

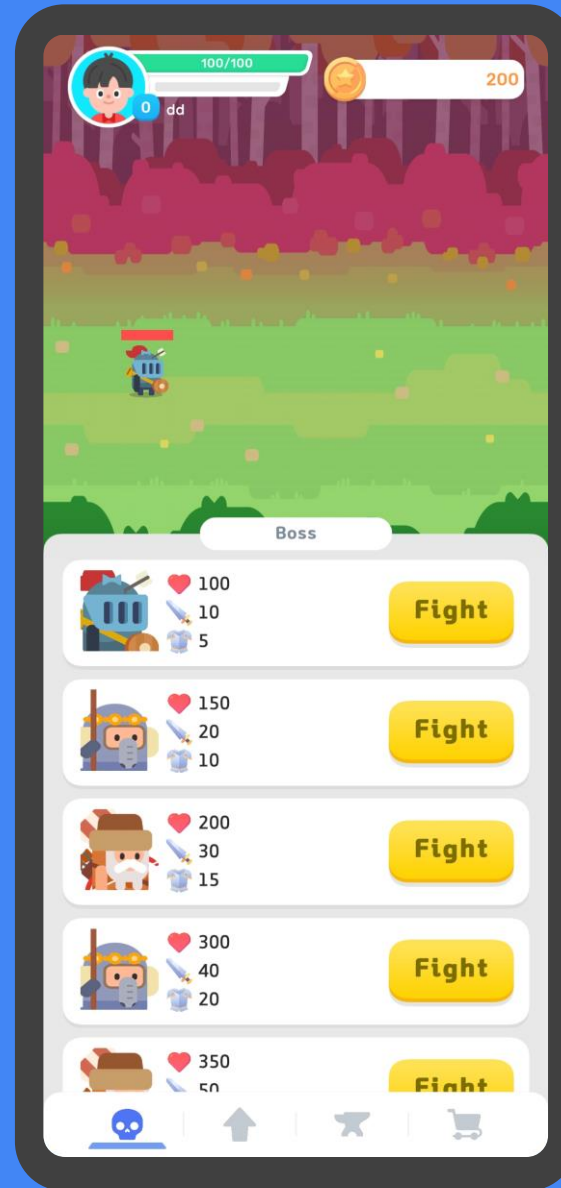
TAB HERO

PORTFOLIO



|게임 소개

무기를 조합하여 강화하고
스탯을 업그레이드 하며
점점 강한 몬스터를 클리어 하는
성장형 클릭어 게임 입니다



|목차

시스템 소개	일정/데이터 관리	스크립트	최적화
전투	진행 일정	플레이어	UI
업그레이드	테이블	전투	
제작	자동화	세이브/로드	
상점	로컬라이징	싱글 톤 사용 이유	

