<포트폴리오
기술적 문서>

지원자 : 김 지 호
프로젝트 규모 : 개 인

<목 차>

1. 변수 네이밍 규칙

- 유니티 엔진은 c#을 스크립트 언어로 사용한다.
- c#은 기본적으로 클래스 단위로 작성이 된다.
- 따라서 메서드가 아닌 필드는 모두 m_TypeNameNumber의 형식을 갖는다.
- Number의 경우, 두 개 이상 사용하는 경우에만 사용한다.
- Number는 00 부터 +1 씩 증가한다.
- 필드가 아닌 메서드는 내부적으로 사용하는 것과 외부로 노출시켜 사용되는 것으로 분류한다.
- 전자는 F_NameNumber, 후자는 C_NameNumber으로 표기한다.
- F는 Function의 첫 글자이다.
- C는 Class의 첫 글자이다.
- 프로퍼티는 P_TypeNameNumber의 형식을 따른다.
- P는 Property의 첫 글자이다.
- Enum은 E_TypeNameNumber의 형식을 따른다.
- E는 Enum의 첫 글자이다.

Type	의미
n	Int
f	Float
v	Vector3
vec	
is	Bool
p	유니티 엔진 에디터에 노출 시킨 변수 혹은 게임 오브젝트에 있는 스크립트 컴포넌트
TypeArr	해당 Type의 배열
str	String
Rect	Rect 타입(유니티 전용)

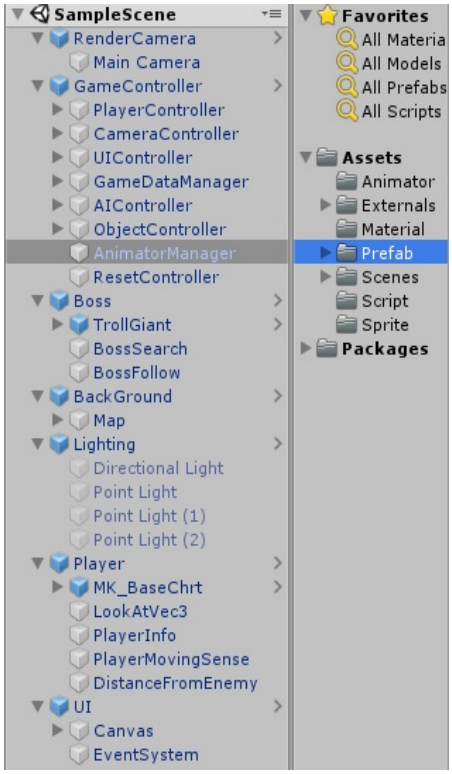
↑ <Type 에 관련된 표> ↓<예시 사진>

```
#region Public Variables

[SerializeField]
[Tooltip("Proper Player Prefab")]
public GameObject m_pPlayer;

[SerializeField]
[Tooltip("Proper Player Inventory Controller Prefab")]
public GameObject m_pUIInven;

[SerializeField]
[Range(0.001f, 0.01f)]
[Tooltip("Min 0.001f Max 0.01f")]
public float m_fMoveSpeed;

[SerializeField]
[Range(0.0f, 0.1f)]
[Tooltip("Min 0.0f Max 1.0f")]
public float m_fRotateSpeed;

[SerializeField]
public GameObject m_pAnimManager;

[SerializeField]
public GameObject m_pUIMenu;


#endregion
```

```
AnimatorManager m_pAnimScript;
UIMenuMsg m_pUIMenuScript;
UIInvenWork m_pUIInvenWorkScript;

float m_fTime;
float m_fHandleCoord;

bool m_isMove;

public bool P_IsPlayerMove
{
    get { return m_isMove; }

}

bool m_isZMove;
bool m_isXMove;

bool m_isAttack;
bool m_isRotate;

//bool m_isGameExit;
bool m_isAniMotion;
bool m_isRunAction;
bool m_isModRunSpeed;
bool m_isActiveInven;
bool m_isActivePauseMenu;
bool m_isInvenPopUp;
bool m_isGamePopUp;

public bool P_IsActiveInven
{
    get { return m_isActiveInven; }
    set { m_isActiveInven = value; }
}
```

```
#region Class Method

public bool C_GetIsRotate()
{
    return m_isRotate;
}

public float C_GetfYRotate()
{
    return m_fYRotate;
}

public Vector3 C_GetVec3Translate()
{
    return m_dir;
}
void F_CheckInput()...

void F_CheckMotion()
{
    if (m_pAnimScript[m_pAnimScript.P_nPlayer].GetCurrentAnimatorStateInfo
        (m_pAnimScript[m_pAnimScript.P_nPlayer].GetLayerIndex("Base Layer")).IsName("MK_attackWhirl"))
    {
```

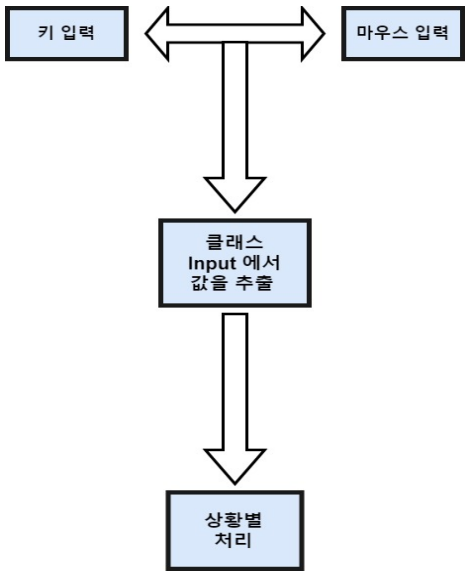
2. 하이어라키에서의 게임 오브젝트 분류



게임 오브젝트의 OOP의 원리를 적용시켰다. 일반적으로 유니티 개발자들은 스크립트에서 외부의 모듈을 사용하기 위해 특정한 스크립트를 상속을 해서 기능을 구현하는 경향이 존재한다.

이는 기존의 네이티브한 개발 방식의 습관이다. 엔진에서 이러한 습관대로 할 경우, 스파게티 소스 코드가 될 확률이 증가하고, 버그를 픽스하기가 힘들다.

3. 플레이어의 키 입력




```
void F_CheckInput()
{
    if (Input.GetKeyDown(KeyCode.W))
    {
        m_isMove = true;
        m_isZMove = true;
    }
    else if (Input.GetKeyUp(KeyCode.W))
    {
        m_isZMove = false;
    }
    if (Input.GetKeyDown(KeyCode.S))
    {
        m_isMove = true;
        m_isZMove = true;

    }
    else if (Input.GetKeyUp(KeyCode.S))
    {
        m_isZMove = false;
    }

    if (Input.GetKeyDown(KeyCode.D))
    {
        m_isMove = true;
        m_isXMove = true;

    }
    else if (Input.GetKeyUp(KeyCode.D))
    {
        m_isXMove = false;
    }
```

```
void F_CheckMotion()
{
    if (m_pAnimScript[m_pAnimScript.P_nPlayer].GetCurrentAnimatorStateInfo
        (m_pAnimScript[m_pAnimScript.P_nPlayer].GetLayerIndex("Base Layer")).IsName("MK_attackWhirl"))
    {
        m_isRotate = false;
        m_fYRotate *= 0.0f;

    }
}

void F_Action()
{
    if (m_isGamePopUp || m_isInvenPopUp)
    {

    }
    else
    {
        if (m_isRotate )
        {
            if (!m_isMove && !m_isAttack && !m_pAnimScript[m_pAnimScript.P_nPlayer].GetCurrentAnimatorStateInfo
                (m_pAnimScript[m_pAnimScript.P_nPlayer].GetLayerIndex("Base Layer")).IsName("MK_attackWhirl"))
            {
                m_fYRotate = Input.GetAxisRaw("Mouse X");

                m_fYRotate *= m_fTime * m_fRotateSpeed;

            }

            if (m_pAnimScript[m_pAnimScript.P_nPlayer].GetCurrentAnimatorStateInfo
                (m_pAnimScript[m_pAnimScript.P_nPlayer].GetLayerIndex("Base Layer")).IsName("MK_attackWhirl"))
            {
                m_fYRotate *= 0.0f;

            }
            m_pPlayer.transform.Rotate(Vector3.up, m_fYRotate, Space.Self);
            m_isMove = false;
            m_isAttack = false;
        }


        if (m_isMove)
        {
```

```
#region Unity Event Function

// Start is called before the first frame update
void Start()
{

    m_pAnimScript = m_pAnimManager.GetComponent<AnimatorManager>();
    m_pUIMenuScript = m_pUIMenu.GetComponent<UIMenuMsg>();
    m_pUIInvenWorkScript = m_pUIInven.GetComponent<UIInvenWork>();

    m_fYRotate = 0.0f;
    m_fXTranslate = 0.0f;
    m_fZTranslate = 0.0f;
    m_fMoveSpeed = 20.0f;
    m_fRotateSpeed = 100.0f;
    m_fTime = Time.deltaTime;
    m_isRunAction = false;
    m_isModRunSpeed = false;
    m_isActiveInven = false;
    m_isActivePauseMenu = false;
    m_isInvenPopUp = false;
    m_isGamePopUp = false;
    m_isMove = false;
    m_isRotate = false;
    m_isZMove = false;
    m_isXMove = false;

    m_dir = Vector3.zero;
    m_fHandleCoord = -1.0f;
    m_nClikCount = 0;
}

// Update is called once per frame
void Update()
{
    F_CheckInput();

    F_CheckMotion();

    F_Action();

}
```

