EGOISM

게임 소개

- 제목 : EGOISM
- 개발인원 : 4명
- 장르 : 추리게임
- 개발기간 : 2022.04~2023.03
- 개발환경 : Unity,Visual Studio

Localization.cs

```
const string langURL = "
public event System.Action LocalizeChanged = () => { };
public event System.Action LocalizeSettingChanged = () => { };


public int curLangIndex;    // 현재 언어의 인덱스
public List<Lang> Langs;    // 언어 데이터 클래스의 리스트


// InitLang 함수에서는 저장해놓은 언어 인덱스값이 있다면 가져오고 , 없다면 기본언어(영어)의 인덱스 값을 가져온다.
참조 1개
void InitLang()
{
    int langIndex = PlayerPrefs.GetInt("LangIndex", -1);
    int systemIndex = Langs.FindIndex(x => x.lang.ToLower() == Application.systemLanguage.ToString().ToLower());
    if (systemIndex == -1) systemIndex = 0;
    int index = langIndex == -1 ? 0 : langIndex;
    SetLangIndex(index);
}
//선택한 언어로 변경
참조 2개
public void SetLangIndex(int index)
{
    curLangIndex = index;
    PlayerPrefs.SetInt("LangIndex", curLangIndex);
    LocalizeChanged();
    LocalizeSettingChanged();
}
```

```
IEnumerator GetLangCo()
{
    UnityWebRequest www = UnityWebRequest.Get(langURL);
    yield return www.SendWebRequest();
    SetLangList(www.downloadHandler.text);  //스프레드 시트의 데이터 값을 SetLangList에 넣어준다.
}
참조 1개
void SetLangList(string tsv)
{

    string[] row = tsv.Split('\n'); //스페이스를 기준을 행 분류
    int rowSize = row.Length;
    int columnSize = row[0].Split('\t').Length; //탭을 기준으로 열 분류
    string[,] Sentence = new string[rowSize, columnSize];

    for (int i = 0; i < rowSize; i++)
    {
        string[] column = row[i].Split('\t');
        for (int j = 0; j < columnSize; j++)
            Sentence[i, j] = column[j];
    }

    Langs = new List<Lang>();

    for (int i = 0; i < columnSize; i++)
    {
        Lang lang = new Lang();
        lang.lang = Sentence[0, i];
        lang.langLocalize = Sentence[1, i];

        for (int j = 2; j < rowSize; j++) lang.value.Add(Sentence[j, i]);
        Langs.Add(lang);
    }
    check = true;
}
```

구글시트링크를 csv파일로 변경하여 실시간으로 나라별 언어로 변경

API

참조 4개
```
public class UserData //api로 받을 유저의 데이터를 클래스화 해서 선언함
{
    public string uid;
    public string email;
    public double exp;
    public int gold;
    public int level;
    public int cardPiece;
    public string nickName;
    public int normalItem;
    public int luxuryItem;

    public List<CardList> cardList;
    public List<CostumeList> costumeList;
    public List<string> humanList, hunterList;

    public int currentCustomNum;
}
```

(예시)Api로 userdata 받는 부분

```
IEnumerator PostLoadUserDataCour(UnityAction p_successCallback, UnityAction p_failCallback) //api로 유저의 데이터를 받는 코루틴이며 결과에 따라 UnityAction 처리
{
    UnityWebRequest request = new UnityWebRequest();

    WWWForm form = new WWWForm();
    form.AddField("uid", GameManager.gInstance.userId);

    using (request = UnityWebRequest.Post(TARGET_DOMAIN + "member", form)) //post방식으로 api 호출
    {
        yield return request.SendWebRequest();

        if (request.isNetworkError || request.isHttpError) //연결 에러 처리
        {
            Debug.LogError(request.responseCode + " / " + request.error + "/" + request.downloadHandler.text);
            _ErrorResponseCode = request.responseCode.ToString();
            if (JsonMapper.ToObject(request.downloadHandler.text).ToString().Contains("error"))
            {
                _ErrorMessage = JsonMapper.ToObject(request.downloadHandler.text)["error"].ToString();
            }

            if (p_failCallback != null)
                p_failCallback();
        }
        else
        {
            Debug.Log(request.downloadHandler.text);

            _userData = JsonMapper.ToObject<UserData>(request.downloadHandler.text); //클래스화한 UserData에 받은 유저 정보 넣음

            if (p_successCallback != null)
                p_successCallback();
        }

        request.Dispose();
    }
}
```

Api를 호출하여 유저의 데이터를 받거나 저장하는 용도로 사용함.

firebase realtime Database 접근

```
public void ReadUserData()
{
    FirebaseDatabase.DefaultInstance.GetReference("") //firebase realtime Database 접근
        .GetValueAsync().ContinueWithOnMainThread(task =>
        {
            if (task.IsFaulted)
            {
                // Handle the error...
            }
            else if (task.IsCompleted)
            {
                DataSnapshot snapshot = task.Result;
                // Do something with snapshot...
                for (int i = 0; i < snapshot.ChildrenCount; i++)
                {
                    Debug.Log(snapshot.Child(i.ToString()).Child("username").Value);
                }
                //SystemInfo.deviceUniqueIdentifier
                string s = snapshot.Child(SystemInfo.deviceUniqueIdentifier).Value.ToString(); //본인의 기기 고유정보로 본인임을 구별

                Debug.Log(s);

                if (s == "user") //해당 고유정보내의 값에 uid가 있는지 판별하는 재귀함수
                {
                    Invoke("ReadUserData", 0.5f);
                }
                else
                {
                    GameManager.gInstance.userId = s;
                    LoginStateCheck();
                }
            }
        });
}
```

Google 로그인시 api를 통해 웹에서 구글 로그인 창이 열리고 웹에서 로그인을 성공했을시엔 firebase realtime Database에 해당 유저의 uid가 저장됨. 해당 firebase realtime Database를 유니티에서 연동하여 접근 후 해당 유저의 uid를 가져오기 위해 사용함.

