PFA 매뉴얼

ChasePlayer_Camera.cs

: 카메라가 항상 플레이어를 바라보며 일정 거리 멀어지게 되면 플레이어를 쫓아가도록 설계되었습니다.

```
private GameObject player;
private CinemachineVirtualCamera vcam;

[SerializeField]
private Transform cameraArm;

private float _cinemachineTargetYaw;
private float _cinemachineTargetPitch;

public float BottomClamp = 0.0f;
public float TopClamp = 180.0f;
public float Sensitivity = 0.5f;
```

[변수설명_Part.1]

GameObject player : 플레이어 오브젝트를 가리킵니다.

CinemachineVirtualCamera vcam : 가상카메라로, 플레이어 오브젝트를 다양한 각도에서 비춰줍니다.

Transform cameraArm : 움직이는 카메라의 transform 정보를 갖습니다.

float _cinemachineTargetYaw : 가상카메라의 상하 회전각

float _cinemachineTargetPitch : 가상카메라의 좌우 회전각

float BottomClamp : 아래 방향의 회전 최댓값 제한시킵니다.

float TopClamp : 위 방향의 회전 최댓값 제한시킵니다.

Sensitivity : 회전 속도

```
[Header("CameraSetting")]
public Vector3 Camera_StartPos;
public GameObject CinemachineCamera_Target;
float Camera_MaxDistance = 15.0f;
float Camera_MinDistance = 0.0f;

CinemachineComponentBase componentBase;
```

[변수설명_Part.2]

Vector3 Camera_StartPos : 카메라 시작 위치 설정합니다.
GameObject CinemachineCamera_Target : 가상카메라가 따라갈 오브젝트를 설정합니다.
float Camera_MaxDistance : 카메라와 플레이어의 거리 최대값
float Camera_MinDistance : 카메라와 플레이어의 거리 최소값
CinemachineComponentBase componentBase : 가상카메라의 기본 컴포넌트

```
Unity 메시지 | 참조 0개
void Start()
{
    this.transform.position = Camera_StartPos;
    vcam = GetComponent<CinemachineVirtualCamera>();
    player = null;
    player = GameObject.Find("Player_character(Clone)");
    CinemachineCamera_Target = player.transform.GetChild(0).gameObject;
    _cinemachineTargetYaw = CinemachineCamera_Target.transform.rotation.eulerAngles.y;
    componentBase = vcam.GetCinemachineComponent(CinemachineCore.Stage.Body);
}
```

[코드설명]

Start : 처음 시작했을 때 기본값들을 초기화시켜주는 함수입니다.
카메라의 시작위치와 따라갈 오브젝트를 플레이어로 지정해주고, 가상카메라의 Y값 위치를 지정해줍니다.
vcam 변수도 가상카메라 컴포넌트로 설정해줍니다.

```
Unity 메시지 | 참조 0개
void Update()
{
    vcam.Follow = CinemachineCamera_Target.transform;


    RaycastHit hit;
    Vector3 Cam_Dir = cameraArm.transform.position - CinemachineCamera_Target.transform.position;

    Debug.DrawRay(CinemachineCamera_Target.transform.position, Cam_Dir, Color.green);
    int LayerMaskRayCast = LayerMask.NameToLayer("wall");
    if (Physics.Raycast(CinemachineCamera_Target.transform.position, Cam_Dir, out hit, Camera_MaxDistance, LayerMask.GetMask("Wall")))
    {
        if (componentBase is Cinemachine3rdPersonFollow)
        {
            float final_distance = Mathf.Clamp(hit.distance, Camera_MinDistance, Camera_MaxDistance);
            (componentBase as Cinemachine3rdPersonFollow).CameraDistance = final_distance;
        }
    }
    else
    {
        if (componentBase is Cinemachine3rdPersonFollow)
        {
            if ((componentBase as Cinemachine3rdPersonFollow).CameraDistance < 15.0f)
            {
                (componentBase as Cinemachine3rdPersonFollow).CameraDistance = Camera_MaxDistance;
            }
        }
    }
}
```

[코드설명]

vcam.Follow를 사용하여 시네머신 카메라가 지정한 타겟의 위치로 이동하도록 하였습니다.

