

최종현 포트폴리오 – Labyrinthos 개발 개요

1. 프로젝트 역할

1) 팀장 업무

- 팀원의 역량 파악 및 업무 분배
- 매주마다 진행되는 프로젝트 진행 상황 발표를 위한 발표 대본 작성 및 발표, 질의응답
- GitHub를 통한 멤버 및 프로젝트 관리

2) 기획 단계 업무('21.10.05 ~ '21.11.02)

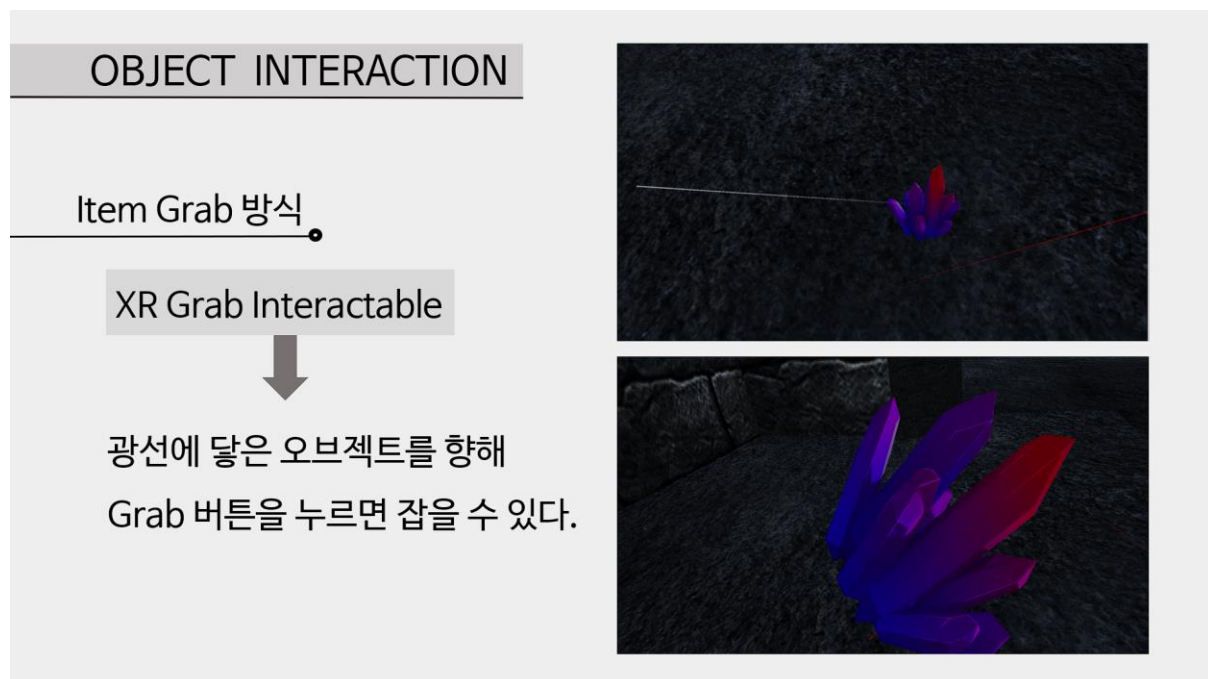
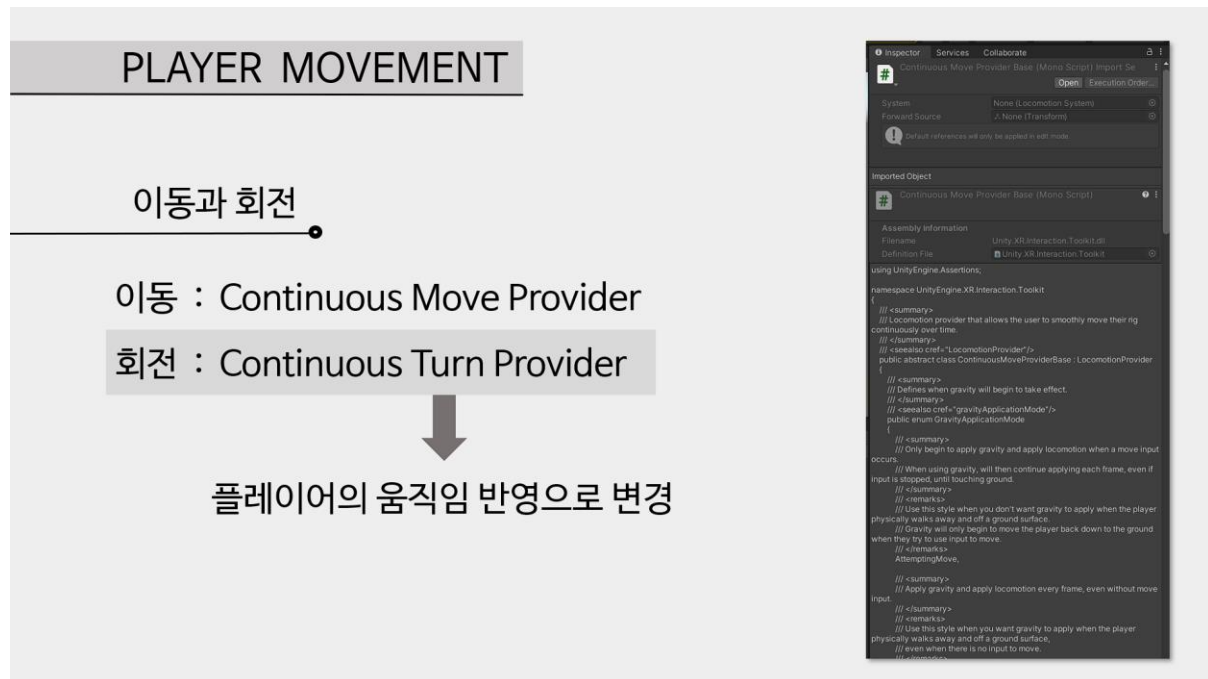
- 콘셉트 아이디어 제시 및 채택
- 게임 상세 스토리 작성
- 게임 Asset 탐색 및 적용

