Game Portfolio

정재영
Jeong Jae Young

Index

3D Portfolio / 2D Portfolio

Introduction / Workflow / Data / Scene / In Game / Source

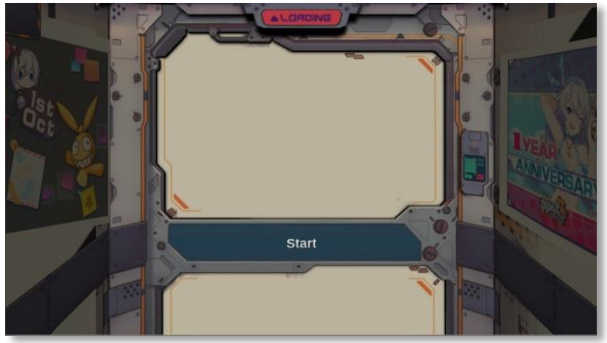
Index

3D Portfolio

https://youtu.be/rk3ORfg-7MU

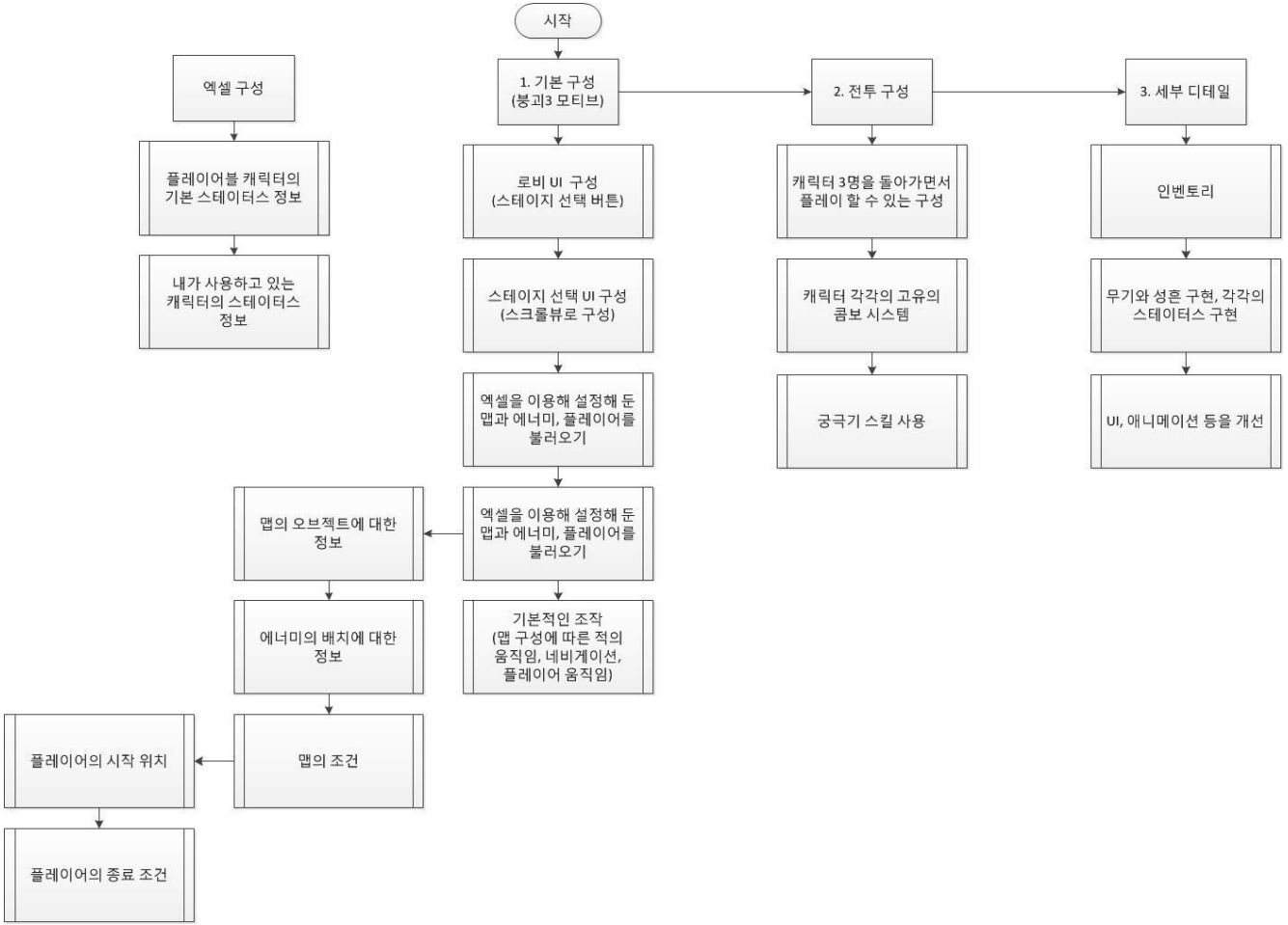
Introduction

총 제작 기간 : 2019/08/01 ~ 2019/09/06(37일)



포트폴리오로 제작한 게임은 현재 mihoyo에서 제작하고 현재 서비스 중인 ‘붕괴 3rd’를 모작하여서 제작하였다.

Workflow


시작
엑셀 구성
플레이어블 캐릭터의 기본 스테이터스 정보
내가 사용하고 있는 캐릭터의 스테이터스 정보
1. 기본 구성 (붕괴3 모티브)
로비 UI 구성 (스테이지 선택 버튼)
스테이지 선택 UI 구성 (스크롤뷰로 구성)
엑셀을 이용해 설정해 둔 맵과 에너미, 플레이어를 불러오기
엑셀을 이용해 설정해 둔 맵과 에너미, 플레이어를 불러오기
맵의 오브젝트에 대한 정보
에너미의 배치에 대한 정보
맵의 조건
플레이어의 시작 위치
플레이어의 종료 조건
기본적인 조작 (맵 구성에 따른 적의 움직임, 네비게이션, 플레이어 움직임)
2. 전투 구성
캐릭터 3명을 돌아가면서 플레이 할 수 있는 구성
캐릭터 각각의 고유의 콤보 시스템
궁극기 스킬 사용
3. 세부 디테일
인벤토리
무기와 성흔 구현, 각각의 스테이터스 구현
UI, 애니메이션 등을 개선

Data

게임 데이터 관리

UserInfoData.json* | UserCharInfoData.json | INVENTORY_WEAPON.json | INVENTORY_STIGMA.json

```
{
"NickName":"altair",
"Level":2,
"CurEnergy":45,
"CurEXP":5,
"Gold":200,
"MainChar":0
}
```

Json Data

	A	B	C
1	USER_INFO_LEVEL	USER_INFO_MAX_ENERGY	USER_INFO_MAX_EXP
2	1	50	45
3	2	55	55
4	3	55	65
5	4	60	75
6	5	60	85
7	6	65	95
8	7	65	105
9	8	70	115
10	9	70	125
11	10	75	135
12	11	75	145
13	12	80	155
14	13	80	165
15	14	85	175
16	15	85	185
17	16	90	195
18	17	90	205
19	18	95	215
20	19	95	225
21	20	100	235
22	21	100	245
23	22	105	255
24	23	105	265
25	24	110	275
26	25	110	285
27	26	115	295
28	27	115	305
29	28	120	315
30	29	120	325
31	30	125	335

Excel Table.csv

게임에서 관리되는 모든 데이터는 Unity 자체 제공 JsonUtility와 Excel .csv 파일로 관리하고 있다.

