유니티 포트폴리오

목차

- 팀원소개
- 프로젝트 소개
- 씬 구성
- 사용 된 주요 기술소개
- 사용된 Asset

팀원소개

김성호:

- 팀장

- 게임 구성 및 기획
- 프로젝트 구상 및 일정관리
각종 문서 작업
- Stage 제작과 콘텐츠 구성
- SaveLoad시스템 구성 및 제작
- 캐릭터 이동 및 공격과 회피 등 게임 플레이와 관련된 전반적 구상 및 제작
- Shader 제작
- BGM 추가, 쉐이더 제작 및 적용 등등

팀원소개

최호진:

- 팀원

- UI/UX 매니져 구상 및 구현
- 게임 매니져 시스템 구현
- SaveLoad시스템 보완
- Main menu 및 Ending Scene 제작
- Player 및 Enemy 모델링 Asset 적용
- 애니메이션과 사운드 적용
- 기능 테스트 Scene 제작
- LevelUp System UI
- 코드 보완 및 최적화

프로젝트 소개

- 제목 : OneMan Army
- 유형 : 3D 쿼터뷰 , 3인칭 시점의 게임
- 조작 방식 : 키보드, 마우스

게임을 이루는 씬 구성

- MainMenuScene
- SubwayScene
- MazeScene
- Ending Scene

MainMenuScene

- 게임을 실행하면 가장 먼저 보게 될 화면입니다.

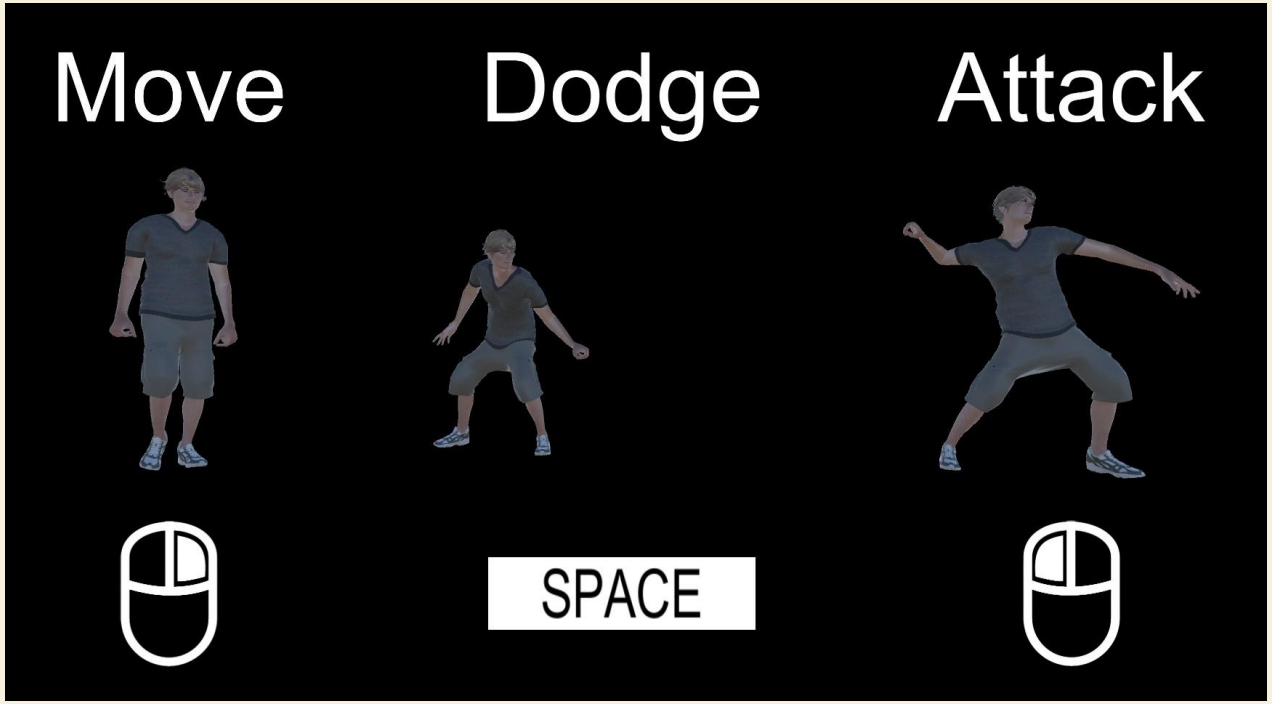
- 게임 시작(Play)
- 이어하기(Continue)
- 종료 (Quit)

- 세 개의 버튼의 간편한 배치로 사용자의 접근성을 고려하여 직관적으로 구성하였습니다.
- 또한 배경으로 앞으로 진행하게 될 플레이어와 적으로 등장할 캐릭터를 대략적으로 알 수 있게 하였습니다.