3) 개발 단계 업무('21.11.03 ~ '21.12.07)

- 플레이어 충돌 메커니즘 설계 및 구현
- 플레이어의 시야에 따라 위치를 바꾸는 NPC 아리아드네의 로직 설계 및 구현
- 방탈출을 위한 3개의 맵 중에서 첫 번째 맵 설계 및 구현
- 방탈출을 위한 3개의 맵 중에서 두 번째 맵의 1번 문제 설계 및 구현
- 스토리 연출을 위한 모든 회상 장면 설계 및 구현
- 엔딩을 위한 마지막 맵 디자인 연출

2. 기술 스택 및 아키텍처

- 플레이어의 이동과 상호 작용



방탈출 게임이라는 콘셉트를 살리기 위해서 플레이어의 자유로운 움직임을 보장하는 조이스틱 기반의 컨트롤을 채용했습니다. 또한 오브젝트와의 상호 작용은 플레이어가 직접 움직이는 것이 피로할 것으로 판단해서 직접 오브젝트를 잡는 방식이 아닌, 오브젝트를 손으로 끌어당기는 방식으로 구현했습니다. 이 두 가지를 위해서 Oculus 개발자 센터에서 제공하는 SDK의 컴포넌트를 활용했습니다.

- 플레이어 충돌 메커니즘

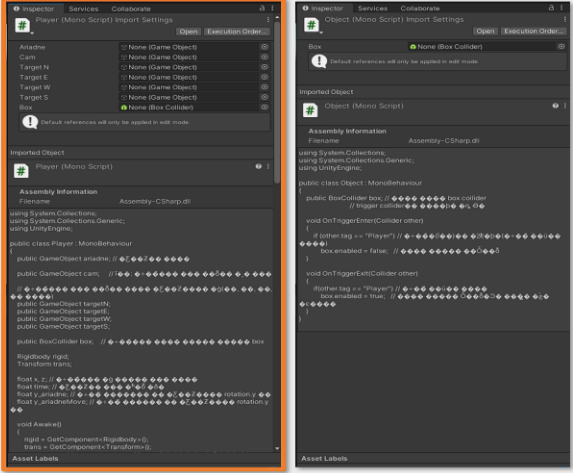
OBJECT INTERACTION

충돌 예외 처리

1. Player Script

↓

플레이어와 지형의
물리 영역을 판정함



OBJECT INTERACTION

충돌 예외 처리

2. Object Script

↓

플레이어와 오브젝트,
오브젝트와 지형의 물리 영역 판정



VR 기기의 특성상 플레이어가 이동하면서 벽에 부딪히면 벽 너머가 보이거나 벽을 뚫고 지나가는 문제가 있어서 수정할 필요가 있었습니다. 간단히 Box Collider를 설정해서 해결하려 했으나, 오브젝트 근처에서 오브젝트와 충돌하거나, 오브젝트를 잡았을 때 오브젝트가 튕겨 나가는 문제가 발생했습니다. 모든 문제를 동시에 해결하기 위해서 평소에는 플레이어의 Collider를 비활성화했다가 Trigger이벤트를 통해 벽과 충돌할 때만 Box Collider가 활성화되도록 코드를 작성했습니다. 동시에 오브젝트 전용 스크립트인 "object.cs"를 제작해서 오브젝트와 플레이어가 충돌했을 때 오브젝트의 Collider를 비활성화하고, 플레이어의 범위를 벗어날 때 다시 Collider를 활성화시키는 방식으로 코드를 구성했습니다. 한편 플레이어가 벽에 닿았을 때 속도와 각속도가 남아 있으면 당구공처럼 튕겨 나가는 현상을 막기 위해 수치를 조정하도록 설정했습니다. 또한 오브젝트의 Collider가 켜지기 전에 바닥에 닿으면 그대로 바닥을 통과할 가능성을 고려해서 플레이어의 Box Collider와 바닥 사이에 여유 공간을 만들어서 문제를 방지했습니다.