Data

JsonUtil

```
namespace JSON
{
    public class JsonUtil
    {
        [Serializable]
        private class Wrapper<T>
        {
            public T[] Items;
        }

        public static string ToJson(object obj)
        {
            return JsonUtility.ToJson(obj);
        }

        public static string ToJson<T>(T[] arr)
        {
            Wrapper<T> wrapper = new Wrapper<T>();
            wrapper.Items = arr;
            return ToJson(wrapper);
            //배열 형식의 JSON ToJson
        }

        public static T FromJson<T>(string jsonData)
        {
            return JsonUtility.FromJson<T>(jsonData);
        }

        public static T[] FromArrJson<T>(string FileName)
        {
            Wrapper<T> wrapper = FromJson<Wrapper<T>>(FileName);
            return wrapper.Items;
            //배열 형식의 JSON FromJson
        }
        public static void CreateJson(string FileName, string jsonData)
        {
            FileStream fileStream = new FileStream(FileRoute(FileName), FileMode.Create);
            byte[] data = System.Text.Encoding.UTF8.GetBytes(jsonData);
            fileStream.Write(data, 0, data.Length);
```

유니티에서 자체적으로 제공하는 JsonUtility 기능을 이용하여 Json의 Read & Write 기능을 구현하였고, 배열 형식의 데이터를 사용하기 위해서 Wrapper Class를 구현하였다.

현재 4가지 종류의 Json 파일을 관리하고 있습니다.

Data JsonFile

UserInfoData.json*

스키마: <선택된 스키마가 없음>

```
{
    "NickName": "altair",
    "Level": 2,
    "CurEnergy": 45,
    "CurEXP": 5,
    "Gold": 200,
    "MainChar": 0
}
```

UserInfoData.json

유저의 자체적인 데이터를 관리

UserCharInfoData.json*

스키마: <선택된 스키마가 없음>

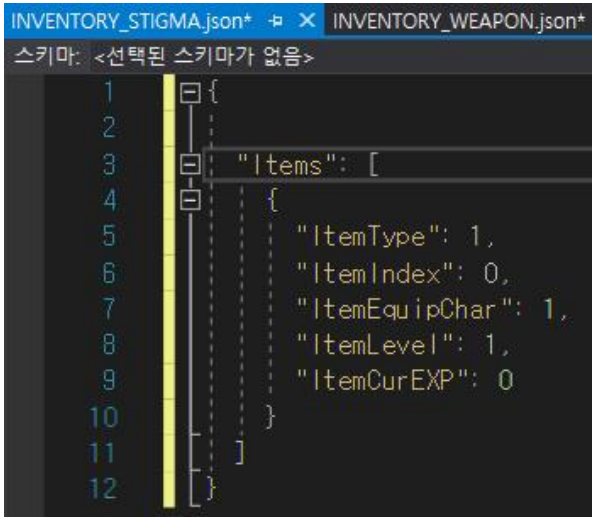
```
{
    "Items": [
        {
            "CharIndex": 0,
            "CharLevel": 2,
            "CharCurEXP": 0,
            "CharWeaponType": 4,
            "CharWeapon": 0,
            "CharStigmaTop": -1,
            "CharStigmaCenter": -1,
            "CharStigmaBottom": -1
        },
        {
            "CharIndex": 1,
            "CharLevel": 2,
            "CharCurEXP": 0,
            "CharWeaponType": 5,
            "CharWeapon": 1,
            "CharStigmaTop": 0,
            "CharStigmaCenter": -1,
            "CharStigmaBottom": -1
        }
    ]
}
```

UserCharInfoData.json

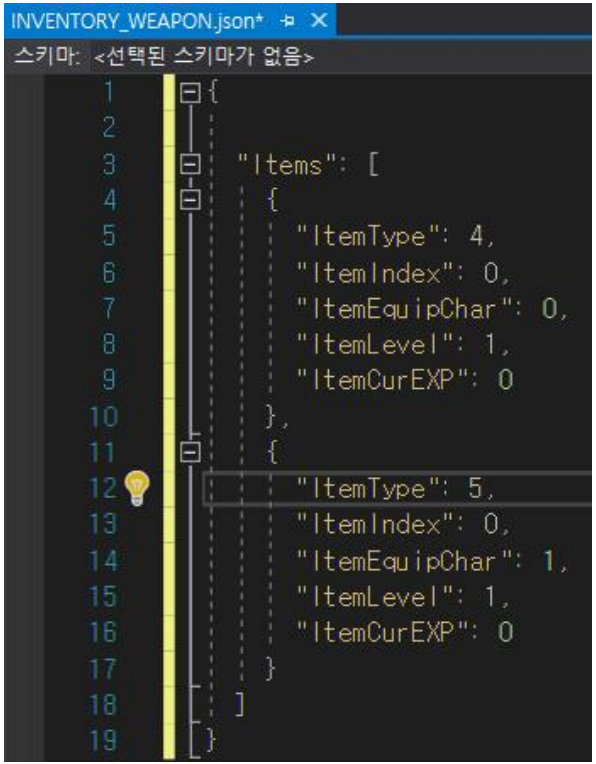
유저가 현재 소지 중인 캐릭터의 데이터를 관리
캐릭터가 장착한 장비등의 인덱스등을 소지 중

