

# 포트폴리오

---

-클라이언트 프로그래머 지원-

-서현재-

기술스택

- DirectX
- 백준 플레
- C#
- Unity
- 형상관리 툴
  - Git
  - GithubDesktop

# 목차

## 1. The Sword

스팀 출시 예정작

## 2. MessMath

3회 웅진씽크빅 게임 개발 챌린지 본선 진출

## 3. 던전포차

인디크래프트 출판작

## 4. UpToSky

구글 플레이 출시 게임

# The Sword

# 타이틀 화면 및 게임 영상

<https://youtu.be/Swc5l2-JCMQ>

# The Sword

개요: RPG형식 1인 전략 게임

개발 기간 : 2024.05~

사용 도구 : Unity, C#, Git, GithubDesktop

개발 인원 : 4인(프로그래머2, 아트1, 기획자1)

담당 업무: 던전 맵 자동생성 툴 제작, 전투 게임 로직 구현, UI자동화, Excel 데이터 연동, 다국언어

# UI 자동화

기술적인 도전과제	코드 내에서 UI 관리하기
도입배경	Hierarchy창에서 Drag Drop방식은 코드 내에서 로직을 쉽게 파악하기 어렵다
개선된 사항	코드를 보면서 로직을 총괄적으로 관리 가능

```
public class UI_PlayerCard : UI_Base
{
    #region Enum
    enum Images
    {
        PlayerImage,
        HPBar,
        HPBarGauge,
        AttackDelayGauge,
        DefenceDelayGauge,
    }
}
```

1. UI Prefab 속 UI 이름을 Enum에 적기

```
public override bool Init()
{
    if (base.Init() == false)
        return false;

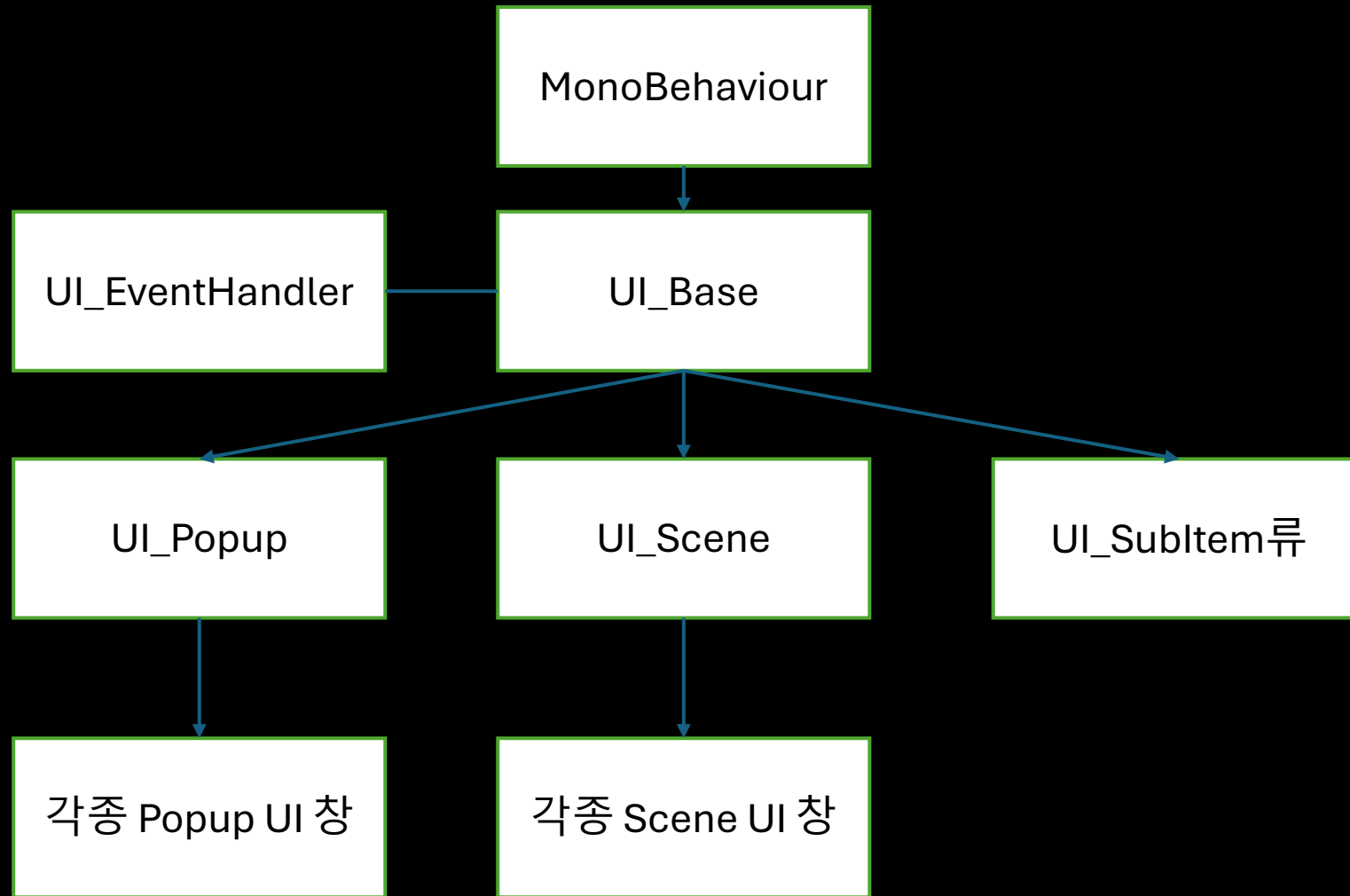
    #region Bind
    BindImage(typeof(Images));
    BindText(typeof(Texts));
    #endregion
}
```

2. 각 속성에 맞게 Bind

```
GetText((int)Texts.CreatureName).text = "Player!";
GetText((int)Texts.HPBarText).text = Managers.Game.CurPlayerData.CurHP.ToString();
GetText((int)Texts.AttackStatusText).text = Managers.Game.CurPlayerData.Attack.ToString();
GetText((int)Texts.DefenceStatusText).text = Managers.Game.CurPlayerData.Defence.ToString();
```

3. 코드로 불러올때는 Get~(enum의 인덱스)로 불러오기

# UI 상속 구조



# UI\_Base

```
public abstract class UI_Base : MonoBehaviour
{
    ...protected Dictionary<Type, UnityEngine.Object[]> _objects = new Dictionary<Type, UnityEngine.Object[]>();
    ...protected bool _init = false;
```

UI\_Base로는 사용하지  
않으므로 abstract로  
구현되었다.  
Dictionary로 오브젝트들을  
삽입하여 관리할 예정.

```
protected void Bind<T>(Type type) where T : UnityEngine.Object
{
    ...string[] names = Enum.GetNames(type);
    ...UnityEngine.Object[] objects = new UnityEngine.Object[names.Length];
    ..._objects.Add(typeof(T), objects);

    ...for (int i = 0; i < names.Length; i++)
    ...{
        ...if (typeof(T) == typeof(GameObject))
        ...{
            ...objects[i] = Util.FindChild(gameObject, names[i], true);
        }
        ...else
        ...{
            ...objects[i] = Util.FindChild<T>(gameObject, names[i], true);
        }

        ...if (objects[i] == null)
        ...{
            ...Debug.Log($"Failed to bind({names[i]})");
        }
    }
}
```

C#의 reflection을 이용하여 Enum을  
읽어올 수 있다.  
또 generic을 이용하여 어떤 타입이  
들어와도 가능하게 만들었다.

```
protected void BindObject(Type type) { Bind<GameObject>(type); }
protected void BindImage(Type type) { Bind<Image>(type); }
protected void BindText(Type type) { Bind<TMP_Text>(type); }
protected void BindButton(Type type) { Bind<Button>(type); }
protected void BindToggle(Type type) { Bind<Toggle>(type); }
```

자주 사용되는 타입들은 헬퍼함수로  
만들어서 더 쉽게 사용하도록 만들었다.

# UI\_Base

```
protected T Get<T>(int idx) where T : UnityEngine.Object
{
    ... UnityEngine.Object[] _objects = null;
    ... if (_objects.TryGetValue(typeof(T), out objects) == false)
    ...     return null;

    ... return objects[idx] as T;
}
```

Get함수는 Bind된  
오브젝트들이 Dictionary에  
들어가 있으므로 그것을  
index에 맞는 오브젝트로  
불러온다.

```
protected GameObject GetObject(int idx) { return Get<GameObject>(idx); }
protected TMP_Text GetText(int idx) { return Get<TMP_Text>(idx); }
protected Button GetButton(int idx) { return Get<Button>(idx); }
protected Image GetImage(int idx) { return Get<Image>(idx); }
protected Toggle GetToggle(int idx) { return Get<Toggle>(idx); }
```

