업작품 사진

### Main Scene



### HowToPlay Scene



## Stage1 Scene



## Stage2 Scene



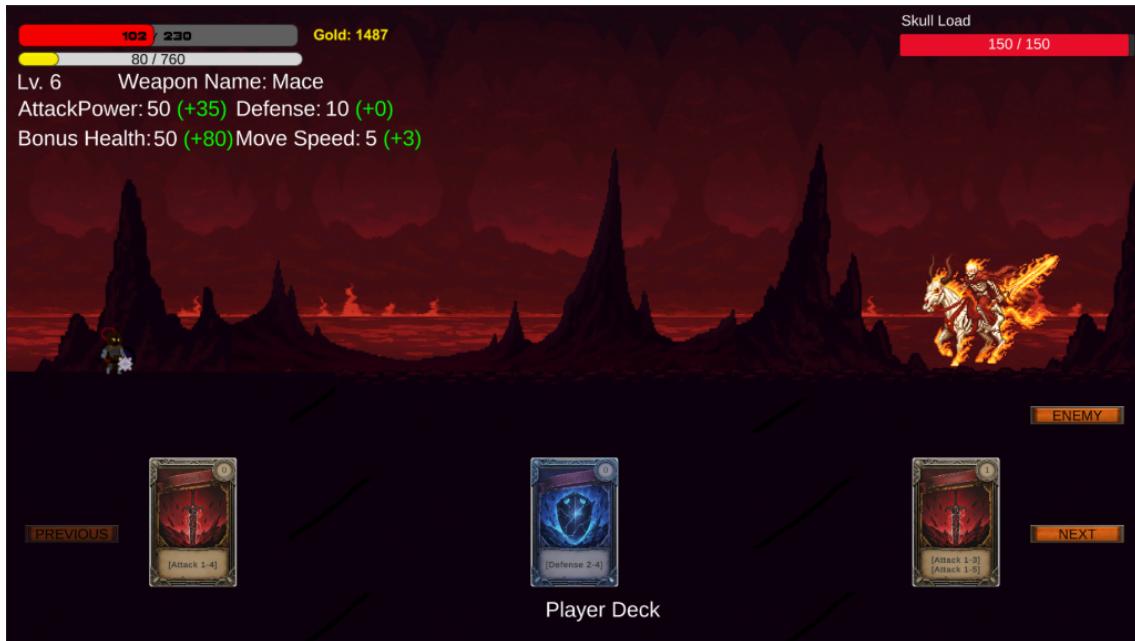
### Stage3 Scene



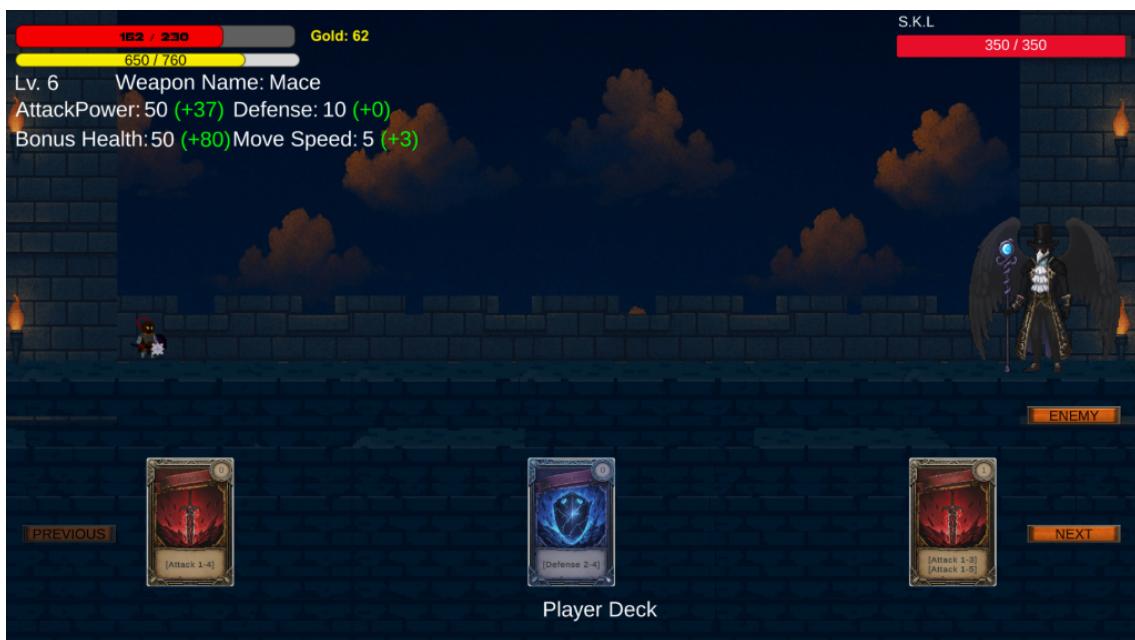
### Shop Scene (1,2,3)



