

PORTFOLIO

전 홍 현 HongHyeon Jeon



GitHub



010-3255-0364



toinbee3@gmail.com



PROJECT



좀비평평화르륵

기업협약 프로젝트

Unity 2D
퍼즐 | 캐주얼

2024. 11. ~
2024. 12.
(7 주)

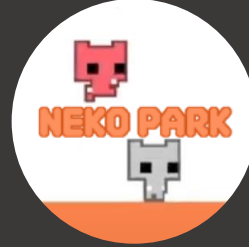


SORI

2인 팀 프로젝트

Unity 3D
퍼즐 | 어드벤처

2024. 08. 28. ~
2024. 10. 04.
(4 주)



NEKO PARK

4인 토이 프로젝트

Unity 2D
퍼즐 | 협동

2024. 08. 12. ~
2024. 08. 19.
(1 주)



The Wild Four

4인 팀 프로젝트

Unity 3D
생존 | 어드벤처

2024. 07. 08. ~
2024. 07. 31.
(3 주)



Terraria

개인 프로젝트

Unity 2D
어드벤처 | 샌드박스

2024. 06. 13. ~
2024. 06. 24.
(1 주)

Overview



좀비평평화르륵

Unity 2D 기업협약 Project

게임 장르 : 퍼즐 / 캐주얼

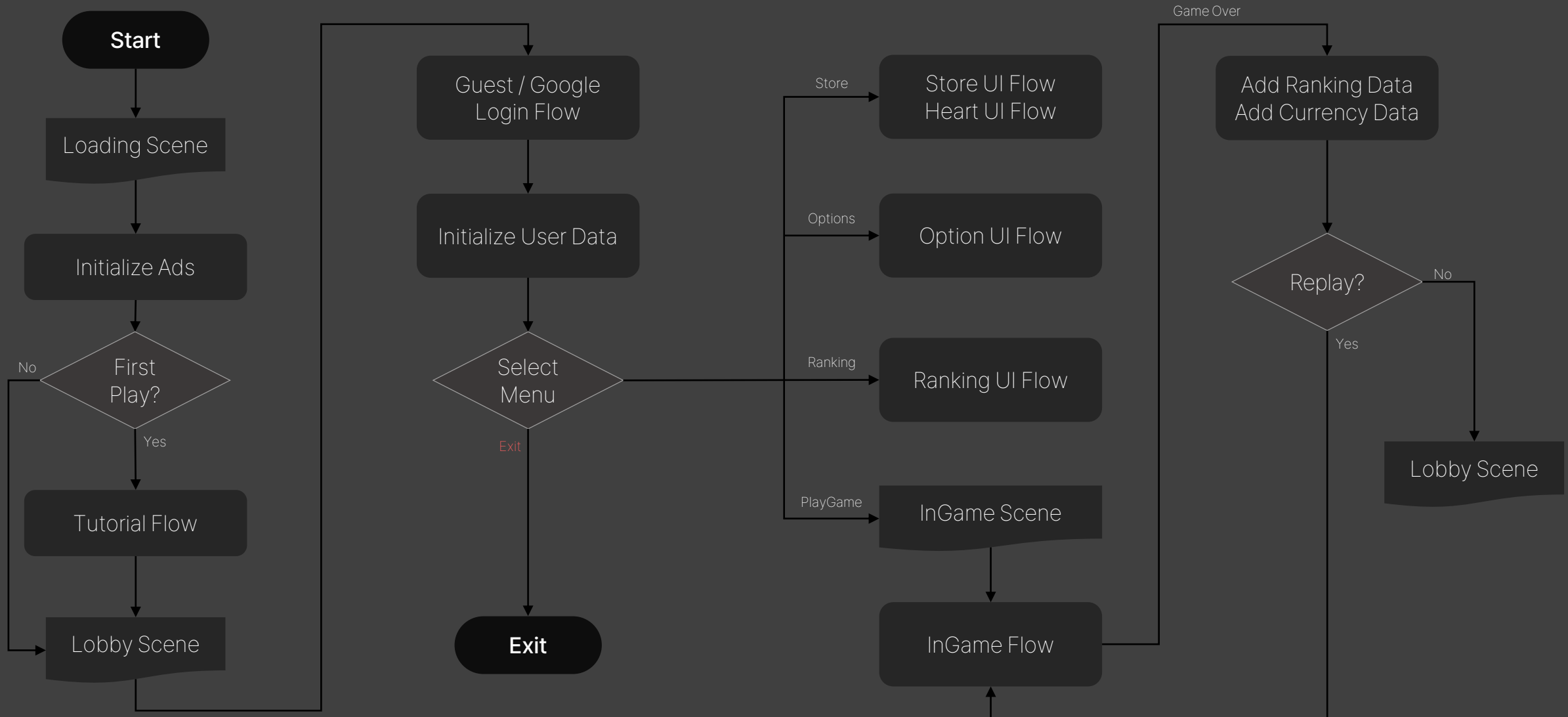
개발 인원 : 기획 3 명, 개발 4 명

개발 환경 : Unity 2022.3.50 LTS

개발 기간 : 2024.11. ~ 2024.12. (7 주)

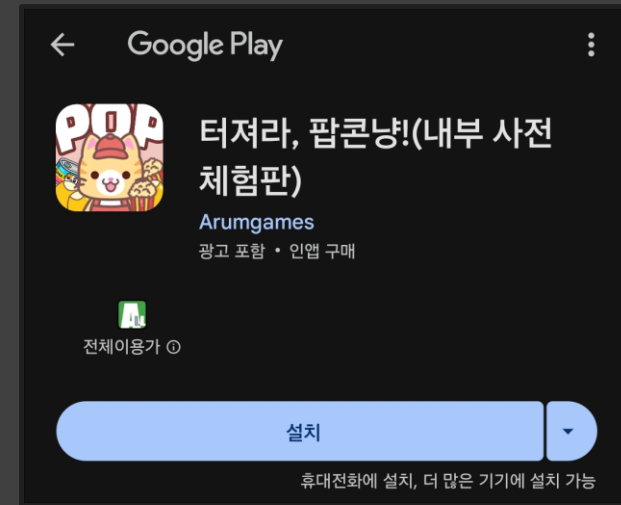


FlowChart



Features

□ GPGS Plugin



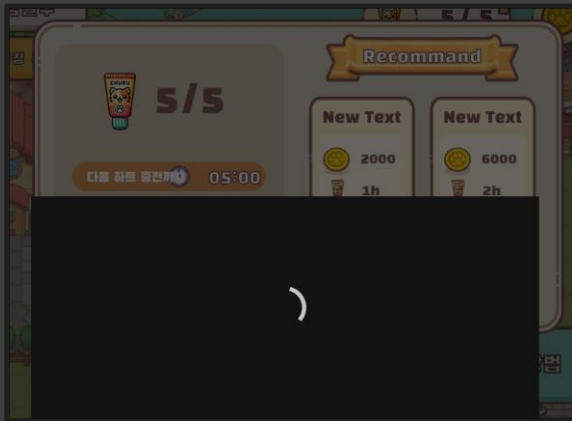
- GPGS Plugin을 연동하여 Google 계정을 이용한 PlayGames Login 기능을 구현.
- Google Login 성공 시 userID, userEmail, UUID 데이터를 DB에 저장, 게임 내의 Purchase Data, Ranking Data 등을 DB와 연동할 수 있도록 처리.
- 이를 기반으로 플레이어는 각 로컬 기기 간 데이터를 연동할 수 있음.
- GPGS Setup을 위해 Google PlayConsole에 앱을 등록하는 과정을 진행함.

(사진은 내부 리소스 변경 전 버전이며 현재 빌드 버전과 다를 수 있음)

Features

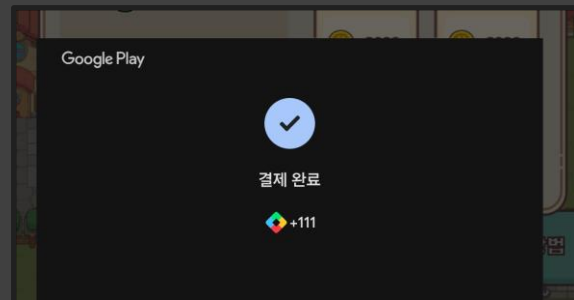
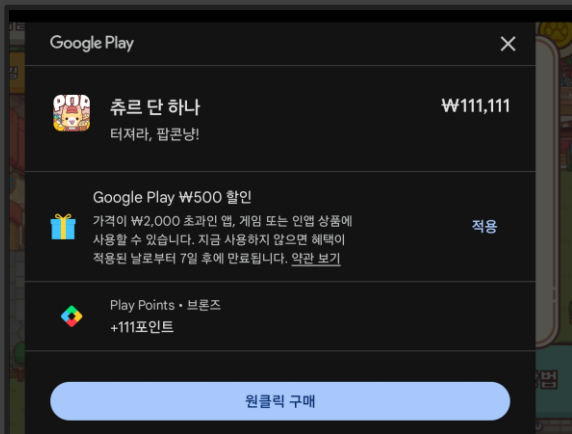


□ Unity IAP



- Unity In-App Purchasing Package 를 사용하여 Google 인앱 결제를 구현.
- Unity IAP를 사용하면 많은 Platform의 앱 스토어를 쉽게 연동할 수 있음.

- 인앱 결제를 위해서 PlayConsole에서 인앱 상품을 등록하고, IAP Catalog Setting 과정을 진행.
- Script에서 IAP를 Initialize할 때 ConfigurationBuilder의 addProduct를 API를 통해 DB에서 받아온 값으로 처리하여 유연한 상품 관리 및 유지보수가 간편하도록 구현.



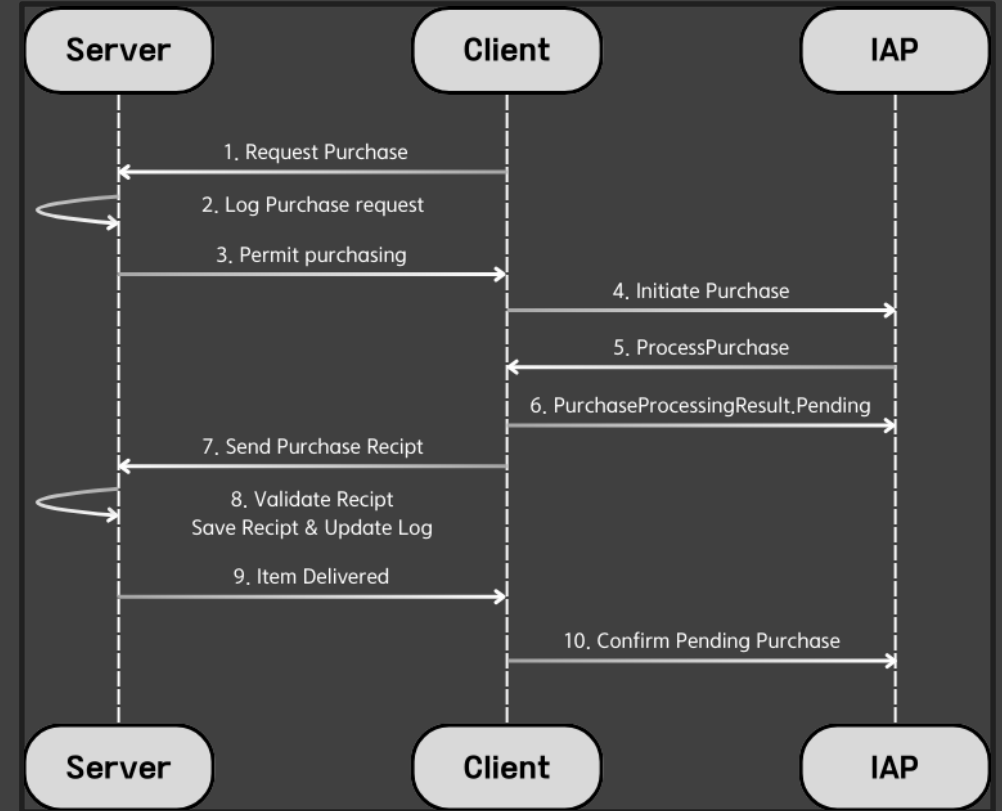
- 구매 완료 시, Receipt Validation 과정을 Client와 BackEnd Server 양 측에서 같이 진행하여, 구매 요청의 무결성을 검증하고 결제 관련 보안을 강화.