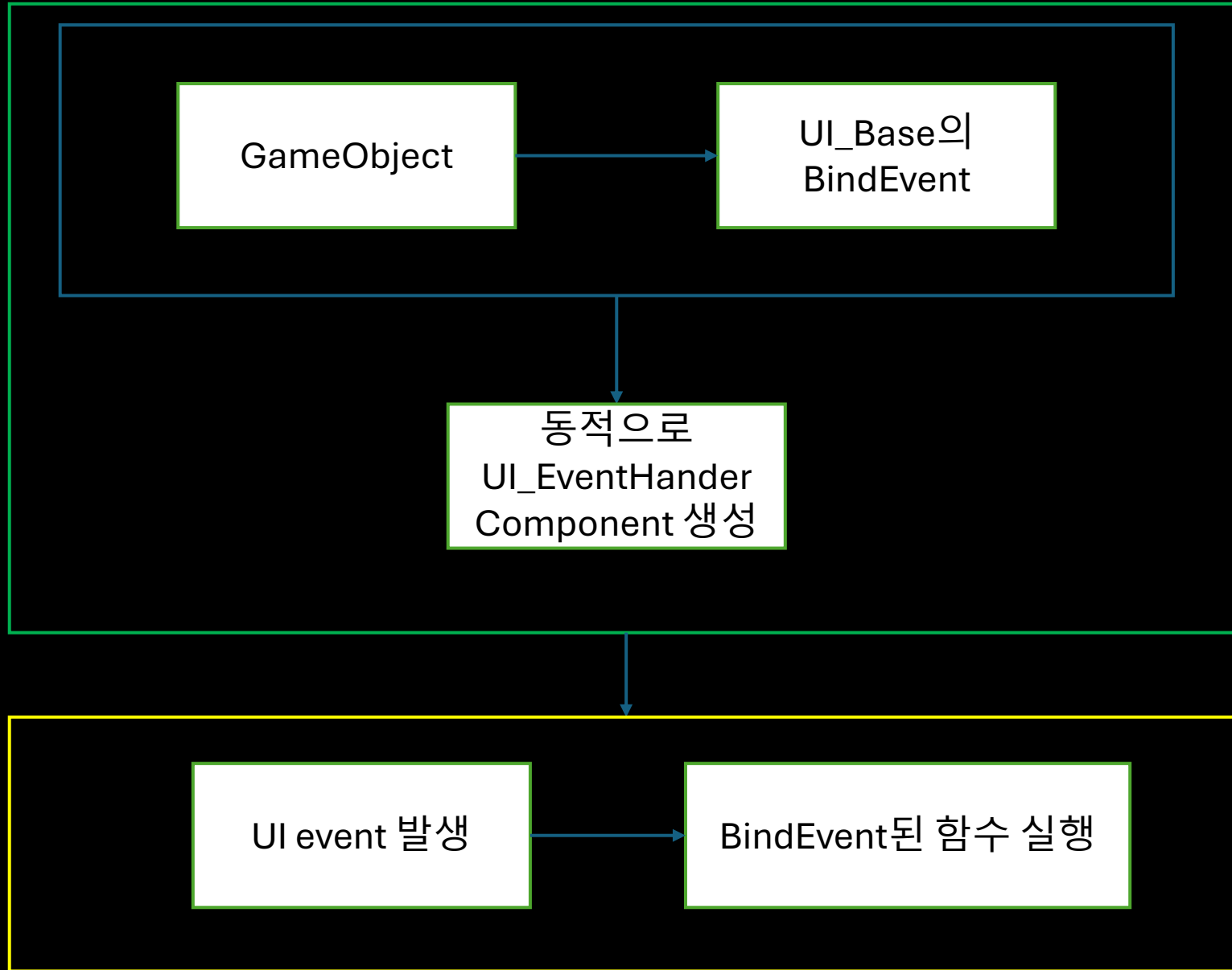
자주 사용되는 타입들은 헬퍼함수로  
만들어서 더 쉽게 사용하도록 만들었다.

```
GetText((int)Texts.CreatureName).text = "Player!";
GetText((int)Texts.HPBarText).text = Managers.Game.CurPlayerData.CurHP.ToString();
GetText((int)Texts.AttackStatusText).text = Managers.Game.CurPlayerData.Attack.ToString();
GetText((int)Texts.DefenceStatusText).text = Managers.Game.CurPlayerData.Defence.ToString();
```

이런식으로 코드내에서 쉽게  
변경가능하다.



# UI\_EventHandler 의사코드



# UI\_EventHandler

UI의 Event들을 관리하게하는 클래스이다.

```
public class UI_EventHandler : MonoBehaviour, IPointerClickHandler, IPointerDownHandler,
{
    public Action OnClickHandler = null;
    public Action OnPressedHandler = null;
    public Action OnPointerDownHandler = null;
    public Action OnPointerUpHandler = null;
    public Action OnPointerEnterHandler = null;
    public Action OnPointerExitHandler = null;
    public Action<BaseEventData> OnDragHandler = null;
    public Action<BaseEventData> OnBeginDragHandler = null;
    public Action<BaseEventData> OnEndDragHandler = null;

    bool _pressed = false;

    private void Update()
    {
        if (_pressed)
        {
            OnPressedHandler?.Invoke();
        }
    }
}
```

```
public void OnPointerClick(PointerEventData eventData)
{
    if (OnClickHandler != null)
    {
        OnClickHandler.Invoke();
    }
}

public void OnPointerDown(PointerEventData eventData)
{
    _pressed = true;
    OnPointerDownHandler?.Invoke();
}
```

Action에 구독하고 Event가 호출될때 연동된 함수가  
실행된다.

```
public static void BindEvent(GameObject go, Action action = null, Action<BaseEventData> dragAction = null, Define.UIEvent type = Define.UIEvent.Click)
{
    UI_EventHandler evt = Util.GetOrAddComponent<UI_EventHandler>(go);

    switch (type)
    {
        case Define.UIEvent.Click:
            evt.OnClickHandler -= action;
            evt.OnClickHandler += action;
            break;
        case Define.UIEvent.Pressed:
            evt.OnPressedHandler -= action;
            evt.OnPressedHandler += action;
            break;
        case Define.UIEvent.PointerDown:
            evt.OnPointerDownHandler -= action;
            evt.OnPointerDownHandler += action;
            break;
        case Define.UIEvent.PointerUp:
            evt.OnPointerUpHandler -= action;
            evt.OnPointerUpHandler += action;
            break;
        case Define.UIEvent.Drag:
            dragAction -= action;
            dragAction += action;
            break;
    }
}
```

앞선 UI\_EventHandler가 UI\_Base에서  
BindEvent함수속에서 사용된다.

# UI\_EventHandler

```
public static void BindEvent(this GameObject go, Action action = null, Action<BaseEventData> dragAction = null, Define.UIEvent type = Define.UIEvent.Click)
{
    UI_Base.BindEvent(go, action, dragAction, type);
}
```

Extionsion기능을 이용하여 GameObject에서 .을붙여서  
BindEvent함수를 사용할 수 있게 만든다.

```
Define.UIEvent type = Define.UIEvent.Click;
```

기본적으로는 click을 사용하게 만들었다.

```
public enum UIEvent
{
    Click,
    Preseed,
    PointerDown,
    PointerUp,
    BeginDrag,
    Drag,
    EndDrag,
    PointerEnter,
    PointerExit,
}
```

다른 이벤트들도 쉽게  
구독할수있는  
함수이다.

```
public static void BindEvent(GameObject go, Action action = null, Action<BaseEventData> dragAction = null, Define.UIEvent type = Define.UIEvent.Click)
{
    UI_EventHandler evt = Util.GetOrAddComponent<UI_EventHandler>(go);

    switch (type)
    {
        case Define.UIEvent.Click:
            evt.OnClickHandler -= action;
            evt.OnClickHandler += action;
            break;
        case Define.UIEvent.Preseed:
            evt.OnPressedHandler -= action;
            evt.OnPressedHandler += action;
            break;
        case Define.UIEvent.PointerDown:
            evt.OnPointerDownHandler -= action;
            evt.OnPointerDownHandler += action;
            break;
        case Define.UIEvent.PointerUp:
            evt.OnPointerUpHandler -= action;
            evt.OnPointerUpHandler += action;
            break;
        case Define.UIEvent.Drag:
            evt.OnDragHandler -= dragAction;
            evt.OnDragHandler += dragAction;
            break;
    }
}
```

# UI\_EventHandler

```
public static void BindEvent(GameObject go, Action action = null, Action<BaseEventData> dragAction = null, Define.UIEvent type = Define.UIEvent.Click)
{
    UI_EventHandler evt = Util.GetOrAddComponent<UI_EventHandler>(go);
    switch (type)
    {
        case Define.UIEvent.Click:
            evt.OnClickHandler -= action;
            evt.OnClickHandler += action;
            break;
        case Define.UIEvent.Pressed:
            evt.OnPressedHandler -= action;
            evt.OnPressedHandler += action;
            break;
        case Define.UIEvent.PointerDown:
            evt.OnPointerDownHandler -= action;
            evt.OnPointerDownHandler += action;
            break;
        case Define.UIEvent.PointerUp:
            evt.OnPointerUpHandler -= action;
            evt.OnPointerUpHandler += action;
            break;
        case Define.UIEvent.Drag:
    }
```

→  
앞선 UI\_EventHandler가 UI\_Base에서  
BindEvent함수속에서 사용된다.

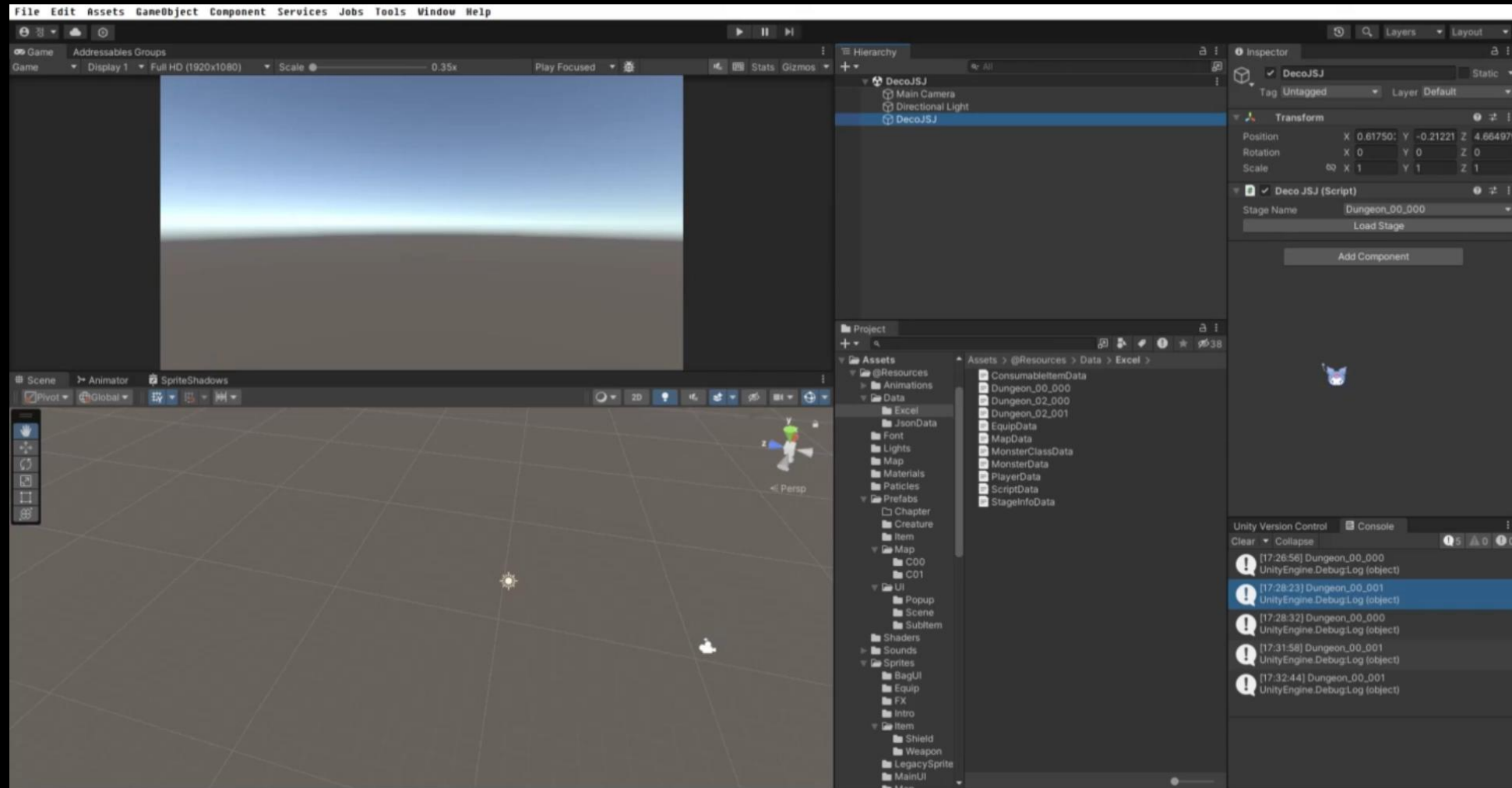
```
public static void BindEvent(this GameObject go, Action action = null, Action<BaseEventData> dragAction = null, Define.UIEvent type = Define.UIEvent.Click)
{
    UI_Base.BindEvent(go, action, dragAction, type);
}
```

↓  
Extionsn기능을 이용하여 GameObject에서 .을붙여서  
BindEvent함수를 사용할 수 있게 만든다.

