UNITY
-CAHPTER6-

SOUL SEEK

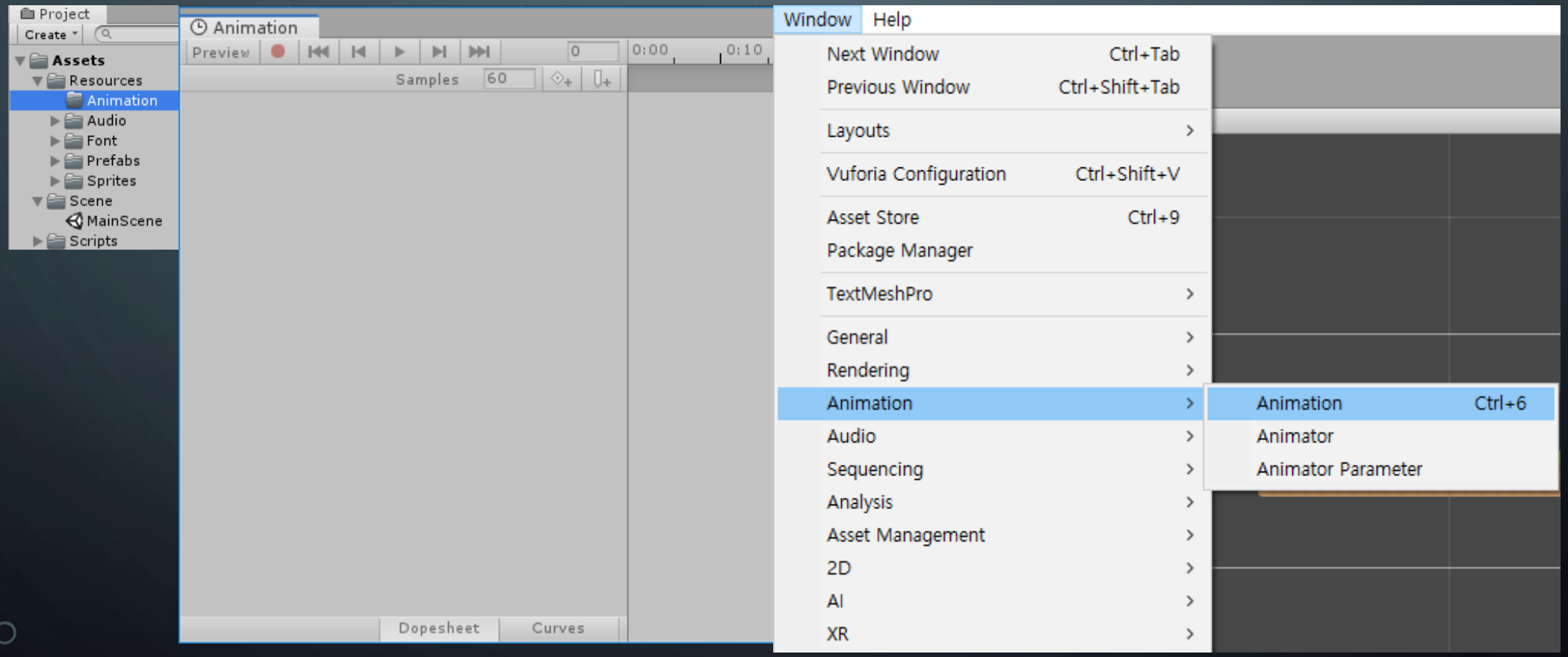
목차

캐릭터 애니메이션

1. 캐릭터 애니메이션 준비하기

애니메이션 폴더를 만들고 애니메이션을 만들 준비를 한다.