RaycastHit을 사용하여 Ray를 직선으로 쏴서 충돌되는 오브젝트에 대한 처리를 해주기 위해 hit 변수를 하나 생성하였습니다.

현재 카메라의 위치값 - Taget오브젝트의 위치값을 하여 플레이어와 카메라의 간격을 체크하기 위한 용도로 Vector좌표로 카메라의 위치를 구하여 저장하였습니다.

Ray를 플레이어의 위치에서 카메라의 위치까지 최대 Camera_MaxDistance 길이로 발사하였을 때 'Wall'이라는 레이어를 가진 오브젝트와 충돌이 될 경우 시네머신 카메라의 3rdPersonFollow 기능을 참조 하고 있다면 플레이어와 카메라의 간격을 Mathf.Clamp 메소드를 사용하여 최소 Camera_MinDistance ~ 최대 Camera_MaxDistnce 수치중 충돌된 오브젝트와의 간격을 구하여 final_distance 변수에 간격수치를 저장합니다. 시네머신의 Distance 기능을 활용해 final_distance 수치만큼 떨어진 위치로 카메라가 이동되도록 하여 벽과의 충돌이 있을 경우 카메라가 벽을 통과하는 않도록 하였습니다.

충돌되었을 때 오브젝트레이어가 'Wall'이 아니라면 카메라의 간격은 Camera_MaxDistance로 고정되도록 하였습니다.

```
참조 1개
private void CameraRotation()
{
    //카메라 회전속도 계산
    _cinemachineTargetYaw += Input.GetAxis("Mouse X") * Sensitivity;
    _cinemachineTargetYaw = ClampAngle(_cinemachineTargetYaw, float.MinValue, float.MaxValue);

    _cinemachineTargetPitch -= Input.GetAxis("Mouse Y") * Sensitivity;
    _cinemachineTargetPitch = ClampAngle(_cinemachineTargetPitch, BottomClamp, TopClamp);

    //카메라 회전변화
    CinemachineCamera_Target.transform.rotation = Quaternion.Euler(_cinemachineTargetPitch, _cinemachineTargetYaw, 0.0f);


}
```

[코드설명]

CameraRotation : 카메라를 마우스를 통해 회전시키는 함수입니다.

Mouse X, Mouse Y를 통해 마우스로 카메라 회전각을 지정해줍니다. Sensitivity 변수를 통해 회전 속도를 조정합니다.

ClampAngle 함수를 호출시켜 카메라 회전각을 제한시켜줍니다.

마지막으로 변화되는 회전각을 CinemachineCamera_Target을 기준으로 회전시켜줍니다.

```
private static float ClampAngle(float lfAngle, float min, float max)
{
    if (lfAngle < -360.0f) lfAngle += 360.0f;
    if (lfAngle > 360.0f) lfAngle -= 360.0f;

    //사이값으로 고정시켜주는 수학함수.
    return Mathf.Clamp(lfAngle, min, max);
}
```

[코드설명]

ClampAngle : 카메라의 회전각의 최대, 최소를 제한시키는 함수입니다.

마우스를 통해 회전된 각도가 360도보다 초과하거나 -360도보다 작으면 알맞게 각도를 계산하여 회전각을 제한시켜줍니다.

마지막 Mathf.Clamp를 통해 ClampAngle 함수를 호출할 때 지정해놓은 최대,최소값의 사이값으로 카메라 회전각을 고정해줍니다.

```
private void LateUpdate()
{
    CameraRotation();

}
```

[코드설명]

LateUpdate 함수 : Update 함수가 호출된 후, 마지막으로 호출되는 함수로, 주로 오브젝트를 따라가게 설정한 카메라에 사용됩니다.

즉, 플레이어를 따라가는 가상카메라의 회전을 관리하는 CameraRotation함수를 호출시킵니다.