사용 예시

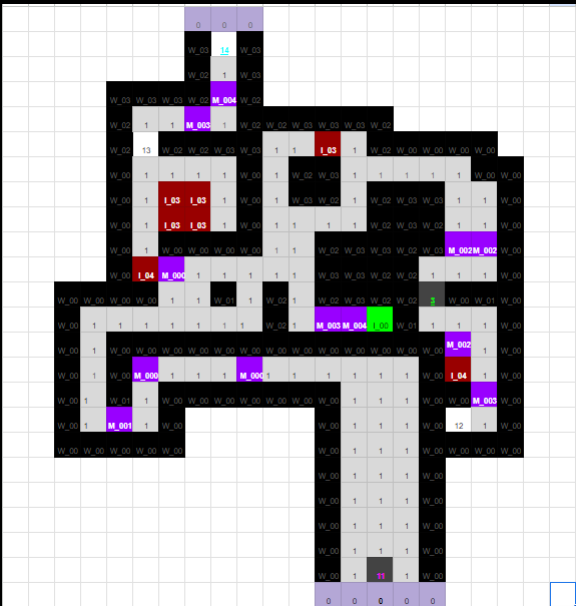
↓  
`GetImage((int)Images.MainUIInventoryImage).gameObject.BindEvent(OnClickMainUIInventoryImage);`

# Map 자동생성 Tool 영상



# Map 자동생성 Tool

Map을 생성하기 위해서 그 데이터를  
Exel에 정리한다.



Exel의 cell에 기입한 글자가 어떤  
데이터를 나타내는지 정의한다.

-	공허 벽	0	0	공허	지나갈 수 없지만, 벽이아름한 공간
3D	바닥	1	1	바닥	움직일 수 있는 바닥 타일
3D	Tilemap_W00	00	W_ W_00	벽 00	수벽
3D	Tilemap_W01	01	W_ W_01	벽 01	나무 그루터기
3D	Tilemap_W02	02	W_ W_02	벽 02	부서진 유적 벽
3D	Tilemap_W03	03	W_ W_03	벽 03	온전한 유적 벽
3D	Tilemap_W04	04	W_ W_04	벽 04	
3D	Tilemap_W05	05	W_ W_05	벽 05	
3D	Tilemap_W06	06	W_ W_06	벽 06	
3D	Tilemap_W07	07	W_ W_07	벽 07	
3D	Tilemap_W08	08	W_ W_08	벽 08	
3D	Tilemap_W09	09	W_ W_09	벽 09	
3D	Tilemap_W10	10	W_ W_10	벽 10	
3D	Tilemap_Door_G	3	3	조름 문 / 가로 / —	조름 열쇠가 있어야 열리는 문
3D		4	4	조름 문 / 세로 /	
3D	Tilemap_Door_Y	5	5	노랑 문 / 가로 / —	노랑 열쇠가 있어야 열리는 문
3D		6	6	노랑 문 / 세로 /	
3D	Tilemap_Door_R	7	7	빨강 문 / 가로 / —	빨은 열쇠가 있어야 열리는 문
3D		8	8	빨강 문 / 세로 /	
2D	Tilemap_StairsUp0	9	9	계단 상 / 철타0	다음 층으로 올라가는 계단 ▲
2D	Tilemap_StairsDown0	10	10	계단 하 / 철타0	이전 층으로 내려가는 계단 ▼
2D	Tilemap_SpawnPoint	11	11	스폰 포인트	캐릭터 최초 스폰 영역
2D	Tilemap_SteelLever	12	12	기믹 / 레버	레버 작동 시, 밑에 모든 철타가 해제.
3D	Tilemap_SteelDoor	13	13	기믹 / 철타 문	평소엔 닫혀있는 철타 문, 레버로 열 수 있다
2D	Tilemap_Road	14	14	일반 길 / 상	계단과 동일한, 길 끝에 닿으면 다음 밑에 이동▲
2D	Tilemap_Road	15	15	일반 길 / 하	계단과 동일한, 길 끝에 닿으면 다음 밑에 이동▼
2D	Tilemap_BossRoad	16	16	보스 길	계단과 동일한, 길 끝에 닿으면 보스 밑에 이동
2D	Tilemap_BossRoad	17	17	투트 마검	투트리의 꽃겨있는 마검
2D	Tilemap_ChapterClear	18	18	철타 클리어 문	보스 처치 시, 북인이 틀려 다음 층으로 이동 가는

# Map 자동생성Tool

Excel 데이터를 바탕으로 map을 생성할 수 있도록 Excel을 파싱한다.

```
#if UNITY_EDITOR
// % (Ctrl), # (Shift), & (Alt)
[MenuItem("Tools/CreateMap %#c")]
private static void CreateMap()
{
    #region ExcelData
    string[] lines = File.ReadAllText($"{Application.dataPath}/@Resources/Data/Excel/MapData.csv").Split("\n");
```

각 타일의 이름에 맞는 오브젝트를 간격에 맞게 생성한다.

```
for (int y = 0; y < lines.Length; y++)
{
    string[] row = lines[y].Replace("\r", "").Split(',');

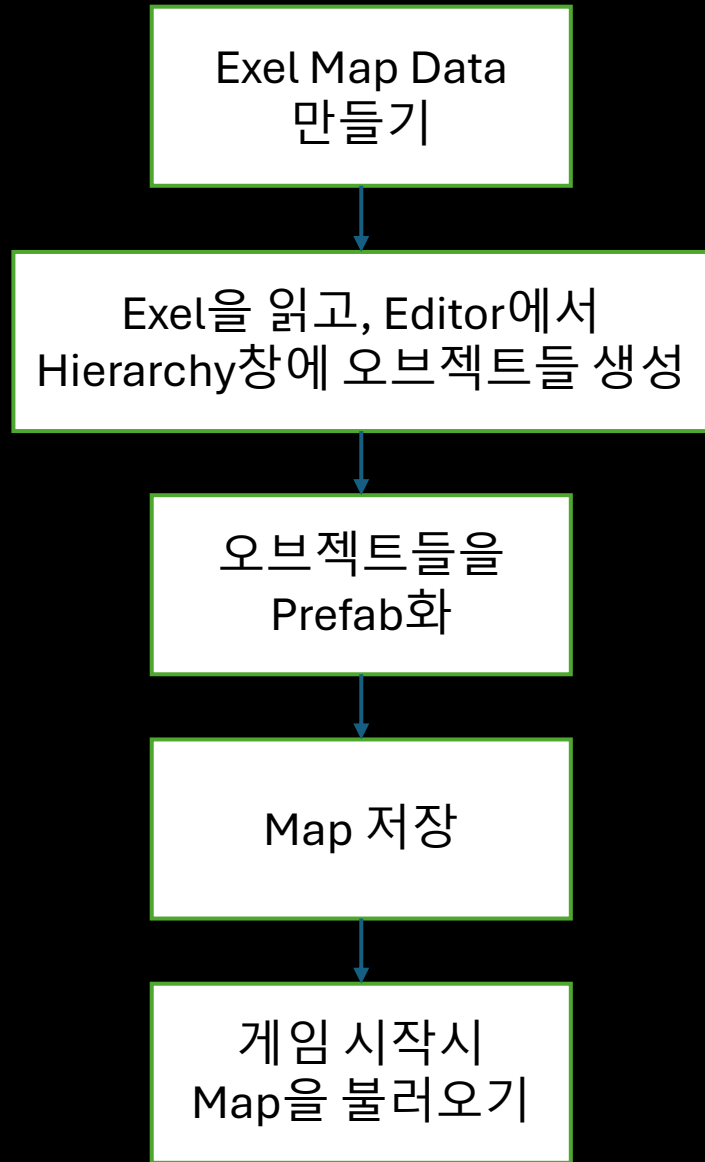
    if (row.Length == 0)
        continue;
    if (string.IsNullOrEmpty(row[0]))
        continue;

    for (int x = 0; x < row.Length; ++x)
    {
        string block = row[x];

        if (block == "0")
        {
            GameObject voidObject = Resources.Load<GameObject>($"Tilemap_0");
            toAdd = voidObject.GetComponentInChildren<BoxCollider>().size.x;
            UnityEngine.Object.Instantiate(voidObject, new Vector3(coX, coY + addToFloorY, coZ), Quaternion.identity, parent.transform);
        }
    }
}
```

생성된 맵 오브젝트들을 하나의 프리팹으로 만든 뒤, 맵을 저장한다.

