Bet With Friends

Manual

작성일 : 2023년 1월 04일

작성자 : 김승준

목차

서론

나는 친구와 내기를 정말 좋아한다. 네이버 사다리타기, 카카오 사다리 타기, 네이버 돌림판 등 다양한 게임으로 내기를 했는데, 내기 할 수 있는 게임이 한 곳에 있지 않고 하나하나 검색해 찾아서 사용하는게 불편했다. 또한 게임의 종류도 많이 없어서 아쉬웠다. 그래서 다양한 내기 게임을 즐길 수 있는 사용이 간편한 앱을 만들게 되었다.

이 앱은 오프라인 모드와 온라인 모드가 있다. 아무래도 이런 종류의 게임은 친구와 직접 만나 사용할 때가 많기 때문에 오프라인 모드에 치중을 두었고, 온라인 모드는 파이어 베이스 Realtime Database를 사용하여 멀리서도 친구와 내기 할 수 있게 하였다.

***이 문서는 내가 나중에 프로젝트를 수정할 일이 있을 때 사용하거나, 다른사람이 볼 일이 있을 때 이해를 돕기 위해 만들었다. 이 매뉴얼로만은 이해하기 힘들 수 있으므로 스크립트들의 주석과 이 매뉴얼을 같이 보길...**

***이 프로젝트의 소스 이미지들은 픽사베이 무료이미지를 사용하거나, 피스켈(스프라이트 제작 사이트)에서 제작하였다.**

***스크립트들은 하나 하나 직접 만들었다. 기본적인 메소드 사용법을 제외하고는 개인적인 성장을 위하여 다른사람이 만든 코드는 보지않았다.(온라인 가위바위보 게임 만들 때 제외하고는...) 스크립트 보면서 보안할 점 있으면tmdwns711@naver.com 으로 부탁...**

Scene 구성

위 그림과 같이 Home아래 오프라인-온라인 씬이 있다. 오프라인 씬 아래에는 4개의 씬이 있고, 온라인 씬 아래에는 하나의 씬이 존재한다.

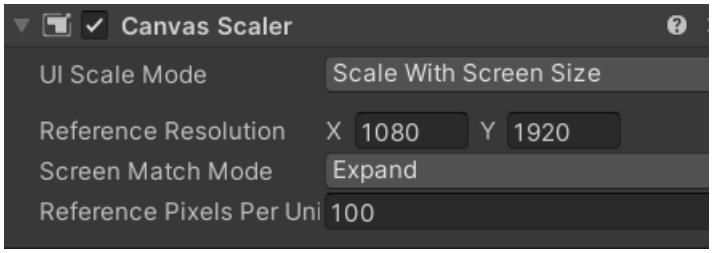
먼저 오프라인 씬에 대해 간략하게 소개하자면 Ladder은 사다리 게임, RPS는 가위바위보 게임, WheelGame은 돌림판 게임, Gacha는 가챠게임이다.

오프라인과 온라인에는 RPS게임이 있는데 오프라인 씬의 RPS게임은 최대 4명까지 이용가능하고 일등부터 꼴등까지 나눌 수 있도록 게임을 구성하였다. 온라인 씬의 게임은 두명이서 이용가능하고 파이어베이스 Realtime Database를 이용했다. 온라인모드는 한명이 방 코드와 사용할 닉네임을 입력하면 방이 만들어지고, 다른 한명이 방코드와 사용 할 닉네임을 입력한 후 접속하여 진행된다.

각 Scene 설명

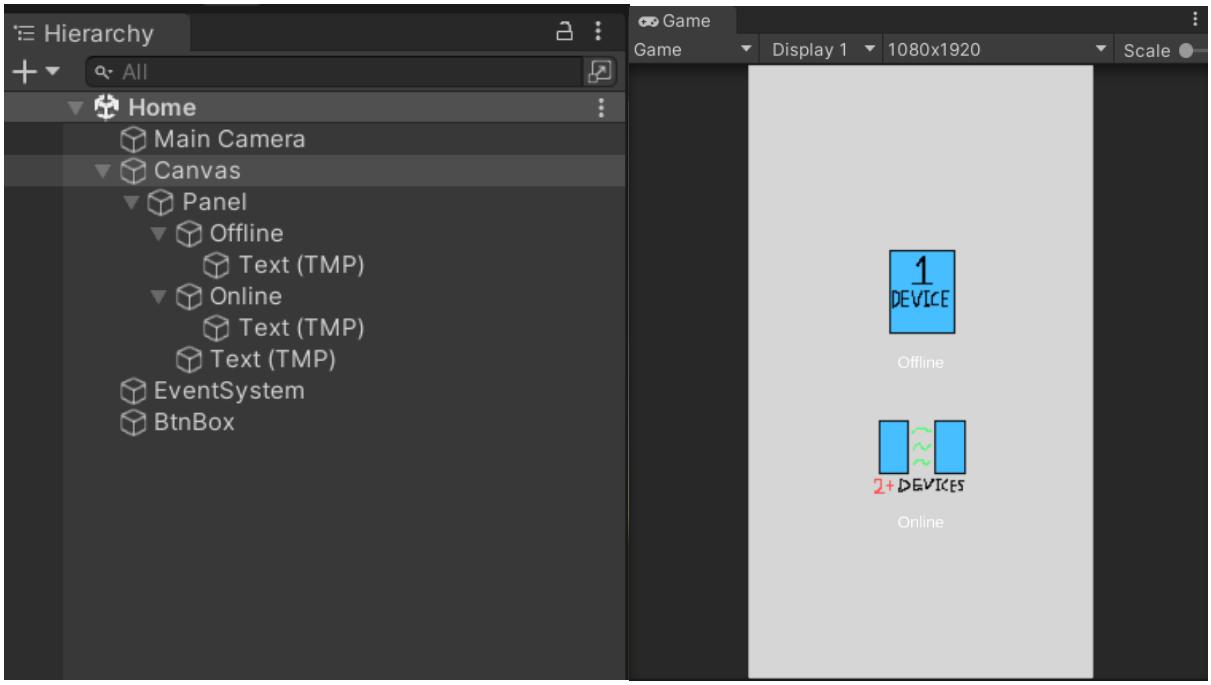
각 Scene들의 핵심적인 부분 및 쉽게 이해하기 힘든 부분만 설명한다.

씬들은 공통적으로 Canvas Scaler가 다음과 같이 설정되었다.



참조 크기인 (1080,1920)을 기준으로 화면의 크기가 달라진다해도 안 보이거나 겹치는 UI들이 없게 하기 위해서 이다.

1. Home Scene



Home씬은 간단하게 2개의 버튼으로 구성된다. 온라인 버튼을 누르면 Home_Offline 씬으로 이동하고, 오프라인 버튼을 누르면 Home_Offline 씬으로 이동한다. 온라인 씬은 인터넷 연결이 접속되어 있을 때만 사용 가능 하도록 하였다. (이미지는 피스켈이라는 스프라이트 제작 사이트에서 만들었다.)

