유니티 4장. 주인공 캐릭터 제작

  // Start와 Update 함수는 가장 기본적인 함수이자 이벤트 함수.

  // Event 함수?? 특정 조건을 만족할 때 시스템에서 자동으로 호출해주는 함수를 말한다.

  // 이때 시스템은 유니티엔진을 말하며, Start와 Update 함수는 유니티 엔진에서 자동으로 호출해주기 때문에

  // 유니티 이벤트 함수라고 한다!

    void Awake() { }   // Start 함수가 호출되기 전에 맨 먼저 호출됨, 스크립트가 비활성화돼 있어도 실행됨

  // 스크립트가 실행될 때 한번만 호출되는 함수

  // 주로 게임의 상태 값 또는 변수의 초기화에 사용

  // 코루틴으로 실행 불가

    void OnEnable() {}

    // 게임오브젝트 또는 스크립트가 활성화됐을 때 호출됨

    // 주로 사용자 정의 이벤트 연결에 사용한다

    // 코루틴 사용 불가

  void Start() {}

    // Start 함수?

    // Update 함수가 호출되기 전에 한 번만 호출됨

    // 스크립트가 활성화돼 있어야 실행된다

    // 다른 스크립트의 모든 Awake가 모두 다 실행된 이후에 실행됨

    // 코루틴으로 실행 가능

    void Update() {}

    // Update 함수?

    // 프레임마다 호출되는 함수로 주로 게임의 핵심 로직을 작성함

    // 스크립트가 활성화돼 있어야 실행된다

    void LateUpdate () {}

    // LateUpdate 함수?

    // 모든 Update 함수가 호출되고 난 이후에 호출된다

    // Update 함수에서 전처리가 끝난 후 실행해야 하는 로직에 사용한다

    // 스크립트가 활성화돼 있어야 실행된다

    void FixedUpdate () {}

    // 물리 엔진의 시뮬레이션 계산 주기로 기본값은 0.02초다

    // 발생하는 주기가 일정

    void OnDisable () {}

    // 게임오브젝트 또는 스크립트가 비활성화됐을 때 호출된다

    // 이벤트 연결을 종료할 때 사용한다

    // 코루틴 사용 불가

        // 캐릭터 이동방법

    // 유니티에서 어떤 물체를 이동시키거나 회전시키는 방법은 2가지로 분류할 수 있다.

    // 첫 번째 방법은 모든 게임 오브젝트에 있는 Transform 컴포넌트의 position, rotation 속성값을

    // 지속해서 변경하는 것.

    // 두 번째 방법은 내장된 물리 엔진을 이용해 Force와 Torque를 가해 변경시키는 것이다.

    // Transform / PhysX, Box2D

        // 컴포넌트 캐시 처리

    // Update 함수는 프레임마다 한번씩 호출되기 때문에 항상 최적화에 신경써야 한다.

    // 예를 들어, 게임을 실행하는 디바이스가 30프레임이라면 Update 함수는 이론적으로 1초에 30번 호출된다.

    // 따라서 조금이라도 부하가 걸리는 함수나 잘못된 로직은 자제해야 한다.

    // 특히 이동 로직은 Transform 컴포넌트의 Position 속성을 조금씩 변경하는 것으로

    // 프레임마다 Transform 컴포넌트에 접근하는 방식은 바람직하지 않다!

}

**Time.deltaTime**

얘는 이전 프레임의 시작 시각부터 현재 프레임이 시작되는 시각의 차(델타)를 말한다. 쉽게 풀이하면 이전 프레임부터 현재 프레임까지 걸린 시간의 차.

게임이 초당 30프레임으로 구동된다면 Update 함수는 1초에 30번 호출되므로 총 300유닛을 이동한다.

해당 코드를 초당 60프레임으로 구동되는 환경에서 실행한다면? 1초에 600유닛을 이동하겠지.

같은 소스지만 Frame rate가 다른 환경에서 실행하면 속도가 달라지는 문제가 발생한다.

-> 이 문제는 deltaTime을 곱하는 것으로 해결이 가능하다!

30프레임으로 구동되는 기기의 Time.deltaTime 값은 1/30임. 1/30 \* 10 \* 30 = 10유닛 이동.

60프레임 구동되는 기기의 Time.deltaTime 값은 1/60임. 1/60 \* 10 \* 60 = 10유닛 이동.

Time.deltaTime을 곱하지 않았을 때는 프레임당 지정한 유닛만큼 이동하고, Time.deltaTime을 곱하면 초당 지정한 유닛만큼 이동한다. 특히 Update 함수에 이동 및 회전을 처리하는 로직을 작성했다면 반드시 Time.deltaTime 속성을 사용해야 한다.