# Map 생성 로직





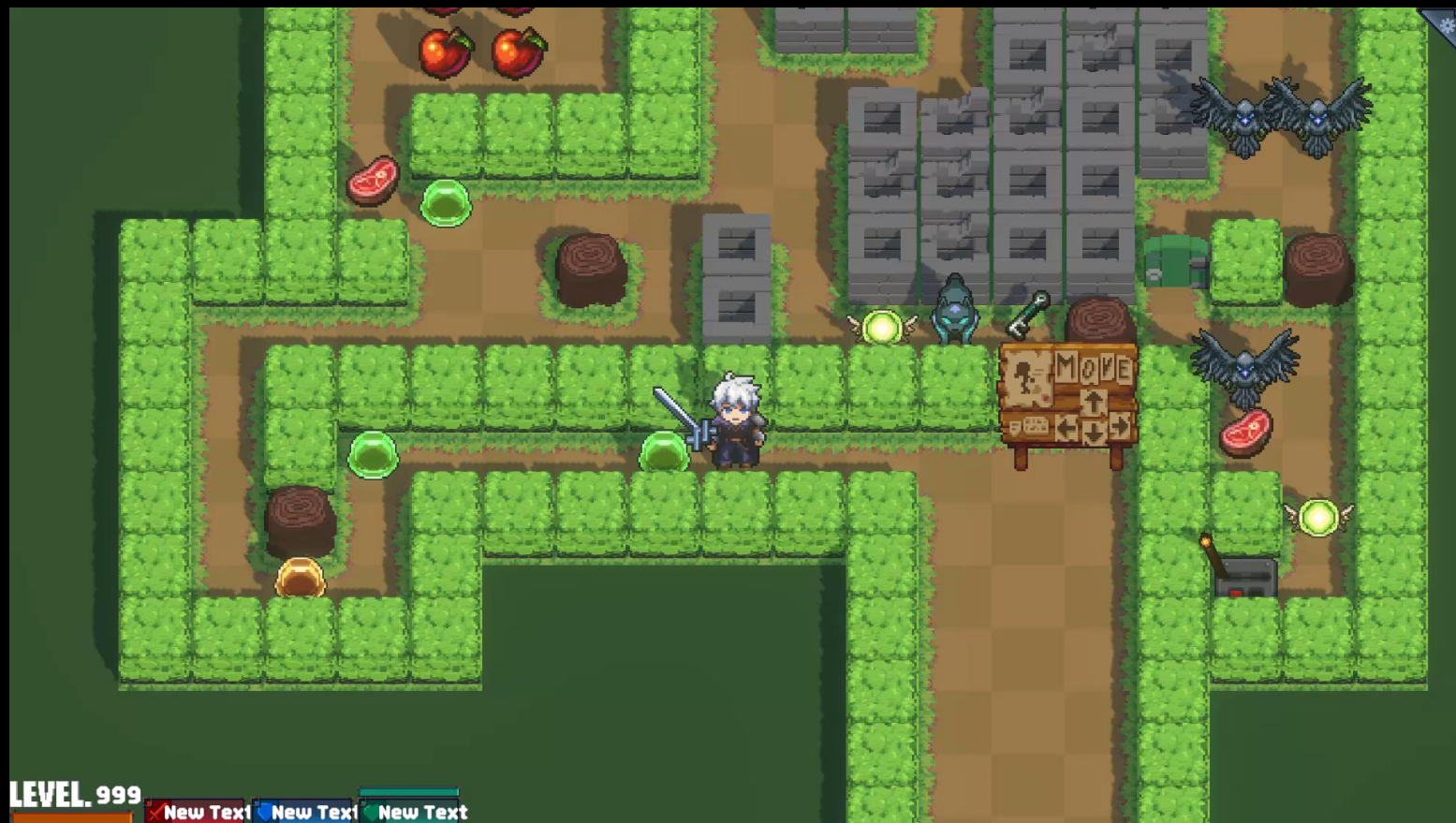
# Map 생성 고찰 및 개선

문제점	개선
맵이 단층으로만 만들수있다.	2층 이상은 층마다 데이터를 따로 만들어서 병합.
타일이 많아질수록 규칙이 늘어난다.	챕터, 스테이지에 따라 컨벤션을 정해서 간략한 규칙으로 쉽게 생성되게 변경했다.
몬스터 데이터와 연동을 어떻게 할것인지	예를들어 M_12 이런 형식으로 MonsterData의 index를 확인하여 그것에맞게 몬스터를 생성한다. 몬스터 생성시 기본 monster prefab으로 생성하고 id값만 달라지면 다른 몬스터가 생성되게 만들었다.

스테이지 구성 정보			윗층 / 던전 ID	아래층 / 던전 ID	보스방 / 던전 ID				바닥 필드 타일		스크립트	
D	Dungeon_ID	Type	Up_Stairs	Down_Stairs	Boss_Room	ATK_배수 (n)	DEF_배수 (n)	EXP_배수 (%)	FloorField	BGM	맵 이름_ID	설명
0	00_000	Normal	00_001	-	-	0	0	100	00_000	BGM_000	5000	튜토리얼_#0
1	00_001	Normal	00_002	00_000	00_003	1	1	100	00_001	BGM_000	5000	튜토리얼_#1
2	00_002	Combine	-	00_001	-	4	4	100	99_999	BGM_001	5001	튜토리얼_마검방
3	00_003	Boss	00_004	-	-	5	5	200	00_003	BGM_002	5002	튜토리얼_보스방
4	00_004	Normal	-	01_000		0	0	100	00_004	BGM_003	5003	가브마을
5	01_000	Normal	01_001	-		6	6	100	01_000	BGM_100		타워_1절터_#0
6	01_001	Normal	01_002	01_000		7	7	100	01_001	BGM_100		타워_1절터_#1
7	01_002	Normal	01_003	01_001		8	8	100	01_002	BGM_100		타워_1절터_#2
8	01_003	Normal	01_004	01_002		9	9	100	01_003	BGM_100		타워_1절터_#3
9	01_004	Normal	01_005	01_003		10	10	100	01_004	BGM_100		타워_1절터_#4
10	01_005	Normal	01_006	01_004		11	11	200	01_005	BGM_100		타워_1절터_#5
11	01_006	Normal	01_007	01_005		12	12	300	01_006	BGM_100		타워_1절터_#6
12	01_007	Normal	01_008	01_006		13	13	400	01_007	BGM_100		타워_1절터_#7
13	01_008	Normal	01_009	01_007		14	14	500	01_008	BGM_100		타워_1절터_#8
14	01_009	Special	01_010	-		15	15	600	01_009	BGM_101		타워_1절터_특수
15	01_010	Boss	-	-		16	16	700	01_010	BGM_102		타워_1절터_보스방

챕터, 스테이지에 따라 정해진 컨벤션

# 전투 영상



# 전투 로직

CurPlayerData

현재 player data는  
GameManager에서 관리한다.

```
public class ContinueData
{
    public int Level { get; set; } = 1; // Lv
    public float curExp;
    public float CurExp;
    public float MaxHP { get; set; }
    public float CurHP { get; set; }
    public float Attack { get; set; }
    public float Defence { get; set; }
    public float AttackSpeed { get; set; }
    public float DefenceSpeed { get; set; }
    public float Critical { get; set; }
    public float CriticalAttack { get; set; }
    public float MoveSpeed { get; set; }
    public bool IsDefence { get; set; }
    //public Dictionary<int, int> Inventory = new Dictionary<int, int>();
    public List<List<int>> Inventory = new List<List<int>>();
    public List<int> KeyInventory = new List<int>();
    public int CurSword { get; set; }
    public int CurShield { get; set; }
    public int CurNecklace { get; set; }
    public int CurRing { get; set; }
    public int CurShoes { get; set; }
}
```

UI\_PlayerCard

전투는 자동 전투로 공격  
Coroutine이 전투가 끝이  
날때까지 반복해서 공격한다.

```
IEnumerator CoDelayAttack()
{
    float maxAttackCoolTime = 3f;
    float attackCoolTime = 0f;
    maxAttackCoolTime = maxAttackCoolTime / Managers.Game.CurPlayerData.AttackSpeed;

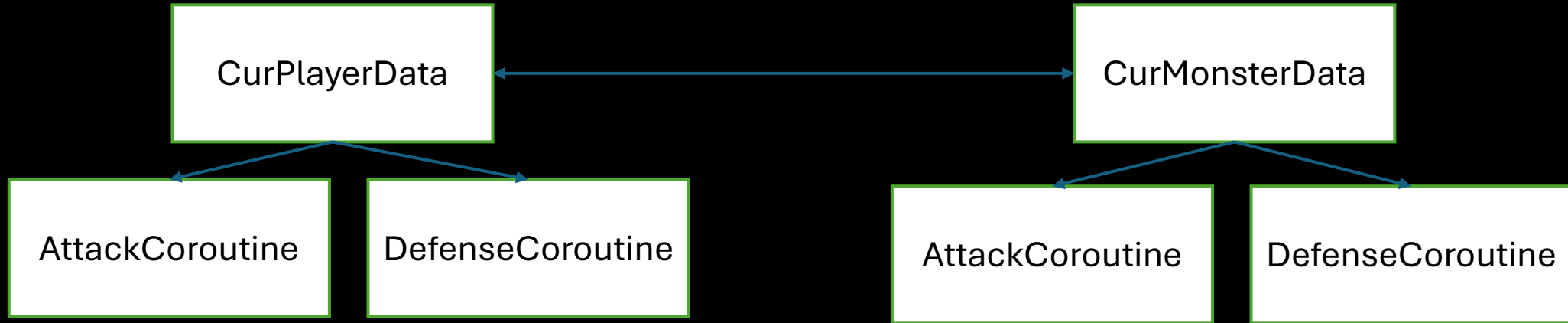
    while (true)
    {
        if (attackCoolTime >= maxAttackCoolTime)
        {
            attackCoolTime = 0f;
            Attack();
        }
        attackCoolTime += Time.deltaTime * Managers.Game.GameSpeed;

        GetImage((int)Images.AttackDelayGauge).fillAmount = attackCoolTime / maxAttackCoolTime;

        yield return new WaitForFixedUpdate();
    }
}
```

# 전투 로직

적과 부딪힐 시 자동 전투 진입



부딪힌 몬스터의 데이터를 가져와서 전투 후 결과 도출  
각자의 Coroutine이 돌아가서 공방을 주고 받는다.

데이터가 갱신될 때 마다 GameManager의  
CurPayerData가 변경되고 UI들을 Refresh한다.

# 전투 로직 트러블슈팅

Coroutine이 실행될 때 간헐적으로  
UI가 꺼졌는데도 Coroutine이  
실행돼서 에러를 내는 경우가 생겼다.

```
public void BattleEnd()
{
    Destroy(playerCard.gameObject);

    switch (Managers.Game.MonsterData.id)
    {
        case Define.KingSlime:
            Destroy(kingSlimeCard.gameObject);
            break;
        default:
            Destroy(monsterCard.gameObject);
            break;
    }

    Managers.Game.OnBattleDataRefreshAction = null;
    Managers.Game.OnBattleCreatureDefeceAction = null;
    Managers.Game.OnBattleCreatureDamagedAction = null;
    Managers.Game.OnBattleAction = null;
    if (Managers.Game.GameScene != null)
        Managers.Game.GameScene.SetPlayerInfo();
    Managers.Game.SaveGame();
    ClosePopupUI();
}
```

Action에 구독된 Coroutine들을  
null로 하고, Coroutine이  
실행될때에도 null 체크를해서  
에러를 없앴다.

