Rising Of Citadel 프로젝트 내부 코드 설명

주의 사항 : 해당 문서는 제가 담당한 모든 내용을 설명하기엔 규모가 크므로 핵심적인 일부만 가져와 설명을 진행하였습니다.

[플레이어 관련 클래스]

1. **Player :** 플레이어 체력, 마나 등의 플레이어가 가진 수치 값들을 다루고, 싱글톤 패턴을 이용해 게임 내에서 항상 하나만 존재하며 언제 어디서든지 접근 가능하도록 함.

[코드 설명]

```
private void Awake()
{
    startPosition = transform.position;
    if (PlayerSaveData.goldLock) // goldLock : 게임을 시작 할 때 gold 값을 초기값으로 되돌리는지 여부 / 테스트나 특별한 경우 제외하곤 true이다.
    {
        // PlayerSaveData = 플레이어 데이터 JS 저장할 때 사용하는 클래스 + 일시적으로 게임 실행 중에만 데이터를 저장하기도 함.
        // 아래는 플레이어가 게임 시작 시 데이터를 초기화하는 코드
        PlayerSaveData.goldLock = false;
        PlayerSaveData.turretAmount = 0; // 터렛 개수 초기화
        PlayerSaveData.gold = 200; // 골드 초기화
        PlayerSaveData.itemList = new List<string>(); // 보유 아이템 초기화
    }

    try // 데이터 존재 시 데이터 로드
    {
        playerSaveData = SaveManager.Load<PlayerSaveData>("PlayerData");
    }
    catch // 그 외는 새로 데이터 생성해 초기 값을 사용한다 보면 됩니다.
    {
        playerSaveData = new PlayerSaveData();
    }

    // hp, mp 초기화
    hp = playerSaveData.maxHP;
    mp = playerSaveData.maxMP;

    // Player에 어디서든 접근 가능하도록 Player 객체를 Static 변수로 저장.
    if (!instance)
        instance = this;

    // 플레이어 관련 기능을 가지는 클래스 객체들을 변수에 저장해 둠.
    animationController = GetComponent<PlayerAnimationController>();
    playerAttackSkill = GetComponent<PlayerAttackSkill>();
    movement = GetComponent<PlayerMovement>();

    if (!ui) // 플레이어 전용 UI 생성
    {
        GameObject instance = Instantiate(uiPrefab);
        ui = instance.GetComponent<UIController>();

        PauseGame.instance = ui.setting.GetComponent<PauseGame>();
    }
    if (instance && !instance.playerCamera) // 플레이어 전용 카메라 생성
        playerCamera = Instantiate(cameraPrefab, transform.position, Quaternion.identity);
}
```

- Awake 함수 내부에는 게임 시작 시 데이터를 전부 초기화한다.
- 또한, 캐릭터에 사용되는 다른 클래스 객체(ex: PlayerMovement 등)를 변수로 저장하고, UI나 카메라(CameraManager) 등 게임에 필요한 요소들을 자동으로 생성하게 한다.

```
private void Update()
{
    if (!isRecoverMp && !Player.instance.playerAttackSkill.isAttack)
    {
        isRecoverMp = true;
        StartCoroutine(RevertMp());
    } // Mp Recover

    HpRefresh(); // Always hp value is changing.
    MpRefresh(); // Always mp value is changing.

    if (mp < 0) mp = 0;

}
```

- Update 함수 내부는 mp 가 시간이 지나면 자동으로 채워지도록 조절하는 코루틴 RevertMP 을 실행하고 있으며, HPRefresh, MPRefresh 라는 두 함수를 이용해

  UI 에서 자연스럽게 hp 와 mp 의 표시를 값에 따라 천천히 변화시키고 있다.