MainMenu Scene

- 메인메뉴의 새로시작(Play) 버튼을 누르게 되면 화면이 서서히 암전하는 연출이후

- 게임의 조작을 설명하는 화면이 나오게 됩니다.



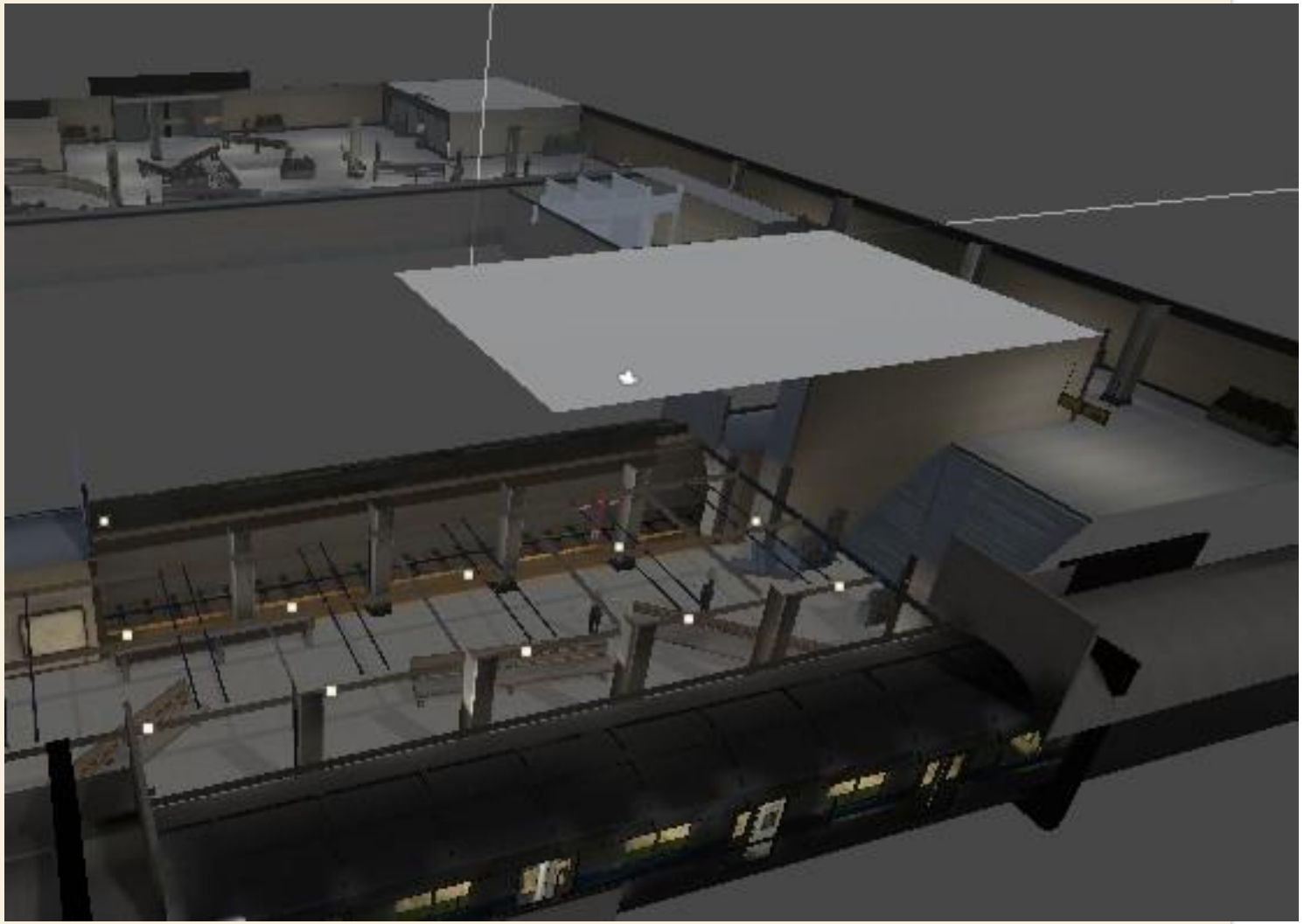
SubwayScene

- Unity AssetStore에서 구매한 Asset입니다.

