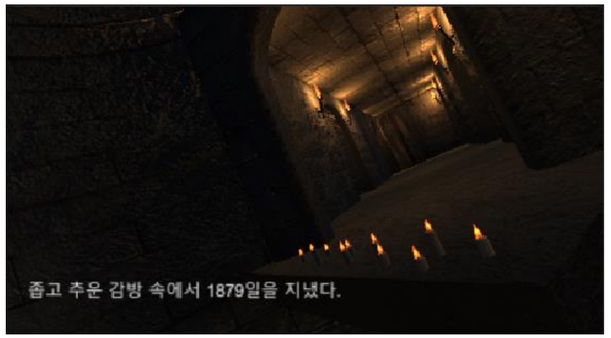
BreakOut : The Prison Escape

by Unity

시작 메뉴

게임의 목표

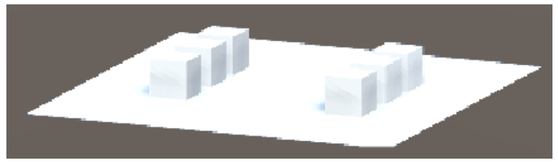


시네머신 오프닝

시네머신을 이용한 카메라 오프닝

3D 교도소 아이템과 동료 수집

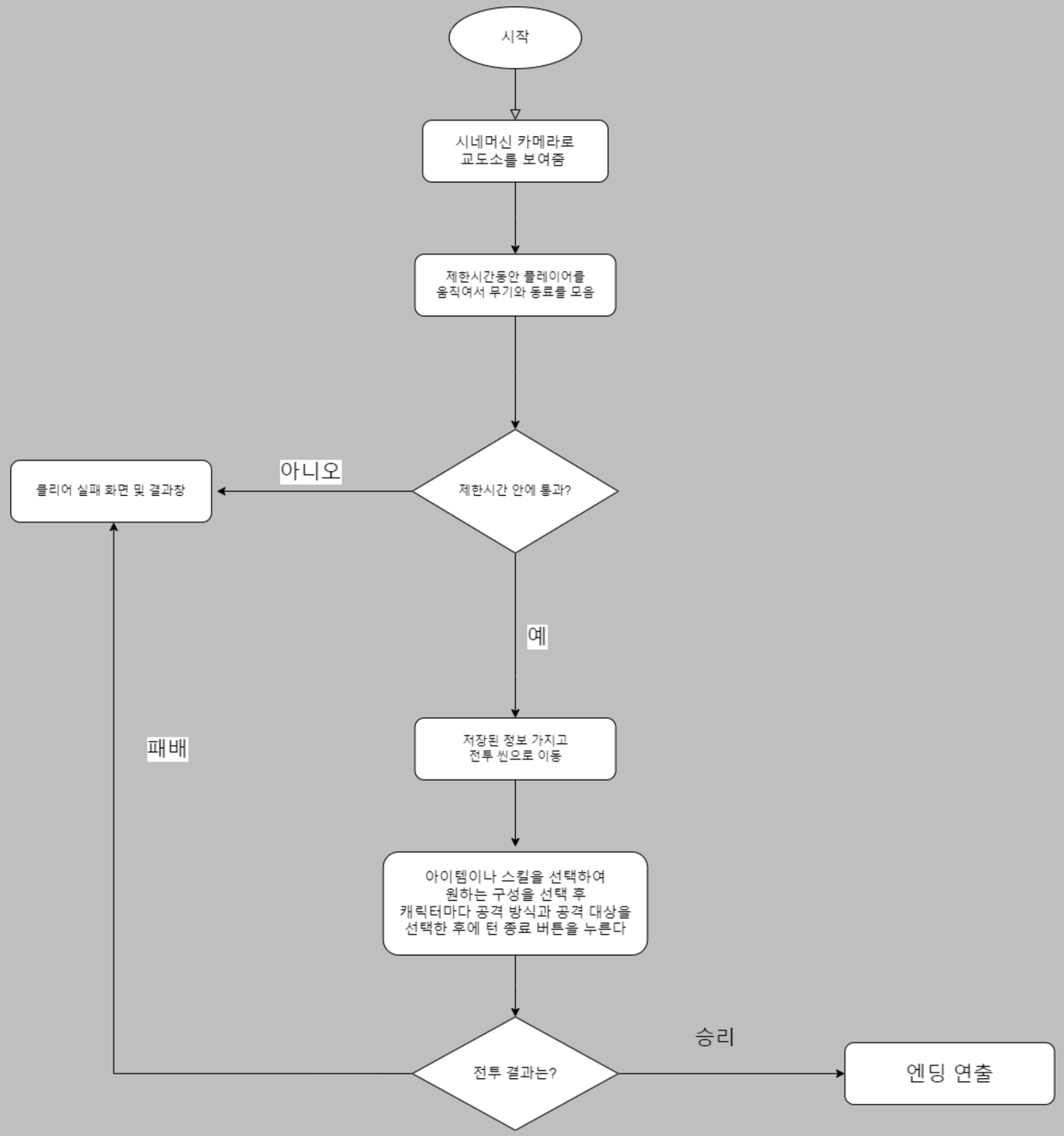
3D 감옥의 수집 요소 데이터



2.5D TRPG

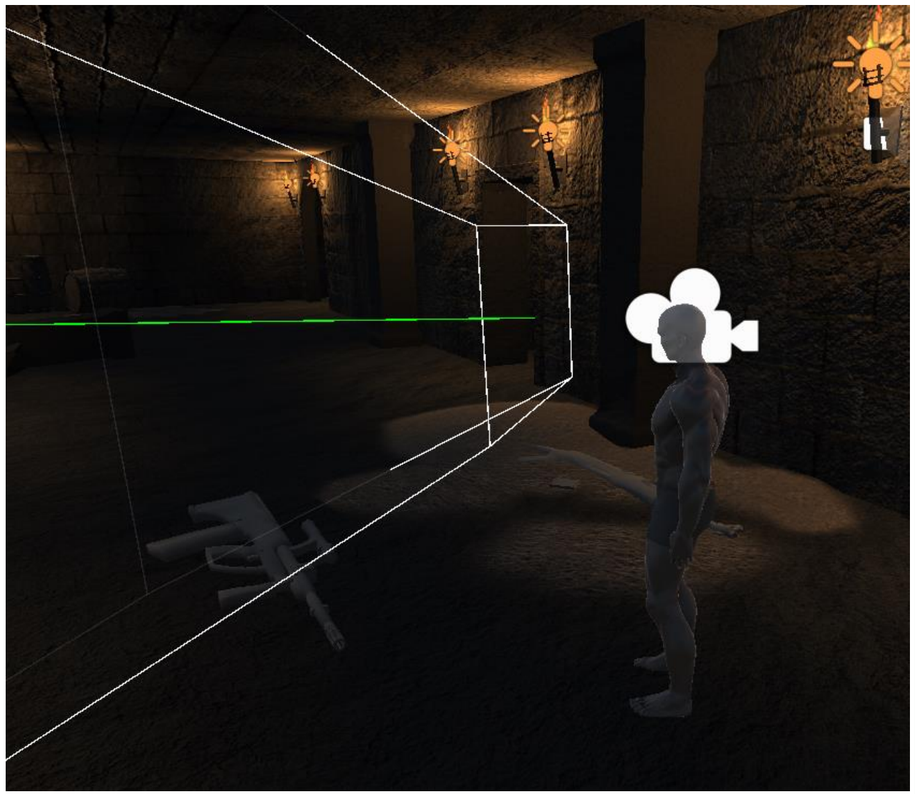
데이터의 요소들을 사용하여 전투

플로우차트


시작
시네머신 카메라로
교도소를 보여줌
제한시간동안 플레이어를
움직여서 무기와 동료를 모음
제한시간 안에 통과?
아니오
클리어 실패 화면 및 결과창
예
저장된 정보 가지고
전투 씬으로 이동
아이템이나 스킬을 선택하여
원하는 구성을 선택 후
캐릭터마다 공격 방식과 공격 대상을
선택한 후에 턴 종료 버튼을 누른다
전투 결과는?
패배
승리
엔딩 연출

목차

다룰 주제



1.Player

현실적인 움직임 요소 구현

PlayerMovement

```
public float speed;      // 캐릭터 움직임 스피드.

[SerializeField]
float jumpSpeed; // 캐릭터 점프 힘.
[SerializeField]
float gravity;    // 캐릭터에게 작용하는 중력.

[SerializeField]
Transform playerCam;  // 1인칭 캠
[SerializeField]
Transform playerCapsule;  // 캐릭터의 전방을 가리키는 오브젝트

[SerializeField]
CharacterController controller; // 현재 캐릭터가 가지고있는 캐릭터 컨트롤러 콜라이더.

Vector3 MoveDir;                // 캐릭터의 움직이는 방향.
Vector3 crouchScale = new Vector3(1, 0.5f, 1);  // 웅크릴 때 크기 변경할 비율
Vector3 playerScale;  // 일어설 때 변경되는 플레이어 크기

// Assingables
float horizontal, vertical; // 수평, 수직 입력값
float desiredX;

bool jumping;

// Rotation and look
float xRotation;  // x축 회전값
float sensitivity = 50f;  // 마우스 감도
float sensMultiplier = 1f;  // 감도 배수
```

플레이어 기본 자료형 설정값입니다.
Rigidbody는 계단을 오를 때 추가 연산이 생기는 문제가 발생하여 **CharacterController**를 사용하였습니다.