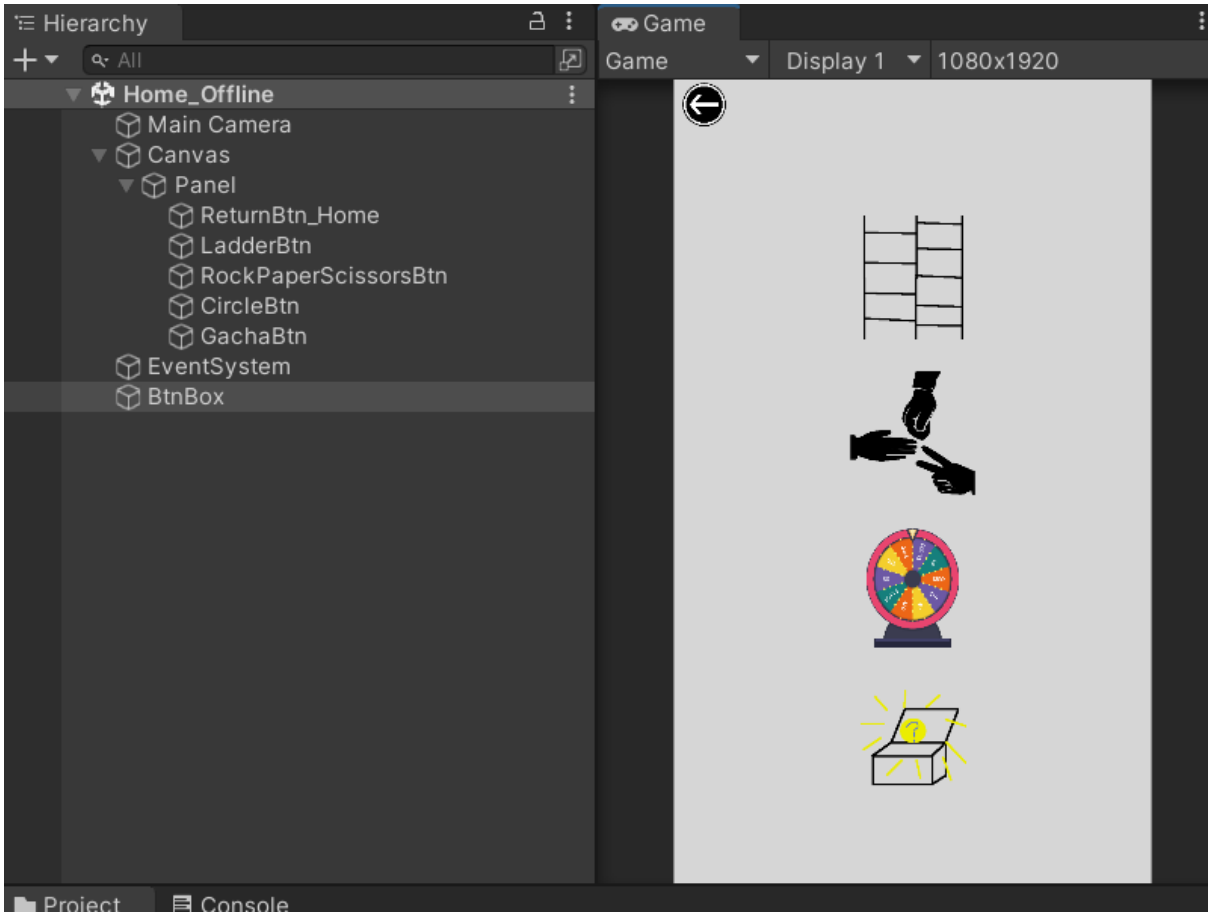
```
using UnityEngine;
using UnityEngine.SceneManagement;
using TMPro;

public class OnOffMoveScene : MonoBehaviour
{
    public TMP_Text onOffText;

    public void OfflineScene()
    {
        SceneManager.LoadScene("Home_Offline");
    }
    public void OnlineScene()
    {
        if (Application.internetReachability == NetworkReachability.NotReachable)
            onOffText.text = "Check network!";
        else
            SceneManager.LoadScene("Home_Online");
    }
}
```

https://docs.unity3d.com/ScriptReference/NetworkReachability.ReachableViaLocalAreaNetwork.html

