Twin Travelers

유니티 포트폴리오 기능 정리

Dev: leeinhwan0421

포트폴리오 가이드

- 본 포트폴리오는 1인 개발 프로젝트로서 리소스를 제외한 모든 부분을 직접 구현하였습니다.
- 최대한 핵심 내용만을 정리하고자 하였습니다.

- 사용한 에셋 리스트는 아래와 같습니다.
  - 스프라이트: 2D Platformer Tileset, Trampoline from Cuphead, Kenney Cursor Pack
  - 네트워크: PUN 2 – FREE
  - 폰트: Rix X 수박양
  - 효과음: Pro Sound Collection, Keyboard Soundpack #1

- 프로젝트 코드 및 자세한 구성을 확인하고 싶으신 분들은 여기 에서 확인해주세요.
- 추가적인 질문이 있으신 경우, lsnan421@naver.com 이메일로 문의를 남겨주시면 최대한 빠른 시일 내에 답변하겠습니다.

사용한 기술 스택

엔진 및 언어

Unity 2022.3.45f1

.NET Framework 2.1

관리

Notion (일정 관리)

Github (버전 관리)

개발 환경

Visual Studio 2022

사용한 패키지

- Shader Graph
- Tilemap
- PUN 2

목차

1 게임 소개

‘Twin Travelers’는 2022년 전국기능경기대회 출품작 ‘삼척 더블 댄스‘를 리마스터한 작품입니다.

리마스터 과정에서 기존보다 향상된 비주얼과 스테이지 디자인을 가지고 있으며 멀티 플레이가 가능해 협력 플레이가 가능합니다.

‘Twin Travelers’는 단순히 리마스터된 외형에 그치지 않고, 기존 작품이 지니고 있던 매력을 극대화하는 동시에 게임 볼륨을 확대시켰습니다.

“어둠의 탐험가가 되어 미지의 숲과 던전 속으로 들어가자!”

숨겨진 비밀과 잊혀진 이야기를 찾아가며 위협적인 존재와 맞서 싸우고 각종 위협을 피해 살아남으세요!

플레이 및 시연 영상: YouTube

삼척 더블 댄스

Twin Travelers

2 플레이어 코드

1. 플레이어 오브젝트

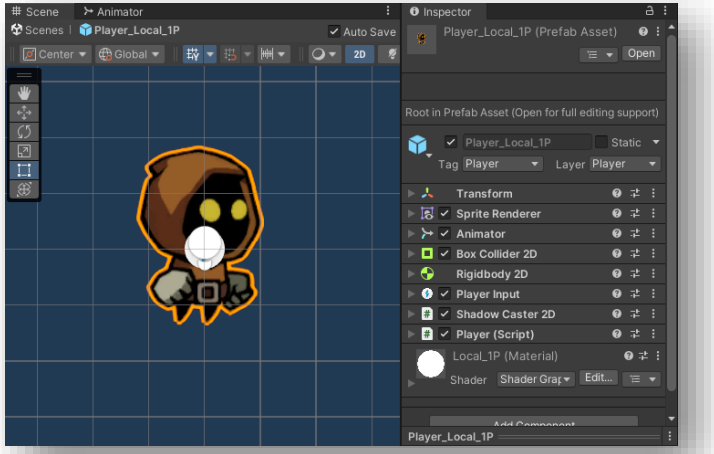
의도 | 플레이어는 단순히 걷고, 점프하는 조작만 존재하기 때문에 최대한 간결하게 코드를 작성하였고, 2D Light를 쓰는 프로젝트 특성 상, Shadow Caster 2D 컴포넌트를 첨부해 그림자가 실시간으로 생기도록 하였습니다.

방식 | New Input System을 사용하여 플레이어 클래스에 이동 로직을 실행하는 이벤트를 보내며 현재 네트워크 상태에 따라 플레이어의 위치 동기화 로직 실행 여부를 정합니다. 네트워크 환경에서 위치 동기화 로직은 지연 보간을 활용해 부드럽게 처리하였습니다.

효과 | 로컬/멀티 플레이 환경에서 플레이어의 이동 로직 및 사망 로직이 정상적으로 작동됩니다.

전체 코드: 바로가기

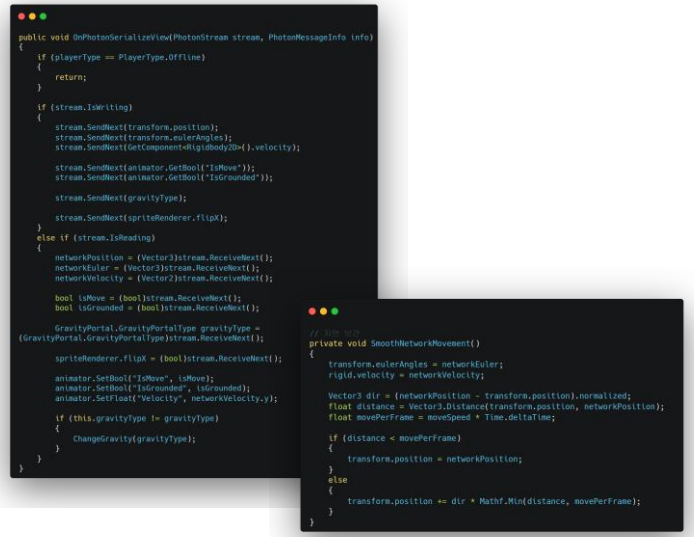
플레이어 구성



이동 코드

```
/// 이동 처리 로직
private void Move()
{
    if (moveDirection == Vector3.zero)
        return;

    transform.Translate(moveDirection * Time.deltaTime * moveSpeed);
    spriteRenderer.flipX = moveDirection.x < 0;
}

/// Player Input Send Message
private void OnMove(InputValue value)
{
    float axis = value.Get<float>();

    moveDirection.x = axis;
}

/// Player Input Send Message
private void OnJump()
{
    if (!isGrounded) return;

    AudioManager.Instance.PlaySFX("Jump");

    rigid.velocity = transform.up.normalized * JumpVelocity;
}
```