**Public, private 접근 제한자**

C#에서 지원하는 접근 제한자는 public, private, protected, internal이 있다.

접근 제한자는 외부 클래스(구조체), 멤버 변수 등의 접근을 허용하는 범위를 지정한다.

Public : 외부 클래스(스크립트)에서 접근 가능

Private : 동일 클래스에서만 접근 가능, 외부에서 접근 불가능

Protected : private와 동일하게 외부에서는 접근 불가능, 상속받은 파생 클래스에서만 접근 가능

Internal : 같은 어셈블리에서만 접근 가능. 클래스의 경우 접근 제한자를 생략하면 internal이 기본값

**인스펙터 뷰에 노출된 변수의 우선순위**

유니티에서는 접근제한자가 public인 변수는 스크립트에서 지정된 값보다 인스펙터 뷰에서 조정된 값이

우선순위가 높다.

**Private 변수의 인스펙터 뷰 노출**

유니티를 실행하면 PlayerCtrl 스크립트가 실행되며 가장 먼저 Start 함수가 실행된다. tr 변수는 private로 선언했기 때문에 추출된 Transform 컴포넌트가 연결됐는지 현재로선 확인할 수 없다.

하지만 인스펙터 뷰의 모드를 디버그 모드(Debug Mode)로 설정하면 private 변숫값도 확인할 수 있다. 단, 디버그 모드에서 확인을 완료했다면 반드시 노멀 모드로 전환한다. 디버그 모드에서는 노멀 모드의 많은 속성이 보이지 않는다.

또 다른 방법은 **SerializeField 속성(Attribute)을 사용**하는 것이다. PlayerCtrl 스크립트를 다음과 같이 수정한 다음 인스펙터 뷰를 확인하면 노멀 모드의 인스펙터 뷰에서도 private 변수가 노출된 것을 확인할 수 있다. SerializeField는 private 접근 지시자의 속성을 유지한 채 인스펙터 뷰에 노출하는 기능이다

**벡터의 덧셈 연산**

스크립트 관련이라 playerCtrl 예제 참고(147p)

전진/후진 방향 벡터와 좌우 방향 벡터를 다음과 같이 벡터의 덧셈 연산을 이용해 하나의 벡터로 만든 다음 속력을 곱하면 전후좌우로 이동시킬 수 있다. 근데 이렇게 하면 속도가 살짝 빠르다.

MoveDir 변수는 Vector3 타입으로, 키보드 입력값을 이용해 벡터 연산을 했다. 전진 방향의 벡터와

좌우 방향의 벡터를 덧셈 연산하면 대각선 방향의 벡터가 생성된다. 이 대각선 벡터의 길이는 피타고라스 정리에 의해 루트 2다. 대각선으로 이동할 때 속도가 약간 빠른 원인. 1.141414……

따라서 길이가 1인 벡터로 변환해 방향 성분만 사용해야 한다.

이처럼 길이가 1인 벡터 단위를 단위벡터 또는 정규화 벡터(Normalize Vector)라고 한다.

벡터의 방향 성분만 추출하기 위해 정규화 벡터로 변경하려면 Vector3.normalized 속성을 이용해 정규화 벡터값을 사용한다.

**캐릭터 회전 – Rotate**

게임오브젝트를 회전할 때는 Transform.rotation 속성값을 변경하거나 Rotate 게열의 함수를 사용할 수 있다. 가장 기초적인 Rotate 함수는 다음과 같이 다양한 인자를 사용할 수 있다.

\* void Rotate(Vector3 eulerAngles, [Space relativeTo]);

\* void Rotate(float xAngle, float yAngle, float zAngle, [Space relativeTo]);

\* void Rotate(Vector3 axis, float angle, [Space relativeTo]);

**씬 뷰에서의 카메라 이동**

메인 카메라가 주인공을 따라가게끔 만드는 로직을 구현하기 전까지는 씬 뷰를 렌더링하는 가상 카메라가 주인공 캐릭터를 따라가는 기능을 활용해보자. 이 기능은 필자가 오프라인 강의에서 반드시 소개하는 기능이랜다.

먼저 하이러키 뷰에서 플레이어 선택 후 Shift + F 누르면 씬 뷰의 가상 카메라가 플레이어 위치로 이동한다. 즉 플레이어가 화면 가운데에 위치함.