Data JsonFile

INVENTORY_STIGMA.json



유저가 가지고 있는 성흔(방어구 개념)의 데이터를 관리

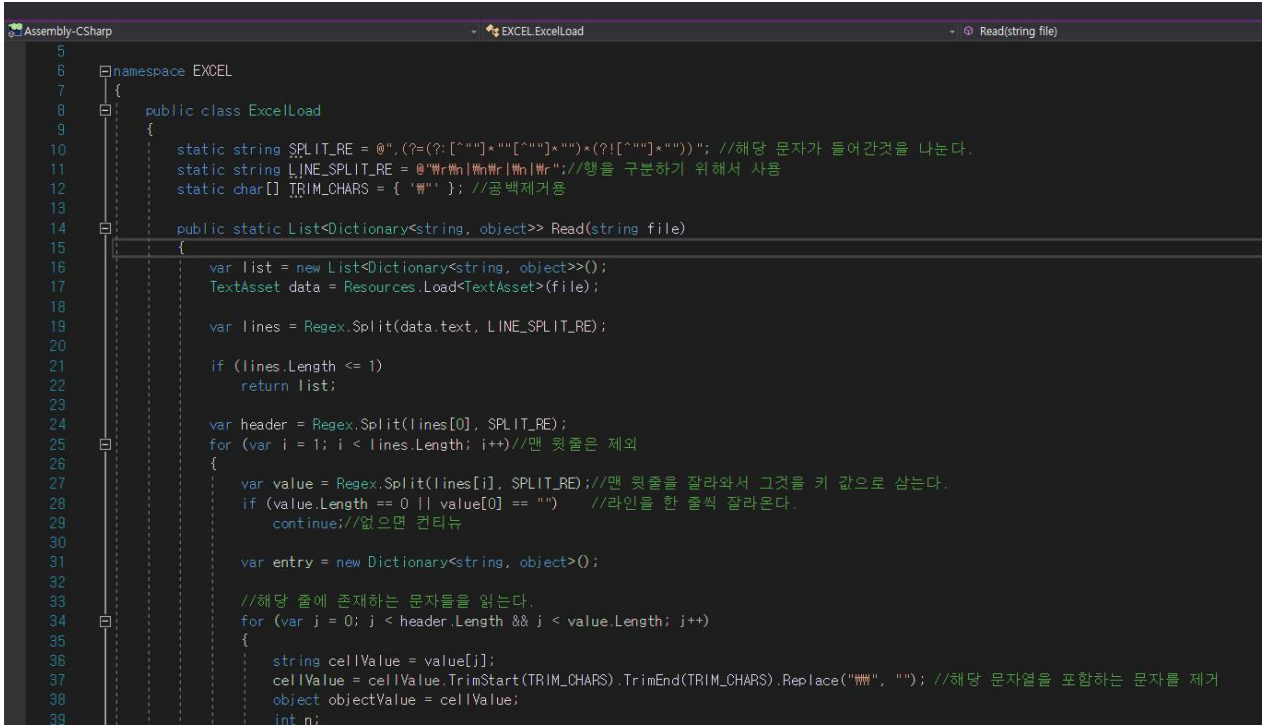


INVENTORY_WEAPON.json

유저가 가지고 있는 무기의 데이터를 관리

Data

ExcelLoad



Excel의 csv 파일을 읽기 위해서 구현한 class로 Excel의 테이블 형식 데이터를 읽어 List<Dictionary<Key, Value>>로 저장하여 반환한다.

캐릭터의 Table Data와 스테이지 등의 정보를 관리하고 있다.

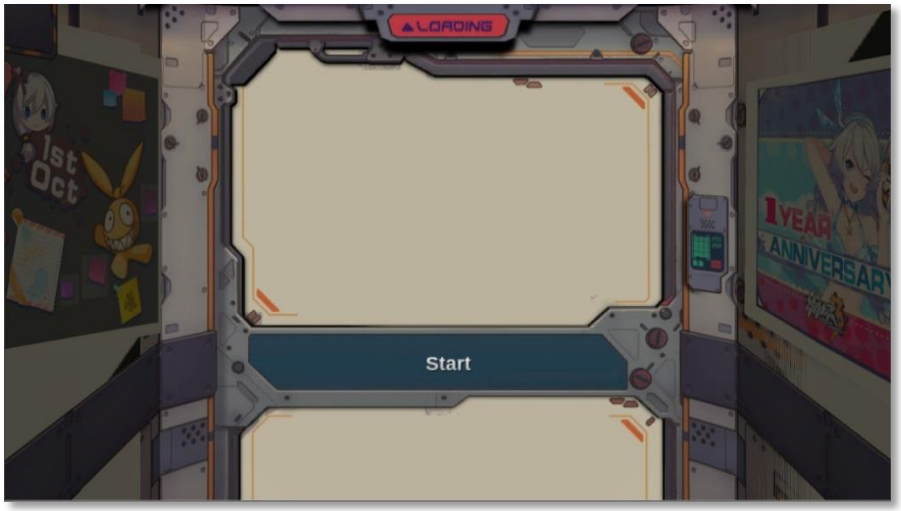
Data ExcelFile

	A	B	C	D	E	F	G	H
1	CHAR_NAME	CHAR_LEVEL	CHAR_MAX_HP	CHAR_MAX_SP	CHAR_MAX_EXP	CHAR_ATK	CHAR_DEF	CHAR_CRI
2	UnityChan	1	450	100	50	65	30	12
3	UnityChan	2	500	100	70	75	35	15
4	UnityChan	3	550	100	90	90	40	18
5	UnityChan	4	600	100	110	105	45	21
6	UnityChan	5	650	100	130	120	50	24
7	UnityChan	6	700	100	150	135	55	27
8	UnityChan	7	750	100	170	150	60	30
9	UnityChan	8	800	100	190	165	65	33
10	UnityChan	9	850	100	210	180	70	36
11	UnityChan	10	900	110	230	195	75	39
12	UnityChan	11	950	110	250	210	80	42
13	UnityChan	12	1000	110	270	225	85	45
14	UnityChan	13	1050	110	290	240	90	48
15	UnityChan	14	1100	110	310	255	95	51
16	UnityChan	15	1150	110	330	270	100	54
17	UnityChan	16	1200	110	350	285	105	57
18	UnityChan	17	1250	110	370	300	110	60
19	UnityChan	18	1300	110	390	315	115	63
20	UnityChan	19	1350	110	410	330	120	66
21	UnityChan	20	1400	120	430	345	125	69
22	UnityChan	21	1450	120	450	360	130	72
23	UnityChan	22	1500	120	470	375	135	75
24	UnityChan	23	1550	120	490	390	140	78
25	UnityChan	24	1600	120	510	405	145	81
26	UnityChan	25	1650	120	530	420	150	84
27	UnityChan	26	1700	120	550	435	155	87
28	UnityChan	27	1750	120	570	450	160	90
29	UnityChan	28	1800	120	590	465	165	93
30	UnityChan	29	1850	120	610	480	170	96
31	UnityChan	30	1900	120	630	495	175	99

	A	B	C
1	USER_INFO_LEVEL	USER_INFO_MAX_ENERGY	USER_INFO_MAX_EXP
2	1	50	45
3	2	55	55
4	3	55	65
5	4	60	75
6	5	60	85
7	6	65	95
8	7	65	105
9	8	70	115
10	9	70	125
11	10	75	135
12	11	75	145
13	12	80	155
14	13	80	165
15	14	85	175
16	15	85	185
17	16	90	195
18	17	90	205
19	18	95	215
20	19	95	225
21	20	100	235
22	21	100	245
23	22	105	255
24	23	105	265
25	24	110	275
26	25	110	285
27	26	115	295
28	27	115	305
29	28	120	315
30	29	120	325
31	30	125	335

Excel Table 데이터는 레벨 당 스테이터스나 스테이지에서 사용되는 데이터나, 스테이지 적의 정보 등 유저가 가지고 있을 필요성이 적인 데이터를 관리하고 있다.

Scene | FirstLoadScene



평소 상태

유저 데이터가 없으면 새로운 데이터를 만든다.

게임을 처음 시작하면 처음으로 로딩이 되는 Scene으로 여기서 유저의 데이터 Json을 로딩한다. 만약, 데이터가 없으면 이름을 입력 받아서 새로운 데이터를 만든다.