플레이어의 키 입력 처리는 다음과 같다. 키보드나 마우스에서 입력한 값을 유니티의 **Input** 클래스에서 추출하고, 입력의 유무를 판별한 뒤, 상황 별로 처리하면 된다. 입력의 유무를 확인하기 위해서 **Bool**형 변수를 이용을 하였다. 해당 타입의 배열을 이용한 방법도 존재하지만, 좀 더 코드의 가독성을 위해서 단일 변수를 이용하였다.

4. 플레이어의 이동

```
if (m_isMove)
{
    if (!m_isRotate)
    {
        m_fZTranslate = Input.GetAxisRaw("Vertical");
        m_fXTranslate = Input.GetAxisRaw("Horizontal");

        if (m_pAnimScript[m_pAnimScript.P_nPlayer].GetCurrentAnimatorStateInfo
            (m_pAnimScript[m_pAnimScript.P_nPlayer].GetLayerIndex("Base Layer")).IsName("MK_attackWhirl"))
        {
            m_isAniMotion = false;


            m_fXTranslate *= 0.0f;
            m_fZTranslate *= 0.0f;

        }
        else
        {
            m_isAniMotion = true;

            m_fXTranslate *= m_fTime * m_fMoveSpeed;
            m_fZTranslate *= m_fTime * m_fMoveSpeed;

        }

        m_vDir = new Vector3(m_fXTranslate, 0.0f, m_fZTranslate);

    }

    if (m_isZMove)
    {
        m_pPlayer.transform.Translate(Vector3.forward * m_fZTranslate * m_fHandleCoord);
    }
    if (m_isXMove)
    {
        m_pPlayer.transform.Translate(Vector3.right * m_fXTranslate * m_fHandleCoord);
    }
}
else
{
    m_isAniMotion = false;
}
```

*특정한 축에 관한 이동량 : **f(x)***

Input**에서 추출한 값 : **I

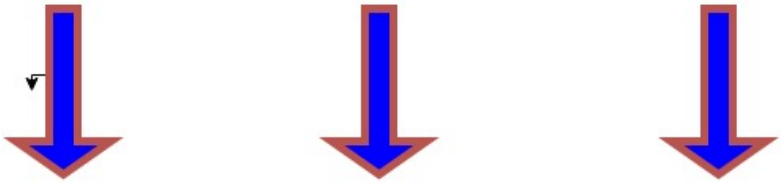
Time**의 값 : **T

*이동속도의 값 : **S***

*이동정지의 값 : **Z***

HandleCoord** : 캐릭터가 앞으로 가는 방향의 값, **H

*이동방향의 벡터 : **V***



<최종 이동량 계산식>

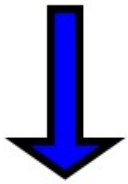
$f(x)_{n+1} = f(x)_n$

$* I * T * S$

<이동 정지 계산식>

$f(x)_{n+1} = f(x)_n$

$* Z$



Parameter : P

P = f(x) * H * V

Translate(*P*)

5. 플레이어의 회전

```
if (m_isRotate )
{
    if (!m_isMove && !m_isAttack && !m_pAnimScript[m_pAnimScript.P_nPlayer].GetCurrentAnimatorStateInfo
        (m_pAnimScript[m_pAnimScript.P_nPlayer].GetLayerIndex("Base Layer")).IsName("MK_attackWhirl"))
    {
        m_fYRotate = Input.GetAxisRaw("Mouse X");

        m_fYRotate *= m_fTime * m_fRotateSpeed;


    }

    if (m_pAnimScript[m_pAnimScript.P_nPlayer].GetCurrentAnimatorStateInfo
        (m_pAnimScript[m_pAnimScript.P_nPlayer].GetLayerIndex("Base Layer")).IsName("MK_attackWhirl"))
    {
        m_fYRotate *= 0.0f;

    }
    m_pPlayer.transform.Rotate(Vector3.up, m_fYRotate, Space.Self);
    m_isMove = false;
    m_isAttack = false;
}
```

특정한 축에 관한 회전량 : **f(x)**

Input에서 추출한 값 : **I**

Time의 값 : **T**

회전속도의 값 : **S**

회전정지의 값 : **Z**

회전축의 벡터 : **V**

<최종 회전량 계산식>

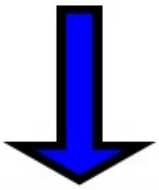
$f(x)_{n+1} = f(x)_n$

$* I * T * S$

<회전 정지 계산식>

$f(x)_{n+1} = f(x)_n$

$* Z$

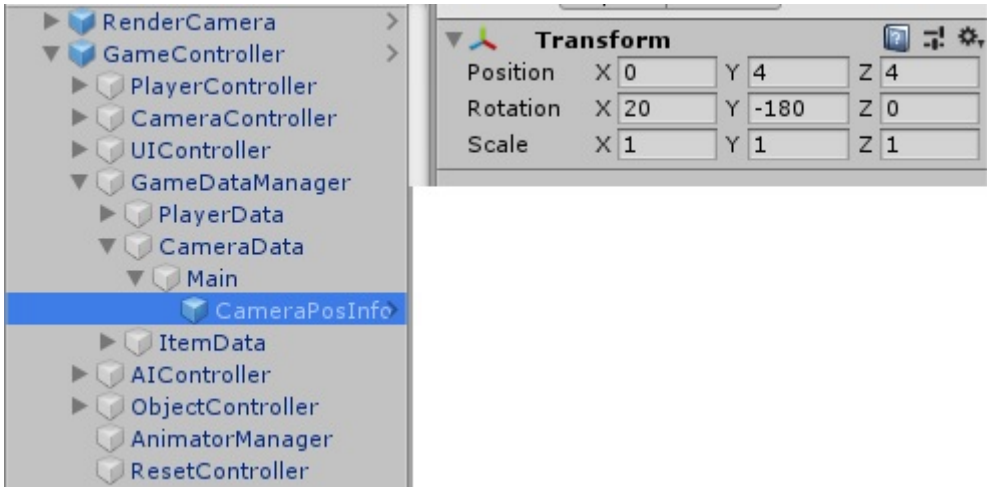


Parameter : P

P = f(x)

Rotate(V, *P, Space.Self*)