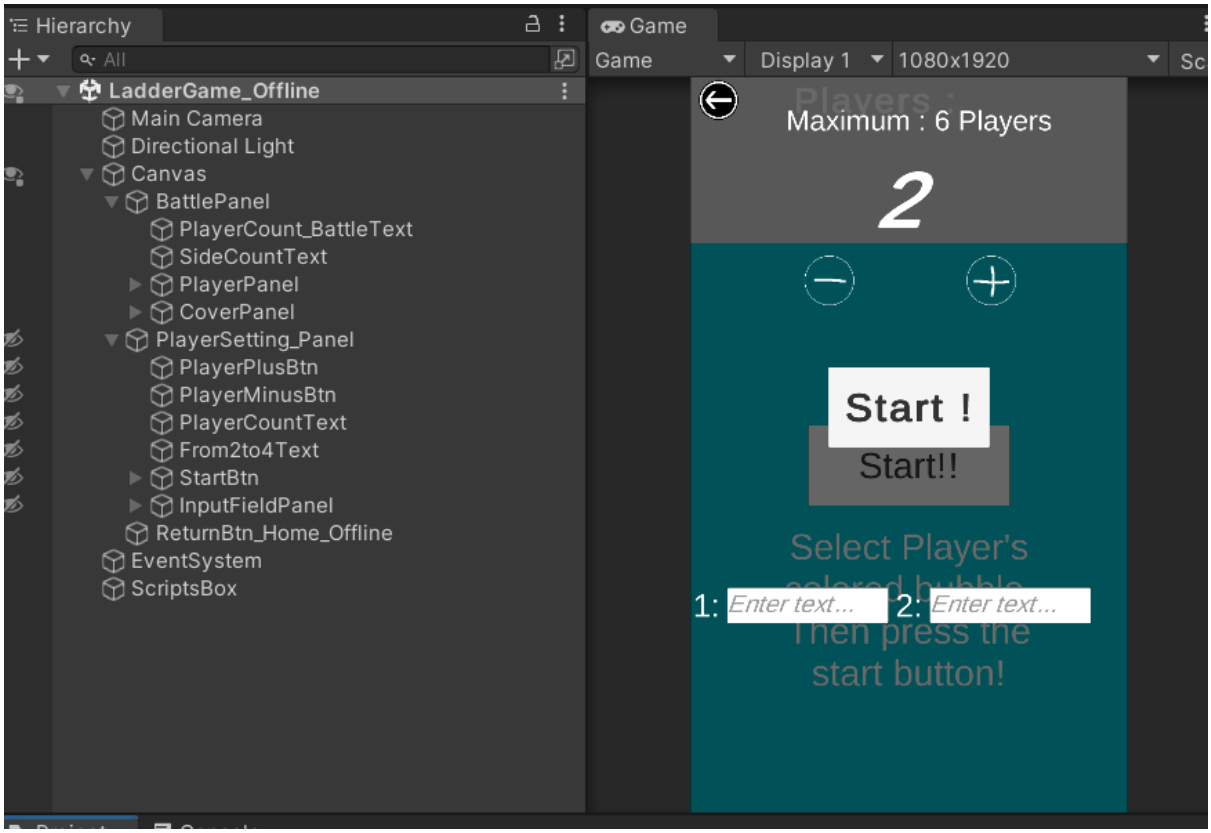
2. Home_Offline Scene

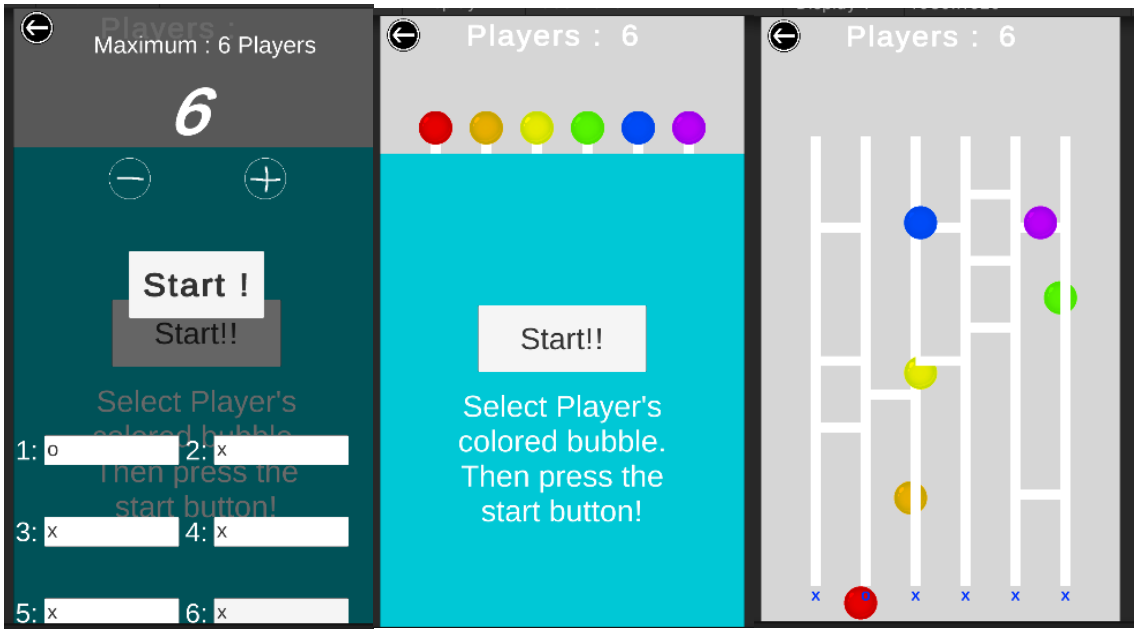


Home 오프라인 씬 또한 단순하게 버튼 5개로 제작하였다. 각 게임 씬으로 이동하는 4개의 버튼과 뒤로가기 버튼이 존재한다. 사용된 스크립트는 다음과 같다.

```
using UnityEngine;
using UnityEngine.SceneManagement;
public class OfflineSceneBtnScripts : MonoBehaviour
{
    public void RPS_SceneEnter()
    {
        SceneManager.LoadScene("RPS_Offline");
    }
    public void Home_SceneReturn()
    {
        SceneManager.LoadScene("Home");
    }
    public void Gacha_SceneEnter()
    {
        SceneManager.LoadScene("Gacha_Offline");
    }
    public void LadderGame_SceneEnter()
    {
        SceneManager.LoadScene("LadderGame_Offline");
    }
    public void WheelGame_SceneEnter()
    {
        SceneManager.LoadScene("WheelGame_Offline");
    }
}
```

3. Ladder Game Scene





사용된 스크립트 (6개)

BattlePanel_StartBtn -부착된 오브젝트 경로 : ScriptsBox

BridgeActive -부착된 오브젝트 경로 : Canvas/BattlePanel/PlayerPanel/1~6/Bridges

PlayerMove_LadderGame -부착된 오브젝트 경로 : Canvas/BattlePanel/PlayerPanel/1~6/Player

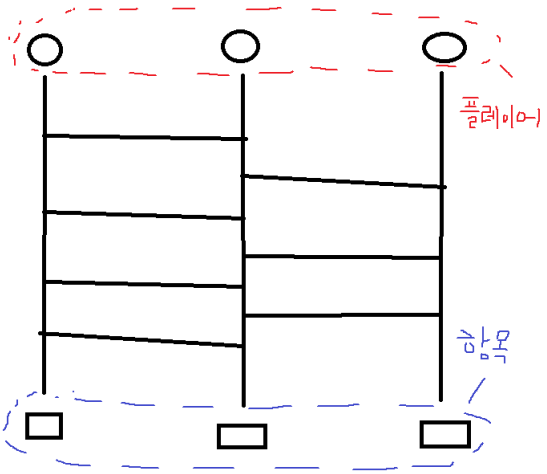
PlayerSelectBtn -부착된 오브젝트 경로 : Canvas/BattlePanel/PlayerPanel/1~6/Player/Button

PlayerSetting_LadderOffline -부착된 오브젝트 경로 : ScriptsBox

ReturnHomeOfflineScene -부착된 오브젝트 경로 : ScriptsBox

2명에서 6명까지 이용 가능 하도록 만들었다. 크게 세팅 패널과 배틀 패널로 나누었다.

세팅 패널



대략적인 과정은 플러스버튼이나 마이너스 버튼으로 플레이어 숫자를 정하고, 이와 같은 개수의 활성화 된 인풋필드에 항목을 입력한 후 스타트 버튼을 누르면 해당 사람 수에 맞게 사다리가 생성된다. 사다리의 가로방향의 다리들 양쪽 끝에는 콜라이더를 넣고 플레이어가 닿을시 이동 방향이 꺾이게 만들었다.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using Random = UnityEngine.Random; //System과 UnityEngine 중에 무슨 Random 함수를 사용 할지 정해줌.

public class PlayerSetting_LadderOffline : MonoBehaviour
{
    public TMP_Text playerCountText;
    public TMP_Text playerCountText_Battle_Panel; //배틀패널 상단의 플레이어 수
    public GameObject settingPanel; // 스타트 버튼 누를시 세팅패널이 사라지게 하기 위해서
    public Transform inputFieldPanel; //자식오브젝트를 찾고 활성 및 비활성 하기 위해서
    public GameObject[] players = new GameObject[6]; //플레이어 수에 맞게 배틀필드의 플레이어 오브젝트들을 활성화 시키기 위하여
    public TMP_Text[] selectedOptionsText = new TMP_Text[6]; // 플레이어 패널에 있는 각 플레이어의 text.

    private List<int> randomList = new List<int>(); //인풋필드에 입력한 내용을 사다리의 랜덤한 위치에 배정하기 위해서

    public void PlayerPlusBtn() //플러스 버튼
    {
        if (Convert.ToInt32(playerCountText.text) > 1 && Convert.ToInt32(playerCountText.text) < 6)
        {
            playerCountText.text = Convert.ToString(Convert.ToInt32(playerCountText.text) + 1);
            inputFieldPanel.Find(playerCountText.text).gameObject.SetActive(true);
        }   //총 6명까지.인풋필드패널의 자식을 찾아 활성화 시킨다.gameObject 는 참조되는 대상의 자기자신을 가리킴.
    }
    public void PlayerMinusBtn() //마이너스 버튼
    {
        if (Convert.ToInt32(playerCountText.text) > 2 && Convert.ToInt32(playerCountText.text) < 7)
        {
            inputFieldPanel.Find(playerCountText.text).gameObject.SetActive(false);
            playerCountText.text = Convert.ToString(Convert.ToInt32(playerCountText.text) - 1);
        }
    }
```

플러스 버튼과 마이너스 버튼 누르면 텍스트 값을 증가 또는 감소시키고, 그 값을 이용하여 인풋필드 패널의 자식 오브젝트를 찾아 활성화 시킨다.(Find 메소드는 메모리를 많이 잡아 먹는 것으로 알고 있으나 업데이트 함수 내에 사용한 것이 아닐 뿐더러, 나중에 프리팹을 복제하는 방식으로 사용할 수도 있을 것 같아 다음과 같이 작성했다.)

