ESCAPE NIGHTMARES

게임 기획서

TEAM: NULL DEVELOP
MEMBER: 최영근, 하재헌, 임영균

1_게임정보

게임정보

게임 타이틀 – Escape NightMares
제작 기간 – 약 3주간

참고 게임 – Little NightMares

사용 엔진 – Unity Engine

* 참고 게임

* 사용 엔진

게임 방식 – Single Play

장르 – 퍼즐, 추리, 탈출

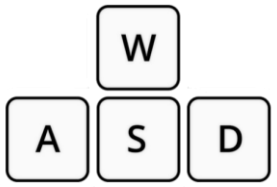
플레이 방식 – 참고 게임 플레이 방식을 참고하여 Quarter View를 통해 보이는 맵 안에서 등장하는 오브젝트들을 상호작용 하여 나아가는 방식

세계관 – 플레이어 캐릭터가 실험실에 갇히면서 시작된다. 그 안에서 탈출 하려고 내부로 이동하게 되는데 내부로 이동하는 플레이어는 그 장소에 비밀을 하나씩 알아가게 되면서 더욱 깊숙한 곳으로 나아가게 된다.…

조작법 – 뒷장에서 설명 참고~

조작법 – PC게임을 기반으로 키보드를 이용한 간단한 조작과 마우스 좌클릭을 이용하여 오브젝트와 상호작용할 수 있도록 구현

플레이어 이동



* 이동(a,w,s,d)

앉기 / 기어가기

* 앉기
* 기어가기

라이트 켜기 / 끄기

* 라이트 켜기
* 라이트 끄기

달리기

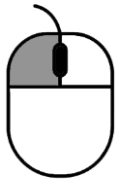
* 달리기

점프

* 점프

조작법 – PC게임을 기반으로 키보드를 이용한 간단한 조작과 마우스 좌클릭을 이용하여 오브젝트와 상호작용할 수 있도록 구현

오브젝트 상호작용



매달리기

* 매달리기

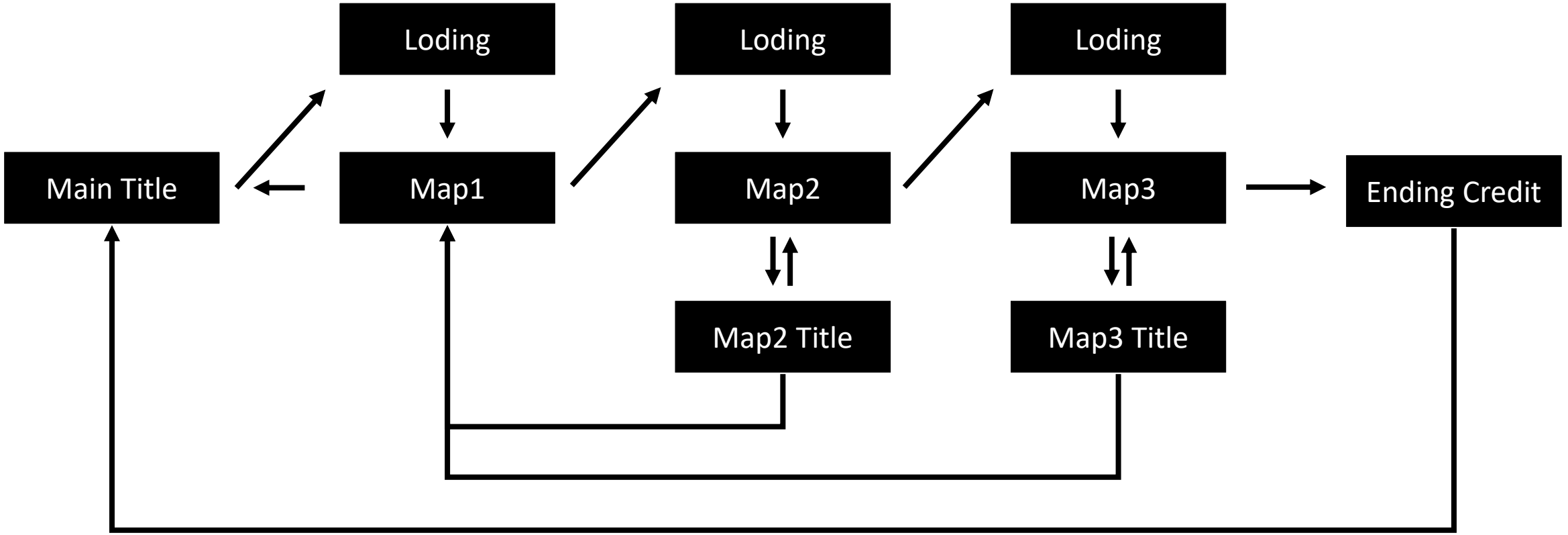
밀기&당기기

* 당기기

* 밀기

2_제작

제작 _장면


Loding
Loding
Loding
Main Title
Map1
Map2
Map3
Ending Credit
Map2 Title
Map3 Title

하재헌 파트_Map1

Scene – Map1 Title Scene / Map1 Play Scene

UI – Map1 Title UI / Map1 Setting UI

Object – Map1 Player / Map1 Enemy_Boss / Map1 Game Space

최영근 파트_Map2

Scene – Map2 Title Scene / Map2 Play Scene

UI – Map2 Title UI / Map2 Setting UI

Object – Map2 Player / Map2 Enemy_Boss / Map2 Game Space

임영균 파트_Map3

Scene – Map3 Title Scene / Map3 Play Scene / Ending Scene

UI – Map3 Title UI / Map3 Setting UI / Loding UI / Ending Credit UI

Object – Map3 Player / Map3 Enemy_Spider, Boss / Map3 Game Space

에셋

- 플레이어 캐릭터의 모델과 애니메이션 등을 믹사모(Mixamo) 사이트를 통하여 제작
- 오브젝트, Enemy 등은 Unity Asset Store를 이용하여서 제작
- Unity Asset Store에서 찾지 못한 모델은 Unity Game Object를 통하여 제작

Part_Map1

Player

- **기본조작** – Axis 값을 이용하여 구현
- **이동** – "Horizontal", "Vertical" 로 기본 상하좌우 조작
- **점프** – RigidBody.AddForce를 사용하고 Impulse로 "Space"입력 시 위로 뛰게 구현
- **앉기/기어가기** – "Left Ctrl"입력 시 캐릭터가 앉는 애니메이션과 그에 맞게 Collider을 줄였고 앉은 후 상하좌우 조작 시 기어가는 애니메이션을 추가함
- **라이터 켜고 끄기** – "F" 입력 시 라이터를 키고 Point Light를 이용하여 주변을 밝힘
- **상호작용** – Ray와 Collision, Trigger를 이용하여 구현
- **사물 잡기** – 상호작용 시에 잡는 애니메이션을 하며 잡을 수 있는 물체가 아니면 잡히지 않음, 잡히는 물체면 FixedJoint를 이용하여 끌려오거나 밀 수 있게 함
- **밀고 당기기** – 사물을 잡은 후 이동하면 사물의 위치와 캐릭터의 위치 차를 계산하여 밀고 당기기를 구현
- **문 열기(버튼형)** – 버튼에 올라가게 되면 Trigger가 작동하여 버튼이 눌리는 애니메이션과 동시에 문이 열리게 된다.
- **사다리 오르기** – 전방으로 Ray를 발생시켜 LADDER레이어와 만나면 사다리 오르는 애니메이션을 하며 상하로만 움직일 수 있음, 꼭대기에 도착하게 되면 도착지로 이동
- **양초 및 사물에 불 붙이기** – Trigger를 이용하여 라이터를 켜져 있거나 가까이에 가서 라이터를 켜면 불이 붙게 구현

Part_Map1

Enemy

- **이동** – 기본적인 움직임은 NavMeshAgent를 이용하여 움직일 수 있는 공간을 한정지음
- **순찰상태** – 주인공이 사정거리 밖에 있을 시에 지정한 4개의 장소를 맴돌며 순찰함
- **추적상태** – 주인공이 사정거리 안으로 들어올 때 주인공을 향하여 뜀
- **공격상태** – 주인공이 공격범위 안으로 들어올 때 이동을 멈추고 제자리에서 공격을 시도함
- **추적 후 순찰상태** – 추적 후에 주인공이 사정거리 밖으로 나가게 되면 지정했던 4개의 장소 중에 가장 가까운 곳으로 먼저 이동후에 순찰상태로 바뀜
- **죽음** – 지정된 3개의 오브젝트에 불을 지르게 되면 보스가 죽게 되고 클리어 문이 열리게 된다.