시스템 소개

전투

업그레이드

제작

상점

| 전투

적의 스탯을 보고 골라서 전투를
시작할 수 있고 한번씩 번갈아 가며 공격합니다

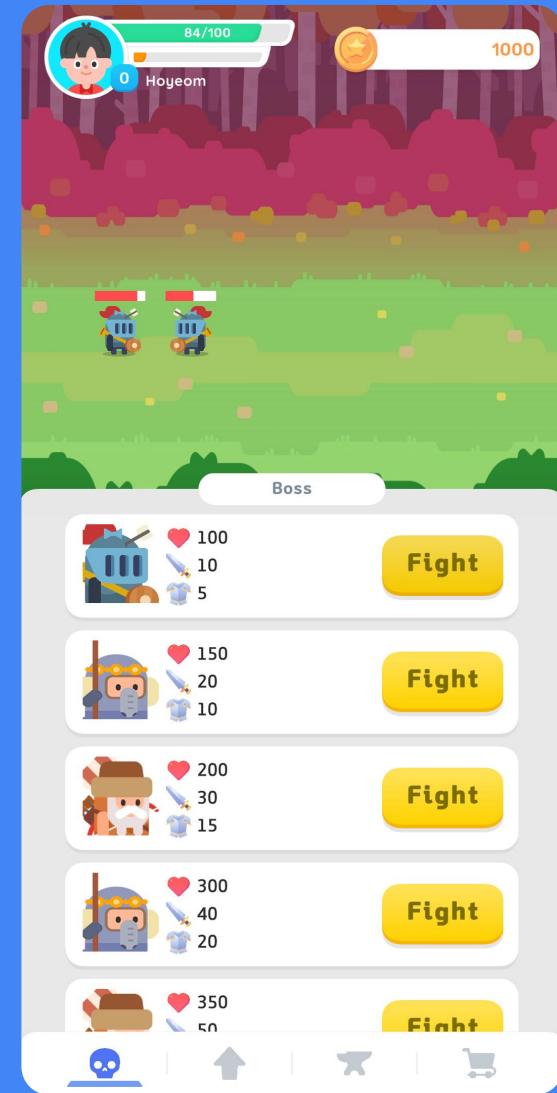


1. 몬스터 정보

보스의 능력치를 불러오며 표기한다

2. 전투 시작 버튼

터치 시 몬스터를 소환한다



I 업그레이드

전투를 하며 쌓인 재화를 이용하여
업그레이드를 합니다

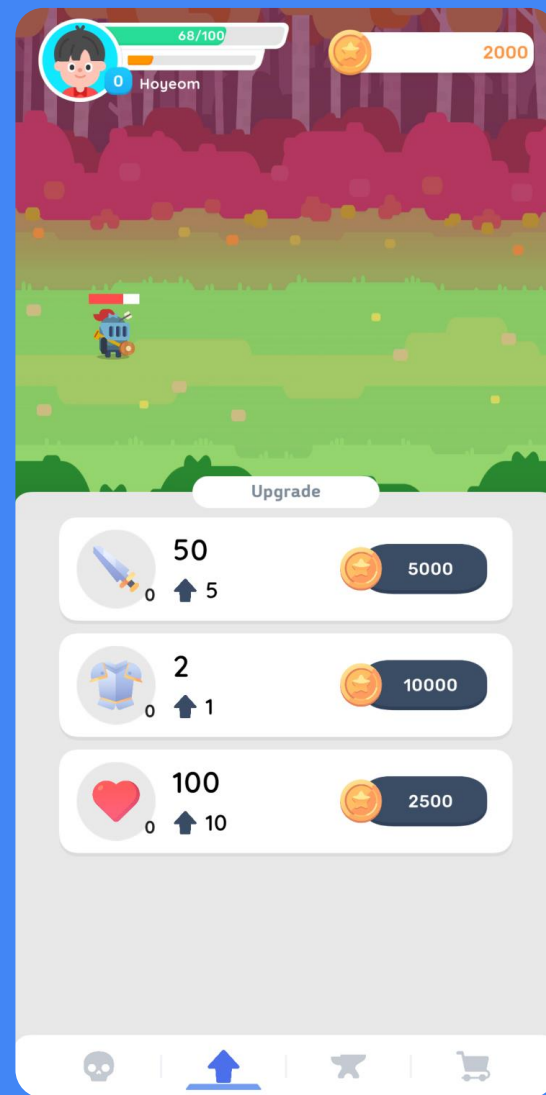


1. 캐릭터 정보

강화 수치와 현재 능력치와 강화되는 값을 출력한다

2. 업그레이드 버튼

업그레이드 비용을 표기하고 터치 시
비용이 현재 재화의 양과 같거나
작다면 업그레이드한다



|제작

일정 시간이 지난 후 모루를 터치시