Scene FirstLoadScene

- UserInfo : GSingleton<UserInfo>

```
7   public class UserInfo : GSingleton<UserInfo>
8   {
9       private UserData m_UserData;    //유저 데이터
10      private UserInventory m_UserInventory;  //유저 인벤토리, 아이템등의 데이터
11      private UserCharData m_UserCharData;    //유저 소유 캐릭터 데이터
12      // Start is called before the first frame upda te
13
14      /// <summary>
15      /// 셋팅 관련
16      /// </summary>
17      public void Init()
18      {
19          FirstLoadScene first = GameObject.Find("FirstLoad").GetComponent<FirstLoadScene>();
20          UnityEvent Event = new UnityEvent();
21          Event.AddListener(first.UserInfoComplete);
22          Event.Invoke();
23          if (JSON.JsonUtil.FileCheck("UserInfoData"))
24          {
25              //해당 데이터를 확인,
26              //유저 데이터를 모두 셋팅 하면 호출할 이벤트 설정
27              var UserTable = EXCEL.ExcelLoad.Read("Excel/Table/UserTable");
28              m_UserData = new UserData(UserTable);
29          }
30
31          //인벤토리 셋팅
32          m_UserInventory = new UserInventory();
33
34          //내가 가진 캐릭터
35          if (JSON.JsonUtil.FileCheck("UserCharInfoData"))
36          {
37              m_UserCharData = new UserCharData();
38              CharInfoData[] Char = JSON.JsonUtil.LoadArrJson<CharInfoData>("UserCharInfoData");
39              for (int i = 0; i < Char.Length; i++)
40              {
41                  string file = "Excel/Table/" + Char[i].CharIndex + "_Char_Table";//테이블 데이터
42                  var CharTable = EXCEL.ExcelLoad.Read(file);
43                  m_UserCharData.Init(Char[i], CharTable);
44              }
45          }
```

FirstLoadScene에서 Json과 Excel 데이터를 이용해서 Singleton Class인 UserInfo Class에 해당 데이터들을 저장, 관리한다.

Scene

FirstLoadScene

AlwaysFirst.cs

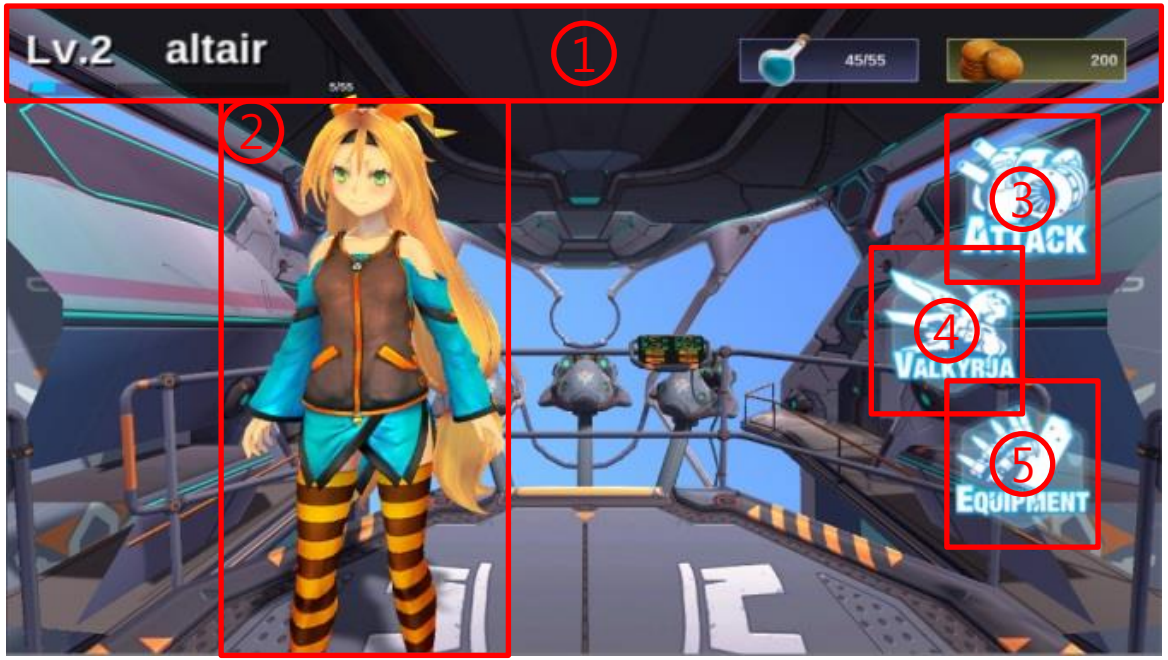
Assembly-CSharp

AlwaysFirst

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
5   public class AlwaysFirst : MonoBehaviour
6   {
7       [RuntimeInitializeOnLoadMethod(RuntimeInitializeLoadType.BeforeSceneLoad)]
8       private static void Awake()
9       {
10          if (SceneManager.GetActiveScene().name.CompareTo("FirstLoadScene") != 0)
11          {
12              SceneManager.LoadScene("FirstLoadScene");
13          }
14      }
15  }
```

처음 씬을 고정시켜주기 위해서 RuntimeInitializeOnLoadMethod를 사용하였다.

Scene | LobbyScene



LobbyScene에서 대부분의 기능이 집중되어 있으므로 이를 중심적으로 설명하겠다.

Scene | LobbyScene

- **유저 인터페이스**

유저의 닉네임, 레벨, 현재 경험치를 Slider와 Label로 구현

유저가 현재 소지하고 있는 Gold와 출격 시 소모되는 Energy의 구현

본 프로젝트의 모든 UI는 NGUI를 이용해서 제작하였으며, Panel 단위로 관리하고 있다.

Scene | LobbyScene

- 유저 메인 캐릭터

캐릭터 창에서의 모델링

캐릭터 정보 창에서의 모델링

유저가 메인으로 지정한 캐릭터를 표시하는 기능으로 메인 외에도 다양한 상황에서 사용되므로 재활용이 가능하게 오브젝트 풀링으로 구현하였다.

Scene | LobbyScene

- **CharPoolManager**

CharPoolManager.cs
Assembly-CSharp

```cs
using UnityEngine;
using System.Collections.Generic;
using System;

//서로 다른 오브젝트 풀링 기법
public class CharPoolManager : GSingleton<CharPoolManager>
{
    public List<CharObjectPool> m_PoolManger = new List<CharObjectPool>();

    public void Set(string strPoolName, string [] strPrefabs, int [] iarr, int iObjectCount)
    {
        //서로 다른 오브젝트를 풀링으로 관리
        try
        {
            GameObject ObjectPool = ResourceLoader.CreatePrefab("Prefabs/CharObjectPool");   //오브젝트 풀링
            ObjectPool.name = strPoolName;
            CharObjectPool Pool = ObjectPool.GetComponent<CharObjectPool>();
            Pool.Init(strPrefabs, iarr, iObjectCount, Pool.transform);
            m_PoolManger.Add(Pool);
            //게임 오브젝트로 생성한 뒤 하위 컴포넌트로 셋팅
        }
        catch (NullReferenceException ex)
        {
            Debug.Log(ex);
        }
    }

    public bool PushToPool(string strPoolName, int iIndex, GameObject item)
    {
        //인덱스 기반 반납
        CharObjectPool pool = GetPoolItem(strPoolName);
        if (pool.name == string.Empty)
            return false;

        pool.PushToPool(item, iIndex, pool.transform);
        return true;
    }
```

CharPoolManager.cs
Assembly-CSharp

```cs
    [System.Serializable]
    public class CharObject
    {
        public GameObject st_Object;
        public int st_Index;
    }
```

CharObjectPool Class를 가진 Prefabs을 만들어 해당 Class의 CharObject 구조체에 정보를 저장, CharObjectPool을 부모로 가진 캐릭터 Object를 생성한다.

Scene | LobbyScene

- CharPoolManager

CharPoolManager.cs

Assembly-CSharp | CharPoolManager