# The Sword

## 데이터 연동

Lv	NeedEXP	TotalExp	공격력	방어력	체력	공격속도	방어속도	치명타	치명공격력	이동속도
1	0	0	8	2	50	5	10	10	200	5
2	70	70	1	1	5	0	0	0	0	0
3	76	146	1	1	5	0	0	0	0	0
4	82	228	1	1	5	0	0	0	0	0
5	89	317	1	1	5	0	0	0	5	0
6	97	414	1	1	5	0	0	0	0	0
7	105	519	1	1	5	0	0	0	0	0
8	114	633	1	1	5	0	0	0	0	0
9	124	757	1	1	5	0	0	0	0	0
10	135	892	2	2	10	0	0	0	5	0

Excel의 열은 데이터 종류, 행은 index에 맞는 데이터 값을 넣는다.

```
static void ParsePlayerData(string filename)
{
    PlayerDataLoader loader = new PlayerDataLoader();

    #region ExcelData
    string str = File.ReadAllText($"{Application.dataPath}/Resources/Data/Excel/{filename}Data.csv");
    Debug.Log(str);
    string[] lines = File.ReadAllText($"{Application.dataPath}/Resources/Data/Excel/{filename}Data.csv").Split("\n");
```

파일의 상대경로를 따라서 csv파일을 parsing하고 각 행을 배열에 저장한다.

```
    for (int y = 1; y < lines.Length; y++)
    {
        string[] row = lines[y].Replace("\r", "").Split(',');

        if (row.Length == 0)
            continue;
        if (string.IsNullOrEmpty(row[0]))
            continue;

        int i = 0;
        PlayerData cd = new PlayerData();
        cd.id = ConvertValue<int>(row[i++]);
        cd.NeedExp = ConvertValue<float>(row[i++]);
        cd.TotalExp = ConvertValue<float>(row[i++]);
        cd.Attack = ConvertValue<float>(row[i++]);
        cd.Defence = ConvertValue<float>(row[i++]);
        cd.MaxHP = ConvertValue<float>(row[i++]);
        cd.AttackSpeed = ConvertValue<float>(row[i++]);
        cd.DefenceSpeed = ConvertValue<float>(row[i++]);
        cd.Critical = ConvertValue<float>(row[i++]);
        cd.CriticalAttack = ConvertValue<float>(row[i++]);
        cd.MoveSpeed = ConvertValue<float>(row[i++]);
        loader.creatures.Add(cd);
    }
    #endregion
```

Csv파일은 \r 로 셀을 구분 지으므로 \r를 Replace하여 원하는 값을 뽑고 그 값을 클래스에 담는다. 이 때 클래스는 데이터를 담는 클래스로 다시 배열로 치환된다.

# The Sword

## 데이터 연동

```
public interface ILoader<Key, Value>
{
    Dictionary<Key, Value> MakeDict();
}
```

Loader를 이용하는데 모든 Loader함수는 ILoader Interface를 상속 받는다.

```
[Serializable]
public class PlayerDataLoader : ILoader<int, PlayerData>
{
    public List<PlayerData> creatures = new List<PlayerData>();
    public Dictionary<int, PlayerData> MakeDict()
    {
        Dictionary<int, PlayerData> dict = new Dictionary<int, PlayerData>();
        foreach (PlayerData creature in creatures)
        {
            dict.Add(creature.id, creature);
        }
        return dict;
    }
}
```

클래스 List를 Dictionary 형태로 변환시켜서 반환하는 함수이다. 이 것은 실제 게임에 들어갈 때 데이터를 파싱한다.

```
string jsonStr = JsonConvert.SerializeObject(loader, Formatting.Indented);
File.WriteAllText($"{Application.persistentDataPath}/{filename}Data.json", jsonStr);
AssetDatabase.Refresh();
```

List를 Json으로 Convert한 후 경로를 지정하고 Json으로 저장한다.

# The Sword

## 데이터 연동

```
public class DataManager
{
    public Dictionary<int, Data.PlayerData> PlayerDic { get; private set; } = new Dictionary<int, Data.PlayerData>();
    public Dictionary<int, Data.MonsterData> MonsterDic { get; private set; } = new Dictionary<int, Data.MonsterData>();
    public Dictionary<int, Data.ConsumableItemData> ConsumableItemDic { get; private set; } = new Dictionary<int, Data.ConsumableItemData>();
    public Dictionary<int, Data.MonsterClassData> MonsterClassDic { get; private set; } = new Dictionary<int, Data.MonsterClassData>();
}
```

```
public void Init()
{
    AssetDatabase.Refresh();

    PlayerDic = LoadJson<Data.PlayerDataLoader, int, Data.PlayerData>("PlayerData").MakeDict();
    MonsterDic = LoadJson<Data.MonsterDataLoader, int, Data.MonsterData>("MonsterData").MakeDict();
    ConsumableItemDic = LoadJson<Data.ConsumableItemDataLoader, int, Data.ConsumableItemData>("ConsumableItemData").MakeDict();
    MonsterClassDic = LoadJson<Data.MonsterClassDataLoader, int, Data.MonsterClassData>("MonsterClassData").MakeDict();
    MapDic = LoadJson<Data.MapDataLoader, string, Data.MapData>("MapData").MakeDict();
}
```

```
Loader LoadJson<Loader, Key, Value>(string path) where Loader : ILoader<Key, Value>
{
    TextAsset textAsset = Managers.Resource.Load<TextAsset>($"*{path}*");

    if (path == "MapData")
    {
        return JsonConvert.DeserializeObject<Loader>(textAsset.text, new JsonSerializerSettings
        {
            TypeNameHandling = TypeNameHandling.Auto
        });
    }
    else
    {
        return JsonConvert.DeserializeObject<Loader>(textAsset.text);
    }
}
```

Json파일을 사용할 때는 게임을 시작하면 DataManager가 Init함수를 호출한다.

Init함수에서 LoadJson함수를 사용하는데 Iloader를 상속받은 generic함수로 Json을 다시 string값으로 전환한다.

loader에 있는 MakeDict함수로 Dictionary형태로 다시 전환하고 이것을 멤버변수인 Dictionary에 넣어서 데이터를 관리한다.



# The Sword

## 데이터 관리

데이터는 Singleton패턴으로 관리했다.  
Managers하나에서 모든 걸 편하게  
관리하기 위함.

```
public static void Init()
{
    ... if (s_instance == null)
    ... {
        ... GameObject go = GameObject.Find("@Managers");
        ... if (go == null)
        ... {
            ... go = new GameObject { name = "@Managers" };
            ... go.AddComponent<Managers>();
            ... }
        ...

        ... GameObject cursor = GameObject.Find("@Cursor");
        ... if (cursor == null)
        ... {
            ... cursor = new GameObject { name = "@Cursor" };
            ... cursor.AddComponent<SpriteRenderer>();
            ... cursor.AddComponent<Animator>();
            ... cursor.AddComponent<CursorManager>();
            ... }

        ... DontDestroyOnLoad(go);
        ... DontDestroyOnLoad(cursor);
        ... s_instance = go.GetComponent<Managers>();
        ... //s_instance._sound.Init();
        ... //s_instance._time = go.AddComponent<TimeManager>();
        ... }
    }
}
```

Manager클래스가 모든 Manager류  
클래스들을 관리한다.

하나의 Manager만 있어야하므로  
static이고 Manager멤버로 각  
Manager들이 들어간다.

```
static Managers s_instance; // 유일성이 보장된다
static Managers Instance { get { Init(); return s_instance; } } // 유일한 매니저를 갖고온다

#region Contents
GameManager _game = new GameManager();
DirectingManager _directing = new DirectingManager();

public static GameManager Game { get { return Instance?._game; } }
public static DirectingManager Directing { get { return Instance?._directing; } }
```

# The Sword

## 다국 언어

앞에서 제작한 데이터  
관리를 응용하여 다국  
언어도 지원할 수 있다.

No.	Script_Kr	Script_En	Script_Jp	Script_Cn
0	PRESS ANY KEY			
1	Game Start			
2	Load Game			
3	Setting			
4	Exit			
5	New Game			
6	모험가			
7	{size a=-20 d=0.1}<color=#ff0000><size=			
8	[ 마검 계약 ]\n계약자는 이 순간부로 마검			

```
public static string GetString(int id)
{
    int scriptType = (int)Managers.Game.ScriptType;

    string ret = "";
    switch (scriptType)
    {
        case 0:
            ret = Managers.Data.ScriptDic[id].ScriptKr;
            break;
        case 1:
            ret = Managers.Data.ScriptDic[id].ScriptKr;
            break;
        case 2:
            ret = Managers.Data.ScriptDic[id].ScriptEn;
            break;
        case 3:
            ret = Managers.Data.ScriptDic[id].ScriptJp;
            break;
        case 4:
            ret = Managers.Data.ScriptDic[id].ScriptCn;
            break;
        default:
            ret = Managers.Data.ScriptDic[id].ScriptKr;
            break;
    }

    ret = ret.Replace("\n", "\n");
    ret = ret.Replace("<", "<");

    return ret;
}
```

이렇게 Excel을 잡고 데이터를 파싱한 후, Json으로 저장되고, 게임  
시작시 Dictionary로 변환된 뒤,  
사용은 GetString함수를 만들어서 언어 type에 따라서 값을  
불러오는 방식이다.