```
public void GameStartBtn()
{
    playerCountText_Battle_Panel.text = playerCountText.text; //활성화 전에 미리 넣어야 브릿지 활성화 할때
    RandomListSelect();                                       //꼬이지 않는다.
    settingPanel.SetActive(false);
}
void RandomListSelect()//입력된 값들을 랜덤한 위치에 배정하기
{
    int convertPlayerCount = Convert.ToInt32(playerCountText.text);
    int random_int = 0;
    for (int i = 0; i < convertPlayerCount; i++)//리스트에 0에서 플레이어 수 - 1 만큼 값을 할당한다.
    {
        randomList.Add(i);
        players[i].SetActive(true);
    }
    for (int j = 0; j < Convert.ToInt32(playerCountText.text); j++) // 리스트에서 랜덤한 인덱스의 값을 뽑고 그 값을 제거한다.
    {
        random_int = Random.Range(0, convertPlayerCount);
        selectedOptionsText[randomList[random_int]].text = inputFieldPanel.Find((j+1).ToString()).gameObject.GetComponent<TMP_InputField>().text;
        randomList.RemoveAt(random_int); // 해당 인덱스 값 제거
        convertPlayerCount--;
    }
}
```

사다리 타기는 공평해 보이지만 플레이어 위치(사다리 상단)와 항목(사다리 하단)의 위치가 멀수록 덜 걸린다는 유튜브 영상을 본 적이 있다. 그래서 공평한 게임을 위해 입력한 항목들을 랜덤한 위치에 배정해주는 기능이 필요 했다.

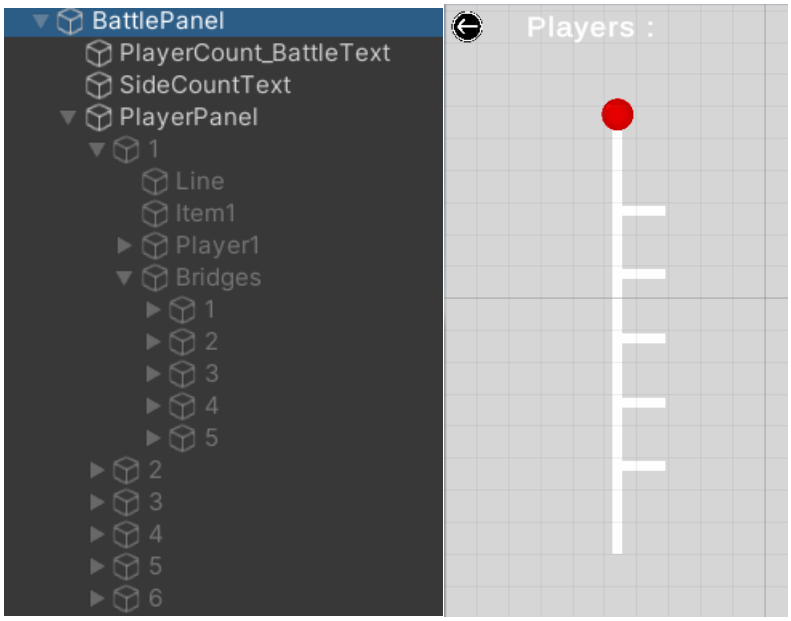
예를 들어, 3명의 플레이어가 게임을 한다고 했을 때 리스트에는 {0,1,2} 값을 넣고 0~2사이의 랜덤한 정수를 뽑는다(그것이 1이라고 가정하면). 리스트[랜덤숫자] 이므로 randomlist[1] 값인 1이 나오게 된다. 이 값을 맨 위에서 선언해 놨던 항목오브젝트의 인덱스 값으로 넣는다. 내부적인 과정은 아래와 같다.

`selectedOptionsText[randomList[random_int]] -> selectedOptionsText[randomList[1]] -> selectedOptionsText[ 1 ]`

이렇게 선택된 항목오브젝트에 인풋필드의 첫 번째 값을 배정한다. 이렇게 된다면 첫 번째 인풋필드 텍스트 값이 사다리의 2번 항목에 들어가게 된다.

그리고 리스트에서 방금 뽑은 1 값을 제거해준다. 그러면 리스트에는 {0, 2} 만 남게 된다. for문을 통해 사람 수(=항목 수) 만큼 반복하게 되는데 반복할 때 마다 랜덤 정수 값의 범위를 1씩 줄이므로 중복없고 누락없이 인풋필드 텍스트 값들이 랜덤한 항목 위치에 전부 배정 된다.

배틀패널




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

public class BridgeActive : MonoBehaviour
{
    public TMP_Text playerCount_BattleText; // 마지막 플레이어의 브릿지는 나타내지 않기 위해서
    public GameObject[] bridge = new GameObject[5];
    void Start()
    {
        if(transform.parent.name != playerCount_BattleText.text) //만약 같다면 마지막플레이어 임을 알 수 있다.
        {
            int randomCount = 0;
            for (int i = 0; i < 5; i++)
            {
                randomCount = Random.Range(0, 2); //0, 1 포함
                if (randomCount == 0)
                    bridge[i].SetActive(true);
            }
        }
    }
}
```

세팅 패널에서 설정된 플레이어 수 만큼 플레이어 오브젝트를 활성화 시켜주었는데, 그것이 PlayerPanel의 1,2, ... 오브젝트이다. 이 프리팹들의 자식에는 브릿지(사다리의 가로방향 다리부분)라는 오브젝트가 있는데 위와 같이 랜덤하게 활성화 시켜준다. 마지막 플레이어는 브릿지를 생성하지 않는다. 이 스크립트는 브릿지 오브젝트에 붙여준다.(플레이어 오브젝트가 활성화되면 그 때 작동)