```
37      }
38
39      public GameObject PopFromPool(string strPoolName, int iIndex)
40      {
41          //인덱스 기반 대출
42          CharObjectPool pool = GetPoolItem(strPoolName);
43          if (pool == null)
44              return null;
45
46          return pool.PopFromPoll(iIndex);
47      }
```

CharPoolManager.cs

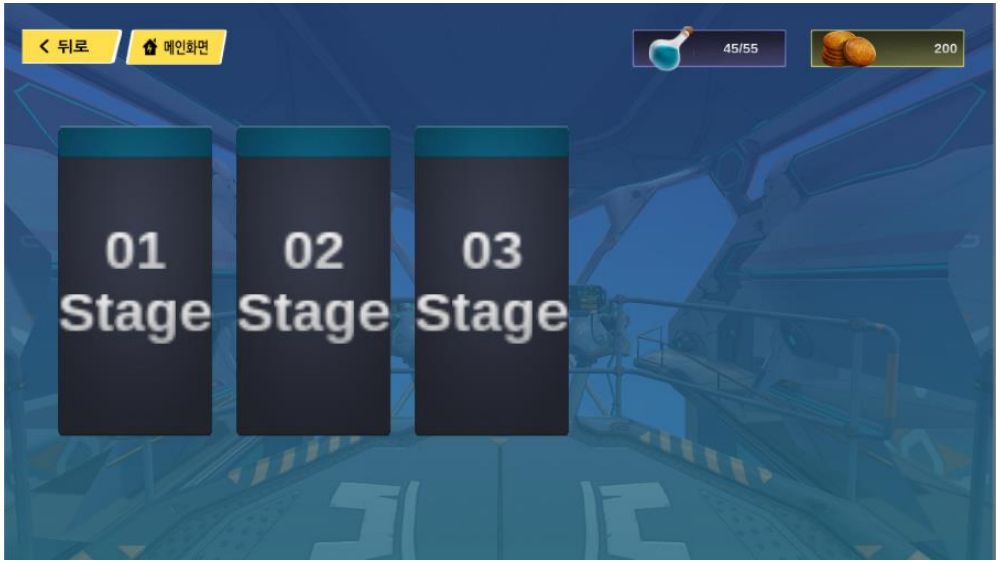
Assembly-CSharp | CharObject

```
49      public GameObject PopFromPoll(int iIndex)  //순서가 중요한 오브젝트
50      {
51          if (iIndex > m_ListPool.Count || m_ListPool.Count == 0)  //인덱스 초과
52              return null;
53
54          CharObject Class = FindCharObject(iIndex);
55          if(Class != null)
56          {
57              GameObject Object = Class.st_Object;
58              Class.st_Object = null;
59              return Object;  //해당 캐릭터의 오브젝트
60          }
61          else
62              return null;
63      }
64
65      private CharObject FindCharObject(int iIndex)
66      {
67          foreach (var s in m_ListPool)
68          {
69              if (s.st_Index == iIndex)   //해당 캐릭터의 인덱스와 맞는가?
70              {
71                  return s;   //반환
72              }
73          }
74          return null;
75      }
```

캐릭터의 오브젝트를 사용할 때는 캐릭터의 고유의 인덱스를 기반으로 해당 캐릭터의 Object를 반환 받는다.

Scene | LobbyScene

- **Attack(StagePanel)**



Attack 버튼을 누르면 스테이지를 선택 할 수 있는 창이 나오며
Scrollview와 Grid를 이용하였다.

Scene | LobbyScene

- **Stage Information**

	A	B	C	D	E	F	G	H	I	J	K
1	Prefab	LocX	LocY	LocZ	QuaW	QuaX	QuaY	QuaZ	ScaleX	ScaleY	ScaleZ
2	chengqiar	0	0	0	0.707107	0.707107	0	0	25	25	1
3	Zhandaos	-8	2	-2	0.670566	-0.67057	0.224368	0.224368	0.7	1.4	0.8
4	Zhandaos	8	2	9	0.702904	-0.7029	0.07698	0.07698	0.7	0.8	0.7

Map_Object를 관리

	A	B	C	D
1	Index	LocX	LocY	LocZ
2	0	0	0	-10

시작 지점 등의 Pos 관리

	A	B	C	D	E	F
1	CHAR_INDEX	CHAR_NAME	CHAR_MAX_HP	CHAR_ATK	CHAR_DEF	CHAR_CRI
2	0	Brute Warrior	150	10	15	5
3	1	Ninja Warrior	100	20	10	15
4	2	Sorceress Warrio	50	30	10	20

Stage_Enemy 정보 관리

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	MaxWave	CurWave	Index	LocX	LocY	LocZ	QuaW	QuaX	QuaY	QuaZ	ScaleX	ScaleY	ScaleZ
2	2	0	0	-9	5.5	-1.5	1	0	0	0	1	1	1
3	2	0	1	8.6	4.9	8.5	1	0	0	0	1	1	1
4	2	0	2	8.8	0.6	-1.5	1	0	0	0	1	1	1
5	2	1	0	-6	5.5	-5	1	0	0	0	1	1	1
6	2	1	1	8.6	4.9	8.5	1	0	0	0	1	1	1
7	2	1	2	8.8	0.6	-7	1	0	0	0	1	1	1

Wave 형식의 게임 플레이 타입에 맞춰서 각 웨이브에 따른 Enemy의 Transform 관리

Stage를 구성하는 정보는 총 네 가지로 MapManager로 만든 Excel .csv 파일로 관리하며 GameScene으로 넘어갈 때 이를 모두 런타임으로 생성한다.

Scene | LobbyScene

- **Attack(StageReadyPanel)**

	A	B	C	D	E	F
1	MAP_TYPE	MAP_TIME	MAP_ENERGY	MAP_CLEAR_EXP	MAP_CLEAR_GOLD	MAP_CLEAR_ITEM
2	Normal	0.0f	10	50	200	1;1;0;
3	Normal	0.0f	10	50	200	2;1;0;
4	Normal	0.0f	10	50	200	3;1;0;

Stage_Table.csv
스테이지의 타입과 소모 Energy, 획득 EXP, Gold, Item등을 정리하여 가지고 있다.

Stage를 클릭하면 캐릭터를 선택할 수 있는 칸과 함께 해당 스테이지의 정보가 나오는 데 이는 Stage_Table.csv로 관리하였다.

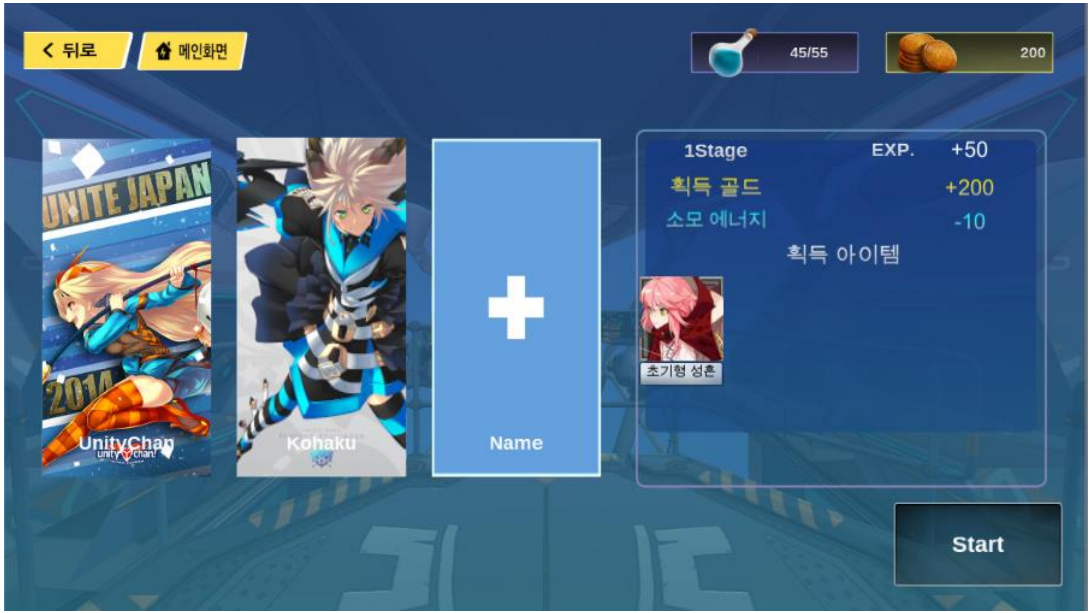
Scene | LobbyScene

- **Attack(ValkyrjaPanel)**

캐릭터 선택 창을 클릭하면 현재 내가 소지하고 있는 캐릭터를 보여주며 이는 메인 로비에 있는 Valkyrja버튼을 클릭하여도 동일한 화면을 재활용 한다.

Scene | LobbyScene

- **Attack(StageReadyPanel)**



스테이지의 선택과 유저가 선택한 캐릭터 리스트 정보는 MonoSingleton 형식의 GameManager에 정보를 저장하고 GameScene으로 넘어간다.

Scene | LobbyScene

- GameManager : MonoSingleton<GameManager>

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GameManager : MonoSingleton<GameManager>
{
    /// <summary>
    /// 캐릭터 관련 인자들
    /// </summary>
    private int[] m_ListCharIndex;      //내가 선택한 캐릭터 인덱스들
    private int m_iCurSelectChar = -1;   //캐릭터 선택 단계에서 내가 선택한 캐릭터
    private int m_iCurGameChar = -1;     //게임 속에서 내가 현재 선택한 캐릭터
    private GameObject m_SelectCharMain = null;
    private GameObject m_SelectChar = null;    //내가 선택한 캐릭터의 프리팹

/// <summary>
/// 맵 관련 인자들
/// </summary>
    private int m_iCurStage = -1;       //현재 선택한 스테이지
    public List<Dictionary<MAP_DATA, object>> m_ListMapData = new List<Dictionary<MAP_DATA, object>>();

    /// <summary>
    /// 아이템 장착에 관련된 인자들
    /// </summary>
    private int m_iCurSelectItme = -1;
    private ITEM_TYPE m_eItemType;
    private INVENTORY_TYPE m_eInvenType;


    public void Init()
    {
        m_ListCharIndex = new int[] { -1, -1, -1 };   //3개
        m_iCurGameChar = -1;
        m_iCurSelectChar = -1;
        m_iCurSelectItme = -1;
        m_SelectCharMain = GameObject.Find("SelectCharModel");
    }

    /// <summary>
```