# MessMath

# 시연 영상



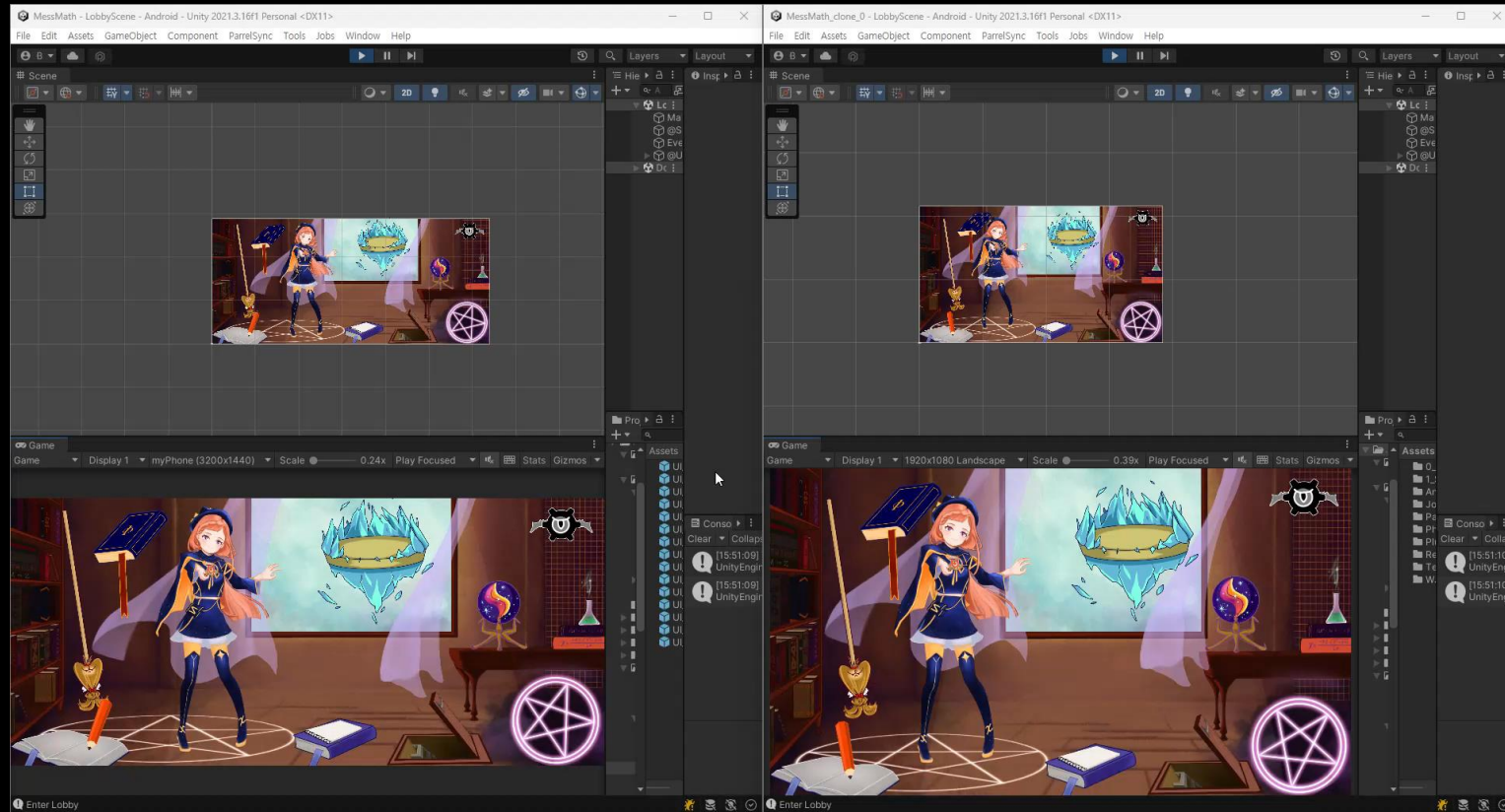
## 오합지졸

<https://youtu.be/UUPn5YygFgE?si=fOjGS7frNqcJWdBN>

# MessMath

목적:	게임 개발 공모전을 위한 수학 게임
개발 기간 :	2023.04~ 2023.08
담당 기능을 개발하기 위한 사용 도구 :	Unity, C#, Git, GithubDesktop, 포톤, MathpidAPI, Firebase
개발 인원 :	5인(프로그래머4, 아트1)
담당 업무:	애니메이션, UI, 포톤 서버, 멀티, MathpidAPI연동, 데이터 연동, 스토리 모드, 연습모드

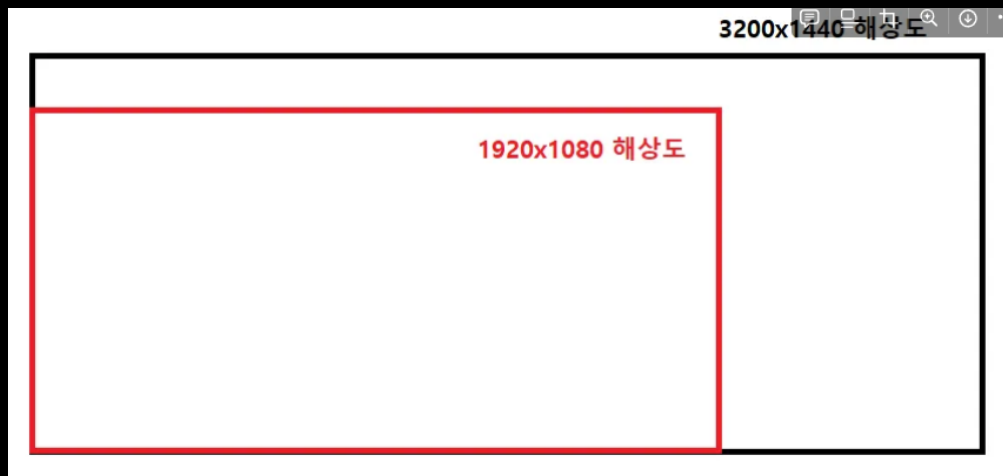
# 해상도 트러블 슈팅



서로 다른 해상도를 가진 두 기기로 PVP를 진행하면  
플레이가 이상해진다.

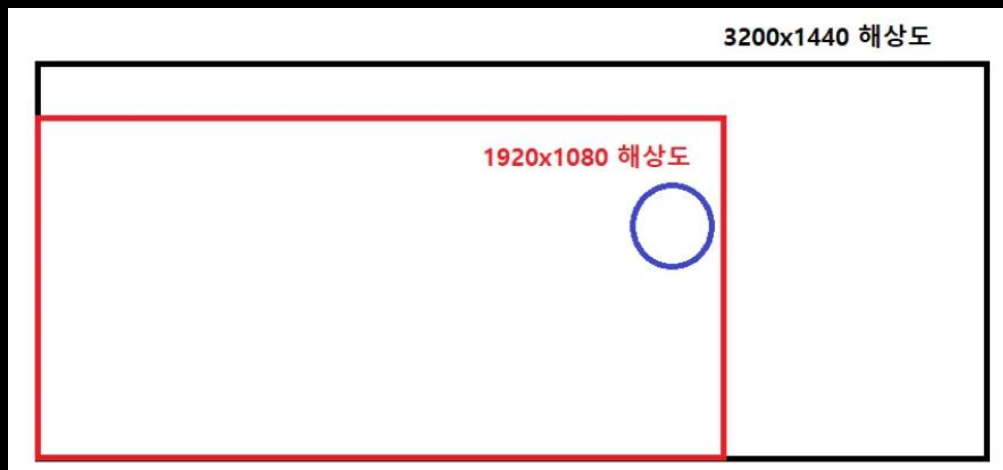
해상도가 다른 두 환경에서는 물체의 위치가 이상하게  
잡힌다.

# 문제 이해



이렇게 되는 경우 두 화면에서 동일한 좌표를 가지지만 화면상의 위치는 달라지게 된다.

1920x1080 해상도의 기기에서는 오른쪽 가장자리에 매우 가까이 있다고 보일 것이고, 3200x1440 해상도에서는 거의 화면의 정중앙에서 살짝 오른쪽위로 이동한 정도의 위치에 있다고 보일 것이다.



그리고 플레이어의 이동속도도 문제가 있었다. 플레이어는 동일한 단위(unit)으로 이동하므로 작은 해상도를 가진 플레이어가 상대적으로 더 높은 이동속도를 가지게 되었다.

# 문제 해결

## 1) 해상도에 상대적인 위치 잡기

앞선 상황에서 문제를 해결하기 위해 오브젝트 위치를 왜곡하여 해결하려했다.



MasterClient에서 생긴 이 오브젝트의 정보를, 다른 Client에서 수신할 때는 자기 해상도에 맞게끔 조정해서 가져오는것이다.



그러면 동일한 위치에 있는것처럼 보일 것이다.



그래서 우리는, MasterClient에서 오브젝트의 전체 해상도 대비 위치값, 그러니까 x좌표를 그냥 보내는것이 아니라, (x좌표 / 가로 해상도)와 (y좌표 / 세로 해상도)를 보내기로 결정했다.



이 비율을 전송하면, 수신한 Client는 자신의 해상도에 이 비율을 곱해서 이 오브젝트가 있어야 할 위치를 도출해낼 수 있을것이다.

# 문제 해결

## 1) 해상도에 상대적인 위치 잡기

```
Vector3 transPosIntoRatio()
{
    float actualH = Screen.width / 3200f * 1440f;
    float xRatio = RT.position.x / Screen.width;
    float yRatio = (RT.position.y - ((Screen.height - actualH) / 2f)) / actualH;

    return new Vector3(xRatio, yRatio, 0f);
}

void transRatioIntoPos(Vector3 vector3)
{
    float xRatio = vector3.x;
    float yRatio = vector3.y;
    float actualH = Screen.width / 3200f * 1440f;

    float xPos = Screen.width * xRatio;
    float yPos = actualH * yRatio + ((Screen.height - actualH) / 2f);

    curPos = new Vector3(xPos, yPos, 0f);
}
```

연산 과정을 보면, 좀 특이한 과정이 하나 있는데, 이는 레터박스 때문이다.

우리는 여태까지 가변 해상도를 고려하지않고  
3200x1440 해상도를 기준으로 작업했고,  
1920x1080 해상도처럼 가로로 눌렀을 때 세로로 더 높은 해상도는  
위 아래로 레터박스가 생긴다.

그래서 왼쪽과 같이 구현했다.



# 문제 해결

## 2) 캐릭터 속도 동기화

```
public class PlayerControllerOnlyinPvp : MonoBehaviourPun, IPunObservable
{
    float referenceWidth = 3200f; // 기준 해상도의 너비
    float referenceHeight = 1440f; // 기준 해상도의 높이
    float currentWidth = Screen.width; // 현재 화면의 너비
    float currentHeight = Screen.height; // 현재 화면의 높이

    float widthRatio;
    float heightRatio;
    float adjustedSpeed;

    void Awake()
    {
        widthRatio = currentWidth / referenceWidth;
        heightRatio = currentHeight / referenceHeight;

        adjustedSpeed = _speed * Mathf.Min(widthRatio, heightRatio);
    }
}
```

캐릭터 속도 동기화 문제도 해상도와 비슷하게 해결할 수 있을거같았다.

