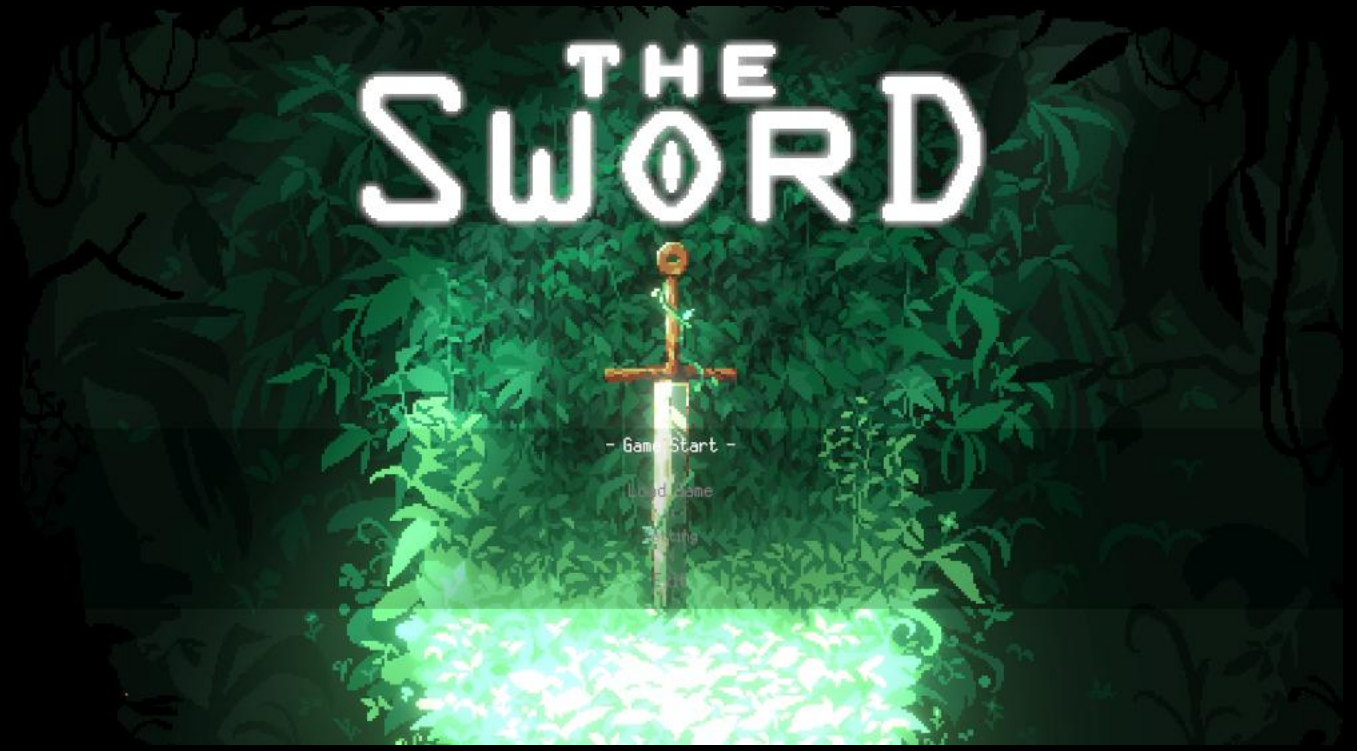
# The Sword




https://youtu.be/Swc5I2-JCMQ

| | |
|---|---|
| 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, Exel 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,
        HPHar,
        HPHarGauge,
        AttackDelayGauge,
        DefenceDelayGauge,
```