Player.cs

```
{
    public int jumpCount = 0;
    private float speed = 0.0f;
    public float WalkSpeed = 5.0f;
    public float RunSpeed = 15.0f;
    public float jumpPower = 15.0f;

    bool jumping;
    Rigidbody rb;
    public GameManager GM;
    Animator anim;
    public Vector3 Save_Pos;
    bool isBorder;
    AudioSource audioSource;
    [SerializeField] AudioClip[] EffectSound;

    [Tooltip ("Camera Roteate Object")]
    private GameObject Follow_Cam;
    private float Target_Rotate;
    private float rotationVelocity;
    private float Rotation_SmoothTime = 0.12f;

    public float gravity = -20;
    float Jump_Timer = 0;
```

[변수 설명]
jumpCount : 점프 횟수를 세기 위한 변수
speed : 플레이어의 이동속도
WalkSpeed : 플레이어 걷기 속도
RunSpeed : 플레이어 달리기 속도
jumpPower : 플레이어의 점프 힘
jumping : 플레이어가 점프중인지 확인하는 변수
rb : 오브젝트를 물리 제어로 동작하도록 하기 위한 변수
GM : 게임 매니저 스크립트를 참조하기 위한 변수
anim : 애니메이션을 동작시키기 위한 변수
isBorder : 플레이어 앞에 벽이 있는지 확인하는 bool 변수
Save_Pos : 현재 세이브 Vector 위치값을 저장하기 위한 변수
audioSource : 오디오를 출력시킬 오브젝트를 지정하는 변수
EffectSound : 외부 스크립트의 접근을 차단하고 유니티 인스펙터 창에서 오디오 파일을 적용하는데 불편함이 없도록 SerializeField를 사용하여 오디오 클립 배열 생성
Follow_Cam : 플레이어를 따라오는 카메라 오브젝트를 정의하기 위한 변수
Target_Rotate : 캐릭터의 회전값을 저장하기 위한 float형 변수
rotationVelocity : 회전 속도를 정하기 위한 float 변수
Rotation_SmoothTime : 플레이어가 회전 시 매끄럽게 회전되도록 회전에 걸리는 시간을 저장한 float 변수
gravity : 점프 시 중력수치를 주어 바닥으로 자연스럽게 떨어지도록 하기 위한 float 변수
Jump_Timer : 점프 후 특정 시간 이후에 더블점프가 가능하도록 시간을 확인하기 위한 float 함수

플레이어 이동

```
void Move()
{
    Vector2 moveInput = new Vector2(Input.GetAxisRaw("Horizontal"), Input.GetAxisRaw("Vertical"));

    bool isMove = moveInput.magnitude != 0;

    if (isMove)
    {
        if (!(moveInput.x == 0 && moveInput.y == 0))
        {
            Target_Rotate = Mathf.Atan2(moveInput.x, moveInput.y) * Mathf.Rad2Deg + Follow_Cam.transform.eulerAngles.y;
            float rotaton = Mathf.SmoothDampAngle(transform.eulerAngles.y, Target_Rotate, ref rotationVelocity, Rotation_SmoothTime);
            transform.rotation = Quaternion.Euler(0.0f, rotaton, 0.0f);

            Vector3 targetDirection = Quaternion.Euler(0.0f, Target_Rotate, 0.0f) * Vector3.forward;
            transform.position += targetDirection.normalized * speed * Time.deltaTime;
        }
    }

    anim.SetBool("is_Walking", isMove);
}
```

[코드 설명]
moveInput : 키보드 입력 값을 눌렀을 때 즉시 반응하도록 GetAxisRaw 함수를 사용하였으며 유니티 프로젝트의 InputManager에서 설정한 입력키를 누를 경우 입력값이 넘어오도록 설계되었습니다.
moveInput.magnitude 메소드를 통해 키가 입력이 되었는지 확인하며 isMove 불값에 true,

false로 값이 저장되도록 하였습니다.

키가 눌린 상태에서 moveInput의 x와 y의 값이 0이 아니라 이동 중인 경우에 해당하는 코드가 실행되도록 하였습니다. 움직일 대상의 회전 값은 Mathf 라이브러리의 Atan2 메소드를 사용하여 누르고 있는 키의 방향으로 라디안 값을 체크하며 플레이어를 추적하는 카메라의 각도에 이 라디안 값을 더해 누르는 방향으로 회전 값이 적용되도록 하였습니다.
회전을 할 때 자연스럽게 미끄러지듯이 회전하도록 하기 위해 Mathf.SmoothDampAngle 메소드를 사용하여 Rotation_SmoothTime 내에 현재 오브젝트의 회전 값을 rotation 변수에 저장하였으며 이를 Quternion.Euler 메소드를 사용하여 해당 각도로 회전되도록 하였습니다.