- 회상 장면 구현

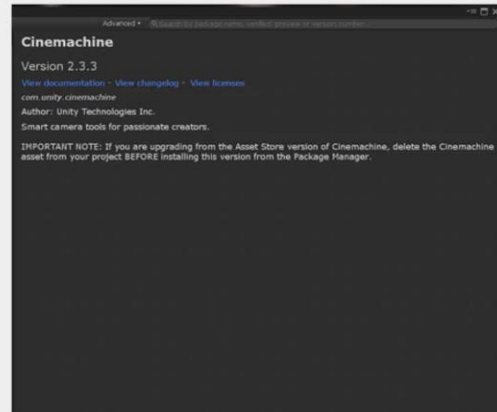
FLASHBACK SCENE

회상 씬 연출

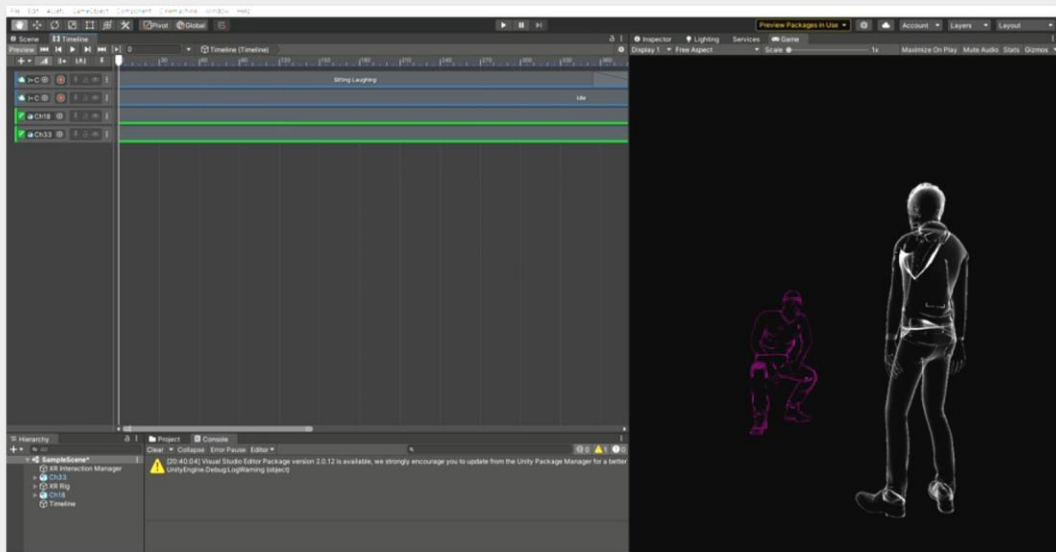
Cinemachine



Timeline에 따른 애니메이션 연출



FLASHBACK SCENE

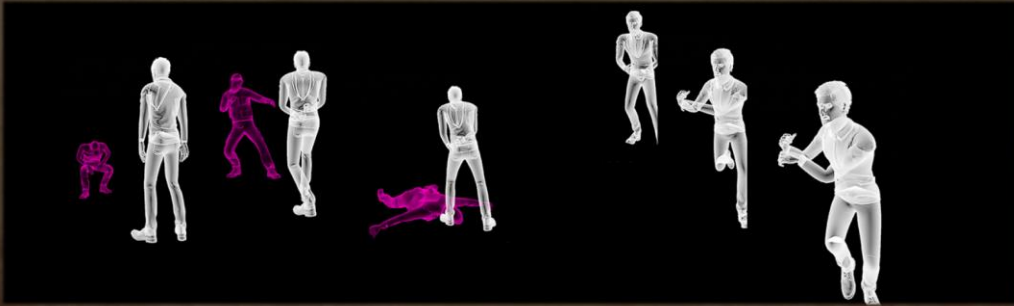


회상 장면은 유니티의 시네머신 기능을 이용해서 구현했습니다. 기본적인 애니메이션을 제공하는 Mixamo 사이트의 애니메이션과 캐릭터를 활용하고, 홀로그램 셰이더를 적용해서 플레이어에게 무대 위에서 연극을 보는 듯한 느낌을 주도록 제작했습니다. 처음에는 테두리만으로 캐릭터의 모습을 구현했지만 테스트 결과 시각적인 전달이 부족하다는 피드백이 있어서 셰이더 설정을 통해 다음과 같이 보다 선명하게 캐릭터를 구현했습니다.

FLASHBACK 1

회상 1 < 첫 번째 살인 >

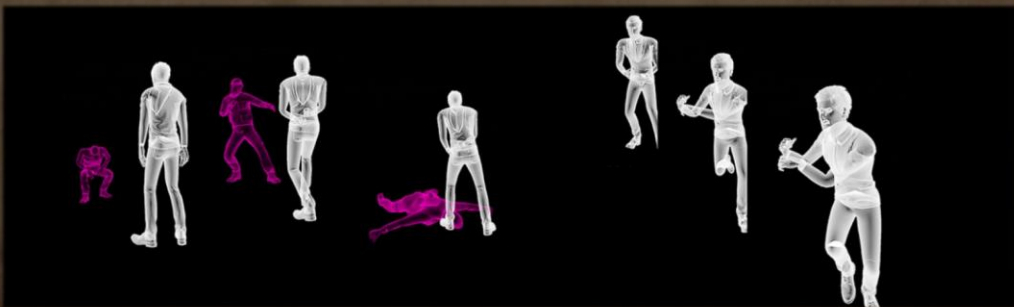