1. UI Prefab 속 UI 이름을 Enum에 적기

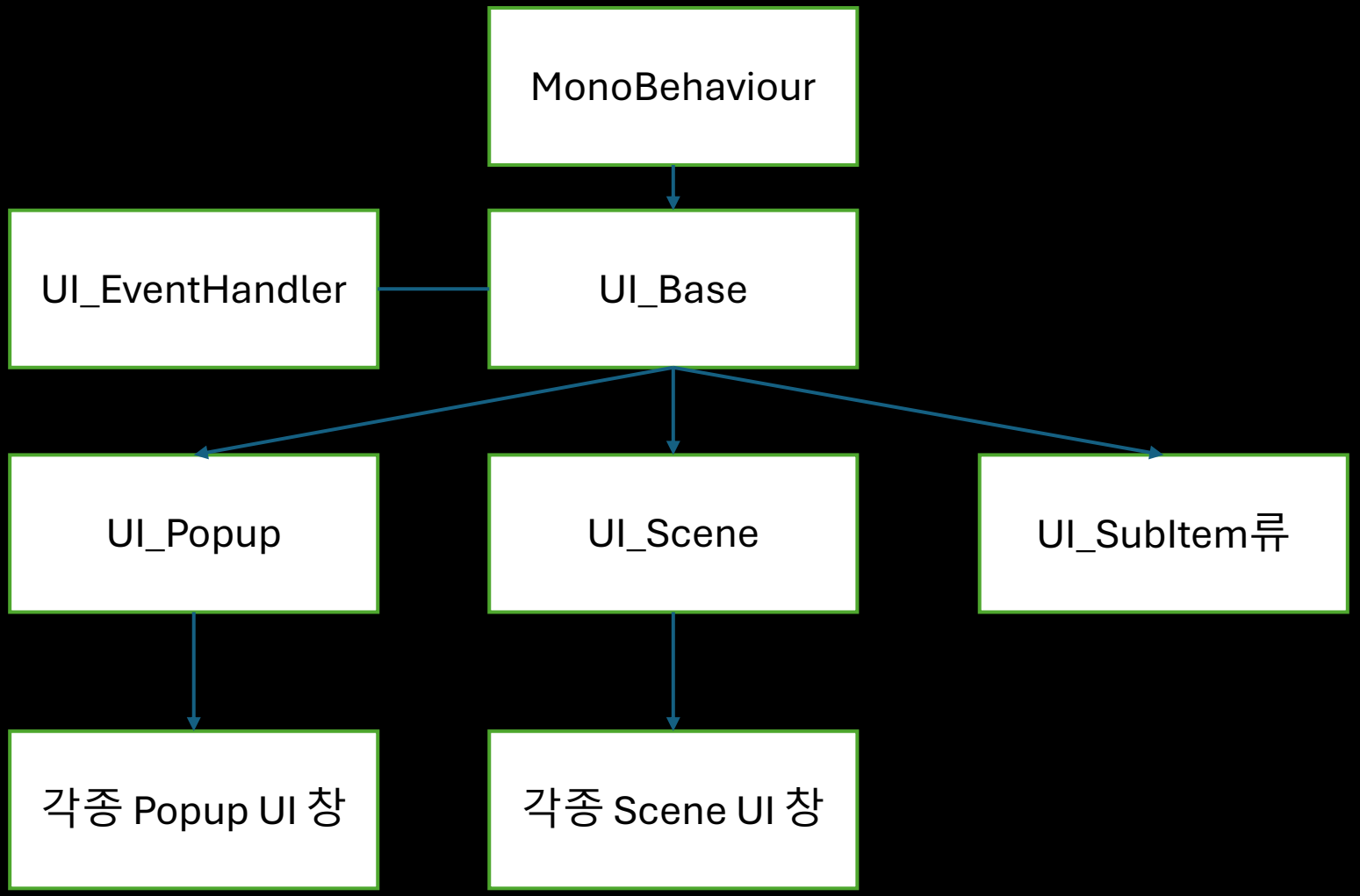
```
public override bool Init()
{
    if (base.Init() == false)
        return false;

    #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


MonoBehaviour
UI_EventHandler
UI_Base
UI_Popup
UI_Scene
UI_SubItem류
각종 Popup UI 창
각종 Scene UI 창