현재 플레이어는 회전만 할 뿐 이동은 하지 않고 있기 때문에 targetDirection Vector3 값을 활용하여 회전을 한 이후에 Vector3.forward를 사용하여 바라보는 방향으로 Vector3의 값이 변하도록 하여 targetDirection 변수에 저장하였습니다. 이동하고 있을 때 매번 위치값이 바뀌어야 하기 때문에 이전에 구한 targetDirection의 백터 값을 정규화 하여 지정한 스피드와 매초 프레임을 곱하여 위치값이 변경되도록 하였습니다.

 # 플레이어 점프

```
private void Update()
{
    Move();

    if(jumpCount < 2)
    {
        jump();
    }
    else jumping = false;

    run();

    Die();

    StopToWall();
}
```

```
public void jump()
{
    if (Input.GetKeyDown(KeyCode.Space))
    {
        jumpCount++;


        if (jumping)
        {
            rb.AddForce(Vector3.up * jumpPower, ForceMode.Impulse);
            anim.SetBool("is_Jumping", true);
        }
        else
        {
            anim.SetBool("is_Jumping", false);
        }

    }
}
```

[코드 설명]
매 프레임마다 JumpCount를 체크하게 되며 현재 점프 횟수가 2보다 미만일 때 점프 함수를 실행하도록 하였으며 JumpCount가 2를 초과하게 되면 점프를 할 수 없도록 jumping bool 값을 false로 만들었습니다.

Jump 함수에서 스페이스 키를 누르게 되면 JumpCount가 1씩 더해지게 되며 Rigidbody의 AddForce(벡터 값, ForceMode) 함수를 사용하여 위쪽 방면으로 Vector 값이 증가하도록 하였습니다. ForceMode를 Impulse로 설정하여 짧은 순간에 뛰어오르도록 설계하였습니다.
애니메이션의 파라미터를 설정하여 플레이어가 점프 중일 때만 is_Jumping bool 값이 true가 되도록 하여 점프 애니메이션이 동작하도록 하였습니다.

OnCollisionEnter 이벤트를 사용하여 오브젝트와 충돌하게 되었을 때 해당 오브젝트의 태그가 ground라면 Jump_Reset() 함수가 실행되도록 하였습니다.

Jump_Reset() 함수를 통해 설정된 애니메이션의 파라미터의 is_Jumping bool 값이 false가 되도록 하여 점프 애니메이션이 종료되도록 하였으며 jumping bool 값을 true로 변경하여 점프가 가능한 상태로 만들었습니다. 또한 jumpCount를 0으로 초기화 하여 현재 점프한 횟수가 0회가 되도록 하여 다시 더블 점프가 가능하도록 설계하였습니다.

 # 플레이어 달리기

```
private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("ground"))
    {
        Jump_Reset();
    }

    Check_Savepoint(collision);
}

void Jump_Reset()
{
    anim.SetBool("is_Jumping", false);
    jumping = true;
    jumpCount = 0;
}
```

```
public void run()
{

    if (Input.GetKey(KeyCode.LeftShift))
    {
        anim.SetBool("is_Running", true);
        speed = 9;
    }
    else
    {
        anim.SetBool("is_Running", false);
        speed = 4;
    }
}
```

[코드 설명]

플레이어가 LeftShift 키를 누르게 되면 이동속도가 변경되게 되며 애니메이션의 파라미터를 설정하여 플레이어가 달리기 중일 때만 is_Running bool 값이 true가 되도록 하여 달리기 애니메이션이 동작하도록 하였습니다.