동기화 코드



```
public void OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)
{
    if (playerType == PlayerType.Offline)
    {
        return;
    }

    if (stream.IsWriting)
    {
        stream.SendNext(transform.position);
        stream.SendNext(transform.eulerAngles);
        stream.SendNext(GetComponent<Rigidbody2D>().velocity);

        stream.SendNext(animator.GetBool("IsMove"));
        stream.SendNext(animator.GetBool("IsGrounded"));

        stream.SendNext(gravityType);

        stream.SendNext(spriteRenderer.flipX);
    }
    else if (stream.IsReading)
    {
        networkPosition = (Vector3)stream.ReceiveNext();
        networkEuler = (Vector3)stream.ReceiveNext();
        networkVelocity = (Vector2)stream.ReceiveNext();

        bool isMove = (bool)stream.ReceiveNext();
        bool isGrounded = (bool)stream.ReceiveNext();

        GravityPortal.GravityPortalType gravityType =
(GravityPortal.GravityPortalType)stream.ReceiveNext();

        spriteRenderer.flipX = (bool)stream.ReceiveNext();

        animator.SetBool("IsMove", isMove);
        animator.SetBool("IsGrounded", isGrounded);
        animator.SetFloat("Velocity", networkVelocity.y);

        if (this.gravityType != gravityType)
        {
            ChangeGravity(gravityType);
        }
    }
}
```

```
// 싱크 맞추
private void SmoothNetworkMovement()
{
    transform.eulerAngles = networkEuler;
    rigid.velocity = networkVelocity;

    Vector3 dir = (networkPosition - transform.position).normalized;
    float distance = Vector3.Distance(transform.position, networkPosition);
    float movePerFrame = moveSpeed * Time.deltaTime;

    if (distance < movePerFrame)
    {
        transform.position = networkPosition;
    }
    else
    {
        transform.position += dir * Mathf.Min(distance, movePerFrame);
    }
}
```

3 네트워크 코드

1. RoomManager 클래스

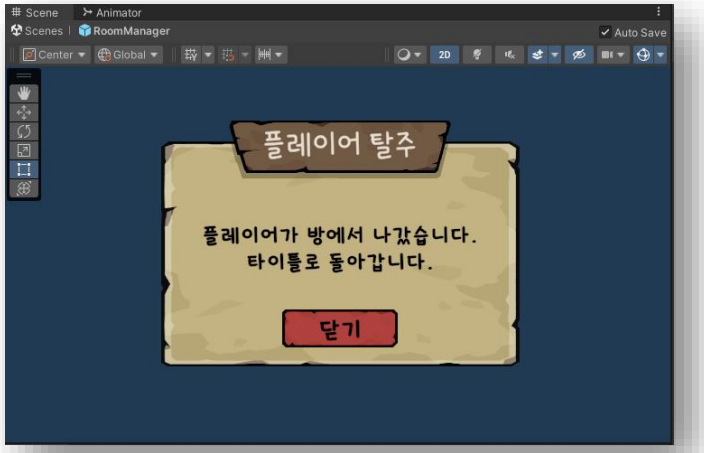
의도 | Photon을 활용한 멀티플레이 환경에서 방(Room) 생성, 참가, 퇴장을 관리하고 플레이어 간의 연결 상태 및 이벤트를 처리하기 위한 목적으로 제작하였습니다. 누군가가 탈주하거나 연결이 끊긴 경우 등 각종 시나리오에 대해 처리할 수 있게 했습니다.

방식 | OnCreateRoom, OnJoinedRoom, OnJoinedFailed 등 Photon에서 제공하는 콜백 메서드를 통해 방 생성 및 입장 성공/실패 상황에 따른 UI 상태 변경을 수행합니다. 또한 커스텀 이벤트를 통해 씬 로드를 처리합니다.

효과 | Photon을 활용해 로비 연결, 방 생성, 참가, 퇴장을 성공적으로 수행하며 방에 있는 모든 클라이언트에 대해 씬 로딩을 수행할 수 있습니다.

전체 코드: 바로가기

플레이어 탈주 UI



방 생성 및 참가 UI

생성/참가/퇴장 코드

2. ConnectionChecker 클래스

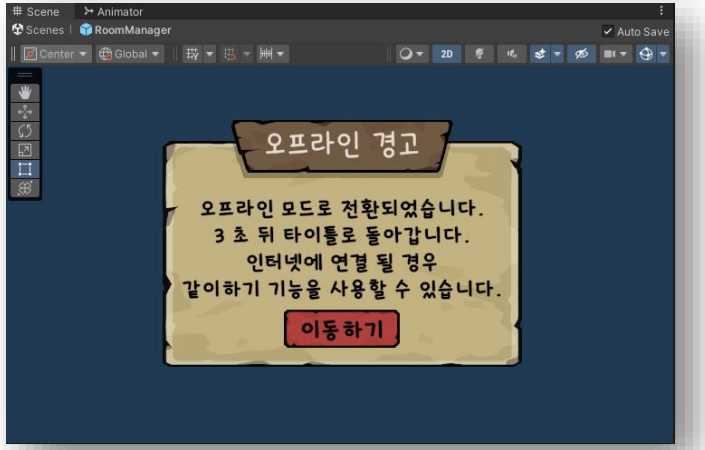
의도 | 현재 네트워크 연결 상태가 온라인에서 오프라인으로 전환 또는 오프라인에서 온라인으로 전환 될 때 Photon 서버 로비에서 퇴장/입장을 수행할 목적으로 제작하였습니다.