- 또한, mp 가 음수 값이 되지 않도록 체크 후 수정한다.

```
public void Die()
{
    Camera.main.gameObject.SetActive(false);
    ui.gameObject.SetActive(false);
    GameObject gameover = Instantiate(gameoverPrefab);
}

참조 0개
public void Rebirth()
{
    hp = PlayerAttackSkill.passiveSkillData.rebirthHp;
    mp = PlayerAttackSkill.passiveSkillData.rebirthMp;

    StopPlaying(false);
    spawn = playerCamera.GetComponent<StructureSpawn_Test>();
    itemInteraction = playerCamera.GetComponent<ItemInteractionManager>();
    spawn.enabled = true;
    itemInteraction.enabled = true;

}
```

- 이외에도 캐릭터의 죽음을 처리하는 Die 나 부활을 처리하는 Rebirth 등의 다양한 함수가 존재해 플레이어 캐릭터를 제어한다.

2. **PlayerMovement :** 플레이어 이동 조작 및 움직임을 담당하는 클래스

[코드 설명]

```
// Default move value.

float horizontalMove = Input.GetAxisRaw("Horizontal");
float verticalMove = Input.GetAxisRaw("Vertical");

// Default move value while player jumping.

if (airSpeedX is > 0 or < 0)
{
    airSpeedX -= airSpeedX * Time.unscaledDeltaTime;
}
else { airSpeedX = 0; }

if (airSpeedZ > 0)
{
    airSpeedZ -= airSpeedZ * Time.unscaledDeltaTime;
}
else if (airSpeedZ < 0)
{
    airSpeedZ += airSpeedZ * Time.unscaledDeltaTime;
}
else { airSpeedZ = 0; }

////////////////////////////////////////////////////////////////////// Stop speed part

if ((airSpeedX) > 0)
{
    airSpeedX += horizontalMove / 10 * smoothStopIntensity * Time.unscaledDeltaTime;
    if (airSpeedX < 0) airSpeedX = 0;
}
else if ((airSpeedX) < 0)
{
    airSpeedX += horizontalMove / 10 * smoothStopIntensity * Time.unscaledDeltaTime;
    if (airSpeedX > 0) airSpeedX = 0;
}

if ((airSpeedZ) > 0)
{
    airSpeedZ += verticalMove / 10 * smoothStopIntensity * Time.unscaledDeltaTime;
    if (airSpeedZ < 0) airSpeedZ = 0;
}
else if ((airSpeedZ) < 0)
{
    airSpeedZ += verticalMove / 10 * smoothStopIntensity * Time.unscaledDeltaTime;
    if (airSpeedZ > 0) airSpeedZ = 0;
}

//////////////////////////////////////////////////////////////////

MovePlayer(horizontalMove, verticalMove);
JumpPlayer(horizontalMove, verticalMove);
```

- 플레이어 이동에 관한 값은 키보드의 WASD를 통해 4방향에 대해 -1, 0 ,1 값으로 전달받는다.
- 점프 중에는 자유로운 이동이 불가능하도록 이동에 영향을 주는 airSpeed를 추가하였다. airSpeed는 플레이어가 공중에 있을 때의 스피드를 말한다.
  - ■ 해당 부분으로 인해 점프 도중 방향을 쉽게 바꿀 수 없다.

    (타 TPS 장르처럼 움직임에 현실감을 부여하고, 부자연스러움을 방지함.)
- MovePlayer 함수 : 플레이어로부터 전달받은 값들로 캐릭터를 움직인다.
- JumpPlayer 함수 : 플레이어의 점프 가능성(공중 및 경사에선 불가)을 체크하고, 점프 버튼을 누를 시 점프를 가능하게 함.

3. **PlayerAttackSkill :** 플레이어의 공격 및 스킬 사용에 대한 부분을 담당.