 # 플레이어 죽음

```
void Die()
{
    if (gameObject.transform.position.y <= dieCoordinate)
    {
        audioSource.clip = EffectSound[0];
        audioSource.Play();
        Debug.Log("플레이어 죽음");
        this.gameObject.transform.position = Save_Pos;
    }
}
```

[코드 설명]
플레이어 오브젝트의 y 값이 지정한 수치 이하로 떨어지게 되면 EffectSound 배열의 0번째 오디오 클립이 재생되며 플레이어가 죽기 이전에 세이브된 위치로 돌아가게 됩니다.

 # 세이브 포인트

```
private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("ground"))
    {
        Jump_Reset();
    }

    Check_Savepoint(collision);
}
```

```
void Check_Savepoint(Collision collision)
{
    if (GM.is_Children_Room)
    {
        Save_LightON(collision);
    }
    else if (GM.is_School)
    {
        Save_LightON(collision);
    }
    else if (GM.is_PlayGround)
    {
        Save_LightON(collision);
    }
}
```

[코드 설명]
OnCollisionEnter 이벤트를 사용하여 오브젝트와 충돌하게 되면 Check_Savepoint 함수에 충돌된 오브젝트의 정보가 넘어가게 됩니다.
각 맵의 세이브 포인트 지점은 다르기 때문에 Check_Savepoint 함수에서 게임매니저 스크립트를 참조하여 현재 플레이어가 존재하는 맵의 정보를 받아오도록 하였습니다.
현재 맵의 정보를 확인한 후에 충돌한 오브젝트 정보를 Save_LightON 함수에 한번 더 전달하게 됩니다.

```
void Save_LightON(Collision collision)
{
    if (collision.gameObject.CompareTag("Save_point_1"))
    {
        Debug.Log("test");
        GameObject obj1 = GameObject.Find("Save_1");
        obj1.gameObject.transform.GetChild(0).gameObject.SetActive(true);
        Save_Pos = obj1.transform.position;
        Jump_Reset();
    }
    else if (collision.gameObject.CompareTag("Save_point_2"))
    {
        GameObject obj1 = GameObject.Find("Save_2");
        obj1.gameObject.transform.GetChild(0).gameObject.SetActive(true);
        Save_Pos = obj1.transform.position;
        Jump_Reset();
    }
    else if (collision.gameObject.CompareTag("Save_point_3"))
    {
        GameObject obj1 = GameObject.Find("Save_3");
        obj1.gameObject.transform.GetChild(0).gameObject.SetActive(true);
        Save_Pos = obj1.transform.position;
        Jump_Reset();
    }
    else if (collision.gameObject.CompareTag("Save_point_4"))
    {
        GameObject obj1 = GameObject.Find("Save_4");
        obj1.gameObject.transform.GetChild(0).gameObject.SetActive(true);
        Save_Pos = obj1.transform.position;
        Jump_Reset();
    }
    else if (collision.gameObject.CompareTag("Save_point_5"))
    {
        GameObject obj1 = GameObject.Find("Save_5");
        obj1.gameObject.transform.GetChild(0).gameObject.SetActive(true);
        Save_Pos = obj1.transform.position;
        Jump_Reset();
    }
}
```

충돌된 오브젝트의 태그를 체크하게 되며 해당 오브젝트의 하위 객체의 PointLight의 SetActive를 true로 변경하여 플레이어가 세이브포인트에 충돌하게 되면 해당 오브젝트 주변이 밝게 빛나도록 하였습니다.

Save_Pos 변수에 충돌된 오브젝트의 위치값을 저장하게 되며 플레이어가 죽게 되었을 때 해당 위치로 부활할 수 있도록 구성하였습니다. 또한 오브젝트와 충돌되었을 때 점프가 되지 않는 현상을 방지하기 위해 Jump_Reset() 함수를 실행하도록 하였습니다.

