angst

Capstone Design 002

최종 보고서

김두현

장규석

조병수

보고서 목차

게임정보

개발 환경

게임 사양

사용 에셋

기타 리소스

게임소개

게임 소개 및 게임 특징

조작법, UI

스테이지

구현

플레이어 조작, 애니메이션

상호작용 가능한 오브젝트

게임 내 이벤트

중력의 변화

사운드

슬라이드 퍼즐

적 오브젝트 & 길 찾기 AI

Book UI

자물쇠 해제 UI

데이터의 저장

최적화 & 마무리

드로우콜과 수동 배칭

개선 사항 및 소감

**\*보고서를 읽기 전에......**

**Unity**에서 제공하는 기능과 저희가 직접 구현한 클래스들을 구분하기 위해 **Unity**에서 제공하는 기능들은 **노란색**으로, 저희가 구현한 클래스들은 **초록색**으로 하이라이트 처리하였습니다.

구현 부분에서 코드들의 상세 내용을 담기에는 보고서의 분량이 적으니 간단한 설명과 함께 관련 스크립트를 명시하였습니다. 필요시 해당 스크립트의 주석을 참고하시면 됩니다.

게임 정보

개발 환경

**Unity 2019 3.91f**

**Unity** **Collaborate**를 사용한 프로젝트 공유

데모영상 링크 : <https://www.youtube.com/watch?v=49n5vgRoh_E&feature=youtu.be>

게임 사양

\* 다양한 기기에서 시험해 봐야 하지만 측정 여건이 되지 않기 때문에 조원들 중 가장 낮은 스펙의 기기를 명시

**CPU** : Ryzen 5 3500X **GPU** : GeForce GTX 1060 3GB **RAM** : 16GB

**필요 디스크 용량** : 500MB

사용 에셋

DOTween : <https://assetstore.unity.com/packages/tools/animation/dotween-hotween-v2-27676>

병원 : <https://assetstore.unity.com/packages/3d/environments/urban/the-old-hospital-12995>

적 : <https://assetstore.unity.com/packages/3d/characters/creatures/total-horror-3138>

귀신 : <https://assetstore.unity.com/packages/3d/characters/humanoids/rin-new-anime-style-character-for-games-and-vrchat-174995>

자물쇠 : <https://assetstore.unity.com/packages/3d/props/padlock-pack-153882>

인형 : <https://assetstore.unity.com/packages/3d/props/horror-assets-69717>

알약 : <https://assetstore.unity.com/packages/3d/props/tools/survival-game-tools-139872>

제단 : <https://assetstore.unity.com/packages/3d/environments/grave-18-81974>

플레이어 : <https://assetstore.unity.com/packages/3d/characters/humanoids/humans/man-in-a-suit-51662>

기타 리소스

이미지나 사운드들은 모두 변형이 가능하고 상업적 이용과 출처를 밝히지 않아도 사용가능한 CC0 라이센스들 입니다.

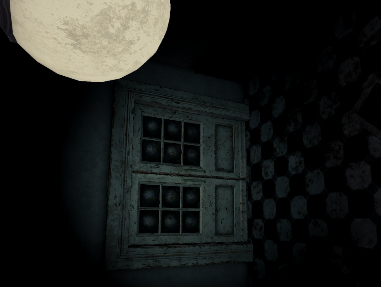
애니메이션은 해당 사이트에서 제공하는 무료 모션 캡쳐 애니메이션을 사용하였습니다.

MIXAMO : <https://www.mixamo.com/#/>

게임소개

Angst는 3인칭 공포 어드벤처 게임입니다. 플레이어는 제한된 공간에서 단서를 찾아 비밀번호를 얻어 병원을 탈출해야 합니다. 서로 다른 특징을 가지고 있는 두 개의 스테이지가 구현되어 있으며 예상되는 플레이 시간은 총 20 -30분 입니다.

Angst의 특징이라면 중력이 변하여 플레이를 혼란스럽게 하는 것입니다. 같은 공간이라도 다른 시점에서 바라볼 수 있기 때문에 플레이어에게 새로운 경험을 제공합니다.