[코드 설명]

```
private void FixedUpdate()
{
    Cursor.visible = false;
    Cursor.lockState = CursorLockMode.Locked;
    // Mouse Fix.

    if (fireCountdown > 0) // fireCountdown : 일반 공격의 대기를 위한 값
    {
        fireCountdown -= Time.deltaTime; // 0까지 줄어들도록 함.
    }
    if (!target) // End Attack when target is null.
    {
        Player.instance.animationController.AttackEnd();
        targetIsActive = false;
    }

    #region View Distance
    // VIEW DISTANCE ------------------------------------
    var col = Physics.OverlapSphere(transform.position, viewDistance, monster); // 근거리 적을 타겟팅 가능하도록 감지 영역을 생성함.

    screenTargets.Clear(); // screenTargets : 자동 적 조준 대상이자 화면에서 조준 가능한 적.
    target = null; // target : 공격 대상

    for (int i = 0; i < col.Length; i++)
    {
        Vector3 targetAngle = col[i].transform.position - transform.position; // targetAngle : 이를 이용해 적이 플레이어의 카메라 시점 범위 안에 들어오는지 탐지.

        // 아래 부분은 카메라 시점 내 적이 있는지 체크하는 것이다.
        // fieldOfView : 카메라 시점 범위를 나타내는 값
        if (Vector3.Angle(transform.forward, targetAngle) < fieldOfView && col[i].GetComponent<Animator>() != null && !col[i].GetComponent<Animator>().GetBool("Death"))
            screenTargets.Add(col[i].transform);
    }
    #endregion

    if (Input.GetMouseButton(1)) // Targeting : 마우스 오른쪽 키를 누르면 적을 자동 선택한다.
    {
        var targetIndex = TargetIndex();
        if (screenTargets.Count > targetIndex)
        {
            target = screenTargets[targetIndex];
        }
    }

    ///// Skill Setting//////////////////////////////////////////////////////////////////
    if (q3killData != null && q3kill != q3killData.thisSkill) q3kill = q3killData.thisSkill;
    if (e3killData != null && e3kill != e3killData.thisSkill) e3kill = e3killData.thisSkill;
    if (r3killData != null && r3kill != r3killData.thisSkill) r3kill = r3killData.thisSkill;
    if (passiveSkillData != null && passiveSkill != passiveSkillData.thisSkill) passiveSkill = passiveSkillData.thisSkill;

    ///// Skill ////////////////////////////////////////////////////////////////////////
    int usedSkillNumber = 9;

    if (q3killData != null && Input.GetKeyDown(KeyCode.Q) && Player.instance && Player.instance.mp > 0 && Player.instance.mp >= q3killData.usedMp) // Use Q Skill
    {
        usedSkillNumber = 0;
    }
    else if (e3killData != null && Input.GetKeyDown(KeyCode.E) && Player.instance && Player.instance.mp > 0 && Player.instance.mp >= e3killData.usedMp) // Use E Skill
    {
        usedSkillNumber = 1;
    }
    else if (r3killData != null && Input.GetKeyDown(KeyCode.R) && Player.instance && Player.instance.mp > 0 && Player.instance.mp >= r3killData.usedMp) // Use R Skill
    {
        usedSkillNumber = 2;
    }
```

- FixedUpdate 에서 플레이어 입력 처리와 대부분의 중요 작업을 진행한다.
- 이곳에서는 커서를 보이지 않게 숨기고, 커서가 화면 밖으로 나가지 않도록 고정했다.
- 일반 공격이 에너지 탄을 날리는 것인데 이 공격은 타겟을 설정해 날릴 수 있어 해당 부분에 대한 처리를 진행하였다.
- 스킬의 경우 Q,W,E 등의 키로 사용이 가능한데 해당 부분도 이곳에서 구현되어 현재 보유중인 스킬이 사용되도록 조정하고 있다.

4. **PlayerAnimationController :** 플레이어 캐릭터가 상황에 맞는 애니메이션을 재생하도록 돕는 클래스이다. 다른 클래스에서 애니메이션 재생을 위해 접근하기도 한다.

