[유니티 기초] 17. 아이템 줍고 장착하기

유니티·개발

2015. 11. 2. 0:41

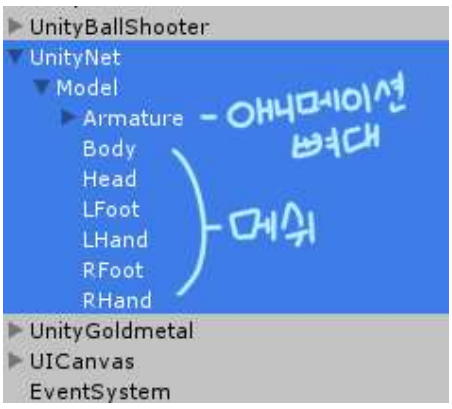
http://blog.naver.com/gold_metal/220526236275

안녕하세요.
골드메탈입니다.

오늘은 액션 RPG에서는 없어선 안될
아이템을 줍고 장착하는 기능을 구현해보는
시간을 갖도록 하겠습니다.

16편의 프로젝트에 이어서 진행하겠습니다.

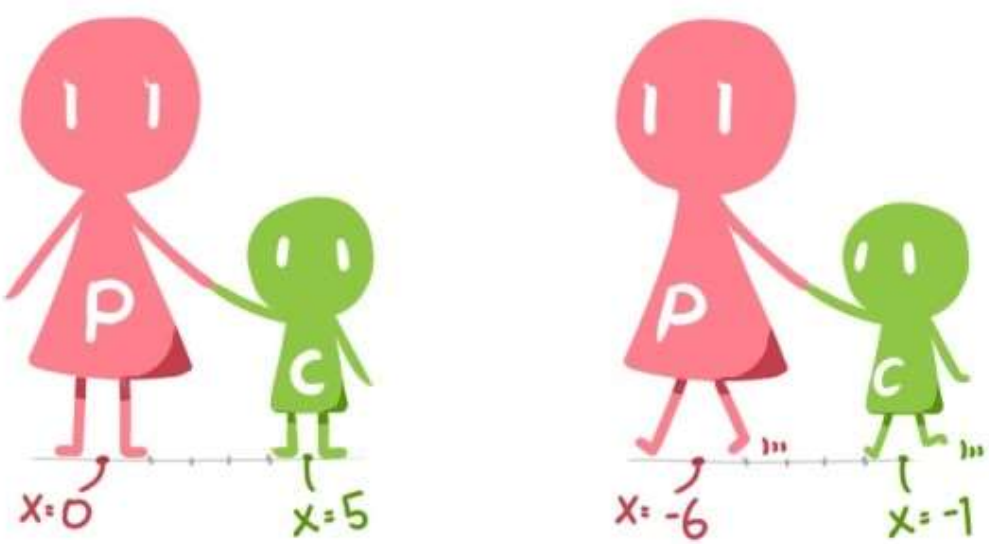
자, 먼저 플레이어 오브젝트를 살펴보도록 합시다.



오브젝트를 펼쳐보면

애니메이션 뼈대와 각 파츠별 메쉬들이 들어가 있습니다.
(뼈대 이름은 애니메이션 작업 환경에 따라 다를 수 있습니다.)

애니메이션 뼈대와 메쉬들은
서로 부모-자식 관계로 이루어져 있는데,
이러한 관계를 **종속관계**라고 합니다.



부모 오브젝트에 종속된 자식 오브젝트는
서로 위치가 다를 수 있지만
부모의 움직임에 따라 자식도 같이 움직이게 됩니다.