Map

- **Floor** – Floor 태그와 FLOOR 레이어를 이용하여 점프가 가능한 구간과 안되는 구간을 나누었다
- **Drop Floor** – 떨어지는 발판을 밟을 시에는 RigidBody에 2초 후 isKinematic을 false로 만들어 발판이 떨어지게 함
- **라이트** – LightMapping과 LightProb을 사용함

Part_Map1

```
void Run()
{
    if (isRun)
        moveSpeed = 3f;
    if (!isRun)
        moveSpeed = 2f;
}

public void MoveRot(float x)
{
    if (h == 0 && v == 0)
        return;
    moveVec = new Vector3(h * x, 0, v);
    Quaternion rot = Quaternion.LookRotation(moveVec);
    tr.rotation = Quaternion.Slerp(tr.rotation, rot, rotSpeed * Time.d
}

void Jumped()
{
    Vector3 moveVec = new Vector3(h, 0, v).normalized;
    Vector3 dir = (moveVec * 0.4f) + Vector3.up;
    if (jBD && !isjump)
    {
        if (pRay.isFloor)
        {
            if (isRun)
            {
                isjump = true;
                ani.SetTrigger("isJump");
                ani.SetBool("isGround", false);
                rbody.AddForce(dir * jumpPower, ForceMode.Impulse);
            }
            if (!isRun)
            {
                isjump = true;
```

struct System.Single

```
if (isCrawl)
{
    ani.SetBool("isCrawling", true);
    moveSpeed = isMove ? 2f : 0f;
    col.height = isMove ? 1.6f : 1.2f;
    col.direction = isMove ? 2 : 1;
    if (isMove)
    {
        col.center = new Vector3(0f, 0.5f, 0f);
    }
    else if (!isMove)
    {
        col.center = new Vector3(0f, 0.6f, 0f);
    }
}
else if (!isCrawl)
{
    ani.SetBool("isCrawling", false);
    moveSpeed = isMove ? 2f : 0f;
    col.height = 2f;
    col.direction = 1;
    if (isMove)
    {
        col.center = new Vector3(0f, 1f, 0f);

        if (isjump)
            return;
        if (!isjump)
            Run();

    }
    else if (!isMove)
    {
```

```
void LightOnOff()
{
    if (Input.GetKeyDown(KeyCode.F))
    {
        isLight = !isLight;
        StartCoroutine(Lighting());
        LightOn();
        LightOff();
    }

}

void LightOn()
{
    if (isLight)
    {
        Lighter.gameObject.SetActive(true);
        ani.SetBool("isLight", true);
    }
}
void LightOff()
{
    if (!isLight)
    {
        Lighter.gameObject.SetActive(false);
        ani.SetBool("isLight", false);
    }
}


IEnumerator Lighting()
{
    while (isLight)
    {
        Plight.intensity = Random.Range(2f, 5f);
        yield return new WaitForSeconds(0.1f);
```

Part_Map1

```
void Ladder()
{
    Vector3 ladPoint = new Vector3(tr.position.x, tr.position.y + 0.5
    ladRay = new Ray(ladPoint, tr.forward);
    isLadder = Physics.Raycast(ladRay, out hit, 1f, ladderLayer);

    if (isLadder && !pCtrl.isCrawl)
    {
        if (Input.GetMouseButton(0))
        {
            ani.SetBool("isLadder", true);
            pCtrl.enabled = false;

            rbody.useGravity = false;
            rbody.isKinematic = true;
            LadderMove();
        }
    }
    else if (!isLadder)
        return;
}
void LadderMove()
{
    v = Input.GetAxis("Vertical");
    float moveSpeed = 3f;
    ani.SetFloat("Laddering", Mathf.Clamp(v * 10, -1, 1));

    Vector3 moveVec = new Vector3(0f, v, 0f);
    moveVec = moveVec.normalized * moveSpeed * Time.deltaTime;
    rbody.MovePosition(tr.position + moveVec);

    OutOfLadder();

}
```

```
            BurnCount();
            fire.gameObject.SetActive(true);
            Bcol.enabled = false;
            isBurn = true;
        }
    }
}

private void OnTriggerEnter(Collider col)
{
    if (col.CompareTag("Player"))
    {
        isTri = true;
        if (pLighter.isLight && !isBurn)
        {
            BurnCount();
            fire.gameObject.SetActive(true);
            Bcol.enabled = false;
            isBurn = true;
        }

    }
}
private void OnTriggerExit(Collider col)
{
    if (col.CompareTag("Player"))
    {
        isTri = false;
    }
}

void BurnCount()
{
    bDoor.burned += 1;

}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class H_ButtonDown1 : MonoBehaviour
{
    [SerializeField]
    BoxCollider bCol;
    [SerializeField]
    Animator doorAni;
    [SerializeField]
    Animator buttonAni;

    private void Start()
    {
        bCol = GetComponent<BoxCollider>();
        buttonAni = GetComponent<Animator>();
        doorAni = GameObject.Find("ProfileDoor").GetCom
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.CompareTag("Player"))
        {
            buttonAni.SetBool("isPress", true);
            Invoke("DoorOpen", 1f);
        }
    }
    void DoorOpen()
    {
        doorAni.SetBool("isOpen", true);
    }
}
```

Part_Map2

Player

- **FSM**(Finite state machine) 으로 구현
- **기본조작** – Axis 값을 이용하여 구현
- **이동** – 기본 Input값인 Horizontal과 Vertical에 Run을 왼쪽 시프트키로 추가하였다. 각 Axis값으로 속도를 측정하여 애니메이션 블랜드 트리로 구현하였다.
- **점프** – RigidBody.AddForce를 사용했다. Ground 태그에 닿으면 점프가 끝나도록 구현하였다.
- **앉기/기어가기** – Ctrl 값을 사용하여 캐릭터의 상태를 변화시켰다. 앉기 상태 시 위쪽 Ray에 물체가 있다면 다시 일어설 수 없도록 하였다. 앉은 상태에서 움직임이 있다면 기어가는 것을 애니메이션 블랜드 트리로 구현하였다.
- **상호작용** – Ray를 사용하여 오브젝트별 상호작용을 구현.
- **밀기** – 전방으로 Ray를 발사하여 움직이는 오브젝트가 있으면 애니메이션 블랜드 처리를 하였다.
- **당기기** – Fixed joint를 사용했다. 밀기와 같은 방법으로 충돌 중 상호작용시 오브젝트에 Fixed joint를 AddComponent하고 캐릭터의 RigidBody를 추가시켰다.
- **문 열기** – 위쪽으로 Ray를 발사하여 손잡이가 있다면 애니메이션을 재생시켰다. 문에는 SendMessage를 활용해 열리는 애니메이션을 재생시켰다.
- **올라가기** – 전방 Ray를 활용하여 클릭 시 애니메이션이 재생되고 Y축으로 이동 값을 변경하여 움직일 수 있도록 하였다.
- **공차기** – 충돌을 캐릭터와 적으로 구분하였고 캐릭터와 충돌 중 클릭 시 캐릭터와의 방향을 구하여 반대쪽으로 날아가도록 하였다. 적과 충돌 시 SendMessage를 활용하여 적이 기절하도록 구현하였다.
- **양초** – PlayerPrefs을 사용하여 상호작용시 현재 위치가 저장된다.

Part_Map2

Enemy

- **FSM**(Finite state machine) 으로 구현
- **이동** – NavMeshAgent를 사용하여 캐릭터와의 거리에 따라 상태를 전환하도록 하였다.
- **패트롤** – 주인공이 거리 밖에 있을 경우 Point지점을 따라 이동하도록 했다.
- **추적** – 주인공이 추적거리 안에 들어올 경우 장애물을 피해 캐릭터 방향으로 이동하도록 했다.
- **공격** – 주인공이 공격거리 안에 들어올 경우 주인공을 집어 들며 적의 손에 잡히도록 했다.
- **죽음** – Player가 공을 찬 상태일 때 충돌하면 HP가 소모되고 HP가 0이 되면 기절한다.

Map

- **Floor** – 캐릭터의 이동경로에는 Ground 태그를 이용하였다.
- **Wall/Object** – Wall이나 Object에는 Physics Material을 사용해 자연스러운 충돌을 구현.
- **Light** – LightMapping과 LightProb를 사용하였다..
- **Background Bug** – 벌레들이 이동하는 것을 Object Pooling을 이용하여 생성하였다.
- **공차기** – 충돌을 캐릭터와 적으로 구분하였고 캐릭터와 충돌 중 클릭 시 캐릭터와의 방향을 구하여 반대쪽으로 날아가도록 하였다. 적과 충돌 시 SendMessage를 활용하여 적이 기절하도록 구현하였다.
- **양초** – PlayerPrefs을 사용하여 상호작용시 현재 위치가 저장된다.

Part_Map2

```
public enum STATE { Move, Jump, Crawl, Push, Pull, Door, Ladder, Save, Die }
public STATE state;
```

```
private void FixedUpdate()
{
    if (player.state != C_Player.STATE.Die || !isInteraction)
    {
        switch (player.state)
        {
            case C_Player.STATE.Move:
                Move();
                Jump();
                ChangeCrawl();
                break;
            case C_Player.STATE.Jump:
                MovePos();
                break;
            case C_Player.STATE.Crawl:
                CrawlMove();
                ChangeCrawl();
                break;
        }
        IsBool_AnimatorKey();
    }
}
```

```
#region ----- 전방 레이 -----
void Forward_Interaction()
{
    if (hit.targetForward != null)
    {
        switch (player.state)
        {
            case C_Player.STATE.Move:
                ToPush();
                ToPull();
                Ladder();
                SaveCandle();
                break;
            case C_Player.STATE.Push:
                ToPull();
                PushMove();
                break;
            case C_Player.STATE.Pull:
                ToMove();
                PullMove();
                break;
            case C_Player.STATE.Ladder:
                LadderMove();
                ani.SetFloat("LadderSpeed", v);
                break;
        }
    }
    else if (hit.targetForward == null)
    {
        switch (player.state)
        {
            case C_Player.STATE.Move:
                break;
            case C_Player.STATE.Jump:
                break;
            case C_Player.STATE.Crawl:
                break;
            case C_Player.STATE.Push:
                player.state = C_Player.STATE.Move;
                movement.isInteraction = false;
                break;
            case C_Player.STATE.Pull:
                player.state = C_Player.STATE.Move;
                movement.isInteraction = false;
                break;
            case C_Player.STATE.Door:
                break;
            case C_Player.STATE.Ladder: // 꼭대기에 닿으면 실행
                Vector3 ladderTop = new Vector3(0f, 1f, 0.5f);
                tr.Translate(ladderTop.normalized);
                player.state = C_Player.STATE.Move;
                movement.isInteraction = false;
                rbody.useGravity = true;
                rbody.isKinematic = false;
                rbody.collisionDetectionMode = CollisionDetectionMode.Continuous;
                break;
            case C_Player.STATE.Die:
                break;
        }
    }
```