[코드 설명]

```
namespace ROC
{
    참조 19개
    internal static class AnimatorHashID
    {
        internal static readonly int OnAttackID = Animator.StringToHash("OnAttack");
        internal static readonly int AngleID = Animator.StringToHash("Angle");
        internal static readonly int AttackID = Animator.StringToHash("Attack");
        internal static readonly int JumpID = Animator.StringToHash("Jump");
        internal static readonly int OnAirID = Animator.StringToHash("OnAir");
        internal static readonly int JumpWaitingID = Animator.StringToHash("JumpWaiting");
        internal static readonly int VerticalID = Animator.StringToHash("Vertical");
        internal static readonly int HorizontalID = Animator.StringToHash("Horizontal");
        internal static readonly int FrontAttackID = Animator.StringToHash("FrontAttack");
        internal static readonly int HandUpID = Animator.StringToHash("HandUp");
        internal static readonly int OngroundID = Animator.StringToHash("Onground");
        internal static readonly int DeathID = Animator.StringToHash("Death");
        internal static readonly int DamagedID = Animator.StringToHash("Damaged");
        internal static readonly int RebirthID = Animator.StringToHash("Rebirth");
        internal static readonly int SpinID = Animator.StringToHash("Spin");
    }
}
```

```
public enum AnimationState
{
    Normal, Jump, Air, Attack, Die, Spin
}
```

```
private void Update()
{
    animator.SetBool(AnimatorHashID.OngroundID, Player.instance.movement.isJumping);

    if (CameraManager.fpsMode == true) // Fps mode don't need animation.
    {
        animator.Rebind();
        animator.enabled = false;
    }
    else
    {
        animator.enabled = true;
    }

    if (Input.GetMouseButtonUp(0))
    {
        AttackEnd();
    }

    if(state == AnimationState.Spin && !Player.instance.movement.isSpin)
    {
        Player.instance.movement.isSpin = true;
    }

    if (state != AnimationState.Spin && Player.instance.movement.isSpin)
    {
        Player.instance.movement.isSpin = false;
    }
}
```

- AnimatorHashID 에서 저장한 해시 값을 이용하여 애니메이션을 호출해 애니메이터의 성능을 높였다.
- Update 에선 점프, 공격 애니메이션을 체크하고, fpsMode 가 참(1 인칭 시점)일 경우

  애니메이션을 중단시키고 있다. 또한, 애니메이션에는 각각의 상태(enum 을 이용하여 제작)가 존재하며 각 상태에 따라 별도의 처리(이동, 공격 처리 등)를 진행시키고 있다.

[카메라 관련 클래스]

1. **CameraManager :** 플레이어가 보는 카메라에 대한 모든 처리가 들어가는 클래스이다.

   해당 카메라는 1인칭 시점과 3인칭 시점의 카메라 기능이 나뉘는데 1인칭 시점의 카메라는 다른 게임과 동일하나 3인칭 시점의 카메라는 게임 플레이에 초점을 맞추어 보여줘 일반적인 3인칭 시점과 조금 다르게 만들어졌다. 또한, 3인칭 시점에서 캐릭터가 벽에 가려지지 않게 만들어 플레이어 캐릭터를 볼 수 없는 상황이 거의 없다.

[코드 설명]

```
// Singleton stuff
참조 3개
public static CameraManager instance { get; private set; }
public static bool fpsMode; // if fpsMode = true, camera will changed like first person shooting game.

Unity 메시지 | 참조 0개
private void Awake()
{
    instance = this;
}

Unity 메시지 | 참조 0개
private void Start()
{
    targetPosition = Player.instance.cameraPos.middleViewCamera.transform;
    target = Player.instance.transform;
    earlyTopAngle = topAngle;
    earlyDownAngle = downAngle;

    //Shake noise setting
    float rand = 32.0f;
    noiseOffset.x = Random.Range(0.0f, rand);
    noiseOffset.y = Random.Range(0.0f, rand);
    noiseOffset.z = Random.Range(0.0f, rand);
    // semi-disable clipping
    Camera.main.nearClipPlane = 0.01f;
}

Unity 메시지 | 참조 0개
private void Update()
{
    // Prevent exception from method calls below
    if (!target) return;

    if (!stop)
    {
        MoveCamera();
        RotateCamera();
    }
    // Toggle fps mode (first-person-perspective?)
    if (Input.GetKeyDown(KeyCode.V))
    {
        if (bookVisibleFps)
        {
            fpsMode = !fpsMode;
            // Set visibility of held book
            bookVisibleFps.SetActive(fpsMode);
        }
    }

    if (timeRemaining <= 0)
        return;

    float deltaTime = Time.deltaTime;
    timeRemaining -= deltaTime;
    float noiseOffsetDelta = deltaTime * frequency;

    noiseOffset.x += noiseOffsetDelta;
    noiseOffset.y += noiseOffsetDelta;
    noiseOffset.z += noiseOffsetDelta;

    noise.x = Mathf.PerlinNoise(noiseOffset.x, 0.0f);
    noise.y = Mathf.PerlinNoise(noiseOffset.y, 1.0f);
    noise.z = Mathf.PerlinNoise(noiseOffset.z, 2.0f);

    noise -= Vector3.one * 0.5f;
    noise *= amplitude;

    float agePercent = 1.0f - (timeRemaining / duration);
    noise *= smoothCurve.Evaluate(agePercent);
}
```

```
public IEnumerator Shake(float amp, float freq, float dur, float wait)
{
    yield return new WaitForSeconds(wait);
    float rand = 32.0f;
    noiseOffset.x = Random.Range(0.0f, rand);
    noiseOffset.y = Random.Range(0.0f, rand);
    noiseOffset.z = Random.Range(0.0f, rand);
    amplitude = amp;
    frequency = freq;
    duration = dur;
    timeRemaining += dur;
    if (timeRemaining > dur)
    {
        timeRemaining = dur;
    }
}
```