## Boss1 Scene



## Boss2 Scene



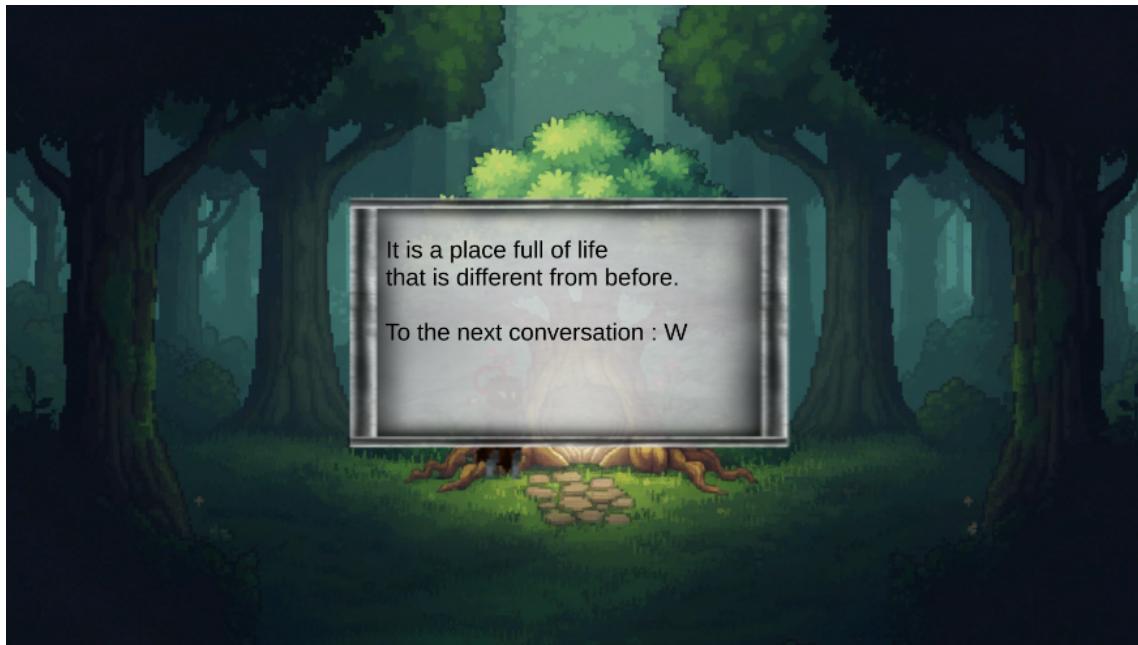
## Boss3 Scene



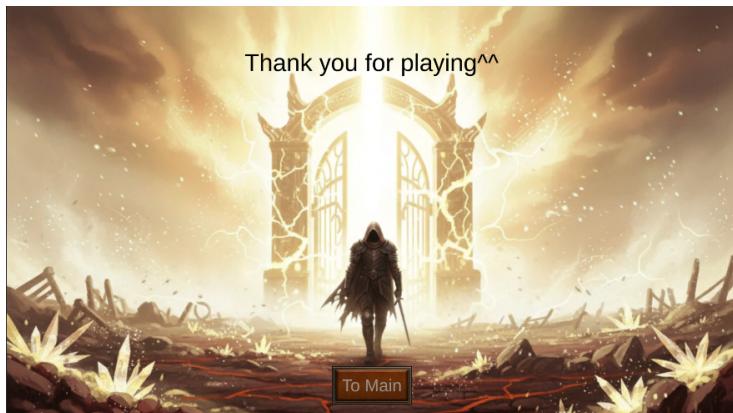
## AncientBlacksmith Scene



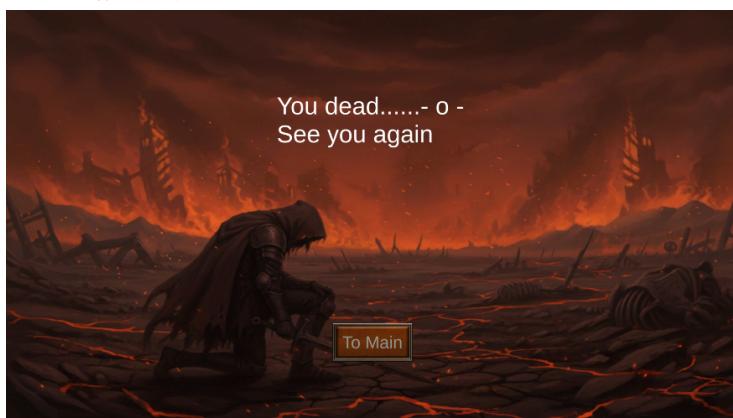
## LifeHeartMap Scene



## ClearScene



## DeathScene



## 부록 II. 졸업작품 포스터

# Unity Hybrid 2D RPG

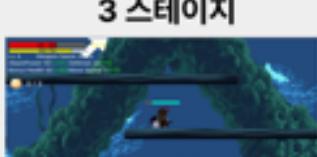
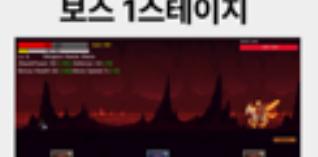
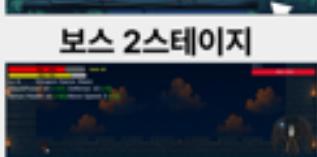
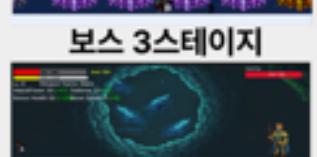
컴퓨터공학전공  
20201531 송진혁  
20201536 김기찬  
20201098 임승태

### 게임 개요

Unity 6 엔진을 사용하여 개발한 2D 흥스크롤 액션 RPG 게임입니다. 플레이어는 다양한 무기를 선택하여 적들과 전투하고, 보스를 물리치며 성장하는 여정을 경험합니다.