LobbyScene에서 선택한

StageIndex, 유저가 선택한 캐릭터

인덱스 배열을 GameScene에서 사

용하기 위해서

DontDestroyOnLoad를 적용한

GameManager에 저장하였다.

Scene | LobbyScene

- **Valkyrja(ValkyrjaPanel)**

스테이지의 캐릭터 선택 창과 똑같은 기능을 하며 해당 Panel에서 캐릭터의 정보를 확인하거나 장비를 장착 할 수 있다.

Scene | LobbyScene

- **Valkyrja(CharInfoPanel)**

캐릭터 스테이터스

캐릭터 착용 장비

캐릭터 착용 성흔

CharInfoPanel의 각각의 창은 TweenControll을 이용하여서 표현하였으며, 각각의 캐릭터 정보는 Json과 Excel을 이용해서 관리하고 있다.

Scene | LobbyScene

- Valkyrja(ItemSelectPanel)

장비 장착 또한, 캐릭터가 가지고 있는 장비 타입과 성흔 부위(Top, Center, Bottom)를 구분하여 같은 타입의 아이템이면 장착할 수 있도록 데이터를 이용해서 구현하였다.

- **Equipment(EquipmentPanel)**

EquipmentPanel은 ItemSprite Prefabs를 생성하여 RunTime으로 ScrollView와 Grid를 구성하였다.

Scene

LoadScene


```
private void Start()
{
    StartCoroutine(SceneLoad());
}

public static void SceneLoad(string sceneName)
{
    nextScene = sceneName;
    SceneManager.LoadScene("LoadScene", LoadSceneMode.Single);
}

private IEnumerator SceneLoad()
{
    yield return null;

    async_operation = SceneManager.LoadSceneAsync(nextScene);   //씬 매니저로 로딩
    async_operation.allowSceneActivation = false;
    float timer = 0.0f;

    while(!async_operation.isDone)
    {
        yield return null;
        timer += Time.deltaTime;
        if (async_operation.progress >= 0.9f)
        {
            LoadSlider.value = Mathf.Lerp(LoadSlider.value, 1.0f, timer);

            if (LoadSlider.value == 1.0f)
                async_operation.allowSceneActivation = true;
        }
        else
        {
            LoadSlider.value = Mathf.Lerp(LoadSlider.value, async_operation.progress, timer);
            if (LoadSlider.value >= async_operation.progress)
            {
                timer = 0f;
            }
        }
    }
```

Loading은 SceneManager와 Coroutine을 이용해서 구현하였고,로딩을 통해서 GameScene을 구성한다.

GameScene

GameScene의 구성은 '붕괴 3rd'와 유사하게 제작하였으며
캐릭터 교체 등도 구현하였다.

Enemy AI & Navmesh

```
void Start()
{
    m_CallBack = Camera.main.GetComponent<FollowCam>().CameraSet;   //카메라 셋팅 콜백

    string strStage = Util.ConvertToString(GameManager.instance.ReturnStage()); //첫시작
    string strFile = "Excel/StageExcel/" + strStage + "/Map_Object";  //해당스테이지의 맵 Info
    List<Dictionary<string, object>> Info = EXCEL.ExcelLoad.Read(strFile);
    strFile = "Excel/Table/Stage_Table";   //전체 맵의 table 데이터
    List<Dictionary<string, object>> Table = EXCEL.ExcelLoad.Read(strFile);
    strFile = "Excel/StageExcel/" + strStage + "/Event_Pos";   //해당 맵의 시작 등의 좌표
    List<Dictionary<string, object>> Pos = EXCEL.ExcelLoad.Read(strFile);
    m_MapManager = new MapManager(m_arrObject[(int)OBJECT_INDEX.OBJECT_BACKGROUND].transform, Info, Table, Pos);
    //배경 오브젝트 설정
    m_arrObject[(int)OBJECT_INDEX.OBJECT_BACKGROUND].GetComponent<NavMeshSurface>().BuildNavMesh();
    //네비메쉬 서페이스로 런타임 베이크

    m_PlayerManager = new PlayerManager(m_arrObject[(int)OBJECT_INDEX.OBJECT_PLAYER].transform, m_arrObject[(int)OBJECT_INDEX.OBJECT_PARTICLE].transform);
    //플레이어 셋팅

    strFile = "Excel/StageExcel/" + strStage + "/Enemy_Pos";
    Pos = EXCEL.ExcelLoad.Read(strFile);
    strFile = "Excel/StageExcel/" + strStage + "/Enemy_Info";
    Info = EXCEL.ExcelLoad.Read(strFile);
    m_EnemyMangaer = new EnemyManager(m_arrObject[(int)OBJECT_INDEX.OBJECT_ENEMY].transform, Pos, Info);
    //에너미 셋팅

    var vecPos = m_MapManager.ReturnEventPos();
    m_PlayerManager.PlayerSet(0, vecPos[0], Quaternion.identity, JumpEnd);  //가장 첫번째 캐릭터와, 포지션 셋팅
    //스타트에서 처음 포지셔닝을 셋팅

    m_CallBack(m_PlayerManager.GetCharTR());    //카메라 콜백 함수 선언
    m_EnemyMangaer.TrSetting(m_PlayerManager.GetCharTR()); //타겟 셋팅
    m_EnemyMangaer.ActiveWave();    //액티브

    PoolManager.instance.Set(POOL_INDEX.POOL_HP_ITEM.ToString(), "Prefabs/HP", 10);
    PoolManager.instance.Set(POOL_INDEX.POOL_SP_ITEM.ToString(), "Prefabs/SP", 10);

    InvokeRepeating("WaveClear", 2.0f, 1.0f);
    InvokeRepeating("PlayerDie", 2.0f, 1.0f);
}
```