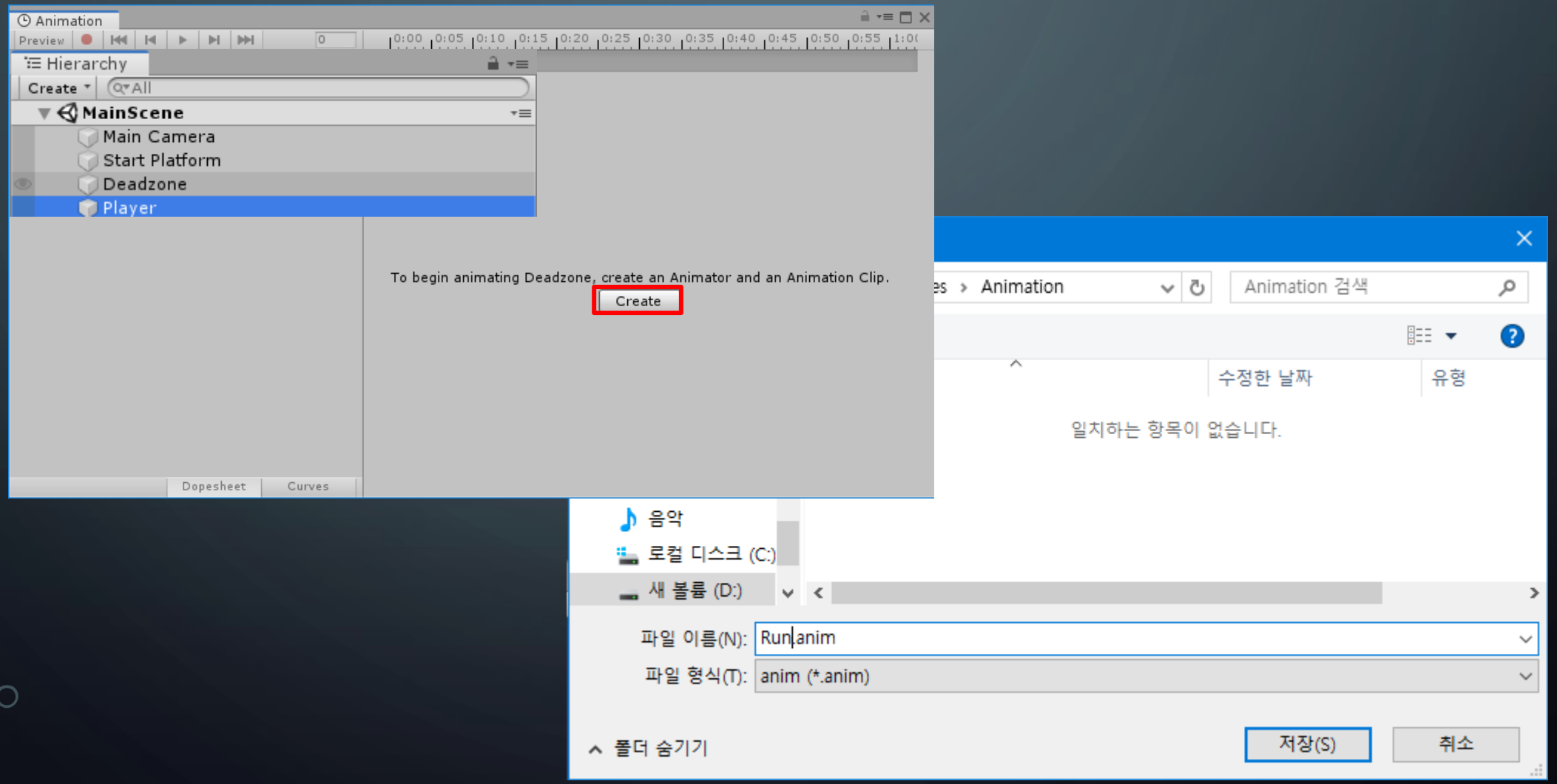
1. **Project View**의 **Resources** 폴더에 새로운 폴더 하나 생성하고 **Animations**로 변경
2. **Window > Animation > Animation** 클릭
3. 열린 **Animation View**을 적절한 곳에 배치한다.