(사진은 내부 리소스 변경 전 버전이며 현재 빌드 버전과 다를 수 있음)

Features

□ UnityWebRequest

- GPGS Login, InApp Purchase, Ranking 등 DB와 연동해야 하는 데이터를 Server로 전송하기 위해 UnityWebRequest Library를 사용한 API Manager를 구현.
- API Manager는 각 Component가 Server와 통신이 필요할 때, RestAPI를 활용하여 GET, POST Request를 처리.
- Login Data, Ranking Data, Receipt 등 DB에 저장할 Data는 POST의 Body에 JSON format으로 넣어 전달하도록 처리.
- 이 과정에서 Newtonsoft.Json을 활용한 Body Data 직렬화 구현.
- Coroutine으로 응답을 기다리기 때문에 OnResponse, OnFailed Event를 정의 및 사용하여 게임 진행이 정지되지 않도록 처리.
- 한 Request의 Response가 들어오기 전에 다른 Request가 발생하는 동시성 문제를 방지하기 위해 Pessimistic Lock 방식을 적용.
- TimeOut 및 Connection Failure 예외 처리를 위해 재시도 로직과 네트워크 연결 상태 확인 로직을 구현.

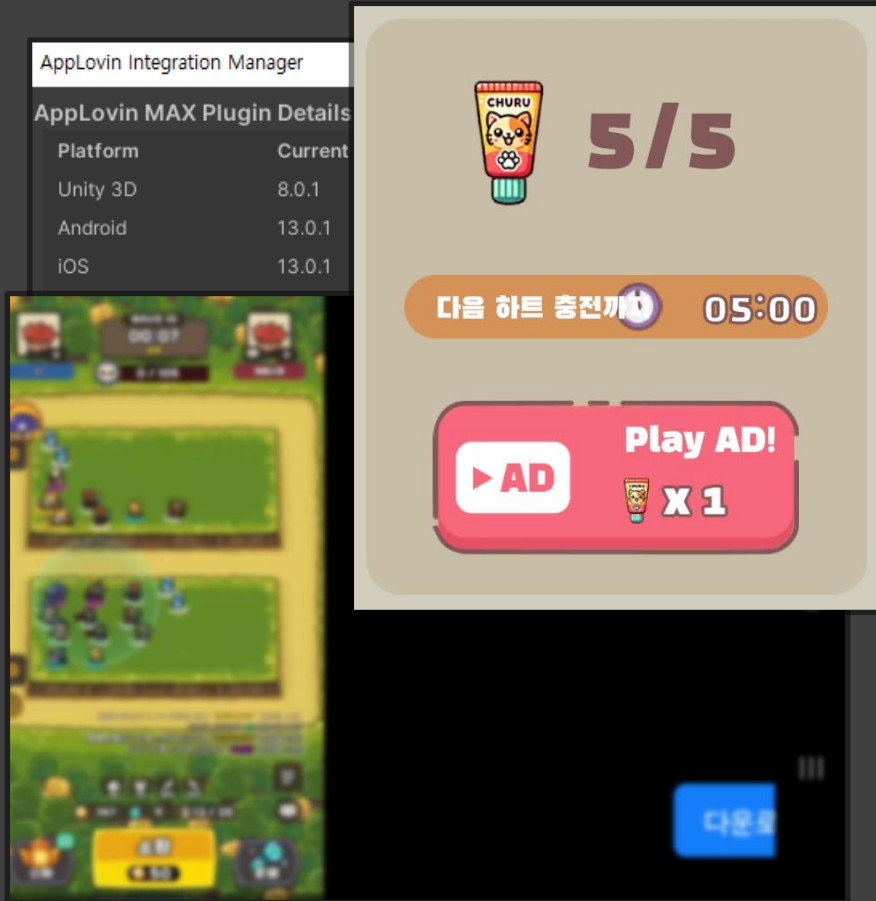


IAP Purchasing - Validate Receipt Flow
Server-Client 통신은 API Manager를 통해 처리

Features



□ Ad Mediation : Applovin Max SDK



- 인앱 광고를 게재하기 위해 광고 Mediation Platform인 Applovin Max SDK를 게임에 연동 및 광고 게재를 구현.
- Google Admob, Applovin Max와 같은 Mediation Platform은 다양한 광고 네트워크를 연결해줘, 쉽고 효율적으로 광고 수익화를 실현 가능.
- Applovin Max SDK 를 연동하기 위해 Applovin Dashboard에서 Ad Unit을 생성 및 설정. 또한 Unity Ads Adapter를 연동하기 위해 Unity Ads Monetization 설정.
- 게임 내 전면 광고(Interstitial), 리워드 광고, 배너 광고를 게재하기 위해 각 광고마다 Ad Unit ID 발급 처리 및 초기화 Script를 작성하여 수익화 테스트를 진행함.

(사진은 내부 리소스 변경 전 버전이며 현재 빌드 버전과 다를 수 있음)

Features

□ Block Manager

- 인게임에서 사용되는 모든 Block을 일괄적으로 초기화, 생성 및 관리하기 위해 BlockManager Script를 작성.
- 각 Block은 Script 외부에 JSON으로 초기 데이터가 지정되어 있으며, 게임이 시작될 때 BlockManager가 JSON을 읽어와 Block을 초기화.
- 또한 Block이 생성되어야 할 때 Unity에서 제공하는 Library인 UnityEngine.Pool을 활용하여 Block을 Pooling 하도록 구현.
- UnityEngine.Pool은 Pooling Object의 Create, Get, Release를 Generic으로 간편하게 구현할 수 있어 Script를 간결하게 작성할 수 있음.
- 그 외 Block 생성 시 Mass 값 관리, 상호작용 시 무적 부여, 공통 속성 관리 로직 등을 처리하도록 구현함.



Features



□ Block Editor : External JSON Editor



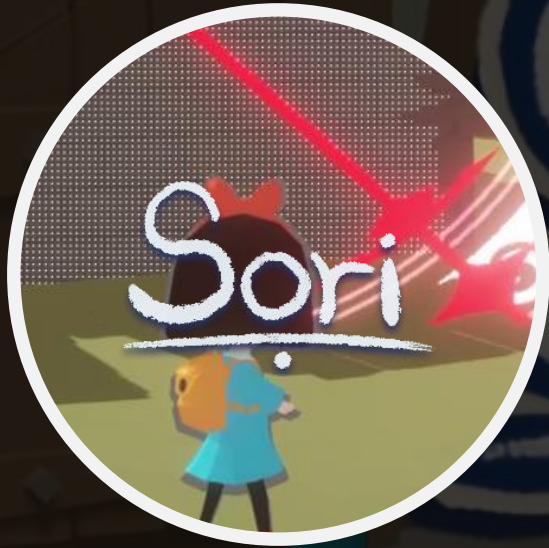
- 게임에 사용된 변동될 수 있는 모든 수치 값은 JSON Data로 Script 외부에 저장되었다가, 게임이 시작될 때 불러올 수 있도록 처리.
- 본 프로젝트에서 기획팀이 Balancing 조정을 담당했는데, 직접 JSON 파일을 열어 수정하는 방법 외에 쉽고 간편하게 수치 값을 조정할 수 있도록 외부 JSON Editor를 제작.

- 각 JSON Data를 편집할 수 있는 탭을 제작하고, InputField를 통해 Type이 지정된 값을 받아들이어서 오타자 또는 형 지정 오류 등 사소한 실수에도 Parsing할 때 오류를 일으키는 JSON Format의 취약점을 보완할 수 있음.

- 또한 특정 Data는 Editor에서 인게임과 동일한 환경을 구축하여, 각 수치 값을 조정할 때 즉시 직관적으로 테스트할 수 있도록 구현.
- 이 방법을 통해 Balancing 작업의 효율성을 대폭 향상시킬 뿐만 아니라 안정성 또한 보장할 수 있었음.



Overview



Sori

Unity 3D Project

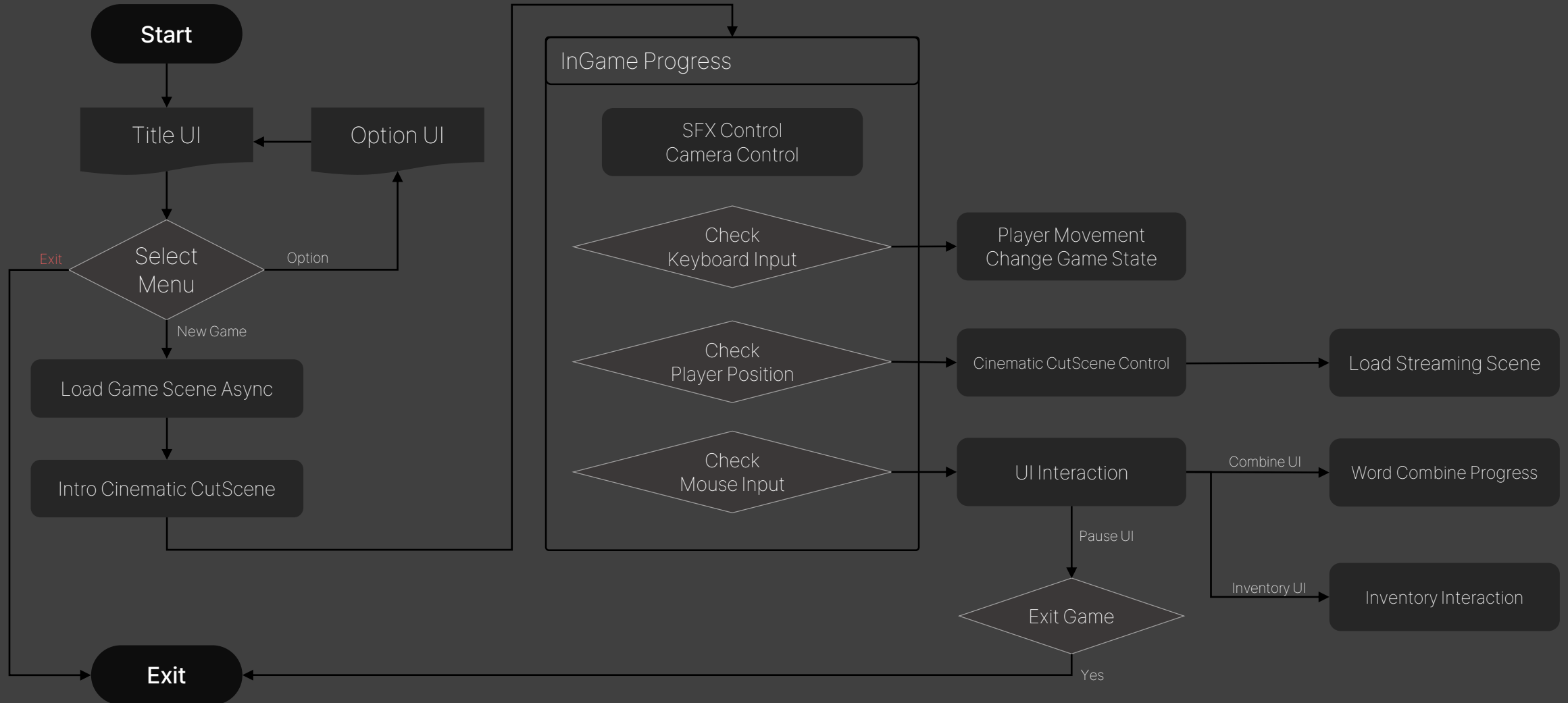
게임 장르 : 퍼즐 / 어드벤처

개발 인원 : 개발 2 명

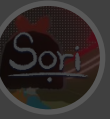
개발 환경 : Unity 2020.3.36 LTS

개발 기간 : 2024. 08. 28. ~ 2024. 10. 04. (4 주)