키 입력값과 캠의 방향을 기준으로 1인칭 움직임을 구현하였습니다.

```
void CharacterMove()
{
    // 현재 캐릭터가 땅에 있는가?
    if (controller.isGrounded)
    {

        // 위, 아래 움직임 셋팅.
        MoveDir = new Vector3(horizontal, 0, vertical);

        // 벡터를 로컬 좌표계 기준에서 월드 좌표계 기준으로 변환한다.
        MoveDir = transform.TransformDirection(MoveDir);

        // 스피드 증가.
        MoveDir *= speed;


        // 캐릭터 점프
        if (jumping)
            MoveDir.y = jumpSpeed;
    }

    // 캐릭터에 중력 적용.
    MoveDir.y -= gravity * Time.deltaTime;

    // 캐릭터 움직임.
    controller.Move(MoveDir * Time.deltaTime);
}
```

플레이어의 벡터를 순서대로 작동하게 하였습니다.

#땅에 닿아있을 때만 움직일 수 있습니다.
입력값 x, z
월드좌표계
속도값
점프 y

#항상 유지되는 것
중력값
MoveDir 벡터 따라서 움직이기

```
void Look()
{
    float mouseX = Input.GetAxis("Mouse X") * sensitivity * Time.fixedDeltaTime * sensMultiplier;  // 마우스의 X 축 입력 값을 받아옵니다.
    float mouseY = Input.GetAxis("Mouse Y") * sensitivity * Time.fixedDeltaTime * sensMultiplier;  // 마우스의 Y 축 입력 값을 받아옵니다.

    Vector3 rot = playerCam.transform.localRotation.eulerAngles;  // 현재 카메라의 회전 값을 가져옵니다.
    desiredX = rot.y + mouseX;  // 카메라의 Y 회전 값을 마우스 입력에 따라 조정합니다.

    xRotation -= mouseY;  // 카메라의 X 회전 값을 마우스 입력에 따라 조정합니다.
    xRotation = Mathf.Clamp(xRotation, -90f, 90f);  // X 회전 값을 최소 -90도부터 최대 90도까지로 제한합니다.

    playerCam.transform.localRotation = Quaternion.Euler(xRotation, desiredX, 0);  // 카메라의 회전 값을 설정합니다.
    playerCapsule.transform.localRotation = Quaternion.Euler(0, desiredX, 0);  // 플레이어의 회전 값을 설정합니다.
}
```

```
void StartCrouch()
{
    transform.localScale = crouchScale;  // 플레이어 크기 조정
    speed /= 2f;
}

void StopCrouch()
{
    transform.localScale = playerScale;  // 플레이어 크기 복원
    transform.position = new Vector3(transform.position.x, transform.position.y + 0.5f, transform.position.z);  // 위치 조정
    speed *= 2f;
}
```

입력값의 감도와 감도 배수를 곱하여 mouseX,mouseY에 할당합니다.

카메라의 회전 벡터값을 마우스 입력에 따라 X,Y값을 조정합니다.

카메라 위,아래회전 값을 90도 제한합니다.

플레이어 회전은 Y값만 조정합니다.

플레이어가 Ctrl을 눌렀을 때는 몸 숙이기로 스케일과 속도를 조절하였습니다.

PlayerSpeed

```
void MyInput()
{
    if (Input.GetKey(KeyCode.LeftShift))
    {
        if (!tire)
        {
            running = true;
        }
        if (startRunTime > runTimer && running) //지쳤을 때
        {
            running = false;

            StartCoroutine(Tired());
        }
    }

    if (Input.GetKeyUp(KeyCode.LeftShift))
    {
        running = false;
    }
}
```

```
void RunTime()
{
    if (running)
    {
        IsRunning();
        startRunTime += Time.deltaTime;
    }
    else
    {
        IsWalking();
        startRunTime = 0f;
    }

    // 뛰기와 걷기 시간에 따라 속도 조절
    playerMovement.speed = Mathf.Lerp(walkSpeed, runSpeed, startRunTime / runFastTimer);
}
```

```
IEnumerator Tired()
{
    tire = true;
    SoundManager.Instance.PlaySound(tiredlipName, bigSound);
    yield return new WaitForSeconds(walkTimer);

    tire = false;
}
```

전체적인 뛰고 걷는 로직입니다.
현실적인 게임성 경험을 바탕으로 구현하였습니다.

Mathf.Lerp를 사용해 플레이어의 속도를 walkSpeed부터 runSpeed까지 점점 증가하는 식으로 속도 조절이 됩니다.

또한 뛰는 시간이 일정량에 도달했을 때 Tired함수를 호출하여 일시적으로 호흡사운드를 재생합니다.

```
if (Input.GetKey(KeyCode.W) || Input.GetKey(KeyCode.S) || Input.GetKey(KeyCode.A) || Input.GetKey(KeyCode.D))
{
    AnimationState.ChangeAnimationState("player_Walking");
    if (controller.isGrounded)
    {
        SoundTimer(smallSound, 0.6f);
    }
}
else
{
    AnimationState.ChangeAnimationState("player_Idle");
}
```

```
void IsLandSound()
{

    RaycastHit hit;
    float rayLength = 2f;

    if (Physics.Raycast(transform.position, Vector3.down, out hit, rayLength))
    {
        if (hit.collider != null && Input.GetKeyDown(KeyCode.Space))
        {
            isFly = true;
        }
    }


    if (isFly && controller.isGrounded)
    {
        SoundManager.Instance.PlaySound(landClipName, bigSound);
        isFly = false;
    }
}
```

```
float timer = 0f;
void SoundTimer(float volume, float interval) //간격으로 측정
{
    timer += Time.deltaTime;

    if (timer >= interval)
    {
        SoundManager.Instance.PlayRandomSound(walkClipName, volume);
        timer = 0f;
    }
}
```

전체적인 플레이어의 걷는 소리와 착지 소리입니다.