6. 카메라 제어




```
#region Public Variables

[SerializeField]
[Tooltip("Proper Player Prefab")]
public GameObject m_pPlayer;

[SerializeField]
[Tooltip("Proper Camera Prefab")]
public GameObject m_pCamera;

[SerializeField]
[Tooltip("Proper Camera Init Position Data Prefab")]
public GameObject m_pCameraInitPos;

#endregion


/// <summary>
/// Class Members in PlayerKeyController Script
/// </summary>


#region Class Field

Vector3 m_vInitPos;
Vector3 m_vLookAtPos;
Vector3 m_vDir;
Vector3 m_vNowDir;
Vector3 m_vMoveDir;
Vector3 m_vMoveNowDir;
Vector3 m_vMoveFixedDir;
Vector3 m_vMovePointDir;

PlayerKeyController m_pPlayerInputScript;
PlayerMovingSenseSystem m_pPMSSScript;

float m_fDistance;
float m_fNowDistance;
float m_fFixedDistance;
float m_fMoveFixedDistance;
float m_fMoveDistance;

#endregion
```

```
#region Class Method


void F_CalculateDirAndDistance()
{
    m_vMoveNowDir = new Vector3(m_pPlayer.transform.position.x, m_pCamera.transform.position.y + m_pPMSSScript.C_GetYMoveValue().y
        , m_pPlayer.transform.position.z);

    m_vMovePointDir = m_vMoveNowDir - m_pCamera.transform.position;
    m_fNowDistance = m_vMovePointDir.magnitude;

    m_vNowDir = m_vMoveNowDir - m_vMoveDir;

    m_fMoveFixedDistance = m_fNowDistance - m_fDistance;
    m_fMoveDistance = m_vNowDir.magnitude;

    m_vNowDir.Normalize();
}

void F_CheckAndRotateCamera()
{
    if (m_pPlayerInputScript.C_GetIsRotate())
    {

        if (!m_pPlayerInputScript.P_IsPlayerMove)
        {
            m_pCamera.transform.RotateAround(m_pPlayer.transform.position, Vector3.up, m_pPlayerInputScript.C_GetfYRotate());
        }

    }
}

void F_MoveCamera()
{
    if (m_fMoveFixedDistance != 0.0f)
    {

        m_pCamera.transform.position += m_fMoveDistance * m_vNowDir;

    }

    m_vMoveDir = m_vMoveNowDir;
}

#endregion
```

항상 특정한 거리를 유지하여
의도된 위치에서 플레이어를 향해
바라봐야만 한다.

문제점 발생!

1) 2D 와 **3D**의 차이점에서 생기는 문제

2) 플레이어의 회전에 의한 정렬 문제

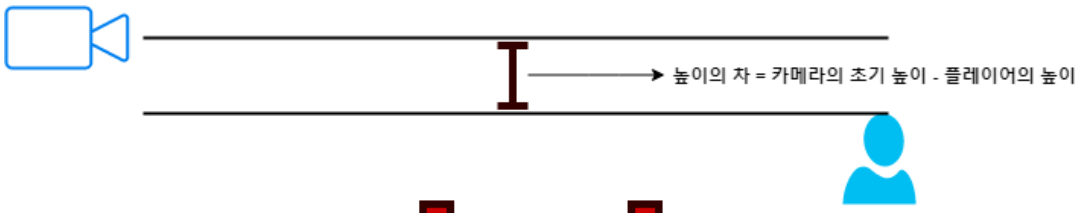
3) 충돌에 의한 높이 값의 변화 문제

```
#region Unity Event Function

// Start is called before the first frame update
void Start()
{
    m_vInitPos = m_pCameraInitPos.transform.position;

    m_pCamera.transform.position = Vector3.zero + m_pPlayer.transform.position + m_vInitPos;

    m_pPlayerInputScript = GameObject.Find("GameController").transform.Find("PlayerController")
        .Find("Input").gameObject.GetComponent<PlayerKeyController>();

    m_pPMSSScript = m_pPlayer.transform.Find("PlayerMovingSense").gameObject.GetComponent<PlayerMovingSenseSystem>();

    m_vMoveDir = new Vector3(m_pPlayer.transform.position.x, m_pCamera.transform.position.y, m_pPlayer.transform.position.z);

    m_vDir = m_pCamera.transform.position - m_vMoveDir;

    m_vMoveFixedDir = m_vMoveDir - m_pCamera.transform.position;

    m_fFixedDistance = m_vMoveFixedDir.magnitude;

    m_fDistance = m_fFixedDistance;

}

// Update is called once per frame
void Update()
{

}
void LateUpdate()
{

    F_CalculateDirAndDistance();

    F_CheckAndRotateCamera();

    F_MoveCamera();


}

#endregion
```

1) 2D 와 3D의 차이점에서 생기는 문제

<문제의 원인>

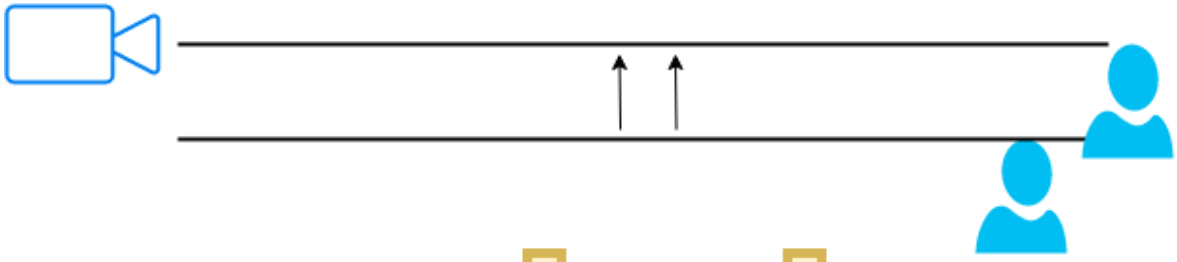
하나의 축이 늘어나서
높이 조정에 문제가 생김



**높이의 차이가 존재해서
두 위치의 차이를 이용한 방향 벡터를 구하기가
어려웠음.**

<문제의 해결>

카메라의 위치와
플레이어의 위치를
동일하게 놓으면
해결된다



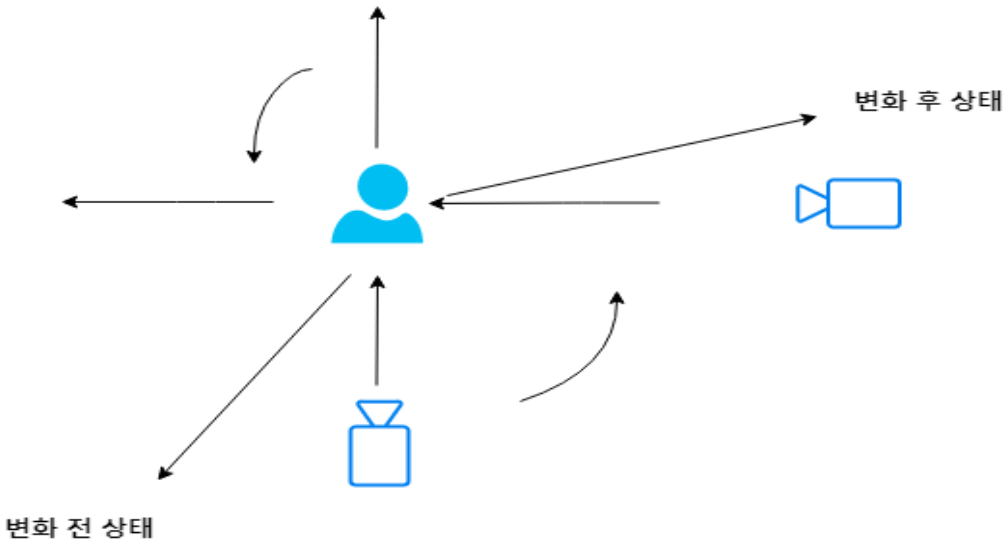
**매 주기마다 플레이어의 높이와
미리 설정된 카메라의 높이를
동일하게 해준다.**

결과적으로 방향 벡터를 구하기가 훨씬 쉬워졌다

2) 플레이어의 회전에 의한 정렬 문제

<문제의 원인>

플레이어가 회전한 뒤,
카메라의 위치를
재정렬해야 한다.



<문제의 해결>

플레이어가 회전하면
회전한 각도만큼
카메라를 정렬하면 된다.