- Start 함수 내에선 초기값을 설정하게 되는데 여기서 설정하는 값들은 이후에 카메라 효과나 카메라 거리와 각도 등을 설정할 때 사용한다.
- Update 함수에선 시점을 바꾸는 키를 동작하게 하거나 화면 흔들림 효과가 작동하면 이를 서서히 사라지도록 조정하고 있다.
- MoveCamera 와 RotateCamera 는 카메라의 움직임을 마우스로 조작할 수 있게 하는 함수이다.
- RotateCamera 함수는 3 인칭 시점에선 추가적으로 카메라 각도나 카메라와 캐릭터의 거리를 조절하도록 하였다. 이때 카메라의 위치, 각도들은 플레이어 내부에 Camera 오브젝트를 자식으로 두어 이를 이용함.

2. **MouseSmoothInterpolation :** 3인칭 시점일 경우 마우스 y축 이동에 따라 플레이어의 거리가 변화하는데 이때 이를 자연스럽게 하기 위해 별도로 보간 처리를 구현함.

```
public class MouseSmoothInterpolation : MonoBehaviour
{
    private readonly Vector3 top = new(0.12f, -1.41f, -1.82f); // Interpolate Camera Move (TopView).
    private readonly Vector3 bottom = new(0.12f, 1.46f, -0.69f); // Interpolate Camera Move (DownView).
    private Vector3 original;

    private Transform cam; // camera transform.

    Unity 메시지 | 참조 0개
    private void Awake()
    {
        original = transform.localPosition; // Original middle camera position.
    }
    Unity 메시지 | 참조 0개
    private void Start()
    {
        cam = Camera.main.transform; // Main camera position.
    }
    Unity 메시지 | 참조 0개
    private void Update()
    {
        // This code's purpose is that Camera move is more slowly.
        if (CameraManager.instance.targetNum == 1)
        {
            if (Input.GetAxisRaw("Mouse Y") > 0.1f)
            {
                LocalPosition((cam.localEulerAngles.x > 300 && cam.localEulerAngles.x < 350) ? top : original); // Camera move up slowly.
            }
            else if (Input.GetAxisRaw("Mouse Y") < -0.1f)
            {
                LocalPosition((!(cam.localEulerAngles.x > 300 && cam.localEulerAngles.x < 360) && cam.localEulerAngles.x > 10) ? bottom : original); // Camera move down slowly.
            }
        }
    }
    참조 2개
    public void LocalPosition(Vector3 pos) // Camera position is moving in real time.
    {
        float speed = 10f * Time.deltaTime; // Camera move speed
        Vector3 localPos = transform.localPosition;
        transform.localPosition += (pos - localPos) * speed;
    }
}
```

- Update에선 마우스 Y축 이동에 따라 카메라가 다르게 이동하도록 조절함.
- LocalPosition 함수에선 카메라의 이동을 처리한다.