방식 | Coroutine을 사용해서 3초마다 네트워크 상태를 확인합니다. 온라인에서 오프라인으로 변경 시 접속해있던 방/로비에서 나가집니다. 오프라인에서 온라인으로 변경 시 Photon 서버 로비에 입장을 수행합니다.

효과 | 온라인에서 오프라인, 오프라인에서 온라인으로 네트워크 상태가 변경 될 때, 알맞은 UI와 알맞은 동작을 잘 수행합니다.

전체 코드: 바로가기

오프라인 경고 UI



네트워크 상태 확인 코드

```
private IEnumerator PeriodicConnectionCheck()
{
    while (true)
    {
        bool isCurrentlyOffline = !IsInternetAvailable();

        if (isCurrentlyOffline && !wasOffline)
        {
#if UNITY_EDITOR
            Debug.Log("Internet disconnected. Showing offline warning...");
#endif
            ShowOfflineWarning();
            wasOffline = true;

            RoomManager.Instance.LeaveRoom();
            PhotonNetwork.Disconnect();
        }
        else if (!isCurrentlyOffline && wasOffline)
        {
#if UNITY_EDITOR
            Debug.Log("Internet reconnected. Hiding offline warning and reconnecting...");
#endif
            HideOfflineWarning();
            wasOffline = false;

            PhotonNetwork.ConnectUsingSettings();
        }

        yield return new WaitForSeconds(3);
    }
}
```

4 유틸리티 코드

1. CameraMovement 클래스

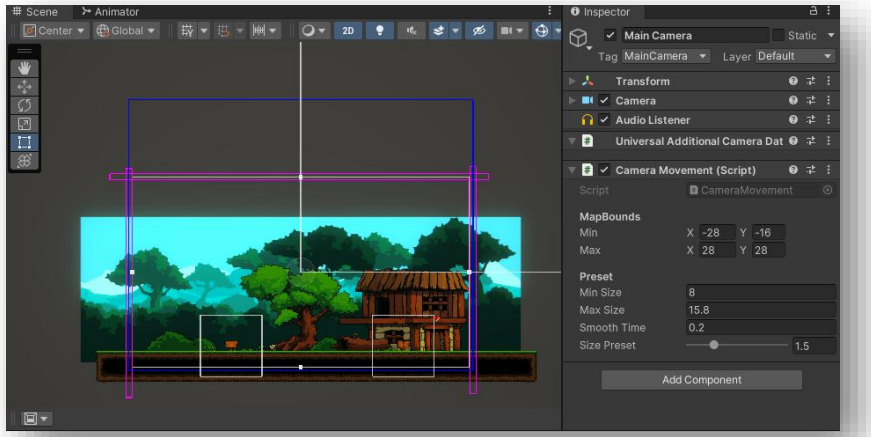
의도 | 두 플레이어 거리에 비례해서 크기가 커지는 카메라 컴포넌트를 바라고 제작하였습니다. 또한 편의성 기능으로 줌 인/아웃을 지원합니다.

방식 | 두 플레이어의 중심점에 카메라를 위치시키고, 두 플레이어 거리에 따라 카메라의 크기를 조정합니다. 또한 카메라의 영역을 설정하여 그 바깥으로 나가지 못하게 하였습니다. 줌 아웃 모드가 바뀌면 즉시 최대 크기로 카메라가 커집니다.

효과 | 두 플레이어의 위치에 따라 카메라가 자동으로 중앙에 배치되며 사이즈가 동적으로 변경됩니다. 카메라가 지정된 경계를 벗어나지 않도록 제한함으로 의도치 않은 곳이 노출되지 않도록 하였습니다.

전체 코드: 바로가기

카메라 구성



확대/축소 비교

2. INIParser 클래스

의도 | INI 파일을 통해 게임 설정을 저장하는 '아이작의 번제' 게임에서 착안해 INI 파일을 통해 게임 설정을 쓰기/불러오기 하기 위한 목적으로 제작하였습니다.

방식 | 파일을 저장/쓰기 할 때 '='를 구분자로 사용하며 '키=값' 형태의 데이터를 읽거나 저장할 수 있도록 구현하였습니다.

효과 | INI 파일을 읽어서 Dictionary<Key, Value> 형태로 반환하거나, Dictionary<Key, Value> 형태의 데이터를 저장할 수 있습니다.

전체 코드: 바로가기

읽기 코드

```
/// <summary>
/// INI 파일을 읽어서 Dictionary로 반환합니다.
/// </summary>
/// <param name="path">INI 파일의 위치</param>
/// <returns>INI 파일 내부 데이터를 , <Key, Value> 형태로 반환합니다.</returns>
public static Dictionary<string, string> ReadINI(string path)
{
    var data = new Dictionary<string, string>();

    if (!File.Exists(path))
        return data;

    foreach (var line in File.ReadAllLines(path))
    {
        if (string.IsNullOrWhiteSpace(line) || line.StartsWith(";") || line.StartsWith("#"))
            continue;

        var keyValue = line.Split(new[] { '=' }, 2);
        if (keyValue.Length == 2)
        {
            var key = keyValue[0].Trim();
            var value = keyValue[1].Trim();
            data[key] = value;
        }
    }

    return data;
}
```

쓰기 코드

```
/// <summary>
/// Dictionary를 INI 파일로 저장합니다.
/// </summary>
/// <param name="path">INI 파일의 위치</param>
/// <param name="data">Dictionary</param>
public static void WriteINI(string path, Dictionary<string, string> data)
{
    StreamWriter writer = new StreamWriter(path);

    foreach (var entry in data)
    {
        writer.WriteLine($"{entry.Key}={entry.Value}");
    }

    writer.Close();
}
```