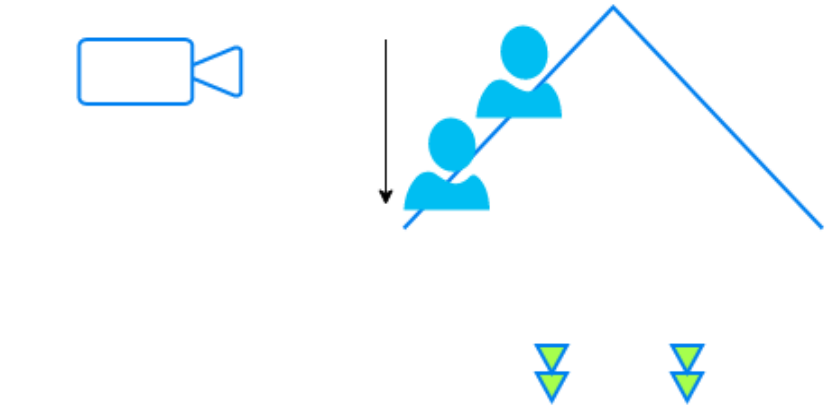
* RotateAround() 사용

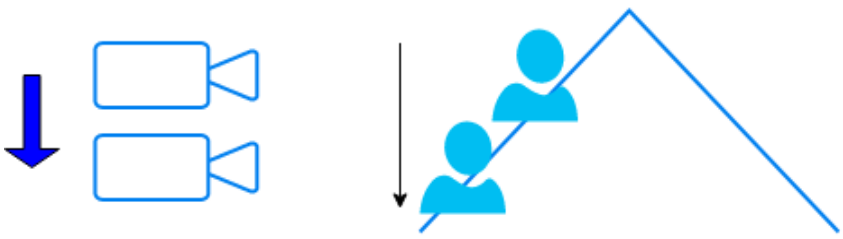
3) 충돌에 의한 높이 값의 변화 문제

```
public class PlayerMovingSenseSystem : MonoBehaviour
{
    [SerializeField]
    public GameObject m_pPlayer;


    float m_fYpos;
    float m_fPrevYpos;

    float m_fYposScalar;

    bool m_IsYMove;


    public bool C_GetIsYMove()
    {
        return m_IsYMove;
    }

    public Vector3 C_GetYMoveValue()
    {
        Vector3 tmpVec = new Vector3(0.0f, m_fYposScalar, 0.0f);
        return tmpVec;
    }
    // Start is called before the first frame update
    void Start()
    {
        m_fYpos = m_pPlayer.transform.position.y;
        m_IsYMove = false;
        m_fYposScalar = 0.0f;
    }

    // Update is called once per frame
    void Update()
    {
        m_fPrevYpos = m_fYpos;

        m_fYpos = m_pPlayer.transform.position.y;

        m_fYposScalar = m_fYpos - m_fPrevYpos;


        if(m_fYposScalar > 0.0f || m_fYposScalar < 0.0f)
        {
            m_IsYMove = true;
        }
        else
        {
            m_IsYMove = false;
        }
    }
}
```

문제의 원인

물리적 충돌로 인해서
플레이어의 높이 값이
변화한다. 그러므로
카메라의 위치를
재정렬을 해야한다





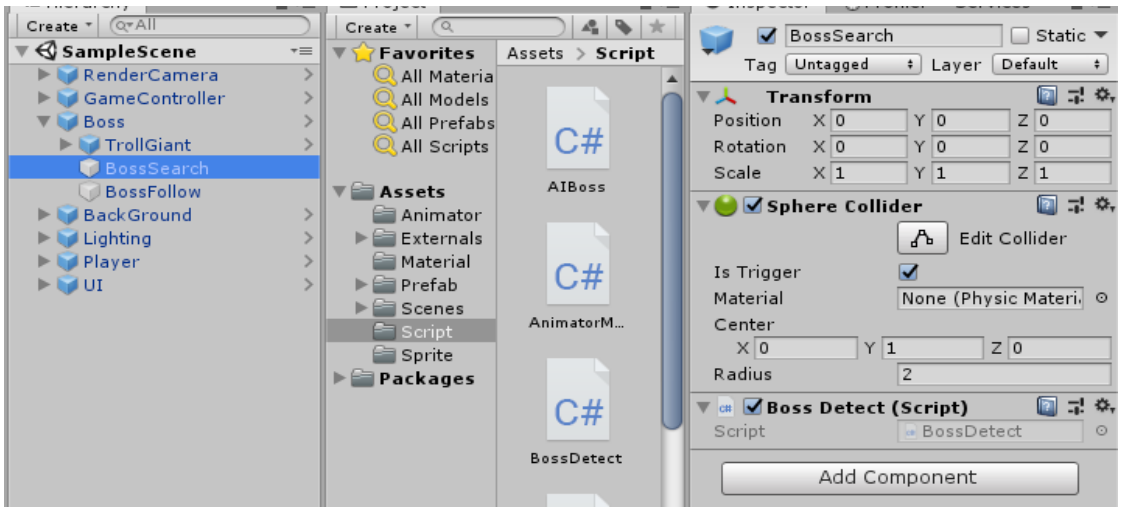
문제의 해결

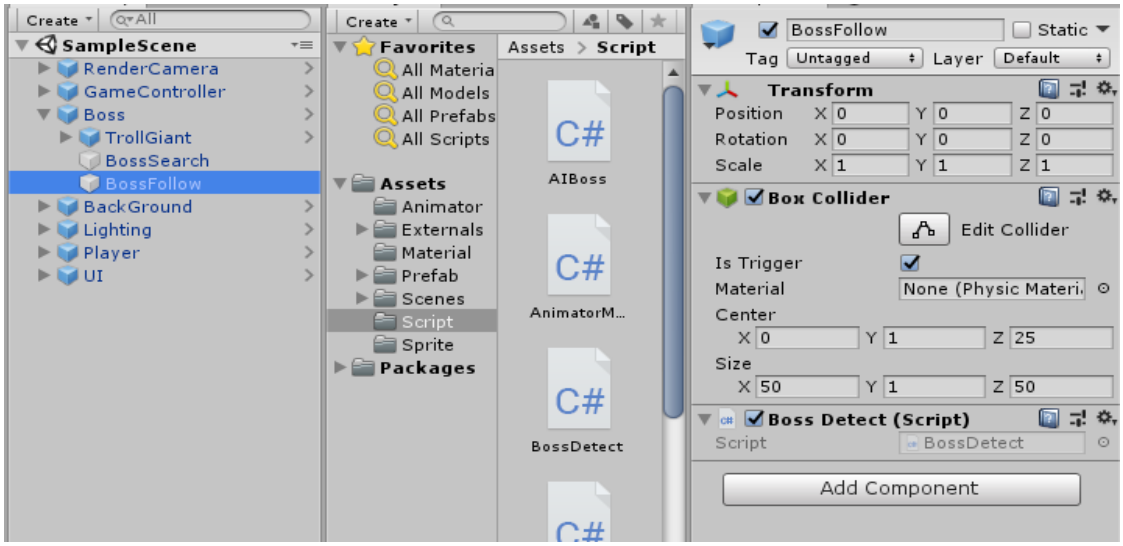
플레이어의 높이 값의 변화를
감지한다. 그리고 높이 값의 변화만큼
카메라의 위치를 재정렬해준다