2. 캐릭터 애니메이션 클립 만들기

애니메이션 클립 만들기

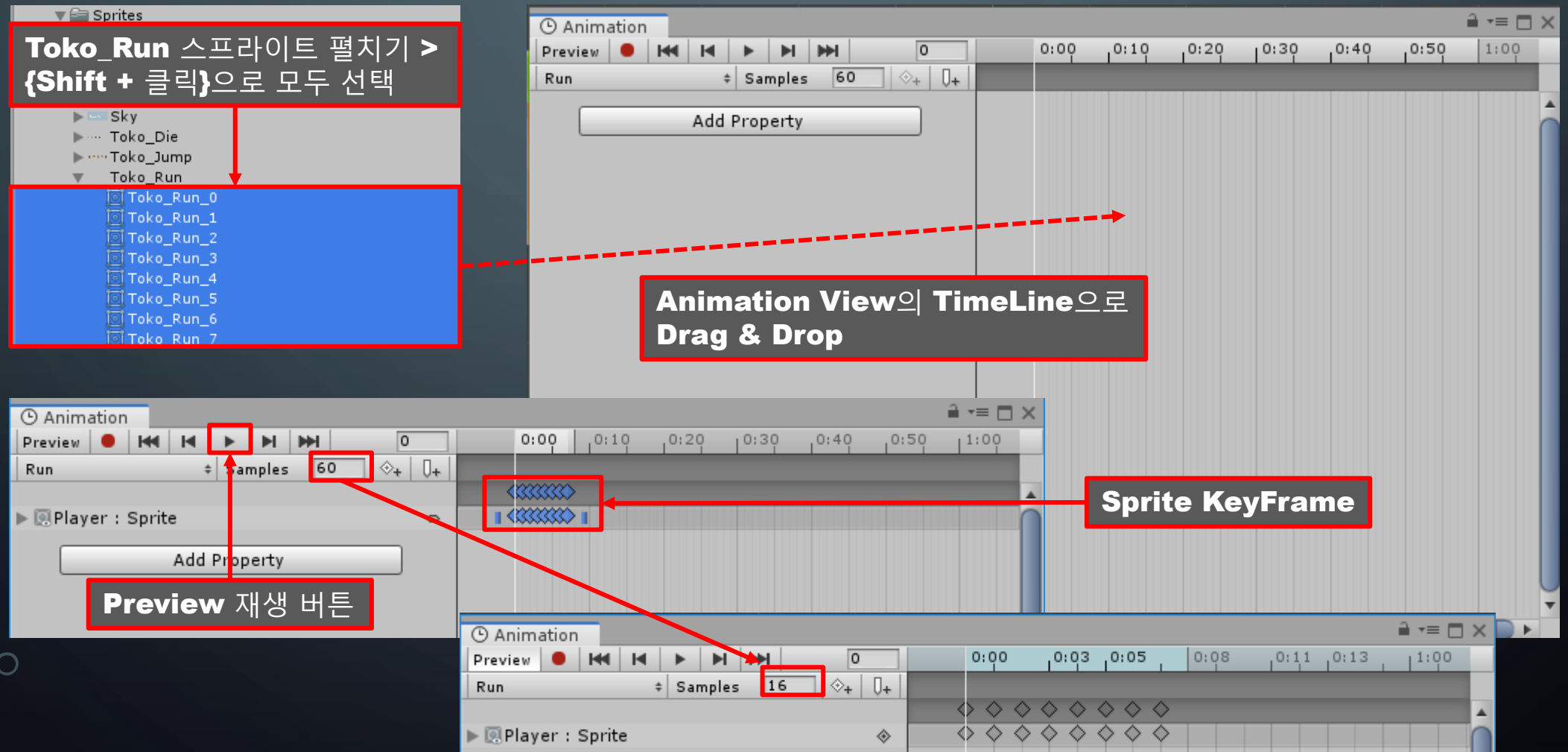
1. **Player Object**를 선택한 상태에서 **Animation View** 나타난 **Create** 버튼을 이용해 생성
2. 새로운 **Animation Clip**을 **Run**이라는 이름으로 **Assets** 폴더 내부의 **Animation** 폴더에 저장