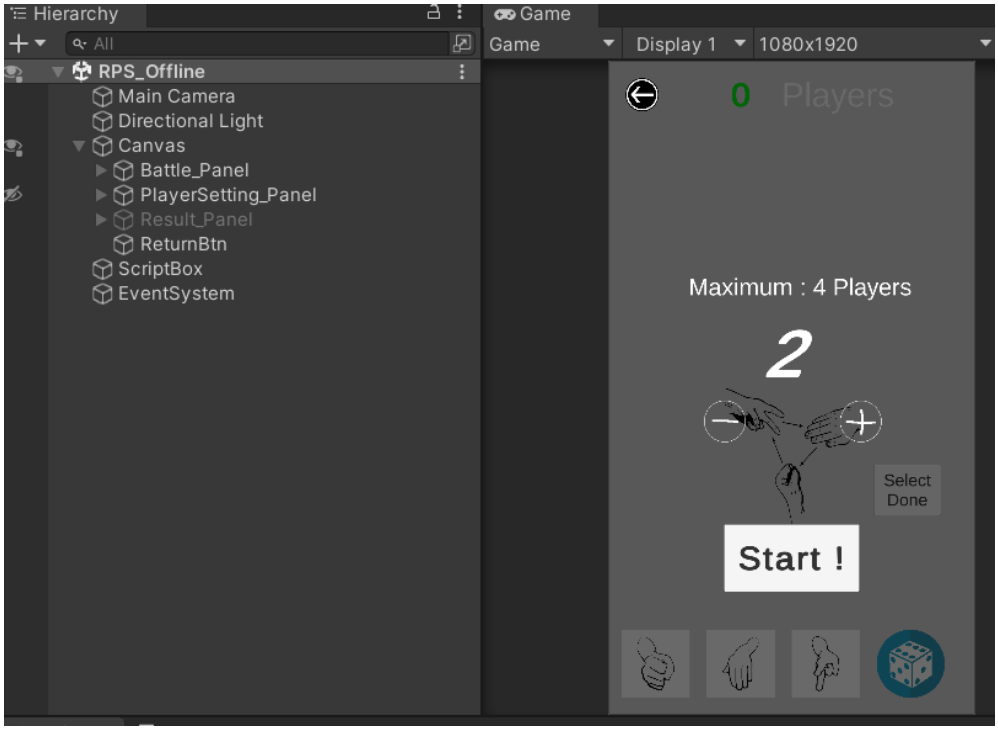
* 발생한 문제

- Screen.SetResolution(1080, 1920, false) 은 화면비율을 고정시켜주지만 ui가 찌그러져 보이는 문제 때문에 제거함.

-공이 움직이기 시작하고 브릿지에 닿았을 때 움직이는 방향을 바꾸어 주었는데 trigger 이벤트를 통해 감지했다. 처음에는 업데이트 내에 Move 스크립트를 넣어주었지만 trigger이벤트 자체가 fixed 업데이트 에서 구동되기 때문에 공이 브릿지에 닿더라도 방향전환이 잘 되지 않았다. 그래서 Move 메소드를 fixed 업데이트 안에 넣어주니 훨씬 나아졌다.

```
void FixedUpdate()
    {
        if(meBtn.activeSelf == false)//IsDisabled
        {
            transform.position  +=  new Vector3(directionX * speed, directionY * speed, 0f);
        }
    }
```

4. RPS_Offline Scene(가위바위보 오프라인 게임 씬)



사용된 스크립트 (6개)

(RockPaperScissors) 네임스페이스 존재. RPS_Online 폴더에 있음.

-부착된 오브젝트 경로 : ScriptBox

PlayerCountScript -부착된 오브젝트 경로 : ScriptBox

PlayerCreatScript -부착된 오브젝트 경로 : Canvas/BattlePanel/PlayerSelectPanel

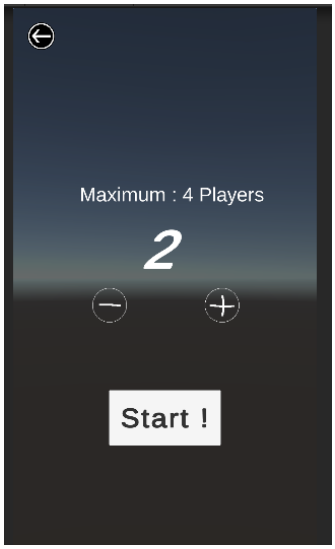
PlayerTextChange -부착된 오브젝트 경로 : Player 프리팹/Text

Return_Home_Offline -부착된 오브젝트 경로 : ScriptBox

Visual_ImageChangeScript -부착된 오브젝트 경로 : Canvas/BattlePanel/Visual_RPS

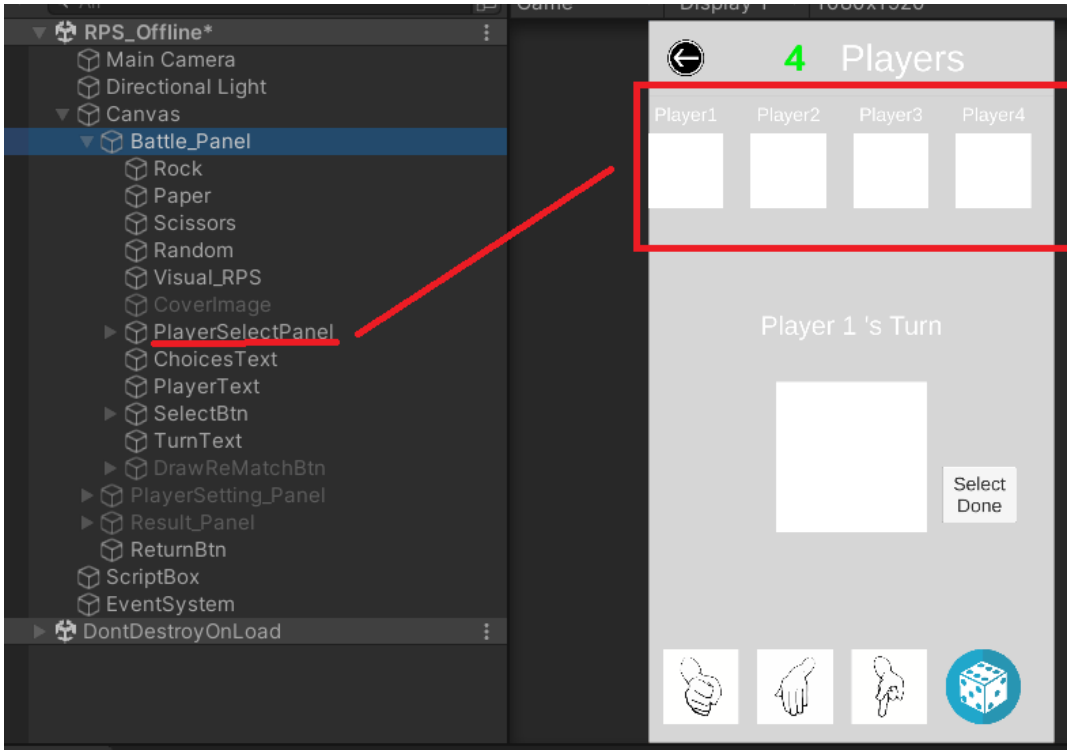
크게 3개의 패널로 구성되어 있다. 게임이 이루어지는 battlePanle, 시작 전 플레이어 수를 정하는 SettingPanel, 게임 후 결과가 나오는 ResultPanel 이다.

Setting Panel



세팅 패널 부분은 다른 씬들과 유사하다. 스타트 버튼을 누르면 플레이어 수가 배틀패널 상단의 텍스트에 들어가고, 배틀패널 상단의 각 플레이어가 고른 손모양 이미지가 나오는 PlayerSelectPanel가 활성화 된다.

Battle Panel



세팅 패널에서 스타트 버튼을 누르면 PlayerSelectPanel이 활성화 된다. 이 오브젝트에는 각 플레이어가 고른 이미지가 나오는 곳으로 마지막 플레이어 선택 후 무슨 손모양을 골랐는지 나오게 된다. 수평방향으로 정렬하기 위해 Horizontal layout group 을 넣었다.

대략적인 작동 방식은 세팅패널에서 플레이어 수를 선택하고, 순서대로 원하는 손모양을 누른 후 Select done버튼을 누른다. 마지막 플레이어 순서가 끝나고, 전부 같은모양이나 3가지 손모양이 모두 나온다면 비긴 것으로 간주하고 패널의 왼편에 리매치 버튼을 활성화한다. 승자 패자가 나뉜다면 결과 창과 함께 승자 매치 버튼(승자 2명 이상)과, 패자 매치(패자 2명이상) 버튼이 활성화 된다. 이것으로 일등부터 꼴등까지 가릴 수 있게 된다.