- 플레이어가 게임을 시작하게 되면 제일 처음으로 경험하게 될 스테이지입니다.

- 지하철을 배경으로 오브젝트들을 여기저기 흩뜨려놓아 일상적인 상황이 아님을 플레이어가 인지할 수 있도록 하였습니다.

- 시작장소와 가까운 위치에 적을 적은 수 배치하여, 직전에 나온 조작방식을 플레이어가 빠르게 적용하여 적응할 수 있도록 유도합니다.



MazeScene

- 두 번째이자 저희가 기획한 마지막 스테이지로, 미로를 컨셉으로 하여 제작하였습니다.

- 이 스테이지에서 플레이어는 오직 가로등의 불빛을 보고 탈출구를 찾아나가야 합니다.

- 안개를 짙게 깔고 으스스한 BGM을 추가하여 공포스러운 분위기를 조성합니다.

- 또한 여기저기에 적을 배치하여 플레이어를 잘못된 방향으로 유도하도록 구성하였습니다.

EndingScene

- 게임의 클리어를 알리는 씬 입니다.
- 메인메뉴로 이동하여 종료하거나 새로 시작 할 수 있습니다.
- 클리어와 동시에 저장 파일을 삭제하여 이어하기가 되지않습니다.

사용된 주요 기술

- SaveLoad System
- UI/UX
- Shader
- Animation Control
- Collider/Trigger

SaveLoad System

```
public static SaveLoad instance;

PlayerStat ps;
Data_set saveData;
string SLdata; // save Load data
string path; // 경로
const string FILENAME = "SaveFile.json";

private void Awake()
{
    if (instance == null)
    {
        instance = this;
    }
    else
    {
        Destroy(instance.gameObject);
        instance = this;
    }

    //다른 씬에서도 사용 가능하도록 유지, 필요없을시 삭제
    //메인화면에서는 Load 기능만, Data_set을 static으로 안 하는 방식사용
    DontDestroyOnLoad(this.gameObject);

}

private void Start()
{
    ps = GameObject.FindObjectOfType<PlayerStat>();
    saveData = new Data_set();
    path = Application.persistentDataPath + "/"; //유니티 기본경로 // C:₩Users₩----₩AppData₩LocalLow₩DefaultCompany₩...
}
```

```
public void SaveData()
{
    ps = GameObject.FindObjectOfType<PlayerStat>();
    saveData.MaxHP = ps.MaxHP.GetStat();
    saveData.Current_HP = ps.Current_HP;
    saveData.Move_speed = ps.Move_speed;
    saveData.Attack_power = ps.Attack_power.GetStat();
    saveData.Stealth = ps.Stealth.GetStat();
    saveData.Armor = ps.Armor.GetStat();
    saveData.EXP = ps.EXP;
    saveData.Level = ps.Level;
    saveData.SceneNumber = SceneManager.GetActiveScene().buildIndex;//현재 씬
    saveData.Pos = ps.transform.GetChild(0).position; //!!!

    SLdata = JsonUtility.ToJson(saveData);
    File.WriteAllText(path + FILENAME, SLdata);// 이부분은 파일 저장
}

public void LoadData()
{
    SLdata = File.ReadAllText(path + FILENAME);
    Data_set Load = JsonUtility.FromJson<Data_set>(SLdata); // 읽어오는 부분

    if (Load.MaxHP == 0)
        return;

    //Player 찾아서 Stat 변경
    ps = GameObject.FindObjectOfType<PlayerStat>();
    ps.MaxHP.SetStat(Load.MaxHP);
    ps.Current_HP = Load.Current_HP;
    ps.Move_speed = Load.Move_speed;
    ps.Attack_power.SetStat(Load.Attack_power);
    ps.Stealth.SetStat(Load.Stealth);
    ps.Armor.SetStat(Load.Armor);
    ps.EXP = Load.EXP;
    ps.Level = Load.Level;
    ps.transform.GetChild(0).position = Load.Pos;//!!!
    ps.transform.GetChild(0).GetComponent<Move>().MoveStop();//이동 멈추기

    UIManager.instance.UpdateHp(ps.Current_HP); // UI 업데이트

    Debug.Log(path);
}
```

핵심이 되는 함수들
저장, 불러오기 와 관련된 함수입니다.

