PORTFOLIO

포트폴리오

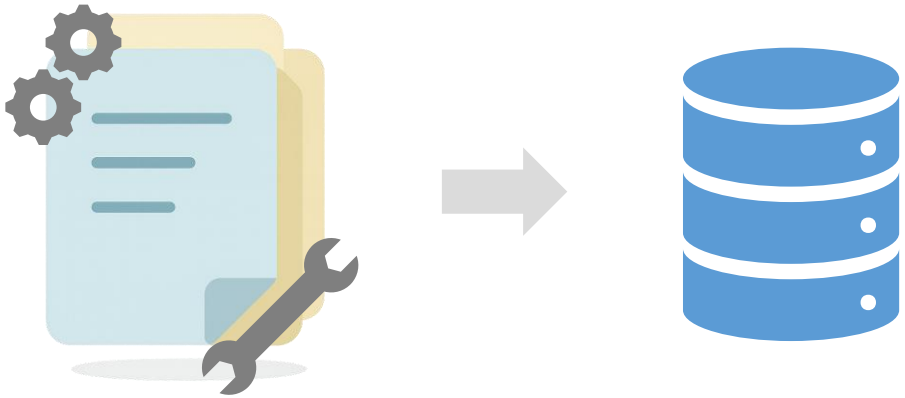
주 용 우
010-8332-3415
parsnip618@outlook.com
https://github.com/JuYongwoo

CONTENTS

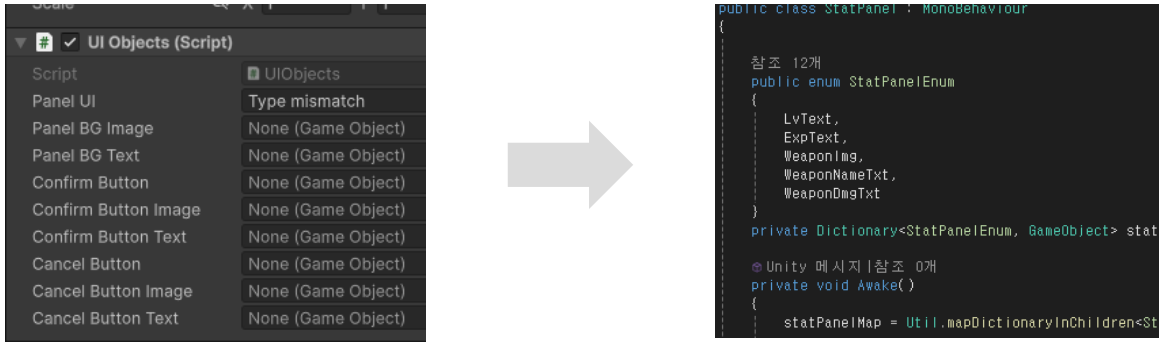
INTRODUCTION

Ju Yongwoo

Game Developer | Unity Specialist | AI & Systems Enthusiast
협업과 유지보수를 무엇보다 중요하게 생각하는 게임 개발자 주용우입니다.
유지보수의 용이성을 통해 **빠른 개발과 함께 오랜 기간 서비스 할 수 있는 구조**를
설계하는 것을 철학으로 삼고 있습니다.



리소스를 수정할 때마다 코드 수정 X
Addressable과 SO를 사용한 유동적인 관리



인스펙터 수동 드래그 앤 드롭 형태 지양
코드 중심의 자동화 작업, 유연한 코드 작성

INFORMATION

기본 정보 증빙서류

EDUCATION

- 2022 청주대학교 컴퓨터정보공학과 전공 (3.89/4.5)
- 2025 명지대학교 컴퓨터공학과 석사 (4.38/4.5)

CAREER

- 2021.9~2021.12 (주)퍼니뎁 모바일 게임 개발 인턴
- 2021~2022 (주)퍼니뎁 모바일 게임 개발 정규직

AWARDS

- 2017 교내 학습동아리 우수상
- 2017 교내 학습동아리 최우수상
- 2020 충북게임아카데미 게임 그래픽 우수상
- 2021 충북게임아카데미 게임 프로그래밍 최우수상
- 2022 교내 우수졸업(CJU 우수인재 인증)

EXPERIENCE

- 2021 충북게임아카데미 게임 프로그래밍 교육 수료
- 2021~2022 (주)퍼니뎁 "배틀웁스!" 프로젝트 참여
- 2023~2025 "1인칭 슈팅 게임 속 AI 분류 및 개선" 논문 작성 및 발표

LICENSE

- 정보처리기사 (2021.6 취득)
- 정보처리산업기사 (2019.12 취득)
- 리눅스마스터 2급 (2021.4 취득)

프로젝트

팀 프로젝트

배틀웁스

DiaryWalk

개인 프로젝트

RandomSurvival

Tunnel

JewelPop

논 문

1인칭 슈팅 게임 속 AI 분류 및 개선

배틀윱스

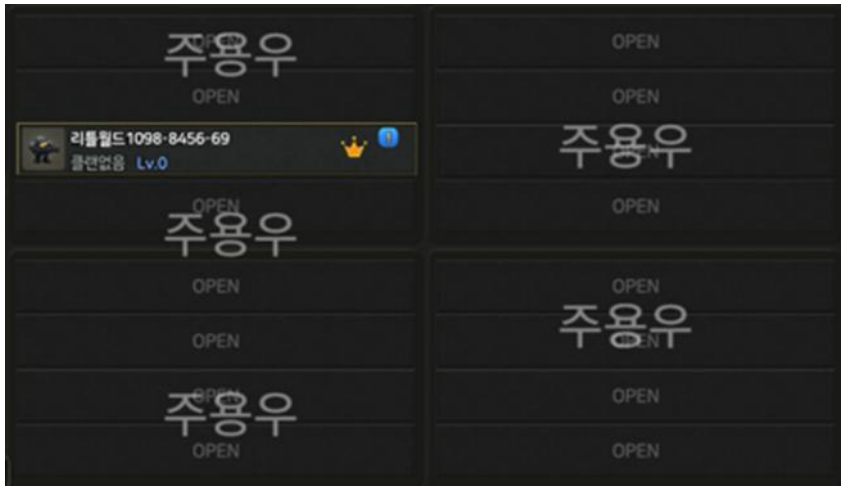
- 프로젝트 기본정보 [구글 플레이 스토어]

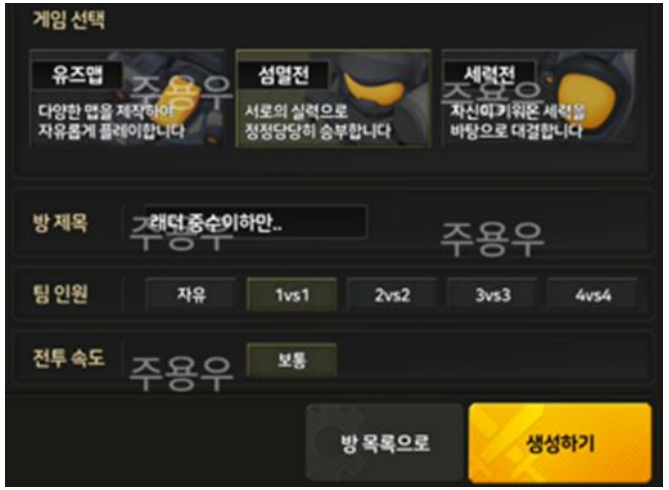
프로젝트 명	배틀윱스 모바일 게임 개발 (개발 당시 이름 "익스트림 월드")
프로젝트 기간	2021.09~2022.12
프로젝트 인원	6명
설명	모바일 RTS 게임
담당 역할	클라이언트, 서버 사이드 개발
깃허브 주소	회사 계정으로 private 상태입니다.