아이템을 생성하고 아이템을
드래그 드롭 하여 업그레이드 합니다

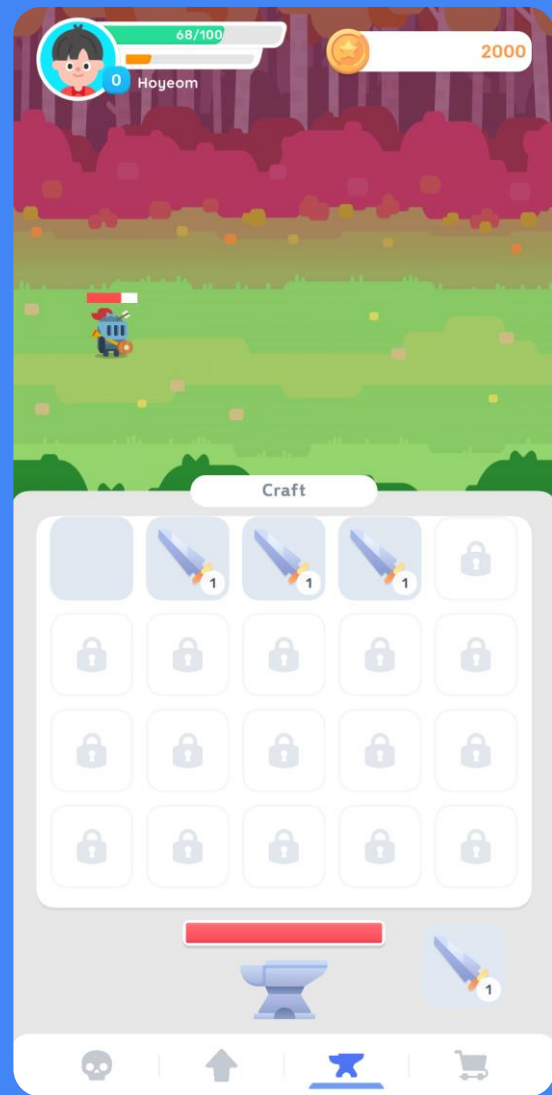


1. 아이템 슬롯

강화 수치, 잠김 여부를 출력하고
같은 레벨의 아이템을 합칠 경우 업그레이드 된다

2. 제작 버튼

게이지가 가득 차 있을 때 모루를 터치 시
플레이어 레벨에 따른 아이템을 생성한다



|상점

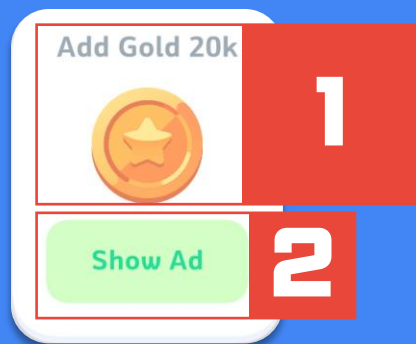
보상형 광고를 통해 재화를
수급할 수 있는 상점입니다.

1. 상품 정보

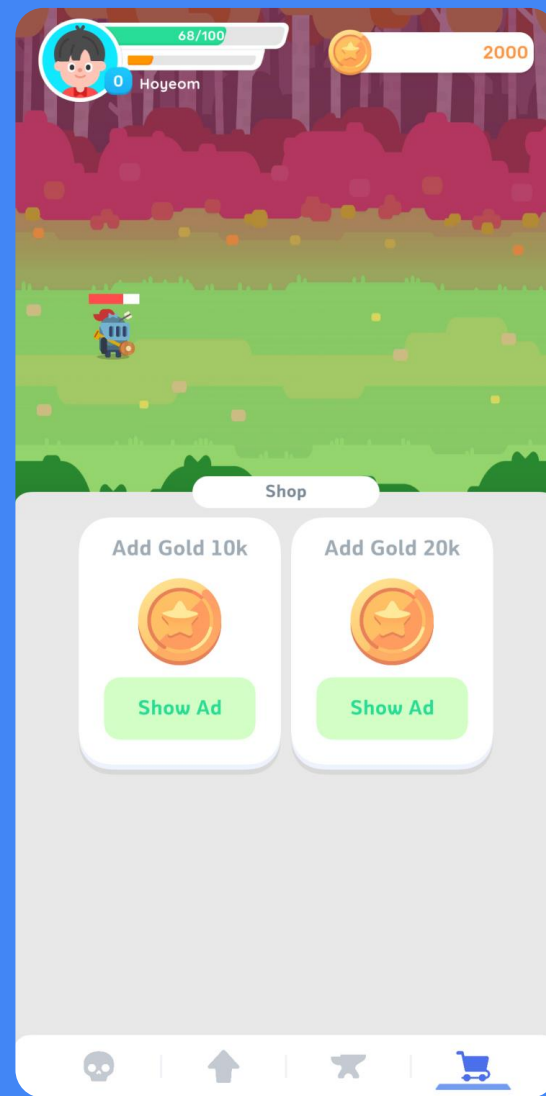
상품의 이름과 이미지를
출력한다

2. 구매 버튼

터치 시 비용지불 하며 보상을 받는다



비용지불 (ex: 광고, 결제)