아래 코드는 Select Done 버튼의 클릭 이벤트 이다.

```
public void SelectDoneChoice() //여기서 대부분의 기능들이 수행된다.
{
    if (btnCount < count)
    {
        playerValue[btnCount] = RockPaperScissorsScripts.choiceRPS;
        Debug.Log("value :" + playerValue[btnCount]);
        btnCount += 1;
        coverImage.SetActive(true);
        if (btnCount == count)
        {
            playerOrderText.text = "";
            ResultBattle();
            for (int i = 0; i < count; i++) //마지막 참여자 순서까지 마치면 이미지를 넣는다.
            {
                if (playerValue[i] == 1)
                    this.transform.Find("Player" + (i + 1)).GetComponent<Image>().sprite = rockImage;
                else if (playerValue[i] == 2)
                    this.transform.Find("Player" + (i + 1)).GetComponent<Image>().sprite = paperImage;
                else
                    this.transform.Find("Player" + (i + 1)).GetComponent<Image>().sprite = scissorsImage;
            }
        }
        else
            playerOrderText.text = "Player " + (btnCount + 1).ToString() + " 's Turn";
    }
    else
        playerOrderText.text = "";
}
```

각 플레이어가 고른 모양 값을 playerValue배열에 넣고 버튼 카운트를 1씩 높여준다. 버튼 카운트가 count(플레이어 수) 값보다 작을 때만 if문이 실행 되게 하였고, 마지막 유저가 버튼을 눌렀을 때 결과 값을 나오게 하기 위해 if(btnCount == count) 문을 만들고 각 플레이어에 해당되는 이미지 오브젝트에 스프라이트를 넣어준다.

코드 중앙 부분의 ResultBattle()은 결과 값을 계산해 주는 메소드로 먼저 비겼는지 판단하고(비기면 리매치버튼 활성화) 비기지 않았다면 승자, 패자를 나누고 결과창을 띄운다.

자세한 것은 스크립트에 주석을 많이 달아 두었다.

```
public void ResultBattle()
{
    if(IsDraw(playerValue) == true)
    {
        playerOrderText.text = "Draw!";
        drawReMatchBtn.SetActive(true);
    }
    else
    {
        WhoIsWinner(playerValue);
        ResultPanel.SetActive(true);
    }
}
```

IsDraw(int[] arr) 메소드는 비길경우(모든 손모양이 다 나오거나, 모두 같은 손모양을 냈을 때) true를 return하고 아니면 false를 return한다. 만약 false라면 WhoIsWinner() 메소드를 실행한다.

플레이어가 낸 손모양 value를 중복을 제거하고 리스트에 넣는다.(주먹-보, 주먹-가위, 보-가위 : 3개의 경우의 수) 그리고 리스트를 오름차순 정렬 해주는데, 이유는 리스트라는게 들어있는 값이 같더라도 인덱스 값이 다르다면 서로 다른 리스트라고 판단하기 때문에, 오름차순 정렬 함으로써

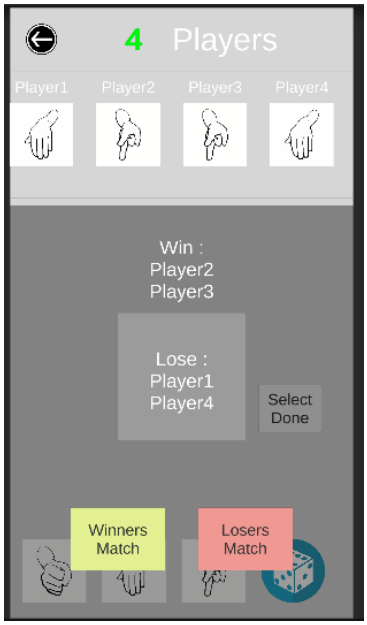
```
if(kinds_List[0] == 1 && kinds_List[1] == 2) // 주먹, 보
else if (kinds_List[0] == 1 && kinds_List[1] == 3) // 주먹, 가위

else //(kinds_List[0] == 2 && kinds_List[1] == 3)  보, 가위
```

위와 같은 3개의 경우의 수로 압축 할 수 있다.(예를 들어 플레이어가 낸 손모양 값에 따라 리스트가 {3,1}이 될 수도 있는데 이 리스트의 목적이 플레이어들이 어떤 손모양 종류를 냈는지만을 확인하기 위함이므로 오름차순 정렬 함으로써 위와 같은 3가지의 경우의 수로 압축. 앞에서 비기는 경우를 제외 했으므로 나올 수 있는 가짓 수는 2종류 뿐이다.)

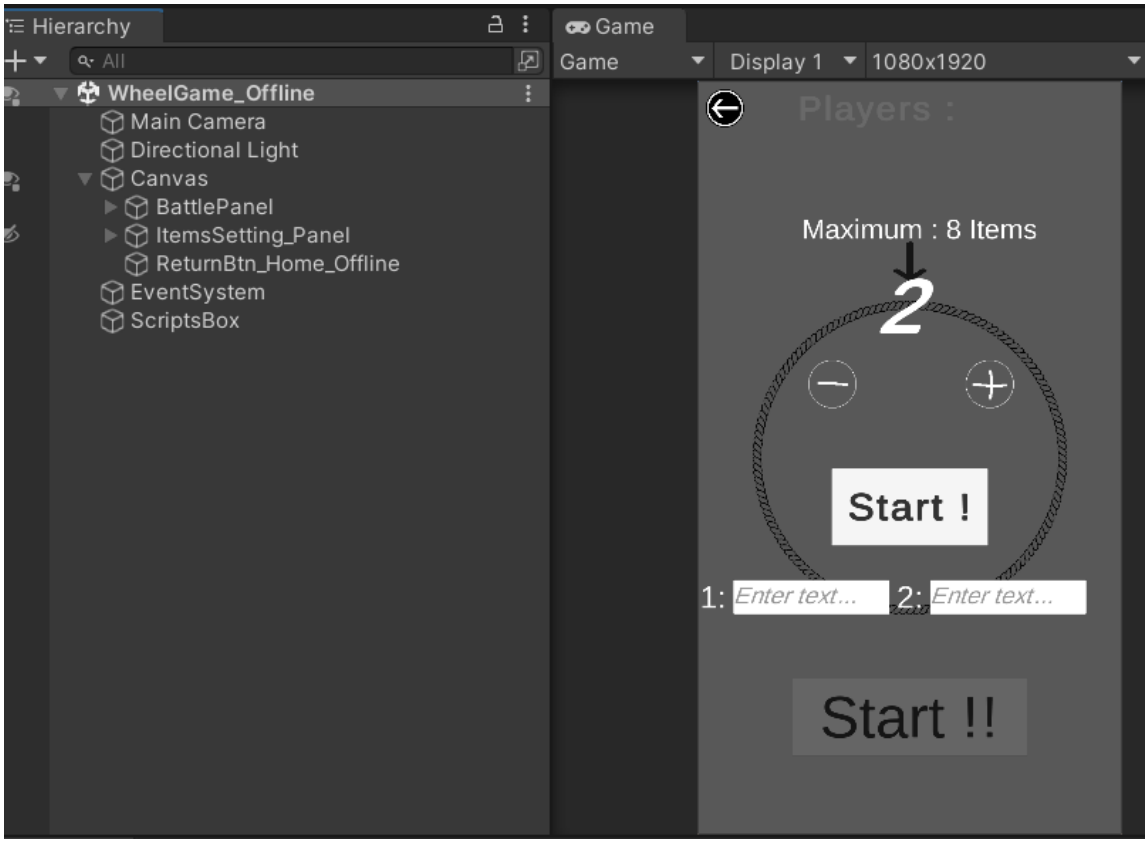
참고로 위너 매치와 루저 매치 둘다 활성화 되는 경우는 4명의 플레이어가 이긴사람 2명, 진사람 2명이 나온 경우이다.