조작법 & UI

이동 & 화면 회전 : WASD + 마우스

점프 & 올라가기 : 스페이스 바

아이템 사용 : 상단 숫자 1

천천히 걷기 : C

라이트 켜기 / 끄기 : 마우스 왼쪽

메뉴 : ESC

\*\* 간혹 컴퓨터 환경에 따라 한글 입력 상태에서는 조작이 되지 않을 수 있습니다. 한/영 키를 눌러 영문 입력 상태로 전환해주세요.

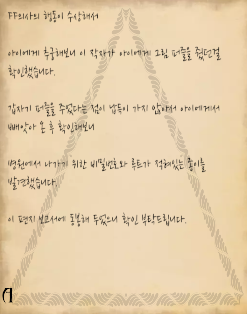


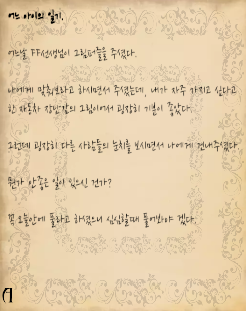
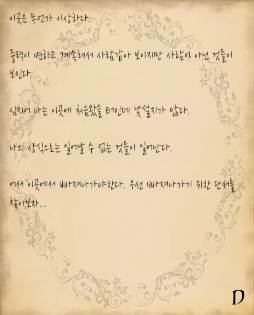
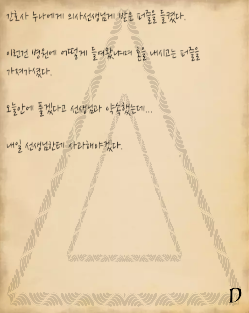
1 : 스트레스 저항 수치, 0이 되면 게임 오버

2 : 획득한 아이템 개수, 쿨 타임 표현

스테이지

스테이지1은 튜토리얼의 성격도 가지고 있기 때문에 쉬운 난이도로 디자인하였습니다. 플레이어는 맵을 따라서 단서를 얻고 이벤트들을 보며 얻은 단서를 바탕으로 탈출 비밀번호를 알아내면 됩니다.

단서를 얻을 때 마다 Book UI에는 단서 텍스트가 추가되어 플레이어는 언제든지 내용을 확인할 수 있습니다. 텍스트와 함께 책 페이지에는 문양이 랜덤하게 추가되는데 스테이지 1 에서는 이 문양을 비밀번호의 힌트로 사용합니다.



총 문양의 종류는 6가지가 있으며 이 문양들은 매 게임마다 랜덤으로 알파벳 A B C D E F에 대응됩니다. 게임을 진행하다 보면 각 문양들이 어떤 알파벳과 대응되는지 주어지며 이를 토대로 알파벳 자물쇠를 풀면 게임 클리어에 필요한 비밀번호를 얻을 수 있습니다.

스테이지2 에서는 적이 등장하여 플레이어의 게임 클리어를 방해합니다. 플레이어는 적을 피해 맵 이곳저곳에 흩어진 인형들을 모두 모아 제단에 바치면 비밀번호를 얻을 수 있습니다.



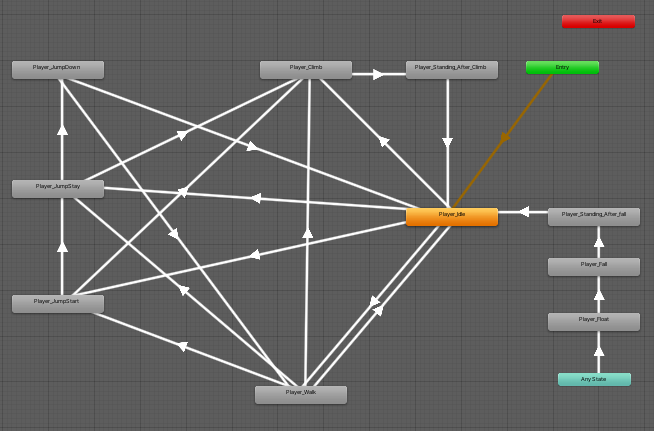
플레이어는 라이트 불빛만으로 어둠속을 돌아다녀야 합니다. 이때 적의 소리를 듣고 위치를 파악하여 적절한 곳으로 피해야 합니다.