주요 기능	개발 환경
- 플레이어 조작 및 성장시스템 - 스테이지 및 보스전 시스템 - 게임 관리 및 저장 시스템	 

### 구현 결과

메인화면	1 스테이지	2 스테이지
		
3 스테이지	상점 스테이지	보스 1스테이지
		
보스 2스테이지	보스 3스테이지	
		

## 부록 III. 프로젝트 파일 구조

### 1) Assets 폴더 구조

#### 1.1) Scripts/ - C# 스크립트 (17개 카테고리)

Assets/Scripts/

```
|   └── Ancient/          # 대장간 미니게임 시스템
|       ├── BlacksmithMinigameManager.cs    # 미니게임 관리
|       ├── DialogueController.cs           # 대화 시스템
|       ├── MidBossController.cs          # 중간보스 이벤트
|       ├── StatueInteraction.cs         # 석상 상호작용
|       ├── UnifiedFlameTrap.cs        # 불꽃 함정
|       ├── SoulFlame.cs               # 혼의 불꽃
|       ├── FlameTrapSpawner.cs       # 함정 스포너
|       └── ...
|
|   └── BossBattle/          # 주사위 기반 카드 전투 시스템
|       ├── BossGameManager.cs      # 게임 상태 관리 (Exploration/Battle)
|       ├── BattleController.cs    # 전투 흐름 조율
|       ├── CharacterStats.cs     # 캐릭터 스탯 및 텍
|       ├── CombatPage.cs         # 카드 데이터 (ScriptableObject)
|       ├── CombatDice.cs        # 주사위 로직
|       ├── ClashManager.cs       # 클래시 해결
|       ├── DiceAnimationManager.cs # 주사위 애니메이션 총괄
|       ├── DiceVisual.cs        # 개별 주사위 UI
|       ├── CardUI.cs            # 카드 UI
|       ├── CharacterVisuals.cs   # 캐릭터 비주얼
|       ├── BattleTrigger.cs      # 전투 시작 트리거
|       └── CameraController.cs    # 전투 카메라
|
|   └── Player/              # 플레이어 시스템
|       ├── PlayerController.cs    # 메인 플레이어 제어 (Singleton)
|       ├── PlayerStats.cs        # 레벨, XP, 스탯 관리
|       ├── PlayerHealth.cs       # 체력 관리
|       ├── PlayerSpawnPoint.cs   # 스폰 위치
|       └── SwordAttack.cs / SwordItem.cs / Sword.cs
```

```
|   |   |   |   |   LanceAttack.cs / LanceItem.cs / Lance.cs
|   |   |   |   |   MaceAttack.cs / MaceItem.cs / Mace.cs
|   |   |   |   |   TurnManager.cs           # 턴제 전투 (레거시)
|   |   |   |   |   ActionData.cs          # 액션 데이터
|   |   |   |   |   BossAreaTrigger.cs    # 보스 영역 트리거
|
|   |   |   |   |
|   |   |   |   |   Monster/          # 적 AI 시스템
|   |   |   |   |   |   MonsterController.cs      # 일반 몬스터 AI
|   |   |   |   |   |   MonsterHealth.cs        # 몬스터 체력
|   |   |   |   |   |   MonsterHealth1.cs       # 몬스터 체력 (변형)
|   |   |   |   |   |   Stage1MidBossAI.cs     # Stage1 중간보스 (마법)
|   |   |   |   |   |   Stage2MidBossAI.cs     # Stage2 중간보스 (텔레포트)
|   |   |   |   |   |   MidBossAnimationEvents.cs # 애니메이션 이벤트 브릿지
|   |   |   |   |   |   BossAI.cs            # 보스 AI
|   |   |   |   |   |   BossAttack.cs         # 보스 공격
|   |   |   |   |   |   BossHealth.cs        # 보스 체력
|   |   |   |   |   |   ItemDrop.cs          # 아이템 드롭
|   |   |   |   |   |   SpellAttack.cs       # 마법 투사체
|   |   |   |   |   |   MobSpawner.cs        # 몬스터 스포너
|
|   |   |   |   |
|   |   |   |   |   SaveSystem/        # 세이브/로드 시스템
|   |   |   |   |   |   GameManager.cs      # 게임 상태 조율 (Singleton)
|   |   |   |   |   |   SaveManager.cs       # 파일 I/O (Singleton)
|   |   |   |   |   |   SaveData.cs        # 저장 데이터 구조
|
|   |   |   |   |
|   |   |   |   |   UI/              # UI 시스템
|   |   |   |   |   |   MainMenuController.cs # 메인 메뉴
|   |   |   |   |   |   WeaponChoice.cs      # 무기 선택
|   |   |   |   |   |   LoadGameUI.cs        # 세이브 슬롯 선택