 # 벽 충돌

```
void StopToWall()
{
    Debug.DrawRay(transform.position, transform.forward * 5, Color.green);
    isBorder = Physics.Raycast(transform.position,
        transform.forward, 5, LayerMask.GetMask("Wall"));
}
```

[코드 설명]
DrawRay(위치값, 방향, 색상)함수를 사용하여 플레이어 캐릭터 앞쪽에 녹색선을 그려 테스트하기 용이하도록 하였습니다.
Physics.Raycast(위치, 방향, 최대사거리, 레이어 마스크) 함수를 사용하여 플레이어의 전방에 Ray를 발사하여 닿는 충돌체의 레이어가 Wall인지 확인하여 isBorder의 bool값을 정하게 설계되었습니다.
isBorder이 true가 되면 플레이어가 이동시 위치값이 변경되지 않도록 하였습니다.
※ 플레이어 이동 부분 참조

 # 포탈 구현

```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("ChildRoom"))
    {
        SceneManager.LoadScene("School");
    }
    else if (other.CompareTag("School"))
    {
        SceneManager.LoadScene("PlayGround");
    }
    else if (other.CompareTag("PlayGround"))
    {
        SceneManager.LoadScene("End");
    }
}
```

[코드 설명]
두 게임 오브젝트가 충돌하였을 때 물리적 연산을 하여 서로 튕겨나가는 현상을 방지하고자 OnTriggerEnter 이벤트를 사용하였으며 충돌된 오브젝트의 태그를 확인하고 다음 Scene으로 전환 되도록 구성하였습니다.

GameManager.cs

: 게임의 전반적인 상태, 규칙을 관리하기 위해 설계되었습니다.

```
public static GameManager instance;
public GameObject player;
public bool isSoundOn, isMusicOn;
public Vector3 PlayerGround_Pos; //포탈 이동시 플레이어 위치값 변경
public Vector3 School_Pos;
public Vector3 Children_Room;
public bool is_School;
public bool is_PlayGround;
public bool is_Children_Room;
public AudioSource bgmPlayer;
[SerializeField] AudioClip[] BackGround_Clip;
```

[변수 설명]
instance : 싱글톤 패턴을 적용하기 위한 목적으로 사용
player : 플레이어 오브젝트
isSoundOn, isMusicOn : 옵션 창에서 배경음과 효과음 Mute 설정을 위한 변수
PlayerGround_Pos : Ground 맵에서 오브젝트 생성 위치
School_Pos : School 맵에서 오브젝트 생성 위치
Children_Room : Children_Room 맵에서 오브젝트 생성 위치
is_School : 현재 적용 중인 맵 체크를 위한 bool 값
is_PlayGround : 현재 적용 중인 맵 체크를 위한 bool 값
is_Children_Room : 현재 적용 중인 맵 체크를 위한 bool 값
bgmPlayer : 오디오를 출력시킬 오브젝트를 지정하는 변수
BackGround_Clip : 외부 스크립트의 접근을 차단하고 유니티 인스펙터 창에서 오디오 파일을 적용하는데 불편함이 없도록 SerializeField를 사용하여 오디오 클립 배열 생성

싱글톤 패턴 적용

```
private void Awake()  //싱글톤 패턴 사용
{
    if (instance == null)
    {
        instance = this;
    }
    else if (instance != this)
    {
        Destroy(gameObject);
    }

    DontDestroyOnLoad(gameObject);
}
```

[코드 설명]
Awake()함수를 통해 게임이 시작되기전 자기자신을 instance 객체로 선언하며 씬이 전환되더라도 이 오브젝트가 파괴되지 않고 유지되도록 하기 위해 DontDestroyOnLoad(오브젝트) 함수를 사용하였습니다.

Scene 전환 이벤트

```
void OnSceneLoaded(Scene scene, LoadSceneMode mode) // 씬전환시 이벤트 호출
{
    if (scene.name == "ChildRoom")
    {
        bgmPlayer.clip = BackGround_Clip[0];
        bgmPlayer.Play();
        is_Children_Room = true;
        is_School = false;
        is_PlayGround = true;
        Instantiate(player, Children_Room, transform.rotation);
    }
    else if (scene.name == "SchoolSang")
    {
        bgmPlayer.clip = BackGround_Clip[1];
        bgmPlayer.Play();
        is_Children_Room = false;
        is_School = true;
        is_PlayGround = false;
        Instantiate(player, School_Pos, transform.rotation);
    }
    else if (scene.name == "PlayGround")
    {
        bgmPlayer.clip = BackGround_Clip[2];
        bgmPlayer.Play();
        is_Children_Room = false;
        is_School = false;
        is_PlayGround = true;
        Instantiate(player, PlayerGround_Pos, transform.rotation);
    }
}
```