자동 로그인

```
if (!PlayerPrefs.HasKey("UserID")) //유저가 로그인했다면 해당 uid 저장
{
    PlayerPrefs.SetString("UserID", userId);
    Debug.Log(userId);
}
else
{
    userId = PlayerPrefs.GetString("UserID");
    Debug.Log(userId);
}

if (!PlayerPrefs.HasKey("LoginState")) //유저가 로그인 했를 때 구글, 게스트 등 어떤 방식으로 로그인 했는지 저장
{
    PlayerPrefs.SetInt("LoginState", 0);
    LoginState = PlayerPrefs.GetInt("LoginState");
    Login = (State)LoginState;
}
else
{
    LoginState = PlayerPrefs.GetInt("LoginState");
    Login = (State)LoginState;
}
```

```
public void CustumToken() //저장된 uid로 토큰 생성 후 firebase 인증
{
    auth.SignInWithCustomTokenAsync(APIManager._AInstance._AccessToken).ContinueWith(task => {
        if (task.IsCanceled)
        {
            Debug.LogError("SignInWithCustomTokenAsync was canceled.");
            return;
        }
        if (task.IsFaulted)
        {
            Debug.LogError("SignInWithCustomTokenAsync encountered an error: " + task.Exception);
            return;
        }

        Firebase.Auth.FirebaseUser newUser = task.Result;
        Debug.LogFormat("User signed in successfully: {0} ({1})",
            newUser.DisplayName, newUser.UserId);
    });

    LoginStateCheck(); //인증 완료 이후 유저 데이터를 받아 로그인 성공
}
```

유저가 로그인시 PlayerPrefs로 uid와 로그인 방식을 저장한 후 해당 기기에서 게임을 다시 실행시켰을 경우 저장되어 있던 uid와 로그인 방식을 통해 API를 통해 firebase에 인증할 토큰 생성 후 firebase에 인증하여 자동 로그인이 되도록 설정.

사운드 설정

```
//BGM과 Effect의 value를 저장하여 다시 접속해도 해당 수치가 남아있도록 설정
if (!PlayerPrefs.HasKey("BGM"))
{
    BGMSlider.value = 0;
    masterMixer.SetFloat("BGM", 0);
}
else
{
    BGMSlider.value = PlayerPrefs.GetFloat("BGM");
    masterMixer.SetFloat("BGM", PlayerPrefs.GetFloat("BGM"));
}

if (!PlayerPrefs.HasKey("Effect"))
{
    EffectSlider.value = 0;
    masterMixer.SetFloat("Effect", 0);
}
else
{
    EffectSlider.value = PlayerPrefs.GetFloat("Effect");
    masterMixer.SetFloat("Effect", PlayerPrefs.GetFloat("Effect"));
}
```

```
//슬라이드 값에 따라 사운드 변화 적용
참조 4개
public void AudioSlideControl(bool b, Slider s, Button btn, Sprite[] i)
{                           //BGM, Effect구별
    float sound;
    string audioname;
    if (b)
    {
        sound = s.value;
        audioname = "BGM";
    }
    else
    {
        sound = s.value;
        audioname = "Effect";
    }

    if (sound == -40f) //소리 최소화시 음소거 표현
    {
        btn.GetComponent<Image>().sprite = i[1];
        masterMixer.SetFloat(audioname, -80);
        PlayerPrefs.SetFloat(audioname, -80);

    }
    else
    {
        btn.GetComponent<Image>().sprite = i[0];
        masterMixer.SetFloat(audioname, sound);
        PlayerPrefs.SetFloat(audioname, sound);
    }
}
```

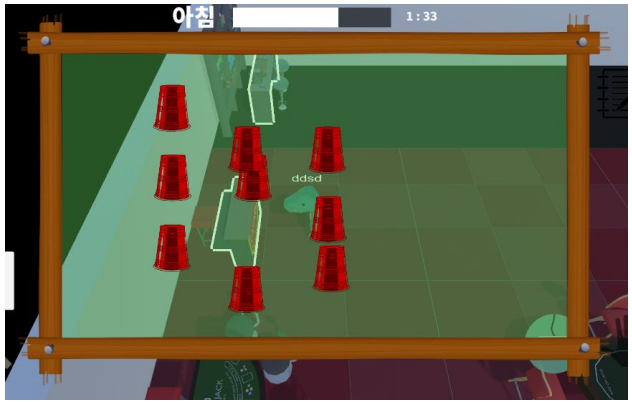
```
//음소거 설정, 해제 버튼 클릭 효과
참조 4개
public void AudioBtnClick(bool b,Slider s,Button btn, Sprite[] i)
{                           //BGM, Effect구별
    string audioname;
    if (b)
    {
        audioname = "BGM";
        if (s.value != -40f)
        {
            btn.GetComponent<Image>().sprite = i[1];
            masterMixer.SetFloat(audioname, -80);
            PlayerPrefs.SetFloat(audioname, -80);
            s.value = -40f;
        }
        else
        {
            btn.GetComponent<Image>().sprite = i[0];
            masterMixer.SetFloat(audioname, -20);
            PlayerPrefs.SetFloat(audioname, -20);
            s.value = -20f;
        }
    }
    else
    {
        audioname = "Effect";
        if (s.value != -40f)
        {
            btn.GetComponent<Image>().sprite = i[1];
            masterMixer.SetFloat(audioname, -80);
            PlayerPrefs.SetFloat(audioname, -80);
            s.value = -40f;
        }
        else
        {
            btn.GetComponent<Image>().sprite = i[0];
            masterMixer.SetFloat(audioname, -20);
            PlayerPrefs.SetFloat(audioname, -20);
            s.value = -20f;
        }
    }
}
```

Unity Audio Mixer를 사용하여 BGM, Effect로 그룹을 나누고 PlayerPrefs로 저장하여 해당 기기에서 세팅한 사운드 값은 유지되도록 설정함. 사운드 수치는 설정창에서 슬라이더로 조정하며 음소거 설정, 해제 버튼에 대한 내용까지 적용함.