플레이어의 Vector3.down쪽에 레이캐스트로 착지체크를 합니다.

걷는 소리는 랜덤, 착지 소리는 일정합니다.

소리는 일정 시간 간격으로 한번씩 재생합니다.

2.Camera

오프닝 위주로 구현되었습니다.

CameraMove

```
void Start()
{
    mainCamera.nearClipPlane = 1f; // 값을 원하는 거리로 조정

    CameraSet(cameraParent, playerEye);
    CameraSet(mainCamera.transform, playerEye);
    mainCamera.gameObject.transform.SetParent(cameraParent);


}
```

```
void Update()
{
    cameraParent.transform.position = playerEye.transform.position;

    if (Input.GetKey(KeyCode.LeftShift))
    {
        CameraMoving();
    }
    else
    {
        CameraSet(mainCamera.transform, playerEye);
    }
}
```

```
float delta = 1f;
float elapsedTime = 0f;
float startTime = 1f;

public void CameraMoving()
{
    elapsedTime += Time.deltaTime;
    if (elapsedTime <= startTime)
    {
        Vector3 vector3 = transform.position;
        vector3.y += Time.deltaTime * delta;
        mainCamera.transform.position = vector3;
    }
    else
    {
        delta *= -1;
        elapsedTime = 0f;
    }
}

public void CameraSet(Transform trans1, Transform trans2)
{
    trans1.transform.position = trans2.transform.position;
}
```

전체적인 플레이어 눈에 달린 카메라 움직임입니다.

1인칭이므로 몸 안을 가릴 수 있도록 nearClipPlane을 초기설정하였습니다.

뛰는 동안에는 카메라가 상하로 조금씩 반복하여 움직임입니다.

카메라의 위치를 고정하기 위해 플레이어의 눈에 자식으로 설정하였습니다.

이는 카메라 초기위치 설정입니다.

CameraRays

```
void LateUpdate()
{
    ray = mainCamera.ViewportPointToRay(new Vector3 (0.5f, 0.5f, 0.5f));

    if (Physics.Raycast(ray, out rayHit, MAX_RAY_DISTANCE))
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            ObjectRecognition();
        }
        Debug.DrawLine(ray.origin, rayHit.point, Color.green);
    }
    else
    {
        Debug.DrawRay(ray.origin, ray.direction * 100, Color.red);
    }
}
```

```
public void GetWeapon()
{
    Debug.Log(rayHit.transform.name + "획득");

    DataManager.Instance.playerData.items.Add(rayHit.transform.gameObject.GetComponent<ObjData>().ID);
    DataManager.Instance.SavePlayerDataToJson(DataManager.Instance.playerData);
    DataManager.Instance.LoadPlayerDataFromJson(DataManager.Instance.playerData);
    Debug.Log(DataManager.Instance.playerData.items.Count);
    Destroy(rayHit.transform.gameObject);
}
```

```
void ObjectRecognition()
{
    if (rayHit.transform.CompareTag("Weapon"))
    {
        GetWeapon();

    }
    else if (rayHit.transform.CompareTag("NPC"))
    {
        TalkWriting();
    }
}
```

```
public void TalkWriting()
{
    while (textString.Length > textNumber)
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            print(startMessage);
            startMessage.DOText(textString[textNumber], 3f);
            textNumber++;
        }
    }

    textNumber = 0;

}
```

전체적인 E키를 눌렀을 때
무기나 대화를 호출하는 함수입니다.

무기는 Json에 저장하였습니다.

대화는 DoTween의 DoText를
사용하였습니다.

CameraSwitcher - Cinemachine

```
void SetPositionCam1(int i)
{
    cam1.transform.SetPositionAndRotation(camPositions[i], Quaternion.Euler(camRotations[i]));
}

void SetPositionCam2(int j)
{
    cam2.transform.SetPositionAndRotation(camPositions[j], Quaternion.Euler(camRotations[j]));
}

void ChangeCam1()
{
    cam1.Priority = 2;
    cam2.Priority = 1;
}

void ChangeCam2()
{
    cam1.Priority = 1;
    cam2.Priority = 2;
}
```

```
IEnumerator CamRotate(CinemachineVirtualCamera cam, Vector3 targetRotation)
{
    yield return new WaitForSeconds(CameraTransformTime);

    float duration = 1f;
    Quaternion start = cam.transform.rotation;
    Quaternion target = Quaternion.Euler(targetRotation);
    float elapsedTime = 0f;

    while (elapsedTime < duration)
    {
        cam.transform.rotation = Quaternion.Lerp(start, target, elapsedTime);
        elapsedTime += Time.deltaTime;
        yield return null;
    }

    cam.transform.rotation = target;
}
```

```
if (i == 4)
{
    yield return new WaitForSeconds(3f);
    StartCoroutine(EndOpening());
    break;
}

yield return StartCoroutine(CamRotate(camValue, camRotateValues[currentPos]));

if (camValue == cam1)
{
    SetPositionCam2(currentPos);
}
else
{
    SetPositionCam1(currentPos);
}
currentPos++;
```

전체적인 시네머신을 이용한 오프닝 연출입니다.