배틀윱스 Google Play Store 이미지

- 개발 주요사항
  - 퍼니델 자체 개발 엔진을 사용하여 개발
  - OpenGL을 사용하여 삼각형 그리기 등 수학적 업무 수행
  - 클라이언트에서 UI/UX 작업과 콘텐츠 로직, Firebase GoogleAD 연동
  - 서버에서 클라이언트로부터 수신한 패킷 관리
  - 서버와 데이터베이스 간 연동 관리

배틀윱스 UI 작업 화면





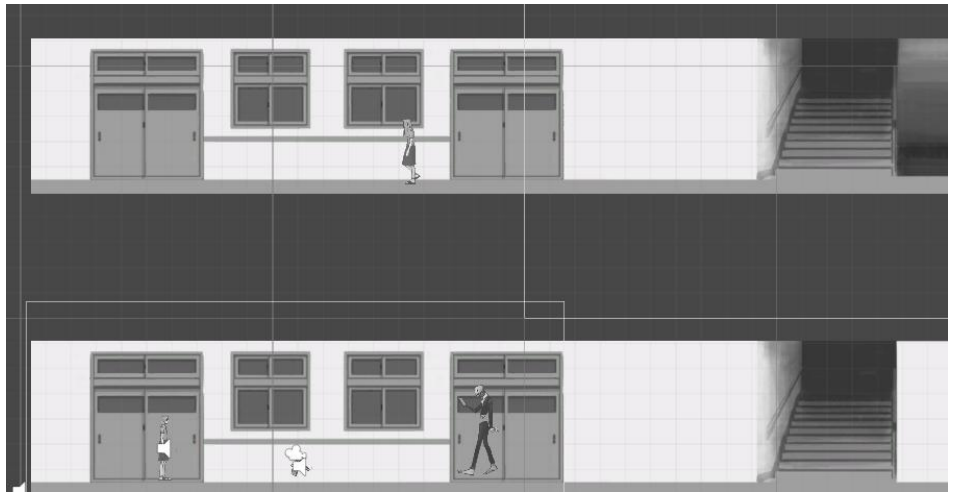
개발 당시 게임 속 일부 UI들

UI를 배치할 때 모바일 게임 특성 상의 UX를 디자이너와 협업하여 많은 UI 작업을 완성하였습니다.
게임 진행에 필요한 UI들은 기획자와 협업하여 상세한 UI 업무를 받았습니다.
프로그램 개발 팀원들과는 작성한 코드에 주석을 통해 구현 과정을 작성하는 등 긴밀하게 협업하였습니다.

DiaryWalk

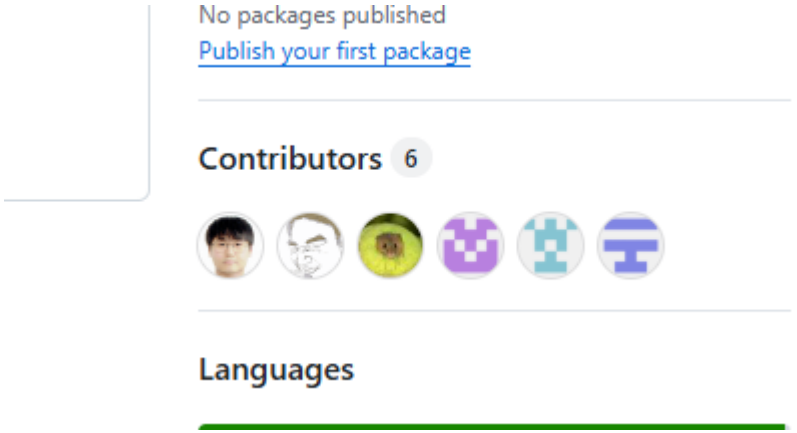
- 프로젝트 기본정보

프로젝트 명	DiaryWalk 공포 게임 개발
프로젝트 기간	2021.03~2021.6
프로젝트 인원	6명
설명	횡스크롤 2D 공포 게임
담당 역할	클라이언트 개발



DiaryWalk 이미지

- 개발 주요사항
  - UNITY 엔진을 이용한 게임 개발
  - UI 외 게임 로직 구현
  - 플레이어, 몬스터 등 캐릭터 간의 상호작용 이벤트 구현



DiaryWalk 코드 스니펫

```
public GameObject[] buttontemp;
public Image diaryimage;
public Image keyimage;
public static Button instance; //싱글톤 선언
Unity 메시지 | 참조 0개
void Awake(){
Button.instance = this; //싱글톤 부여
    buttontemp = new GameObject[30];
    for(int i=0;i<10;i++) {
    buttontemp[i] = GameObject.Find("Button" + i.ToString()); //Button이라는 이름의 오브젝트들을 찾아서 배열에 넣음
    }
}
참조 3개
public void ChangeImageToDiary(int value) {
    buttontemp[value].GetComponent<Image>().sprite=diaryimage.sprite; //배열에 넣은 오브젝트의 이미지 컴포넌트의 스프
    buttontemp[value].GetComponent<Image>().color=new Color(255,255,255,1);
}
참조 3개
public void ChangeImageToKey(int value) {
    buttontemp[value].GetComponent<Image>().sprite=keyimage.sprite;
```

Button 관리 스크립트

게임에 존재하는 많은 버튼들을 관리할 방법에 대해 고민하였습니다.
당시에도 지금과 같이 유지 보수가 용이하게 하기 위한 방법을 항상 생각했습니다.
다시 생각해보면 결코 효율적인 방식은 아니지만, Find를 사용하여 버튼을 일괄적으로 캐싱하여 관리하였습니다.

RandomSurvival

- 프로젝트 정보 다운로드

프로젝트 명	“RandomSurvival” 액션 게임 개발
프로젝트 기간	2025.07~
프로젝트 인원	1인 개발
설명	30분 간 살아남는 탑다운 뷰 액션 게임
담당 역할	게임 기획, 개발

RandomSurvival 게임 이미지

- 개발 주요사항
  - UNITY Engine을 사용하여 개발
  - Addressable 라이브러리를 라벨과 key를 이용한 동적 리소스 관리와 비동기 로드
  - 리소스, 자식 오브젝트들을 Enum-ObjectType 매핑하여 유연하게 관리
  - ScriptableObject를 이용한 플레이어, 무기 정보 이용
  - CSV를 통한 게임 맵 로드/수정/저장 방식 채택
  - NavMesh가 아닌 A*를 이용한 움직임 최적화