Part_Map2

```
public GameObject bugPrefab;
public List<GameObject> bugPool = new List<GameObject>();

int maxCount = 20;
public bool isGameover = false;

void Start()
{
    spawnPoint = GameObject.Find("BugWayPoint").transform;

    for (int i = 0; i < maxCount; i++)
    {
        GameObject bug = Instantiate(bugPrefab, transform);
        bug.name = "Bug" + i.ToString();
        bug.SetActive(false);
        bugPool.Add(bug);
    }

    StartCoroutine(Spawn());
}

IEnumerator Spawn()
{
    while (!isGameover)
    {
        yield return new WaitForSeconds(1f);

        foreach (GameObject _bug in bugPool)
        {
            if (!_bug.activeSelf)
            {
                _bug.transform.position = spawnPoint.position;
                _bug.SetActive(true);
                break;
            }
        }
    }
}
```

```
#region ----- 이동관련 함수 -----

public void InputMove(float Speed, float rotSpeed, Vector3 vec)
{
    h = Input.GetAxis("Horizontal");
    v = Input.GetAxis("Vertical");
    baseSpeed = Mathf.Clamp(Mathf.Abs(h) + Mathf.Abs(v), 0f, 1f);
    addSpeed = Input.GetAxis("Run");

    if (baseSpeed > 0)
        moveSpeed = Speed;
    else if (baseSpeed == 0 && moveSpeed > 0)
        moveSpeed -= 0.1f;

    MovePos();
    MoveRot(vec, rotSpeed);
}

public void MovePos() // 점프시 이동값 고정하기 위해 별도 함수사용
{
    moveVec = new Vector3(h, 0, v);
    moveVec = moveVec.normalized * moveSpeed * Time.deltaTime;
    rbody.MovePosition(tr.position + moveVec);
}

public void MoveRot(Vector3 moveVec, float rotSpeed) // 당기기시 반대회전을 위해 별도 함수사용
{
    if (moveVec == Vector3.zero) return;
    Quaternion rot = Quaternion.LookRotation(moveVec);
    rbody.rotation = Quaternion.Slerp(rbody.rotation, rot, rotSpeed * Time.deltaTime);
}
#endregion
```

Part_Map3

Player

- **이동** – Axis에 Horizontal과 Vertical 값을 이용하여 float 값으로 속도 값을 설정하여서 이동하도록 구현.
- **달리기** – Shift키 값을 주어 Shift를 누를 시 float 값을 변경하여 속도를 높이도록 구현.
- **점프** – Space키 값을 주고 Ray를 바닥으로 쏴서 바닥에 Ray가 닿아 있을 경우에만 Space를 누르면 RigidBody.AddForce를 이용하여서 위로 올라가도록 구현
- **앉기** – Ctrl키 값을 주어 Ctrl을 누를 시 float 값을 변경하여 속도를 낮추고 collider y값을 낮추어 천장이 낮은 길을 이동하도록 구현.
- **기어가기** – 앉은 모션에서 이동 값이 올라갈 경우 기어서 이동하도록 구현.
- **밀기, 당기기** – Ray를 Player앞으로 쏘아서 Object가 닿을 시 닿은 Mouse 좌 클릭 Object에 FixedJoint를 이용하여 Player가 Object를 밀고 당길 수 있도록 구연하였고 LocalPosition값을 계산하여 Mathf.Clamp값으로 –와 +를 계산 후 블랜드 트리를 통해 앞으로 이동시 미는 모션과 뒤로 당길 시 당기는 모션이 나오도록 구현.
- **문 열기** – 문 손잡이에 Collider를 넣어서 Player와 OnTriggerEnter로 충돌 시 상호작용하여 문 손잡이에 매달리며 문이 열리도록 구현
- **사다리 타기** – Player에서 Ray를 쏴서 사다리에 닿을 때 상호작용 시 올라가는 모션과 함께 y축으로 이동하도록 구현
- **라이터 켜기 끄기** – F키 값을 주어서 켜고 끄기를 반복할 수 있도록 구현

Part_Map3

Enemy_Spider

- **이동** – NavMeshAgent를 이용하여 이동하도록 구현.
- **스폰** – Object Pooling을 이용하여 저장된 위치에서 지정된 스폰 값만큼 Enemy가 계속 스폰되게 구현
- **패트롤** – Player가 거리 내에 존재하지 않을 시 미리 지정해 놓은 Points Position값들 중 하나로 이동하게 하였고 지정된 Position에 도착 시 다음 포인트로 Random.Range를 사용하여 다음 Points로 랜덤하게 이동.
- **추적** – Player가 지정해 놓은 Distance안으로 들어오면 NavMeshAgent가 플레이어를 추적하도록 구현.
- **공격** – Player가 지정해 놓은 Distance안으로 들어오면 NavMeshAgent를 정지하고 플레이어를 공격하도록 하였고 공격 범위 내에 BoxCollider를 설치하여 공격 모션 시 작동하여서 Player가 Trigger할 시 Hit하게 구현.
- **죽음** – Enemy 위로 Ray를 설치하여 Player가 Ray에 닿을 시 Hp를 줄여 Hp가 0이 될 시 죽도록 구현.

Enemy_Boss

- **이동** – NavMeshAgent를 이용하여 이동하도록 구현.
- **잠들기** – BossEnemy가 일정 행동 이후 지정된 Point로 돌아가서 잠들게 구현.
- **데미지** – BossEnemy가 잠들기를 실행하게 되면 꼬리 쪽에 설치된 BoxCollider가 작동하게 되면서 Player와 Trigger할 시 Hp를 1씩 줄이도록 구현.
- **스킬** – BossEnemy가 데미지를 입게 되면 Enemy_Spider을 소환되게 하고 소환된 Enemy는 Enemy_Spider와 같이 작동
- **추적** – 소환된 Enemy를 모두 잡을 시 BossEnemy가 Player을 지정된 시간동안 추적하도록 구현하였고 지정 시간이 지나면 잠들기로 들어가도록 구현.
- **공격** – 추적 중 Player가 지정해 놓은 Distance안으로 들어오면 NavMeshAgent를 정지하고 플레이어를 공격하도록 하였고 공격 범위 내에 BoxCollider를 설치하여 공격 모션 시 작동하여서 Player가 Trigger할 시 Hit하게 구현.
- **죽음** – BossEnemy가 데미지를 3번 입으면 죽도록 구현.
- **즉사기** – BossEnemy가 잠들기나 스킬이 실행 중에 BossEnemy 앞에 BoxCollider를 실행하여 Player가 Trigger될 때 Player를 잡아서 즉사 시키도록 구현.

Part_Map3

Stage / Map

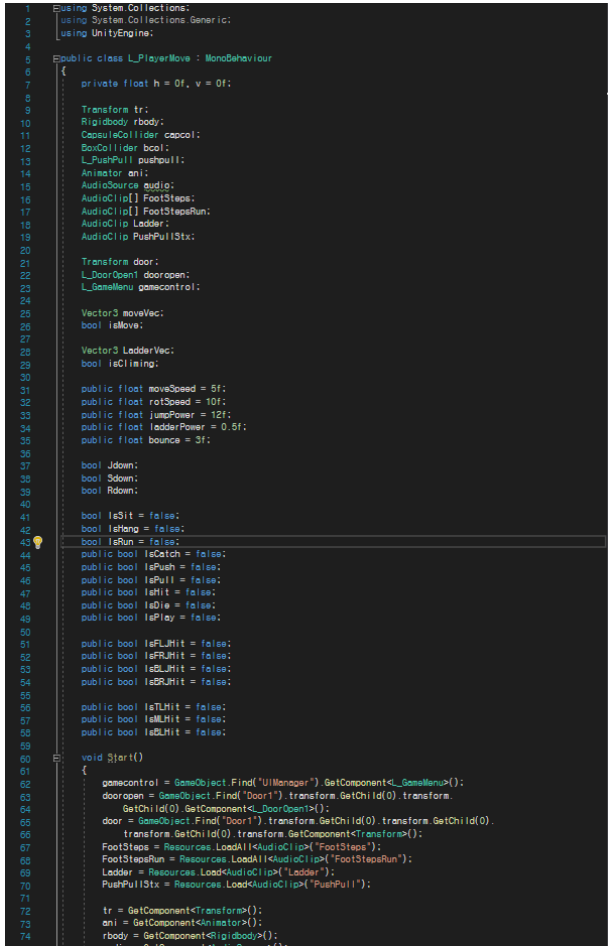
- **맵 라이트** – 맵 전체에 깔려 있는 Light는 LightMapping과 LightProb를 이용하여 구현.
- **세이브** – 맵 내에 세이브를 할 수 있는 Object를 설치하여 플레이어가 라이터를 근처에 있을 때 라이터가 켜지면 불이 켜지면서 PlayerPrefs으로 Player위치 값과 회전 값이 저장되도록 구현.
- **시네마 카메라** – 플레이어가 이동함에 따라서 Cinemachine VirtualCamera를 이용해 View를 전환하여 역동적인 View를 생성.
- **레버** – 레버에 Collider를 설치하여 Player가 Trigger하였을 때 상호작용을 하면 레버가 작동하여 문이 열리도록 구현.
- **도어1** – 레버가 작동 시 애니메이션이 작동하여 열리는 문 구현.
- **도어2** – 레버가 작동 시 열렸다가 자동으로 닫히는 문 구현.
- **도어3** – Boss방을 입장 시 입장한 공간을 닫아버리는 문 구현.
- **도어4** – Boss가 죽었을 때 열리는 문 구현.
- **텔레포트** – Player가 낭떠러지로 떨어질 시 지정된 위치로 되돌아가는 구간 구현 / 보스방에서 보스가 추적 중 방으로 들어가면 다른 방으로 이동하는 텔레포트 구현.
- **엔딩** – 세이브 Object와 마찬가지로 근처에 있을 때 라이터가 켜져 있으면 불이 붙으면서 Scene Ending Credit으로 Scene이동 시켜주는 상호작용 구현.