플레이어가 적에게 대항할 수단은 주어지지 않습니다. 일부로 난이도 있는 스테이지로 구성하였습니다.

**구현**

**플레이어 조작, 애니메이션**

Angst에서는 플레이어의 조작과 그에 따른 애니메이션을 효과적으로 컨트롤하기 위해 **StateMachineBehaviour** 클래스를 이용하여 FSM으로 디자인하였습니다. 덕분에 각각의 상태에서 받아들이는 입력 값들을 적절히 사용하여 애니메이션 상태전환을 시키고 물리적인 이동을 컨트롤할 수 있었습니다.

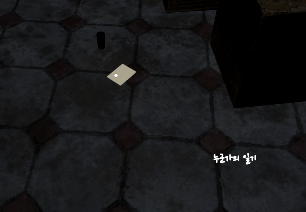


(Unity Mecanim을 이용한 FSM 구성)

관련 스크립트 : Assets/Scripts/PlayerControl, PlayerState 폴더 내 모든 스크립트

**상호작용 가능한 오브젝트**

모든 상호작용 가능한 오브젝트들은 **Interactable** 클래스를 상속합니다. 이 클래스에서 마우스가 올라가 있고 일정거리 이내일 경우에만 상호작용이 가능해집니다. 또한 상호작용 중 게임 내 시간을 멈출지 안 멈출지도 컨트롤 할 수 있습니다.

이러한 기본 클래스를 상속하여 다양한 오브젝트들의 동작을 하나의 공통된 형태로 구현하여 개발기간을 단축하였습니다. 게임 내 단서, 아이템, 자물쇠, 제단, 퍼즐 등이 이런 식으로 구현되었습니다.

(왼쪽부터 단서, 아이템, 제단)

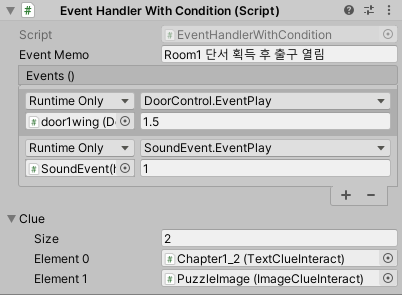
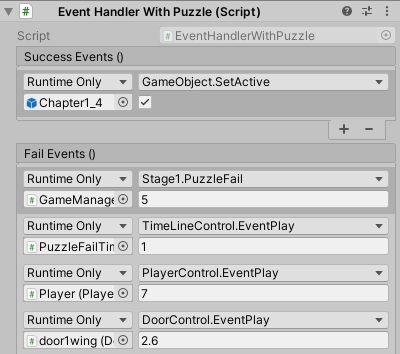
관련 스크립트 : Assets/Scripts/Interactable 폴더 내 모든 스크립트

**게임 내 이벤트**

게임 진행 시 특정 상황에서 특정 게임 이벤트가 발생하는 것을 볼 수 있습니다. 이를 효과적으로 구현하기 위해 게임 이벤트의 발동 조건을 확인하여 게임 이벤트를 실행하는 클래스를 구현하였습니다. 이 클래스들은 공통적으로 **EventHandlerBase** 클래스를 상속하며, 각각 콜라이더의 충돌여부, 퍼즐의 성공 또는 실패, 특정 단서의 획득과 같은 조건을 확인합니다.

이때 발동되는 게임 이벤트들은 중력의 변화, 사운드의 출력, 오브젝트의 이동, 특정 애니메이션의 재생 등 모두 제각각 다른 형태입니다. 이를 이벤트라는 하나의 개념으로 묶기 위해 이런 이벤트들은 **IEvent** 인터페이스를 상속하여 관리합니다.

**IEvent**에서는 이벤트의 발동 시, 중지 시 행동들을 정의할 수 있고, 몇 초 뒤에 해당 동작들을 실행할지 도 조정 가능하게 구현하여 효과적으로 이런 이벤트들을 배치할 수 있었습니다.



다만 이런 모든 이벤트들을 단순히 코드로 정의하는 것은 비효율적인 작업입니다. 예를 들면 스테이지1 첫 번째 중력변화 시 실행되는 게임 이벤트는 단순 코드만으로 자연스러운 움직임들을 구현하기가 어렵습니다. 따라서 **Unity**에서 지원하는 **시네머신**(Cinemachine)과 **타임라인**(Timeline)을 이용하여 하드 코딩이 아닌 드래그앤 드랍으로 짧은 시간내에 화면 연출을 구현하였습니다. 그리고 위에서 언급한 기능들과 함께 사용하여 더욱 효과적인 작업이 가능했습니다.