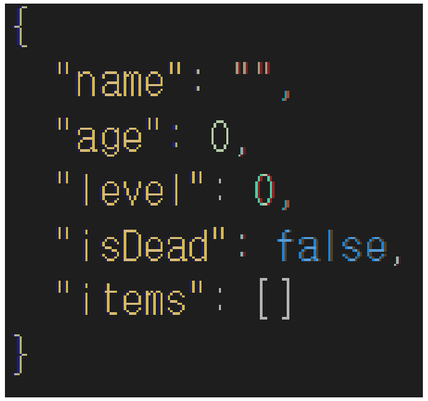
Priority순서에 따라 코루틴을 적극 활용하여 자연스러운 카메라 움직임을 구현하였습니다.

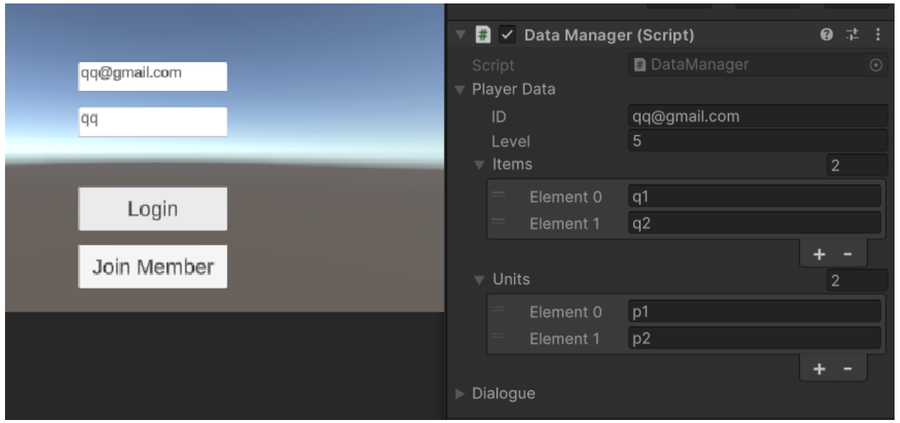
CameraTransition - DoTween

```
using DG.Tweening;
```

```
public void StartScreen()
{
    blackGround.enabled = true;
    StartCoroutine(StartBlackScreenEffect(new Color(0f, 0f, 0f, 0f), 3f));
}

public void EndScreen()
{
    StartCoroutine(StartBlackScreenEffect(new Color(0f, 0f, 0f, 1f), 3f));
}

IEnumerator StartBlackScreenEffect(Color color, float duration)
{
    blackGround.DOColor(color, duration);

    yield return new WaitForSeconds(duration);
}


public IEnumerator TextWriting()
{
    for (int i = 0; i < textString.Length; i++)
    {
        startMessage.DOText(textString[i], 3f);

        yield return new WaitForSeconds(6f);
        startMessage.text = "";
    }

    startMessage.text = "";
}
```

DOTween을 사용한 화면 어두워짐과
밝아짐 연출입니다.





3.Data

단순 DB저장보다 안전한
Firebase 공식 사이트를 사용한
Json과 서버 저장 로직입니다.

DataManager – Json

```
public class DataManager : Singleton_DontDestroy<DataManager>
{
    /**PlayerController로서 필요한 변수와 메서드가 있다 **/
    public PlayerData playerData;
    public Dialogue dialogue;
```

```
void Awake()
{
    Dialogue dial = new Dialogue(true);
    SavePlayerDataToJson(dial);
    LoadPlayerDataFromJson(dial);
    Debug.Log("dial:" + dialogue.contexts.Length);
}
```

```
[ContextMenu("To Json Data")] // 컴포넌트 메뉴에 아래 함수를 호출하는 To Json Data 라는 명령어가 생성됨
public void SavePlayerDataToJson<T>(T data)
{
    // ToJson을 사용하면 JSON형태로 포멧팅된 문자열이 생성된다
    string jsonData = JsonUtility.ToJson(data, true);
    // 데이터를 저장할 경로 지정
    string fileName = data.GetType().Name;
    string path = Path.Combine(Application.persistentDataPath, fileName + ".json");
    // 파일 생성 및 저장
    File.WriteAllText(path, jsonData);
}
[ContextMenu("From Json Data")]
public void LoadPlayerDataFromJson<T>(T dataType)
{
    // 데이터를 불러올 경로 지정
    string fileName = dataType.GetType().Name;
    string path = Path.Combine(Application.persistentDataPath, fileName + ".json");
    // 파일의 텍스트를 string으로 저장
    string jsonData = File.ReadAllText(path);
    // 이 Json데이터를 역직렬화하여 playerData에 넣어줌
    dataType = JsonUtility.FromJson<T>(jsonData);
}
```

```
[System.Serializable]
public class PlayerData
{
    public string ID;
    public int level;
    public List<string> items;
    public List<string> units;
}
[System.Serializable]
public class Dialogue
{
    public string name;
    public string[] contexts;

    public Dialogue(bool isSet)
    {
        if (isSet)
        {
            contexts = new string[] { "First Line", "Second Line", "Third Line" };
        }
    }
}
```

전체적인 Json 데이터 생성과 저장입니다.