현실의 주인공이 아버지와의 다툼을 살해한 조폭들을 만나는 장면.
조폭 한 명과의 대화 끝에 주인공이 흥기를 꺼내 들지만, 상대는 가소롭다는 듯이 비웃는다.
이 후 조폭은 주인공을 이길 수 있다는 생각에 덤비지만, 단칼에 목숨을 잃고 만다.



FLASHBACK 1

회상 1 < 첫 번째 살인 >

첫 번째 살인이 끝나기 무섭게 주인공은 주변의 다른 조폭들을 베기로 한다.
플래시 백 연출을 위해 유니티 시네머신을 이용한 애니메이션 장면 제작.
VR의 특징을 살리기 위해 주인공이 플레이어에게 달려드는 모습을 구현.



FLASHBACK 2

회상 2 < 황소파 보스와의 싸움 >

주인공이 복수를 위해 황소파 보스와 싸움을 하는 장면.
보스를 최대한 고통스럽게 만들겠다는 의도와 달리,
검을 꺼내지 않아 일방적으로 당하는 주인공.



FLASHBACK 2

회상 2 < 황소파 보스와의 싸움 >

결국 검을 꺼내기로 한 주인공은 여러 번의 공격 끝에 보스를 쓰러뜨린다.
마치 실제로 눈 앞에서 싸움이 일어나는 것처럼 구도를 연출하여 VR의 특성을 살림.