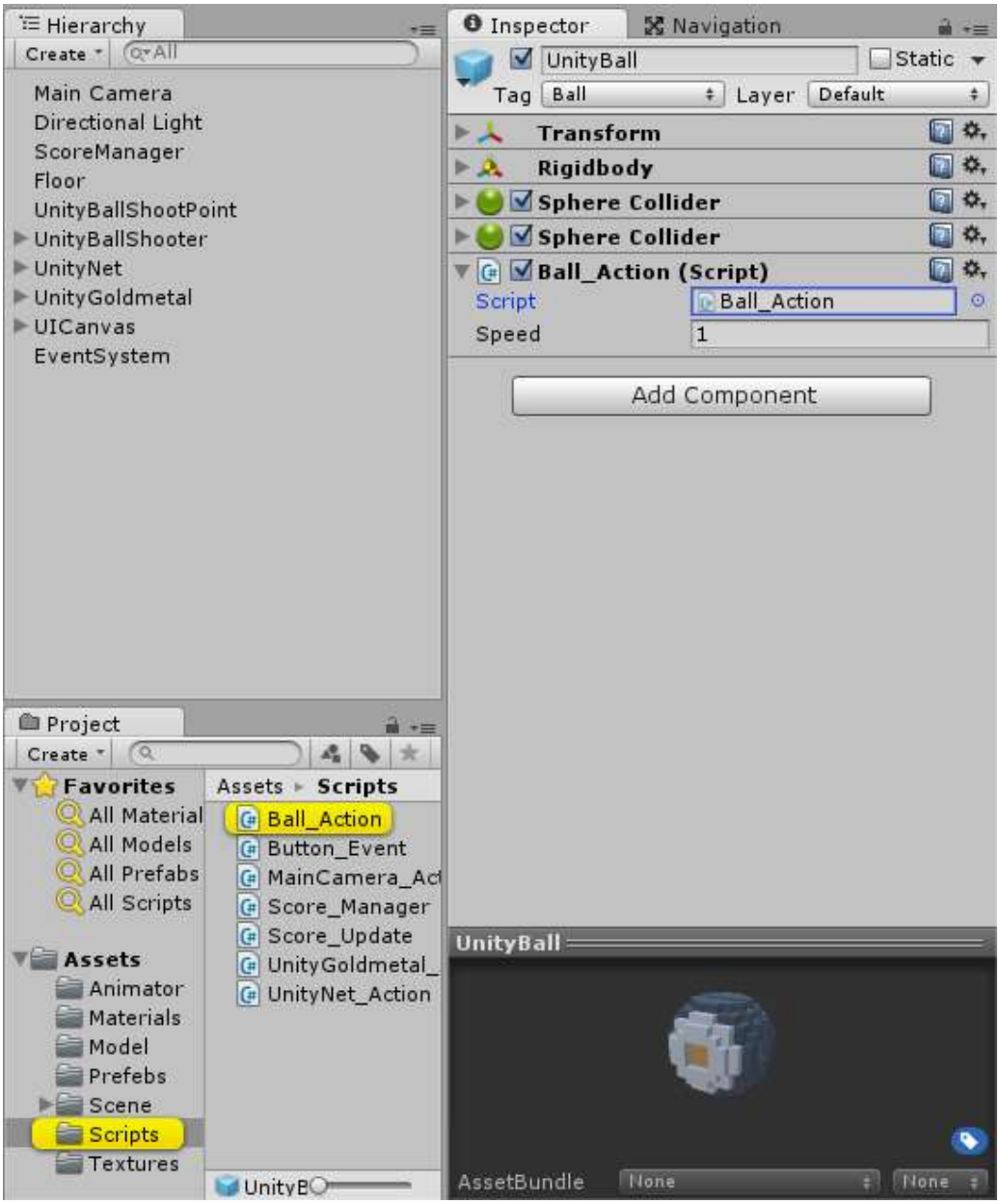
플레이어를 이루는 각 파츠 메쉬들도
애니메이션의 뼈대의 움직임에 따라 움직여서
실제로 우리 눈에 보여지는 애니메이션이 되는 것이지요.

우리는 이 점을 이용하여
외부 오브젝트가 플레이어 애니메이션에 참여하여
움직일 수 있도록 할 것입니다.

먼저 아이템을 장착할 곳의 애니메이션 뼈대에
장착 위치가 될 자식 오브젝트를 만듭니다.
필자는 오른손에 아이템을 장착하기 위해 오른손 뼈대에 만들었습니다.

그런 다음 장착 위치를 적당히 조절합니다.
여기서 좌표 기준(0,0,0)은 부모 오브젝트의 위치가 됩니다.
이걸 **지역위치(localPosition)**라고 부르지요.

마지막으로 스크립트에서 불러오기 위해
이름과 태그를 설정합니다.



이제 아이템의 스크립트로 들어가봅시다.
필자는 프리펩으로 저장해둔 공을 재사용하겠습니다.

```
GameObject player;
GameObject playerEquipPoint;
UnityNet_Action playerLogic;

Vector3 forceDirection;
bool isPlayerEnter;

//-----------------------------------------------------[ Override Function ]

void Awake ()
{
    player = GameObject.FindGameObjectWithTag ("Player");
    playerEquipPoint = GameObject.FindGameObjectWithTag ("EquipPoint");

    playerLogic = player.GetComponent<UnityNet_Action> ();
}

void Update ()
{

}

void OnTriggerEnter (Collider other)
{
    if (other.gameObject == player){
        isPlayerEnter = true;
    }
}

void OnTriggerExit (Collider other)
{
    if (other.gameObject == player){
        isPlayerEnter = false;
    }
}
```

위에서 지정했던 태그를 통해서 장착 위치 오브젝트를 불러옵니다.
그리고 플레이어의 스크립트도 불러와줍시다.

플레이어가 충돌체에 접근한 상태로 액션 키를 누르면
아이템이 장착되도록 로직을 구현할 것입니다.

```
void Update ()
{
    if (Input.GetButtonDown("Action") && isPlayerEnter){
        transform.SetParent(playerEquipPoint.transform);
        transform.localPosition = Vector3.zero;
        transform.rotation = new Quaternion(0,0,0,0);

        isPlayerEnter = false;
    }
}
```

아이템을 장착하려면 먼저 장착 위치 오브젝트에 자식으로 종속되야 합니다.
(결국 애니메이션 뼈대의 손자가 되는 격.)

transform.SetParent(tranform) 함수를 사용해서 부모를 지정해준 다음,
위치가 정확히 장착 위치에 오도록 지역 위치를 원점으로 설정해줍시다.