(시네머신과 타임라인을 같이 사용한 모습)

예외로 각 스테이지의 게임 클리어, 게임 오버 이벤트는 위에서 언급된 두 방법이 아닌 편의상 하드 코딩으로 구현하였습니다. 특정방법만이 아닌 상황에 따라 효율적인 방법으로 게임 이벤트들을 구현하였습니다.

(Assets/Scripts/Stage 폴더 내 Stage1.cs, Stage2.cs 의 GameOver, GameClear 메서드, 코루틴)

(Assets/Scripts/Enemy/MotherEnemyScreen.cs)

\***시네머신**과 **타임라인**의 주요 기능들은 본 보고서에서는 분량문제로 인해 링크로 대처합니다

타임라인 : <https://youtu.be/42QZ3aUDnV0>

시네머신 : <https://youtu.be/x6Q5sKXjZOM>

관련 스크립트 : Assets/Scripts/GameEvent 폴더 내 모든 스크립트

**중력의 변화**

개발 기간 중 중력의 변화를 구현하기위해 여러가지 시행착오를 겪었습니다. 실제로 중력을 변화시키는 것은 개발기간 내에 마무리하기에는 너무 복잡하고 난이도가 높은 작업입니다. 그렇기 때문에 중력을 직접 바꾸는 것이 아닌, 플레이어를 기준으로 맵 자체를 회전시켜 눈속임 연출로 구현하였습니다.

문제는 플레이어가 이동 중에 중력이 바뀌는 식으로 구현한 초기 버전에서 충돌 판정에 대한 버그가 지속적으로 발생하였습니다. 그래서 맵이 회전하는 동안 플레이어와의 충돌을 피하기 위해 중력의 변화를 랜덤이 아닌 지정된 장소와 특정 조건에서만 발동되도록 구성하였고, 플레이어를 공중에 띄워 해당 문제를 해결하였습니다.

중력 변화 이벤트는 위에서 언급된 이벤트들로 컨트롤 하였습니다. 또한 별도의 매니저 클래스를 두어 관리하였습니다.



(위 사진과 같이 실제로는 맵이 회전하게 됩니다.)

관련 스크립트 : Assets/Scripts/GameEvent 폴더 내 GravityEvent.cs

Assets/Scripts/Manager/GravityChangeManager.cs

**사운드**

전체적인 사운드들을 효과적으로 관리하기위해 **AudioManager** 클래스에서는 게임 시작 시 Dictionary 자료형에 <사운드 파일 이름, 사운드 파일> 쌍으로 Resouces/Sounds 폴더에 존재하는 모든 사운드 파일들을 저장합니다. 그리고 필요할 때마다 Key 값으로 Dictionary를 탐색하여 원하는 사운드 파일을 바로 얻어오도록 설계하였습니다.

게임을 진행하다 보면 게임 내 시간을 멈춰야 할 경우가 생깁니다. 이때 주변에서 재생되는 사운드들도 잠시 멈춰야 자연스럽습니다. 하지만 이를 위해 사운드를 출력하는 모든 오브젝트들의 리스트를 저장하는 것은 비효율적인 방법입니다. Angst에서는 C# 델리게이트를 이용하여 간편하게 이것들을 제어합니다. 게임 내 시간의 영향을 받는 오브젝트들의 사운드들은 **SoundDele** 클래스에 의해 관리됩니다.

스테이지2 플레이시 어두운 화면에서 적의 위치를 아는 것은 매우 중요합니다. 플레이어는 라이트의 불빛과 적의 사운드만으로 적의 위치를 피해 다녀야 합니다. 이때 입체 사운드가 잘 구현되어 있어야 플레이어가 더욱 게임에 몰입할 수 있습니다.