Part_Map3

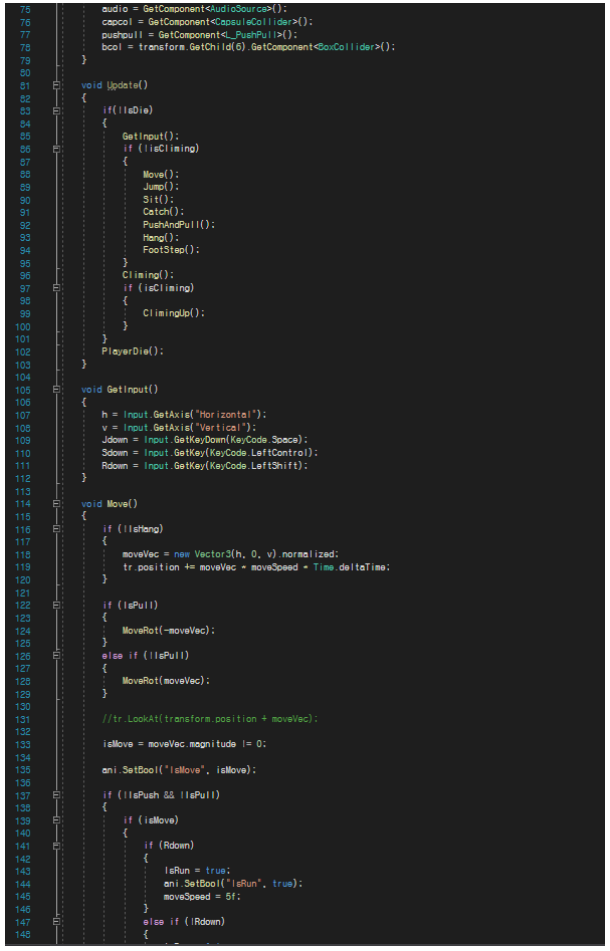
UI System / Scene

- **Title UI** – Title UI에는 New Start버튼, Load버튼, Quit버튼을 세팅.
- **New Start Button** – New Start버튼은 SceneManager.LoadScene로 씬 이동 후 PlayerPrefs.DeleteAll을 사용하여 저장되어 있는 값을 삭제하고 처음부터 시작하도록 구현
- **Load Button** – Load버튼은 SceneManager.LoadScene로 씬 이동 후 PlayerPrefs값에 저장된 위치에서 시작되도록 구현
- **Quit Button** – Quit버튼은 Application.Quit를 이용해 어플리케이션을 종료하도록 구현.
- **Setting UI** – Escape키 값을 설정하여 Escape키를 입력 시 Setting UI가 등장하면서 Time.timescale 값을 0으로 주어 게임을 정지시키고 다시 한 번 Escape값을 입력 시 Setting UI가 꺼지면서 Time.timescale 값을 1로 주어 정상시간으로 되돌리도록 구현.
- **Game Sound Button** – Player에게 AudioListener을 넣어 놓고 AudioListener Volume값을 설정.
- **Music Sound Button** – MainCamera안 AudioSource에 배경음악을 넣어 놓고 Button값에 MainCamera AudioSource Volume값을 설정.
- **Light Intensity Slider** – Directional Light에 있는 Intensity값을 설정.
- **Restart Button** – Restart버튼은 SceneManager.LoadScene로 씬 이동 후 PlayerPrefs.DeleteAll을 사용하여 저장되어 있는 값을 삭제하고 처음부터 시작하도록 구현.
- **SavePoint Button** – SavePoint버튼은 SceneManager.LoadScene로 씬 이동 후 PlayerPrefs값에 저장된 위치에서 시작되도록 구현.
- **Title Button** – Title버튼은 SceneManager.LoadScene로 Title 씬으로 이동 하도록 구현.
- **Back Button** – Back버튼은 누르면 Setting UI가 꺼지면서 Time.timescale 값을 1로 주어 정상시간으로 되돌리도록 구현.

제작 _ 스크립트



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class L_PlayerMove : MonoBehaviour
{
    private float h = 0f, v = 0f;

    Transform tr;
    Rigidbody rbody;
    CapsuleCollider capcol;
    BoxCollider bcol;
    L_PushPull pushpull;
    Animator ani;
    AudioSource audio;
    AudioClip[] FootSteps;
    AudioClip[] FootStepsRun;
    AudioClip Ladder;
    AudioClip PushPullSfx;

    Transform door;
    L_DoorOpen1 dooropen;
    L_GameMenu gamecontrol;

    Vector3 moveVec;
    bool isMove;

    Vector3 LadderVec;
    bool isCliming;

    public float moveSpeed = 5f;
    public float rotSpeed = 10f;
    public float jumpPower = 12f;
    public float ladderPower = 0.5f;
    public float bounce = 3f;

    bool Jdown;
    bool Sdown;
    bool Rdown;

    bool IsSit = false;
    bool IsHang = false;
    bool IsRun = false;
    public bool IsCatch = false;
    public bool IsPush = false;
    public bool IsPull = false;
    public bool IsHit = false;
    public bool IsDie = false;
    public bool IsPlay = false;

    public bool IsFLJHit = false;
    public bool IsFRJHit = false;
    public bool IsBLJHit = false;
    public bool IsBRJHit = false;

    public bool IsTLHit = false;
    public bool IsMLHit = false;
    public bool IsBLHit = false;

    void Start()
    {
        gamecontrol = GameObject.Find("UIManager").GetComponent<L_GameMenu>();
        dooropen = GameObject.Find("Door1").transform.GetChild(0).transform.
            GetChild(0).GetComponent<L_DoorOpen1>();
        door = GameObject.Find("Door1").transform.GetChild(0).transform.GetChild(0).
            transform.GetChild(0).transform.GetComponent<Transform>();
        FootSteps = Resources.LoadAll<AudioClip>("FootSteps");
        FootStepsRun = Resources.LoadAll<AudioClip>("FootStepsRun");
        Ladder = Resources.Load<AudioClip>("Ladder");
        PushPullSfx = Resources.Load<AudioClip>("PushPull");

        tr = GetComponent<Transform>();
        ani = GetComponent<Animator>();
        rbody = GetComponent<Rigidbody>();
```



```
        audio = GetComponent<AudioSource>();
        capcol = GetComponent<CapsuleCollider>();
        pushpull = GetComponent<L_PushPull>();
        bcol = transform.GetChild(6).GetComponent<BoxCollider>();
    }

    void Update()
    {
        if(!IsDie)
        {
            GetInput();
            if (!isCliming)
            {
                Move();
                Jump();
                Sit();
                Catch();
                PushAndPull();
                Hang();
                FootStep();
            }
            Climing();
            if (isCliming)
            {
                ClimingUp();
            }
        }
        PlayerDie();
    }

    void GetInput()
    {
        h = Input.GetAxis("Horizontal");
        v = Input.GetAxis("Vertical");
        Jdown = Input.GetKeyDown(KeyCode.Space);
        Sdown = Input.GetKey(KeyCode.LeftControl);
        Rdown = Input.GetKey(KeyCode.LeftShift);
    }

    void Move()
    {
        if (!IsHang)
        {
            moveVec = new Vector3(h, 0, v).normalized;
            tr.position += moveVec * moveSpeed * Time.deltaTime;
        }

        if (IsPull)
        {
            MoveRot(-moveVec);
        }
        else if (!IsPull)
        {
            MoveRot(moveVec);
        }

        //tr.LookAt(transform.position + moveVec);

        isMove = moveVec.magnitude != 0;

        ani.SetBool("IsMove", isMove);

        if (!IsPush && !IsPull)
        {
            if (isMove)
            {
                if (Rdown)
                {
                    IsRun = true;
                    ani.SetBool("IsRun", true);
                    moveSpeed = 5f;
                }
                else if (!Rdown)
                {
```

```
                    IsRun = false;
                    ani.SetBool("IsRun", false);
                    moveSpeed = 3.5f;
                }
            }
            else if (!isMove)
            {
                if (Rdown)
                {
                    IsRun = false;
                    ani.SetBool("IsRun", false);
                    ani.SetBool("IsMove", false);
                }
            }
            if (IsSit)
            {
                if (isMove)
                {
                    ani.SetBool("IsCrawl", true);
                    moveSpeed = 2f;
                }
                else if (!isMove)
                {
                    ani.SetBool("IsSit", true);
                    ani.SetBool("IsCrawl", false);
                }