SellGameMgr.cs

```
참조 1개
public Vector3 BezierCurve(Vector3 p1_, Vector3 p2_, Vector3 p3_, Vector3 p4_, float Value)
{
    Vector3 A = Vector3.Lerp(p1_, p2_, Value);
    Vector3 B = Vector3.Lerp(p2_, p3_, Value);
    Vector3 C = Vector3.Lerp(p3_, p4_, Value);

    Vector3 D = Vector3.Lerp(A, B, Value);
    Vector3 E = Vector3.Lerp(B, C, Value);

    Vector3 F = Vector3.Lerp(D, E, Value);

    return F;
}
```

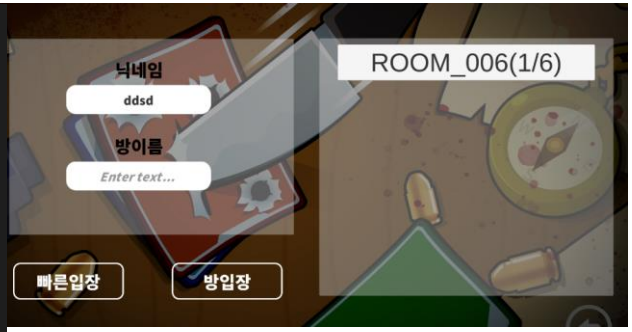
2차 베지어곡선을 사용하여 4개의 좌표값을 받아와 컵이 자연스럽게 서로 섞는 듯한 연출을 표현함.



PhotonRoomMgr.cs

```
public override void OnRoomListUpdate(List<RoomInfo> roomList)
{
    base.OnRoomListUpdate(roomList);
    GameObject tempRoom = null;

    foreach (var room in roomList)
    {
        if (room.RemovedFromList == true)
        {

            roomDict.TryGetValue(room.Name, out tempRoom);
            Destroy(tempRoom);
            roomDict.Remove(room.Name);
        }
        else
        {
            if (roomDict.ContainsKey(room.Name) == false)
            {
                Debug.Log("ASD");
                GameObject _room = Instantiate(roomPrefab,scrollContent);

                _room.GetComponent<RoomData>().RoomInfo = room;
                roomDict.Add(room.Name, _room);

            }
            else
            {

                roomDict.TryGetValue(room.Name, out tempRoom);
                tempRoom.GetComponent<RoomData>().RoomInfo = room;
            }
        }
    }
}
```

```
public void OnRandomBtn()
{
    if(string.IsNullOrEmpty(userIdText.text))
    {
        userId = $"USER_{Random.Range(0, 100):00}";
        userIdText.text = userId;
    }

    PlayerPrefs.SetString("USER_ID", userIdText.text);
    PhotonNetwork.NickName = userIdText.text;
    PhotonNetwork.JoinRandomRoom();
}

참조 0개
public void OnMakeRoomClick()
{
    loadingImg.gameObject.SetActive(true);
    RoomOptions ro = new RoomOptions();
    ro.IsOpen = true;
    ro.IsVisible = true;
    ro.MaxPlayers = 6;
    PhotonNetwork.NickName = userIdText.text;

    if (string.IsNullOrEmpty(roomNameText.text))
    {
        roomNameText.text = $"ROOM_{Random.Range(1, 100):000}";
    }
    PhotonNetwork.CreateRoom(roomNameText.text, ro);

}
```