3. InteractableCollision/Trigger 클래스

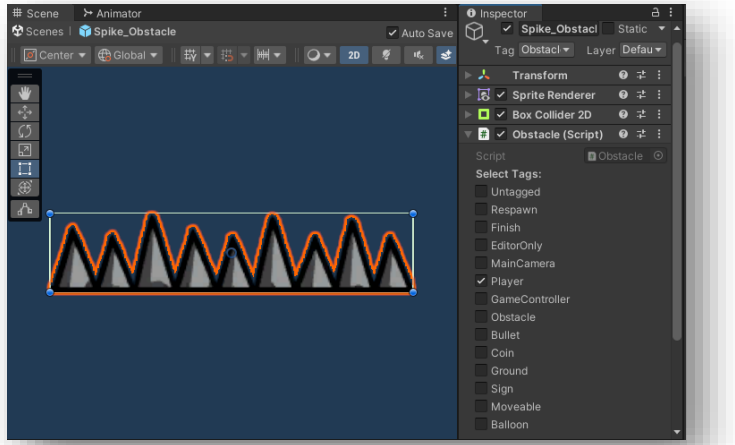
의도 | 각 태그마다 오브젝트를 지정한 뒤 플레이어가 충돌 처리할 때 하나씩 비교하는 것은 효율적이지 않다고 생각해 특정 콜라이더와 콜리전이 특정 태그와 반응하는 콜라이더/트리거를 제작했습니다.

방식 | 커스텀 에디터를 활용해 충돌할 태그를 인스펙터 상에서 지정합니다. 충돌 시 OnCollision~2D, OnTrigger~2D 유니티 콜백 함수에서 충돌할 태그와 충돌했는지 검사를 진행하고, 충돌했다면 EnterEvent/ExitEvent 메서드를 호출합니다.

효과 | 로컬/멀티 플레이 환경에서 플레이어의 이동 로직 및 사망 로직이 정상적으로 작동됩니다.

메인 코드: 바로가기
에디터 코드: 바로가기

인스펙터(커스텀 에디터 활용) 뷰



충돌 코드

```
/// <summary>
/// 충돌 이벤트를 처리합니다.
/// </summary>
/// <param name="collision">충돌체</param>
protected abstract void EnterEvent(Collision2D collision);

/// <summary>
/// 충돌 이벤트를 처리합니다.
/// </summary>
/// <param name="collision">충돌체</param>
protected abstract void ExitEvent(Collision2D collision);

/// <summary>
/// 태그 검사 후 메서드 호출
/// </summary>
/// <param name="collision">충돌체</param>
private void OnCollisionEnter2D(Collision2D collision)
{
    if (selectedTags.Contains(collision.gameObject.tag))
    {
        EnterEvent(collision);
    }
}

/// <summary>
/// 태그 검사 후 메서드 호출
/// </summary>
/// <param name="collision">충돌체</param>
private void OnCollisionExit2D(Collision2D collision)
{
    if (selectedTags.Contains(collision.gameObject.tag))
    {
        ExitEvent(collision);
    }
}
```

에디터 코드

```
public override void OnInspectorGUI()
{
    DrawDefaultInspector();

    InteractableTrigger triggerScript = (InteractableTrigger)target;
    string[] allTags = UnityEditorInternal.InternalEditorUtility.tags;

    EditorGUILayout.LabelField("Select Tags:", EditorStyles.boldLabel);

    for (int i = 0; i < allTags.Length; i++)
    {
        tagChecks[i] = EditorGUILayout.ToggleLeft(allTags[i], tagChecks[i]);

        if (tagChecks[i] && !triggerScript.selectedTags.Contains(allTags[i]))
        {
            triggerScript.selectedTags.Add(allTags[i]);
        }
        else if (!tagChecks[i] && triggerScript.selectedTags.Contains(allTags[i]))
        {
            triggerScript.selectedTags.Remove(allTags[i]);
        }
    }

    if (GUI.changed)
    {
        EditorUtility.SetDirty(triggerScript);
    }
}
```

각주 | 작성일 기준 진행중인 프로젝트에서, List를 HashSet으로 변환하여 시간 복잡도를 줄였고 ($O_{(n)} \rightarrow O_{(1)}$), 인스펙터에서 체크박스가 아닌, LayerMask처럼 드롭다운 형태로 변경하였습니다.

5 저장 구조

1. 개요

방식 | Theme 클래스에 여러 개의 개별 스테이지 정보가 담겨 있으며, 개별 스테이지에는 스테이지 이름, 획득한 별 개수, 잠금 해제 여부를 가지고 있습니다. 이 데이터 구조는 LevelManager 클래스에서 저장 및 불러오기합니다. 저장 및 불러오기는 PlayerPrefs를 사용하며 스테이지 완료 시 다음 스테이지를 잠금 해제하고 저장합니다.

효과 | 게임 재시작 시에도 게임 진행 상황이 유지됩니다.