**애니메이션**

유니티는 레거시 애니메이션(Legacy Animation)과 메카님 애니메이션(Mecanim Animation)이라는 두 가지 유형의 애니메이션을 지원한다.

\* 레거시 애니메이션 : 하위 호환성을 고려한 애니메이션, 소스 코드로 컨트롤해야 함

\* 메카님 애니메이션 : 모션 캡처 애니메이션, 리타게팅 가능

\*\* 리타게팅(Retargetting) : 애니메이션을 재사용하는 기능을 말함.

해당 챕터에선 주인공을 레거시 애니메이션으로, 적 캐릭터는 메카님 애니메이션으로 구현할 것임.

Animation Type 속성은 다음과 같다.

\* None : 애니메이션을 사용하지 않음

\* Legacy : 하위 호환성을 유지하기 위한 이전 방식의 애니메이션

\* Generic : 메카님 애니메이션. 인체형 모델이 아닌 3D 모델에 적용. 리타게팅 X

\* Humanoid : 메카님 애니메이션. 2족 보행하는 모델에 적용. 리타게팅 O

레거시 애니메이션의 컴포넌트 이름 : Animation

메카님 애니메이션의 컴포넌트 이름 : Animator

이 둘은 작동방식이 완전히 다르다. 따라서 3d 모델에 추가된 컴포넌트의 종류를 보고 해당 모델의

애니메이션 유형이 무엇으로 설정됐는지 구별할 수 있어야 한다.

**애니메이션 클립**

캐릭터의 걷기, 달리기, 점프, 총 쏘기와 같은 동작을 기록한 파일을 말한다. 그리고

애니메이션 컴포넌트는 애니메이션 클립에 기록된 관절의 위치와 회전 값을 프레임 단위로 재생시키는 역할을 한다.

3D 모델링 툴에서 제작한 애니메이션 클립을 애니메이션 파일로 만드는 방법에는 세 가지 방식이 있다.

1. 모든 애니메이션 클립이 하나의 파일에 들어있고, 각 애니메이션 클립이 시작 프레임과 종료 프레임을 가지는 방식

2. 모든 애니메이션 클립이 하나의 애니메이션 파일에 들어가 있지만, 미리 분리된 방식

3. 애니메이션 클립을 동작별로 분리해 별도의 파일로 생성하는 방식

**애니메이션 적용**

일단 이번 챕터에선 Idle, RunB, RunF, RunL, RunR 5개만 쓸거다.

애니메이션 클립 선택 -> 인스펙터 뷰에서 Wrap Mode 수정.

**애니메이션 블렌딩**

CrossFade 함수 관련 이야기. 스크립트는 166~168p 참고

무기 장착 완료.

**그림자와 빛**

그림자는 너무 중요한 요소이다. 그림자를 표현하는 방법은 아래와 같다 :

**실시간 그림자 :**

유니티는 Directional Light, Point Light, Spotlight 총 3가지 광원에 대해 실시간 그림자를 지원한다. 하지만 모바일 디바이스의 성능에 따라 실시간 그림자는 처리 비용이 크다.

유니티는 실시간 그림자가 기본으로 적용되어 있으며 Directional Light의 속성 중 Shadow Type 옵션으로 설정한다.

Shadow Type 속성 3가지 : No Shadow, Hard Shadow, Soft Shadow로 세 가지 옵션이 있다.

하드 쉐도우나 소프트 쉐도우를 선택하면 실시간 그림자가 적용된다.

실시간 그림자의 효과는 매우 탁월하지만 엔진에 많은 부하를 준다. 따라서 실시간 그림자 효과가 필요 없는 3d 모델은 실시간 그림자 영향에서 제외하는 설정을 빠뜨리면 안된다.

3d 모델은 Mesh Renderer 또는 Skinned Mesh Renderer 중 한 컴포넌트를 반드시 갖고 있다.

그리고 이 2개의 컴포넌트에는 실시간 그림자와 관련된 Cast Shadows와 Receive Shadows 속성이 있다.

\* **Cast Shadows** : 빛을 받아서 자신의 그림자를 만들 것인지 결정하는 속성

Off – 그림자 안만듬

On – 그림자 만듬

Two Sided – 백 페이스 컬링을 무시하고 그림자를 양면으로 만든다.

Shadows Only – 그림자를 만들지만, 자신은 렌더링하지 않는다.

즉, 화면에 보이지 않는 특성이 있어 그림자 처리만을 위한 로우 폴리 3D 모델에 활용할

수 있다.