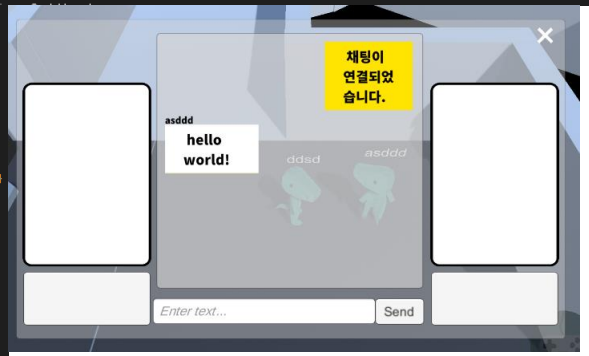


유저가 빠른입장 클릭시 방이없으면 방을 만들고 있으면 오른쪽 방리스트중 랜덤한 방에 입장,
방입장시 유저의 방 생성

PhotonChatMgr.cs

```
public IEnumerator CompareChat()
{
    bool doingPrivateChat = this.chatClient.PrivateChannels.ContainsKey(this.selectedChannelName);
    if (chatPanel.activeSelf == false && doingPrivateChat)
    {
        for (int i = 0; i < content.transform.childCount; i++)
        {
            Destroy(content.transform.GetChild(i).gameObject);
        }
        StopAllCoroutines();
    }
    string[] temp = chatDisplay.text.Split(Environment.NewLine.ToCharArray(), StringSplitOptions.RemoveEmptyEntries)
    //열었을때 기존 채팅한번 받아옴

    for (int i = content.transform.childCount; i < temp.Length; i++)
    {

        string[] contents = temp[i].Split(':');
        if (contents[0] == username) isSend = true;
        else isSend = false;
        Area yellow = Instantiate(isSend ? YellowArea : WhiteArea).GetComponent<Area>();
        if (isSend == false)
        {
            yellow.userText.GetComponent<TMP_Text>().text = contents[0];
        }
        yellow.transform.SetParent(content.transform, false);
        yellow.BoxRect.sizeDelta = new Vector2(100, yellow.BoxRect.sizeDelta.y);
        yellow.TextRect.GetComponent<TMP_Text>().text = contents[1];
        Fit(yellow.BoxRect);
        float X = yellow.TextRect.sizeDelta.x + 42;
        float Y = yellow.TextRect.sizeDelta.y;
        if (Y > 49)
        {
            for (int j = 0; j < 200; j++)
            {
                yellow.BoxRect.sizeDelta = new Vector2(X - j + 2, yellow.BoxRect.sizeDelta.y);
                Fit(yellow.BoxRect);

                if (Y != yellow.TextRect.sizeDelta.y) { yellow.BoxRect.sizeDelta = new Vector2(X - (j + 2) + 2, Y); break; }
            }
        }
        else yellow.BoxRect.sizeDelta = new Vector2(X, Y);

        Fit(yellow.BoxRect);
        Fit(yellow.AreaRect);
        LastArea = yellow;
    }
    Invoke("ScrollDelay", 0.03f);
    yield return new WaitForSeconds(0.3f);
    StartCoroutine(CompareChat());

}
```

```
public void Chat(string target)
{
    isConnected = true;
    this.chatClient.SendPrivateMessage(target, "채팅이 연결되었습니다.");
    bool doingPrivateChat = this.chatClient.PrivateChannels.ContainsKey(this.selectedChannelName);
    if (doingPrivateChat)
    {
        this.chatClient.PrivateChannels.Remove(this.selectedChannelName);
        chatDisplay.text = "";
    }
    else
    {
        ChatChannel channel;
        if (this.chatClient.TryGetChannel(this.selectedChannelName, doingPrivateChat, out channel))
        {
            channel.ClearMessages();
        }
    }
}
```

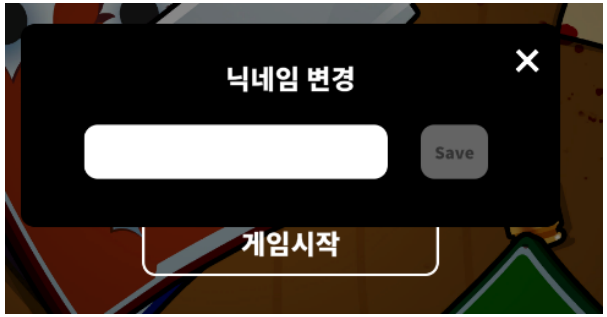


유저간 채팅버튼 클릭시 채팅창이 활성화가 되며 기존에 채팅을 받아오며 Send버튼을 누르면 서로간의 채팅이 가능하다.

NicknameChange

```
public void PostNickNameChange(string s_NickName , UnityAction p_successCallback, UnityAction p_failCallback)
{
    StartCoroutine(PostNickNameChangeCour(s_NickName, p_successCallback, p_failCallback));
}

참조 1개
IEnumerator PostNickNameChangeCour(string s_NickName, UnityAction p_successCallback, UnityAction p_failCallback)
{
    UnityWebRequest request = new UnityWebRequest();

    WWWForm form = new WWWForm();
    form.AddField("uid", GameManager.gInstance.userId);
    form.AddField("nickName", s_NickName);

    using (request = UnityWebRequest.Post(TARGET_DOMAIN + "member/nickNameChange", form))
    {
        yield return request.SendWebRequest();

        if (request.isNetworkError || request.isHttpError)
        {
            Debug.LogError(request.responseCode + " / " + request.error + "/" + request.downloadHandler.text);
            _ErrorResponseCode = request.responseCode.ToString();

            if ((request.downloadHandler.text).ToString().Contains("message"))
            {
                _ErrorMessage = JsonMapper.ToObject(request.downloadHandler.text)["message"].ToString();
            }

            if (p_failCallback != null)
                p_failCallback();
        }
        else
        {
            Debug.Log(request.downloadHandler.text);
            _userData.nickName = s_NickName;
            if (p_successCallback != null)
                p_successCallback();
        }

        request.Dispose();
    }
}
```

```
참조 0개
public void CheckNickName()
{
    APIManager._AInstance.PostNickNameChange(CheckNickName_IF.text, NickNameSuccessCallBack, null);
}
```



유저의 uid와 변경할 Nickname을 API를 통해 Nickname을 변경할수있다. 게임을 다시 실행했을때 변경한 Nickname으로 표시가 된다.

MyCostume

```
참조 0개
public void CostumeSelect()
{
    GameObject g = EventSystem.current.currentSelectedGameObject.transform.GetChild(0).gameObject;
    for (int i = 0; i < costumeList.Count; i++)
    {
        if (g.GetComponent<Image>().sprite == costumeList[i])
        {
            selectBorder.transform.parent = g.transform;
            selectBorder.transform.localPosition = new Vector3(0,0, 0);
            costumeNumber = i;
            GameManager.gInstance.costumeNum = costumeNumber+1;
            APIManager._AInstance._userData.currentCustomNum = costumeNumber+1;
            GameManager.gInstance.myCostume = GameManager.gInstance.costumePrefab[i];
            APIManager._AInstance.PostCurCostume(null, null);
        }
    }
}
참조 0개
public void MyCostume(GameObject g)
{
    selectBorder.transform.parent =
        g.transform.GetChild(APIManager._AInstance._userData.currentCustomNum-1).transform;
    selectBorder.transform.localPosition = new Vector3(0, 0, 0);
    GameManager.gInstance.costumeNum = APIManager._AInstance._userData.currentCustomNum;

    for (int i=0; i< APIManager._AInstance._userData.costumeList.Count; i++)
    {
        for (int j=1; j<=costumeList.Count; j++)
        {
            if (APIManager._AInstance._userData.costumeList[i].number == j)
            {
                g.transform.GetChild(i).gameObject.SetActive(true);
                g.transform.GetChild(i).transform.GetChild(0).GetComponent<Image>().sprite =
                    costumeList[j-1];
            }
        }
    }
}
참조 0개
```

```
IEnumerator PostCurCostumeCour(UnityAction p_successCallback, UnityAction p_failCallback)
{
    SelectCostume curNum = new SelectCostume();
    curNum.uid = GameManager.gInstance.userId;
    curNum.currentCustomNum = GameManager.gInstance.costumeNum;
    JsonData json = JsonMapper.ToJson(curNum);
    Debug.Log(json);

    UnityWebRequest request = new UnityWebRequest();


    using (request = UnityWebRequest.Post(TARGET_DOMAIN + "waitRoom/costume", json.ToString()))
    {
        byte[] jsonToSend = new System.Text.UTF8Encoding().GetBytes(json.ToString());
        request.uploadHandler = new UploadHandlerRaw(jsonToSend);
        request.downloadHandler = (DownloadHandler)new DownloadHandlerBuffer();
        request.SetRequestHeader("Content-Type", "application/json");


        yield return request.SendWebRequest();

        if (request.isNetworkError || request.isHttpError)
        {
            Debug.LogError(request.responseCode + " / " + request.error + "/" + request.downloadHandler.text);
            _ErrorResponseCode = request.responseCode.ToString();
            if ((request.downloadHandler.text).ToString().Contains("message"))
            {
                _ErrorMessage = JsonMapper.ToObject(request.downloadHandler.text)["message"].ToString();
            }

            if (p_failCallback != null)
                p_failCallback();
        }
        else
        {
            Debug.Log(request.downloadHandler.text);

            if (p_successCallback != null)
                p_successCallback();
        }

        request.Dispose();
    }
}
참조 2개
```