전체 코드: 바로가기

테마 및 레벨 코드

```
[System.Serializable]
public class Stage
{
    public string stageName;    // 스테이지 이름
    public int starCount;    // 별 획득 개수
    public bool isUnlocked;   // 스테이지 잠금 여부
}

[System.Serializable]
public class Theme
{
    public string themeName;  // 테마 이름
    public List<Stage> stages;     // 해당 테마에 포함된 스테이지 목록
}
```

저장 및 불러오기 코드

```
/// 모든 스테이지 정보 저장
public static void SaveProgress()
{
    for (int i = 0; i < themeCount; i++)
    {
        for (int j = 0; j < stageCount; j++)
        {
            PlayerPrefs.SetInt($"Theme{i + 1}_Stage{j + 1}_Stars", themes[i].stages[j].starCount);
            PlayerPrefs.SetInt($"Theme{i + 1}_Stage{j + 1}_Unlocked", themes[i].stages[j].isUnlocked ? 1 : 0);
        }
    }
}

/// 모든 스테이지 정보 불러오기
public static void LoadProgress()
{
    for (int i = 0; i < themeCount; i++)
    {
        for (int j = 0; j < stageCount; j++)
        {
            themes[i].stages[j].starCount = PlayerPrefs.GetInt($"Theme{i + 1}_Stage{j + 1}_Stars", 0);
            themes[i].stages[j].isUnlocked = PlayerPrefs.GetInt($"Theme{i + 1}_Stage{j + 1}_Unlocked", 0) == 1 ? true : false;

            if (i == 0 && j == 0) // 첫번째 스테이지일 경우
                themes[i].stages[j].isUnlocked = true;
        }
    }
}
```

각주 | Sokoland 프로젝트에서 ScriptableObject를 사용하여 에디터에서 편히 수정할 수 있고, 더욱 더 다양화된 옵션을 제공할 수 있습니다.

6 게임 설정 코드

1. 개요

의도 | 4-2에서 만든 INIParser 클래스를 사용해 게임 설정 데이터를 한 곳에서 관리할 수 있도록 하였습니다.

방식 | Application.persistentDataPath 경로에 게임 설정을 읽기/저장하며 그 값을 static 변수로 선언하여 필요한 클래스에서 바로 참조할 수 있도록 하였습니다.

효과 | 게임 재시작 시에도 게임 설정이 유지됩니다.

전체 코드: 바로가기

게임 설정 데이터

```
resolutionIndex=0
isFullscreen=False
BGMVolume=0
SFXVolume=9
```

저장 코드

```
public static void SaveSettings()
{
    var iniData = new Dictionary<string, string>
    {
        { "resolutionIndex", ResolutionIndex.ToString() },
        { "isFullscreen", IsFullScreen.ToString() },
        { "BGMVolume", BGMVolume.ToString() },
        { "SFXVolume", SFXVolume.ToString() },
    };

    INIParser.WriteINI(filePath, iniData);
}
```

읽기 코드

```
public static void LoadSettings()
{
    var iniData = INIParser.ReadINI(filePath);

    if (iniData.TryGetValue("resolutionIndex", out var resolutionIndexStr) &&
        int.TryParse(resolutionIndexStr, out var resolutionIndex))
    {
        ResolutionIndex = resolutionIndex;
    }

    if (iniData.TryGetValue("isFullscreen", out var isFullscreenStr) &&
        bool.TryParse(isFullscreenStr, out var isFullscreen))
    {
        IsFullScreen = isFullscreen;
    }

    if (iniData.TryGetValue("BGMVolume", out var bgmVolumeStr) &&
        int.TryParse(bgmVolumeStr, out var bgmVolume))
    {
        BGMVolume = bgmVolume;
    }

    if (iniData.TryGetValue("SFXVolume", out var sfxVolumeStr) &&
        int.TryParse(sfxVolumeStr, out var sfxVolume))
    {
        SFXVolume = sfxVolume;
    }
}
```

7 매니저 코드

1. GameManager 클래스

의도 | 스테이지 전반적으로 관리하는 클래스로 GameManager보다는 LevelHandler 느낌으로 제작하였습니다. 모든 플레이어가 로드 된 이후 게임이 시작하는 로직, 승리, 패배, 일시 정지 등 스테이지 구성에 필요한 함수를 넣으려고 하였습니다.

방식 | DontDestroyOnLoad()를 사용하지 않는 단순 Singleton으로 구현하였으며, 모든 이벤트(패배, 승리 등)를 RPC를 사용해 모든 클라이언트에 동기화합니다. 또한 각종 매니저 인스턴스를 포함하고 있어 외부에서 쉽게 접근할 수 있도록 하였습니다.

효과 | 스테이지 초기화부터 클리어까지 GameManager 클래스에서 관리 및 호출 할 수 있습니다.

전체 코드: 바로가기

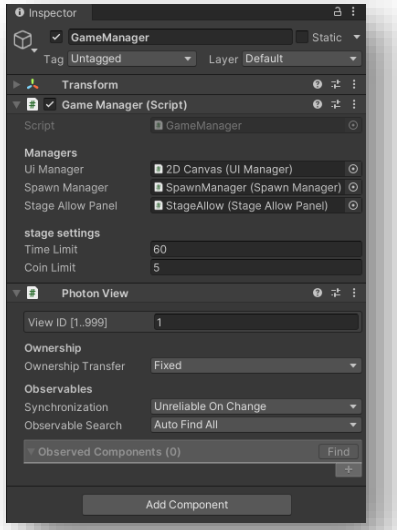
RPC 함수

```
/// <summary>
/// 플레이어가 로드 될 시, 호출되는 RPC 이벤트
/// </summary>
[PunRPC]
private void RPC_PlayerLoaded()
{
    loadedPlayerCount++;

    Debug.Log($"Player Loaded:
{loadedPlayerCount}/{RoomManager.Instance.maxPlayerCount}");

    if (loadedPlayerCount >= RoomManager.Instance.maxPlayerCount)
    {
        photonView.RPC("RPC_AllPlayersLoaded", RpcTarget.AllBuffered);
    }
}

/// <summary>
/// 모든 플레이어가 로드되었을 때, 호출되는 RPC 이벤트
/// </summary>
[PunRPC]
private void RPC_AllPlayersLoaded()
{
    stageAllowPanel.gameObject.SetActive(true);

    Debug.Log("All players loaded. Stage allow panel activated.");

    foreach (var player in PhotonNetwork.PlayerList)
    {
        ExitGames.Client.Photon.Hashtable props = new
ExitGames.Client.Photon.Hashtable
        {
            { "SceneLoaded", null }
        };
        player.SetCustomProperties(props);
    }
}
```