RandomSurvival 맵 로드 / 맵 수정 및 저장

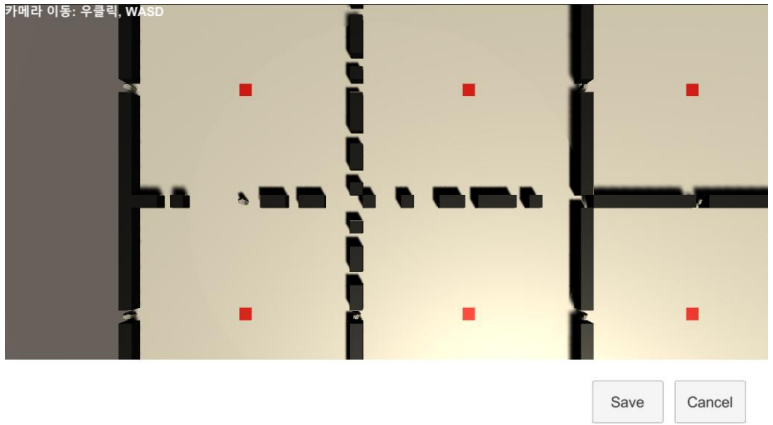
MapMaker

```
public class MapMaker : MonoBehaviour
{
    private TextAsset mapCSV;
    public static string[,] sMap;
    public Dictionary<string, TileSO> TileMap = new Dictionary<string, TileSO>();
    public void Awake()
    {
        mapCSV = Addressables.LoadAssetAsync<TextAsset>("Map").WaitForCompletion();
        string[] lines = mapCSV.text.Split(new[] { '₩n', '₩r' }, StringSplitOptions.RemoveEmptyEntries);

        sMap = new string[lines.Length, lines[0].Trim().Split(',').Length];
        for (int y = 0; y < sMap.GetLength(0); y++)
        {
            string[] line = lines[y].Trim().Split(',');
            for (int x = 0; x < sMap.GetLength(1); x++)
            {
                if (!TileMap.ContainsKey(line[x])) TileMap.Add(line[x],
Addressables.LoadAssetAsync<TileSO>(line[x]).WaitForCompletion());
                sMap[y, x] = line[x];
            }
        }
    }
    public void Start()
    {
        List<GameObject> gos = new List<GameObject>();
        for (int i = 0; i < sMap.GetLength(0); i++)
        {
            for (int j = 0; j < sMap.GetLength(1); j++)
            {
                gos.Add(Instantiate(TileMap[sMap[i, j]].preFab, GridHelper.GridToWorld(new Vector2Int(j, i)),
Quaternion.identity));
                gos[gos.Count-1].name = sMap[i, j];

            }
        }
```

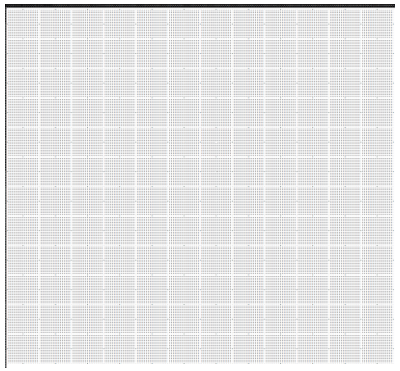
Map.csv

...
,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,x,gw,gw,gw,gw,
gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,x,gw,gw,gw,g
w,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,x,gw,gw,g
w,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,x,gw,g
w,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,x,g
w,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,
x,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,g
w,x,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,g
w,gw,x,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,g
w,gw,gw,x,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,gw,g
w,gw,gw,gw,x,gw,gw,gw,gw,gw,gw,
...

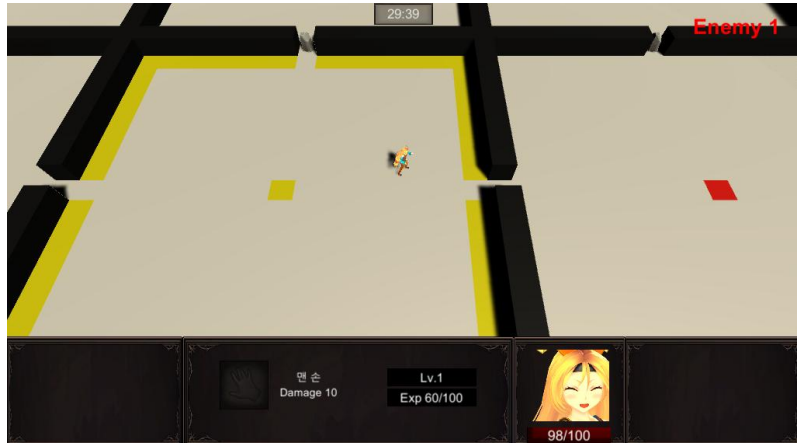
RandomSurvival 맵 로드 / 맵 수정 및 저장



맵 에디터를 통한 맵 수정



CSV로 저장



게임 시작 시 맵을 로드

스타크래프트처럼 오랫동안 사랑받는 게임들은 대부분 게임을 자신의 입맛대로 고칠 수 있다는 것입니다.
사용자들이 쉽게 맵을 수정하고 저장할 수 있도록 맵 에디터를 만들어 보았습니다.
그렇다고 꼭 사용자들이 이 기능을 사용하도록 할 필요는 없습니다.
개발, 운영에 있어 맵을 손쉽게 수정할 수 있다는 것은 회사의 게임 유지보수에 큰 도움이 되기 때문입니다.

RandomSurvival Scriptable Object(SO) 구성

ManagerObject : MonoBehaviour

```
...
 static public StatManager playerStatObj = new StatManager();
...
```

StatManager

```
public WeaponDatabase WeaponStatDB { get; private set; }
public PlayerDatabase PlayerStatDB { get; private set; }
```

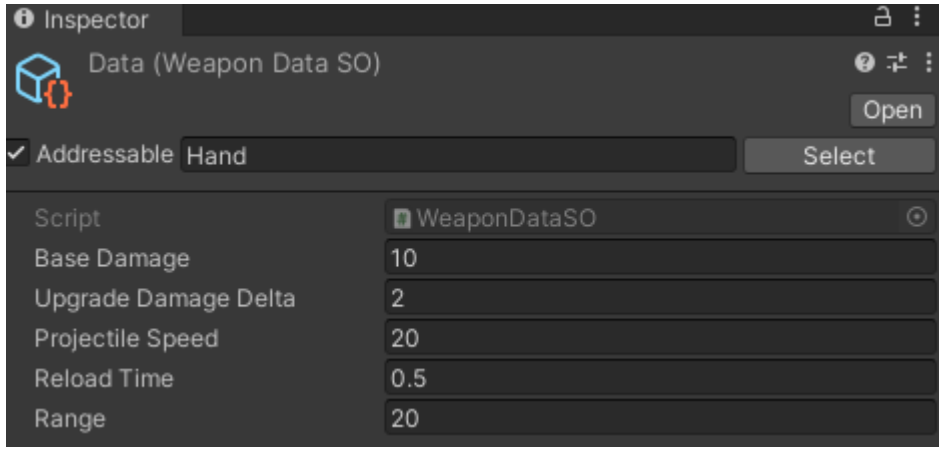
WeaponDatabase

```
 public WeaponDatabase()
    {
...
        var soMap = Util.MapEnumToAddressables<Weapons,
WeaponDataSO>("WeaponDataSO"); //라벨 통해서 맵 등록
...
}
```

PlayerDatabase

```
 public PlayerDatabase()
    {
...
        PlayerDataSO playerData = new PlayerDataSO();
        playerData =
Addressables.LoadAssetAsync<PlayerDataSO>("PlayerDataSO").WaitForCompl
etion();
...
    }
```

RandomSurvival Scriptable Object(SO) 구성



WeaponDataSO 인스펙터

SO 수정 값 자동 적용

실제 런칭되는 게임들은 업데이트에 따라 잦은 수정이 생기게 됩니다.
그럴 때 마다 코드를 뜯어고치는데 불필요한 비용을 들이는 것을 방지하는 것이 좋습니다.
위처럼 SO와 같은 데이터에셋을 이용하면 게임 업데이트를 쉽게하여 유지보수를 극대화할 수 있습니다.