아바타 버튼 클릭시 내가 가지고있는 아바타 리스트가 나오며 사용할 아바타 클릭시 유저의 uid 와 현재 선택한 아바타숫자를 API를 통해 넘겨 다음 게임에 들어와도 이전에 선택한 아바타가 저장되어있다.

WaitRoom(userName)

```
pv.RPC("ReadTextSend", RpcTarget.AllBufferedViaServer, PhotonNetwork.LocalPlayer.NickName);
//플레이어가 로딩이 끝나고 접속완료되었을 시 본인 닉네임을 RPC로 호출해 접속된 전체 플레이어한테 전달

[PunRPC]
참조 0개
void ReadTextSend(string Name)
{
    PlayerReadyCheck.Add(Name);//준비완료된 플레이어 닉네임 저장
}
```

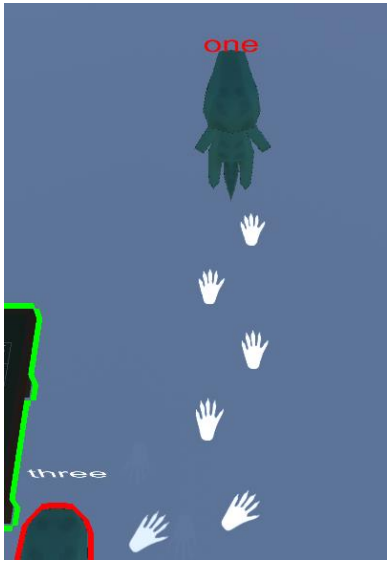
```
void chatterUpdate()
{
    string chatters = "Player List (" + PhotonNetwork.PlayerList.Length + " / " + PhotonNetwork.CurrentRoom.MaxPlayers + ")₩n";

    for (int j = 0; j < PlayerReadyCheck.Count; j++) //준비완료된 유저는 흰색으로 닉네임 표시
    {
        chatters += "<color=#FFFFFF>" + PlayerReadyCheck[j] + "</color>" + "₩n";
    }

    for (int i = PlayerReadyCheck.Count; i < PhotonNetwork.PlayerList.Length; i++) //방입장 로딩중인 유저는 검은색으로 닉네임 표시
    {
        chatters += "<color=#000000>" + PhotonNetwork.PlayerList[i].NickName + "</color>" + "₩n";
    }


    playerList.text = chatters;
}
```

플레이어가 로비에서 방입장 버튼을 누르고 인게임으로 씬 전환 및 포톤에서 방입장되는 내용 로딩하는 동안 대기방에서 표출되는 닉네임의 색에 변화를 줘서 게임을 시작할 준비가 되어있는지 확인할 수 있도록 함.

FootStep Particle

```
if (pv.IsMine) //이동처리 부분에서 발자국의 경우 다른사람에게도 보여야 함으로 RPC사용을 위해 ismine체크
{
    if (Vector3.Distance(last_footstep, player_position) > gap_with_pl_pos) //다음 발자국이 생겨야할 거리 조절
    {
        var pos = player_position + (transform.right * gap_footsteps * footstep_controller);
        footstep_controller *= -1;
        pos.y = 0.2f;
        pv.RPC("FootStep", RpcTarget.All, pos); //좌표값 조정 후 RPC로 호출
    }
}
```

```
[PunRPC]
참조 0개
void FootStep(Vector3 pos) //파티클이 생성되야 할 포지션을 받아 RPC로 호출
{
    ParticleSystem.EmitParams emit = new ParticleSystem.EmitParams();
    emit.position = pos;
    emit.rotation = transform.rotation.eulerAngles.y;
    footsteps.Emit(emit, 1);
    last_footstep = player_position;
}
```



플레이어가 이동할 때 교차하서 생성되는 발자국 파티클을 다른 플레이어들도 확인 할 수 있도록 RPC를 활용하여 제어함.