3. 프로젝트 개발 단계

- 플레이어의 시야에 따라 위치를 바꾸는 NPC 아리아드네의 로직 구현

```
void Awake()
{
    rigid = GetComponent<Rigidbody>();
    trans = GetComponent<Transform>();
    x = trans.position.x; z = trans.position.z; // 최초 플레이어의 위치 저장
}

void FixedUpdate()
{
    AriadneLookAt();
}

// 플레이어의 시점 기준, 왼쪽에 아리아드네가 위치하도록 만들
// 플레이어의 시점이 바뀌면 아리아드네의 위치도 시점에 맞게 이동함
// 플레이어가 움직이면 아리아드네도 플레이어의 정면 방향을 볼
void AriadneLookAt()
{
    time += Time.deltaTime;

    if (cam.transform.rotation.y >= -0.35f && cam.transform.rotation.y < 0.25) // 플레이어가 북쪽 통로를 볼 때
        AriadneMove(targetN.transform.position, 130, 0);
    else if (cam.transform.rotation.y >= 0.45f && cam.transform.rotation.y < 0.8f) // 플레이어가 동쪽 통로를 볼 때
        AriadneMove(targetE.transform.position, -130, 90);
    else if (cam.transform.rotation.y <= -0.55f && cam.transform.rotation.y > -0.8f) // 플레이어가 서쪽 통로를 볼 때
        AriadneMove(targetW.transform.position, 50, -90);
    else if (cam.transform.rotation.y >= 0.95f || cam.transform.rotation.y <= -0.95f) // 플레이어가 남쪽 통로를 볼 때
        AriadneMove(targetS.transform.position, -50, 180);

    if (time > 0.1f) // 시간이 경과하면서
    {
        if (x == trans.position.x && z == trans.position.z) // 플레이어가 움직이지 않았으면
            ariadne.transform.rotation = Quaternion.Euler(0, y_ariadne, 0); // 플레이어를 바라봄
        else // 플레이어가 이동하는 경우
        {
            ariadne.transform.rotation = Quaternion.Euler(0, y_ariadneMove, 0); // 플레이어가 가는 방향을 바라봄
            x = trans.position.x; z = trans.position.z; // 이후 현재 위치를 저장
        }
        time = 0; // 판정 시간 초기화
    }
}

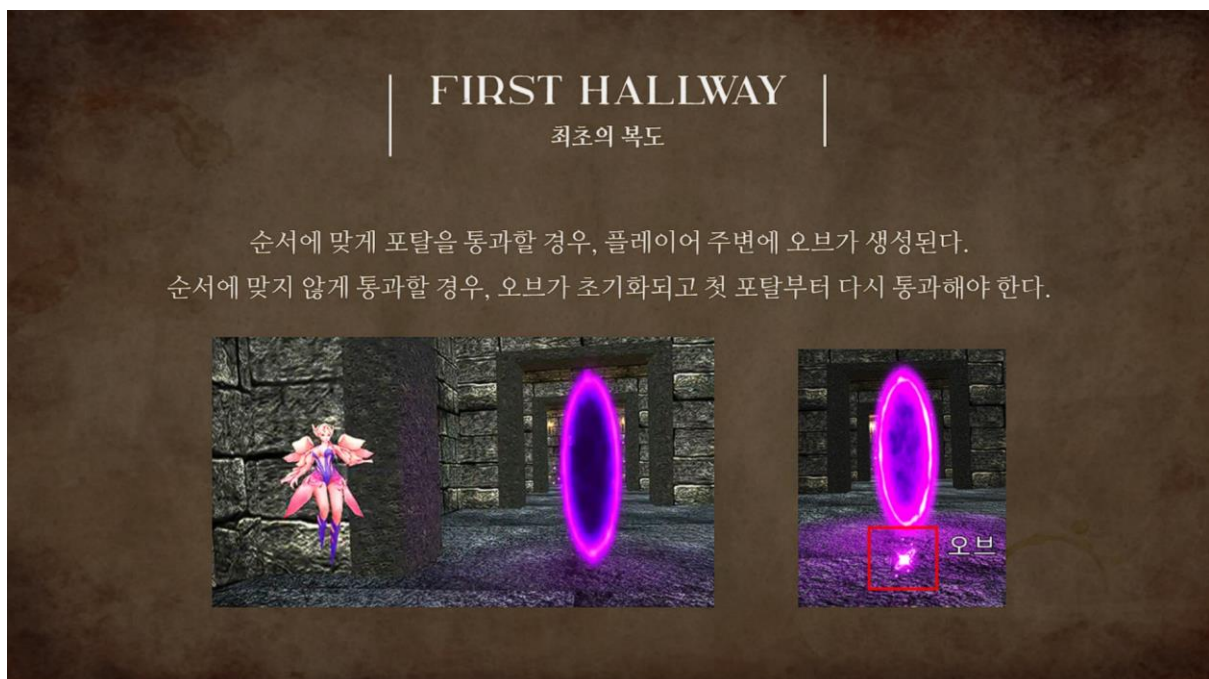
void AriadneMove(Vector3 target, int y, int y_move)
{
    ariadne.transform.position = Vector3.Lerp(ariadne.transform.position, target, 0.05f); // 아리아드네 이동
    y_ariadne = y; // 멈춰있는 플레이어의 시점에 맞게 아리아드네의 y축 조정
    y_ariadneMove = y_move; // 움직이는 플레이어의 시점에 맞게 아리아드네의 y축 조정
}
```