Singleton_DontDestroy를 상속받아 싱글톤으로 구현되었습니다.

PlayerData와 Dialogue 인스턴스를 보유합니다.

Awake에서 Dialogue를 생성하고 JSON으로 저장 및 불러오는 기능이 호출됩니다.

제네릭 타입 T를 사용하여 임의의 데이터를 JSON으로 저장합니다.
JsonUtility.ToJson을 이용하여 데이터를 JSON 문자열로 변환합니다.

데이터의 타입 이름을 파일 이름으로 사용하여 지속적인 데이터 경로에 저장합니다.

제네릭 타입 T를 사용하여 JSON 파일을 읽어 **해당 데이터 타입으로 역직렬화합니다.**
JsonUtility.FromJson을 이용하여 **JSON 문자열을 데이터 객체로** 변환합니다.

게임 플레이어 정보를 저장하는 클래스로, ID, 레벨, 아이템 목록, 유닛 목록 등의 Field를 포함합니다.

생성자를 통해 초기 대화 내용을 설정할 수 있습니다.

CFirebase

```
if (child.Key.Equals("items"))
{
    string[] items = ((string)child.Value).Split(' ', StringSplitOptions.RemoveEmptyEntries);

    DataManager.Instance.playerData.items = items.ToList();

}
```

```
if (child.Key.Equals("level"))
{
    DataManager.Instance.playerData.level = int.Parse((string)child.Value);
```

```
if (child.Value.Equals(email))
{
    DataManager.Instance.playerData.ID = email;
    Debug.Log(data.Key);
    userNumber = data.Key;
```

```
public void ReadUserData(string email)
{
    FirebaseDatabase.DefaultInstance.RootReference.Child("users")
        .GetValueAsync().ContinueWithOnMainThread(task =>
        {
            if (task.IsFaulted)
            {
                Debug.Log(task.IsFaulted);
                // Handle the error...
            }
            else if (task.IsCompleted)
            {

                DataSnapshot snapshot = task.Result;
                // Do something with snapshot...

                Debug.Log("task.IsCompleted");



                foreach (var data in snapshot.Children)
                {
                    Debug.Log("Email:" + email);


                    foreach (var child in data.Children)
                    {
```

Login하면 Firebase 서버의 값을 읽어오는 로직입니다.

편하고 오류없도록 빈칸을 기준으로 나누었습니다.

Email이 맞다면 자식들은 모두 Json Data에 저장되는 로직입니다.

비동기적으로 실행됩니다.

```
public async void CountUser()
{
    await FirebaseDatabase.DefaultInstance.GetReference("users")
        .GetValueAsync().ContinueWithOnMainThread(task =>
        {
            if (task.IsFaulted)
            {
                Debug.Log(task.IsFaulted);
                // Handle the error...
            }
            else if (task.IsCompleted)
            {
                DataSnapshot snapshot = task.Result;
                Debug.Log(snapshot.ChildrenCount);

                if (snapshot.Children == null)
                {
                    join.countNumber = 0;
                    Debug.Log("join.countNumber = 0;");
                    return;
                }

                join.countNumber = snapshot.ChildrenCount;
```

```
public void WriteUserData(string userID, string useremail)
{
    Debug.Log("WriteUserData");
    m_Reference.Child("users").Child(userID).Child("username").SetValueAsync(useremail);
}
```

```
public void SaveChildInfo(string objKey, string objValue)
{
    m_Reference.Child("users").Child(userNumber).Child(objKey).SetValueAsync(objValue);
}
```

서버의 데이터 값들의 개수를 체크해 생성 차례를 검사하는 로직입니다.

또한 서버에 처음 회원가입을 하게되면 순서에 따라 useremail을 새로 생성합니다.

Join Membership

```
using Firebase.Auth; // 계정인증기능 사용
```

```
[SerializeField]
TMP_Text m_ID;
[SerializeField]
TMP_Text m_Password;

FirebaseAuth auth; // 인증 객체 불러오기

public long countNumber;

public void FinshJoinButton()
{
    if (m_ID != null && m_Password != null)
    {
        auth = FirebaseAuth.DefaultInstance; // 인증 객체 초기화
        Join(m_ID.text, m_Password.text); // 해당 이메일,비밀번호로 가입하기
    }
}
```

```
async Task Join(string email, string password)
{
    // 이메일과 비밀번호로 가입하는 함수
    await auth.CreateUserWithEmailAndPasswordAsync(email, password).ContinueWith(
        async task =>
        {
            if (task.IsCompleted)
            {
                Debug.Log(email + " 로 회원가입 하셨습니다.");

                await firebase.CountUser();
                firebase.WriteUserData(countNumber.ToString(), email);
            }
            else
            {
                Debug.Log("회원가입에 실패하셨습니다.");
            }
        }
    );
}
```

회원가입하는 로직입니다.