\* **Receive Shadows** : 다른 그림자에 들어갔을 때 표면에 그림자의 영향을 받을지 결정

따라서 실시간 그림자의 영향을 받지 않아도 시각적 효과에 크게 상관없는 3D 모델은 이 두 가지 옵션을 적절히 설정해 그림자를 생성하지 않게 한다.

**Quad를 이용한 그림자 구현**

이 방법은 부하가 적어 간단하게 많이 사용되지만, 굴곡이 많은 맵에선 그림자가 가려짐.

개인적으로는 잘 안쓸 듯.

**Level Of Detail 설정**

3D 모델의 품질이 조금 이상하다고 느꼈다. 이미지가 깔끔하지 않고 뭔가 겹쳐보임.

LOD(Level Of Detail) 기능은 화면을 렌더링하는 카메라로부터 멀리 떨어질수록 낮은 폴리곤으로 변경해 렌더링 부하를 줄여주는 기법이다. 따라서 주인공 캐릭터는 3단계의 폴리곤으로 이뤄진 모델로서,

Slade\_mesh가 가장 하이 폴리곤이며 slade\_meshLod1이 미들 폴리곤,

slade\_meshLod2가 로우 폴리곤이다.

**LOD Group 컴포넌트**

카메라와 피사체의 거리에 따라 렌더링할 메시를 결정하는 역할을 함.

LOD Group은 기본적으로 3단계의 LOD 구간과 Culled 구간으로 설정돼 있다. 마지막 culled는 카메라의 거리가 아주 멀리 떨어져 화면에서 렌더링하지 않는 구간을 의미한다. 구간의 개수를 늘리거나 줄이고 싶다면 lod 구간 위에서 마우스 오른쪽 버튼을 클릭한 다음 팝업된 메뉴에서 처리할 수 있다.

또한 lod 구간과 구간 사이를 마우스 드래그해 구간의 범위를 변경할 수 있다.

아마 멀티플레이 게임에서 최적화를 위해 많이 사용될 것이다.

**Follow 카메라 로직 \***

자 이제 Main Camera로 3인칭 시점을 만들 것 같다. 184~186p 참고

[Range(min, max)] 어트리뷰트를 사용하면 다음 라인에 선언한 변수의 입력 범위를 (min, max)로 제한시킬 수 있고, 인스펙터 뷰에 슬라이드바를 표시한다.

**Vector3.Lerp, Vector3.Slerp**

선형 보간(Linear Interpolation)과 구면 선형 보간(Spherical Linear Interpolation)은

시작점과 끝점 사이의 특정 위치의 값을 추정할 때 사용된다. 이러한 보간 함수는 현재 값을 목표값으로

변경할 때 갑자기 변경하지 않고 부드럽게 변경시키는 로직에 많이 활용된다.

**선형 보간에서**, 선형은 직선을 의미한다. 그림 189p 참조.

시작점에서 끝점으로 이동하는데 걸리는 시간 t를 1초로 가정한다면 0.25초일 때의 이동 거리와 0.5초일 때 이동 거리가 동일하다. 따라서 선형 보간은 균일한 속도로 이동시키거나 회전시킬 때 사용한다.

유니티에서는 Lerp 선형 보간 함수를 제공하며 Vector3.Mathf, Quaternion, Color 구조체에서 사용할 수 있다.

Vector3.Lerp(시작 좌표, 종료 좌표, t);

Mathf.Lerp(시작 값, 종료 값, t);

Quaternion.Lerp(시작 각도, 종료 각도, t);

**구면 선형 보간은** 구체 형태로 값을 추론한다.시작점과 종료점은 느리게 증가하고 중간지점은 빠르게 이동하는 특성이 있다. 구면 선형 보간은 주로 회전 로직에 사용된다.

유니티에서는 Slerp로 Vector3, Quaternion 구조체에서 사용할 수 있다.

Vector3.Slerp(시작 좌표, 종료 좌표, t);

Quaternion.Slerp(시작 각도, 종료 각도, t);

부드럽게 이동시키는 방법 중 Vector3.SmmothDamp 함수를 쓸 수도 있다. 보통 카메라의 Follow 로직에 많이 사용 되며 사용법은 다음과 같다.

Vector3.SmmothDamp(Vector3 current, 시작위치

Vector3 targer, 목표위치

Ref Vector3 currntVelocity, 현재 속도

Float smoothTime, 목표 위치까지의 도달 시간

Float maxSpeed, 최대 속력 제한 값(기본값 무한대), 생략 가능

Float deltaTime); 프레임 보정을 위한 델타타임, 생략 가능