Json 포맷을 이용한 이유는 Unity에서 지원해주는 기능인 데다가

하나의 클래스를 파일로 남기기에 편하기 때문입니다.

SaveData함수에서는 플레이어의 정보를 파일로 변환하여 저장하고

LoadData함수에서는 파일의 Data를 플레이어에게 적용합니다.

```
public void LoadScene() //메인화면에서 이어하기 선택할 시 이 함수 사용 후 LoadData함수 호출
{
    SLdata = File.ReadAllText(path + FILENAME);
    Data_set Load = JsonUtility.FromJson<Data_set>(SLdata);

    SceneManager.LoadScene(Load.SceneNumber);
}

public void InitData() //메인에서 start시 세이브데이터 초기화
{
    saveData = new Data_set();
    SLdata = JsonUtility.ToJson(saveData);
    File.WriteAllText(path + FILENAME, SLdata);
}

public void DeleteData()
{
    System.IO.File.Delete(path + FILENAME);
}
```

UX/UI


DEFEAT
MAIN MENU
RESTART

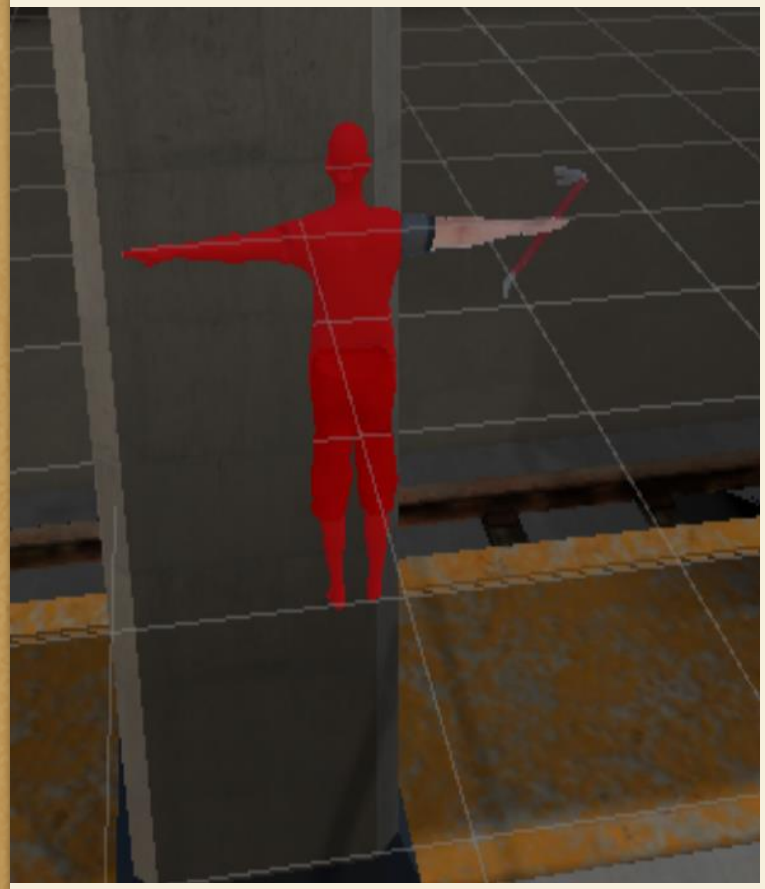

2
Health
Damage
Stealth
Armor
Health : 100(+0)
Damage : 25(+0)
Stealth : 100(+0)
Armor : 0(+2)
UNDO
ACCEPT

UX/UI(Level UP)

```
public void ActiveStatUI(bool active)
{
    if (active == true)
    {
        Time.timeScale = 0;
        ResetStatPoint();
        UpdatePlayerStatText();
        statUI.SetActive(active);
    }
    else
    {
        if (curStatPoint == 0)
        {
            player.GetComponent<PlayerStat>().MaxHP.SetStat(player.GetComponent<PlayerStat>().MaxHP.GetStat() + tmpStatAmount[0, 0]);
            player.GetComponent<PlayerStat>().Current_HP += tmpStatAmount[0, 0];
            UpdateHp(player.GetComponent<PlayerStat>().Current_HP);

            player.GetComponent<PlayerStat>().Attack_power.SetStat(player.GetComponent<PlayerStat>().Attack_power.GetStat() + tmpStatAmount[1, 0]);

            player.GetComponent<PlayerStat>().Stealth.SetStat(player.GetComponent<PlayerStat>().Stealth.GetStat() - tmpStatAmount[2, 0]);
            player.GetComponent<PlayerStat>().TraceTriggerUpdate();

            player.GetComponent<PlayerStat>().Armor.SetStat(player.GetComponent<PlayerStat>().Armor.GetStat() + tmpStatAmount[3, 0]);

            for (int i = 0; i < 4; i++)
            {
                statPointUsed[i] = tmpStatPointUsed[i];
            }

            Time.timeScale = 1.0f;
            UpdateStatIcon(Green);
            ResetStatPoint();