Scene을 불러오게 되는 경우 OnSceneLoaded(Scene, LoadSceneMode) 함수가 실행되며 불러온 Scene의 이름을 체크하여 해당 Scene에 맞는 배경음을 재생 시키도록

BackGround_Clip 배열에 [n]번째 클립을 재생하도록 하였습니다. 현재 적용중인 맵을 체크하여 다른 스크립트에서 참조할 수 있도록 하기 위해 is_Children_Room, is_School, is_PlayGround bool 값을 true / false로 지정하도록 설계하였습니다.
Instantiate(오브젝트, 위치값, 회전 값) 함수를 사용하여 Scene이 전환될 때마다 플레이어 오브젝트가 지정한 위치에 회전 값을 가지고 생성되도록 설계하였습니다.

 # 옵션 버튼 구현

```
public void soundToggle()
{
    isSoundOn = !isSoundOn;
}

public void musicToggle()
{
    isMusicOn= !isMusicOn;
}

public void gameExit()
{
    Application.Quit();
}
```

옵션 창에서 효과음 체크 박스를 해제하게 되면 사운드가 들리지 않도록 하기 위해서 soundToggle() 함수를 사용하여 isSoundOn bool 값이 false가 되도록 구성하였습니다.

옵션 창에서 배경음 체크 박스를 해제하게 되면 사운드가 들리지 않도록 하기 위해서 musicToggle()함수를 사용하여 isSoundOn bool 값이 false가 되도록 구성하였습니다.

옵션 창에서 게임 종료 버튼을 클릭 시 게임이 종료 되도록 하기 위해 Application.Quit() 함수를 사용하여 게임 프로세스가 종료되도록 하였습니다.

DoorController.cs

: 플레이어와 상호작용할 수 있는 문을 제작하기 위해 만들어진 코드입니다.
: 키보드 'E'를 누르면 문이 열리며, 타이머 설정을 통해 제한시간이 지나면 자동으로 문이 닫히게끔 설계되어 있습니다.

```
float time = 0;
bool isTrigger = false;
bool isOpen = false;

Animator ani;
```

[변수설명]
float time : 몇 초 뒤에 자동으로 문을 닫게 할지를 설정함.
bool isTrigger : 플레이어가 문 앞에 도착했는지를 판단함.
(true : 플레이어가 문 앞에 있을 때 / false : 플레이어가 문 앞에 없을 때)
bool isOpen : 문이 열렸는지를 판단함.
(true : 문이 열려있는 상태 / false : 문이 닫혀있는 상태)
Animator ani : 문 열리는 애니메이션 실행시키기 위함.

```
Unity 메시지 | 참조 0개
private void Start()
{
    ani = GetComponent<Animator>();
}
```

[코드설명]
Start 함수 : Animator 변수로 선언한 ani를 해당 오브젝트에 포함된 Component 중 Animator에 참조하겠다는 뜻.

```
Unity 메시지 | 참조 0개
private void Update()
{
    if (isTrigger)
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            ani.SetTrigger("doorOpen");
            isOpen=true;
            isTrigger = false;
        }
    }

    if(isOpen)
    {
        time+=Time.deltaTime;
        if (time>3.0f)
        {
            ani.SetTrigger("doorClose");
            time =0;
            isOpen = false;
        }
    }

}
```

[코드설명]

Update 함수 : 플레이어가 문 앞에 있다면 첫 번째 if문이 실행된다.
이후 'E'키를 누르면 Animator를 통해 문이 열리는 애니메이션이 실행되고, 문이 열린 상황에 대한 변수들로 초기화된다.

문이 열리는 애니메이션이 실행되면, 두 번째 if문이 실행된다.
변수 time을 통해 시간을 측정하며, 설정한 제한시간 3초(변경가능)가 흐르면, 애니메이션을 통해 문이 닫히고, 문이 닫힌 상황에 대한 변수들로 초기화된다.

```
Unity 메시지 | 참조 0개
private void OnTriggerEnter(Collider other)
{
    if(other.gameObject.name =="Player_character(Clone)")
    {
        isTrigger= true;
    }
}
```

[코드설명]

OnTriggerEnter : 문 오브젝트 앞에 제작한 Box Collider에 "Player_character(Clone)" 오브젝트(플레이어)가 부딪혔다면, 문 열리는 조건에 대한 변수 설정.