            }
            else if (!IsSit)
            {
                if (isMove)
                {
                    ani.SetBool("IsCrawl", false);
                }
            }
        }
    }

    void FootStep()
    {
        if (!IsFLJHit | !IsFRJHit | !IsBLJHit | !IsBRJHit)
        {
            if (isMove && !IsRun && !IsSit && !isCliming)
            {
                if (!audio.isPlaying)
                    audio.PlayOneShot(FootSteps[Random.Range(0, 1)], 1.0f);
            }
            if (IsRun)
            {
                if (!audio.isPlaying)
                    audio.PlayOneShot(FootStepsRun[Random.Range(0, 4)], 1.0f);
            }
        }
    }

    void MoveRot(Vector3 moveVec)
    {
        if (h == 0 && v == 0)
            return;
        Quaternion rot = Quaternion.LookRotation(moveVec);
        tr.rotation = Quaternion.Slerp(tr.rotation, rot, rotSpeed * Time.deltaTime).normalized;
    }


    void Jump()
    {
        if (!IsPull && !IsPush)
        {
            if (!IsFLJHit | !IsFRJHit | !IsBLJHit | !IsBRJHit)
            {
                if (Jdown)
                {
                    rbody.AddForce(Vector3.up * jumpPower, ForceMode.Impulse);
                    ani.SetBool("IsJump", false);
```

```
                    ani.SetTrigger("DoJump");
                }
            }
        }
    }

    private void OnCollisionEnter(Collision collision)
    {
        if (collision.gameObject.tag == "Floor" | collision.gameObject.tag == "Object" |
            collision.gameObject.tag == "Landing" | collision.gameObject.tag == "Boss")
        {

            ani.SetBool("IsJump", true);
        }


        if (collision.gameObject.tag == "Enemy")
        {
            if (!IsFLJHit | !IsFRJHit | !IsBLJHit | !IsBRJHit)
            {
                ani.SetTrigger("DoJump");
                rbody.AddForce(Vector3.up * bounce, ForceMode.Impulse);
            }
        }
    }

    void Sit()
    {
        if (Sdown)
        {
            IsSit = true;
            ani.SetBool("IsSit", true);
            capcol.height = 0.9f;
            capcol.radius = 0.4f;
            capcol.center = new Vector3(0, 0.45f, 0);
        }
        if (!Sdown)
        {
            IsSit = false;
            ani.SetBool("IsSit", false);
            capcol.height = 1.7f;
            capcol.radius = 0.4f;
            capcol.center = new Vector3(0, 0.85f, 0);
        }
    }

    void Catch()
    {
        if (!Rdown)
        {
            if (!IsFLJHit | !IsFRJHit | !IsBLJHit | !IsBRJHit)
            {
                if (!IsHit)
                {
                    if (Input.GetMouseButtonDown(0))
                    {
                        IsCatch = true;
                        rotSpeed = 0.5f;
                    }
                    else if (Input.GetMouseButtonUp(0))
                    {
                        IsCatch = false;
                        rotSpeed = 10f;
                    }
                }
                else if (!IsHit)
                {
                    IsCatch = false;
                    rotSpeed = 10f;
                }

            }
        }
    }
```

```
    void PushAndPull()
    {
        if (IsPush && !IsPull)
        {
            ani.SetFloat("Push", pushpull.Move);
            ani.SetBool("IsPush", true);
            if(!audio.isPlaying)
                audio.PlayOneShot(PushPullStx, 1.0f);
        }
        else if (!IsPush && IsPull)
        {
            ani.SetFloat("Push", pushpull.Move);
            ani.SetBool("IsPush", true);
            if (!audio.isPlaying)
                audio.PlayOneShot(PushPullStx, 1.0f);
        }
        else
        {
            if (isMove)
            {
                ani.SetBool("IsPush", false);
                ani.SetBool("IsMove", true);
            }
            else if (!isMove)
            {
                ani.SetBool("IsPush", false);
            }
        }
    }

    void LadderMove()
    {
        LadderVec = new Vector3(0, v, 0).normalized;
        tr.position += LadderVec * 2f * Time.deltaTime;
    }

    void Climing()
    {
        if (!dooropen.isTri)
        {
            if (!IsTLHit | !IsMLHit | !IsBLHit)
            {
                if (Input.GetMouseButtonDown(0))
                {
                    rbody.useGravity = false;
                    isCliming = true;
                }
                else if (Input.GetMouseButtonUp(0))
                {
                    rbody.useGravity = true;
                    isCliming = false;
                    ani.SetBool("IsCliming", false);
                }

            }
            else if (!IsTLHit && !IsMLHit && !IsBLHit)
            {
                rbody.useGravity = true;
                isCliming = false;
                ani.SetBool("IsCliming", false);
                ani.SetBool("IsLanding", false);
            }
        }
    }

    void ClimingUp()
    {
        if (isCliming)
        {
            LadderMove();
            if (!audio.isPlaying)
                audio.PlayOneShot(Ladder, 1.0f);
            ani.SetBool("IsCliming", true);
        }
```

```
            LadderMove();
            if (!audio.isPlaying)
                audio.PlayOneShot(Ladder, 1.0f);
            ani.SetBool("IsCliming", true);
        }
    }

    void Hang()
    {
        if(dooropen.isTri)
        {
            if (Input.GetMouseButtonDown(0))
            {
                IsHang = true;
                tr.parent = door.transform;
                tr.position = new Vector3(25f, 2.3f, -6.1f);
                ani.SetBool("IsHang",true);
                StartCoroutine(Hanging());
            }
            else if(Input.GetMouseButtonUp(0))
            {
                IsHang = false;
                rbody.useGravity = true;
                transform.parent = null;
                ani.SetBool("IsHang", false);
            }
        }
        if (!dooropen.isTri)
        {
            IsHang = false;
            rbody.useGravity = true;
            ani.SetBool("IsHang", false);
            transform.parent = null;
        }
    }

    IEnumerator Hanging()
    {
        ani.SetBool("IsHang", true);
        rbody.useGravity = false;
        rbody.velocity = Vector3.zero;
        rbody.angularVelocity = Vector3.zero;
        dooropen.DoorOpen();

        yield return new WaitForSeconds(1.0f);

        ani.SetBool("IsHang", false);
        IsHang = false;
        rbody.useGravity = true;
        transform.parent = null;
    }

    void PlayerDie()
    {
        if(!isDie)
        {
            rbody.isKinematic = true;
            capcol.enabled = false;
            bcol.enabled = false;
            rbody.constraints = RigidbodyConstraints.FreezeAll;
            ani.SetBool("IsDie", true);
            StartCoroutine(SceneLoder());
        }
    }

    IEnumerator SceneLoder()
    {
        yield return new WaitForSeconds(3f);
        gamecontrol.LoadGame();
    }
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class L_EnemyCtrl : MonoBehaviour
{
    public enum Type { A,B};
    public Type enemyType;

    AudioSource audio;
    AudioClip EDie;
    Transform tr;
    Rigidbody rbody;
    BoxCollider AtkCol;
    NavMeshAgent navi;
    BoxCollider col;
    [HideInInspector]
    public Animator ani;
    L_EnemyPatrol Epatrol;
    L_EnemyDamage Edamage;
    L_EnemyManager Emanager;
    L_DoorClose doorclose;
    L_BossEnemyPool enemypool;
    Transform target;

    float h, v;
    Vector3 lookVec;

    [HideInInspector]
    public bool isChase = false;
    [HideInInspector]
    public bool isAttack = false;
    [HideInInspector]
    public bool isDie = false;
    bool isPlay = false;

    void Start()
    {
        doorclose = GameObject.Find("Map").transform.GetChild(3).transform.GetChild(7).GetComponent<L_Doo
        enemypool = GameObject.Find("GameManager").GetComponent<L_BossEnemyPool>();
        target = GameObject.FindGameObjectWithTag("Player").transform.GetComponent<Transform>();

        Emanager = GameObject.Find("GameManager").transform.GetComponent<L_EnemyManager>();

        Epatrol = GetComponent<L_EnemyPatrol>();
        Edamage = GetComponent<L_EnemyDamage>();
        audio = GetComponent<AudioSource>();
        EDie = Resources.Load<AudioClip>("EnemySound/SpiderDie");
        tr = GetComponent<Transform>();
        rbody = GetComponent<Rigidbody>();
        navi = GetComponent<NavMeshAgent>();
        ani = GetComponent<Animator>();
        col = GetComponent<BoxCollider>();
        AtkCol = transform.GetChild(2).GetComponent<BoxCollider>();
    }

    private void OnEnable()
    {
        Invoke("ChaseStart", 1);
    }

    void ChaseStart()
    {
        if(!isDie)
        {
            ani.SetBool("IsMove", true);
        }
    }