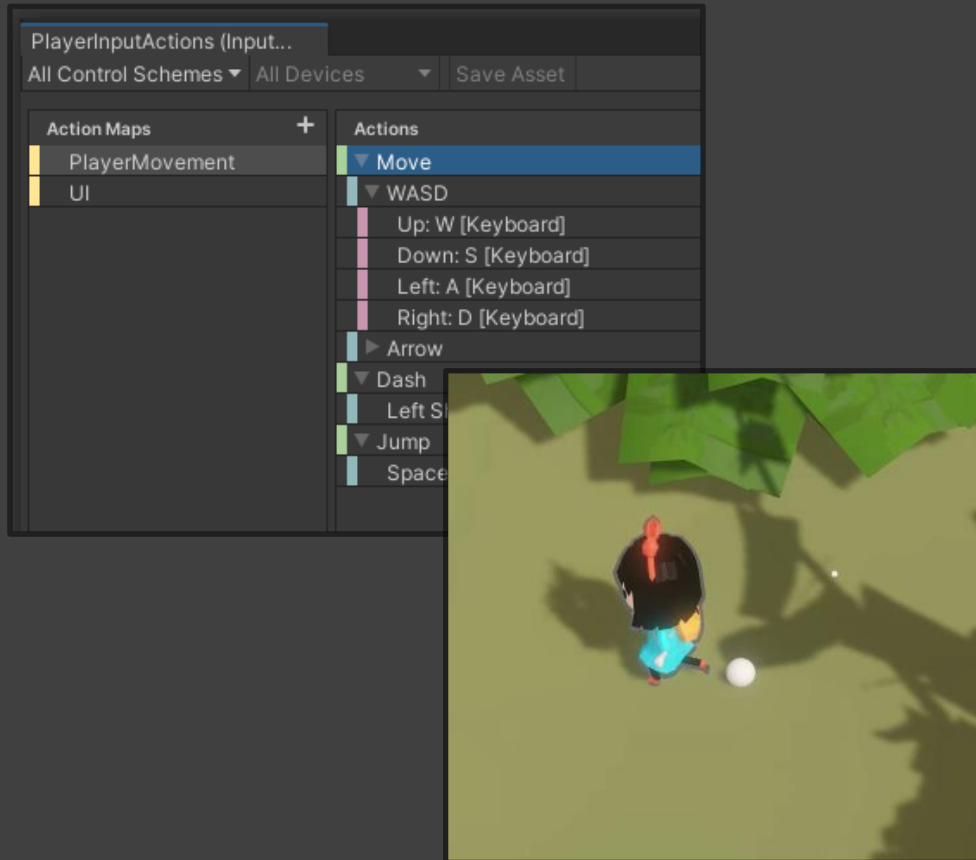
FlowChart



Features

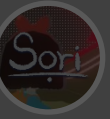


□ New Input System



- 입력을 처리하는 방법으로 **New Input System**을 사용.
- New Input System은 다양한 디바이스 및 플랫폼에 맞춰서 키 매핑을 구성할 수 있어 일일이 키를 지정해줘야 하는 기존의 Input Manager에 비해 유연성이 뛰어남.
- 매 Frame마다 입력을 검사해야 하는 Input Manager보다 입력 발생 시 Event Callback으로 처리할 수 있어 Script 복잡성이 줄어들고 불필요한 연산을 최적화할 수 있음.
- Input Actions의 C# Class를 생성하고, 각각의 키가 performed 될 때 수행되어야 하는 Method를 Event로 등록하여 입력을 효율적으로 처리.

Features

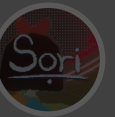


□ Player Movement

- `InputAction.CallbackContext` 로 전달되는 입력 값을 `Vector2` Type으로 읽어들이어 플레이어의 움직임을 처리.
- 플레이어가 매끄럽게 움직이도록 하기 위해, 입력을 지속한 시간에 따라 속도가 부드럽게 변하도록 구현.
- 현재 `Position`에 (입력된 방향 값 * 현재 속도) 를 더한 `Vector3` 값을 `targetPosition`으로 설정한 뒤, `Rigidbody`의 `MovePosition` Method를 사용해 움직임을 처리.
- 플레이어의 `Jump` 초기화를 위해 `Ground Layer`와 충돌할 때 발생하는 `OnCollisionEnter` Event로 처리.
- 또한 지형 외의 다른 물체 위에 올라가는 경우에도 `Jump` 초기화가 필요하기에 `Collision Callback`에서 충돌 `Collider`의 상단에 올라갔는지 `collider.bounds`를 연산하여 검출하는 방식 및 플레이어의 하단으로 `BoxCast`를 쏘서 바닥을 감지하도록 복합적으로 구현.
- `BoxCast`를 사용할 경우 `RayCast`를 사용하는 것보다 넓은 범위를 감지하므로 보다 정확하게 바닥을 검출 가능.



Features

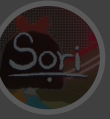


□ Player Movement

- Dash 또한 증감률의 영향을 받아 가속하도록 하기 위해, 현재 속도를 즉시 Dash 속도로 설정하는 것이 아닌 현재 속도의 상한치를 Dash 속도만큼 높이는 방식으로 Dash를 구현.
- 자연스럽게 떨어지는 Jump 모션을 위해 Project Settings 에서 Physics Gravity 값을 조절.
- 플레이어가 빠른 속도로 이동할 때 얇은 벽을 관통하는 이슈를 해결하고자 플레이어의 forward 방향으로 Raycast를 쏘아서 플레이어가 전방의 벽을 관통하려고 하는 순간을 검출 및 충돌 애니메이션을 실행하도록 구현.
- 또한 속도가 빠를 때 과도한 경사를 비정상적으로 올라갈 수 있는 이슈를 방지하기 위해 플레이어의 전방에서 하단으로 RayCast를 쏘아서 RayHit의 normal 이 일정 각도 이상일 때 더 이상 이동할 수 없도록 구현.
- Cinematic Cut Scene이 재생 중일 때는 플레이어의 입력이 불가능하므로, 해당 상황에서 플레이어의 움직임을 연출하기 위해 DoTween을 사용.
- DoTween을 사용할 경우 매 Frame마다 연산을 해야 하는 Lerp 보다 복잡성이 줄어들고 이동하는 데 걸리는 시간을 지정할 수 있어 Cinematic 연출과 연동하기에 용이함.



Features



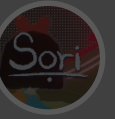
□ Particle System



- 게임 내 시각적 연출을 위해 여러 종류의 Particle System을 사용.
- 작은 Object를 대량으로 사용하는 것 보다 Particle System을 사용하면 Rendering 성능을 향상시킬 수 있으며 직관적인 Interface를 가져 효과 연출에 용이함.
- 플레이어가 이동할 때 이동 속도에 따라 Emission의 Rate over Time을 비례하도록 하여 RunningDust Particle을 생성.
- Simulation Space를 World로 설정하고 Gravity 값을 음수로 설정하여 가볍게 날아가는 먼지 효과를 연출함.



Features



□ Cinemachine Camera

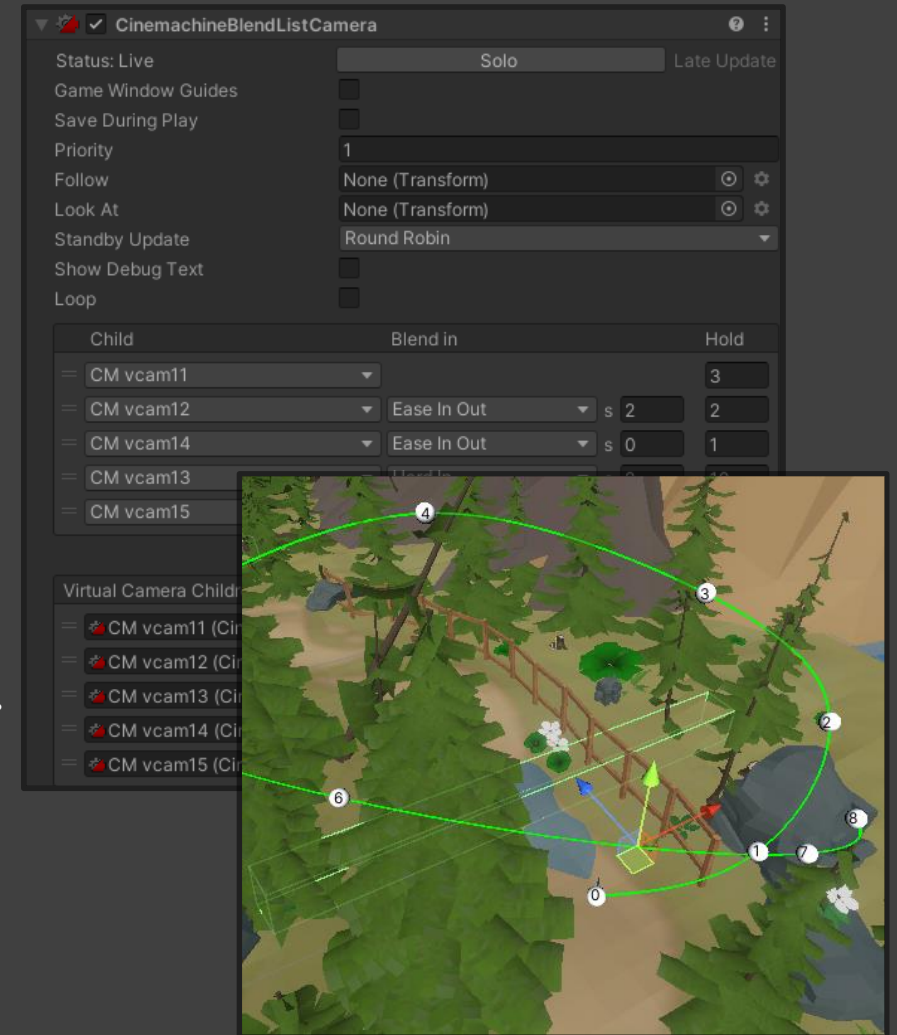


- 게임 내 다양한 카메라 연출을 위해 Cinemachine 을 활용.
- Cinemachine은 가상 카메라를 사용하여 카메라의 이동, 추적, Blending 등 카메라 기능을 제공하기 때문에, 직접 카메라 처리 Logic을 작성하지 않아도 손쉽게 카메라 연출을 구현할 수 있음.
- 게임에서 사용되는 각 카메라 시점마다 Virtual Camera를 사용하고, CameraControl Script를 작성해 시점 변경을 일괄적으로 관리하도록 구현.
- Cinemachine에서 기본으로 제공되는 Perlin Noise를 사용해 HandHeld Camera 연출을 할 수 있도록 처리.
- Mouse Scroll 입력 시 부드러운 Zoom In, Zoom Out을 구현하기 위해, Transposer의 FollowOffset을 Lerp 연산을 통해 조정.
- 마지막으로 Scroll한 시간을 저장하여 일정 시간이 지나면 기본 Zoom 상태로 돌아오도록 구현.

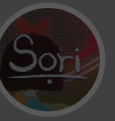
Features

□ Cinemachine Camera : CutScene

- 게임 내 Cinematic CutScene은 모두 Cinemachine Camera를 사용하여 연출.
- Cinemachine BlendList Camera를 사용하면 여러 카메라 사이의 전환을 부드럽고 쉽게 처리할 수 있음.
- Virtual Camera로 CutScene의 각 시점을 설정한 뒤, BlendList Camera에 등록하여 카메라 연출이 자연스럽게 이어지도록 구현.
- 일부 시점은 DollyTrack 을 사용하여 미리 설정해둔 경로를 따라 카메라가 이동하며 CutScene을 연출하도록 처리.
- 맵의 일정 구간마다 Trigger Collider를 배치하여 플레이어가 해당 구간에 진입하면 자동으로 CutScene이 연출되도록 Logic을 작성.