[건축 관련 클래스]

1. **StructureSpawn_Test** : 플레이어가 가지고 있는 아이템을 땅에 설치 가능 하도록 하는 클래스이다. 게임에서 사용될 여부가 불분명한 상태에서 테스트 목적으로 만든 클래스였으나 성능이 괜찮아서 게임에서 사용하게 되었다.

[코드 설명]

```
void Update()
{
    if (target)
    {
        Player.instance.ui.monsterUI.SetActive(true);  // monsterUI => Turret's information UI.
    }
    else
    {
        Player.instance.ui.monsterUI.SetActive(false);
    }

    bool changeNotWork = false;
    if (Time.timeScale != 0)
    {
        if (Input.GetKeyDown(KeyCode.B)) // On/Off structure mode.
        {
            if (skillWindow)
            {
                if (skillWindow.activeSelf == false)
                {
                    Player.instance.playerAttackSkill.enabled = true;

                    if (structureMode == false)
                    {
                        if (area && areaLayer != 0) area.SetActive(true); // Visible area.
                        Player.instance.playerAttackSkill.isAttack = false;
                        structureMode = true;
                        ResetItemChange();
                    }
                    else
                    {
                        if (area && areaLayer != 0) area.SetActive(false); // Unvisible area.
                        structureMode = false;

                    }
                }
            }
        }
        // Change Object//////////////////////////

        for(int i= 0; i < 4; i++)
        {
            if (PlayerSaveData.itemList.Count > 0 && PlayerSaveData.itemList[i] != "1")
            {
                changeNotWork = false;
                break;
            }
                changeNotWork = true;
        }
        if (!changeNotWork)
        {
            if (Input.GetAxisRaw("Mouse ScrollWheel") > 0)
            {
                if (skillWindow)
                {
                    if (skillWindow.activeSelf == false)
                    {
                        ChangeNext();
                    }
                }
            }
            else if (Input.GetAxisRaw("Mouse ScrollWheel") < 0)
            {
                if (skillWindow)
                {
                    if (skillWindow.activeSelf == false)
                    {
                        ChangePrevious();
                    }
                }
            }
        }
    }
        //// Area is exist //////////

        if (areaLayer != 0)
        {
            if (isArea == false)
            {
                if (target != null)
                {
                    Destroy(target);
                }
                return;
            }
        }
        ////////////////////////////
```

- Update 문에서는 플레이어가 B 버튼을 통해 건축 모드로 전환이 가능하도록 하였다.
    - ■ 건축모드에선 전투가 불가능하다. 또한, 건축 모드 전환 시 이전에 건축 모드에서 선택했던 아이템이 선택되는게 아니라 가장 처음에 나왔던 아이템으로 바뀌도록 했다.
- 또한, area 변수로 할당된 건축 영역을 On/Off 하여 건축을 가능하게 하였다.
- 여기서 마우스 휠을 이용해 보유하고 있는 아이템을 선택하는 기능도 구현됐다.
    - ■ 여기서 target 변수는 사물을 땅에 배치하기전에 화면에 잠깐 보여주는 투명한 사물 오브젝트를 의미한다.