# 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 abstrac로 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.

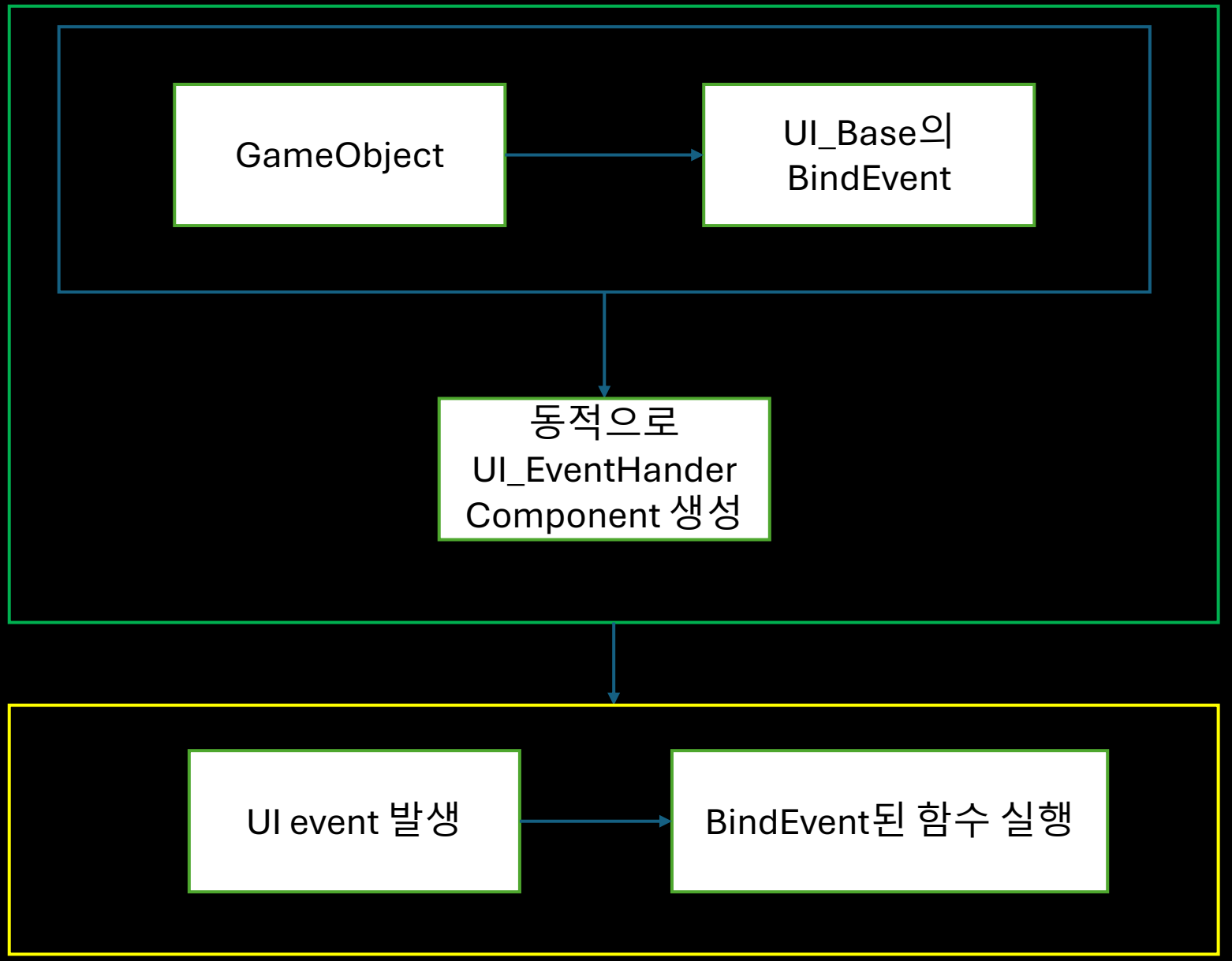
```
protected GameObject GetObject(int idx) { return Get<GameObject>(idx); }
protected TMP_Text GetText(int idx) { return Get<TMP_Text>(idx); }
protected Button GetButton(int idx) { return Get<Button>(idx); }
protected Image GetImage(int idx) { return Get<Image>(idx); }
protected Toggle GetToggle(int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

앞선 UI_EventHander가 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;
```

# 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.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:
```

앞선 UI_EventHander가 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);
}
```