Awake 단계에서 해상도에 맞는 \_speed의 adjustedSpeed를 산출하고 이를 \_speed에 적용했다.

# 던전포차

# 시연 영상

[https://youtu.be/ieY6bY\\_Rgyg?si=J2sLyG0pGJHbj8KL](https://youtu.be/ieY6bY_Rgyg?si=J2sLyG0pGJHbj8KL)

# 던전포차

목적: 인디크래프트 공모전 게임  
개발 기간 : 2024.02~ 2024.04  
사용 도구 : Unity, C#, Git, GithubDesktop  
개발 인원 : 7인(프로그래머3, 아트3, 기획자 1)  
담당 업무: 애니메이션, UI, 전투 구현, 다국 언어, 데이터 연동

# 전투 구현

```
IEnumerator CoSpawnMonster()
{
    while (true)
    {
        WaitForSeconds spawnDelay = new WaitForSeconds(0.2f);
        yield return spawnDelay;

        if (Managers.Game.MonsterCount < 100)
            CreateMonsterIcon();
    }
}
```

던전에 입장하면  
CoSpawnMonster가 실행된다.

0.2초마다 총 몬스터가 100마리  
미만이라면 계속  
CreateMonsterIcon을 실행한다.

```
public void CreateMonsterIcon()
{
    Transform parent = GameObject.Find("MiniDungeon").gameObject.transform;

    GameObject startPoint = GameObject.Find("MonsterStartPoint");
    MonsterController monster = Managers.Resource.Instantiate("Creature/Monster", parent).GetComponent<MonsterController>();
    monster.GetComponent<SpriteRenderer>().sprite = Managers.Resource.Load<Sprite>($"Sprites/MiniDungeon/Monsters/Mon_{Managers.Game.Dungeon}");

    Managers.Game.Monsters.Add(monster.GetComponent<MonsterController>());
    Managers.Game.MonsterCount++;
    monster.Attack = Managers.Game.DungeonStage.MonAtk + Managers.Game.DungeonStage.NextATK * 1;
    monster.Speed = Managers.Game.DungeonStage.MonSPD + Managers.Game.DungeonStage.NextSPD * 1;
    monster.Hp = Managers.Game.DungeonStage.MonHP + Managers.Game.DungeonStage.NextHP * 1;
    monster.name = $"Monster";
    monster.transform.position = new Vector3(startPoint.transform.position.x, startPoint.transform.position.y + Random.value, 1f);
    monster._originalYPos = monster.transform.localPosition.y;
    monster._originalXPos = monster.transform.localPosition.x;
    monster.StartCoMoveToHunter();
}
```

CreateMonsterIcon은 스테이지  
정보에 맞게 몬스터 능력치가  
설정되고 플레이어 스폰 쪽으로  
계속해서 이동한다.

# 전투 구현

```
IEnumerator CoAttack(HunterController hunter)
{
    WaitForSeconds delay = new WaitForSeconds(0.5f);
    if(!_isAttack == false)
    {
        while (true)
        {
            if (hunter == null)
                break;
            if (hunter.Hp <= 0)
                break;

            _isAttack = true;
            gameObject.transform.position += new Vector3(-0.05f, 0, 0);
            hunter.Hp -= this.Attack;
            yield return delay;
            gameObject.transform.position += new Vector3(+0.05f, 0, 0);
            yield return delay;
        }
    }

    _isAttack = false;
    StartCoMoveToHunter();
}
```

플레이어 아이콘을 만나면 그  
상대로 공격을 하고 피가 없을 때  
까지 공격한다.

# 전투 구현

```
public HunterController CreatePlayerIcon()
{
    Transform parent = GameObject.Find("MiniDungeon").gameObject.transform;
    GameObject startPoint = GameObject.Find("StartPoint");
    GameObject hunter = Managers.Resource.Instantiate("Creature/Hunter", parent);
    hunter.transform.position = new Vector3(startPoint.transform.position.x, startPoint.transform.position.y + Random.value, 1f);
    hunter.GetComponent<HunterController>().originalYPos = hunter.transform.localPosition.y;
    hunter.GetComponent<HunterController>().originalXPos = hunter.transform.localPosition.x;
    hunter.GetComponent<SpriteRenderer>().sprite = Managers.Resource.Load<Sprite>($"Sprites/MiniDungeon/Customers/Guest_{string.Format("{0:D3}", UnityEngine.Random.Range(0, 25))}");
    Managers.Game.Hunters.Add(hunter.GetComponent<HunterController>());
    Managers.Game.HunterCount++;

    //hunter.GetComponent<HunterController>().OnMoveToNextMonster();
    hunter.GetComponent<HunterController>().StartCoMoveToMonster();

    return hunter.GetComponent<HunterController>();
}
```

CreatePlayerIcon은 플레이어 Icon을 생성하고 몬스터 스폰쪽으로 이동한다.

```
IEnumerator CoAttack(MonsterController monster)
{
    while (true)
    {
        WaitForSeconds attackSpeed = new WaitForSeconds(Managers.Data.DungeonInfoDic[Managers.Game.curLevelIndex.ToString()].SpeedByLevel); //!!!

        if (monster == null)
            break;
        if (this.Hp <= 0)
            break;
        if (monster.Hp <= 0)
            break;

        _isAttack = true;

        gameObject.transform.position += new Vector3(+0.05f, 0, 0);
        monster.Hp -= this.Attack;
        GameObject dungeonHitStar = Managers.Resource.Instantiate("MiniDungeon/DungeonHitStar", transform);
        dungeonHitStar.transform.position = (gameObject.transform.position + monster.transform.position) / 2;
        float playTime = dungeonHitStar.GetComponent<Animator>().GetCurrentAnimatorStateInfo(0).length;
        Destroy(dungeonHitStar, playTime);

        if ((monster.Hp) > 0)
        {
            //OnPlayRandomAttackSound();
        }

        yield return attackSpeed;
        gameObject.transform.position += new Vector3(-0.05f, 0, 0);
        yield return attackSpeed;
    }

    _isAttack = false;
    StartCoMoveToMonster();
}
```

```
public void OnAttack(MonsterController monster)
{
    if (monster == null)
        return;
    _monster = monster;
    StartCoroutine(CoAttack(monster));
    monster.OnAttack(this);
}
```

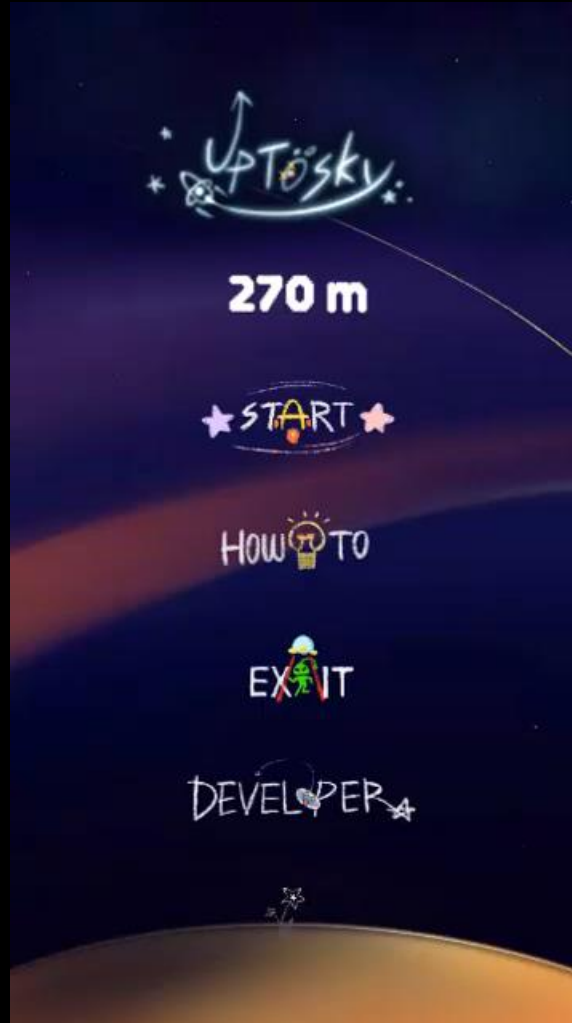
플레이어도 몬스터를 만나면 공격을 시작한다.

# 전투 영상



# UpToSky

## 시연 영상



<https://youtu.be/44kZ5xZ0Jso?si=ShWp25twiPBS69ae>



목적: 구글플레이 출시 게임  
개발 기간 : 2023.02~ 2024.04  
사용 도구 : Unity, C#, Git, GithubDesktop  
개발 인원 : 3인(프로그래머3)  
담당 업무: 애니메이션, UI, 게임 로직, Google  
Ads

# Google Ads

광고를 붙여보았다.

```
--private InterstitialAd interstitialAd;
--public InterstitialAd InterstitialAd { get { return interstitialAd; } }

--public void Init()
--{
--    ---// Initialize the Google Mobile Ads SDK.
--    ---MobileAds.Initialize((InitializationStatus initStatus) =>
--    ---{
--    ---    ---// This callback is called once the MobileAds SDK is initialized.
--    ---    ---LoadInterstitialAd();
--    ---    ---
--    ---});
--}
--// 전면 광고 로드
--// These ad units are configured to always serve test ads.
```

광고 내용을 로드하는 함수.

```
--// 전면 광고 로드
--/// <summary>
--/// Loads the interstitial ad.
--/// </summary>
--public void LoadInterstitialAd()
--{
--    ---// Clean up the old ad before loading a new one.
--    ---if (interstitialAd != null)
--    ---{
--    ---    ---interstitialAd.Destroy();
--    ---    ---interstitialAd = null;
--    ---}

--    --- Debug.Log("Loading the interstitial ad.");

--    ---// create our request used to load the ad.
--    ---var adRequest = new AdRequest.Builder().Build();

--    ---// send the request to load the ad.
--    ---InterstitialAd.Load(_adUnitId, adRequest,
--    ---    --- (InterstitialAd ad, LoadAdError error) =>
--    ---    --- {
--    ---    ---    ---// if error is not null, the load request failed.
--    ---    ---    ---if (error != null || ad == null)
--    ---    ---    ---{
--    ---    ---    ---    ---Debug.LogError("interstitial ad failed to load an ad " +
--    ---    ---    ---    ---    ---"with error : " + error);
--    ---    ---    ---    ---return;
--    ---    ---    ---}

--    ---    ---    --- Debug.Log("Interstitial ad loaded with response :-"
--    ---    ---    ---    ---+ ad.GetResponseInfo());

--    ---    ---    ---interstitialAd = ad;
--    ---    ---    ---});
--}

--}
```