재시작 함수

```
/// <summary>
/// 스테이지 재시작 시, 호출되는 메서드
/// </summary>
public void RestartStage()
{
    switch (RoomManager.Instance.playmode)
    {
        case Playmode.Multi:
            photonView.RPC("RPC_RestartStage", RpcTarget.AllBuffered);
            break;
        case Playmode.Single:
            RPC_RestartStage();
            break;
    }
}

/// <summary>
/// 스테이지 재시작 시, 모든 클라이언트에게 호출되는 RPC 이벤트
/// </summary>
[PunRPC]
private void RPC_RestartStage()
{
    InitailizeUI();

    spawnManager.ResetAll();

    playtime = 0.0f;
    earnedCoin = 0;
    isPause = false;

    uiManager.SetEarnedCoinText(earnedCoin);
    uiManager.SetTimeText(playtime);
}
```

GameManager 인스펙터



2. AudioManager 클래스

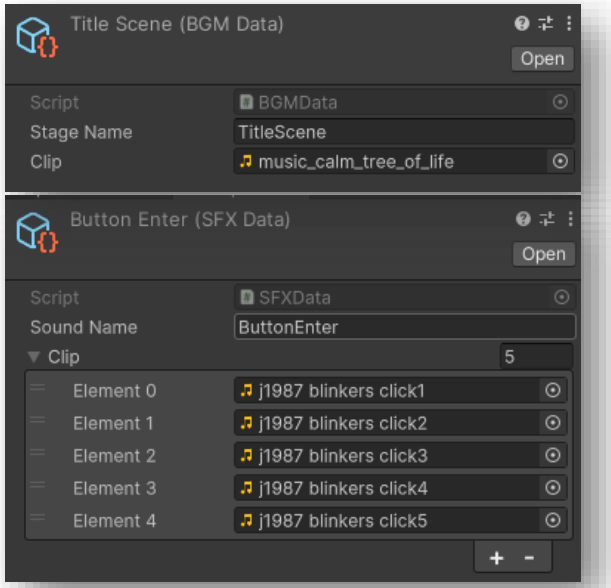
의도 | BGM, SFX와 같은 오디오를 전역적으로 관리하기 위해 제작하였습니다.

방식 | Singleton 패턴을 사용해 오디오 관리 인스턴스를 생성하며, SFX는 오브젝트 풀링 기법을 사용해 구현하였습니다. 또한 ScriptableObject를 사용해 BGM, SFX를 딕셔너리 형태로 간편하게 사용합니다.

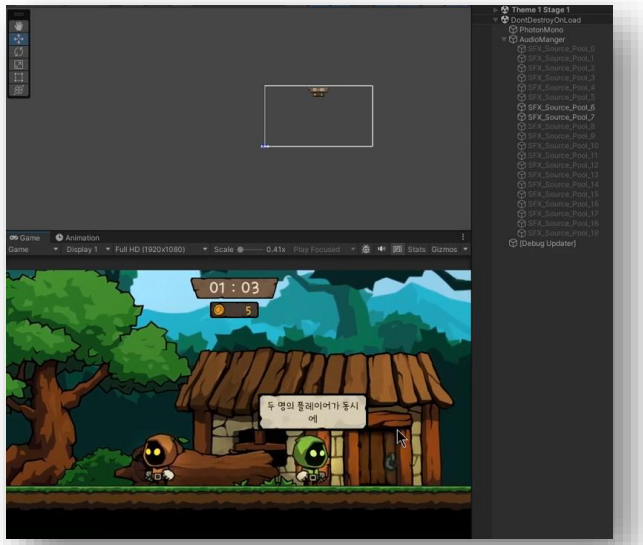
효과 | 스크립트를 통해 오디오를 쉽게 제어할 수 있습니다.

전체 코드: 바로가기

배경, 효과음 데이터



오브젝트 풀링



SFX 실행 코드

```
        /// <summary>
        /// SFX 재생
        /// </summary>
        /// <param name="name">재생할 SFX 이름</param>
        public void PlaySFX(string name)
        {
            if (!sfxs.ContainsKey(name) || sfxs[name] == null)
            {
#if UNITY_EDITOR
                Debug.LogWarning($"wrong Sound FX: {name.ToString()}");
#endif
                return;
            }

            AudioSource source = GetPooledSFXSource();

            var sfx = sfxs[name];
            source.clip = sfx[Random.Range(0, sfx.Length)];
            source.gameObject.SetActive(true);
            source.volume = sfxVolume / 100.0f;
            source.Play();

            StartCoroutine(ReturnSFXAfterPlay(source));
        }
```

8 유니티 툴 활용 사례

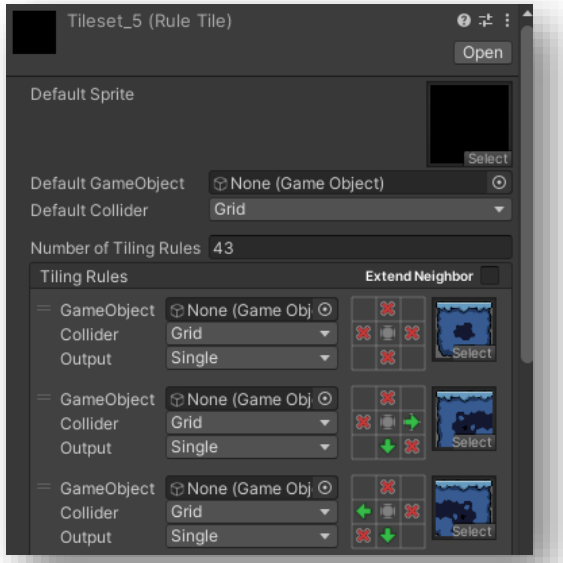
1. 룰 타일

의도 | 맵 디자인 시, 불편함을 최소화하고 시간을 단축시키기 위해 룰 타일을 적용했습니다.