일정/데이터 관리

진행 일정

테이블

자동화

| 진행 일정

조건부 서식과 MATCH 함수를 이용해
일정의 시작과 목표 일정을 표시하고
진행함에 따라 진행과 완료 표시를 한다

[illegible]

|테이블

엑셀로 작업한 이후 csv 파일로 저장 후

ScriptableObject로 변환하여 사용하였다

사용 이유는 어드레서블 에셋을 이용하기 위해 사용하였다

문서 번호	Table	CostType	ItemType	UpgradeType	Language	EnemyType	Cost
0	None	None	None	Attack	Eng	Unknown	Cash
1	Status	Cash	Weapon	Defence	Kor	Boss	Coin
2	Boss	Coin		Health			Ads
3	_						
4	Upgrade						
5	Weapon						
6	Shop						
7	StartStatus						
8	String						
9	Path						

|자동화

자동화 하게 된 이유

개발을 하다 보니 엑셀에 수정사항이 생길 때가 많은데
그때 마다 csv파일로 변경 하다 보니
시간이 오래 걸리게 되어 작업을 효율적으로 하기위해
엑셀을 csv 파일로 저장하는 작업을 자동화 하기로 결정했습니다

|자동화

개선

C#을 이용해 엑셀을 csv파일로 저장하는

프로그램을 만들고

유니티 메뉴에 추가하여 사용

```
[MenuItem("Parser/Excel To Csv")]
Hoyeom
private static void ExcelToCsv()
{
    Process p = new Process();
    p.StartInfo.UseShellExecute = true;
    p.StartInfo.FileName = ExternalExePath;

    p.Start();
}
```

```
Parser Window Help
Excel To Csv
Load All Data

internal class Program
{
    private static string _tablePath = null;
    private static string _csvPath = null;
    private const int CsvUtf8Format = 62;

    Hoyeom
    public static void Main(string[] args)
    {
        InitPath();

        if(!Directory.Exists(_tablePath))
            Console.WriteLine($"Missing Directory: {_tablePath}");

        CreatePath(_csvPath);

        OpenExcel(out Excel.Workbook xlWorkbook);

        foreach (Excel.Worksheet sheet in xlWorkbook.Worksheets)
            SaveSheet(sheet, _csvPath);

        xlWorkbook.Close(SaveChanges: false);
    }

    [1] usage Hoyeom *
    private static void InitPath(){...}
    [1] usage Hoyeom
    private static void OpenExcel(out Excel.Workbook xlWorkbook){...}
    [1] usage Hoyeom
    private static void SaveSheet(Excel.Worksheet sheet, string savePath){...}
    [1] usage Hoyeom
    private static void CreatePath(string path){...}
}
```

