유니티 좀비 서바이버 만들기

그래픽 렌더링에 관한 내용과 , 3d 캐릭 , 다양한 동작들을 구현한 코드를 통해좀비 서버아버 프로그램 만들기

패키지 매니저를 통해서 패키지들을 임포트 함 ,

cinemachine : 스마트 추적 카메라와 복잡한 카메라 연출을 구현

package manager ui : 패지키 매니저의 ui창

post-process : 전체이름은 포스트 프로세싱 스택으로 후처리 효과를 구현

전통적인 유니티의 확장 프로그램 관리는 에셋 스토어를 통해 되지만 패키지 매니저를 통해 공유할시

자신의 폴더에 직접 추가하지 않고, 글로벌 캐시의 패키지를 참조해 사용하므로 프로그램이 가벼워 진다.

레벨아트와 라이팅 설정

레벨아트는 기존에 구현되있는 것을 가져와서 사용, 레벨아트 내에 빛을 내는 라이트 오브젝트가 있으므로 기존 라이트는삭제

라이팅연산 비용은 굉장히 비쌈, 게임내부에서 실제로 빛을 내는 객체가 존재하는 것이 아니라 ,객체가 이동할때 생길 그림자나 광원 효과를 미리 그려놓고 가져오는것이다

이프로젝트에서는 실시간 라이팅 맵 생성은 빼고 처음 한번만 생성 , 런타임시 부화를 줄일수 있음, window - rendering - lighting 에서 설정 변경 autogenerate를 해제해야 수동으로 설정가능하다.

글로벌 일루미네이션 : 물체의 표면에 직접 들어오는 빛 뿐만 아니라 다른 물체의 표면에서 반사되어 들어온 간접광까지 표현한것. 매우 높은 처리량을 요구,

유니티에서는 GI를 라이트맵과 여러 대안을 통해 구현한다.

실시간 글로벌 일루미네이션과 베이크된 글로벌 일루미네이션을 설정하여 사용할수있다.

실시간 GI 는 빛의세기와 방향등이 달라졌을 때 그변화를 실시간으로 반영 , 여러방향에대해 라이트맵을 생성해놓는다.

베이크 GI는 고정된 빛에의한 간접광으로 위치와 시간등에 따라 바뀌진 않지만 높은질과 성능이 좋다

gi는 static 게임 오브젝트에만 적용됩니다. 정적 오브젝트는 게임도중에 위치가 변경될수 없으나 더 많은 성능 최적화를 적용합니다.

리지드바디는 물리 상호작용, collider눈 충돌 상호 작용

애니메이터 레이어

유한 상태머신에서는 하나의 상태만 실행될 수 있으나 여러개의 유한 상태 머신을 병렬로 실행하는 방식으로 여러상태를 중첩할수있음(각각 다른 유한 상태머신)

블렌드 트리 : 애니메이션 클립을 혼합하여 보여주는 것

이 프로젝트를 예시로 들면 basemovment 안에는 여러가지 애니메이션이 섴여있음을 알수있다.

애니메이터 레이어 나누기 : 애니메이터 레이어를 혼합하여 사용할때 위에서 부터 적용시킨순으로 덮어쓰기 방식이 적용된다.

캐릭터 이동 구현

playerInput : 입력감지 다른 컴포넌트에게 전달

playerMovement : 입력에 따라 이동 회전

입력과 액터를 나누기: 이전에는 하나의 대상만을 이용해서 개발했지만 여러대상에게 적용될수 있도록 입력을 나누면 다른 액터들 에게 인풋 컴포넌트로 쉽게 조립가능

move의 input.getAxis는 6장에서 처럼 vertical과 horizontal로 유니티에서 할당된 키로 입력해면 누른 시간만큼의 입력값을 반환하여 여러 타입에서 입력받을수있다

fire와 reload는 버튼 입력방식으로 유니테에서 할당한 키를 가지고 사용하며 true와 false를 반환한다.