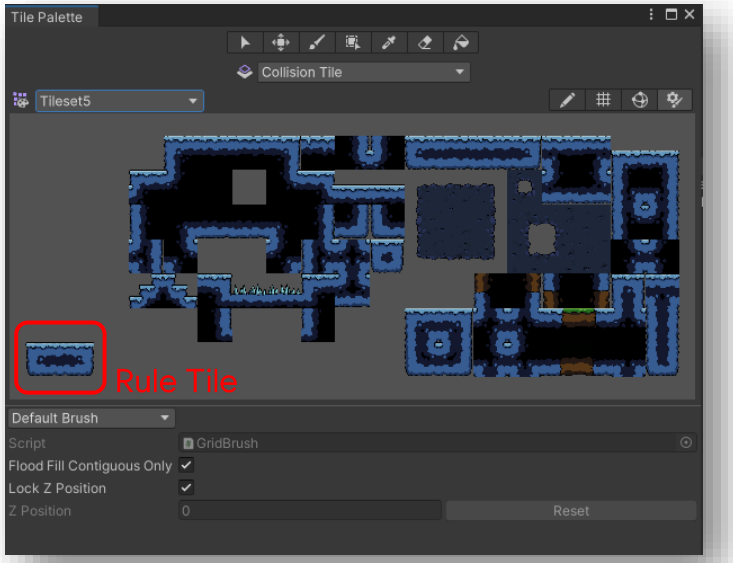
방식 | 룰 타일에 43개의 룰을 추가하여 작업하였습니다.

효과 | 맵 작업 시, 불편함을 크게 느끼지 않고 작업할 수 있었습니다.

룰 타일 인스펙터



팔레트 구성



2. 커스텀 에디터

의도 | 불편했던 작업들을 유니티 에디터 상에서 쉽게 처리할 수 있도록 작업하였습니다.

방식 | UnityEngine.Editor, UnityEngine.EditorWindow를 상속받는 클래스를 제작 후, ShowWindow(), OnGUI() 등 메서드를 사용해 내부 요소들을 커스텀하였습니다.

효과 | 스크린 샷, 태그 지정, 커스텀 버튼(호버, 클릭 시 소리 나오게 하는 UI 요소)을 제작하여 쉽게 작업할 수 있었습니다.

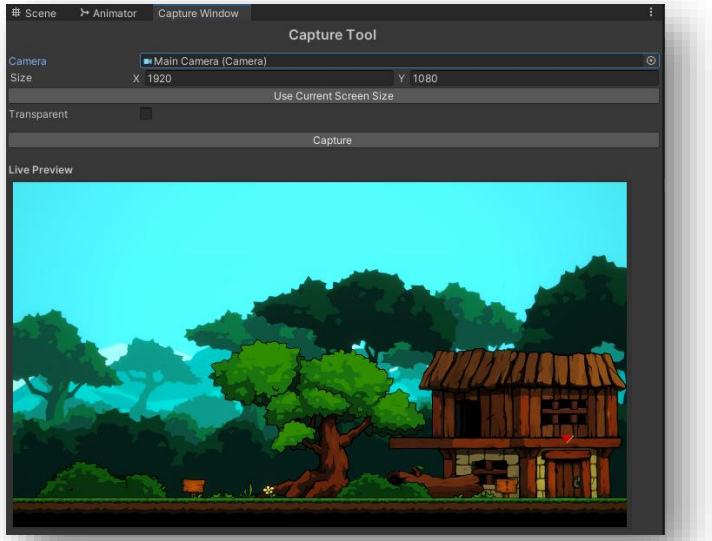
전체 코드: 바로가기

에디터 활용 코드

```
public override void OnInspectorGUI()
{
    DrawDefaultInspector();

    InteractableCollision triggerScript = (InteractableCollision)target;
    string[] allTags = UnityEditorInternal.InternalEditorUtility.tags;

    EditorGUILayout.LabelField("Select Tags:", EditorStyles.boldLabel);

    for (int i = 0; i < allTags.Length; i++)
    {
        tagCh
        if (t
        {
            t
        }
        else
        {
            t
        }
    }

    if (GUI.c
    {
        Edito
    }
}
```

```
public class SoundButtonEditor : ButtonEditor
{
    SerializedProperty buttonType;

    protected override void OnEnable()
    {
        base.OnEnable();
        buttonType = serializedObject.FindProperty("buttonType");
    }

    public override void OnInspectorGUI()
    {
        base.OnInspectorGUI();
        serializedObject.Update();
        EditorGUILayout.PropertyField(buttonType);
        serializedObject.ApplyModifiedProperties();
    }
}
```

사용 예시 (스크린 샷)