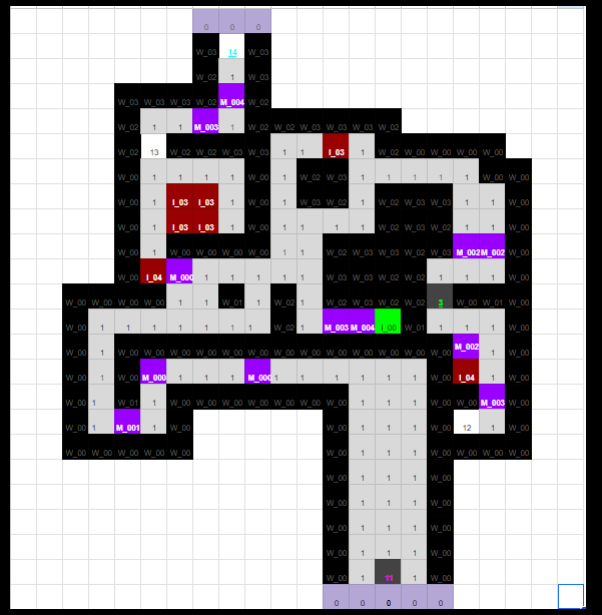
Extionsion기능을 이용하여 GameObject에서 .을붙여서
BindEvent함수를 사용할 수 있게 만든다.

사용 예시

```
GetImage((int)Images.MainUIInventoryAImage).gameObject.BindEvent(OnClickMainUIInventoryAImage);
```

# 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, 바닥이 뚫린 공간 |
|---|---|---|---|---|---|---|
| 3D | 바닥 | 1 | | 1 | 바닥 | 움직일 수 있는 바닥 타일 |
| 3D | Tilemap_W00 | 00 | W_ | W_00 | 벽 00 | 수풀벽 |
| 3D | Tilemap_W01 | 01 | W_ | W_01 | 벽 01 | 나무 그루터기 |
| 3D | Tilemap_W02 | 02 | W_ | W_02 | 벽 02 | 부셔진 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 | 초록 문 / 가로 / — | 초록 열쇠가 있어야 열리는 문 |
| 3D | | 4 | | 4 | 초록 문 / 세로 / I | |
| 3D | Tilemap_Door_Y | 5 | | 5 | 노랑 문 / 가로 / — | 노랑 열쇠가 있어야 열리는 문 |
| 3D | | 6 | | 6 | 노랑 문 / 세로 / I | |
| 3D | Tilemap_Door_R | 7 | | 7 | 빨강 문 / 가로 / — | 붉은 열쇠가 있어야 열리는 문 |
| 3D | | 8 | | 8 | 빨강 문 / 세로 / I | |
| 2D | Tilemap_StairsUp0 | 9 | | 9 | 계단 상 / 챕터0 | 다음 층으로 올라가는 계단 ▲ |
| 2D | Tilemap_StairsDown0 | 10 | | 10 | 계단 하 / 챕터0 | 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, 맵에 모든 철창이 해제. |
| 3D | Tilemap_SteelDoor | 13 | | 13 | 기믹 / 철창 문 | 평소엔 닫혀있는 철창 문, 레버로 열 수 있다 |
| 2D | Tilemap_Road | 14 | | 14 | 일반 길 / 상 | 계단과 동일한, 길 끝에 닿으면 다음 맵에 이동▲ |
| 2D | Tilemap_Road | 15 | | 15 | 일반 길 / 하 | 계단과 동일한, 길 끝에 닿으면 다음 맵에 이동▼ |
| 2D | Tilemap_BossRoad | 16 | | 16 | 보스 길 | 계단과 동일한, 길 끝에 닿으면 보스맵에 이동 |
| 2D | Tilemap_BossRoad | 17 | | 17 | 튜토 마검 | 튜토리얼의 꽂혀있는 마검 |
| 2D | Tilemap_ChapterClear | 18 | | 18 | 챕터 클리어 문 | 보스 처치 시, 본인이 통과 다음 맵으로 이동 가능 |

# Map 자동생성Tool

Exel 데이터를 바탕으로 map을 생성할 수 있도록 Exel을 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 프리펩으로 만든 뒤, 맵을 저장한다.