Result Panel



*설명은 위에서 했으므로 생략

5. Wheel Game Offline Scene(돌림판 게임)



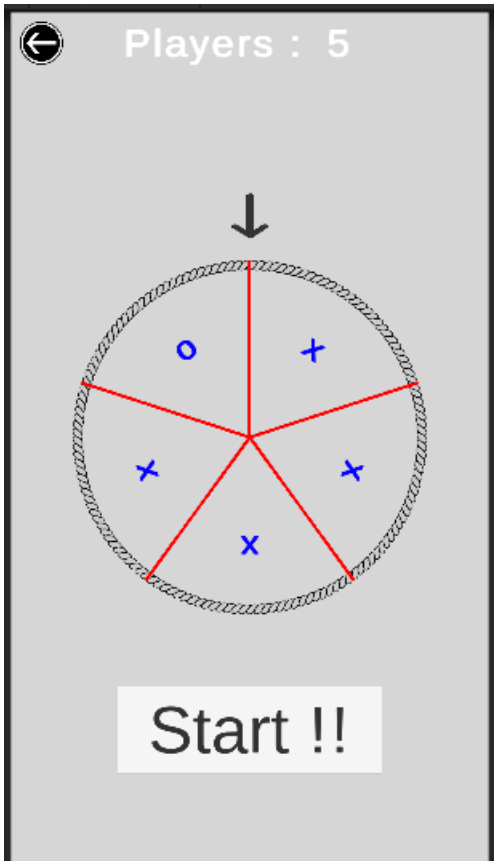
사용된 스크립트 (3개)

BattleStart -부착된 오브젝트 경로 : ScriptsBox

FromWheelToHomeOffline -부착된 오브젝트 경로 : ScriptsBox

ItemSetting_WheelOffline -부착된 오브젝트 경로 : ScriptsBox

이 씬의 설명은 스크립트보다는 UGUI 부분에 중점을 두고 설명하겠다. 크게 배틀패널과 세팅패널로 이루어져 있고 세팅패널은 다른 씬들의 세팅 패널 부분과 유사하게 제작하였다.



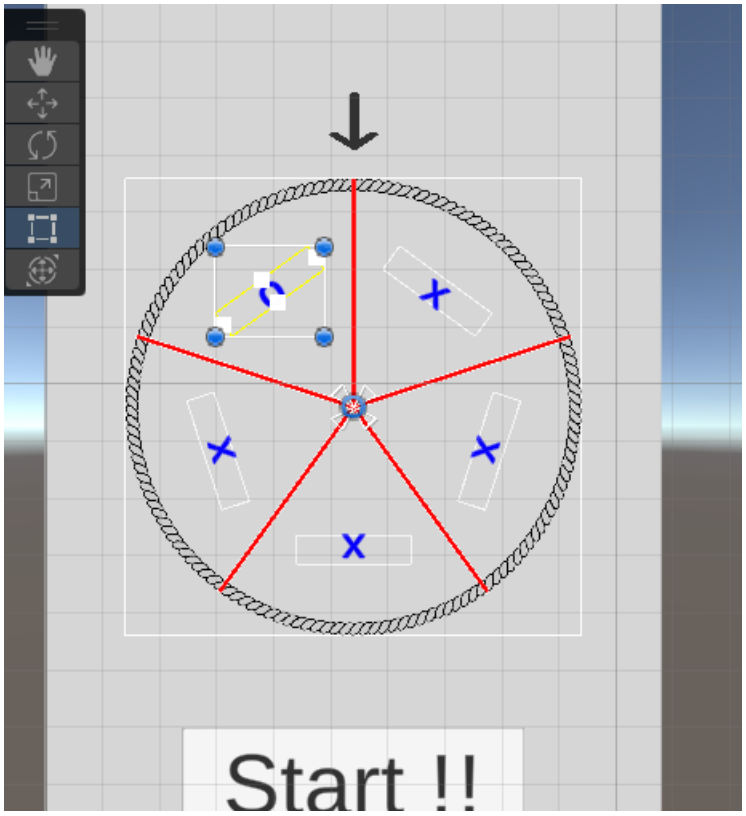
세팅 패널에서 설정한 항목 수 값을 가져와 빨간색 선(Image ui)을 항목 수 만큼 생성했다.

사람수에 맞게 돌림판 영역을 생성하는 것이 중요했는데 인스펙터 창의 Pivot 기능이 핵심이다.

Pivot은 해당 ui의 기준점의 위치좌표값을 설정할 수 있는 기능이다.

UI는 왼쪽하단이 (0, 0) 오른쪽 상단이 (1, 1) 이다.

기본적으로 (0.5, 0.5) 값을 가지고 있는데 이것은 ui의 중앙이 기준점이 된다. 위 사진처럼 항목수 만큼 원형으로 Instantiate 해주기 위해서는 기준점이 빨간색 선 아래 부분에 와야한다.(중앙에 기준점이 있으면 원판의 중앙에 빨간색선의 중앙 부분이 올 것이다) 그래서 Pivot 을 (0.5, 0) 으로 바꿔주었다.



세팅 패널에서 인풋필드에 텍스트를 입력하면 위 사진과 같이 복제된 텍스트 오브젝트에 text값이 들어 간다. 여기서 중요한 점은 복제된 텍스트박스의 기준점이 원판의 원점에 와야한다. 여기서 팁은 맨 처음 프리팹을 생성할 때 원하는 위치에 텍스트박스를 놓고 포지션 값이 0이 될 때까지 pivot값을 조정하는 것이다.(오브젝트의 너비와 높이에 따라 pivot에 들어가는 값이 다름)

여기서는 텍스트 박스의 기준점이 이 박스를 벗어나야 하므로 (0.5, -4.5)로 두었다.