|자동화

자동화 도중 생긴 문제

유니티에서 Microsoft.Office.Interop.Excel.dll

호환되지 않는 문제를 겪었습니다

이것을 해결하기 위해 유니티에서는

프로그램만 실행하는 방법을 택하였고

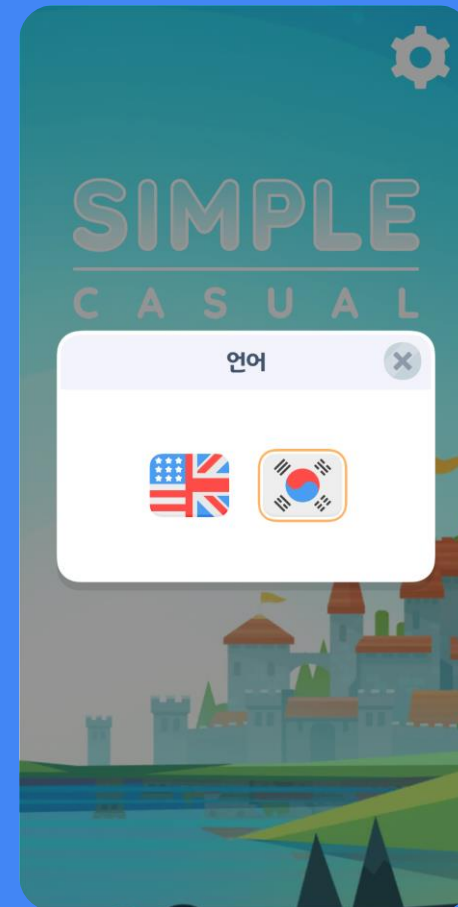
C#을 프로그램을 제작하여 exe 로 빌드하여

유니티 에서 실행하였습니다.

|로컬라이징

다양한 사람이 즐길 수 있도록
언어별 데이터를 관리하여
인 게임 설정에 따라
언어 변경이 가능하게 제작 하였습니다

ID	Eng	Kor
810001	Touch To Play	터치하여 시작
810002	Start Game	게임 시작
810003	Boss	보스
810004	Upgrade	강화
810005	Craft	제작
810006	Shop	상점
810008	Language	언어
810009	You Name	당신 이름
810010	Select	결정
810011	UserName	사용자이름
810012	Fight	싸우기
810013	ExitGame	게임 종료
810014	Really?	진짜?
810015	Exit	나가기



스크립트

플레이어

전투

세이브/로드

싱글 톤 사용 이유

| 플레이어

플레이어의 정보를 가지고 있으며
변경에 따른 이벤트를 발생시키는 클래스

코인의 값이 0 밑으로 내려가지 않도록
Max 함수를 이용해 처리하며
이후 이벤트를 발생시킨다

```
3 usages new *
public string Name{...}
10 usages new *
public int Coin
{
    🔥 Frequently called
    get => _coin;
    🔥 Frequently called
    set
    {
        _coin = Mathf.Max(a: value, b: 0);
        OnChangeCoin?.Invoke(_coin);
    }
}
3 usages Hoyoem
public int Exp{...}
3 usages Hoyoem
public int LevelID{...}
```

| 전투

공격이 끝난 이후 턴종료를 알리기 위해
TurnEnd 함수를 콜 백 하도록 만들었으며
플레이어 또는 적이 죽을 때까지
재귀적으로 번갈아 가며 호출하며 작동한다

```
switch (_turn)
{
    case Turn.PlayerTurn:

        _turn = Turn.EnemyTurn;
        Enemy.Attack(TurnEnd);

        break;
    case Turn.EnemyTurn:

        _turn = Turn.PlayerTurn;
        Player.Attack(TurnEnd);

        break;
}
```

|세이브/로드

플레이어 데이터

json으로 저장하기 위해 필요한 데이터들을
직렬화 하고 인벤토리 내 아이템들을
아이템 ID가 담긴 List<int> 를 직렬화 하여
로드가 되도록 하였으며
반환 값을 bool로 하여
로드 성공 실패 여부를
판단하고 새로운 파일을 만들도록 작성하였습니다.

```
private bool LoadGame()
{
    if (File.Exists(_savePath) == false)
        return false;

    string saveText = File.ReadAllText(_savePath);
    SaveData saveData = JsonUtility.FromJson<SaveData>(saveText);




    if (saveData != null)
    {
        SaveData = saveData;
        saveData.Inventory.LoadData();
        Managers.Data.Language = saveData.Language;
    }

    Debug.Log(message: $"Save Game Loaded : {_savePath}");
    return true;
}
```

|세이브/로드



인 게임 데이터

데이터를 아마존 S3 저장소에 업로드하여
관리하였고 데이터를 불러오고
완료한 이후 게임을 시작합니다

 200002 (Enemy Data)  