기본적으로 **Unity**에서는 **AudioSource** 컴포넌트에 3D 사운드 기능을 제공합니다. 하지만 조금 더 자연스러운 연출을 위해 소리의 근원과 플레이어 사이의 오브젝트들을 고려하여 사운드의 볼륨이나 허용 주파수 범위를 조절하는 **SoundWithWall** 클래스를 구현하였습니다. 이 클래스에서는 **Raycast**를 이용하여 플레이어와 소리의 근원 간의 오브젝트들의 개수를 파악하고, 일정 개수 이상이면 볼륨을 줄이고 **AudioLowPassFilter**를 이용하여 고주파 영역을 차단하는 방법으로 소리의 입체감을 더합니다.

게임의 배경이 되는 장소는 밀폐된 공간입니다. 이를 효과적으로 표현하기 위해 **AudioReverbFilter**를 이용하여 사운드에 리버브(Reverb)효과를 주었습니다.

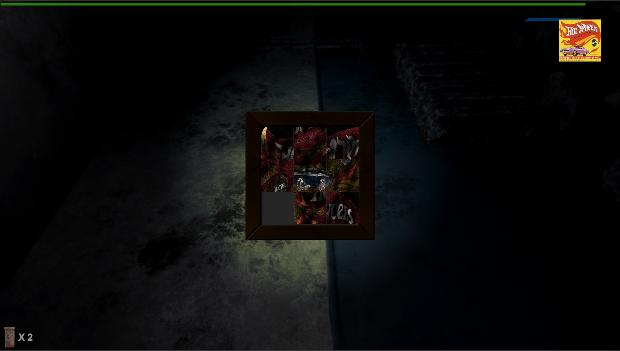
관련 스크립트 : Assets/Scripts/Sound 폴더 내 모든 스크립트

Assets/Scripts/Manager/AudioManager.cs

**슬라이드 퍼즐**

스테이지1에 등장하는 슬라이드 퍼즐입니다. 퍼즐원리는 간단했지만 단순히 무작위로 퍼즐을 배치해서는 안되는 경우가 있었습니다. 슬라이드 퍼즐은 무질서도 라는 것을 고려해야 합니다. 이 무질서도가 짝수일 경우에만 퍼즐이 완성될 수 있습니다.

슬라이드 퍼즐을 푸는 것은 UI에서 이루어 집니다. 슬라이드 퍼즐과 상호작용하면 퍼즐UI가 나오고, 퍼즐을 클릭하여 빈곳으로 밀며 진행합니다. 매 클릭마다 정답을 확인하고, 정답이면 퍼즐 성공 이벤트를, 실패면 퍼즐 실패 이벤트를 실행합니다.



(슬라이드 퍼즐 UI)

슬라이드 퍼즐 무질서도 관련 링크 : https://m.post.naver.com/viewer/postView.nhn?volumeNo=17980703&memberNo=16868720

관련 스크립트 : Assets/Scripts/SlidePuzzle 내 모든 스크립트

Assets/Manager/SlidePuzzleManager.cs

**적 오브젝트 & 길 찾기 AI**

스테이지2에서는 플레이어를 추격하는 적이 존재합니다. 직접 컨트롤하는 플레이어와는 다르게 적 오브젝트는 구현된 AI에 따라 행동합니다.

적 오브젝트는 플레이어와 마찬가지로 FSM의 형태로 구현하였습니다. 다만 사용 애니메이션이 많지 않아 **Unity** **Mecanim** 대신 단순 코드로 구현하였습니다.

적 오브젝트는 시력이 거의 없지만 청력이 민감하다는 설정 하에 디자인되었습니다. 따라서 적의 시야 각 이내이면서 시야를 가리는 장애물이 없고, 일정 거리에 플레이어가 존재할 경우, 소리를 감지하는 영역 이내에서 플레이어가 소리를 낼 경우 추적하도록 구현하였습니다. 플레이어 소리의 감지는 플레이어에 부착된 **AudioSource**들을 확인하여 판단합니다.

적은 기본적으로 정해진 루트를 따라 스테이지를 패트롤(Patrol)하거나 그 자리에서 일정 시간동안 대기(Idle)하기를 반복합니다. 그러다가 위 조건을 만족하는 경우 플레이어를 쫓아가기 시작합니다. 일정 거리 이내 접근하면 플레이어를 공격(Attack)합니다.