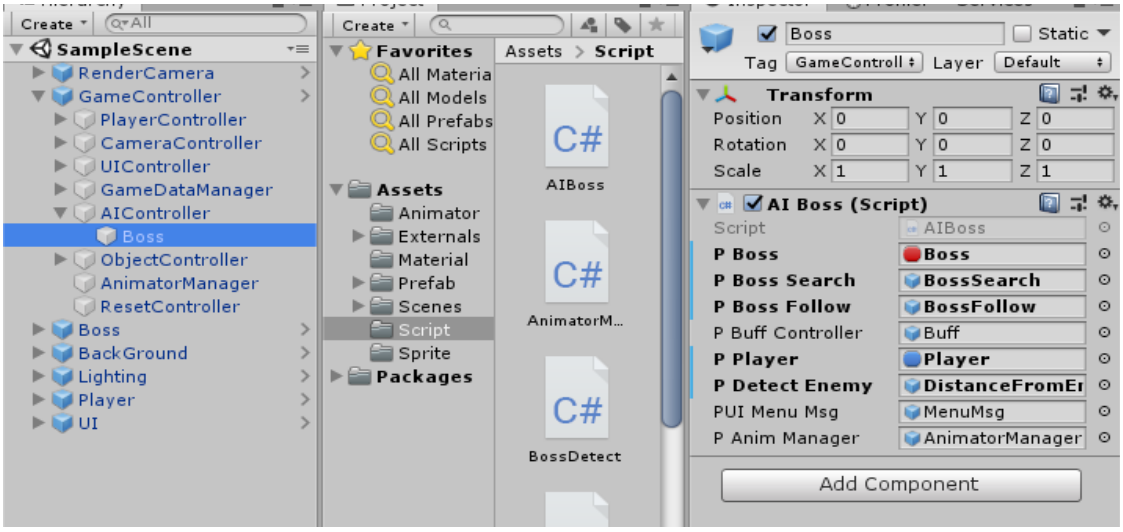
7. 물리적 충돌 감지 와 반응 시스템

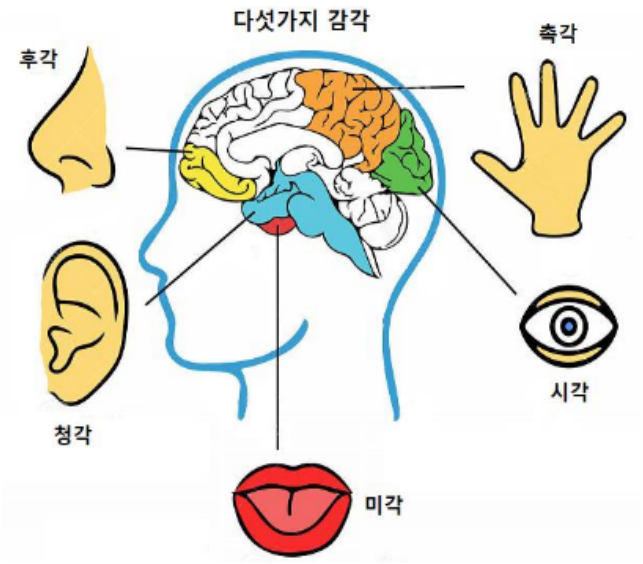
```
public class GeneralDetectPlayer : MonoBehaviour
{
    bool m_isCollision;
    bool m_isTrigger;

    // Start is called before the first frame update
    void Start()
    {
        m_isCollision = false;
        m_isTrigger = false;
    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnCollisionEnter(Collision other)
    {
        if(other.gameObject.CompareTag("Player"))
        {
            m_isCollision = true;
        }
    }

    void OnCollisionExit(Collision other)
    {
        if (other.gameObject.CompareTag("Player"))
        {
            m_isCollision = false;
        }
    }
    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            m_isTrigger = true;
        }
    }

    void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            m_isTrigger = false;
        }
    }

    public bool C_GetIsCollision()
    {
        return m_isCollision;
    }

    public bool C_GetIsTrigger()
    {
        return m_isTrigger;
    }
}
```

```
public class BossDetect : MonoBehaviour
{
    bool m_isCollision;

    // Start is called before the first frame update
    void Start()
    {
        m_isCollision = false;
    }

    // Update is called once per frame
    void Update()
    {

    }

    void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            m_isCollision = true;
        }
    }

    void OnTriggerExit(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            m_isCollision = false;
        }
    }

    public bool C_GetIsCollision()
    {
        return m_isCollision;
    }
}
```


Create
SampleScene
RenderCamera
GameController
Boss
TrollGiant
BossSearch
BossFollow
BackGround
Lighting
Player
UI
Favorites
All Materia
All Models
All Prefabs
All Scripts
Assets
Animator
Externals
Material
Prefab
Scenes
Script
Sprite
Packages
Assets > Script
AIBoss
AnimatorM...
BossDetect
BossSearch
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sphere Collider
Edit Collider
Is Trigger
Material None (Physic Materi
Center X 0 Y 1 Z 0
Radius 2
Boss Detect (Script)
Script BossDetect
Add Component


Create
SampleScene
RenderCamera
GameController
Boss
TrollGiant
BossSearch
BossFollow
BackGround
Lighting
Player
UI
Favorites
All Materia
All Models
All Prefabs
All Scripts
Assets
Animator
Externals
Material
Prefab
Scenes
Script
Sprite
Packages
Assets > Script
AIBoss
AnimatorM...
BossDetect
BossFollow
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Box Collider
Edit Collider
Is Trigger
Material None (Physic Materi
Center X 0 Y 1 Z 25
Size X 50 Y 1 Z 50
Boss Detect (Script)
Script BossDetect
Add Component


Create
SampleScene
RenderCamera
GameController
PlayerController
CameraController
UIController
GameDataManager
AIController
Boss
ObjectController
AnimatorManager
ResetController
Boss
BackGround
Lighting
Player
UI
Favorites
All Materia
All Models
All Prefabs
All Scripts
Assets
Animator
Externals
Material
Prefab
Scenes
Script
Sprite
Packages
Assets > Script
AIBoss
AnimatorM...
BossDetect
Boss
Static
Tag GameControll
Layer Default
Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
AI Boss (Script)
Script AIBoss
P Boss Boss
P Boss Search BossSearch
P Boss Follow BossFollow
P Buff Controller Buff
P Player Player
P Detect Enemy DistanceFromEr
PUI Menu Msg MenuMsg
P Anim Manager AnimatorManager
Add Component



인체의 오감 감지와
두뇌 반응
시스템을 응용함.