FirebaseAuth 객체를 사용하여 계정 인증을 수행하는 auth 변수 countNumber 변수는 사용자 계정이 몇 번째로 생성되었는지를 나타내는 값입니다.

입력된 ID와 Password가 null이 아니라면, Firebase를 통해 계정을 생성하는 Join 메서드를 호출합니다.

async Task함수와 await을 사용하여 가입한 플레이어 길이를 잰 후에 실행되도록 하였습니다.

CreateUserWithEmailAndPasswordAsync 메서드를 사용하여 이메일과 비밀번호로 계정을 생성합니다.

CFirebase 클래스를 사용하여 Firebase와 상호작용합니다.

CountUser 함수를 통해 현재까지 생성된 사용자 수를 가져오고,

WriteUserData 함수로 사용자 데이터를 데이터베이스에 기록합니다.

Login

```
[SerializeField]
CFirebase firebase;

[SerializeField]
TMP_Text m_ID;
[SerializeField]
TMP_Text m_Password;


FirebaseAuth auth;

public void FinshLoginButton()
{
    if (m_ID != null && m_Password != null)
    {
        auth = FirebaseAuth.DefaultInstance;
        Login(m_ID.text, m_Password.text);
    }
}
```

```
void Login(string email, string password)
{
    // 이메일과 비밀번호로 가입하는 함수
    auth.SignInWithEmailAndPasswordAsync(email, password).ContinueWith(
        task => {
            if (task.IsCompleted) //&& !task.IsFaulted && !task.IsCanceled
            {
                Debug.Log(email + " 로 로그인 하셨습니다.");
                firebase.ReadUserData(email);

            }
            else
            {
                Debug.Log("로그인에 실패하셨습니다.");
            }
        }
    );
}
```

로그인하는 로직입니다.

CFirebase 컴포넌트를 참조하는 firebase 변수가 선언되어 있습니다.
사용자가 입력한 ID와 Password를 받아오는
m_ID와 m_Password 변수가 선언하였습니다.

FirebaseAuth 객체를 사용하여 계정 인증을 수행하는
auth 변수를 선언하였습니다.

입력된 ID와 Password가 null이 아니라면, Firebase를 통해 로그인을 시도하는 Login 메서드를 호출합니다.

SignInWithEmailAndPasswordAsync 메서드를 사용하여 이메일과 비밀번호로 로그인을 시도합니다.
로그인이 성공하면 사용자 데이터를 읽어오고, 실패하면 에러 메시지를 출력합니다.
성공한 경우 SettingButtonManager.Instance.LoginButton()을 호출하여 로그인 버튼을 처리합니다.

Generic Singleton

```cpp
public class Singleton<T> : MonoBehaviour where T : Singleton<T>
{
    public static T Instance { get; set; }

    protected virtual void OnAwake() { }
    protected virtual void OnStart() { }

    void Awake()
    {
        if (Instance == null)
        {
            Instance = (T)this;
            OnAwake();
        }
        else
        {
            Destroy(gameObject);
        }
    }

    void Start()
    {
        if (Instance = (T)this)
        {
            OnStart();
        }
    }
```

```cpp
public class Singleton_DontDestroy<T> : MonoBehaviour where T : Singleton_DontDestroy<T>
{
    public static T Instance { get; set; }

    protected virtual void OnAwake() { }
    protected virtual void OnStart() { }

    void Awake()
    {
        if (Instance == null)
        {
            Instance = (T)this;
            OnAwake();

            DontDestroyOnLoad(Instance);
        }
        else
        {
            Destroy(gameObject);
        }
    }

    void Start()
    {
        if (Instance = (T)this)
        {
            OnStart();
        }
    }
```

제네릭으로 만든 싱글톤입니다.

OnAwake, OnStart기능을
첨부하였습니다.

여러 씬에서 데이터매니저를
옮기는데 중심적으로
사용하였습니다.

주요 기능은 인스턴스에 따라
파괴되거나 호출하는 기능입니다.

SoundManager

LoadingManager

4.Assist Quality

블로그 및 공식 자료를
간접인용하였습니다.

SoundManager, AnimationState

```
#region 옵션에서 볼륨조절

[SerializeField]
Slider sliderToBGM;
[SerializeField]
Slider sliderToSFX;

public void SetVolumeBGM()
{
    masterVolumeBGM = sliderToBGM.value;
    bgmPlayer.volume = masterVolumeBGM;
}

public void SetVolumeSFX()
{
    masterVolumeSFX = sliderToSFX.value;
}

#endregion
```

* 해당 블로그의 사운드매니저를 참고하였습니다.