Open

☒ Addressable Data/Enemy/200002 Select

Script	EnemyData 
ID	200002
Enemy Type	Boss 
Icon ID	930002
Health	150
Attack Power	20
Defence Power	10
Drop Coin	5000
Drop Exp	20



|싱글 톤 사용 이유

UI와 데이터를 분리하기 위해
싱글 톤 방식을 사용했습니다

값이 변함에 따라 이벤트를 발생시키고
UI가 구독하여 보이는 값을
변경할 수 있도록 작성하였습니다.

```
public override void SetInfo()  
{  
    UnitImage.sprite = Managers.Game.Player.Sprite;  
    Managers.Game.Player.OnChangeHealth -= SetHealthSlider;  
    Managers.Game.Player.OnChangeHealth += SetHealthSlider;  
}
```

최적화

캔버스

UI 요소

| 캔버스

캔버스 나누기

처음 개발을 할 때 하나의 캔버스로 작업을 하였는데
오브젝트가 하나 움직일 때 마다 스파이크가 생기는 것을 확인하였고
이후 동적 UI는 캔버스를 나누는 형태를 사용하며 최적화를 했습니다

UI 숨기기

게임 오브젝트를 비활성화 하는 대신
캔버스 컴포넌트만 비활성화 하여 숨겼습니다
이로 인해 활성화 시 메시와 버텍스를 재구성하지 않게 되었습니다

| UI 요소

Layout Group

레이아웃 그룹 사용 이후 컴포넌트를
비활성화하여 추가적인
성능 저하를 막았습니다

Tween

애니메이터를 사용할 경우 애니메이션이 사용 중이 아니더라도
다시 그리게 되는 문제를 해결하기 위해
Tween 을 사용하여 간단한 애니메이션을 적용하였습니다

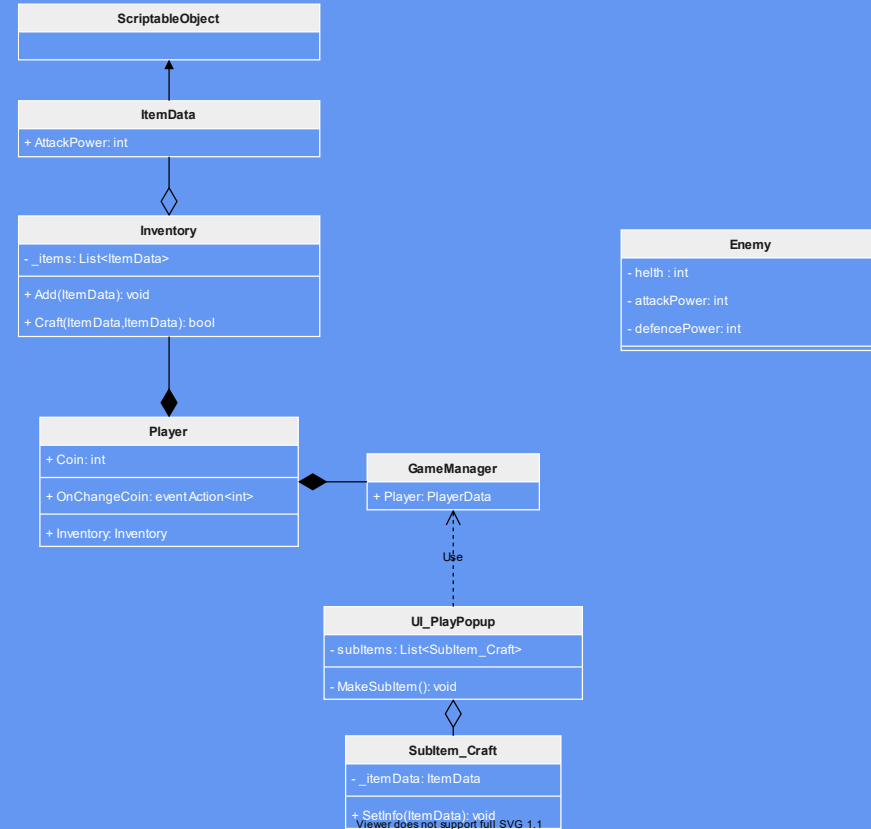
마치며



| 아쉬운 점

UML 활용

계속해서 좋은 구조를 찾아보며 작업하다 보니
처음 UML로 만들어 둔 구조에서
변형된 것이 있어서 앞으로 구조에 대해
더욱 신경 쓰면 좋을 것 같습니다



감사합니다