    void Update()
    {
        h = Input.GetAxis("Horizontal");
        v = Input.GetAxis("Vertical");
```

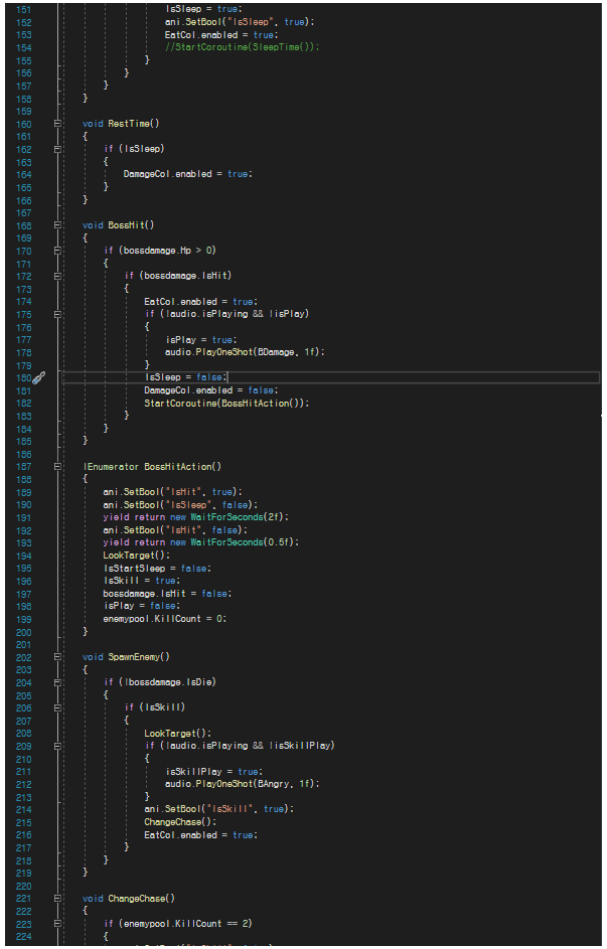
```
    void FreezeVelocity() //적이 플레이어한테 물리 영향 안받게
    {
        rbody.velocity = Vector3.zero;
        rbody.angularVelocity = Vector3.zero;
    }

    private void FixedUpdate()
    {
        if(!isDie)
        {
            if (isChase && !isAttack)
                Chase();
            TargetDis();
            FreezeVelocity();
        }
        EnemyDie();
    }

    void Chase()
    {
        if (navi.isPathStale)
            return;
        navi.SetDestination(target.position);
    }

    void LookTarget()
    {
        lookVec = new Vector3(h, 0, v);
        transform.LookAt(target.position + lookVec);
    }

    void TargetDis()
    {
        float dist = Vector3.Distance(tr.position, target.position);
        if (dist < 7f)
        {
            isChase = true;
            switch (enemyType)
            {

                case Type.A:
                    if (dist < 2f && !isAttack)
                    {
                        StartCoroutine(AttackMotionA());
                        LookTarget();
                    }
                    else if (dist > 2f)
                    {
                        isAttack = false;
                        isChase = true;
                        navi.isStopped = false;
                        ani.SetBool("IsAttack", false);
                        ani.SetBool("IsThink", false);
                        ani.SetBool("IsMove", true);
                    }
                    break;

                case Type.B:
                    if (dist < 5f && !isAttack)
                    {
                        StartCoroutine(AttackMotionB());
                        LookTarget();
                    }
                    else if (dist > 5f)
                    {
                        isAttack = false;
                        isChase = true;
                        navi.isStopped = false;
                        ani.SetBool("IsAttack", false);
                        ani.SetBool("IsThink", false);
                        ani.SetBool("IsMove", true);
                    }
                    break;
            }
```

```
        else
        {
            isChase = false;
        }

    }

    IEnumerator AttackMotionA()
    {
        isAttack = true;
        isChase = false;
        //navi.isStopped = false;
        ani.SetBool("isMove", false);
        ani.SetBool("isThink", false);
        ani.SetBool("isAttack", true);
        yield return new WaitForSeconds(0.35f);
        AtkCol.enabled = true;
        yield return new WaitForSeconds(0.35f);
        AtkCol.enabled = false;
        //navi.isStopped = true;
        ani.SetBool("isAttack", false);
        ani.SetBool("isThink", true);
        yield return new WaitForSeconds(2f);
        isAttack = false;
        isChase = true;
    }

    IEnumerator AttackMotionB()
    {
        isAttack = true;
        isChase = false;
        navi.isStopped = true;
        ani.SetBool("isMove", false);
        ani.SetBool("isThink", true);
        ani.SetBool("isAttack", false);
        yield return new WaitForSeconds(1.0f);
        navi.speed = 10f;
        navi.acceleration = 30f;
        ani.SetBool("isAttack", true);
        ani.SetBool("isThink", false);
        navi.isStopped = false;
        yield return new WaitForSeconds(0.35f);
        AtkCol.enabled = true;
        yield return new WaitForSeconds(0.2f);
        navi.speed = 2f;
        navi.acceleration = 4f;
        yield return new WaitForSeconds(0.15f);
        AtkCol.enabled = false;
        navi.isStopped = true;
        ani.SetBool("isThink", true);
        ani.SetBool("isAttack", false);
        isAttack = false;
        isChase = true;
        yield return new WaitForSeconds(2f);
    }

    void EnemyDie()
    {
        if(isDie)
        {
            if (!audio.isPlaying && !isPlay)
            {
                audio.PlayOneShot(EDie, 1.0f);
                isPlay = true;
            }
            navi.isStopped = true;
            col.isTrigger = true;
            rbody.constraints = RigidbodyConstraints.FreezeAll;
            StartCoroutine(PushPool());
        }
    }

    IEnumerator PushPool()
    {
```

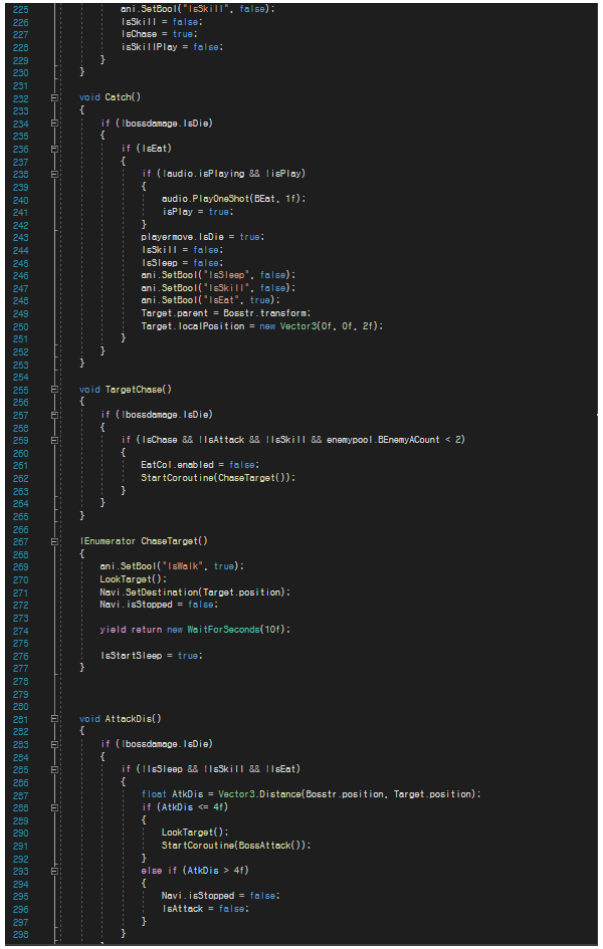
```
    switch (enemyType)
    {

        case Type.A:
            yield return new WaitForSeconds(2.0f);
            isDie = false;
            isChase = false;
            isAttack = false;
            isPlay = false;
            navi.isStopped = false;
            col.isTrigger = false;
            rbody.constraints = RigidbodyConstraints.None;
            rbody.constraints = RigidbodyConstraints.FreezeRotation;
            Epatrol.isPatrol = true;
            Edamage.hp = 1;
            if (!doorclose.IsBoss)
                Emanager.TwoEnemyCount--;
            else if (doorclose.IsBoss)
            {
                enemypool.BEnemyACount--;
                enemypool.KillCount++;
            }
            this.gameObject.SetActive(false);
            yield return new WaitForSeconds(1.0f);
            break;

        case Type.B:
            yield return new WaitForSeconds(2.0f);
            isDie = false;
            isChase = false;
            isAttack = false;
            isPlay = false;
            navi.isStopped = false;
            col.isTrigger = false;
            rbody.constraints = RigidbodyConstraints.None;
            rbody.constraints = RigidbodyConstraints.FreezeRotation;
            Epatrol.isPatrol = true;
            Edamage.hp = 1;
            if (!doorclose.IsBoss)
                Emanager.TwoEnemyCount--;
            else if (doorclose.IsBoss)
                enemypool.BEnemyACount--;
            this.gameObject.SetActive(false);
            yield return new WaitForSeconds(1.0f);
            break;
    }
}
}
```

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;

public class L_Boss : MonoBehaviour
{
    Transform Bosstr;
    Transform Target;
    Transform Resttr;
    Rigidbody rbody;
    AudioSource audio;
    AudioClip BDie;
    AudioClip BEat;
    AudioClip BDamage;
    AudioClip BAngry;
    AudioClip BStart;
    Animator ani;
    Animator DoorAni;
    NavMeshAgent Navi;

    L_BossDamage bossdamage;
    L_BossEnemyPool enemypool;
    L_PlayerMove playermove;

    BoxCollider AtkCol;
    BoxCollider DamageCol;
    BoxCollider EatCol;

    float h, v;
    Vector3 lookVec;

    public bool IsStart;
    public bool IsSkill = false;
    public bool IsEat = false;
    bool isPlay;
    bool isSkillPlay;
    bool IsChase;
    bool IsAttack;
    bool IsSleep;
    bool IsStartSleep;


    void Start()
    {
        DoorAni = GameObject.Find("Map").transform.GetChild(3).transform.GetChild(3).GetComponentInChild
        playermove = GameObject.FindGameObjectWithTag("Player").GetComponent<L_PlayerMove>();

        Bosstr = GetComponent<Transform>();
        Target = GameObject.FindGameObjectWithTag("Player").transform.GetComponent<Transform>();
        Resttr = GameObject.Find("BossStage").transform.GetChild(2).GetComponent<Transform>();
        rbody = GetComponent<Rigidbody>();
        ani = GetComponent<Animator>();
        Navi = GetComponent<NavMeshAgent>();
        audio = GetComponent<AudioSource>();
        BDie = Resources.Load<AudioClip>("EnemySound/BossDie");
        BEat = Resources.Load<AudioClip>("EnemySound/BossAtk");
        BDamage = Resources.Load<AudioClip>("EnemySound/BossDamage");
        BAngry = Resources.Load<AudioClip>("EnemySound/BossAngry");
        BStart = Resources.Load<AudioClip>("EnemySound/BossStart");

        bossdamage = transform.GetChild(0).transform.GetChild(6).transform.GetChild(0).transform.GetChil
        enemypool = GameObject.Find("GameManager").GetComponent<L_BossEnemyPool>();

        AtkCol = transform.GetChild(0).transform.GetChild(1).GetComponent<BoxCollider>();
        DamageCol = transform.GetChild(0).transform.GetChild(6).transform.GetChild(0).transform.GetChild
        EatCol = transform.GetChild(1).GetComponent<BoxCollider>();


    }

    void Update()
    {
        h = Input.GetAxis("Horizontal");
        v = Input.GetAxis("Vertical");
```