RandomSurvival Addressable Label - Enum 매핑

WeaponDatabase

```
public class WeaponDatabase
{
    public enum Weapons { Hand, Bow, Magic, Gun }

    public Dictionary<Weapons, WeaponDataSO> _map = new
Dictionary<Weapons, WeaponDataSO>();


    public WeaponDatabase()
    {
        _map = Util.mapDictionaryWithKeyLoad<Weapons, WeaponDataSO>(); //키
이름과 enum 이름 일치 통해서 맵 등록

    }

    public bool TryGetInfo(Weapons w, out WeaponDataSO info) =>
_map.TryGetValue(w, out info);
    public WeaponDataSO GetInfo(Weapons w) => _map[w];
}
```

WeaponDataSO // 무기 종류마다 존재

```
using UnityEngine;

[CreateAssetMenu(fileName = "NewWeaponData", menuName
= "Game/WeaponData")]
public class WeaponDataSO : ScriptableObject
{
    public GameObject ProjectilePrefab;
    public AudioClip ShootSound;
    public Sprite WeaponIcon;
...
}
```

RandomSurvival Addressable Label - Enum 매핑

Addressable 이중 라벨

```
참조 1개
public WeaponDatabase()
{
    var soMap = Util.MapEnumToAddressablesByLabels<Weapons, WeaponDataSO>("WeaponDataSO"); //라벨 통해서 맵 등록
    var iconMap = Util.MapEnumToAddressablesByLabels<Weapons, Sprite>("WeaponIcon");
    var sfxMap = Util.MapEnumToAddressablesByLabels<Weapons, AudioClip>("WeaponFireSound");
    var projMap = Util.MapEnumToAddressablesByLabels<Weapons, GameObject>("WeaponProjectile");


    _map = new Dictionary<Weapons, WeaponInfo>
    {
        { Weapons.Hand,  new WeaponInfo("맨손", soMap[Weapons.Hand],  iconMap[Weapons.Hand],  sfxMap[Weapons.Hand],  projMap[Weapons.Hand]) },
        { Weapons.Bow,   new WeaponInfo("활",   soMap[Weapons.Bow],   iconMap[Weapons.Bow],   sfxMap[Weapons.Bow],   projMap[Weapons.Bow]) },
        { Weapons.Magic, new WeaponInfo("마법", soMap[Weapons.Magic], iconMap[Weapons.Magic], sfxMap[Weapons.Magic], projMap[Weapons.Magic]) },
    };
}
```

라벨 중에서 Enum과 매칭 되면 라벨을 이중 검색, 매핑

게임에는 앞서 설명한 무기 데이터를 관리하는 SO파일을 포함하여 여러 리소스들이 존재합니다.
단순히 Resources.Load() 함수를 사용하면 리소스가 변경될 때마다 파일의 이름, 위치 등을 통일해야만 하는 문제가 있습니다.
이러한 문제를 방지하고자 Addressable을 사용하여 key를 이용해 로드하면 파일의 이름, 위치를 통일할 필요가 없습니다.
특히 제 프로그램에서는 key 말고도 라벨을 사용하여 자동 매핑하도록 되어 있어 유지보수가 용이합니다.

Tunnel

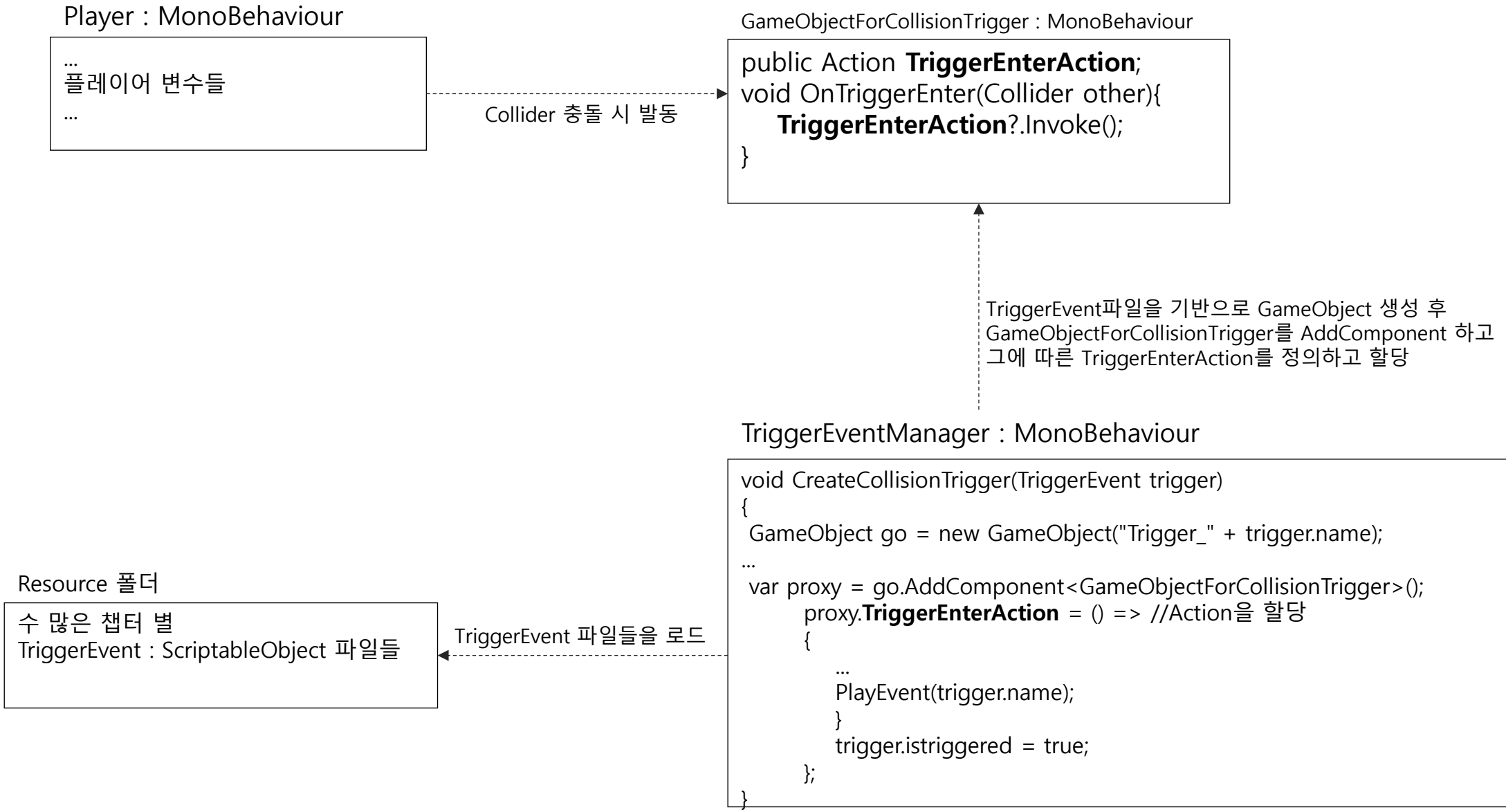
- 프로젝트 정보 다운로드

프로젝트 명	“Tunnel” 공포 게임 개발
프로젝트기간	2025.05~2025.07
프로젝트 인원	1인 개발
설명	짧은 플레이타임의 1인칭 3D 공포 게임
담당 역할	게임 기획, 개발