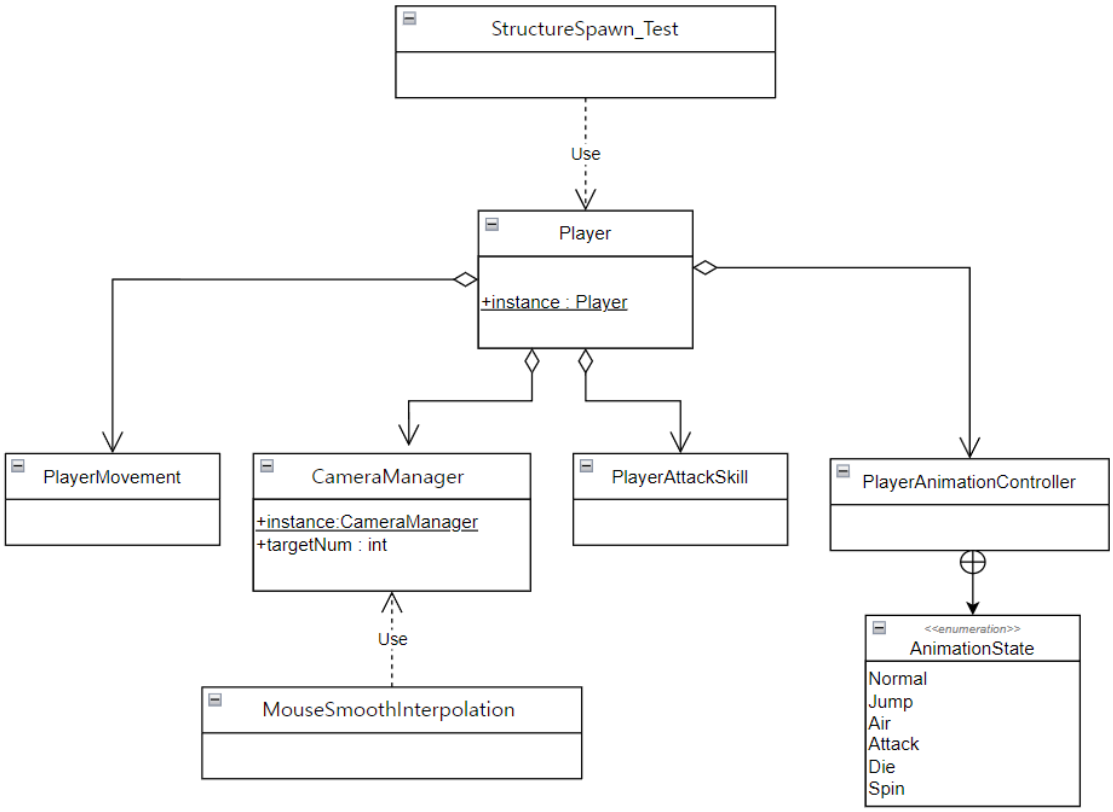
<사물의 설치 코드 설명 – Update 함수 아래쪽에 위치함>

```
if (Physics.Raycast(ray, out hit, distance, installLayer))
{
    Debug.DrawRay(hit.point, hit.normal, Color.blue);

    // install fixed position.
    GridStructure grid;
    if ((grid = hit.collider.gameObject.GetComponent<GridStructure>()) != null)
    {
        changePosValue = grid.posPreview;
    }

    else changePosValue = new(0, 0, 0);

    if (target == null)
    {
        if (selectedPrefab[selectNumber])
        {
            target = Instantiate(selectedPrefab[selectNumber], hit.point, Quaternion.identity); // View Sample Object.
            color = target.GetComponentInChildren<Renderer>().material.color;
        }
    }
    else if (target != null && target.transform.position != hit.point)
    {
        if (changePosValue != new Vector3(0, 0, 0))
        {
            target.transform.position = changePosValue;
            target.transform.rotation = grid.transform.rotation;
        }
        else
        {
            target.transform.position = hit.point;
            target.transform.up = hit.normal; // Set rotation according to ground slope.
        }
    }
}
else if (target != null)
{
    Destroy(target);
}
```

- Raycast 를 이용하여 주변에 설치가능한 땅이 있는지 판단하고,

  설치가 가능한 땅일 경우에만 투명한 사물 오브젝트를 생성해 화면에 띄우고 target 에 저장한다. 해당 투명 사물 오브젝트는 마우스를 움직일 때 같이 움직이도록 하였다.
    - ■ 상자와 같은 물체는 마인크래프트처럼 각 방향에 붙여서 배치할 수 있는데 이는 GridStructure 라는 클래스를 이용한다.
- 투명 사물 오브젝트는 실제로 배치되는 모습을 똑같이 보여주도록 했다.
- 사물을 배치할 때는 배치하는 땅에 기울기에 따라 사물의 기울기도 변하게 했다.

[UML 요약]


StructureSpawn_Test
Use
Player
+instance : Player
PlayerMovement
CameraManager
+instance:CameraManager
+targetNum : int
PlayerAttackSkill
PlayerAnimationController
Use
MouseSmoothInterpolation
<<enumeration>>
AnimationState
Normal
Jump
Air
Attack
Die
Spin