테세우스 신화를 모티브로 하는 점을 살리기 위해 신화에 등장하는 아리아드네라는 캐릭터를 어

시스템트 역할로 제작했습니다. 게임에 영향을 주는 것은 아니지만, 플레이어의 주변을 지키는 펫, 혹은 동료와 같은 존재로 시야에서 벗어나지 않기를 원했고, 플레이어가 몸을 움직여 시야를 바꾸더라도 아리아드네가 언제나 플레이어의 왼쪽에 위치하도록 만들었습니다. 또한 아리아드네의 고정적인 움직임을 피하고자 플레이어가 이동할 경우 플레이어가 이동하는 방향을 함께 바라보고, 정지 상태일 때는 플레이어를 바라보도록 구현했습니다.

- 방탈출을 위한 3개의 맵 중에서 첫 번째 맵 설계 및 구현




```

bool first, second, third, fourth; // 각 방향의 포탈 순서를 지켰는지 확인

void Update()
{
    StageClear();
    OrbOnOff();
}

void StageClear()
{
    if (portal_1.ischeck) // 첫번째 포탈을 통과한 경우
    {
        first = true;
        portal_1.ischeck = false;
    }

    if (portal_2.ischeck) // 두번째 포탈을 통과한 경우
    {
        if (first) // 첫번째를 통과한 기록이 있는 경우
            second = true; // 두번째 통과를 승인
        portal_2.ischeck = false;
    }

    if (portal_3.ischeck) // 세번째 포탈을 통과한 경우
    {
        if (!second) // 순서에 맞게 두번째포탈을 통과한 기록이 없으면
        {
            first = false; // 포탈 통과 기록 초기화
        }
        else // 순서에 맞게 포탈을 통과한 경우
            third = true; // 세번째 포탈 통과를 승인
        portal_3.ischeck = false;
    }

    if (portal_4.ischeck) // 네번째 포탈을 통과한 경우
    {
        if (!third) // 세번째 포탈을 통과한 기록이 없으면
        {
            first = false; // 전체 포탈 통과기록 초기화
            second = false;
        }
        else // 순서에 맞게 통과한 경우
            fourth = true; // 네번째 포탈 통과를 승인
        portal_4.ischeck = false;
    }
}

```

방탈출과 미궁이라는 콘셉트에 맞게 간단한 미로를 제작하는 것을 목표로 개발한 첫 번째 방의 퍼즐입니다. 미로의 느낌을 주기 위해서 순서를 맞추지 못하면 진행 과정을 초기화하도록 bool 변수를 통해 퍼즐 상황을 설계했습니다. 반면 순서를 맞춘 경우에는 플레이어가 인지할 수 있도록 SetActive 설정을 통해 플레이어 주변의 표식을 활성화/비활성화하는 방향으로 구현했습니다.

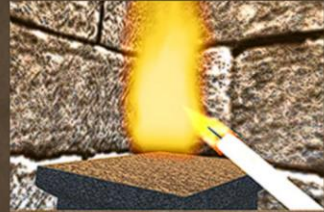
- 방탈출을 위한 3개의 맵 중에서 두 번째 맵의 1번 문제 설계 및 구현



FIRST ROOM

첫 번째 방

책상 위의 촛불을 이용해, 불이 붙은 촛불을 제단에 가져다 대면 불이 피어난다.