|   |   |   |   |   |   SaveSlotButton.cs    # 슬롯 버튼
|   |   |   |   |   |   StatsUIManager.cs # 스탯 UI
|   |   |   |   |   |   LevelUpUIManager.cs # 레벨업 UI
|   |   |   |   |   |   PauseMenuUI.cs       # 일시정지 메뉴
|   |   |   |   |   |   Inventory.cs / InventoryUI.cs # 인벤토리
```

```
|   ├── InventorySlot.cs           # 인벤토리 슬롯
|   ├── DontDestroyOnLoadManager.cs # 씬 간 지속성 관리
|   ├── DamageText.cs             # 대미지 텍스트
|   ├── CameraFollow.cs           # 카메라 추적
|   ├── CameraBounds.cs          # 카메라 경계
|   ├── ItemPickup.cs            # 아이템 획득
|   ├── Coin.cs                  # 코인
|   ├── PotionItemData.cs        # 포션 데이터 (ScriptableObject)
|   ├── CurrencyUI.cs           # 화폐 UI
|   ├── Tooltip.cs / TooltipManager.cs # 툴팁
|   ├── SettingsManager.cs / SettingUI.cs
|   ├── AudioManager.cs          # 오디오 관리
|   └── ...
|
|   └── Shop/                   # 상점 시스템
|       ├── ShopManager.cs        # 상점 관리 (Singleton)
|       ├── ShopItemData.cs      # 상점 아이템 (ScriptableObject)
|       ├── ShopPedestal.cs       # 아이템 표시대
|       ├── RefreshPedestal.cs    # 새로고침 표시대
|       ├── ExchangePedestal.cs   # 교환 표시대
|       └── ShopNPCInteraction.cs # NPC 상호작용
|
|   └── Life/                    # 라이프맵 미니게임 시스템
|       ├── LifeGameManager.cs    # 라이프맵 관리
|       ├── MinigameManager.cs    # 미니게임 관리
|       ├── PortalController.cs   # 포털 시스템
|       ├── PortalReturnData.cs   # 복귀 데이터
|       ├── PortalReturnManager.cs # 복귀 관리
|       └── DialogueTriggerZone.cs # 대화 트리거
|
|   └── Stage3/                 # Stage3 수중 시스템
|       ├── Stage3Manager.cs     # Stage3 씬 관리
|       ├── PlayerSwimming.cs    # 수영 메커니즘
|       └── PlayerOxygen.cs      # 산소 관리
```

```
|   ├── OxygenZone.cs           # 산소 회복 구역
|   ├── Pearl.cs / PearlDisplayUI.cs    # 진주 수집
|   ├── GiantClam.cs            # 거대 조개
|   ├── ClamMonsterController.cs      # 조개 몬스터 AI
|   ├── ParallaxController.cs       # 패럴랙스 배경
|   ├── PortalToStage3.cs         # Stage3 포털
|   └── SeaMosterDeathReporter.cs    # 바다 몬스터 사망 보고
|
|   └── CardSystem/             # 카드 시스템 (미사용/실험적)
|       ├── CardData.cs          # 카드 데이터
|       ├── CardInventoryUI.cs    # 카드 인벤토리 UI
|       ├── CardSlot.cs           # 카드 슬롯
|       ├── DraggableCard.cs      # 드래그 가능 카드
|       └── ...
|
|   └── SceneManagement/        # 씬 관리
|       ├── ClearSceneManager.cs  # 클리어 씬
|       ├── DeathSceneManager.cs  # 사망 씬
|       ├── GoStage.cs            # 스테이지 이동
|       └── Clear.cs
|
|   └── Traps/                  # 함정 시스템
|       ├── Trap.cs              # 기본 함정
|       ├── FlameTrap.cs          # 불꽃 함정
|       └── DamageOnTouch.cs      # 접촉 데미지
|
|   └── Weapons/                # 무기 시스템
|       ├── WeaponStats.cs        # 무기 스탯 (ScriptableObject)
|       └── WeaponSpawner.cs      # 무기 스포너
|
|   └── Camera/                 # 카메라 시스템
|       └── CameraFallBack.cs     # 카메라 폴백
|
|   └── Environment/            # 환경 시스템
```

```

|   ├── ParallaxBackground2.cs      # 패럴랙스 배경
|   └── BackGround.cs            # 배경
|
|   └── Core/                    # 핵심 데이터
|       └── GameData.cs          # 정적 게임 데이터
|
└── Testing/                  # 테스트 스크립트
    └── ScarecrowHealth.cs      # 허수아비 체력 (테스트용)