감지 → 반응

Detect → Process

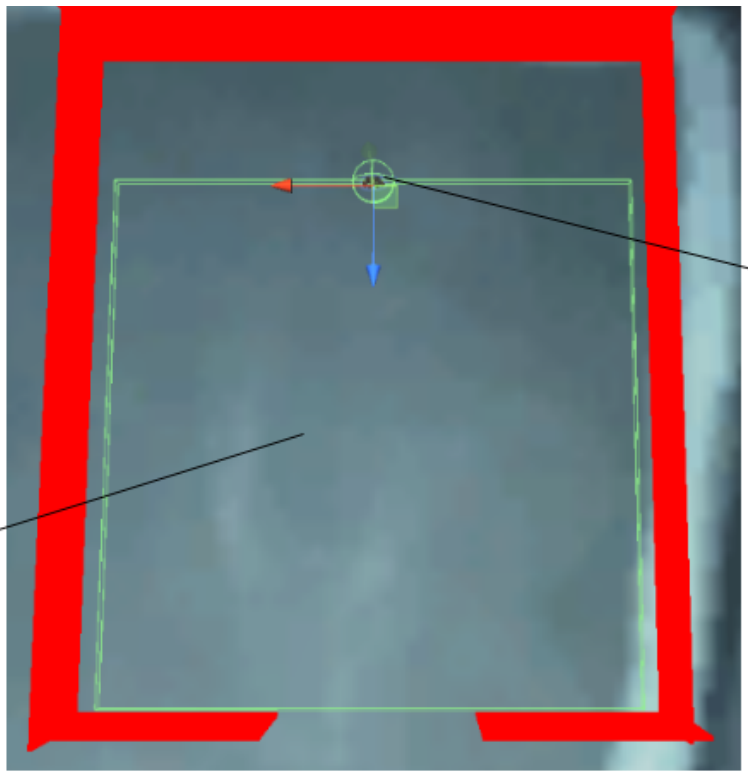
8. 보스의 AI

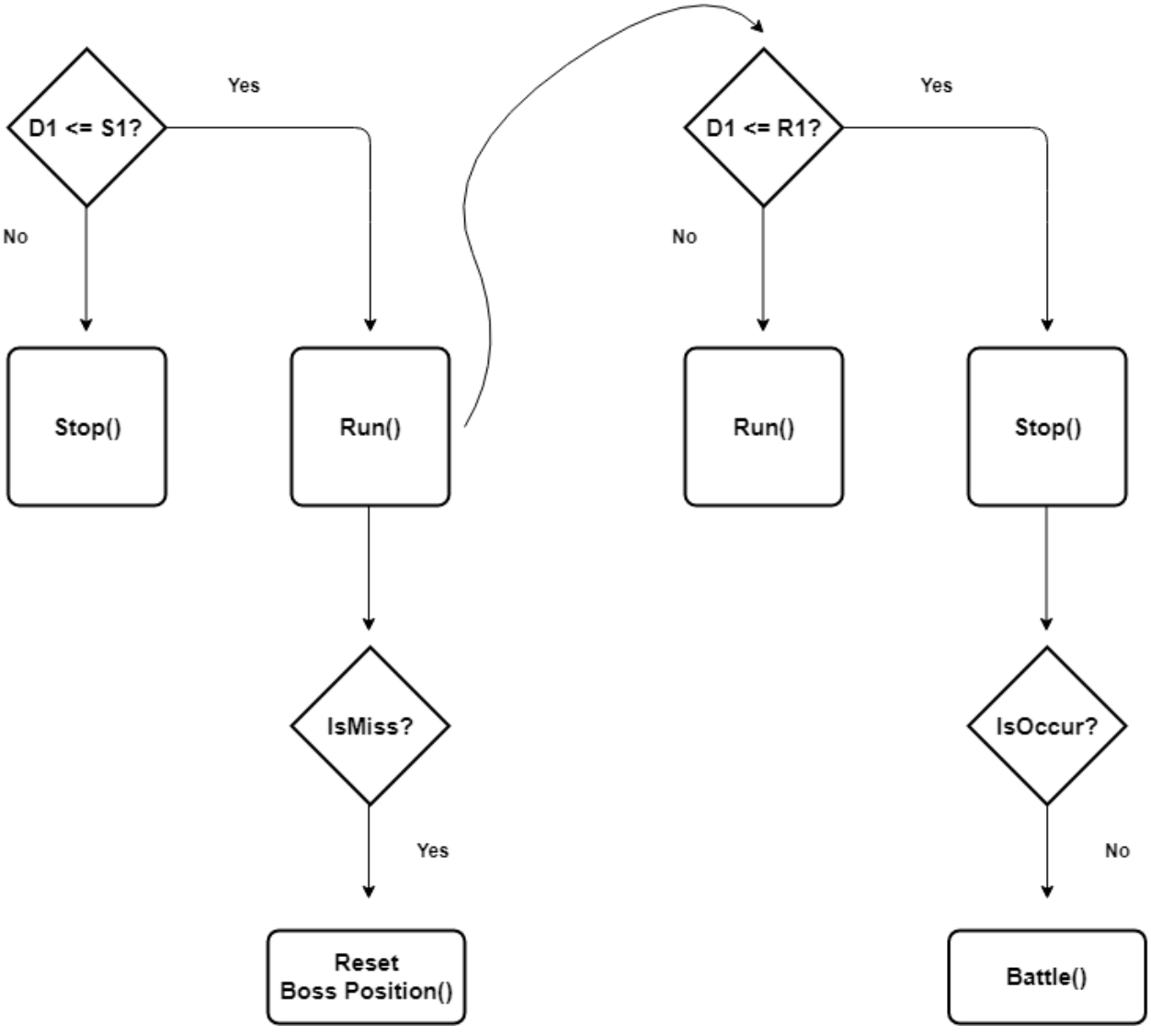
```
public class AIBoss : MonoBehaviour
{
    [SerializeField]
    public GameObject m_pBoss;

    [SerializeField]
    public GameObject m_pBossSearch;

    [SerializeField]
    public GameObject m_pBossFollow;

    [SerializeField]
    public GameObject m_pBuffController;

    [SerializeField]
    public GameObject m_pPlayer;

    [SerializeField]
    public GameObject m_pDetectEnemy;

    [SerializeField]
    public GameObject m_pUIMenuMsg;

    [SerializeField]
    public GameObject m_pAnimManager;

    BossDetect m_pBossRunScript;

    BossDetect m_pBossBattleScript;

    DetectEnemy m_pDEScript;

    BuffWorking m_pBuffWorkingScript;

    AnimatorManager m_pAnimScript;

    UIMenuMsg m_pUIMMScript;

    bool m_isOccur;

    bool m_isMiss;

    float m_fBossRunSpeed;

    float m_fBossRunTime;

    float m_fInitFromBossDistance;

    float m_fResetPosOffset;

    Vector3 m_vecBossFromInit;

    Vector3 m_vecBossDir;

    Vector3 m_vBossInitPos;

    Vector3 m_vBossResetPos;

    Quaternion m_qAngle;
```

```
void F_Run()
{
    m_pBoss.transform.LookAt(m_pPlayer.transform);

    m_vecBossDir = Vector3.forward * m_fBossRunSpeed * m_fBossRunTime;

    m_fInitFromBossDistance = m_vecBossDir.sqrMagnitude;

    m_pAnimScript[m_pAnimScript.P_nBoss].SetBool("IsRun", true);

    m_isMiss = true;

    if(m_pDEScript.C_GetIsCollision())
    {
        m_pAnimScript[m_pAnimScript.P_nBoss].SetBool("IsRun", false);
    }
    else
    {
        m_pBoss.transform.Translate(m_vecBossDir);

    }
}

void F_ResetPosition()
{
    m_vBossResetPos = m_vBossInitPos + new Vector3(0.0f, m_fResetPosOffset, 0.0f);
    m_pBoss.transform.position = Vector3.zero + m_vBossResetPos;
    m_isMiss = false;
}
bool F_Battle()
{
    if(m_pBuffWorkingScript.C_GetBuffHave())
    {
        m_pAnimScript[m_pAnimScript.P_nBoss].SetBool("IsDie", true);

        m_pUIMMScript.C_ShowClearMsg();

    }
    else
    {
        m_pAnimScript[m_pAnimScript.P_nPlayer].SetBool("IsDie", true);

        m_pUIMMScript.C_ShowDiedMsg();
    }

    return true;
}
```

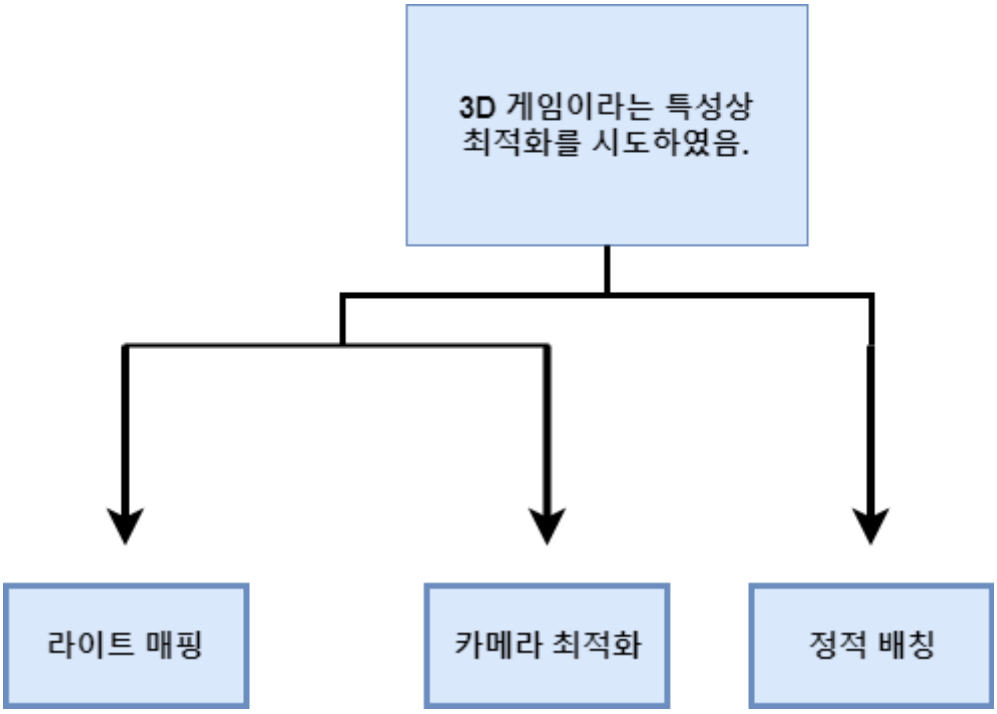
```
    void Start()
    {
        m_pBossBattleScript = m_pBossSearch.GetComponent<BossDetect>();

        m_pBossRunScript = m_pBossFollow.GetComponent<BossDetect>();

        m_pBuffWorkingScript = m_pBuffController.GetComponent<BuffWorking>();

        m_pDEScript = m_pDetectEnemy.GetComponent<DetectEnemy>();

        m_pUIMMScript = m_pUIMenuMsg.GetComponent<UIMenuMsg>();

        m_pAnimScript = m_pAnimManager.GetComponent<AnimatorManager>();


        m_isOccur = false;

        m_isMiss = false;

        m_fBossRunSpeed = 15.0f;

        m_fBossRunTime = Time.deltaTime;

        m_vecBossDir = Vector3.zero;

        m_vBossResetPos = Vector3.zero;

        m_vBossInitPos = m_pBoss.transform.position;

        m_fInitFromBossDistance = 0.0f;

        m_qAngle = Quaternion.identity;

        m_fResetPosOffset = 1.0f;
    }

    // Update is called once per frame
    void Update()
    {
        if (m_pBossRunScript.C_GetIsCollision())
        {
            F_Run();
        }
        else
        {
            m_pAnimScript[m_pAnimScript.P_nBoss].SetBool("IsRun", false);

            if (m_isMiss)
            {
                F_ResetPosition();
            }
        }

        if (m_pBossBattleScript.C_GetIsCollision())
        {
            if (!m_isOccur)
            {
                m_isOccur = F_Battle();
            }
        }
    }
}
```