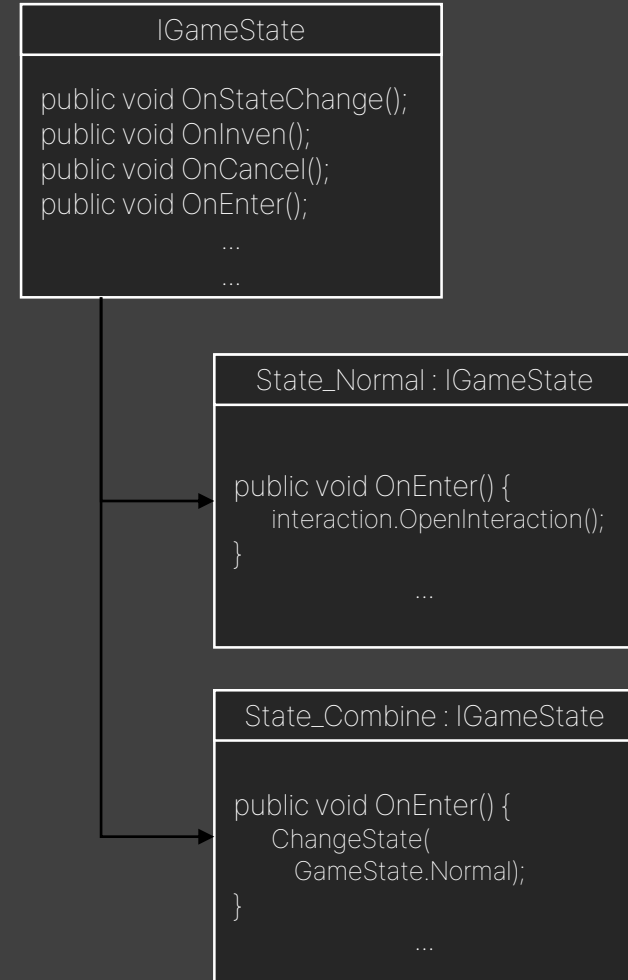


Features

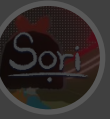


□ Game State : State Pattern

- 특정 State마다 동일한 키 입력을 다르게 처리해야 할 때가 있는데, 이를 위해 모든 Game State를 State Pattern 으로 작성.
- 입력 종류마다 처리해야 할 필수 Method를 Interface로 정의하여, 입력이 발생했을 때 반드시 처리해야 할 Logic을 각 State가 구현하도록 함.
- State가 변경될 때 OnStateChange() Method를 반드시 호출하도록 설계하여, 카메라 시점 변경 등 State마다 필요한 작업을 처리하도록 구현.
- GameManager에서 Dictionary를 선언하고, Enum 값을 키로 하여 IGameState Interface 형을 저장함으로써 각 State를 Enum 값에 매핑하여 캐싱하도록 구현.
- 각 State를 미리 캐싱하면 State 객체를 재사용할 수 있어 GC 호출 부담을 줄이고, State를 전환할 때 새롭게 객체를 생성하지 않아 메모리 사용량을 최적화할 수 있음.

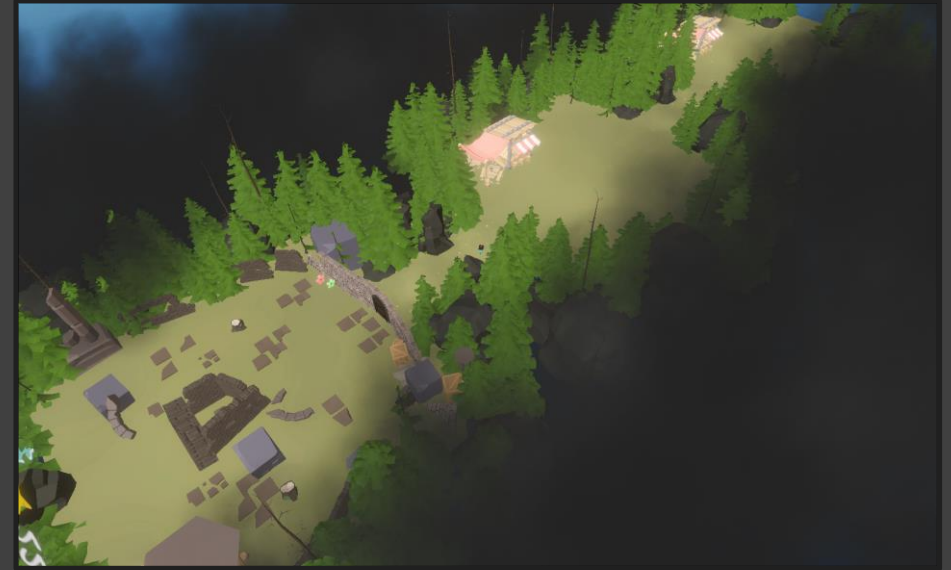


Features

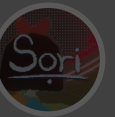


□ Streaming Level

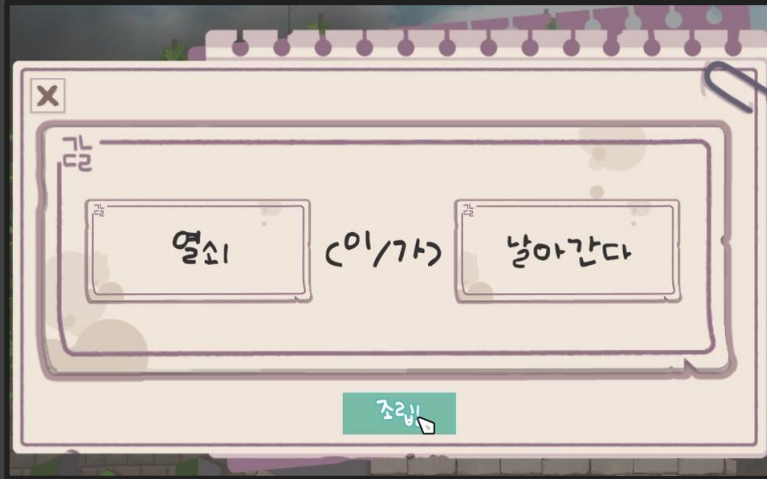
- 게임의 각 Stage는 개별적인 Scene으로 구성되어 있음.
- Stage로 진입할 때, 모든 Stage를 한 번에 Load 하는 것은 시간도 오래 걸리고 메모리 점유율이 비효율적으로 증가함.
- 이를 해결하기 위해 초기 Stage만 Load 한 뒤, 다음 Stage로 진행하면 그 때 Scene을 Load하도록 구현
- Stage의 출구 부분에 Trigger Collider를 배치하고, 플레이어가 이를 지나갈 경우 그 다음 Stage Scene 이 LoadSceneAsync() Method로 비동기적인 Load가 되도록 구현.
- 이 때 현재 Stage가 바로 사라지고 다음 Stage로 즉각 교체되면 어색하게 느껴질 수 있으므로 LoadSceneMode를 Additive로 하여 현재 Load되어 있는 Stage가 사라지지 않고 다음 Stage가 추가로 Load 되도록 처리.
- 현재 Load 되어 있는 Stage의 개수를 StageLoadManager Script가 저장하고 있다가, 일정 이상이 되면 먼저 Load 된 Stage 부터 Unload 하여 메모리 점유율이 계속 증가하지 않도록 관리.



Features

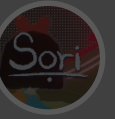


□ Word Combine Logic

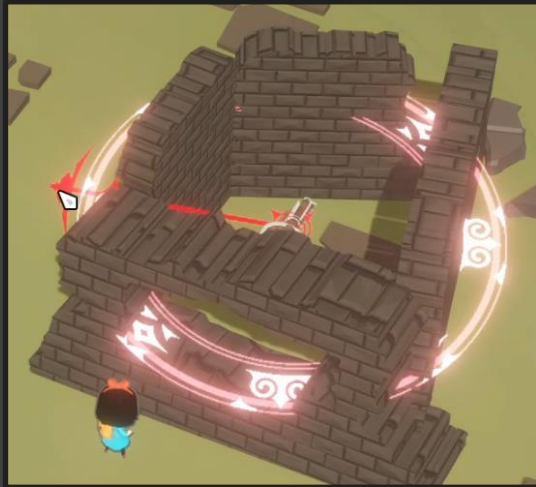


- 랜덤으로 획득할 수 있는 단어 카드를 모아 문장을 구성하여, 문장의 효과를 통해 퍼즐을 해결하는 것이 게임의 핵심 요소임.
- 이 기능을 구현하기 위해서는 조합하려는 문장이 실제로 유효한 조합의 문장인지 검사해야 하며, 조합된 문장이 의미에 맞는 적절한 효과를 발동하도록 처리해야 함.
- 문장들은 Frame Class, 단어 카드는 Word Class로 정의.
- Word는 명사인지, 동사인지 구분하는 WordType 및 각각의 고유한 WordTag를 가짐.
- 또한 동사 Word는 자신의 효과에 적용 대상이 되는 명사 Word의 WordTag를 저장하는 배열을 가짐.
- Combine UI 에서 Frame에 Word를 올리고 조립 Button을 누르면 FrameValidity Script가 Frame의 유효성을 검사.
- Frame Type에 따른 명사-동사 조합을 판별한 뒤, 명사가 동사의 효과 적용 대상인지 확인하기 위해 동사 Word가 가진 WordTag 배열을 참조.

Features

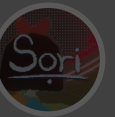


□ Word Combine Activation

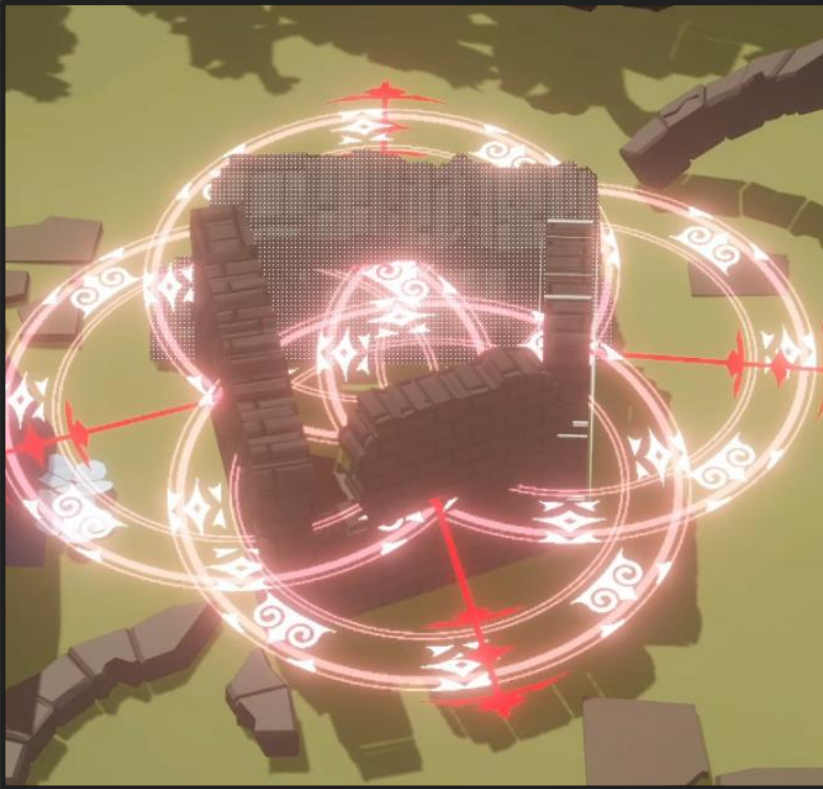


- FrameValidity Script에서 유효성 검사가 통과할 경우, FrameActivate Script에서 동사의 종류에 따른 문장 기능을 발동.
- 발동 대상이 될 Object를 선택하기 위해 Select State로 전환되며, 명사의 WordTag와 동일한 Tag를 가진 Object를 선택할 경우 발동 대상으로 지정할 수 있음.
- SelectControl Script는 Select State 동안 선택된 Object를 저장하고 관리. 현재 Mouse Over 된 Object에 Outline Material을 적용하고 선택된 Object에 Dot Material을 적용함.
- Select State에서 Enter 키가 입력되면 Selected Object List를 FrameActivate Script로 순차적으로 전달.
- FrameActivate Script는 발동 중인 Frame의 Word를 조사하여, 현재 선택된 Object가 문장 효과의 대상일 경우 Object에 동사의 효과를 적용.