            statUI.SetActive(active);

            SaveLoad.instance.SaveData();  // 스탯 분배후 세이브
        }
        else
        {
            Debug.LogWarning("StatPoint remain");
        }
```

```
public void UpdateStatLevel()
{
    if (curStatPoint != 0 && tmpStatPointUsed[0] != 5 && tmpStatPointUsed[1] != 5 && tmpStatPointUsed[2] != 5 && tmpStatPointUsed[3] != 5)
    {
        curStatPoint -= 1;
        switch (statType)
        {
            case "Health":
                tmpStatAmount[0, 0] += tmpStatAmount[0, 1];
                tmpStatPointUsed[0] += 1;
                Health[tmpStatPointUsed[0]].transform.GetChild(0).GetComponent<Image>().color = yellow;
                break;
            case "Damage":
                tmpStatAmount[1, 0] += tmpStatAmount[1, 1];
                tmpStatPointUsed[1] += 1;
                Damage[tmpStatPointUsed[1]].transform.GetChild(0).GetComponent<Image>().color = yellow;
                break;
            case "Stealth":
                tmpStatAmount[2, 0] += tmpStatAmount[2, 1];
                tmpStatPointUsed[2] += 1;
                Stealth[tmpStatPointUsed[2]].transform.GetChild(0).GetComponent<Image>().color = yellow;
                break;
            case "Armor":
                tmpStatAmount[3, 0] += tmpStatAmount[3, 1];
                tmpStatPointUsed[3] += 1;
                Armor[tmpStatPointUsed[3]].transform.GetChild(0).GetComponent<Image>().color = yellow;
                break;
        }

        UpdatePlayerStatText();
    }
}
```

Shader



간단한 쉐이더를 제작하여
장애물 또는 스테이지를 구성하는 오브젝트에
플레이어블 캐릭터가 가려지면 빨간 색으로 플레이어를
표시하도록 만들었습니다.

이는 3인칭 쿼터뷰 시점의 게임 플레이에 매우 유용한
기능입니다.

shader의 Stencil을 이용한 방법으로
가려질 가능성이 있는 오브젝트에 적용하는 쉐이더와
플레이어를 표시하는 쉐이더 두 개를 제작하였습니다

Shader

```
SubShader{

        Tags { "RenderType" = "Opaque + 2" }
        LOD 200
        ZTest Always
        Stencil
    {
        Ref 1
        Comp Equal
        Pass Keep
    }
```

```
SubShader
{
    Tags { "RenderType"="Opaque" }
    LOD 200
        Pass
    {

        Stencil
    {
    Ref 1
    Comp Always
    Pass Replace
    }
    }
```