Kill

```
public void KillPlayer() //헌터가 킬버튼을 눌렀을 때
{
    if (pv.IsMine)
    {
        if (DeadPlayer != null)
        {
            Kill_btn.gameObject.SetActive(false);
            StartCoroutine(DeadPlayer.GetComponent<PlayerManager>().DeadPlay(DeadPlayer.gameObject.name.Replace("(Clone)", ""),
                InGameManager._Iinstance._WeaponNumber, InGameManager._Iinstance.TimeNum, PhotonNetwork.NickName, this.gameObject));
            //죽는 플레이어, 무기 단서, 시간 단서, 헌터 단서를 가지고 처리하는 함수로 이동
        }
    }
}
```

```
public IEnumerator DeadPlay(string DeadBodyName, int WeaponNum, int TimeNum, string KillerName, GameObject Obj)
{
    pv.RPC("DeadPlayerSetting", RpcTarget.All);//사망한 유저 유령화


    yield return new WaitForSeconds(5f);//시체가 등장하는 시간 조절
    GameObject playerTemp = PhotonNetwork.Instantiate(DeadBodyName, this.transform.position, Quaternion.identity, 0);

    yield return new WaitForSeconds(0.1f);

    Obj.GetComponent<PlayerManager>().DeadBody_PM = playerTemp.GetComponent<PlayerManager>();
    Obj.GetComponent<PlayerManager>().DeadBody_PM.pv.RPC("DeadBodySetting", RpcTarget.All, WeaponNum, TimeNum, _NameText.text, KillerName);
    //시체에 대한 내용 및 부검에 필요한 정보를 다른 유저에게 전달

    yield return new WaitForSeconds(0.2f);
    AllIntoDeadBody();//부검할 시체 오브젝트를 인터렉션 가능한 오브젝트로 지정
}
```

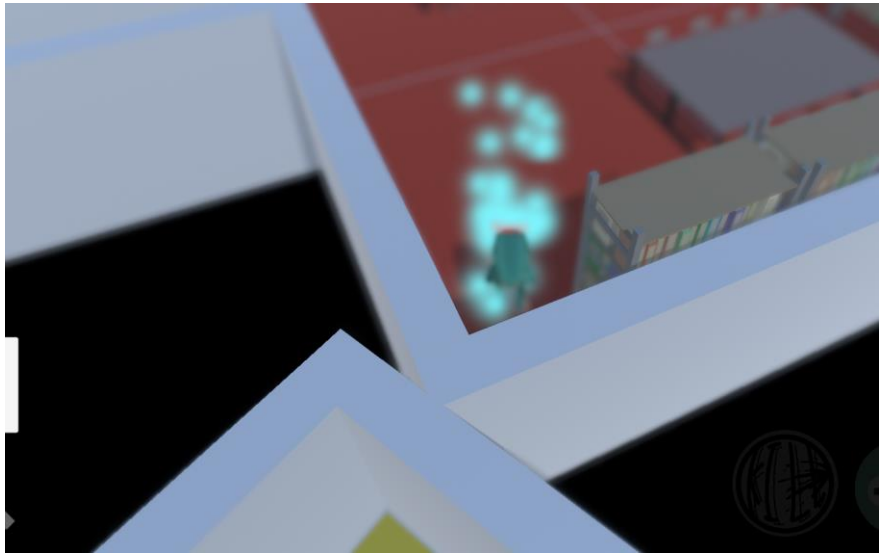
Kill

```
[PunRPC]
참조 0개
void DeadPlayerSetting() //헌터에게 살해당한 유저를 유령으로 처리하고 다른 플레이어들에게도 전달
{
    this.gameObject.GetComponent<PlayerManager>()._pState = PlayerState.Dead;
    this.gameObject.GetComponent<CapsuleCollider>().enabled = false;
    this.gameObject.GetComponent<PlayerManager>()._NameText.color = new Color(0.5f, 0.5f, 0.5f);
    this.gameObject.GetComponent<PlayerManager>().Player_MoveEnd();

    ChangeLayersRecursively(this.gameObject.transform, "Ghost"); //유령으로 레이어 변화
    Camera.main.GetComponent<Main_Camera>().GhostLayerSetting(); //해당 레이어에 맞춰 카메라 레이어 변경
}
```

```
[PunRPC]
참조 0개
void DeadBodySetting(int WeaponNum, int TimeNum, string Name, string KillerName)
{ //생성되는 시체에 대한 정보 입력 후 다른 플레이어들이 부검할 수 있도록 함
    this.gameObject.AddComponent<DeadbodyData>();
    this.gameObject.GetComponent<DeadbodyData>().DeadWeapon = (DeadbodyData.Weapon)WeaponNum;
    this.gameObject.GetComponent<DeadbodyData>().DeadTime = (DeadbodyData.Time)TimeNum;
    this.gameObject.GetComponent<DeadbodyData>().KillerName = KillerName;
    this.gameObject.transform.GetChild(2).GetComponent<TextMesh>().text = Name;
    this.gameObject.GetComponent<Animator>().enabled = false;
    this.gameObject.transform.rotation = Quaternion.Euler(90, this.transform.rotation.y, this.transform.rotation.z);

    this.gameObject.transform.GetChild(0).transform.GetChild(0).transform.GetChild(0).gameObject.AddComponent<cakeslice.Outline>();
    this.gameObject.transform.GetChild(0).transform.GetChild(0).transform.GetChild(0).gameObject.GetComponent<cakeslice.Outline>().color = 2;

    InGameManager._Iinstance.C_Time.Remove(TimeNum);
    InGameManager._Iinstance.C_Time.Remove(TimeNum);

    InGameManager._Iinstance.C_Weapon.Remove(WeaponNum);
    InGameManager._Iinstance.C_Weapon.Remove(WeaponNum);


    Destroy(this.gameObject.GetComponent<PlayerManager>());
}
```

헌터가 플레이어를 살해했을 시 당한 플레이어의 오브젝트는 유령이 되고 다른 플레이어들이 부검할 시체 오브젝트를 생성하여 RPC를 통해 2개의 오브젝트를 분리하여 제어함.