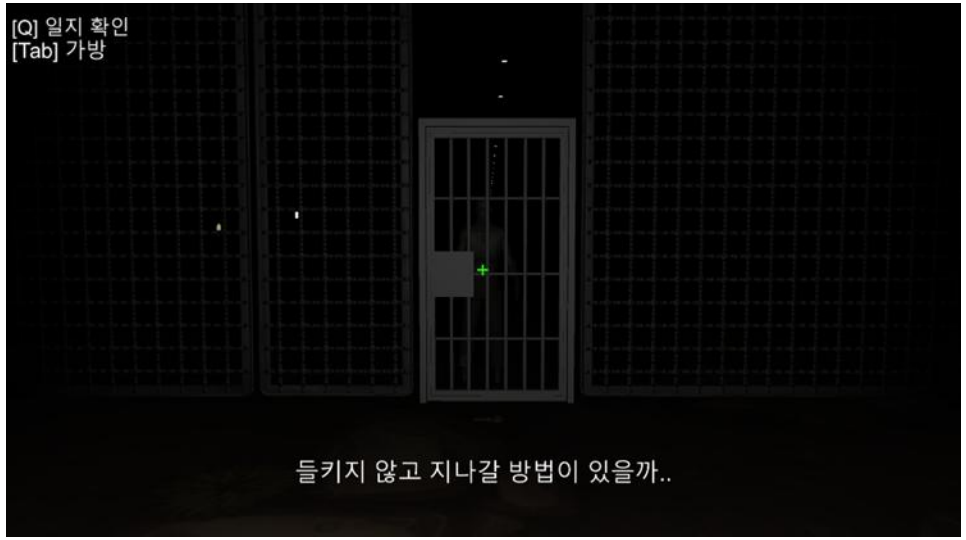
Tunnel 게임 이미지

- 개발 주요사항
  - UNITY Engine을 사용하여 개발
  - UGUI를 사용하여 인터페이스 구현
  - 클래스 간의 행동 분리
  - UI 내의 버튼의 액션에 대해 enum-Action 매핑하여 관리
  - 트리거-이벤트의 ScriptableObject를 이용한 이벤트 관리

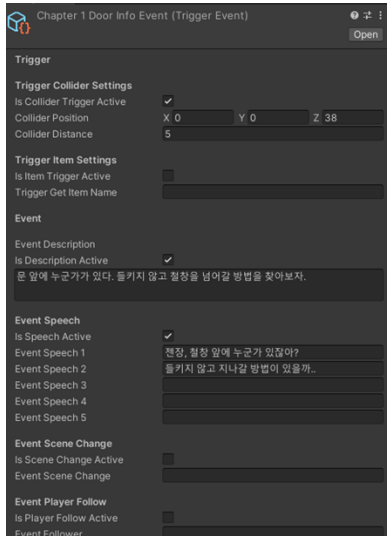
Tunnel 이벤트 시스템 구성



Tunnel 이벤트 시스템 구성



인게임에서 이벤트가 발동한 상황



TriggerEvent의 인스펙터 화면

챕터 별 TriggerEvent들

기획자와 협업 시, 코드를 건드리지 않고 쉽게 게임 이벤트트리거를 설정하도록 하는 것이 필요합니다.
어떤 조건에서 어떤 이벤트를 발생시킬지 인스펙터를 통해 편하게 설정할 수 있게 해야 합니다.
플에이어가 일정 거리 내에 들어오면 게임 목표를 업데이트하고 대사를 재생하도록 누구든 인스펙터 화면에서 손쉽게 설정할 수 있어야 합니다.

Tunnel 이벤트 시스템 구성

```
public List<TriggerEvent> triggerevents;
private Dictionary<string, TriggerEvent> triggereventmap;


참조 1개
public void OnAwake()
{
    triggerevents = new List<TriggerEvent>();
    triggereventmap = new Dictionary<string, TriggerEvent>();
}
```

```
if (triggereventmap[eventname].isSpeech) // 대화 존재할 경우
{
    Speech(triggereventmap[eventname].eventSpeeches);
}

if (triggereventmap[eventname].isSceneChange) //씬 전환 존재할 경우
{
    SceneManager.LoadScene(triggereventmap[eventname].eventscenechange
}

if (triggereventmap[eventname].isFollow)
{
    GameObject.Find(triggereventmap[eventname].eventfollower).GetCompo
```

TriggerEventManager.cs

```
[CreateAssetMenu(menuName = "Game/Event Data")]
Unity 스크립트 | 참조 11개
public class TriggerEvent : ScriptableObject
{

    [NonSerialized]
    public bool istriggered = false;

    public bool isColliderTriggerActive;
    public Vector3 triggerPosition;
    public float triggerdistance;
    public bool isItemTriggerActive;
    public string tiggerGetItem;


    public bool isDescription;
    public string eventdescription;

    public bool isSpeech;
    public string []eventSpeeches = new string[5];

    public bool isSceneChange;
    public string eventscenechange = "";
```

TriggerEvent.cs

TriggerEvent(ScriptableObejct 상속)에서 사용할 변수들을 정의하고, 에디터에서 생성, 관리합니다.
TriggerEventManager에서 챕터 별 폴더 내의 TriggerEvent 파일들을 로드하고 Dictionary로 맵핑하여 각 트리거에 대한 이벤트를 할당합니다.
이제 계속 업데이트 되는 다양한 이벤트들을 에디터 화면에서 손쉽게 제작할 수 있어 유지보수에 큰 도움이 됩니다.

Tunnel UGUI 관리

TitleUI : UIButtonBinder

```
...
 public enum ButtonID
    {
        LoadButton,
        StartButton,
        SettingsButton,
        GameExitButton
    }
...
 protected override Dictionary<ButtonID, UnityAction> GetHandlers()
    {
        return new Dictionary<ButtonID, UnityAction>
        {
            { ButtonID.LoadButton, () =>
                { SceneManager.LoadScene($"Chapter"+PlayerPrefs.GetInt("chap"));
                InGameUIOn();} },
            { ButtonID.StartButton, () => { SceneManager.LoadScene("Chapter1");
                InGameUIOn();} },
            { ButtonID.SettingsButton, () => { settingsScreen.SetActive(true); } },
            { ButtonID.GameExitButton, () => { Application.Quit(); } }
        };
    }
```

SettingsUI : UIButtonBinder

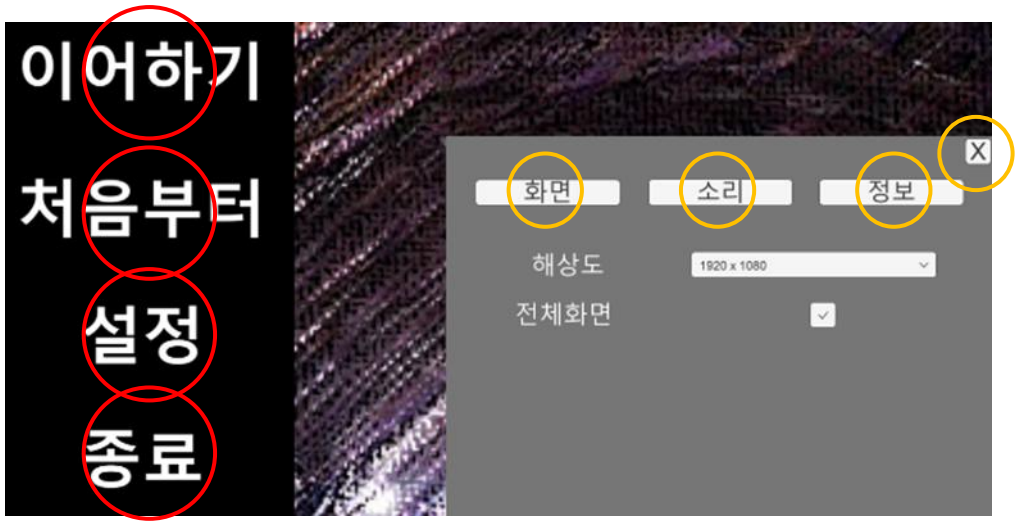
```
public enum ButtonID
    {
        SettingExitButton,
        VideoButton,
        SoundButton,
        InfomationButton,
    }
protected override Dictionary<ButtonID, UnityAction> GetHandlers()
    {
        return new Dictionary<ButtonID, UnityAction>
        {
            { ButtonID.SettingExitButton, () => { settingsScreen.SetActive(false); } },
            { ButtonID.VideoButton, VideoButtonClick },
            { ButtonID.SoundButton, SoundButtonClick },
            { ButtonID.InfomationButton, InformationButtonClick }
        };
    }
```