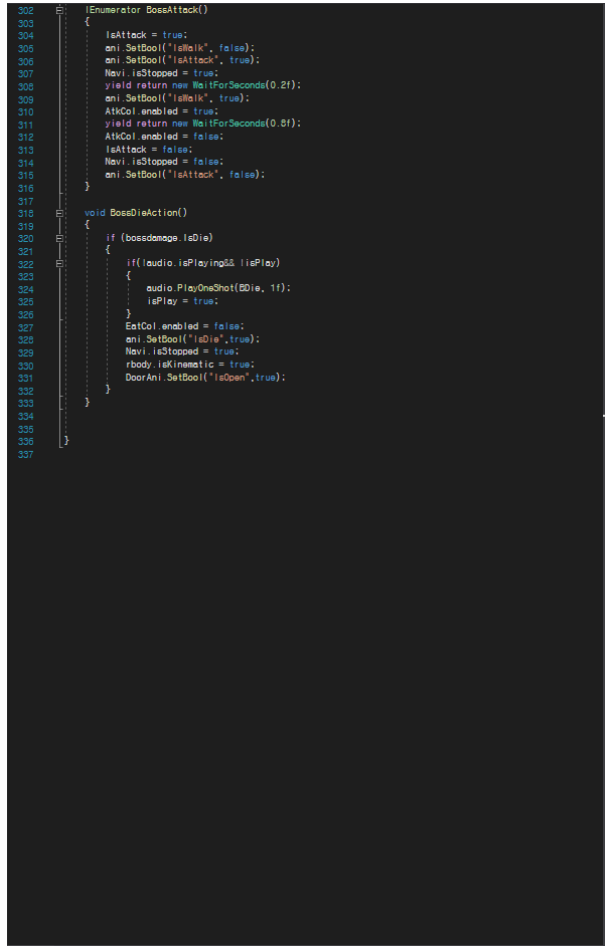
```
    private void FixedUpdate()
    {
        if(!bossdamage.IsDie)
        {
            FreezeVelocity();
            BossStageStart();
            StartSleep();
            AttackDis();
            RestTime();
            BossHit();
            Catch();
            SpawnEnemy();
            TargetChase();
            BossDieAction();
        }
        BossDieAction();
    }

    void FreezeVelocity() //적이 플레이어한테 물리 영향 안받게
    {
        if (!bossdamage.IsDie)
        {
            rbody.velocity = Vector3.zero;
            rbody.angularVelocity = Vector3.zero;
        }
    }

    void LookTarget()
    {
        if (!bossdamage.IsDie)
        {
            lookVec = new Vector3(h, 0, v);
            transform.LookAt(Target.position + lookVec);
        }
    }

    void BossStageStart()
    {
        if (!bossdamage.IsDie)
            if (IsStart)
              StartCoroutine(BossStart());
    }

    IEnumerator BossStart()
    {
        if(!audio.isPlaying)
        {
            audio.PlayOneShot(BStart, 1.0f);
        }
        ani.SetBool("IsStart", true);
        yield return new WaitForSeconds(2.0f);
        ani.SetBool("IsStart", false);
        yield return new WaitForSeconds(1.5f);
        IsStart = false;
        IsStartSleep = true;

    }

    void StartSleep()
    {
        if (!bossdamage.IsDie)
        {
            if (IsStartSleep && !IsSkill)
            {

                IsChase = false;
                Navi.SetDestination(Resttr.position);
                ani.SetBool("IsWalk", true);
                float RestDist = Vector3.Distance(Bosstr.position, Resttr.position);

                if (RestDist <= 3f)
                {
                    Navi.isStopped = true;
                    ani.SetBool("IsWalk", false);
```



```
                    IsSleep = true;
                    ani.SetBool("IsSleep", true);
                    EatCol.enabled = true;
                    //StartCoroutine(SleepTime());
                }
            }
        }
    }

    void RestTime()
    {
        if (IsSleep)
        {
            DamageCol.enabled = true;
        }
    }

    void BossHit()
    {
        if (bossdamage.Hp > 0)
        {
            if (bossdamage.IsHit)
            {
                EatCol.enabled = true;
                if (!audio.isPlaying && !isPlay)
                {
                    isPlay = true;
                    audio.PlayOneShot(BDamage, 1f);
                }
                IsSleep = false;
                DamageCol.enabled = false;
                StartCoroutine(BossHitAction());
            }
        }
    }

    IEnumerator BossHitAction()
    {
        ani.SetBool("IsHit", true);
        ani.SetBool("IsSleep", false);
        yield return new WaitForSeconds(2f);
        ani.SetBool("IsHit", false);
        yield return new WaitForSeconds(0.5f);
        LookTarget();
        IsStartSleep = false;
        IsSkill = true;
        bossdamage.IsHit = false;
        isPlay = false;
        enemypool.KillCount = 0;
    }

    void SpawnEnemy()
    {
        if (!bossdamage.IsDie)
        {
            if (IsSkill)
            {
                LookTarget();
                if (!audio.isPlaying && !isSkillPlay)
                {
                    isSkillPlay = true;
                    audio.PlayOneShot(BAngry, 1f);
                }
                ani.SetBool("IsSkill", true);
                ChangeChase();
                EatCol.enabled = true;
            }
        }
    }

    void ChangeChase()
    {
        if (enemypool.KillCount == 2)
        {
```



```
            ani.SetBool("IsSkill", false);
            IsSkill = false;
            IsChase = true;
            isSkillPlay = false;
        }
    }

    void Catch()
    {
        if (!bossdamage.IsDie)
        {
            if (IsEat)
            {
                if (!audio.isPlaying && !isPlay)
                {
                    audio.PlayOneShot(BEat, 1f);
                    isPlay = true;
                }
                playermove.IsDie = true;
                IsSkill = false;
                IsSleep = false;
                ani.SetBool("IsSleep", false);
                ani.SetBool("IsSkill", false);
                ani.SetBool("IsEat", true);
                Target.parent = Bosstr.transform;
                Target.localPosition = new Vector3(0f, 0f, 2f);
            }
        }
    }

    void TargetChase()
    {
        if (!bossdamage.IsDie)
        {
            if (IsChase && !IsAttack && !IsSkill && enemypool.BEnemyACount < 2)
            {
                EatCol.enabled = false;
                StartCoroutine(ChaseTarget());
            }
        }
    }

    IEnumerator ChaseTarget()
    {
        ani.SetBool("IsWalk", true);
        LookTarget();
        Navi.SetDestination(Target.position);
        Navi.isStopped = false;

        yield return new WaitForSeconds(10f);

        IsStartSleep = true;
    }


    void AttackDis()
    {
        if (!bossdamage.IsDie)
        {
            if (!IsSleep && !IsSkill && !IsEat)
            {
                float AtkDis = Vector3.Distance(Bosstr.position, Target.position);
                if (AtkDis <= 4f)
                {
                    LookTarget();
                    StartCoroutine(BossAttack());
                }
                else if (AtkDis > 4f)
                {
                    Navi.isStopped = false;
                    IsAttack = false;
                }
            }
```



```
    IEnumerator BossAttack()
    {
        IsAttack = true;
        ani.SetBool("IsWalk", false);
        ani.SetBool("IsAttack", true);
        Navi.isStopped = true;
        yield return new WaitForSeconds(0.2f);
        ani.SetBool("IsWalk", true);
        AtkCol.enabled = true;
        yield return new WaitForSeconds(0.8f);
        AtkCol.enabled = false;
        IsAttack = false;
        Navi.isStopped = false;
        ani.SetBool("IsAttack", false);
    }

    void BossDieAction()
    {
        if (bossdamage.IsDie)
        {
            if(!audio.isPlaying&& !isPlay)
            {
                audio.PlayOneShot(BDie, 1f);
                isPlay = true;
            }
            EatCol.enabled = false;
            ani.SetBool("IsDie",true);
            Navi.isStopped = true;
            rbody.isKinematic = true;
            DoorAni.SetBool("IsOpen",true);
        }
    }


}
```

```
using UnityEngine;

public class Save : MonoBehaviour
{
    Transform Paticle;
    L_LightOnOff Lighter;
    BoxCollider Bcol;
    AudioSource audio;
    AudioClip firebust;

    bool isTri = false;
    bool isSave = false;

    void Start()
    {
        player = GameObject.FindGameObjectWithTag("Player");
        Paticle = transform.GetChild(1).GetComponent<Transform>();
        Lighter = GameObject.FindGameObjectWithTag("Player").transform.GetComponent<L_LightOnOff>();
        Bcol = GetComponent<BoxCollider>();
        audio = GetComponent<AudioSource>();
        firebust = Resources.Load<AudioClip>("FireBust");
    }

    private void FixedUpdate()
    {
        TriSave();
    }

    private void OnTriggerEnter(Collider col)
    {
        if(col.CompareTag("Player"))
        {
            isTri = true;
            if (Lighter.IsLightOn && !isSave)
            {
                Save();
                Paticle.gameObject.SetActive(true);
                if(!audio.isPlaying)
                    audio.PlayOneShot(firebust, 1.0f);
                Bcol.enabled = false;
                isSave = true;
            }

        }
    }

    void TriSave()
    {
        if(isTri)
        {
            if (!Lighter.IsLightOn && Input.GetKeyDown(KeyCode.F) && !isSave)
            {
                Save();
                Paticle.gameObject.SetActive(true);
                if (!audio.isPlaying)
                    audio.PlayOneShot(firebust, 1.0f);
                Bcol.enabled = false;
                isSave = true;
            }
        }
    }

    private void OnTriggerExit(Collider col)
    {
        if (col.CompareTag("Player"))
        {
            isTri = false;
        }
    }

    public void Save()
    {
        PlayerPrefs.SetFloat("x", player.transform.position.x);
        PlayerPrefs.SetFloat("y", player.transform.position.y);
        PlayerPrefs.SetFloat("z", player.transform.position.z);
        PlayerPrefs.SetFloat("Rot_y", player.transform.eulerAngles.y);
    }
}
```