# Map 생성 로직


Exel Map Data
만들기
Exel을 읽고, Editor에서
Hierarchy창에 오브젝트들 생성
오브젝트들을
Prefab화
Map 저장
게임 시작시
Map을 불러오기

# Map 생성 고찰 및 개선

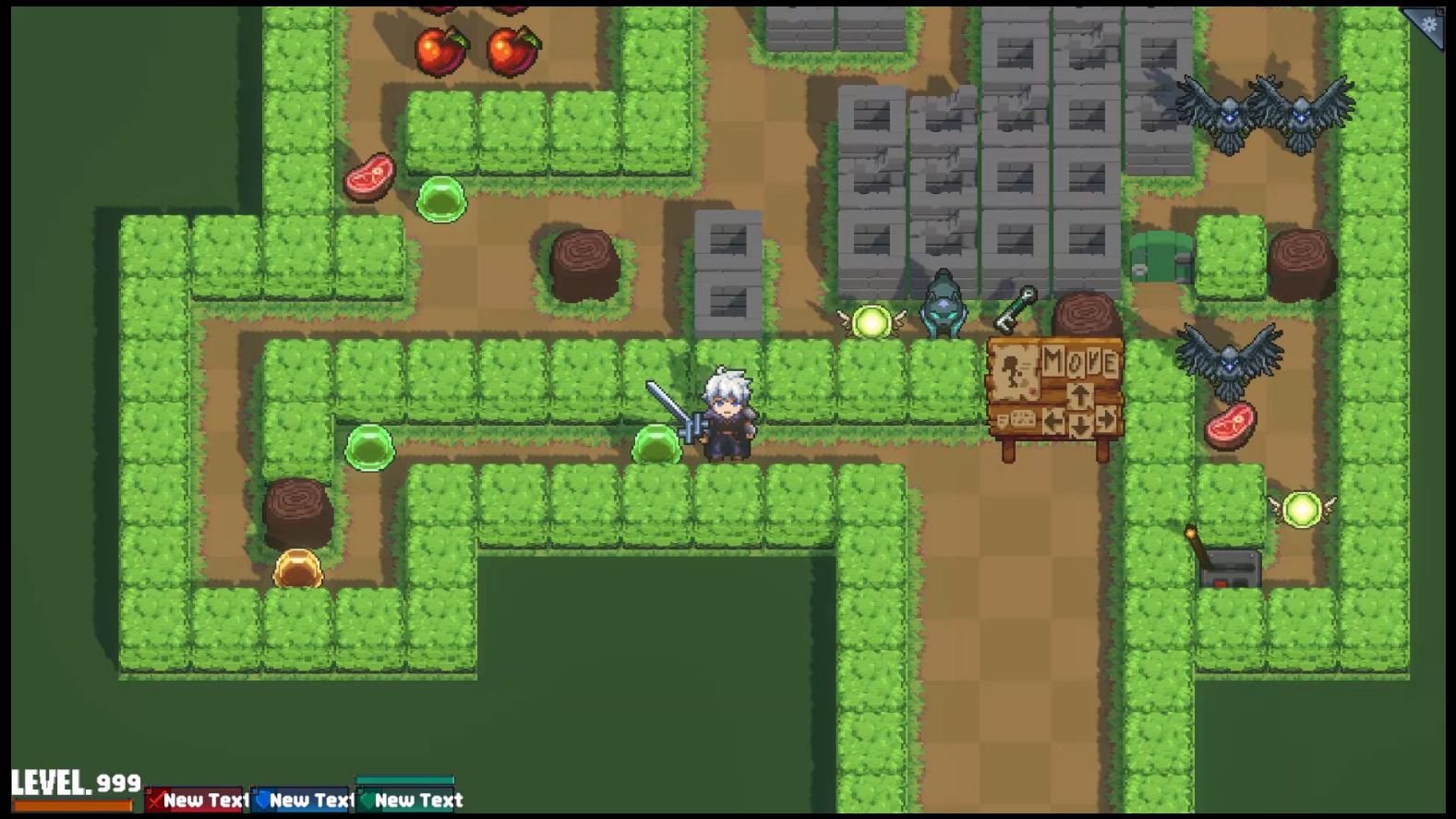
| 문제점 | 개선 |
| --- | --- |
| 맵이 단층으로만 만들수있다. | 2층 이상은 층마다 데이터를 따로 만들어서 병합. |
| 타일이 많아질수록 규칙이 늘어난다. | 챕터, 스테이지에 따라 컨벤션을 정해서 간략한 규칙으로 쉽게 생성되게 변경했다. |
| 몬스터 데이터와 연동을 어떻게 할것인지 | 예를들어 M_12 이런 형식으로 MonsterData의 index를 확인하여 그것에맞게 몬스터를 생성한다. 몬스터 생성시 기본 monster prefab으로 생성하고 id값만 달라지면 다른 몬스터가 생성되게 만들었다. |