UIButtonBinder : MonoBehaviour

```
 protected Dictionary<T, Button> buttonMap;
 protected abstract Dictionary<T, UnityAction> GetHandlers();
protected virtual void Awake()
{
    buttonMap = new Dictionary<T, Button>();

    var buttons = GetComponentsInChildren<Button>(true);
    foreach (var btn in buttons)
    {
        if (Enum.TryParse<T>(btn.gameObject.name, out var id))
        {
            buttonMap[id] = btn;
        }
    }

    foreach (var pair in GetHandlers())
    {
        if (buttonMap.TryGetValue(pair.Key, out var button))
            button.onClick.AddListener(pair.Value);
        else
            Debug.LogWarning($"[UIButtonBinder] Button '{pair.Key}' not found");
    }
}
```

GetCoponentsInChildren<button> 으로 자식 오브젝트의 버튼을 가져오고
미리 GetHandlers의 값을 할당
이후, 상속받은 클래스에서 GetHandlers를 정의하는 것으로 버튼에 대한 이벤트를 등록

Tunnel UGUI 관리



타이틀 UI 화면

```cs
public abstract class UIButtonBinder<T> : MonoBehaviour where T : struct, Enum
{
    protected Dictionary<T, Button> buttonMap;
    참조 3개
    protected abstract Dictionary<T, UnityAction> GetHandlers();

    Unity 메시지 | 참조 4개
    protected virtual void Awake()
    {
        buttonMap = new Dictionary<T, Button>();

        var buttons = GetComponentsInChildren<Button>(true);
        foreach (var btn in buttons)
        {
            if (Enum.TryParse<T>(btn.gameObject.name, out var id))
            {
                buttonMap[id] = btn;
            }
        }
```

UIButtonBinder.cs

```cs
protected override Dictionary<ButtonID, UnityAction> GetHandlers()
{
    return new Dictionary<ButtonID, UnityAction>
    {
        { ButtonID.LoadButton, MainGameLoadButtonClick },
        { ButtonID.StartButton, () => { SceneManager.LoadScene("Chapter1"); } },
        { ButtonID.SettingsButton, () => { settingsScreen.SetActive(true); } },
        { ButtonID.GameExitButton, () => { Application.Quit(); } }
    };
}
```

TitleUI.cs

기획이 수정되면서 계속 수정되는 버튼들을 한 번에 관리할 Map이 필요했습니다.
자식 버튼들을 "enum-버튼" 형식으로 dictionary를 사용하여 맵핑하는 역할을 하는 UIButtonBinder라는 추상 클래스를 구현하였습니다.
UI 캔버스마다 적용되는 컴포넌트 클래스들은 UIButtonBinder를 상속받고 추상 클래스내의 함수를 정의합니다.

Tunnel UGUI 관리

```
        "picturebook" => HandleInteraction(ItemType.PictureBook, obj, "E 읽기"),
        "textbook" => HandleInteraction(ItemType.Textbook, obj, "E 읽기"),
        "door" => HandleInteraction(ItemType.Door, obj, "E 읽기"),
        "key" => HandleInteraction(ItemType.Key, obj, "E 획득"),
        "movingwall" => HandleInteraction(ItemType.MovingWall, obj, "E 밀기"),
        _ => false
    };
}

참조 5개
bool HandleInteraction(ItemType type, GameObject obj, string message)
{
    UseUIOn?.Invoke(message);
    if (Input.GetKeyDown(KeyCode.E))
    {
        UseItem?.Invoke(type, obj);
    }
    return true;
}
```

PlayerInteractor.cs

```
void assignUIEvents()
{
    PlayerInteractor.UseUIOn += (str) => {
        UseUI.transform.GetChild(0).GetComponent<Text>().text = str;
        UseUI.SetActive(true);
    };
    PlayerInteractor.UseItem += (Type, go) =>
    {
        if(Type == PlayerInteractor.ItemType.PictureBook)
        {
            go.GetComponent<AudioSource>().Play();
            PictureBookUI.transform.GetChild(0).GetComponent<Image>().sprite = go.GetC
            PictureBookUI.SetActive(true); // 활성화
        }
        else if (Type == PlayerInteractor.ItemType.Textbook)
        {
            go.GetComponent<AudioSource>().Play();
            TextBookUI.transform.GetChild(0).GetChild(0).GetComponent<Text>().text = g
            TextBookUI.SetActive(true); // 활성화
        }
        else if(Type == PlayerInteractor.ItemType.Door)
        {
```

InGameUI.cs

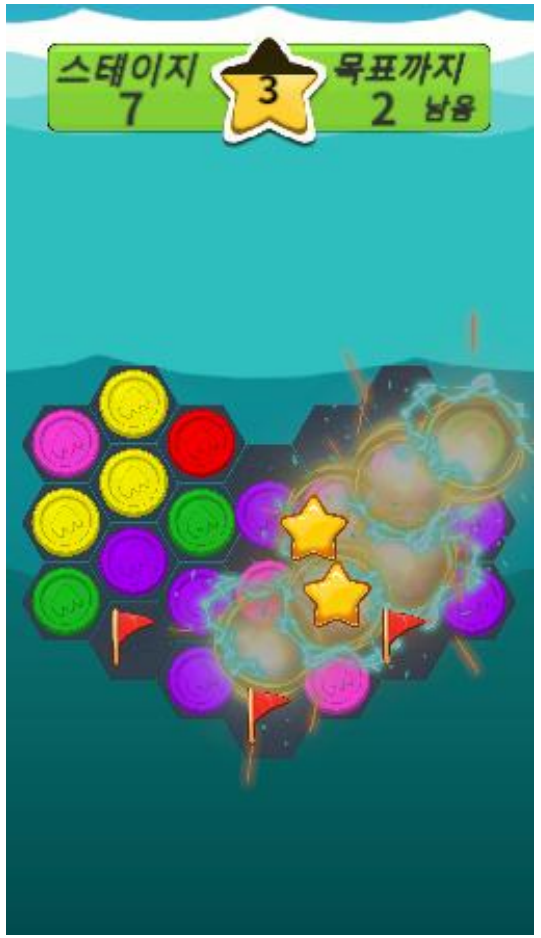
플레이어가 열쇠를 조준하면 "E 획득" 글자가 화면에 출력되어야 하는 등, UI의 변수를 참조해야 할 일이 많이 생깁니다.
그러나 직접적으로 다른 클래스 내 함수를 호출하는 것은 높은 결합도를 가지게 되어 SRP 원칙을 위반합니다.
저는 여기서 PlayerInteractor에서 InGameUI를 직접 참조하는게 아닌 Action 할당을 이용하는 방식을 사용했습니다.
UI는 UI역할을, Player는 Player 역할을 수행하도록 클래스를 구분하는 것이 구조적으로 타당하며 유지 보수에 용이합니다.