# Google Ads

실제로 광고를 보여줄 때  
사용되는 함수

```
// 전면 광고 표시
/// <summary>
/// Shows the interstitial ad.
/// </summary>

public void ShowAd()
{
    if (interstitialAd != null && interstitialAd.CanShowAd())
    {
        Debug.Log("Showing interstitial ad.");
        interstitialAd.Show();
        RegisterEventHandlers(interstitialAd);
        RegisterReloadHandler(interstitialAd);
    }
    else
    {
        Debug.LogError("Interstitial ad is not ready yet.");
    }
}
```

```
private void RegisterEventHandlers(InterstitialAd ad)
{
    /// <summary>
    /// Raised when the ad is estimated to have earned money.
    /// ad.OnAdPaid += (AdValue adValue) =>
    /// {
    ///     Debug.Log(String.Format("Interstitial ad paid {0} {1}.",
    ///         adValue.Value,
    ///         adValue.CurrencyCode));
    /// };
    /// </summary>
    ad.OnAdPaid += (AdValue adValue) =>
    {
        Debug.Log(String.Format("Interstitial ad paid {0} {1}.",
            adValue.Value,
            adValue.CurrencyCode));
    };

    /// <summary>
    /// Raised when an impression is recorded for an ad.
    /// ad.OnAdImpressionRecorded += () =>
    /// {
    ///     Debug.Log("Interstitial ad recorded an impression.");
    /// };
    /// </summary>
    ad.OnAdImpressionRecorded += () =>
    {
        Debug.Log("Interstitial ad recorded an impression.");
    };

    /// <summary>
    /// Raised when a click is recorded for an ad.
    /// ad.OnAdClicked += () =>
    /// {
    ///     Debug.Log("Interstitial ad was clicked.");
    /// };
    /// </summary>
    ad.OnAdClicked += () =>
    {
        Debug.Log("Interstitial ad was clicked.");
    };

    /// <summary>
    /// Raised when an ad opened full screen content.
    /// ad.OnAdFullScreenContentOpened += () =>
    /// {
    ///     Debug.Log("Interstitial ad full screen content opened.");
    /// };
    /// </summary>
    ad.OnAdFullScreenContentOpened += () =>
    {
        Debug.Log("Interstitial ad full screen content opened.");
    };

    /// <summary>
    /// Raised when the ad closed full screen content.
    /// ad.OnAdFullScreenContentClosed += () =>
    /// {
    ///     Debug.Log("Interstitial ad full screen content closed.");
    /// };
    /// </summary>
    ad.OnAdFullScreenContentClosed += () =>
    {
        Debug.Log("Interstitial ad full screen content closed.");
    };

    /// <summary>
    /// Raised when the ad failed to open full screen content.
    /// ad.OnAdFullScreenContentFailed += (AdError error) =>
    /// {
    ///     Debug.LogError("Interstitial ad failed to open full screen content." +
    ///         "with error: " + error);
    /// };
    /// </summary>
    ad.OnAdFullScreenContentFailed += (AdError error) =>
    {
        Debug.LogError("Interstitial ad failed to open full screen content." +
            "with error: " + error);
    };

    /// <summary>
    /// Raised when the ad is estimated to have earned money.
    /// ad.OnAdPaid += (AdValue adValue) =>
    /// {
    ///     Debug.Log(String.Format("Interstitial ad paid {0} {1}.",
    ///         adValue.Value,
    ///         adValue.CurrencyCode));
    /// };
    /// </summary>
    ad.OnAdPaid += (AdValue adValue) =>
    {
        Debug.Log(String.Format("Interstitial ad paid {0} {1}.",
            adValue.Value,
            adValue.CurrencyCode));
    };
}
```

광고를 끄는 등 이벤트가  
발생될 때 사용되는 함수.  
우리의 경우 광고를 끄면  
GameScene으로 이동.  
광고의 에러가 생겨도  
GameScene으로 이동한다.

# Google Ads

다음 전면 광고를 미리 로드하는 함수.

```
// 다음 전면 광고 미리 로드
private void RegisterReloadHandler(InterstitialAd ad)
{
    // Raised when the ad closed full screen content.
    ad.OnAdFullIScreenContentClosed += () =>
    {
        // Debug.Log("Interstitial Ad full screen content closed.");

        // Reload the ad so that we can show another as soon as possible.
        LoadInterstitialAd();
    };

    // Raised when the ad failed to open full screen content.
    ad.OnAdFullIScreenContentFailed += (AdError error) =>
    {
        // Debug.LogError("Interstitial ad failed to open full screen content." +
        // "with error: " + error);

        // Reload the ad so that we can show another as soon as possible.
        LoadInterstitialAd();
    };
}
```

광고를 한번 보여주면 Reload함수가  
실행되어 광고를 저장하고 있다.  
다시 광고가 나오면 저장된 광고가  
나오고 Reload를 하면서 반복.

# Google Ads

하단에 상시로  
나오는 광고

```
void ToScoreGameScene( )
{
    // 게임 시작
    // 스토리 모드에 맞게 게임씬이 바뀌어야 함
    Managers.Sound.Clear( );
    Managers.Sound.Play( "Sound_CloseUI" );
    Managers.UI.ClosePopupUI( this );
    //
    Managers.Game.AdCount++;

    if (Managers.Game.AdCount % 3 != 0)
    {
        // UnityEngine.SceneManagement.SceneManager.LoadScene( "GameScene" );
        return;
    }

    #region

    // 전면 광고 추가
    Managers.Sound.Clear( );

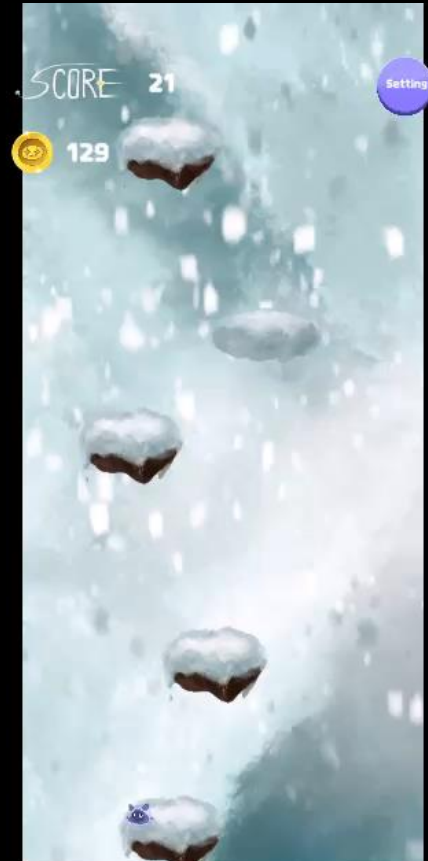
    Managers.Ads.ShowAd( );

    #endregion
}
```



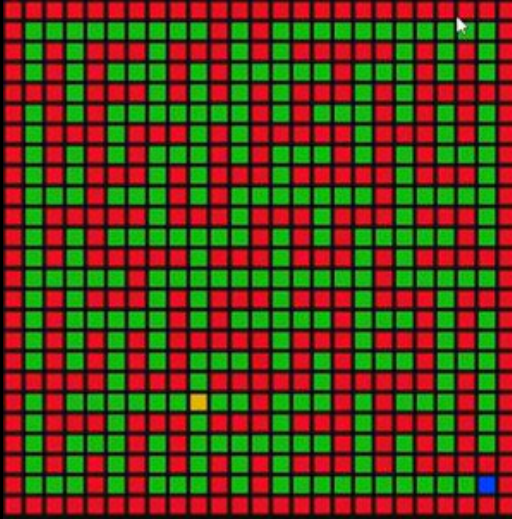
This is a Test  
Adaptive Banner

3번에 1번나오는  
전면 광고

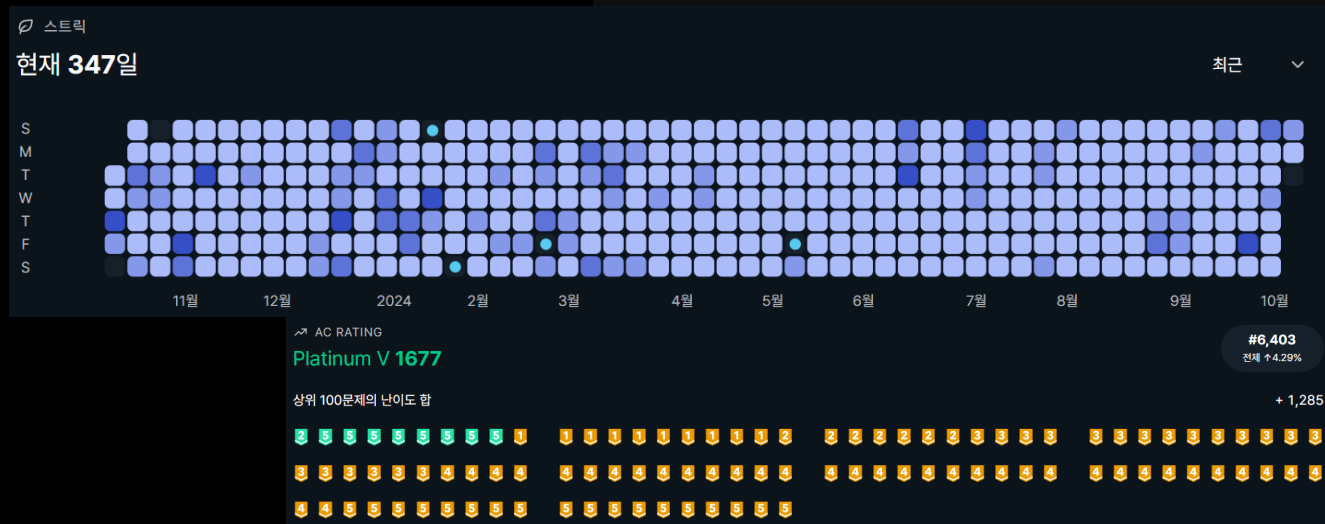


This is a Test  
Adaptive Banner

# 기타이력 알고리즘



## 길찾기 알고리즘 공부용 미로 찾기



# 꾸준한 알고리즘 공부

# 기타이력

## DirectX

렌더링 파이프라인에 따라서 코드를 작성하고 적용해보았습니다.  
DirectX11 책에 나오는 내용을 가져오 되, 제 방식대로 코드를 이해하고 재구성했습니다.