Features

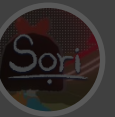


□ Indicator Control



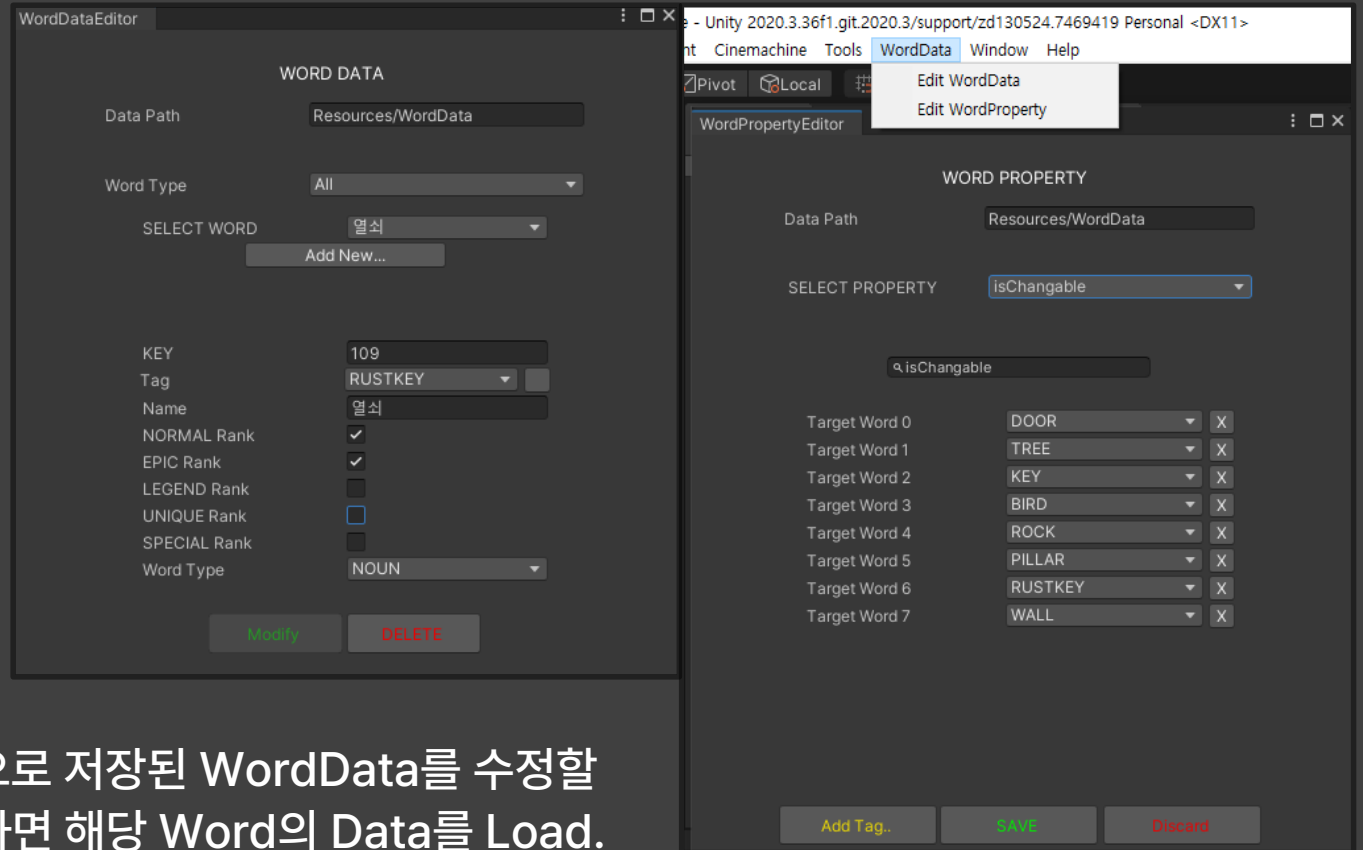
- 동사의 효과가 Object를 움직이게 하는 종류일 경우, Indicator를 표시해 플레이어가 Object를 움직이게 할 방향을 지정.
- 각 Indicator는 현재 활성화 된 카메라를 바라보는 동시에 자체적으로 시계 방향으로 회전하며, Arrow는 마우스를 바라보는 방향으로 회전.
- 방향 선택이 완료되면 Arrow는 Indicator의 회전과 무관하게 회전하지 않고 지정된 방향을 계속 가르쳐야 함.
- 이를 구현하기 위해, Quaternion.LookRotation()으로 Indicator가 카메라를 바라보는 각도 값과, Quaternion.AngleAxis()으로 Indicator의 Transform Local Y축을 기준으로 하는 회전 각도 값을 곱해서 Indicator의 rotation을 지정.
- Arrow는 마우스 포인터 위치에 RayCast를 쏘아서 현재 Transform의 position 값과 연산하여 각도를 구해 마우스가 가르키는 방향을 지정.
- 방향 선택이 완료될 경우 Indicator가 Y축 기준 회전하는 값을 구해 그만큼 역방향으로 회전하여 회전을 상쇄하도록 구현.

Features



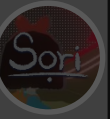
□ Custom Editor Window

- 모든 Word Data 는 Resource 폴더 내에 JSON Format으로 저장되었다가, 게임이 시작할 때 Load 됨. 따라서 Script에 Word Data가 저장되어 있지 않음.
- 외부 JSON 데이터 관리를 쉽게 처리하고자 Custom Editor Window를 작성.
- Custom Editor를 사용할 경우 구현할 기능을 직관적이고 편리하게 구성할 수 있어 개발하는데 편의성을 갖출 수 있음.



- WordDataEditor Window에서는 JSON으로 저장된 WordData를 수정할 수 있음. Dropdown Menu에서 Word를 선택하면 해당 Word의 Data를 Load.
- Load된 Data를 삭제할 수 있고, 필드 값이 하나라도 변경되었다면 변경된 값으로 원본 데이터를 수정할 수 있음.
- WordPropertyEditor Window에서는 동사 Word의 효과 적용 대상이 될 WordTag를 추가 또는 삭제할 수 있음.

Features



□ Light Map / Light Probes

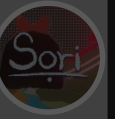


- 가로등과 같이 여러 개의 Light를 처리할 때, Realtime Light로 처리할 경우 무겁고 연산이 많아져 성능에 부담을 발생시킬 수 있음.
- 따라서 Light Map을 활용하여 가로등 빛을 Baked Light로 처리함.
- Light Map을 사용할 경우 Light 정보를 미리 연산해 Texture로 Baking 하여, 효율적으로 Rendering 할 수 있음.
- Baking 작업의 시간을 단축하기 위해 반드시 Light 영향을 받아야 하는 정적 Object만 Static으로 설정하여 작업 속도와 Texture 크기를 효율적으로 줄임.

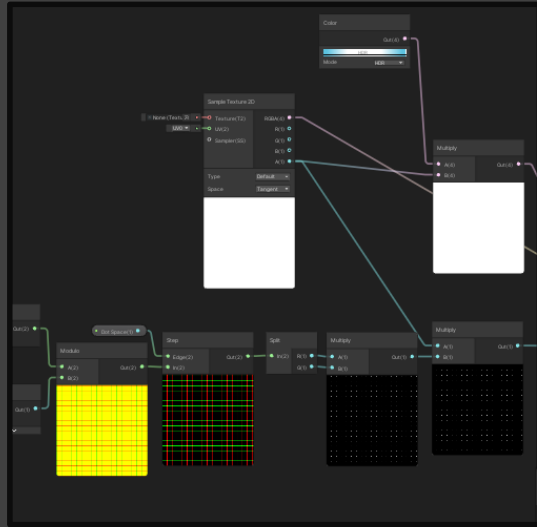


- Light Map을 사용할 경우 동적 Object에 반사광을 표현할 수 없는 단점.
- 이 단점을 해결하기 위해 Light Probes를 활용.
- Light Probes를 사용하면 Realtime Light에 비해서 가볍고, 범위를 지정하여 해당 구역에서만 반사광을 표현하도록 구현할 수 있음.

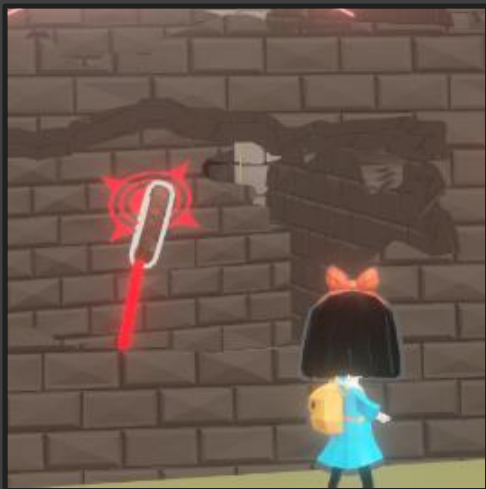
Features



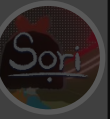
□ Shader Graph : Custom Shader



- 다양한 시각적 연출을 위해 Shader Graph를 활용한 효과를 구현.
- 일정 값 증폭한 Object Scale과 Normal Vector 를 Multiply 하여 Selected Object의 Outline 효과를 구현.
- Object Position을 바탕으로 Noise 값을 Multiply 한 뒤 Position으로 적용시키는 연산을 통해 GuideLight Particle의 공중에 떠다니는 효과를 구현.
- Half-Lambert 조명 공식을 사용해 플레이어 캐릭터의 음영 값을 단순히 조절하여 Cartoon Rendering 효과를 구현.



Features



□ Post Processing / Shader

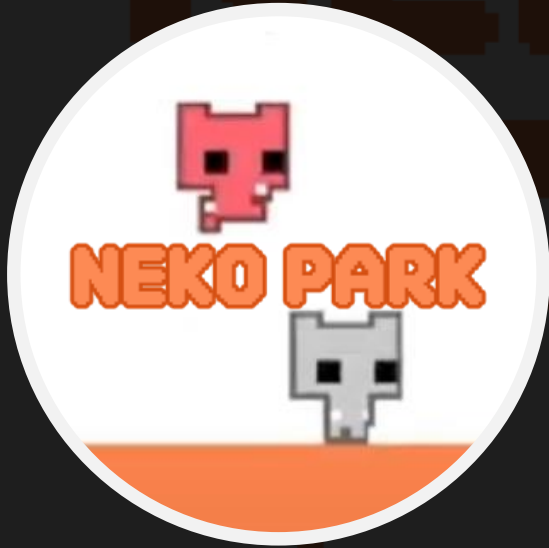


- 플레이어가 Object 뒤에 가려져도 실루엣이 표시되는 기능을 구현하기 위해 URP Renderer의 Renderer Feature에서 Stencil을 활용.
- URP Renderer Feature를 사용할 경우, 카메라에서 가려지는 부분이 생기면 Depth Test 조건을 충족하여, 따로 Logic을 처리하지 않아도 Renderer가 가려진 부분에 대해 추가로 Rendering 하여 편리하게 구현 가능.



- 간단한 설정으로도 Rendering 이후 추가적인 이미지 처리를 하기 위해 Volume Component를 추가해 Post Processing을 적용.
- Bloom 효과를 적용하여 빛이 퍼지며 부드러운 느낌을 연출하고자 의도.

Overview



NekoPark

Unity 2D Toy Project

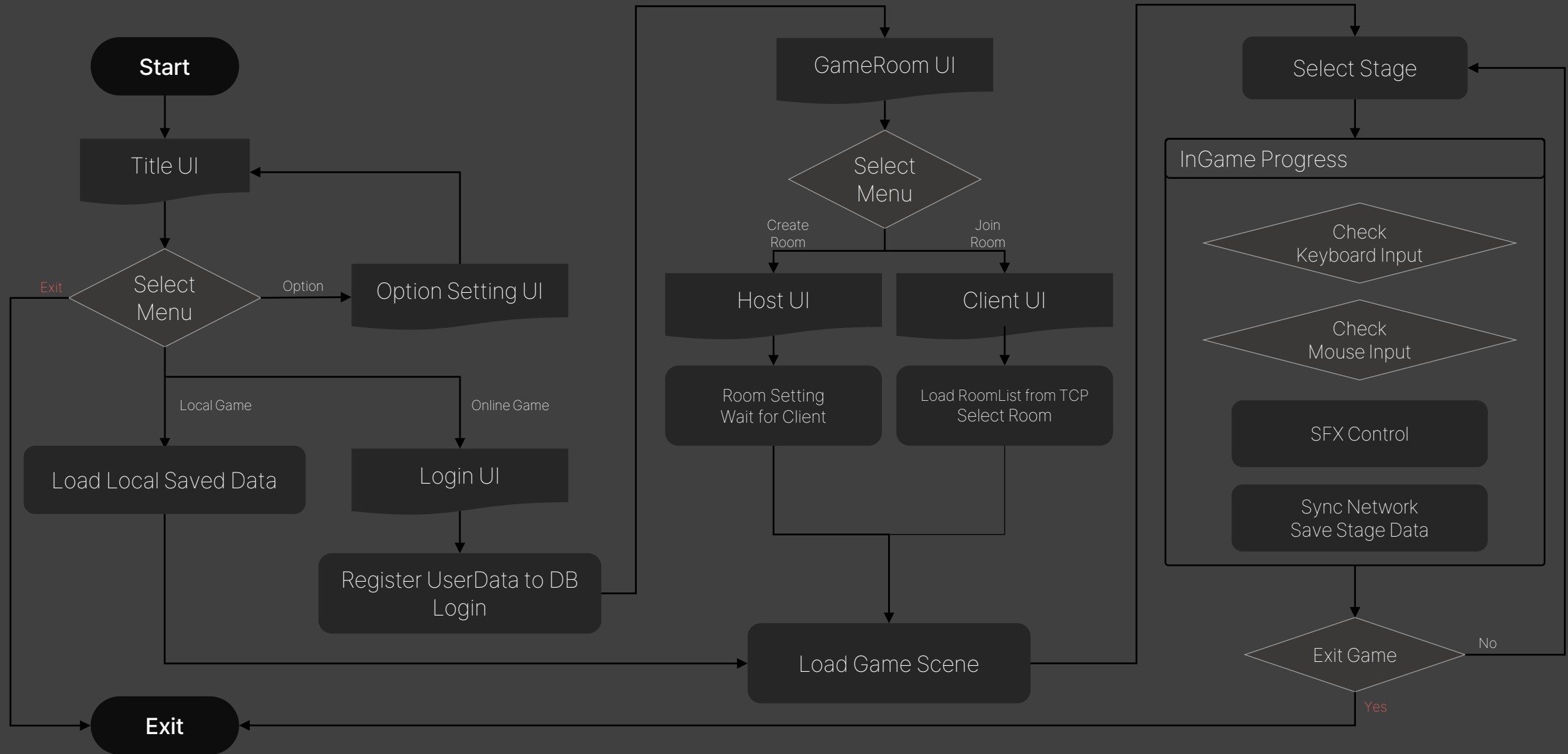
게임 장르 : 퍼즐 / 협동 / 캐주얼

개발 인원 : 개발 4 명

개발 환경 : Unity 2020.3.36 LTS

개발 기간 : 2024. 08. 12. ~ 2024. 08. 19. (1 주)