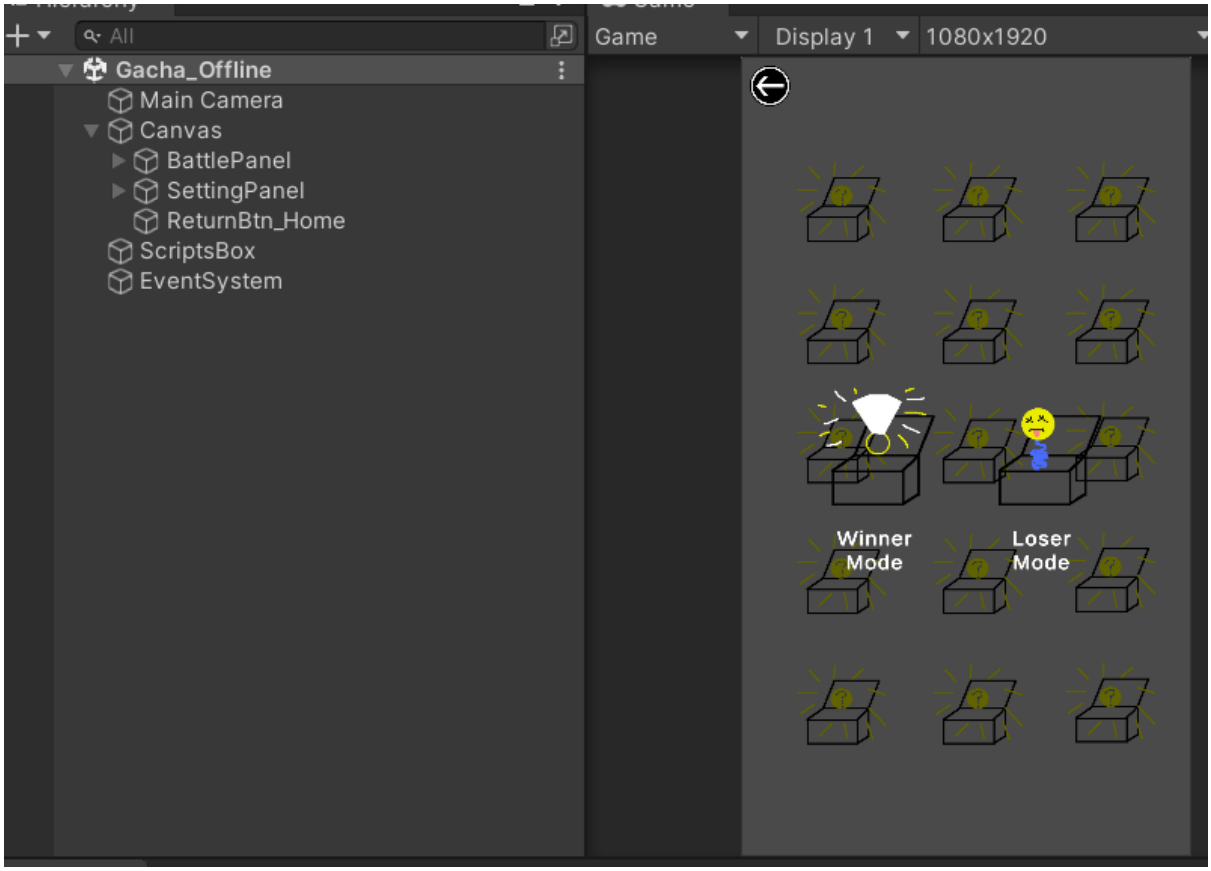
이렇게 생성된 원판은 start버튼을 누르면 돌아간다.

```
public GameObject startButton;
public Transform roulette_Image;
private float speed; //회전속도
private float timer;
private float stopSpeed;
private int startTrigger; //1이면 업데이트문 실행
void Start()
{
    startTrigger = 0;
    timer = Random.Range(3f, 4f);
    speed = Random.Range(20f, 25f);
    stopSpeed = Random.Range(6f, 8f);
}
void Update()
{
   if(startTrigger == 1)
    {
        timer -= Time.deltaTime;
        roulette_Image.Rotate(0f, 0f, 1f * speed);
        if (timer <= 2)
        {
            speed -= Time.deltaTime * stopSpeed;
            if(speed <= 0)
                startTrigger = 0;
        }

    }
}
public void battleStartBtn()
{
    startTrigger = 1;
    startButton.SetActive(false);
}
```

돌아가는 시간, 스피드, 멈추는 속도를 랜덤으로 만들어 원판이 멈추는 위치가 랜덤으로 나오게 하였다. 랜덤한 값을 넣지 않으면 같은 위치가 계속 나오는 일이 생긴다. 랜덤 값을 많이 넣을 수록 경우에 수는 더욱 많아 지므로 더욱 공평한 게임이 된다.

6. Gacha Offline Scene(가챠 게임)



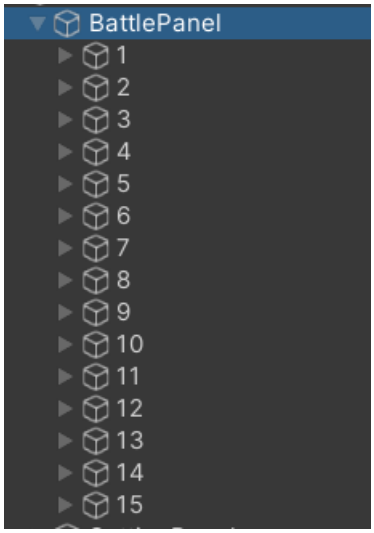
사용된 스크립트 (3개)

Gacha_Offline_Battle -부착된 오브젝트 경로 : ScriptsBox

GachaImageChange -부착된 오브젝트 경로 : Canvas/BattlePanel/1~15/GachaBtn

HomeOfflineReturn -부착된 오브젝트 경로 : ScriptsBox

세팅 패널에서는 위너 모드(보석반지 찾기)와 루저모드(꽝 찾기)를 선택 할 수 있다. 15개의 상자 중에 하나만 들어 있다.



시작 전 1~15 중 랜덤한 숫자를 뽑고 상자를 클릭시 클릭된 이미지(이미지 안에 버튼있음)의 이름과 뽑아놓은 랜덤한 숫자가 같으면 해당모드에 맞는 이미지가 나오는 방식이다.

7. RPS_Online Scene(가위바위보 온라인 배틀)

1대1 모드 밖에 없으며 아직 완벽하진 않은 Scene이다. 앞에서 말했다 싶이 Firebase Realtime Database를 이용했으며 게임을 플레이 하는데 있어서는 문제는 없다. 다만 사용된 방식이 최적화된 방식은 아닌게 분명했고, 앞으로도 계속 보완할 예정이다.

기본적인 원리는 만들기 버튼을 눌러 방코드와 사용될 닉네임을 입력하면 데이터베이스에 저장되고, 조인버튼을 눌러 방코드와 사용할 닉네임을 입력하면 미리 만들어진 방코드에 조인 유저의 닉네임이 저장되는 방식이다. 그리고 방에 두 명이 존재 할 때 방장에게 스타트 버튼이 활성화되고 그것을 누르면 데이터 베이스의 방 코드 아래에 스타트가 추가되고, 로컬 타이머가 작동된다. 조인 유저는 코루틴으로 0.5초에 한번씩 데이터를 읽고 있는데, 스타트 데이터를 읽으면 게임이 시작된다. 호스트와 조인유저는 0.5초정도 딜레이가 있고, 타이머가 다 되면 서로가 낸 손모양 데이터 값이 입력된다. 이 때 서로 간 딜레이가 있으므로 두 사람의 데이터가 다 입력되어야만 게임이 종료된다.

* 데이터 베이스에 쓰레기가 생성이 된다. 호스트가 게임을 나갈 때 방이 같이 삭제되도록 하였는데 앱을 강제종료하는 경우에는 정상적으로 삭제되지가 않는다. 그래서 PlayerPrefs로 방이 만들어 졌을 때 방코드와 닉네임을 로컬에 저장한 후, 후에 앱에 다시 접속했을 때 해당 코드와 닉네임이 들어있는 방이 존재 할 시 삭제 되도록 하였다.

참고문서 : https://firebase.google.com/docs/unity/setup

https://firebase.google.com/docs/database/unity/save-data

https://firebase.google.com/docs/database/unity/retrieve-data