```

## 1.2) Sprites/ - 자체 제작 스프라이트

Assets/Sprites/

```

├── Animations/              # 애니메이션 스프라이트
|   ├── Character/           # 캐릭터 애니메이션
|   ├── Effect/              # 이펙트 애니메이션
|   ├── Item/                # 아이템 애니메이션
|   ├── Lance/               # 창 애니메이션
|   ├── Mace/                # 망치 애니메이션
|   └── Sword/               # 검 애니메이션
|
├── Monster/                 # 몬스터 스프라이트
|   ├── Boss/                # 보스 스프라이트
|   ├── Clam/                # 조개 몬스터
|   ├── Goblin/              # 고블린
|   ├── SeaMonster/          # 바다 몬스터
|   ├── Skeleton/            # 스켈레톤
|   └── Skeleton.shield/     # 방패 스켈레톤
|
├── MiniGame/                # 미니게임 스프라이트
|   ├── BlackSmith/          # 대장간 미니게임
|   |   └── FireTrap/         # 불꽃 함정
|   └── Life/                # 라이프맵 미니게임
|
├── Card/                    # 카드 이미지
└── Item/                    # 아이템 스프라이트

```

```
|  
└── Prefabs/          # 스프라이트 프리팹  
    ├── Items/        # 아이템 프리팹  
    └── Monsters/     # 몬스터 프리팹
```

### 1.3) 기타 자체 제작 폴더

Assets/

```
|── CombatPage/      # 보스 전투 카드 데이터 (ScriptableObjects)  
|   └── *.asset       # CombatPage 에셋들  
  
|── Resources/       # 런타임 로드 리소스  
|   └── ...  
  
└── Tileset/         # 타일셋 (레벨 디자인용)  
    └── ...
```