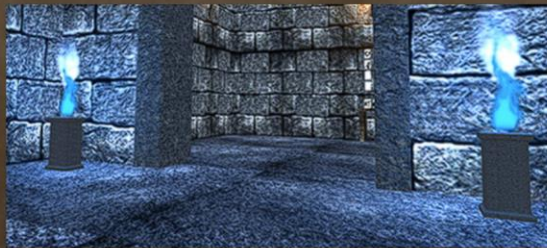
FIRST ROOM

첫 번째 방

4개의 제단에 불을 모두 활성화시키면 아래와 같이 꺼지지 않는 푸른 불꽃으로 변한다.

단, 제단의 불은 20초 동안 지속되므로, 제한시간 안에 4개의 불을 모두 붙여야 한다.

또한 촛불을 바닥에 떨어트리면 불이 꺼지기 때문에 주의하여야 한다.



```

public class PillarFire : MonoBehaviour
{
    public GameObject fire; // 기둥의 불

    public bool isfire; // 불이 붙었는지
    public bool isblue; // 불이 파랗게 변했는지

    float time; // 시간제한

    void Update()
    {
        if(!isblue && isfire)
            FireOff();
    }

    void FireOff() // 일정 시간이 지나면 불이 꺼지도록
    {
        time += Time.deltaTime;
        if(time > 20) // 일정 시간이 지나면
        {
            fire.SetActive(false); // 불이 꺼지고
            isfire = false; // 불이 꺼진 것을 저장하고
            time = 0; // 시간 초기화
        }
    }

    void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Fire" && !isblue) // 파란 불이 아닌데 불이 근처에 있으면
        {
            fire.SetActive(true); // 불 오브젝트를 켜고
            isfire = true; // 불이 붙은 것을 저장
        }
    }
}

```

```

public class PillarManager : MonoBehaviour
{
    // 네 기둥의 일반 불꽃과 푸른 불꽃
    public GameObject fire1;
    public GameObject fire2;
    public GameObject fire3;
    public GameObject fire4;
    public GameObject blueFire1;
    public GameObject blueFire2;
    public GameObject blueFire3;
    public GameObject blueFire4;

    // 불이 꺼지는 것을 컨트롤하는 4개의 트리거
    public PillarFire pillarTrigger1;
    public PillarFire pillarTrigger2;
    public PillarFire pillarTrigger3;
    public PillarFire pillarTrigger4;

    void Update()
    {
        TurnBlue();
    }

    void TurnBlue() // 4개의 불꽃이 활성화되면 푸른 불꽃으로 바꾸고 일반 불꽃이 나오지 않게 된다.
    {
        if(pillarTrigger1.isfire && pillarTrigger2.isfire && pillarTrigger3.isfire && pillarTrigger4.isfire)
        {
            fire1.SetActive(false); fire2.SetActive(false); fire3.SetActive(false); fire4.SetActive(false); // 일반 불꽃 off
            blueFire1.SetActive(true); blueFire2.SetActive(true); blueFire3.SetActive(true); blueFire4.SetActive(true); // 푸른 불꽃 켜기
            pillarTrigger1.isblue = true; pillarTrigger2.isblue = true; pillarTrigger3.isblue = true; pillarTrigger4.isblue = true; // 일반 불꽃 켜짐 제한
        }
    }
}

```

방탈출의 테마가 그리스 신화 모티브인 것을 이용해서 프로메테우스에 대한 이야기를 힌트로 제시했으며, 불을 붙이는 아이디어는 게임 "Zwei!!" 에서 아이디어를 얻었습니다. "Zwei!!" 에서는 빠른 속도로 4개의 초에 불을 붙여야 문이 열리는 퍼즐이 있는데, 이것을 신화 배경에 맞게 기둥에 불을 올리는 것으로 변경했습니다. 퍼즐의 솔루션은 Trigger 이벤트를 통해 구현했으며, SetActive 설정을 통해 불꽃이 색깔을 바뀌서 퍼즐이 해결되었음을 플레이어에게 직관적으로 전달하도록 설계했습니다.