FlowChart



Features

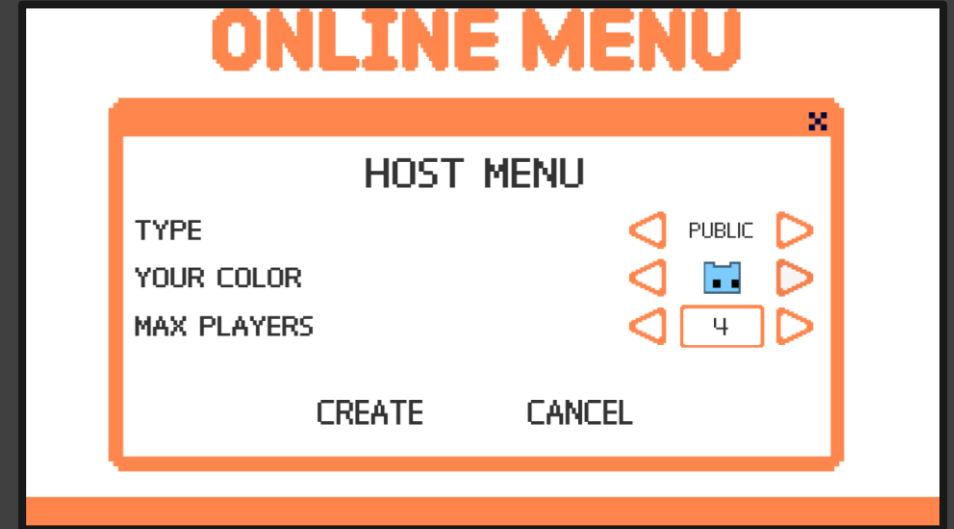


□ Mirror Network : RoomManager

- 멀티 플레이를 구현하기 위해 Mirror Library를 활용.

Mirror는 Documents 및 API가 사용하기 용이하게 되어있어 손쉽게 멀티플레이를 구현 가능

- Mirror의 NetworkManager가 제공하는 StartHost(), StartClient() Method를 활용하기 위해, NetworkRoomManager를 상속받는 RoomManager Component를 작성하여 Host-Client 간 Network 및 방 만들기 / 방 입장하기 기능을 구현.

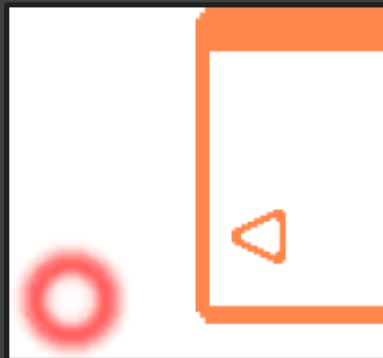


- Host가 방을 생성하거나 Client가 방에 입장할 때 RoomManager는 자동으로 NetworkRoomPlayer Prefab을 Instantiate 함. 이후 Network에 동기화될 수 있도록 NetworkServer.Spawn() Method를 통해 Network에 Spawning 하도록 작성.
- NetworkRoomPlayer를 상속받는 RoomPlayer Component를 작성하여 플레이어 Prefab에 추가함으로써, RoomManager를 통해 생성된 방에 플레이어가 입장하면 Spawning 될 때 플레이어의 색상, 위치 등 필요한 초기화 작업을 처리할 수 있도록 구현.

Features



□ Mirror Network



Click Effect 동기화
화면을 클릭하면 모든 Client의 화면에 동일한 Effect가 출력

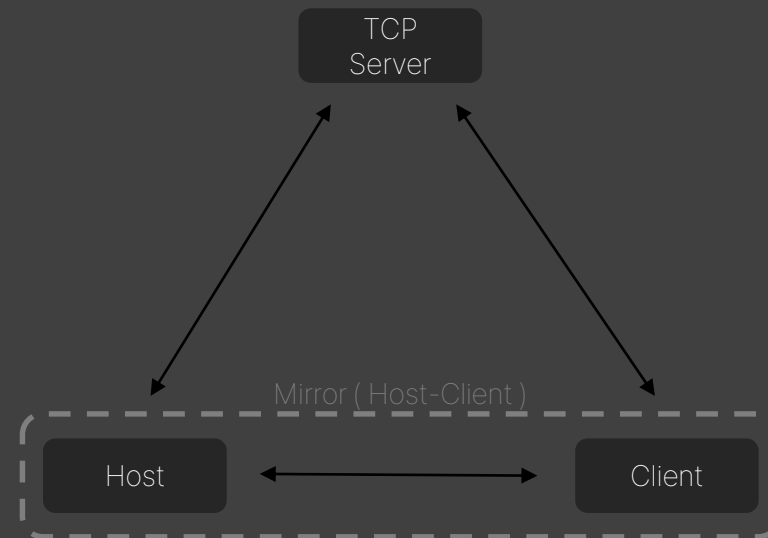
- RoomPlayer Component는 방에 입장하기 위해 필요한 최소한의 데이터만 담고 있기 때문에, 플레이어가 방에 입장하면 실질적으로 조작할 수 있는 객체인 Player Prefab를 별도로 Instantiate 하여 생성.
- 생성된 Player 객체는 Mirror의 Network Transform Component를 추가하여 실시간으로 Network에 자신의 Transform을 동기화하도록 처리.
- 플레이어의 색상, 닉네임은 한 번만 동기화되면 되므로, 플레이어가 생성될 때 Client가 RPC 호출을 통해 Host에서 초기화하고, [SyncVar]의 Hook Attribute에 설정된 Method를 호출하여 각 Client로 동기화하도록 구현.
- 현재 방에 연결된 인원을 확인하고 모든 Client가 동일하게 동기화하기 위해, RoomPlayer가 생성되거나 파괴될 때마다 RoomManager의 roomSlots를 조사해 각 Client의 UI에 표시하도록 구현.
- 플레이어가 화면을 클릭했을 때 생성되는 Clicked Effect 등, 각각의 동기화 요소는 RPC 호출을 통해 각 Client에서 동기화.

Features



□ TCP Server

- Mirror Network는 여러 Host가 방을 생성하고 이를 관리해주는 기능을 제공하지 않기 때문에, 생성된 방 목록을 Client에 전달하고자 별도의 TCP 서버를 활용.
- Host가 방을 생성하면, 방에 입장할 때 필요한 Host의 IP, PORT 등의 정보가 포함된 RoomData를 TCP 서버로 전달.
- 서버는 이를 저장하였다가 Client가 Request 할 때 생성되어있는 RoomData List를 Response로 전달.
- Client는 전달받은 RoomData List에서 방을 선택한 후, 해당 RoomData의 Host IP를 RoomManager의 NetworkAddress로 설정하여 Host의 방에 입장할 수 있도록 구현.



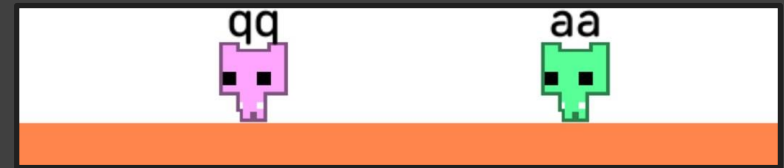
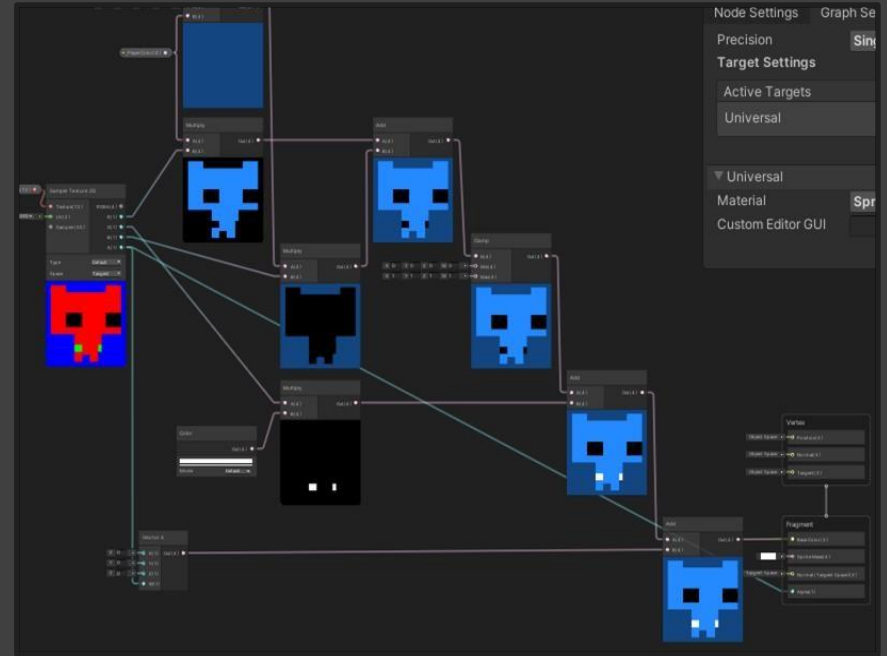
```
InloadTime: 0.534400 ms
ThreadLog initialized.
[2024-10-21 13:48:25] Server Starting...
[2024-10-21 13:48:25] 127.0.0.1:5555
[2024-10-21 13:53:52] Client Connected : 127.0.0.1
[Thread4] [2024-10-21 13:53:52] Client Connected : 127.0.0.1
[2024-10-21 13:53:52] Client Request : {"type":"Request","data":
  "hostPort":5555,"availableColor":[1,2,3,4,5,6,7,8],"hostColor#
  ":1,"maxConnected":8}"}
[Thread4] [2024-10-21 13:53:52] Client Request : {"type":"Reques
  t","hostPort":5555,"availableColor":[1,2,3,4,5,6,7,8],"
  Connected":1,"maxConnected":8}"}
]
[Thread4] 0
[2024-10-21 13:54:04] Client Connected : 127.0.0.1
[Thread4] [2024-10-21 13:54:04] Client Connected : 127.0.0.1
```

Features



□ Shader Graph : Custom Shader

- 각 플레이어가 서로 다른 색상을 지정할 수 있도록 Shader Graph를 활용.
- 모든 색상의 Sprite를 준비하고 Load할 경우 메모리 및 게임 용량이 불필요하게 낭비될 수 있으므로, 한 개의 RGB Color Sprite에 Shader 연산을 통해 색상을 지정함으로써 최적화를 의도함.
- 플레이어의 색상 종류를 Enum으로 정의하고, RGB 색상 값을 각 Enum Label에 매핑하여 Network 통신 시에 값을 전달하거나 플레이어 객체에 색상을 적용할 때 간편하게 사용할 수 있도록 Script를 작성.