프로퍼티 : 변수값을 일거나 쓰는과정에서 유연한 처리를 삽입할수 있는 클래스 멤버. 예를들어 L,ml가 있고 프로퍼티 물 = new 프로퍼티 를 설정해놓을시 l를 ml로 전환

프로퍼티로 값을 저장하는 이유 = 데이터를 안전하게 상황에 맞는값으로 변환시켜놓아서 보관 및 사용하기 위해

Update() 와 fixedUpdate()의 차이

Update는 화면 갱신 주기, FixedUpdate()는 물리 갱신 주기0.02초 rigidbody나 collision등 물리에 관련된 것들은 update보다 fixed에서 처리하는 것이 오차가 날 확률이 줄어든다.

현재 입력에 따라 애니메이터 바꾸는 명령어

playerAnimator.SetFloat(“바꿀모양”,playerInput.Move)

playerInput.move는 입력에 따라 -1부터 1까지 값을 주므로 블렌드 트리에서 사용된다.

Move() 메서드 구현

Vector3 moveDistance = playerInput.move \* transform \*moveSpeed \* Time.deltaTime;

plyaerRigidbody.MovePostion(plyaerRigibody.postion + moveDistance);

transform.translate를 사용하지 않고 rigidbody를 사용하는 이유는 물리 구현에 있어서 좀더 안 끊기고 이동하는거처럼 보일수 있다.

( <https://answers.unity.com/questions/215377/transformtranslate-vs-rigidbodymoveposition.html>)

해외 답변을 찾아본결고 kinematic을 사용하지 않을시 transflate은 물리정보와 오브젝트 정보 둘모두의 정보를 사용하고 moveposition은 리지드바디 컴포넌트만을 사용하므로 메모리 사용과 오류를 줄일수 있기에 저런 특수한 경우에서는 추천된다.

// transform.position = transform.position + moveDistance;

이렇게 바꾸어 보았는데 난 별 차이를 못느꼇다.

++ transform.position은 이동위치에 물체가 있어도 물리처리를 무시하고 위치를 덮어쓸수도 있기 때문에

Rotation은 값을구할 때 vector가 아니라 quataerion.Euler()값을 곱해서 구함을 명심

시네머신 추적 카메라 : 카메라의 위치를 조정하여 원하는 컷신을 사용할수있도록 수학적 모델과 수식을 제공해주는 유니티 공식 에셋

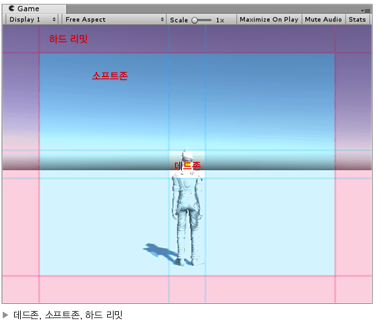
브레인 카메라와 가상 카메라로 구성되 있으며 브레인 카메라는 1개만 존재 가상카메라는 여러 개여 존재할수있고 가상 카메라중 하나의 설정값을 브레인카메라가 그 값으로 변경하며 다양한 각도에서 씬을 촬영한다.

이 게임에서는 카메라 전환 기능은 사용하지 않고 추적 카메라만 사용

메인카메라에 cinemacineBrain컴포넌트를 추가 -> 유니티 상단 메뉴에서 새 가상카메라를 생성->이름변경->follow필드와 lookat필드에 playerChararcter오브젝트를 지정

이렇게 할시 계속 따라다니는 카메라 설정값 생성완료

데드존, 소프트존, 하드리밋



주시하는 물체가 게임 화면 밖으로 벗어나지 않게 하는 단계별 설정

카메레가 보는 물체가 데드존에 있는동안은 카메라 회전 X

소프트존에 있는동안은 부드럽게 회전

하드리밋으로 가까워 질수록 빠르게 회전

followCam오브젝트의 lens에서 시야각을 조정

binding모드를 전역공간으로 설정하여 오브젝트와 카메라거리를 전역공간으로 계산

follwoffset으로 추적 오브젝트와의 거리설정

Damping으로 추적 제동 지연율 설정. 높을수록 부드럽게 이동되지만 신속하게 변경되지 않음

aim값을 조정하여 오브젝트에서 얼마나 떨어진 공간을 조준할건지 설정

소프트존 영역을 0으로 설정하여 즉각적으로 추적하도록 설정