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_1챕터_#0 |
| 6 | 01_001 | Normal | 01_002 | 01_000 | | 7 | 7 | 100 | 01_001 | BGM_100 | | 타워_1챕터_#1 |
| 7 | 01_002 | Normal | 01_003 | 01_001 | | 8 | 8 | 100 | 01_002 | BGM_100 | | 타워_1챕터_#2 |
| 8 | 01_003 | Normal | 01_004 | 01_002 | | 9 | 9 | 100 | 01_003 | BGM_100 | | 타워_1챕터_#3 |
| 9 | 01_004 | Normal | 01_005 | 01_003 | | 10 | 10 | 100 | 01_004 | BGM_100 | | 타워_1챕터_#4 |
| 10 | 01_005 | Normal | 01_006 | 01_004 | | 11 | 11 | 200 | 01_005 | BGM_100 | | 타워_1챕터_#5 |
| 11 | 01_006 | Normal | 01_007 | 01_005 | | 12 | 12 | 300 | 01_006 | BGM_100 | | 타워_1챕터_#6 |
| 12 | 01_007 | Normal | 01_008 | 01_006 | | 13 | 13 | 400 | 01_007 | BGM_100 | | 타워_1챕터_#7 |
| 13 | 01_008 | Normal | 01_009 | 01_007 | | 14 | 14 | 500 | 01_008 | BGM_100 | | 타워_1챕터_#8 |
| 14 | 01_009 | Special | 01_010 | - | | 15 | 15 | 600 | 01_009 | BGM_101 | | 타워_1챕터_특수 |
| 15 | 01_010 | Boss | - | - | | 16 | 16 | 700 | 01_010 | BGM_102 | | 타워_1챕터_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; }
```

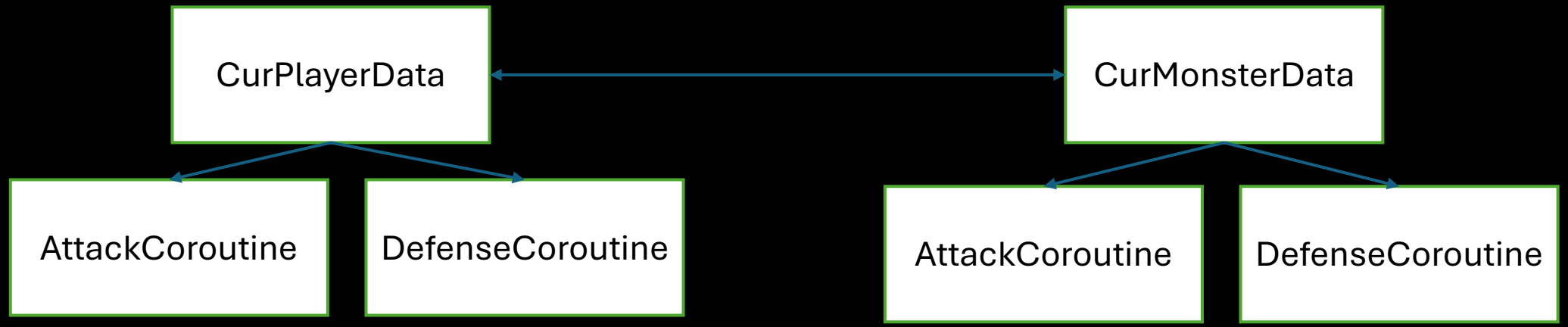
UI_PlayerCard

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 간혈적으로
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 |

Exel의 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; 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.

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