Overview



The Wild Four

Unity 3D Project

게임 장르 : 생존 / 어드벤처

개발 인원 : 개발 4 명

개발 환경 : Unity 2020.3.36 LTS

개발 기간 : 2024. 07. 08. ~ 2024. 07. 31. (3 주)

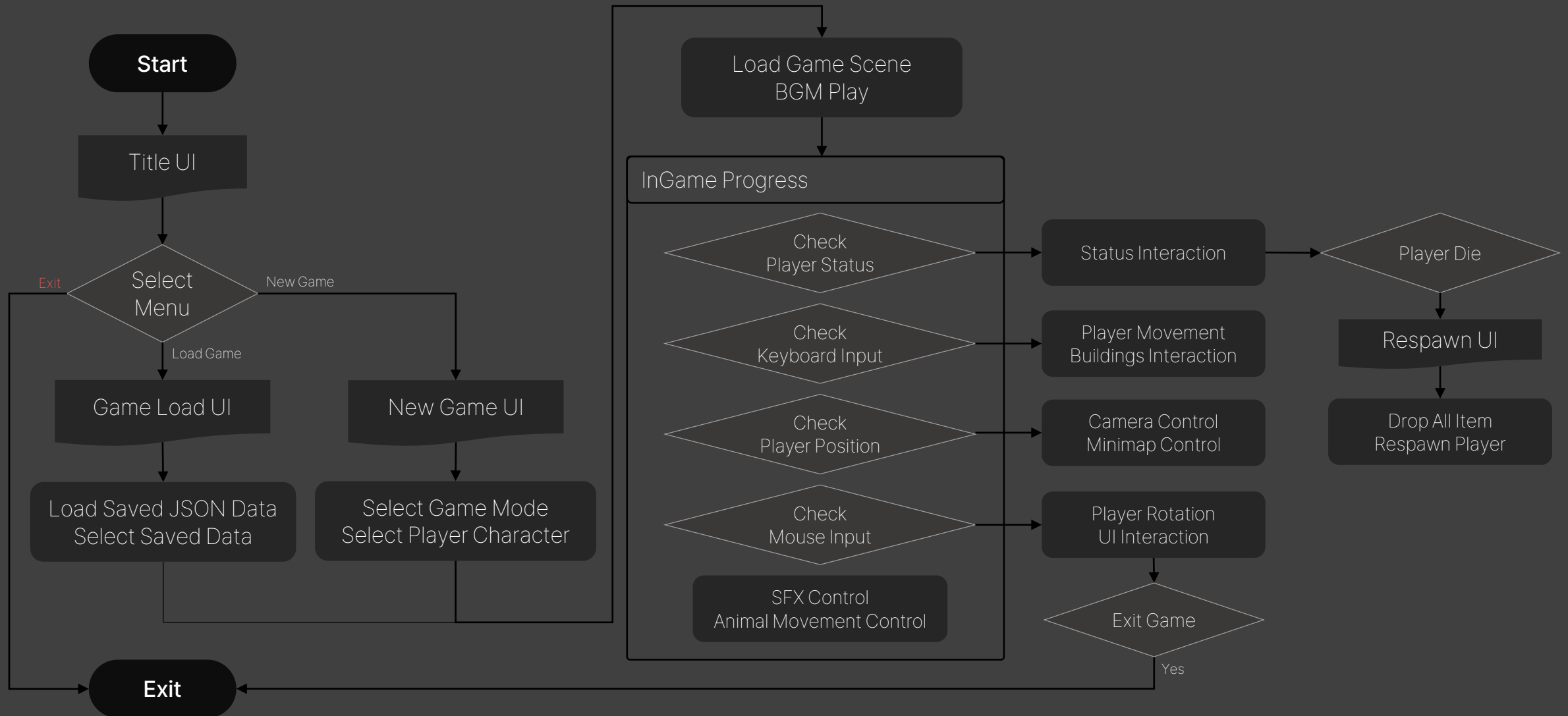


게임 시작

제작진

종료

FlowChart



Features



□ Player



- 플레이어의 부드러운 이동을 위해 `GetAxis()` 로 입력값을 받아서 `Rigidbody MovePosition()` Method를 사용하여 이동을 처리.
- 플레이어가 마우스 위치를 바라보도록 하고자 `Raycast`를 활용.
- 플레이어의 애니메이션은 Mixamo의 무료 에셋을 활용.
- Idle 상태와 이동 상태의 애니메이션은 플레이어의 속도에 따라 모션이 부드럽게 이어지는 효과를 위해 `Blend Tree`를 사용.
- 애니메이션 클립의 Tag를 조사하여 현재 공격, 줍기, 사망 등 애니메이션이 재생 중일 때, 다른 상호작용이 발생하지 않도록 구현.
- 하나의 캐릭터에 대한 `Animator Controller`를 구현한 뒤, 다른 3 개의 캐릭터는 `Override Animator Controller`를 활용. 중복되는 설정 작업이 없이 개발 기간을 단축시킬 수 있었음.
- 마우스 위치를 바라볼 때 상, 하반신이 따로 움직이게 하기 위해 `Character Humanoid Avatar`의 `Chest Spine Rotation`을 직접 Script로 제어.

Features

□ Player : Attack

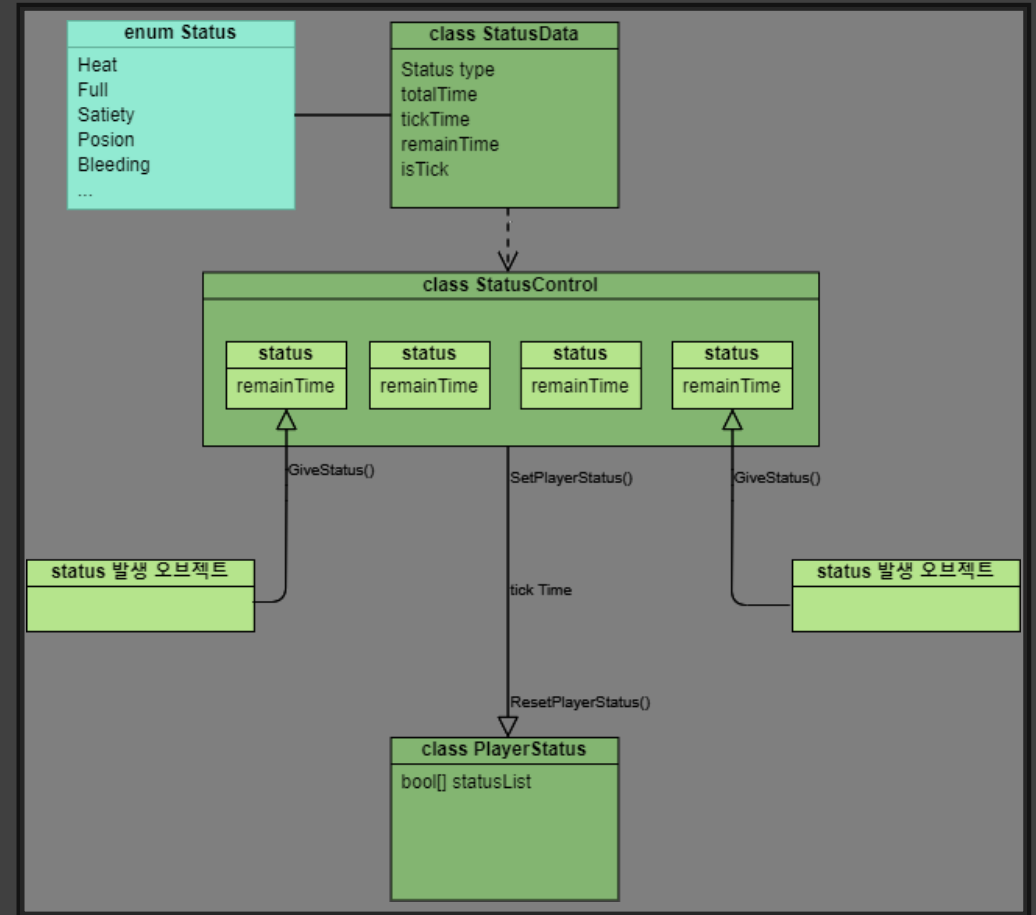


- 플레이어가 무기를 장착할 수 있도록 손의 위치에 WeaponPoint 객체를 생성. 현재 활성화된 무기를 WeaponPoint의 Transform과 동일하게 설정하도록 하여 장착을 구현.
- 플레이어의 공격을 처리하고자 장착 무기 및 손의 위치에 공격 대상을 검출하기 위한 별도의 Collider를 추가.
공격 애니메이션 진행 시 OnTrigger Event를 통해 처리.
- Trigger Event에서 검출된 Collider의 Layer를 조사하여, 공격 대상을 판단한 뒤 상황에 맞게 데미지 처리 Logic을 수행하도록 구현.
- 의도치 않은 중복 입력으로 인해 공격 애니메이션이 반복 또는 스킵되는 경우를 방지하기 위해 Animator State Info의 normalizedTime 을 활용, 현재 애니메이션이 재생 중일 경우 반복재생 되지 않도록 처리.

Features

□ Player : Status Control

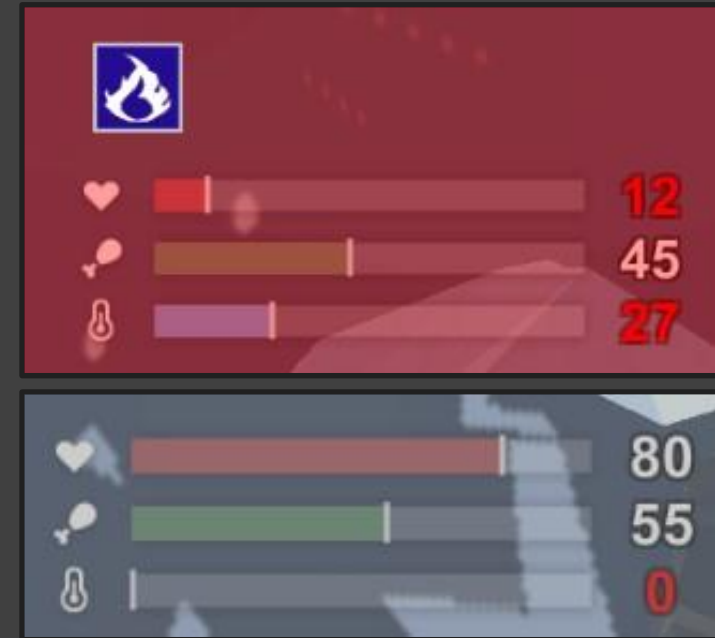
- 플레이어는 특정 행동을 통해서 버프, 디버프 Status를 가짐.
- Status Control을 Singleton Pattern으로 구현하여, Status를 발생시키는 여러 Script에서 접근이 용이하도록 구현.
- 또한, 단일 Instance에서 Status를 전부 관리하게 되므로 Status의 일관된 상태를 보장할 수 있고 메모리 최적화가 가능.
- Status 부여 객체(ex. 모닥불)가 Instance에게 특정 Status를 부여하길 요청하면, Status Control은 요청받은 Status를 플레이어에게 부여.
- 플레이어에게 부여된 각 Status의 총 시간과 잔여 시간은 Coroutine을 사용해 독립적으로 흐름을 제어. 잔여 시간이 모두 소진되면 PlayerStatus에서 부여된 Status를 해제하도록 구현.



Features

□ Player : Status

- 각 Status의 필요한 정보를 담기 위해 StatusData class를 정의.
- Status의 발생, 진행, 소멸에 필요한 데이터 및 메서드를 일관되게 관리할 수 있음.
- 플레이어는 [HP, 허기, 추위] 세 가지의 상태 게이지를 가짐.
추위나 허기 게이지가 0이 되면 체력 게이지가 감소,
체력 게이지가 0이 되면 플레이어가 사망하도록 구현.
- 각 Status에 따라서 상태 게이지가 증감하는 방식으로 작동.



Features

□ Skill



- 이동, 공격, 채집 행동을 할 때마다 해당 행동에 따른 경험치를 일정 비율로 획득하며, 경험치가 일정량 이상 모이면 포인트를 획득.
- [거처] 에서 포인트를 소모해 스킬 레벨을 올려 능력치를 강화.
- 각 스킬마다 상승하는 능력치, 또는 특수 능력의 사용 가능 여부를 List로 관리하며, Inspector에서 값을 조정할 수 있도록 구현.
- 플레이어가 거처에서 스킬 업그레이드를 할 때마다 PlayerAbility Component가 이 List를 순회하며 업그레이드된 스킬을 능력치에 반영하는 방식으로 구현.
- 각 스킬의 이름, 레벨, 능력치 상승 수치 등을 관리하기 위해 Skill class를 정의. Class Method로 스킬 데이터를 접근하도록 하여 스킬 관리가 용이하도록 구현.