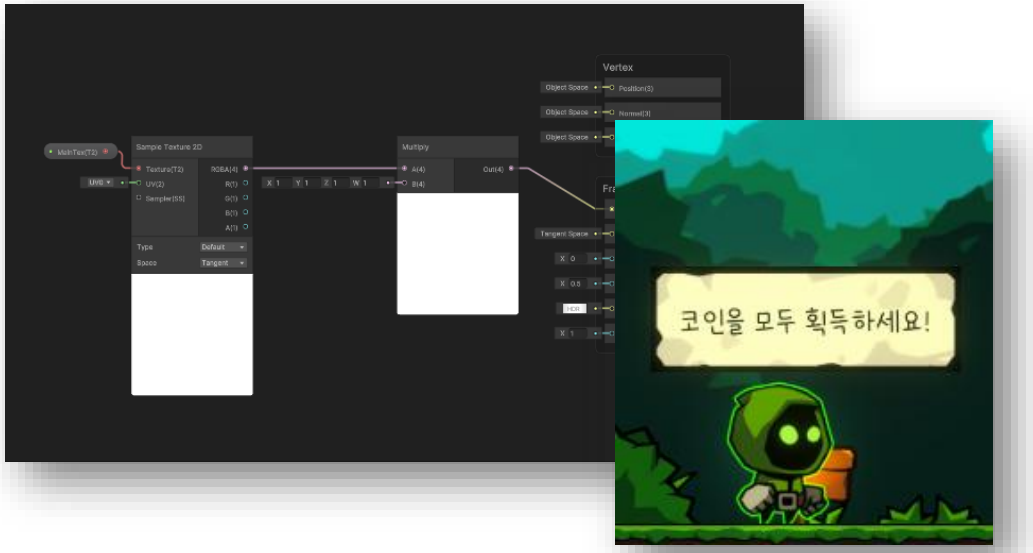
3. 셰이더 그래프

의도 | HDR, Outline 등 간단한 여러 셰이더가 필요하여 셰이더 그래프를 활용해 셰이더를 제작하였습니다.

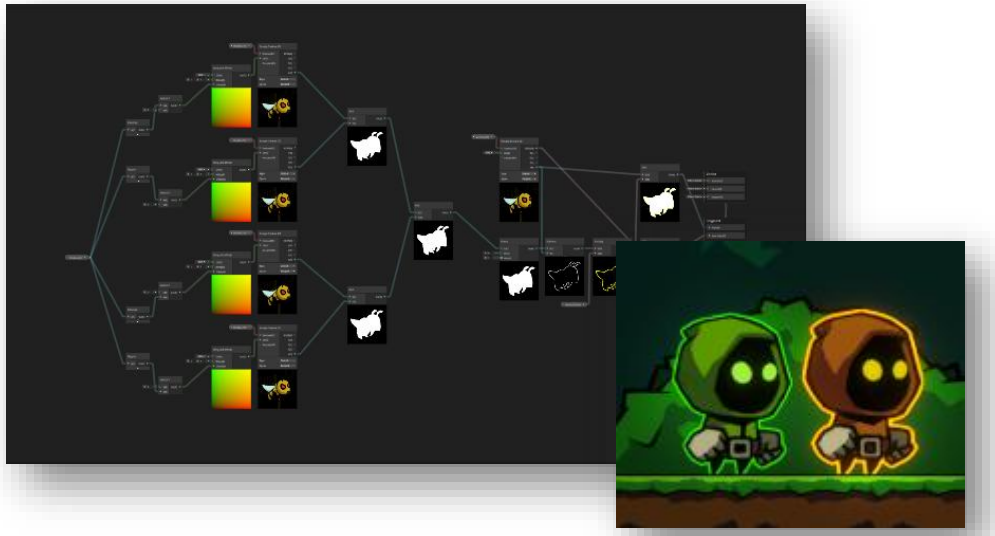
방식 | Unity Shader Graph를 사용해 제작하였습니다.

효과 | 여러 셰이더를 효과적이고 빠르게 구현할 수 있었습니다.

표지판 셰이더



아웃라인 셰이더



9 후기

‘Twin Travelers’ 를 개발하기 이전 해에는 산업기능요원으로 취업하게 된 회사에 도움이 되기 위해 Vue.js, Spring 등 여러 웹 관련 기술을 스터디를 진행하고 있었습니다. 이 즈음 저는 ‘올림픽공원 미세먼지 지도’ 서비스를 개발할 정도로 많은 성장을 일궈냈습니다.

한편, 게임개발에 대한 미련이 계속 남아있었기에 기존에 제작하였던 프로젝트 중 하나를 리마스터 하면서 다시 감을 되찾으려고 하였습니다. 그 작품이 ‘Twin Travelers’ 입니다. ‘Twin Travelers’를 개발하며 감을 다시 잡고 여러 기술을 접목하려 노력하였습니다.

제가 해왔던 게임개발 프로젝트 중 가장 거대한 프로젝트였으며, 여러가지 새로운 기술을 사용했습니다. 가장 기억에 남는 것은 Photon PUN 2 에셋을 사용해 네트워크 시스템을 어떻게든 굴러가도록 구현해냈다는 것입니다. 전 프로젝트인 Low Poly Battle Field 프로젝트에서는 PUN2, Mirror, Fusion을 모두 사용해봐도 힘들었던 구현을 이제는 할 수 있다는 점에서 제 스스로 많이 기뻤습니다.

다만 아쉬운 점이 있다면, UniTask, UniRx, LINQ to GameObject, DOTween 같은 유니티 개발에 있어서 아주 유용한 패키지들을 사용하지 못했다는 것입니다. 그렇기에 다음 프로젝트는 UniTask, UniRx를 사용하기로 마음 먹었고, 이벤트 기반의 프로그래밍을 통해 성능 최적화 및 가독성 향상을 꾀할 예정입니다.

매너리즘에 빠졌던 제가 이번 게임의 개발을 본격적으로 시작하면서 매너리즘의 늪에서 탈출하고 더욱 더 열정을 펼칠 수 있다는 게 너무 신기했습니다. 앞으로도 이 열정을 바탕으로 새로운 도전들을 이어나가도록 하겠습니다.

Twin Travelers

포트폴리오에 대한 모든 의견을 존중하며, 오늘도 사랑스런 하루 보내세요!

Email: lsnan421@naver.com
깃허브: 바로가기
유튜브: 바로가기
노　션 : 바로가기

작성 일지: 2025. 03. 17