GameScene의 Start에서 배경 Object와 Enemy, 플레이어를 셋팅하였고, Enemy AI NavmeshAgent를 위해서 NavmeshSurface 기능으로 베이크 하였다.

InGame

Enemy FSM

```
IEnumerator StateAction()
{
    while(m_eCurState != ENEMY_STATE.STATE_DIE)
    {
        switch (m_eCurState)
        {
            case ENEMY_STATE.STATE_WAIT:    //대기 상태
                m_Animator.SetBool("Attack", false);
                m_Animator.SetBool("Moving", false);
                m_NavMeshAgent.isStopped = true;
                break;
            case ENEMY_STATE.STATE_ATTACK:
                m_Animator.SetBool("Attack", true);
                m_Animator.SetBool("Moving", false);
                transform.LookAt(m_PlayerTR);
                m_NavMeshAgent.isStopped = true;
                m_eCurState = ENEMY_STATE.STATE_WAIT;
                break;
            case ENEMY_STATE.STATE_TRACE:
                m_Animator.SetBool("Attack", false);
                m_Animator.SetBool("Moving", true);
                m_NavMeshAgent.SetDestination(m_PlayerTR.position);
                m_NavMeshAgent.isStopped = false;
                break;
        }
        yield return null;
    }
}
```

```
IEnumerator StateCheck(float fTime)
{
    while(m_eCurState != ENEMY_STATE.STATE_DIE)
    {
        yield return new WaitForSeconds(fTime);

        float Distance = Vector3.Distance(m_PlayerTR.position, transform.position);

        if (Distance <= m_fAttackArea)
        {
            //거리가 사정거리보다 짧아지면 공격 스테이트
            m_eCurState = ENEMY_STATE.STATE_ATTACK;
        }

        if (Distance >= m_fAttackArea)
        {
            //거리가 사정거리보다 길어지면 추적 스테이트
            m_eCurState = ENEMY_STATE.STATE_TRACE;
        }
    }
}
```

Enemy는 유한 상태 머신(FSM)을 구현하고자 State를 체크하는 것과 State 동작을 수행하는 Coroutine을 사용하였다.

PlayerKeyControll

JoyStick과 EventTrigger를 이용해서 Player의 Key를 구현하였으며, 각 캐릭터 마다 고유의 Key Action은 Excel Table을 이용해서 구현하였다.

PlayerKeyControll

- **Excel KeyControll**

	A	B
1	Index	Key
2	0	2;1;100;
3	1	0;0;/0;0;/0;0;
4	2	2;0;/2;0;/2;0;

Index 0 – 캐릭터 궁극기 키
버튼;키 입력 타입;소모 SP;

Index 1 – 캐릭터 액션
1번 액션 버튼;키 입력 타입;/2번 액션 버튼;키 입력 타입;…

#키 입력 타입은 Click/Press로 구분 되며 Coroutine을 사용해서 입력 타입을 구분하였다.

Excel에 순서만 지킨다면 누구나 캐릭터 액션을 구현할 수 있도록 이러한 Excel 타입의 키 입력을 구현하였다.

PlayerKeyControll

- **CharacterChange**

```
public void ChangeChar()
{
    GameObject cur = UIEventTrigger.current.gameObject;
    ChangeButton Button = cur.GetComponent<ChangeButton>();
    if (Button.ChangeOK && m_bChanging)   //바뀌도 됨
    {
        m_Change = Button;
        m_bChanging = false;
        //캐릭터 체인지
        m_EnemyMangaer.Stop();
        //우선 적들을 멈춰 주고
        m_iTmpIndex = Button.ListIndex;
        //버튼의 인덱스를 저장한 다음
        m_PlayerManager.JumpStart();
        //점프 액션 수행
    }
}
```

‘붕괴 3rd’의 주요 특징 중 하나인 캐릭터 교체로 EventTrigger를 이용해 키 입력을 받고 CallBack Event를 통해서 교체를 구현하였다.

```
public void JumpEnd(bool bDie)
{
    //점프 모션이 끝나면 호출
    Transform tr = m_PlayerManager.GetCharTR();

    if(bDie)
    {
        //캐릭터가 죽었을 시에는 현재 남아있는 캐릭터 중 하나로 자동 교체된다.
        m_EnemyMangaer.Stop();
        m_iTmpIndex = m_PlayerManager.DontDie();
    }

    if(m_iTmpIndex >= 0)
    {
        int[] iarr = GameManager.instance.ReturnPlayerList();
        int iCurList = GameManager.instance.ReturnCurPlayer();

        if(m_Change == null)
        {
            GameObject UI = GameObject.Find("GameUI").transform.Find("PlayerKey").gameObject;
            for(int i = 0; i < 2; i++)
            {
                GameObject Button = UI.transform.GetChild(i + 4).gameObject;
                if(Button.activeSelf)
                {
                    m_Change = Button.GetComponent<ChangeButton>();
                    break;
                }
            }
        }

        m_Change.Change(iarr[iCurList], iCurList, m_PlayerManager.GetPlayerData(PLAYER_DATA.PLAYER_CUR_HP),
m_PlayerManager.GetPlayerData(PLAYER_DATA.PLAYER_MAX_HP), m_PlayerManager.GetPlayerData(PLAYER_DATA.PLAYER_CUR_SP),
m_PlayerManager.GetPlayerData(PLAYER_DATA.PLAYER_MAX_SP));
        //캐릭터 버튼 체인지 및 쿨타임 실행

        m_PlayerManager.PlayerSet(m_iTmpIndex, tr.position, tr.rotation, JumpEnd);
        //캐릭터의 위치와 교대하고
        m_EnemyMangaer.TrSetting(m_PlayerManager.GetCharTR());
        //에너미의 타겟을 새로 설정하고
        m_EnemyMangaer.Start();
        //코루틴 스타트
        m_CallBack(m_PlayerManager.GetCharTR());
        //카메라 콜백 함수 재선언
        m_bChanging = true;
        m_Change = null;
    }
    else
    {
        PlayerDie();
    }
```