Features

□ Time Manager



- 게임 내에서 시간의 흐름을 일관적으로 관리하고 전역에서 접근할 수 있도록 하기 위해 Time Manager를 Singleton Pattern 으로 구현.
- Time Manager는 게임의 시작과 동시에 시간을 흐르게 하고, 시간을 요청하는 각 Component에게 게임 시간을 return 하도록 구현.
- 따라서 낮과 밤, 태양, 조명, 생존 기간 계산 등 시간의 흐름에 따라 변화되는 게임 요소를 각자의 기준 값으로 계산하지 않고, 동일한 시간 값으로 오차 없이 계산될 수 있도록 구현.
- 낮과 밤의 조명 밝기 및 그림자의 이동을 표현하기 위해 간단히 Directional Light의 Rotation을 회전하는 방법으로 구현.
- 단, 인게임의 낮과 밤의 시간이 각 17시간, 7시간으로 차이가 있기에 Time Manager에서 받아온 시간 값을 연산하여 낮과 밤의 Rotate 속도가 다를 수 있게 연산 처리.

Features

□ Buildings

- 플레이어는 자재를 소모해 건물을 설치할 수 있고, 각각의 건물은 Interaction Component를 통해 고유의 상호작용을 할 수 있음.
- 플레이어의 일정 범위 내에 건물이 존재하면 상호작용할 수 있는 상태로 활성화. 상호작용 입력이 발생하면 Interaction Component에서 각 건물마다 존재하는 상호작용 Logic을 호출하는 방식으로 구현.
- 각 건물은 설치에 필요한 공통 기능과 개별적으로 필요한 고유 데이터를 구분하기 위해, BuildingCreate Script를 상속받은 고유의 설치 Component를 통해 설치되도록 구현.
- 건물의 설치 모드로 진입하면 해당 건물의 현재 레벨 건물을 활성화. 설치 모드에서는 건물의 Material 을 Transparent로 설정하고 Collider를 Trigger 모드로 하여, 설치 전에 먼저 설치 가능 여부를 확인하도록 구현.
- 설치 모드 중에는 RayCast를 사용하여 마우스의 위치에 건물 설치가 가능하도록 하고, 해당 위치에 다른 Collider가 겹쳐 있거나 지형의 경사가 과도하게 기울어진 경우를 검출하여 설치 가능 여부를 결정.



Features

□ Save & Load

- 플레이어 HP, 레벨, 인벤토리, 능력치 등의 데이터는 JSON 형식으로 저장되고 불러오도록 구현.
- 저장 데이터의 일관적인 관리와 Scene 변경에서도 단일 Instance를 보장하기 위해 Save Manager를 Singleton Pattern 으로 구현.
- 새 게임을 시작할 때는 세이프 파일을 초기 값으로 설정.
- 인게임에서 저장 버튼을 누르면 Save Manager가 저장해야 할 데이터가 존재하는 각 Component를 참조하여 데이터를 수집한 뒤, 현재의 Save Data에 덮어쓰는 방식으로 저장.
- 저장된 게임을 불러올 때는 Save Manager가 현재의 Save Data를 선택한 세이프 파일로 설정.
- Scene이 변경되고 게임이 시작되면 초기화가 필요한 각 Component는 Save Manager를 참조하여 자신의 필드 값을 저장된 데이터로 설정하도록 구현.



Overview



Terraria

Unity 2D Project

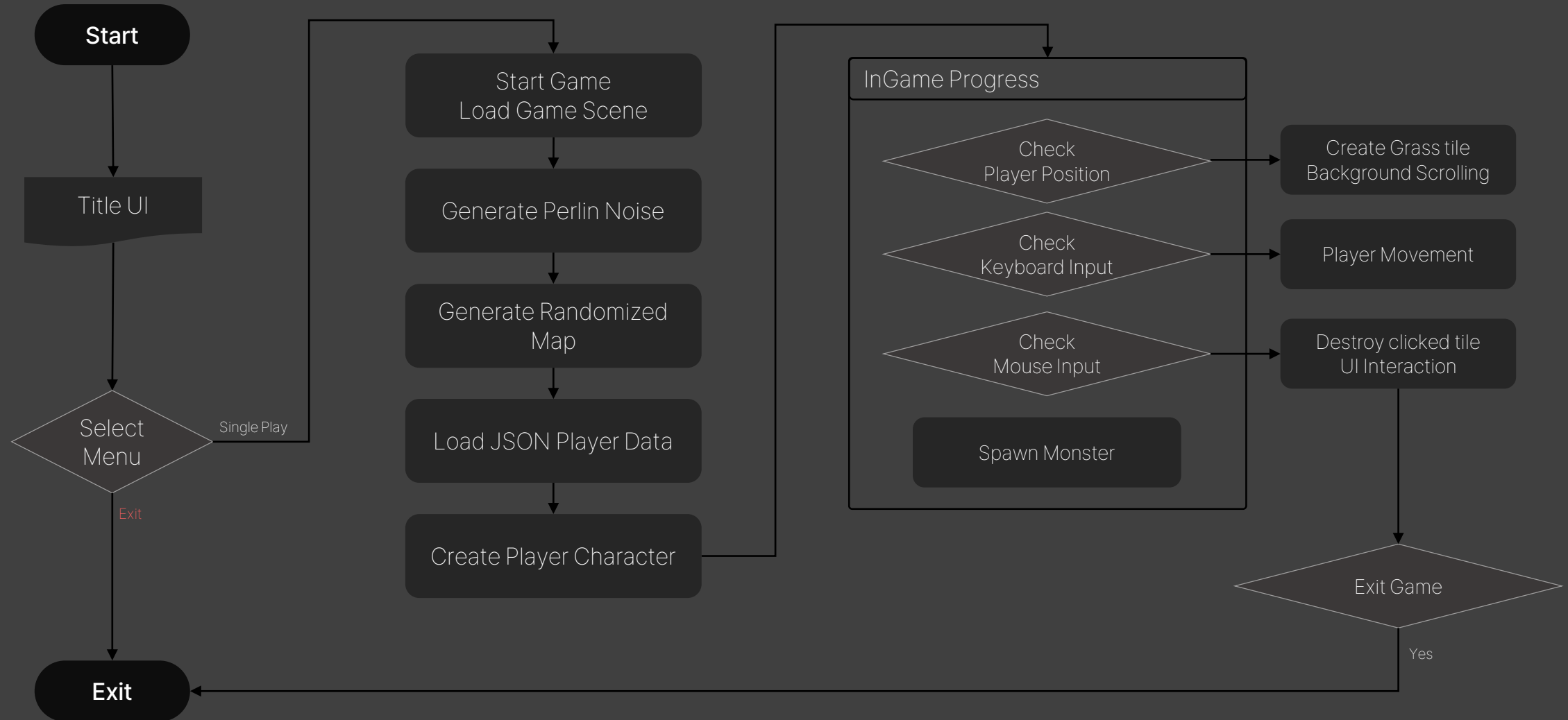
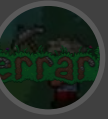
게임 장르 : 어드벤처 / 샌드박스

개발 인원 : 개인

개발 환경 : Unity 2020.3.36 LTS

개발 기간 : 2024.06.13. ~ 2024.06.24. (1주)

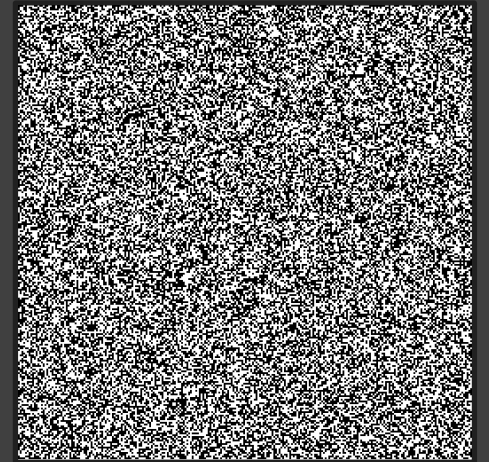
FlowChart



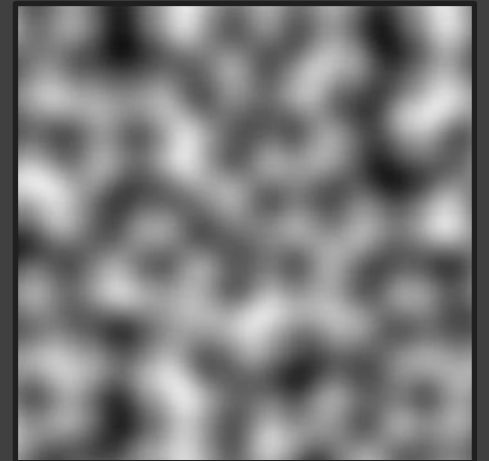
Features

□ Procedural Map Generation

- 무작위 지형을 생성하기 위해서 Noise Texture를 생성한 뒤 각 타일을 Noise Value에 대응시켜 배치하는 방식으로 지형을 생성.
- 완전 무작위 난수로 생성된 Noise는 인접 위치간 유기적인 연결 관계가 없어 단절된 형태를 갖기 때문에 지형 생성에 부적절.
- 인접한 위치의 값이 연속성을 갖는 Perlin Noise를 사용하여, 단절된 구간 없이 자연스럽게 이어지는 지형을 생성하도록 구현.
- Perlin Noise는 Unity에서 제공되는 Mathf 라이브러리에서 생성. 생성된 값을 persistence / scale 등 상수 값에 따른 추가적인 연산을 거쳐 맵 지형 생성에 적합하도록 조정.



Random Noise Texture



Perlin Noise Texture

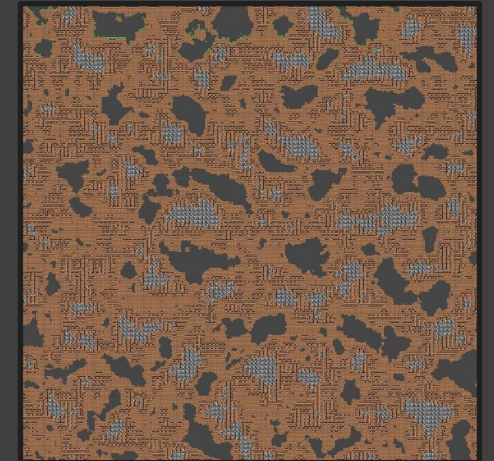
Features

□ Procedural Map Generation

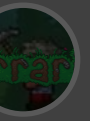
- 상수 값의 변경만으로 지형의 특성을 크게 바꿀 수 있어 차후 다양한 맵 Biome 및 지형 생성에 확장성 있게 적용 가능.
- 지형의 굴곡 및 전체 맵의 생성을 위해 1차원, 2차원 Perlin Noise를 복합적으로 사용.
- Noise 값을 0 ~ 1 사이의 실수로 정규화한 뒤 구간을 나눠 각 구간을 타일에 대응하도록 하여 다양한 타일을 배치하기에 용이하도록 구현.
- 생성된 지형 위에 다시 새로운 Perlin Noise Map을 적용하는 방식으로 맵 상에 좀 더 복잡적이고 다양한 광물 군 등 지형을 생성할 수 있음.



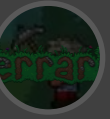
Scale : 50 / Lucunarity : 1 / Octave : 1



Scale : 30 / Lucunarity : 2 / Octave : 5



Features



□ Object Pooling



- 일정 시간마다 몬스터가 생성되도록 Object Pooling 기법을 활용.
- Object Pooling은 객체가 매번 Instantiate / Destroy 되는 것보다 비용이 적고 성능 향상을 기대할 수 있음.
- 게임이 시작할 때 일정 개수의 객체를 미리 생성한 뒤 Pooling을 통해 재활용.
- 만약 부족할 경우 추가적으로 객체를 생성해 Pool에 등록하도록 하여 동적으로 Pooling을 할 수 있도록 구현.

□ HSV Slider UI





- 캐릭터 생성 시 부위 별 색깔을 커스터마이징 할 수 있도록 HSV Slider를 Native로 구현.
- H Value에 따라 S, V Slider의 색깔이 변경되어야 하는데, 각 Slider의 Fill Texture를 모든 조합 가능색상의 Sprite로 지정하는 것은 리소스적으로 비효율적.
- 하나의 Sprite만 사용하고 Texture2D.SetPixel32() Method를 사용해 색상을 연산하여 지정하는 방식으로 구현.
- HSV Slider의 값에 따라 플레이어 캐릭터의 색깔도 같이 변경되며 각 Color Value가 저장되도록 구현.

감사합니다

전 흥 현

 GitHub

 010-3255-0364

 toinbee3@gmail.com