보스와
플레이어의 거리
D1

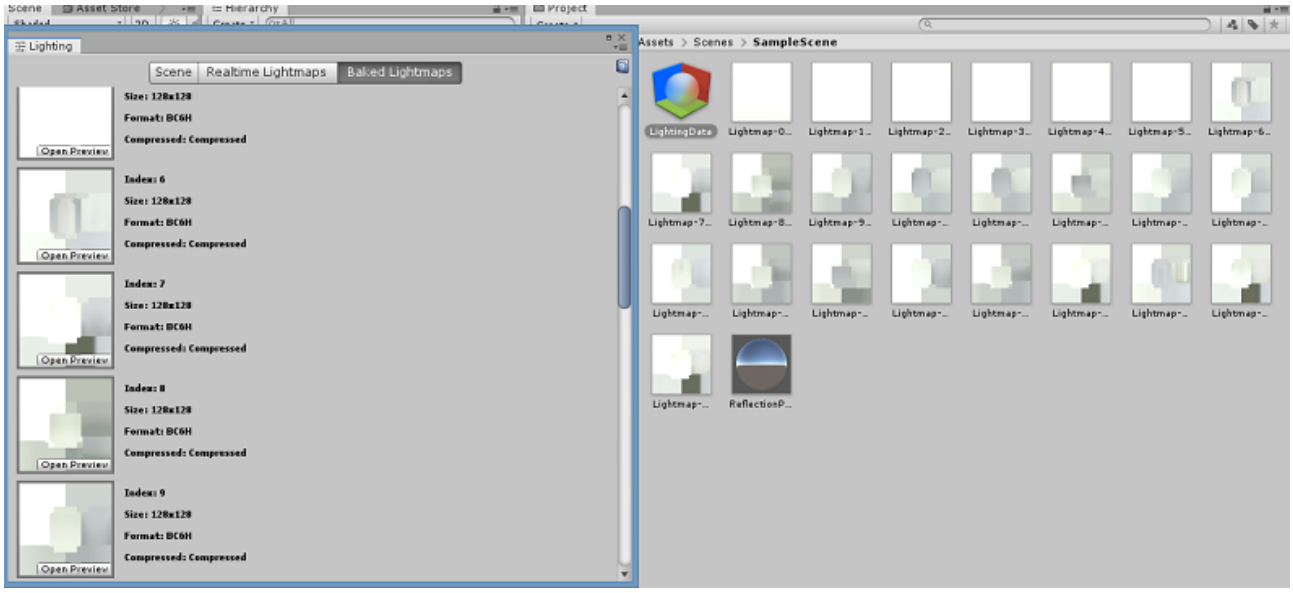




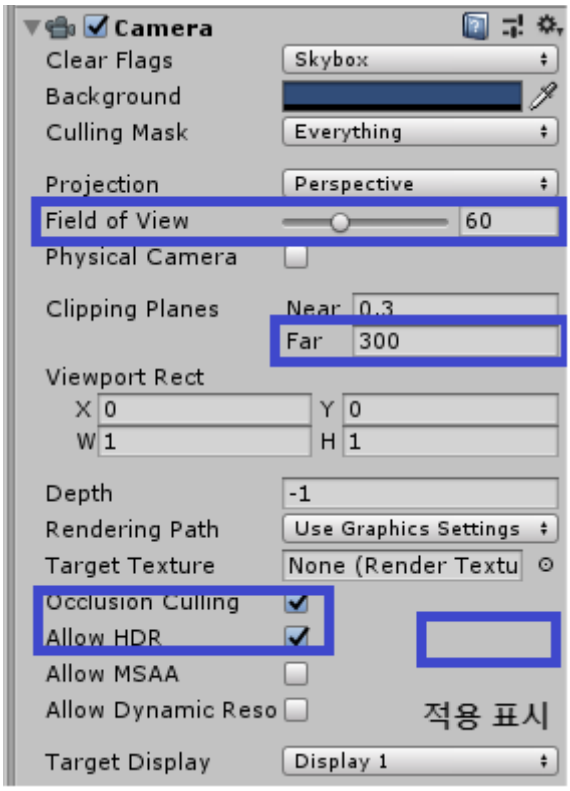
9. 프로젝트 최적화



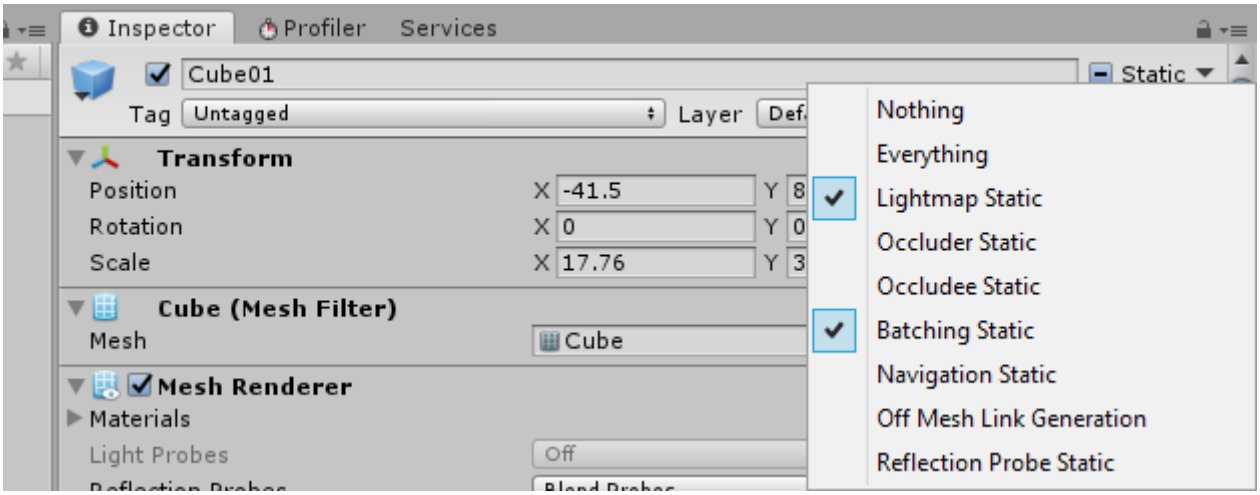
1) 라이트 매핑



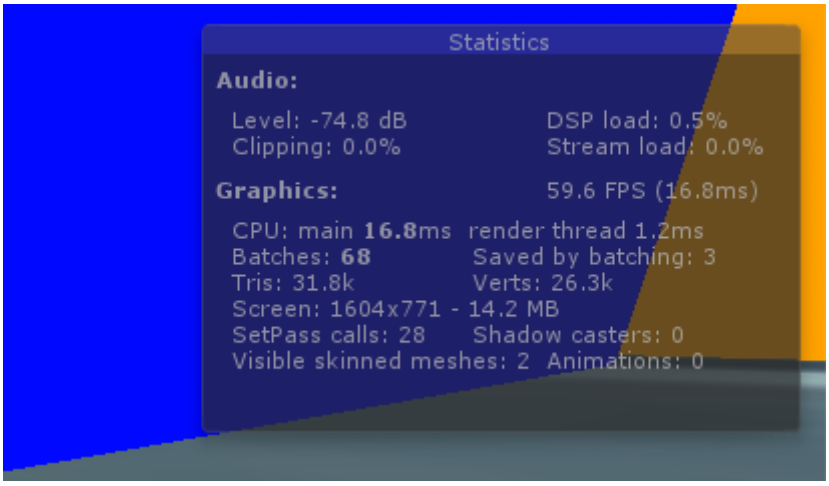
2) 카메라 최적화



3) 정적 배칭

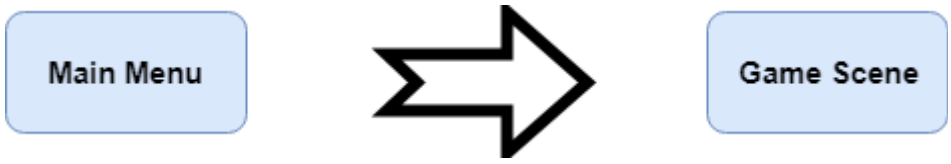


4) 적용 결과물

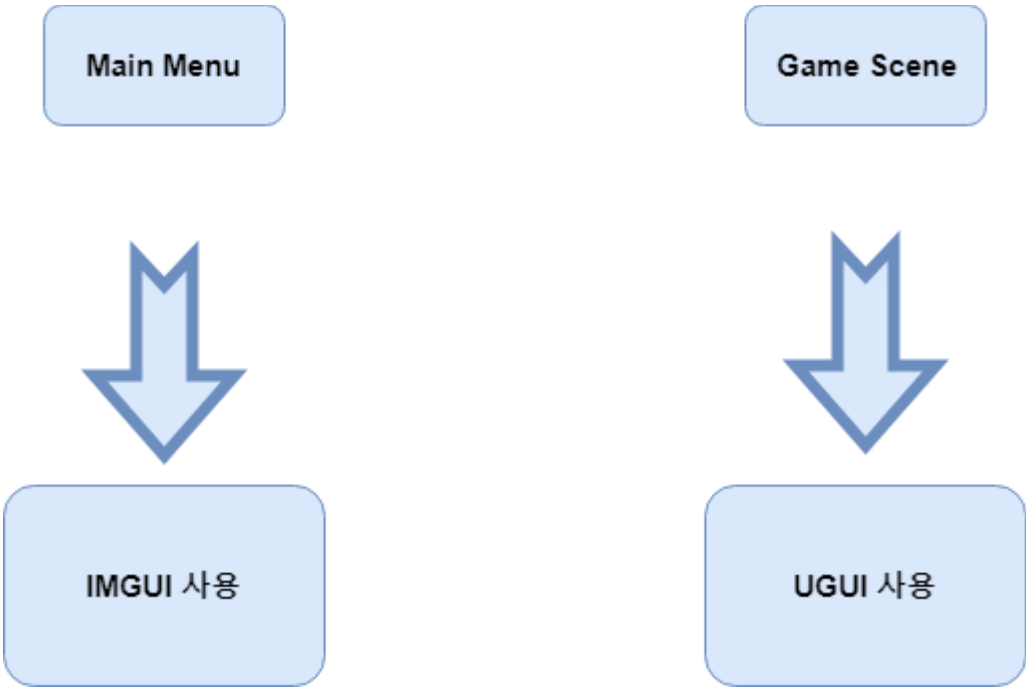

Statistics
Audio:
Level: -74.8 dB
DSP load: 0.5%
Clipping: 0.0%
Stream load: 0.0%