스테이지2에서 플레이어는 적을 피해 인형을 모아야 합니다. 게임 밸런스를 위해 플레이어가 획득하는 순간 아이의 울음소리가 재생되고 해당 위치로 적이 이동하도록 구현하였습니다.

**Unity**에서는 **Navigation** 기능을 이용해 길 찾기를 간단하게 구현할 수 있습니다. **NavMeshSurface**를 이용해 몇 가지 조건을 입력해주면 **NavMeshAgent**가 부착된 오브젝트가 길 찾기를 수행할 수 있는 영역을 Bake 할 수 있습니다.

스테이지2 에서는 한 번의 중력변화 이후 적 오브젝트가 활성화되면서 플레이어 추적을 시작합니다. 개발 단계에서는 중력변화가 끝난 시점에서 이동가능한 영역을 Bake하였는데, 이때 필요한 연산에 의해 게임이 잠시 멈추는 현상이 발생하였습니다. 이를 해결하기 위해 게임 시작전에 미리 영역을 계산하는 식으로 구현하였습니다.

(관련 스크립트 : Assets/Scripts/Stage/Stage2.cs)

플레이어가 적에게 대항할 수단을 구현을 하였으나 밸런스 문제로 최종 단계에서 폐기되었습니다.

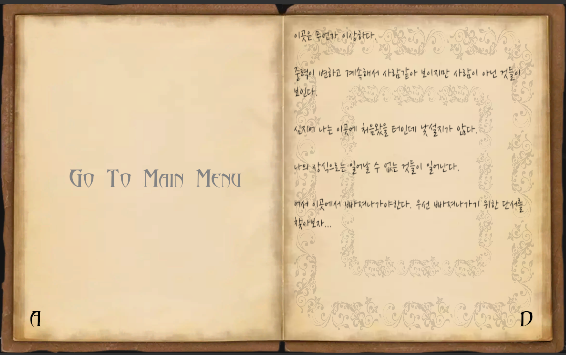
관련 스크립트 : Assets/Scripts/Enemy/MotherEnemy.cs

**Book UI**

플레이어는 획득한 단서를 책에 기록한다는 설정이 있습니다. 이를 위해 얻은 단서의 텍스트를 볼 수 있는 Book UI를 구현하였습니다. 전체적인 디자인이나 동작은 *엘더스크롤 5 : 스카이림(2011)* 에서 많은 참고를 하였습니다.

Book UI의 맨 앞 페이지에는 시작 화면으로 나갈 수 있는 버튼이 있습니다. 그리고 A, D 키를 통해 페이지를 넘기며 획득한 단서를 다시 읽을 수 있도록 하였습니다. 3D 모델을 이용한 *스카이림* 과는 달리 단면 스프라이트를 3D 공간에서 입체적으로 움직이는 식으로 페이지의 넘김을 구현하였습니다. 또한 페이지가 넘어갈 때마다 그 페이지에 맞는 텍스트가 출력되도록 컨트롤 하였습니다.

Book UI를 보는 동안은 게임 내 시간을 멈추어 밸런스를 맞추었습니다.



(Book UI 메뉴와 첫 페이지)

관련 스크립트 : Assets/Scripts/Books 폴더 내 모든 스크립트

Assets/Scripts/Manager/BookUIManager.cs

**자물쇠 해제 UI**

자물쇠 해제시에는 **Unity**의 **UGUI**가 아닌 실제 모델링을 이용해 좀더 실감나게 구현하였습니다. 플레이어가 스테이지 상의 자물쇠 오브젝트와 상호작용하면 다음 이미지와 같은 실제 모델링이 등장합니다.

이를 구현하기 위해 별도의 **Camera**를 두어 상호작용 시 에만 해당 **Camera**를 활성화해 화면에 겹쳐 보이도록 하였습니다. 또한 콜라이더를 설정하여 마우스 클릭으로 각각의 번호들을 설정할 수 있게 하였습니다. 자물쇠가 풀리면 UI가 자동으로 꺼지고, 스테이지에 존재하는 해당 자물쇠 오브젝트는 비활성화 됩니다.