JewelPop

- 프로젝트 정보 다운로드

프로젝트 명	“JewelPop” 캐주얼 게임 개발
프로젝트기간	2025.09~2025.10
프로젝트 인원	1인 개발
설명	보석을 이어 터트리는 캐주얼 퍼즐 게임
담당 역할	게임 개발(CookApps 과제)

- 개발 주요사항
  - UNITY Engine을 사용하여 개발
  - UGUI를 사용하여 인터페이스 구현
  - 클래스 간의 행동 분리
  - UI 내의 버튼의 액션에 대해 enum-Action 매핑하여 관리
  - 트리거-이벤트의 ScriptableObject를 이용한 이벤트 관리



JewelPop 게임 이미지

JewelPop JSON을 통한 스테이지 로드

Stage1.JSON

```
{
  "stage": 1,
  "goalType" : "Joker",
  "goalScore" : 5,
  "grids": [

    {"y":0,"x":0,"type":"p"},
    {"y":0,"x":1,"type":"y"},
    {"y":0,"x":2,"type":"r"},
    {"y":0,"x":4,"type":"y"},
    {"y":0,"x":5,"type":"g"},
    {"y":0,"x":6,"type":"g"},

    {"y":1,"x":0,"type":"y"},
    {"y":1,"x":1,"type":"y"},
    {"y":1,"x":2,"type":"g"},
...
```

JSON 데이터를 클래스화

public class LevelManager<T> where T : JSONVars

```
public T currentLevel;
public void Init(string json)
{
    //JSON 토대로 레벨 데이터 초기화
    currentLevel = JsonUtility.FromJson<T>(json);
    goalScore = currentLevel.goalScore;
    goalType = Enum.Parse<GoalType>(currentLevel.goalType);
    currentStage = currentLevel.stage;
    currentScore = 0;
    //UI 초기화
    ActionManager.setCurrentStageUI(currentStage);
    ActionManager.setScoreUI(currentScore, goalScore);
    AppManager.instance.soundManager.PlaySound(Sounds.BGM1, 0.25f, true);
}
```

클래스화한 데이터를 토대로 맵 제작

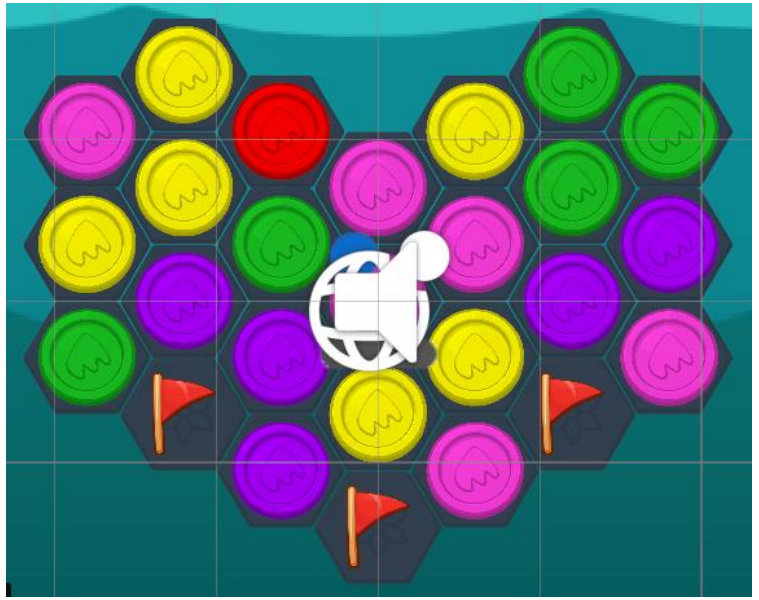
public class MapManager

```
private void SetBlocks(JSONVars jsonVars)
{
    foreach (var grid in jsonVars.grids)
    {
        YX yx = new YX(grid.y, grid.x);
        board.Add(yx,
UnityEngine.Object.Instantiate(AppManager.instance.resourceManager.blockParentObjectPrefab).
GetComponent<BlockParent>());
        board[yx].name = $"y{grid.y}x{grid.x}";
        board[yx].SetGridPositionYX(yx);
        board[yx].SetUnityPositionYX(grid.x % 2 == 1 ? (-grid.y * yStep + yStep * 0.5f, grid.x *
xStep) : (-grid.y * yStep, grid.x * xStep));

        GameObject child =
UnityEngine.Object.Instantiate(AppManager.instance.resourceManager.blockPrefabs[Enum.Parse
<BlockPrefabs>(grid.type)], board[yx].transform);
        child.GetComponent<BlockChild>().SetBlockType(Enum.Parse<BlockPrefabs>(grid.type));
    }
}
```

JewelPop JSON을 통한 스테이지 로드


```
"grids": [

  {"y":0,"x":0,"type":"p"},
  {"y":0,"x":1,"type":"y"},
  {"y":0,"x":2,"type":"r"},
  {"y":0,"x":4,"type":"y"},
  {"y":0,"x":5,"type":"g"},
  {"y":0,"x":6,"type":"g"},

  {"y":1,"x":0,"type":"y"},
  {"y":1,"x":1,"type":"y"},
  {"y":1,"x":2,"type":"g"},
  {"y":1,"x":3,"type":"p"},
  {"y":1,"x":4,"type":"p"},
  {"y":1,"x":5,"type":"g"},
  {"y":1,"x":6,"type":"pp"},

  {"y":2,"x":0,"type":"g"},
  {"y":2,"x":1,"type":"pp"},
  {"y":2,"x":2,"type":"pp"},
  {"y":2,"x":3,"type":"p"},
  {"y":2,"x":4,"type":"y"},
  {"y":2,"x":5,"type":"pp"},
  {"y":2,"x":6,"type":"p"},


  {"y":3,"x":1,"type":"j"},
  {"y":3,"x":2,"type":"pp"},
  {"y":3,"x":3,"type":"y"},
  {"y":3,"x":4,"type":"p"},
  {"y":3,"x":5,"type":"j"},
```


블록 배치 JSON 파일 작성



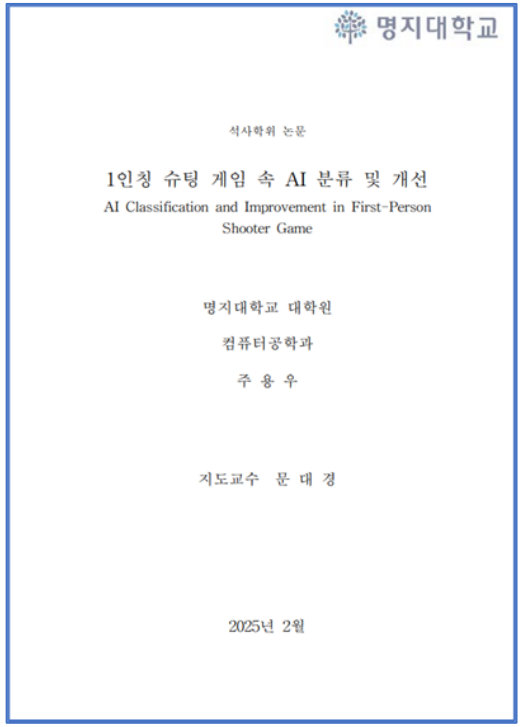
자동 제작된 스테이지

다른 프로젝트들과 마찬가지로 스테이지 구성은 기획자의 영역입니다.
스테이지 정보가 바뀔 때마다 유니티를 키는 것은 매우 비효율적이며 개발 속도를 낮추는 행위입니다.
JSON 파일을 로드하여 맵을 제작하도록 하여 기획자가 쉽게 수정할 수 있게 편안한 기획 환경을 만들어 줍니다.