### 1.4) 임포트된 외부 에셋 (참고용)

Assets/

```
|── 2DRPK/           # 2D RPG 에셋 팩  
|── Bringer Of Death/ # 보스 캐릭터 에셋  
|── Evil Wizard 2/    # 마법사 적 에셋  
|── Monsters Creatures Fantasy/ # 몬스터 에셋  
|── Dark_Brown_GUI_kit/ # UI 에셋  
|── Fantasy/          # 판타지 에셋  
|── PixelFX_vol1/     # 이펙트 에셋  
|── TextMesh Pro/      # 텍스트 렌더링 (Unity 공식)  
└── ...
```

## 2) 스크립트 파일 통계

카테고리	파일 수	설명
Scripts 총계	~ 100개	자체 제작 C# 스크립트
플레이어 시스템	15개	이동, 공격, 무기
적 AI 시스템	12개	몬스터, 중간보스, 보스
보스 전투 시스템	12개	카드, 주사위, 애니메이션
UI 시스템	25개	메뉴, HUD, 인벤토리 등
세이브 / 로드	3개	GameManager, SaveManager, SaveData
미니게임	11개	대장간, 라이프맵
상점 시스템	6개	구매, 새로고침, 표시대
Stage3 시스템	10개	수영, 산소, 환경
Sprites	~ 50+ 폴더	자체 제작 스프라이트
ScriptableObjects	~ 20개	CombatPage, ItemData 등

## 3) 주요 스크립트 상세 목록

### 3.1) 플레이어 관련 스크립트

- PlayerController.cs: 메인 플레이어 제어 (PlayerController.cs:1)
- PlayerStats.cs: 레벨, XP, 스텝 관리 (PlayerStats.cs:1)
- PlayerHealth.cs: 체력 관리 (PlayerHealth.cs:1)
- SwordAttack.cs, LanceAttack.cs, MaceAttack.cs: 무기별 공격 (SwordAttack.cs:1, LanceAttack.cs:1, MaceAttack.cs:1)
- SwordItem.cs, LanceItem.cs, MaceItem.cs: 무기 아이템 (SwordItem.cs:1, LanceItem.cs:1, MaceItem.cs:1)
- PlayerSwimming.cs: 수영 메커니즘 (PlayerSwimming.cs:1)
- PlayerOxygen.cs: 산소 관리 (PlayerOxygen.cs:1)

### 3.2) 적 AI 관련 스크립트

- MonsterController.cs: 일반 몬스터 AI (MonsterController.cs:1)
- Stage1MidBossAI.cs: 스테이지 1 중간보스 (Stage1MidBossAI.cs:1)
- Stage2MidBossAI.cs: 스테이지 2 중간보스 (Stage2MidBossAI.cs:1)
- MidBossAnimationEvents.cs: 애니메이션 이벤트 브릿지 (MidBossAnimationEvents.cs:1)
- MonsterHealth.cs: 적 체력 관리 (MonsterHealth.cs:1)
- BossAI.cs, BossAttack.cs, BossHealth.cs: 보스 AI (BossAI.cs:1, BossAttack.cs:1, BossHealth.cs:1)
- ItemDrop.cs: 아이템 드롭 (ItemDrop.cs:1)
- ClamMonsterController.cs: 수중 적 AI (ClamMonsterController.cs:1)
- MobSpawner.cs: 몬스터 스포너 (MobSpawner.cs:1)

### 3.3) 보스 전투 관련 스크립트

- BossGameManager.cs: 게임 상태 관리 (BossGameManager.cs:1)
- BattleController.cs: 전투 조율 (BattleController.cs:1)
- CharacterStats.cs: 캐릭터 스탯 및 텍 (CharacterStats.cs:1)
- CombatPage.cs: 카드 데이터 (CombatPage.cs:1)
- CombatDice.cs: 주사위 로직 (CombatDice.cs:1)
- ClashManager.cs: 클래시 해결 (ClashManager.cs:1)
- DiceVisual.cs: 주사위 UI (DiceVisual.cs:1)
- DiceAnimationManager.cs: 주사위 애니메이션 (DiceAnimationManager.cs:1)
- CardUI.cs: 카드 UI (CardUI.cs:1)
- CharacterVisuals.cs: 캐릭터 비주얼 (CharacterVisuals.cs:1)
- BattleTrigger.cs: 전투 트리거 (BattleTrigger.cs:1)