(자물쇠 해제 UI)

관련 스크립트 : Assets/Scripts/Lock 폴더 내 모든 스크립트

Assets/Scripts/Manager/LockManager.cs

**데이터의 저장**

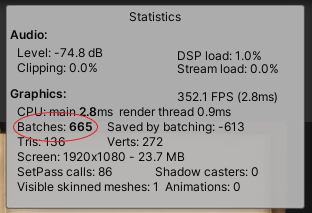
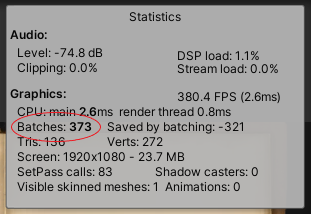
**Unity**의 **PlayerPrefs** 클래스를 이용해 스테이지의 클리어 여부만 저장합니다. 스테이지의 볼륨이 크지 않기 때문에 각 스테이지 내에서의 진행상황은 저장하지 않았습니다.

관련 스크립트 : Assets/Scripts/Manager/SaveManager.cs

**최적화 & 마무리**

**드로우 콜과 수동 배칭**

최적화 단계에서는 드로우 콜(DC)을 줄이는 것에 집중하였습니다. 배칭(Batching)이란 여러 드로우 콜이 필요한 상황을 하나의 드로우 콜로 묶는 과정을 의미합니다. **Unity**에서 **스태틱 배칭(static batching)**과 **다이나믹 배칭(dynamic batching)** 기능을 제공하지만, 전체 맵이 회전하는 특징 상 움직이지 않는 오브젝트 들에만 적용 가능한 스태틱 배칭은 불가능 하였고, 다이나믹 배칭만으로는 원하는 만큼 드로우 콜이 줄지 않았습니다. 해결 방안을 찾던 중 ‘유니티 그래픽스 최적화 스타트업’ 이라는 책에서 원하는 정보를 찾아 프로젝트에 적용하게 되었는데 그것이 바로 코드상에서 수동으로 배칭하는 방법이었습니다. 같은 매터리얼(Material)을 사용하는 서로 다른 매시(Mesh)들을 하나의 매시로 합쳐 드로우콜을 줄이는 방식입니다. 주로 반복되는 벽, 바닥, 장식 등에 적용하였습니다. 다음의 사진과 같이 수동 배칭 적용 후 스테이지1 에서 약 300개의 드로우콜이 준 것을 확인하였습니다.

(수동 배칭 전과 후)

관련 스크립트 : Assets/Scripts/Batching 폴더 내 모든 스크립트

**개선 사항 및 소감**

개선이 필요한 사항은 크게 두가지가 있습니다.

첫 번째는 어떤 물체 위로 올라갈 수 있는지 없는지 감지할 때 판단 기준이 모호한 점 입니다.

게임의 특징인 중력이 바뀌는 설정 덕분에 어떤 상황에서는 올라갈 수 없는 곳이지만, 다른 상황에서는 올라갈 수 있는 곳이 되는 경우가 있었습니다. 그럴 때마다 다른 레이어로 구분하여 혼선을 최소화했지만 그럼에도 불구하고 원하는 감지가 이루어지지 않는 경우도 있어서 이 부분을 100% 완벽하게 해결하지 못한 것이 아쉽습니다.

두 번째는 전체적인 게임 기획입니다. 스토리, 연출 같은 부분들은 단순히 개발 능력이 아닌 기획 능력도 필요로 하는 부분이어서 많이 힘들었던 부분입니다. 엔지니어의 영역과 어떤 부분은 아티스트의 영역도 겸하는 부분이 있어 제약 사항이 많았습니다. 다른 게임들을 참고하고, 게임이 아닌 다른 매체들도 조사하였지만 원하는 만큼의 수준에 도달하지 못해서 아쉬웠습니다.

3D 공포게임은 최근 게임의 동향을 봤을 때 메이저한 장르는 아니지만 공포게임만의 특색이 있습니다. 공포라는 감정이 좋은 감정은 아니지만 게임 속에서 새로운 경험을 해보는 것은 전체적인 게임을 바라보는 시야가 넓어지는데 큰 도움이 되었습니다.

한 학기 동안 고생한 팀원들과 교수님, 조교님께 감사합니다.