PostProcessing(blur)



```
public void PlayDepth()
{
    postProcess.profile.TryGetSettings(out DepthOfField dof);
    dof.enabled.value = true;
    dof.aperture.value = 32;
    StartCoroutine(Play());
}
참조 1개
public void Stop()
{
    postProcess.profile.TryGetSettings(out DepthOfField dof);
    dof.enabled.value = false;
    dof.aperture.value = 32;
}
참조 1개
IEnumerator Play()
{
    postProcess.profile.TryGetSettings(out DepthOfField dof);
    float f = dof.aperture.value;
    float time = 0;
    float curvalue;
    while (time<1)
    {
        curvalue = Mathf.Lerp(f, 0, time);
        dof.aperture.value = curvalue;
        yield return null;
        time +=  Time.deltaTime/TelePort.DesTime;
    }
}
```

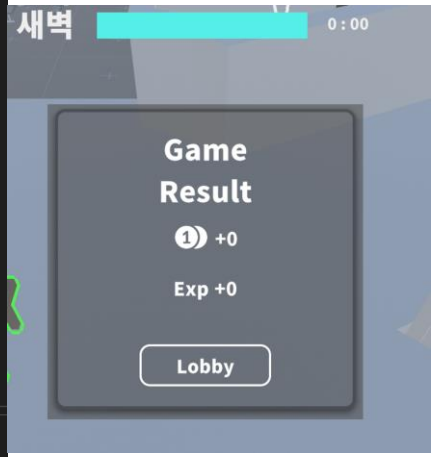
PostProcessing을통한 blur처리

Answer

```
public void AnswerPanelOpen()
{//플레이어가 정답을 맞추기 위한 오브젝트 인터렉트할 시 다른 플레이어들도 과정을 볼 수 있도록 RPC 호출
    pv.RPC("AnwserActive", RpcTarget.All, true);
    AnswerTouchPanel.SetActive(false);
}
```

```
[PunRPC]
참조 0개
void AnwserActive(bool activeValue)
{//정답 맞추는 UI 제어
    if (activeValue)
    {
        //정답 값 초기화
        TimeText.text = "";
        WeaponText.text = "";
        PlayerText.text = "";

        for (int i = 0; i < SetObj.Length; i++)
        {
            SetObj[i].SetActive(false);
        }

        //현재 남아있는 플레이어들의 닉네임만 선택가능한 버튼으로 활성화 되게끔 초기화
        for (int i = 0; i < PlayerAnwserBtn.Length; i++)
        {
            PlayerAnwserBtn[i].onClick.RemoveAllListeners();
            PlayerAnwserText[i].text = null;
            PlayerAnwserBtn[i].gameObject.SetActive(false);
        }

        for (int i = 0; i < PlayerReadyCheck.Count; i++)
        {
            PlayerAnwserBtn[i].gameObject.SetActive(true);
            PlayerAnwserText[i].text = PlayerReadyCheck[i];
            string s = PlayerReadyCheck[i];
            PlayerAnwserBtn[i].onClick.AddListener(() => PlayerSelectBtn(s));
        }
    }
    else
    {
        chatting.SetActive(false);
    }
    AnswerObj.SetActive(activeValue);//UI 액티브 제어
    _pa.NowEnd(activeValue); //플레이어 이동 제어
}

[PunRPC]
참조 0개
void TimeSelect(string TimeValue)
{//플레이어가 시간 UI 선택시 다른 유저에게도 선택한 값이 보이게끔 RPC 호출
    TimeText.text = TimeValue;
}

[PunRPC]
참조 0개
void WeaponSelect(string WeaponValue)
{//플레이어가 무기 UI 선택시 다른 유저에게도 선택한 값이 보이게끔 RPC 호출
    WeaponText.text = WeaponValue;
}

[PunRPC]
참조 0개
void PlayerSelect(string PlayerValue)
{//플레이어가 플레이어 UI 선택시 다른 유저에게도 선택한 값이 보이게끔 RPC 호출
    PlayerText.text = PlayerValue;
}
```

```
public void AnswerCheck()
{//플레이어가 정보를 취합하여 내놓은 답의 결과 처리 함수
    if (_DeadData.DeadTime.ToString() == TimeText.text && _DeadData.DeadWeapon.ToString() == WeaponText.text && _DeadData.KillerName.ToString() == WeaponText.text)
    {
        //경험치와 골드 획득
        APIManager._AInstance._userData.exp += 100;
        APIManager._AInstance._userData.gold += 100;

        ResultGoldText.text = "+100";

        int NowLevel = APIManager._AInstance._userData.level; //해당 경험치로 인해 레벨이 오른 경우에 대한 처리
        if (NowLevel < GameManager.ginstance.ExpList.Length)
        {
            if (GameManager.ginstance.ExpList[NowLevel] <= APIManager._AInstance._userData.exp)
            {
                APIManager._AInstance._userData.exp -= GameManager.ginstance.ExpList[NowLevel];
                APIManager._AInstance._userData.level += 1;

                ResultLevelText.text = "+1";
            }
            else
            {
                ResultLevelText.text = "+0";
            }
        }

        ResultLevelText.text = "+100";

        pv.RPC("ResultTrue", RpcTarget.All);

        APIManager._AInstance.PostUserStateUpdate(ResultPanelOpen, null);//API를 통해 해당 플레이어의 변화된 데이터 값 저장 후 결과 UI 오픈
    }
    else
    {
        // 플레이어의 답이 오답일 시 본인이 들고있던 단서 다시 회수
        // Weapons_card, Time_card, MyCardList
        if (MyCardList.Count != 0)
        {
            // 알파어펑수 넣기
            string cardNums = "Card : ";
            for (int i = 0; i < MyCardList.Count; i++)
            {
                cardNums += MyCardList[i] + "/";
            }

            pv.RPC("DeadUserCardOpen", RpcTarget.All, cardNums);//다른 유저들에게 회수된 카드 정보 RPC로 호출
        }
        else
        {
            Debug.Log("갖고있던 카드 없음");
        }
        //오답으로 인한 소량의 경험치 및 골드 획득 처리
        APIManager._AInstance._userData.exp += 10;
        APIManager._AInstance._userData.gold += 10;

        ResultGoldText.text = "+10";

        int NowLevel = APIManager._AInstance._userData.level;
        if (NowLevel < GameManager.ginstance.ExpList.Length)
        {
            if (GameManager.ginstance.ExpList[NowLevel] <= APIManager._AInstance._userData.exp)
            {
                APIManager._AInstance._userData.exp -= GameManager.ginstance.ExpList[NowLevel];
                APIManager._AInstance._userData.level += 1;

                ResultLevelText.text = "+1";
            }
            else
            {
                ResultLevelText.text = "+0";
            }

            ResultLevelText.text = "+10";
        }
        APIManager._AInstance.PostUserStateUpdate(null, null);

        InGameManager._instance.PlayerRemove(APIManager._AInstance._userData.nickName);//플레이어가 인게임에서 나가도록 처리
    }

    ResultObj.SetActive(true);
    _pa.NowEnd(true);
    AnswerObj.SetActive(false);

    pv.RPC("AnwserActive", RpcTarget.All, false);//모든플레이어들의 정답 맞추는 UI 닫아주기

    StartCoroutine(DestroyPlayer());
}
```