1인칭 슈팅 게임 속 AI 분류 및 개선

- 논문 정보

논문 명	1인칭 슈팅 게임 속 AI 분류 및 개선
논문 저자	주용우, 문대경
지도 교수	문대경(前 아이펀팩토리 대표)
논문 내용	1인칭 슈팅 게임(FPS) 속 AI의 실존하는 문제들을 유명 FPS 게임 속 AI들을 분석하여 알아보고 여러 게임 속 AI 간 전투를 실험을 통해 시뮬레이션 하여 특성에 따라 분류하고 문제를 해결하기 위한 알고리즘을 제안하여 FPS 게임 속 AI를 설계하고 구현하기 위한 가이드라인을 제공한다.
논문 열람 주소	명지대학교 디지털 논문관


명지대학교

석사학위 논문

1인칭 슈팅 게임 속 AI 분류 및 개선

AI Classification and Improvement in First-Person Shooter Game

명지대학교 대학원

컴퓨터공학과

주 용 우

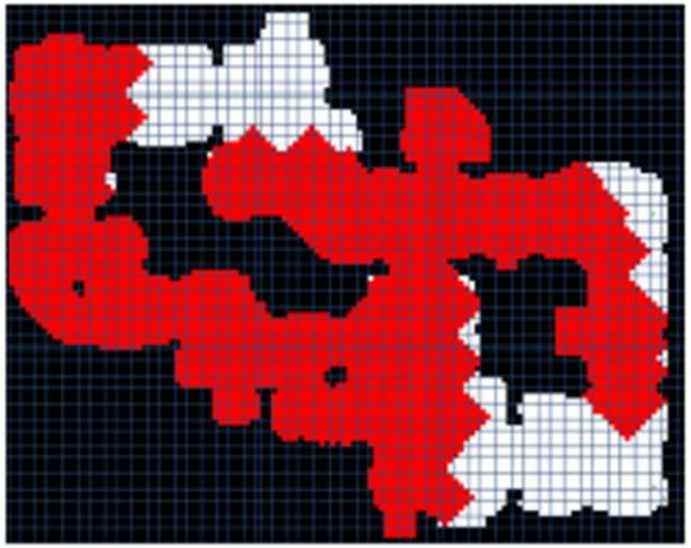
지도교수 문 대 경

2025년 2월

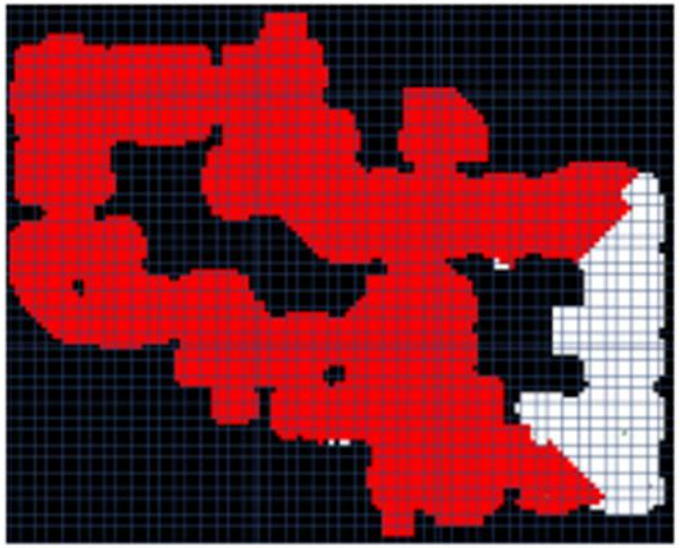
석사 논문 표지

- 논문 주요사항
  - 유명 FPS 게임 4종의 AI 행동 의사코드를 수집 및 정리
  - 각 AI의 협동/전투 지향 성향 분석 및 그래프화
  - AI를 동일 환경에서 전투시켜 시뮬레이션 진행
  - 게임 모드 별 승리를 위한 핵심 행동 알고리즘 도출
  - 기존 AI 이동 로직의 한계(위험 지역 구분 불가) 분석
  - 이동 알고리즘 개선안 FactorPathfinding 제안 및 구현

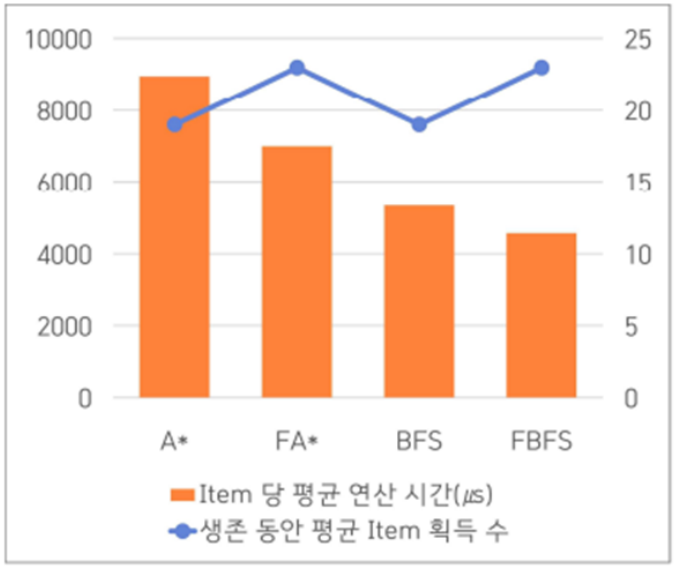
1인칭 슈팅 게임 속 AI 분류 및 개선



[그림 21] 이동 기준 없이 이동할 때



[그림 22] 안전 수치를 기반으로 이동할 때



[그림 20] 상대 위치가 불분명할 때 실험 결과

논문 속 실험 결과 이미지

실제로 실험을 통해 Safetyscore를 통한 명확한 이동 기준을 통해
이동 후보지(이동해야 하는 시야 가장자리)를 줄여 더 빠른 연산과 높은 정확도를 증명하였습니다.
본 논문은 게임을 개발할 때 AI나 캐릭터의 이동 알고리즘을 설계하고 구현하는데 메모리와 연산 시간, 정확도 면에서
큰 도움이 될 수 있어 많은 게임 개발자들에게 게임 AI 이동 알고리즘 개발에 기여합니다.

CONCLUSION

세종이코노미 2022년 4월 호 이미지(우측 첫번째 사진 맨 왼쪽 본인)

"유지보수가 용이해야 게임이 오래 살아남는다"가 제 프로그램 철학입니다.
유지보수가 용이하다는 것은 콘텐츠 추가/수정이 용이하다는 것을 뜻합니다.
기획, 아트, 사운드 등 여러 다른 팀원들이 쉽게 에셋 적용이 가능하게 되고
결국 "**협업**"을 통해 "**빠른 개발**"이 가능해집니다.

수 많은 경험을 통해 결정내린 저의 개발 철학을 통해
귀사에 큰 도움이 될 수 있다는 것을 자신있게 말씀드립니다.

END

주 용 우

Tel: 010-8332-3415

Email: parsnip618@outlook.com

Github: JuYongwoo