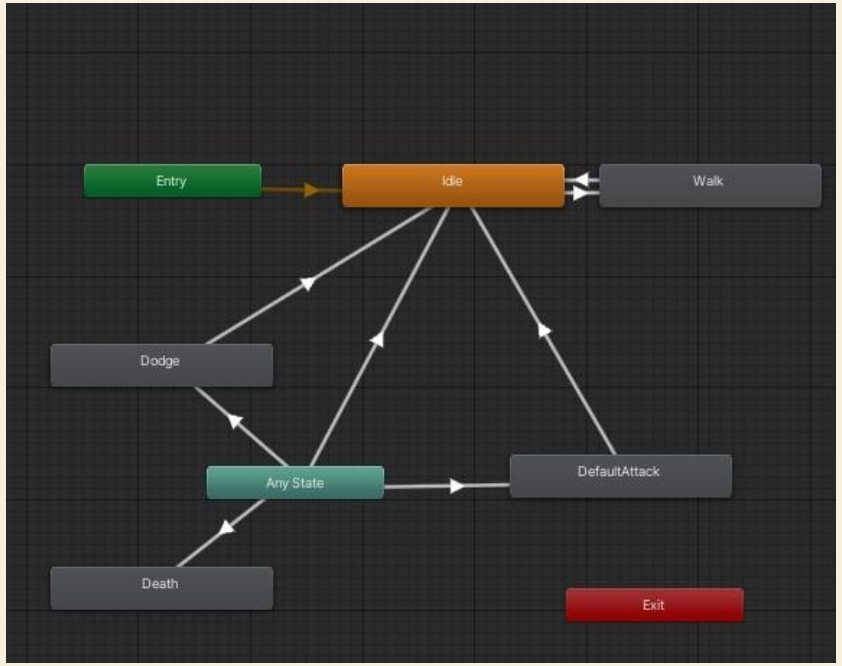
위의 shader는 플레이어를 그리는 부분
아래는 벽에 적용되는 부분

Stencil을 이용하여 같은 값인 1을 가진 픽셀만
그리게 하는 부분입니다.

플레이어를 그리는 Stencil에서 Equal 일때만
화면에 보여지게 하였는데, Z버퍼가 후순위이므로
나중에 그려지게 됩니다.

따라서 앞서 보았던 화면처럼 벽 뒤에 있어도
가려진 부분은 앞에 그려지게 됩니다.

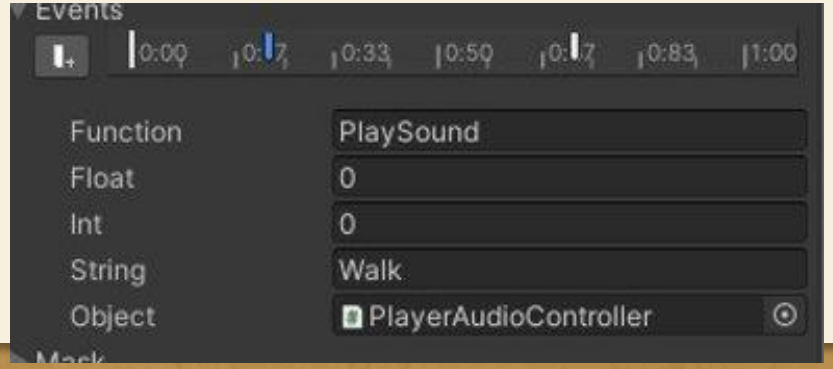
Animation Control



이 그림은 Player캐릭터에 관한 애니메이션 그래프입니다.