플레이어가 정답을 맞추려고 인터렉션을 했을 시 다른 플레이어들은 해당 과정을 관전할 수 있도록 하며 해당 플레이어가 내놓은 답에 의한 결과 보상 처리 및 종료 처리 과정을 담음.

Player Out

```
public override void OnPlayerLeftRoom(Player otherPlayer)
{//플레이어가 강제종료하거나 인게임에서 나가기를 했을 시 호출되는 콜백 함수
    Debug.LogFormat("{0} has left the room", otherPlayer.NickName);
    // 떠난 플레이어의 정보를 출력하거나, 다른 플레이어들에게 알리는 등의 처리를 수행함.

    InGameManager._Iinstance.PlayerRemove(otherPlayer.NickName);
}
```

```
public void PlayerRemove(string Name)
{
    //나가는 플레이어의 이름을 다른 플레이어들에게 지우도록 RPC 호출
    pv.RPC("RemovePlayerName", RpcTarget.AllBufferedViaServer, Name);
    if (WaitObj.active == false) //인게임 도중에만 적용 되도록 확인
    {
        if (_pm._pState != PlayerManager.PlayerState.Dead && !ResultCheck)
        {

            StartCoroutine(OnePlayerLive());//인게임에 혼자 남아있는지 확인하는 코루틴
        }
    }


}
```

Player Out


```
IEnumerator OnePlayerLive()
{
    yield return new WaitForSeconds(1f);

    //혼자 남은 인원이 헌터인지 확인
    bool HunterStayCheck = false;

    for (int i = 0; i < _pm.interaction_objs.Count; i++)
    {
        if (_pm.interaction_objs[i].GetComponent<PlayerManager>() != null)
        {
            if (_pm.interaction_objs[i].GetComponent<PlayerManager>()._pState == PlayerManager.PlayerState.Hunter)
            {
                HunterStayCheck = true;
            }
        }
    }

    if(_pm._pState == PlayerManager.PlayerState.Hunter)
    {
        HunterStayCheck = true;
    }

    yield return new WaitForSeconds(0.1f);

    //결과 보상 처리
    if (ResultObj.active == false && HunterSetCount == 0)
    {
        if (PlayerReadyCheck.Count <= 1 || !HunterStayCheck)
        {
            APIManager._Ainstance._userData.exp += 100;
            APIManager._Ainstance._userData.gold += 100;

            ResultGoldText.text = "+100";

            int NowLevel = APIManager._Ainstance._userData.level;
            if (NowLevel < GameManager.ginstance.ExpList.Length)
            {
                if (GameManager.ginstance.ExpList[NowLevel] <= APIManager._Ainstance._userData.exp)
                {
                    APIManager._Ainstance._userData.exp -= GameManager.ginstance.ExpList[NowLevel];
                    APIManager._Ainstance._userData.level += 1;

                    ResultLevelText.text = "+1";
                }
                else
                {
                    ResultLevelText.text = "+0";
                }
            }

            ResultLevelText.text = "+100";

            //헌터와 시민에 따라 유령으로 남아있던 유저에게 다른 결과 표출을 위한 조건문
            if (!HunterStayCheck)
            {
                APIManager._Ainstance.PostUserStateUpdate(null, null);
            }
            else
            {
                APIManager._Ainstance.PostUserStateUpdate(ResultPanelOpen, null);
            }

            ResultObj.SetActive(true);
            _pm.NowEnd(true);

            StartCoroutine(DestroyPlayer());
        }
    }
}
```

플레이어가 인게임 도중에 나가거나 강제종료 시 호출되는 콜백함수 처리 및 플레이어가 나가면서 인게임에 유령 플레이어를 제외하고 혼자 남아있는 경우에 대한 처리를 함.

Note


```
IEnumerator DrawLine()
{
    GameObject _line = Instantiate(_lineRendererPrefab);
    _line.transform.SetParent(Note.transform);
    lineRenderer = _line.GetComponent<UILineRenderer>();
    if (penCheck)
    {
        lineRenderer.color = SetColor;
    }
    else
    {
        lineRenderer.color = ClearColor;
        lineRenderer.LineThickness = lineThickness;
    }
    while (true)
    {
        Vector2 pos = Camera.main.ScreenToViewportPoint(Input.mousePosition);
        pos = new Vector2(pos.x * 1920, pos.y * 1080);
        points.Add(pos);
        lineRenderer.Points = points.ToArray();
        yield return null;
    }
}
참조 2개
```

LineRenderer를 UI에서 사용하여 Line생성