### 3.4) 씬 관리 관련 스크립트

- PortalController.cs: 포털 시스템 (PortalController.cs:1)
- StatueInteraction.cs: 석상 상호작용 (StatueInteraction.cs:1)
- PortalReturnManager.cs: 복귀 위치 관리 (PortalReturnManager.cs:1)
- PortalReturnData.cs: 복귀 데이터 (PortalReturnData.cs:1)
- PlayerSpawnPoint.cs: 스폰 위치 (PlayerSpawnPoint.cs:1)
- Stage3Manager.cs: Stage3 관리 (Stage3Manager.cs:1)
- PortalToStage3.cs: Stage3 포털 (PortalToStage3.cs:1)

### 3.5) 세이브/로드 관련 스크립트

- SaveManager.cs: 파일 I/O (SaveManager.cs:1)
- GameManager.cs: 게임 상태 조율 (GameManager.cs:1)
- SaveData.cs: 저장 데이터 (SaveData.cs:1)
- LoadGameUI.cs: 슬롯 선택 UI (LoadGameUI.cs:1)
- SaveSlotButton.cs: 슬롯 버튼 (SaveSlotButton.cs:1)

### 3.6) UI 관련 스크립트

- MainMenuController.cs: 메인 메뉴 (MainMenuController.cs:1)
- WeaponChoice.cs: 무기 선택 (WeaponChoice.cs:1)
- StatsUIManager.cs: 스탯 UI (StatsUIManager.cs:1)
- LevelUpUIManager.cs: 레벨업 UI (LevelUpUIManager.cs:1)
- InventoryUI.cs, Inventory.cs: 인벤토리 (InventoryUI.cs:1, Inventory.cs:1)
- InventorySlot.cs: 인벤토리 슬롯 (InventorySlot.cs:1)
- PauseMenuUI.cs: 일시정지 메뉴 (PauseMenuUI.cs:1)
- DontDestroyOnLoadManager.cs: 지속성 관리 (DontDestroyOnLoadManager.cs:1)
- DamageText.cs: 대미지 텍스트 (DamageText.cs:1)
- CameraFollow.cs, CameraBounds.cs: 카메라 (CameraFollow.cs:1, CameraBounds.cs:1)
- CurrencyUI.cs: 화폐 UI (CurrencyUI.cs:1)
- Tooltip.cs, TooltipManager.cs: 툴팁 (Tooltip.cs:1, TooltipManager.cs:1)

### 3.7) 상점 관련 스크립트

- ShopManager.cs: 상점 관리 (ShopManager.cs:1)
- ShopItemData.cs: 아이템 데이터 (ShopItemData.cs:1)
- ShopPedestal.cs: 아이템 표시대 (ShopPedestal.cs:1)
- RefreshPedestal.cs: 새로고침 표시대 (RefreshPedestal.cs:1)
- ExchangePedestal.cs: 교환 표시대 (ExchangePedestal.cs:1)

### 3.8) 미니게임 관련 스크립트

- BlacksmithMinigameManager.cs: 대장간 미니게임 (BlacksmithMinigameManager.cs:1)
- DialogueController.cs: 대화 시스템 (DialogueController.cs:1)
- MidBossController.cs: 중간보스 이벤트 (MidBossController.cs:1)
- UnifiedFlameTrap.cs, SoulFlame.cs: 미니게임 오브젝트 (UnifiedFlameTrap.cs:1, SoulFlame.cs:1)
- LifeGameManager.cs: 라이프맵 관리 (LifeGameManager.cs:1)

### 3.9) 기타 스크립트

- ItemPickup.cs: 아이템 획득 (ItemPickup.cs:1)
- Coin.cs: 코인 획득 (Coin.cs:1)
- PotionItemData.cs: 포션 데이터 (PotionItemData.cs:1)
- WeaponStats.cs: 무기 스탯 (WeaponStats.cs:1)
- GameData.cs: 정적 데이터 (GameData.cs:1)
- Trap.cs: 함정 (Trap.cs:1)
- OxygenZone.cs: 산소 회복 구역 (OxygenZone.cs:1)
- Pearl.cs, PearlDisplayUI.cs: 진주 수집 (Pearl.cs:1, PearlDisplayUI.cs:1)
- GiantClam.cs: 거대 조개 (GiantClam.cs:1)
- ParallaxController.cs: 패럴랙스 배경 (ParallaxController.cs:1)

## 부록 IV. 게임 플레이 가이드

### 1) 조작 방법

키보드 조작은 다음과 같다. 화살표 키 또는 WASD로 이동한다.  
K 키로 점프한다. L 키로 대시한다. J 키로 공격한다(일반 전투).  
Space 바로 턴을 실행한다(보스 전투).  
ESC 키로 일시정지 메뉴를 연다.  
I 키로 인벤토리를 연다(구현된 경우).  
W 키로 상호작용한다(포털, NPC).