3_게임 플레이

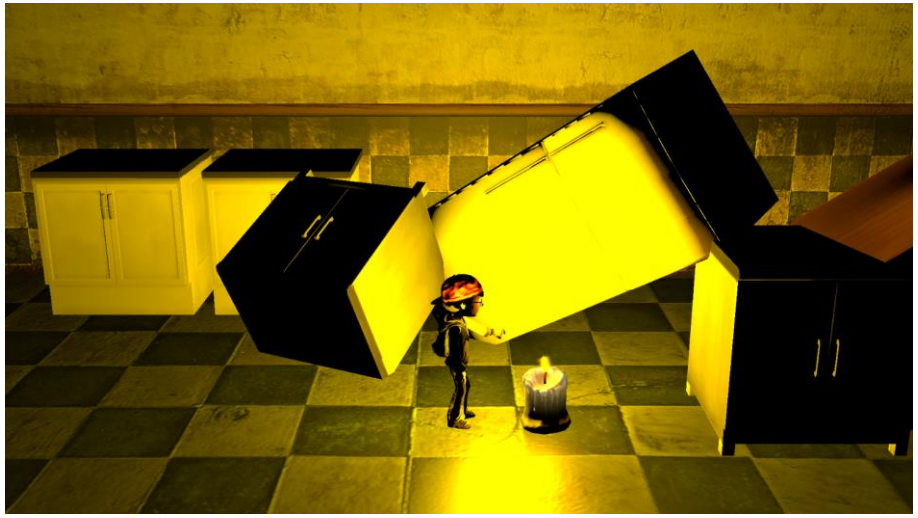
게임플레이 _ 맵 1


ESCAPE
NIGHTMARES
PLAY
LAST CHECKPOINT
SELECT MAP
EXIT


Settings
Light Intensity
Audio
Last CheckPoint
Resume
MainMenu

게임플레이 _ 맵 1

게임플레이 _ 맵 2

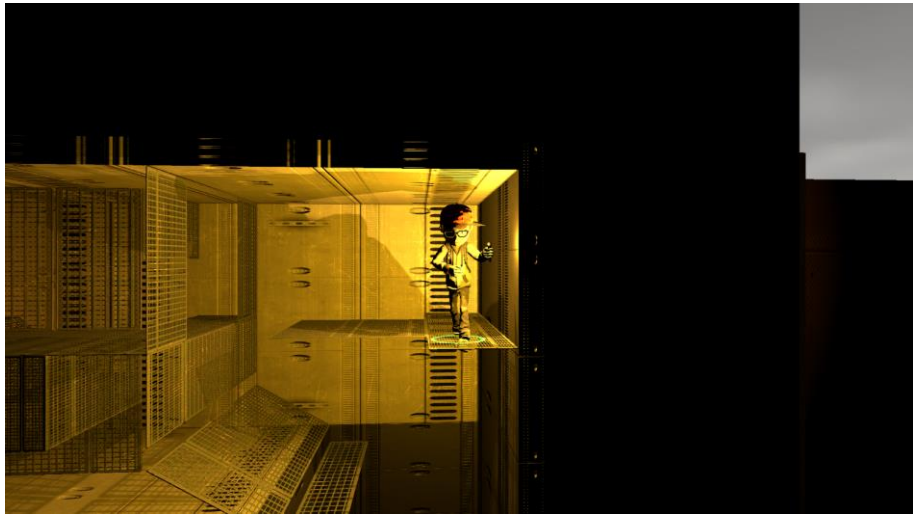


게임플레이 _ 맵 2

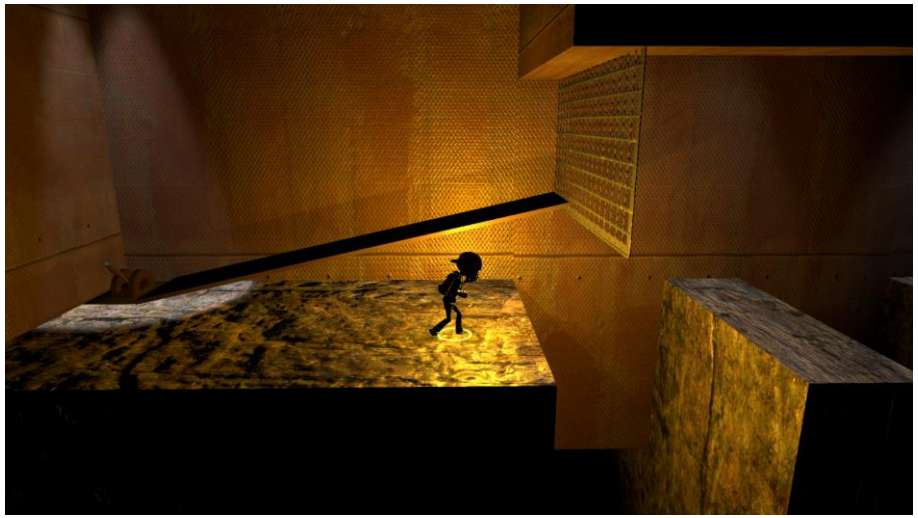
게임플레이 _ 맵 2

게임플레이 _ 맵 3

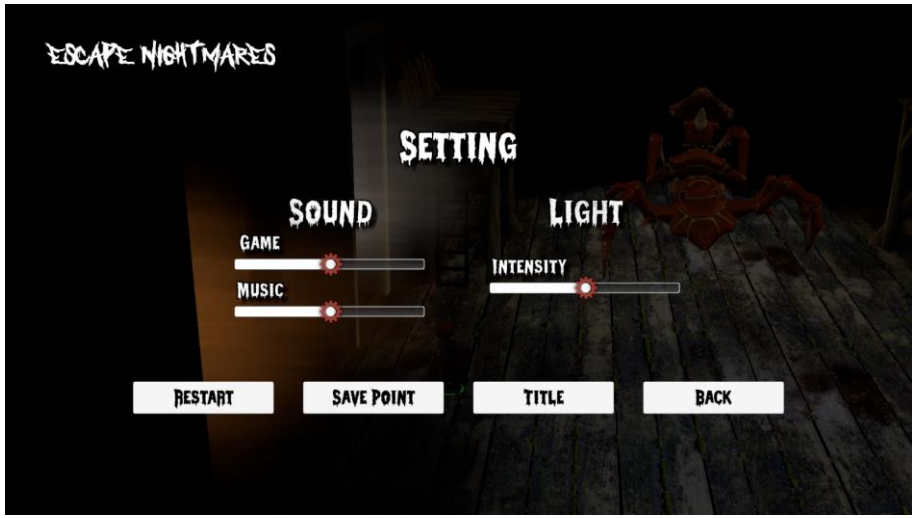
게임플레이 _ 맵 3



게임플레이 _ 맵 3



게임플레이 _ 맵 3


ESCAPE NIGHTMARES
SETTING
SOUND
LIGHT
GAME
MUSIC
INTENSITY
RESTART
SAVE POINT
TITLE
BACK

게임플레이 _ 로딩, 엔딩 크레딧

ESCAPE NIGHTMARES

CLEAR

Team Name _ Null develop

Member "최영근, 하재헌, 임영균"

MAP1 Produce:'하재헌'

Scene
"Map1 Title UI / Play Map"
Prodution
"Map1 Player / Map1 Enemy / Map1 Objects / Setting UI"

MAP2 Produce:'최영근'

Scene
"Map2 Main Title / Play Map"
Prodution
"Map2 Player / Map2 Enemy / Map2 Objects / Setting UI"

MAP3 Produce:'임영균'

Scene
"Map3 Title UI / Play Map / Loding UI / Ending Credit UI"
Prodution
"Map3 Player / Map3 Enemy_Spider, Boss / Map3 Objects / Setting UI"

TITLE

Thank You