RotationTree.cs

: 플랫포머 게임의 장애물 요소 중 쉬지않고 회전하는 원통 오브젝트를 제작하기 위한 코드입니다.

: 원하는 회전속도를 조정할 수 있게 변수로 선언하였습니다.

```
public float rotateSpeed;
```

[변수설명]

float rotateSpeed : 회전속도를 정하기 위해 선언한 변수

(반대 방향으로 돌리고 싶으면 값을 음수로 설정하면 됩니다.)

```
Unity 메시지 | 참조 0개
void Update()
{
    this.transform.Rotate(new Vector3(0f, rotateSpeed, 0f) * Time.deltaTime, Space.Self);
}
```

[코드설명]

Update함수 : 설정한 회전속도만큼 오브젝트를 회전시킨다.

- Rotate를 활용하여 연속움직임을 구현한다.
- Space.Self를 통해 자기 자신을 축으로 설정되어 있어 각도가 틀어져도 틀어진 상태에서 회전한다.

LockerController.cs

: 플랫포머 게임의 장애물 요소 중 플레이어와 사물함 오브젝트의 상호작용을 위해 제작하였습니다.

: 키보드 'E'를 누르면 문이 열리며, 자동으로 사물함이 열리고, 다시 'E'를 누르면 닫히게끔 설계되어 있습니다.

```
Animator animator;
```

[변수설명]

Animator animator : 사물함 여닫는 동작을 위해 애니메이션으로 조정함.

```
Unity 메시지 | 참조 0개
void Start()
{
    animator = GetComponent<Animator>();
}
```

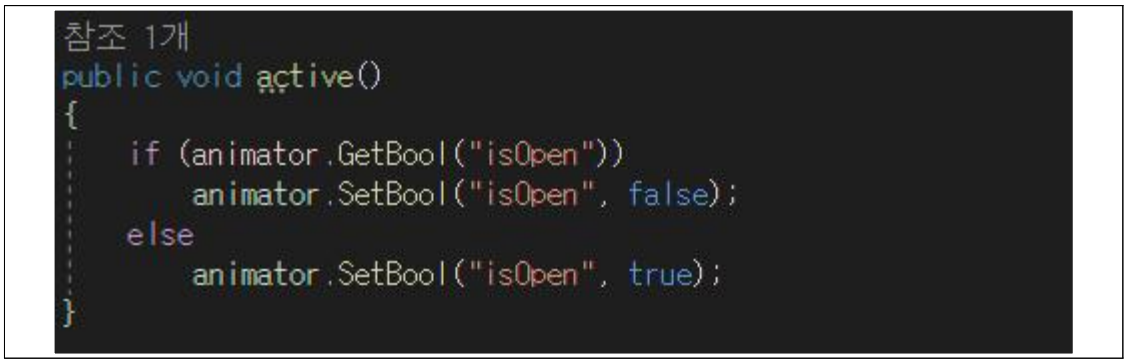
[코드설명]

Start 함수 : Animator 변수로 선언한 animator를 해당 오브젝트에 포함된 Component 중 Animator에 참조하겠다는 뜻.

```
Unity 메시지 | 참조 0개
void Update()
{
    if (Input.GetKeyUp(KeyCode.E))
        active();
}
```

[코드설명]

Update 함수 : 'E'키를 눌렀다 뗐을 때 active() 함수를 실행시킨다.



```
참조 1개
public void active()
{
    if (animator.GetBool("isOpen"))
        animator.SetBool("isOpen", false);
    else
        animator.SetBool("isOpen", true);
}
```

[코드설명]

active 함수 : 애니메이션의 bool값을 이용하여 사물함 문이 열려있다면 닫는 애니메이션을 실행시킨다.

반대로 사물함 문이 닫혀있다면, 여는 애니메이션을 실행시킨다.