### 2) 게임 플로우

게임 시작 시 Main 메뉴에서 New Game 또는 Load Game을 선택한다. New Game을 선택하면 Weapon 씬에서 무기(검/창/메이스)를 선택한다. LoadGame 씬에서 세이브 슬롯(1/2/3)을 선택한다. Stage1에서 게임이 시작된다.

스테이지 탐험 중에는 플랫폼을 점프하여 이동한다. 적을 공격하여 XP와 아이템을 획득한다. 포털을 통해 다른 씬으로 이동한다. 일시 정지(ESC)로 게임을 저장한다(자동 저장도 작동).

보스 전투 중에는 카드를 선택한다(최대 3장). Space 바로 턴을 확인한다. 주사위가 굴러가고 합이 해결되는 것을 관찰한다. 보스의 HP를 0으로 만들어 승리한다.

미니게임 중에는 대장간에서 불꽃을 수집한다. 높은 점수로 더 많은 보상을 얻는다. 라이프맵에서 특수 도전 과제를 완료한다.

상점에서는 아이템을 구매하여 스탯을 업그레이드한다. 새로고침하여 새 아이템을 표시한다(비용 증가). 포션을 구매하여 체력을 회복한다.

### 3) 팁 및 전략

각 무기의 특성을 이해하고 활용한다. 검은 균형 잡힌 성능으로 모든 상황에 적합하다. 창은 긴 리치로 안전한 거리를 유지할 수 있다. 메이스는 높은 데미지와 넉백으로 강력한 적에 유효하다.

레벨업 시 상황에 맞는 스텝을 선택한다. 공격력은 빠른 전투를 위해 선택한다. 방어력은 생존력을 높이기 위해 선택한다. 체력은 최대 HP를 증가시키기 위해 선택한다. 이동 속도는 기동성을 높이기 위해 선택한다.

보스 전투 전략은 다음과 같다. 보스의 템을 확인하여 패턴을 파악한다. 코스트를 효율적으로 관리한다. 카드 쿨타임을 고려하여 선택한다.

저장을 자주한다. 자동 저장이 있지만 중요한 순간에는 수동 저장을 권장한다. 여러 슬롯을 사용하여 다양한 진행 상황을 보관한다.

미니게임에 참여하여 추가 보상을 얻는다. 불꽃 모으기에서 공격력을 얻는다. 라이프 맵에서 특수 보상을 얻는다.

## 부록 V. 리소스 목록

### 1) 사용된 에셋

본 프로젝트는 다음 에셋을 사용했다. Unity Asset Store의 2D 캐릭터 스프라이트, Unity Asset Store의 2D 타일셋(지형, 배경), Unity Asset Store의 2D UI 에셋, Unity Asset Store의 사운드 이펙트 팩, Unity Asset Store의 BGM 팩, 자체 제작 스크립트 및 시스템.

### 2) 라이선스 정보

사용된 모든 에셋은 Unity Asset Store의 표준 라이선스를 따른다. 개인 및 상업적 사용이 가능하다. 에셋 재배포는 금지되어 있다. 자체 제작 스크립트는 MIT 라이선스를 따른다(선택적).

## **부록 VI. 참고 문헌**

### **1) Unity 공식 문서**

- Unity Manual: <https://docs.unity3d.com/Manual/index.html>
- Unity Scripting Reference: <https://docs.unity3d.com/ScriptReference/index.html>
- Unity 2D Documentation: <https://docs.unity3d.com/Manual/Unity2D.html>
- Unity Physics 2D: <https://docs.unity3d.com/Manual/Physics2DReference.html>
- Unity UI: <https://docs.unity3d.com/Packages/com.unity.ugui@latest>

### **2) 참고한 튜토리얼 및 아티클**

- Brackeys - How to make a 2D Game in Unity
- Unity Learn - 2D Platformer Tutorial
- Game Dev Beginner - Unity Input System
- Catlike Coding - Unity C# Tutorials
- Stack Overflow - 다양한 기술적 질문 해결

### **3) 게임 디자인 참고**

- Hollow Knight - Team Cherry
- Dead Cells - Motion Twin
- Blasphemous - The Game Kitchen
- Library of Ruina - Project Moon