이 상태로 한번 작동시켜보죠.

생각과는 달리 바로 놓쳐버리면서
움직임이 멋대로 왔다갔다하네요.

이유는 아이템의 여러 충돌체와 리지드바디의 물리효과로 인해
부모의 움직임에 방해를 주고 있기 때문입니다.

(하긴 이렇게 쉽게 끝날리가 없지..)

플레이어 스크립트로 들어가서
이 문제점을 해결해야겠습니다.

```
public void Pickup (GameObject item)
{
    SetEquip(item, true);
}

void Drop ()
{
    GameObject item = playerEquipPoint.GetComponentInChildren<Rigidbody> ().gameObject;
    SetEquip(item, false);
}

void SetEquip (GameObject item, bool isEquip)
{

}
```

다른 오브젝트에서 호출
: public 키워드 추가

아이템 줍기와 버리기, 그리고 장착에 대한 함수를 새로 만들어주었습니다.
특히 줍기는 아이템 스크립트에서 불러올거니까 **public** 키워드로 꼭 공개해둡시다.

버리기 함수를 잠깐 살펴보면
장착된 아이템 오브젝트를 불러오기 위해
GetComponentInChildren<T>() 함수를 사용한 것을 볼 수 있지요.

GameObject를 바로 불러올 수는 없기 때문에
아이템의 필수 컴포넌트(리지드바디 혹은 충돌체)를 잠시 빌려서
GameObject에 접근해야 합니다.

```
void SetEquip (GameObject item, bool isEquip)
{
    Collider[] itemColliders = item.GetComponents<Collider> ();
    Rigidbody itemRigidbody = item.GetComponent<Rigidbody> ();

    foreach(Collider itemCollider in itemColliders){
        itemCollider.enabled = !isEquip;
    }

    itemRigidbody.isKinematic = isEquip;
}
```

장착 함수에서는 움직임을 방해하는 요소인
물리효과나 트리거 이벤트를 모두 끄거나 켜는 로직을 구현해 놓습니다.

모든 컴포넌트는 **enabled** 속성을 사용해서 켜고 끌 수 있습니다.
리지드바디는 **isKinematic(스크립트에 의해서만 움직임)**을 사용하면 됩니다.

```
public void Pickup (GameObject item)
{
    SetEquip(item, true);

    animator.SetTrigger("doPickup");
    isPicking = true;
}

void Drop ()
{
    GameObject item = playerEquipPoint.GetComponentInChildren<Rigidbody> ().gameObject;
    SetEquip(item, false);

    playerEquipPoint.transform.DetachChildren();
    isPicking = false;
}

void SetEquip (GameObject item, bool isEquip)
{
    Collider[] itemColliders = item.GetComponents<Collider> ();
    Rigidbody itemRigidbody = item.GetComponent<Rigidbody> ();

    foreach(Collider itemCollider in itemColliders){
        itemCollider.enabled = !isEquip;
    }

    itemRigidbody.isKinematic = isEquip;
}
```

마무리 단계로 줍는 함수에 애니메이션도 넣으면서
아이템을 가지고 있다는 플래그 변수도 하나 만들어놓습니다.

버리기 함수에서는
더 이상 종속관계가 되지 않도록
transform.DetachChildren() 함수로 떼어놓습니다.

```
void Update ()
{
    //Walking
    horizontalMove = Input.GetAxisRaw ("Horizontal");
    verticalMove = Input.GetAxisRaw ("Vertical");

    //Running
    isRunning = !(horizontalMove == 0 && verticalMove == 0) && Input.GetButton("Run")
                ? true : false;

    //Jumping
    if(Input.GetButtonDown("Jump"))
        isJumping = true;

    //Dropping
    if(Input.GetButtonUp("Fire1") && isPicking)
        Drop();

    AnimationUpdate();
}
```

Update 사이클에서 버리기 로직도 넣어주어야겠죠?
필자는 마우스 클릭으로 아이템을 버리도록 설정해두겠습니다.

```
void Update ()
{
    if (Input.GetButtonDown("Action") && isPlayerEnter){
        transform.SetParent(playerEquipPoint.transform);
        transform.localPosition = Vector3.zero;
        transform.rotation = new Quaternion(0,0,0,0);

        playerLogic.Pickup(gameObject);
        isPlayerEnter = false;
    }
}
```

마지막으로 아이템 스크립트에서
플레이어의 줍기 로직을 호출합니다.
이렇게해서 스크립트 로직까지 완료!

아주 완벽합니다!
아이템을 줍고 놓고~
정말 게임 같아보이네요.

이렇게하여
열일곱번째 강좌를 마무리하였습니다.

부모-자식 관계만 잘 이해하신다면
그리 어렵지 않을 것입니다.

다음은 마우스를 이용하여
투사체를 발사하는 액션에 대해 다뤄보도록 하겠습니다.
그럼.