소리조절 설정과 여러 효과들을 사용하였습니다.

출처: https://mentum.tistory.com/269

```
public class AnimationState : MonoBehaviour
{
    static string currentState;
    static Animator animator;

    void Start()
    {
        animator = GetComponent<Animator>();
        currentState = "player_Idle";
    }

    public static void ChangeAnimationState(string newState)
    {
        // Stop the same animation from interrupting itself
        if (currentState == newState)
        {
            //Debug.Log("currentState == newState");
            return;
        }
        if (newState == null)
            return;
        // Play the animation
        animator.Play(newState);

        // Reassign the current state
        currentState = newState;
    }
}
```

*해당 영상을 참고하였습니다.

여러 복잡한 애니메이션 로직들을 코드로 간단하게 상태변경으로 구현하였습니다.

출처 : https://www.youtube.com/watch?v=nBkiSJ5z-hE&t=654s&pp=ygUUdW5pdHkgbWFueSBhbmltYXRpb24%3D

LoadingSceneManager, BattleManager

```
IEnumerator LoadScene()
{
    yield return null;
    AsyncOperation op = SceneManager.LoadSceneAsync(nextScene);
    op.allowSceneActivation = false;
    float timer = 0.0f;
    while (!op.isDone)
    {
        yield return null;
        timer += Time.deltaTime;
        if (op.progress < 0.9f)
        {
            progressBar.fillAmount = Mathf.Lerp(progressBar.fillAmount, op.progress, timer);
            if (progressBar.fillAmount >= op.progress)
            {
                timer = 0f;
            }
        }
        else
        {
            progressBar.fillAmount = Mathf.Lerp(progressBar.fillAmount, 1f, timer);
            if (progressBar.fillAmount == 1.0f)
            {
                op.allowSceneActivation = true;
                yield break;
            }
        }
    }
}
```

*해당 블로그를 참고하였습니다.

씬을 교체할 때 다음 씬을 위한
작업을 위한 로딩 씬을
구현하였습니다.

출처: https://wergia.tistory.com/59 [베르의 프로그래밍 노트:티스토리]

```
public class FSM
{
    public FSM(BaseState initState)
    {
        _curState = initState;
        ChangeState(_curState);
    }

    private BaseState _curState;

    public void ChangeState(BaseState nextState)
    {
        if (nextState == _curState)
            return;

        if (_curState != null)
            _curState.OnStateExit();

        _curState = nextState;
        _curState.OnStateEnter();
    }

    public void UpdateState()
    {
        if (_curState != null)
            _curState.OnStateUpdate();
    }
}
```

*해당 블로그를 참고하였습니다.

씬을 교체할 때 다음 씬을 위한
작업을 위한 로딩 씬을
구현하였습니다.

출처: https://dodobug.tistory.com/16

완성 후 느낀 점

인터넷이 불필요한 게임 장단점

장점: 단순 DB 저장과 서버 구현 필요X,
단점: 매번 수동적인 업데이트

인터넷이 필요한 게임 장단점

장점: 파일과 데이터 서버 저장으로
안정성과 용량 확보
단점: 서버에 문제가 생기면 해결이
힘들다, 인터넷 필수

좋았던 점

Firebase서버의 데이터가 실시간으로
바뀌는 모습이 뿌듯했습니다.

힘들었던 점

서버의 데이터를 제대로 사용하려면 자체 서버를 사용하는
기업 단위로 가야한다고 느꼈습니다.
소규모 게임으로는 불필요한 점이 명확하였습니다.

출처

- 3d 총 에셋 상세 사이트

https://assetstore.unity.com/packages/3d/props/guns/guns-pack-low-poly-guns-collection-192553

- 3d 캐릭터 에셋 상세 사이트

https://assetstore.unity.com/packages/3d/characters/humanoids/humans/adventure-character-201384

- 3d 맵 에셋 상세 사이트

https://assetstore.unity.com/packages/3d/environments/dungeons/decrepit-dungeon-lite-33936

- DoTween 상세 사이트

https://assetstore.unity.com/packages/tools/visual-scripting/dotween-pro-32416

- pixabay 상세 사이트

https://pixabay.com/ko/illustrations/%EC%9D%8C%EC%95%85-%EB%A9%94%EB%AA%A8-%EB%AE%A4%EC%A7%80%EC%BB%AC-%EC%82%AC%EC%9A%B4%EB%93%9C-1700490/

- 해당 pptx 폰트 사용

Tlab 레트로라이프 66

맑은 고딕(본문)

TDTD순고딕

- 검은색 초록색
디지털리즘 기본적이고
단순한 프레젠테이션

https://www.canva.com/design/DAF1CXUUvn8/GlBN1rU8B9kwQKGGsgbWeg/edit?locale=ko-KR&ui=eyJBIjp7IkUiOnsiQSI6dHJ1ZX19fQ

감사합니다!

질문할 것이 있으신가요?