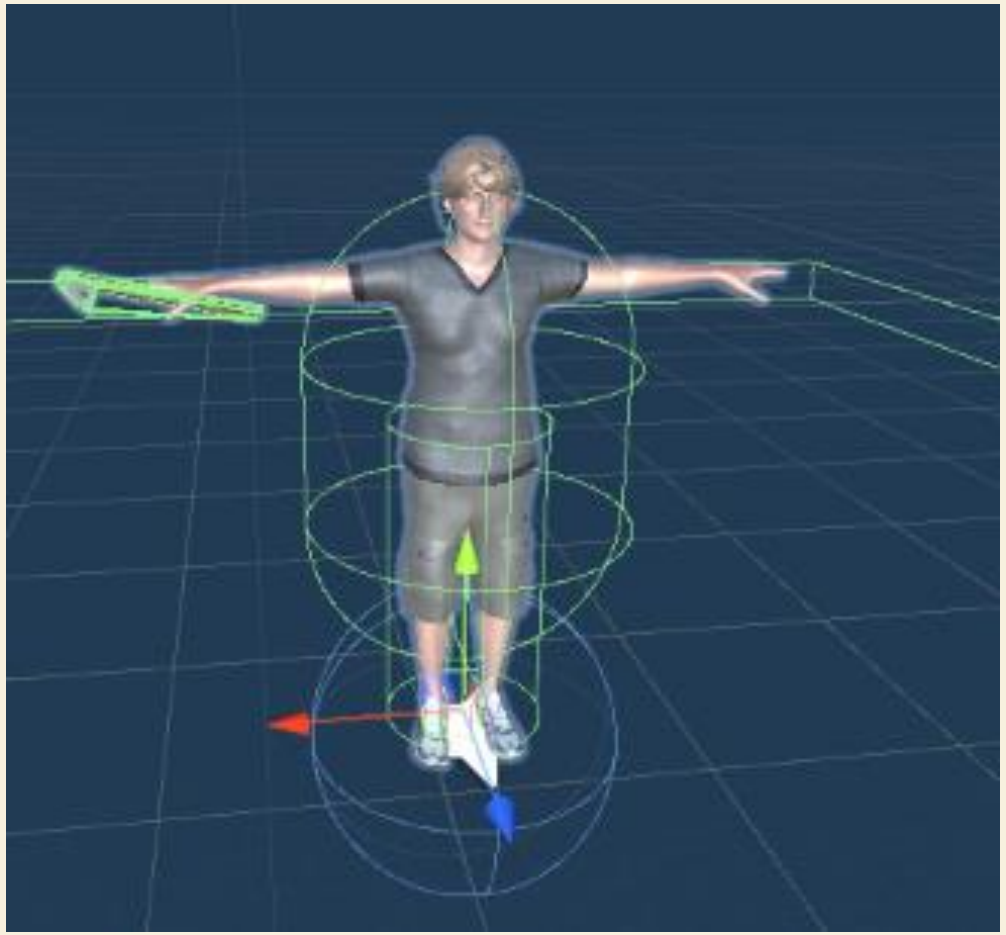
유니티게임엔진에서 제공하는 애니메이션 기능으로, 각 State를 설정하여 애니메이션이 작동하도록 하였습니다.

또한 이를 관리하는 코드를 통해 플레이어의 움직임과 효과음 등을 삽입하였습니다. Non-playable캐릭터인 적 객체 또한 이러한 방식을 적용했습니다.



각 state에 적용된 애니메이션에 이벤트 발생 트리거를 추가하였습니다. 이것으로 PlayerAudioController에 있는 함수를 호출하여 애니메이션 진행 중에 알맞게 소리(효과음)을 재생하였습니다

Collider/Trigger 구성



게임에서 충돌판정은 매우 기초적이고 중요한 요소중 하나입니다. 트리거를 세밀하게 설정 해야만 게임 내에서 다른 물체와 상호작용이 어색하게 보이지 않습니다.

프로젝트 내, 메인 플레이어캐릭터의 트리거들입니다.

제일 바깥쪽은 적이 플레이어를 탐지하는 트리거입니다

 현재는 플레이어가 들고 있는 무기에 씌워진 트리거는 공격 트리거입니다.

플레이어 몸 쪽의 원통은 피격판정 트리거 입니다.

사용된 Asset

•free assests

- https://assetstore.unity.com/packages/2d/gui/icons/free-ui-pack-170878 //UI 관련
- https://fonts.google.com/icons //UI Icon
- https://www.flaticon.com/kr/free-icon/right-click-of-the-mouse_31532# //Main Menu Click Icon
- https://assetstore.unity.com/packages/vfx/particles/3d-games-effects-pack-free-42285 //Next Level Portal
- https://assetstore.unity.com/packages/3d/environments/urban/uk-terraced-houses-pack-free-63481 // Maze Scene
- https://assetstore.unity.com/packages/3d/props/exterior/old-plastic-barrier-182509 // Maze Scene
- https://assetstore.unity.com/packages/audio/sound-fx/retro-sci-fi-pack-61781 // Scene BGM
- https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116 // SFX pack
- https://opengameart.org/ // Sound Effect
- https://www.turbosquid.com/3d-models/3d-model-crowbar-lowpoly-pbr-free-1864881 // Player Weapon
- https://pgtd.tistory.com/200 // Sound
- Mixamo.com 에서 제공하는 무료 모델링과 애니메이션
- unity standard package // camera

•유료

- https://assetstore.unity.com/packages/3d/environments/urban/subway-environment-16741#content //subwayScene
- - $12.99
- https://assetstore.unity.com/packages/3d/characters/zombie-slaughter-119817 // 적(Non - PlayableCharacter) 모델링과 애니메이션
- - $9.99