Graphics:
59.6 FPS (16.8ms)
CPU: main 16.8ms render thread 1.2ms
Batches: 68
Saved by batching: 3
Tris: 31.8k
Verts: 26.3k
Screen: 1604x771 - 14.2 MB
SetPass calls: 28
Shadow casters: 0
Visible skinned meshes: 2
Animations: 0

10. 그 외의 정보

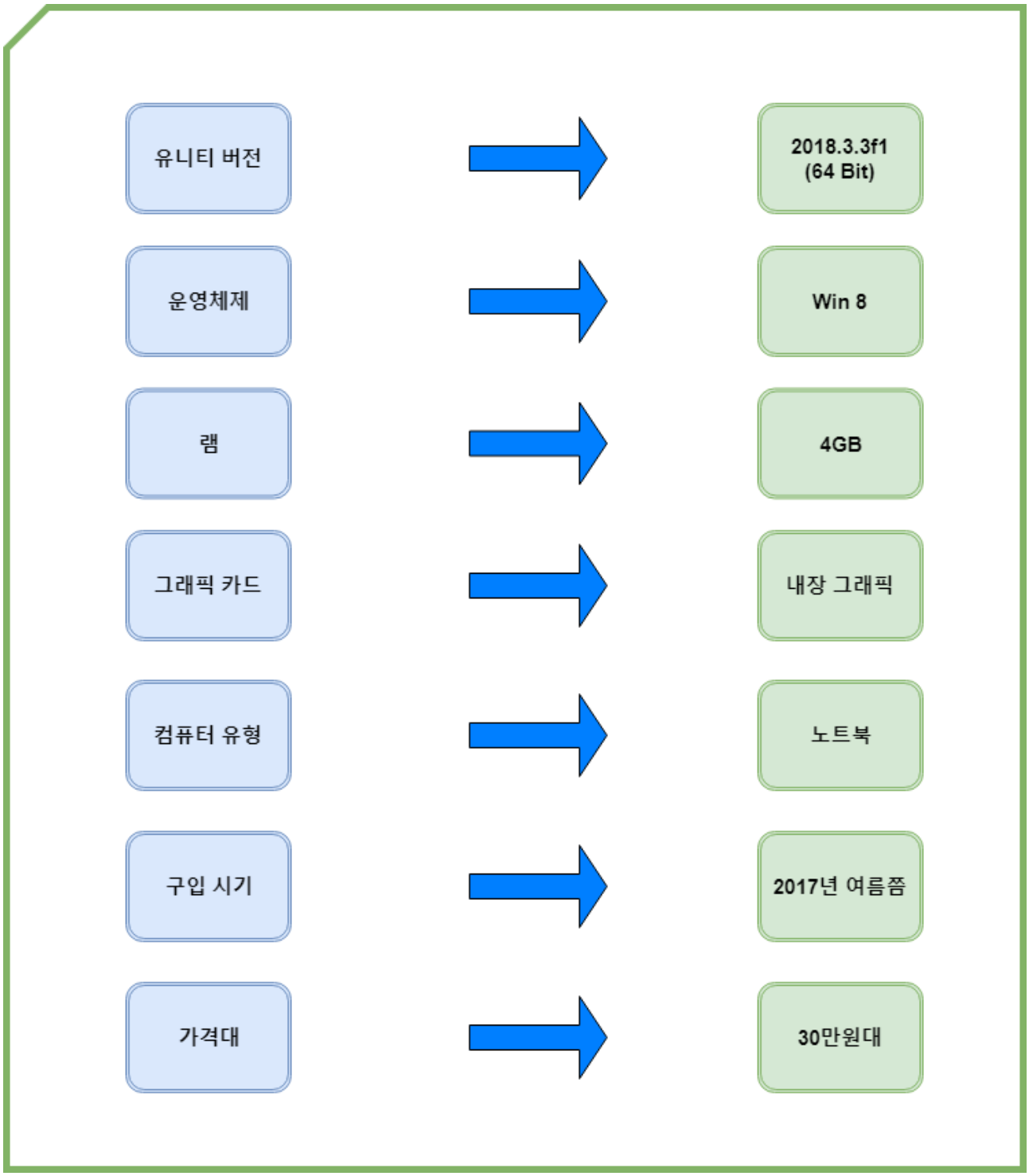
1) 씬 구조도



2) UI



3) 개발 환경


유니티 버전
2018.3.3f1
(64 Bit)
운영체제
Win 8
램
4GB
그래픽 카드
내장 그래픽
컴퓨터 유형
노트북
구입 시기
2017년 여름쯤
가격대
30만원대