Source

Motive : 붕괴 3rd

Resource

- GameScene 3D Model - FantasyScene.asset
- LobbyScene 3D Model - https://www.models-resource.com/ Honkai Impact Model
- Character 3D Model - UnityChan, ThePhantomKnowledge
- Character Animation - FightingUnityChan_FreeAsset.asset, RPG Character Animation Pack FREE.asset
- Character Effect - Realistic Effects Pack 4.asset
- UI -자체 제작, NGUI, HUD&GUI MEDIEVAL ART BUNDLE.asset

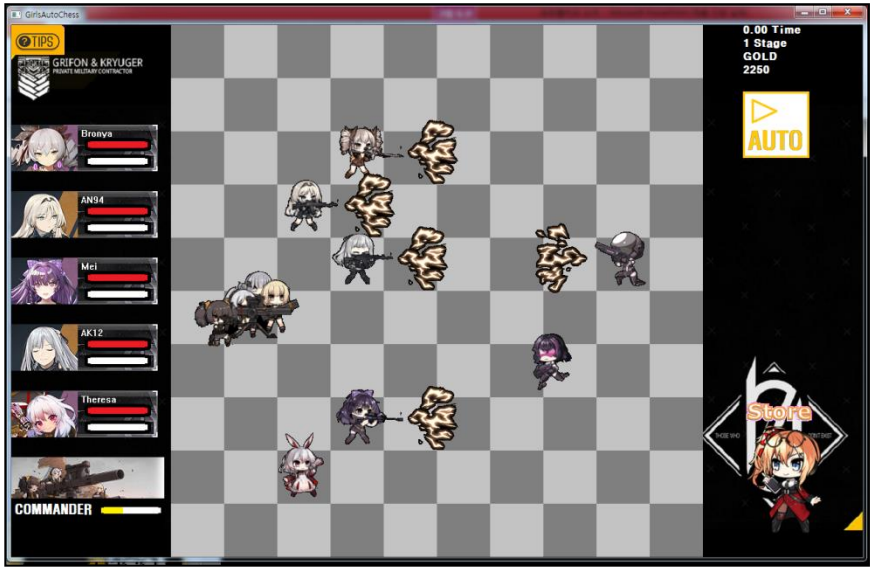
Index

2D Portfolio

https://youtu.be/mViastQSY8M

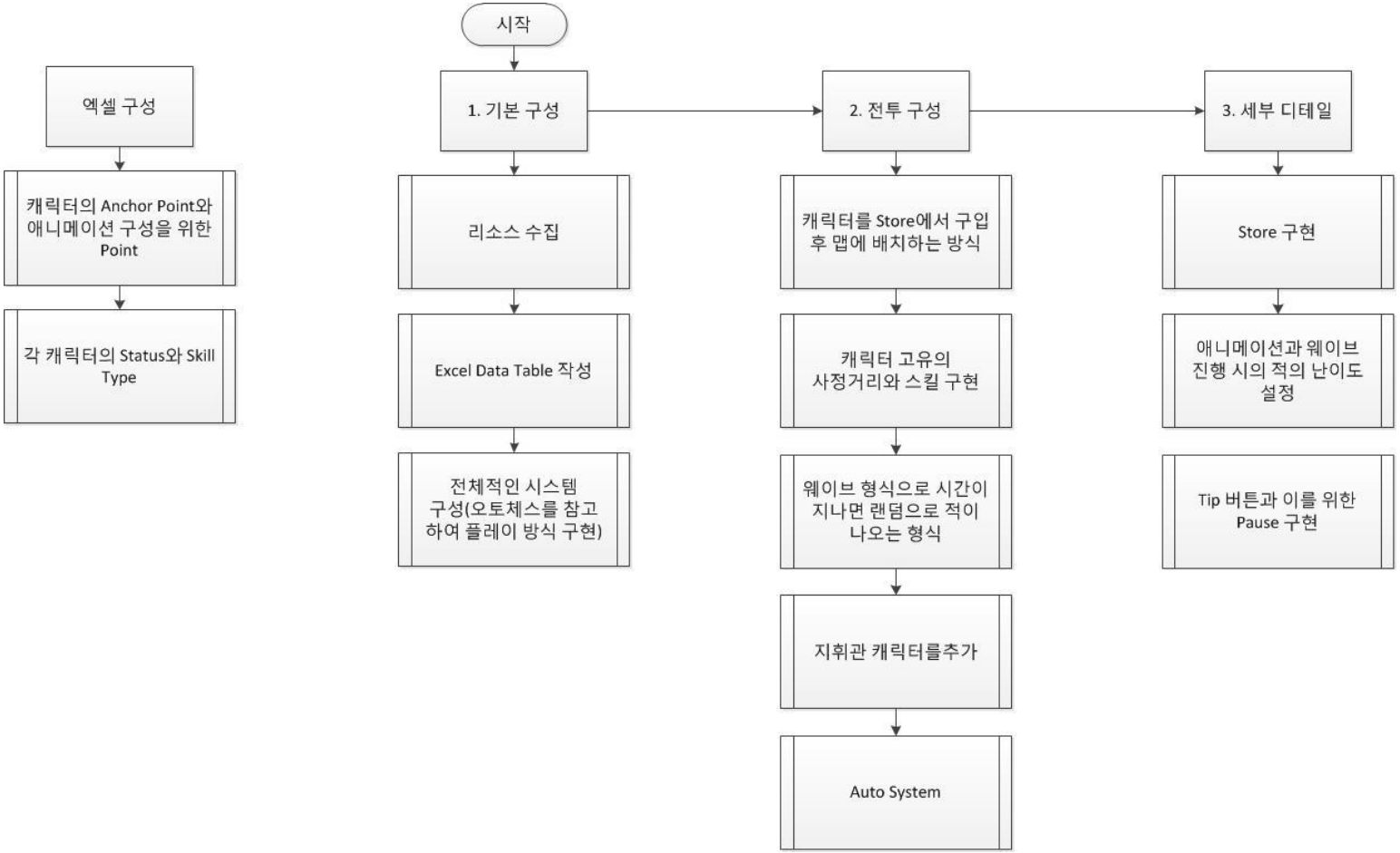
Introduction

총 제작 기간 : 2019/05/13 ~ 2019/06/04(24일)



'소녀전선'의 캐릭터와 'AutoChess'의 게임 방식을 결합하여 WinAPI로 제작하였다.

Workflow


시작
엑셀 구성
캐릭터의 Anchor Point와 애니메이션 구성을 위한 Point
각 캐릭터의 Status와 Skill Type
1. 기본 구성
리소스 수집
Excel Data Table 작성
전체적인 시스템 구성(오토체스를 참고하여 플레이 방식 구현)
2. 전투 구성
캐릭터를 Store에서 구입 후 맵에 배치하는 방식
캐릭터 고유의 사정거리와 스킬 구현
웨이브 형식으로 시간이 지나면 랜덤으로 적이 나오는 형식
지휘관 캐릭터를추가
Auto System
3. 세부 디테일
Store 구현
애니메이션과 웨이브 진행 시의 적의 난이도 설정
Tip 버튼과 이를 위한 Pause 구현

Source

Motive : 소녀전선, AutoChess

Resource

- Character 2D Model - Girls’ Frontline
- UI - 자체 제작, Girls’ Frontline