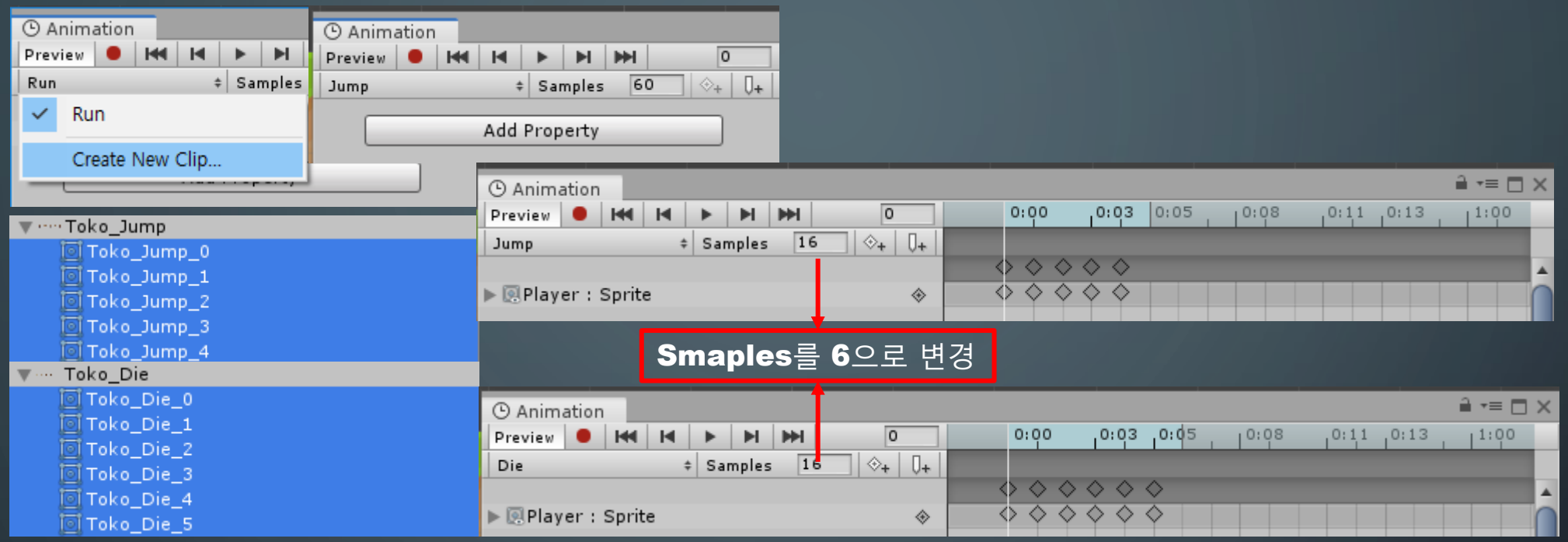
2. 캐릭터 애니메이션 클립 만들기

Run 애니메이션 클립 만들기

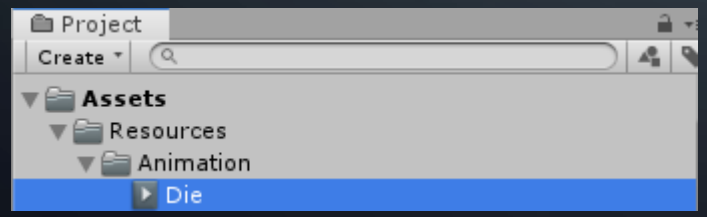
1. **ProjectView**에서 **Sprites** 폴더에 있는 **Toko_Run**의 **Multiple Sprite**들을 선택한다.
2. **Animation View**의 타임 라인으로 **Drag&Drop**한다.
3. 실행해 본 후 **Samples**의 **Frame**을 **60**에서 **16**으로 변경한다. 나머지 동작도 똑같이 작성해 보자.

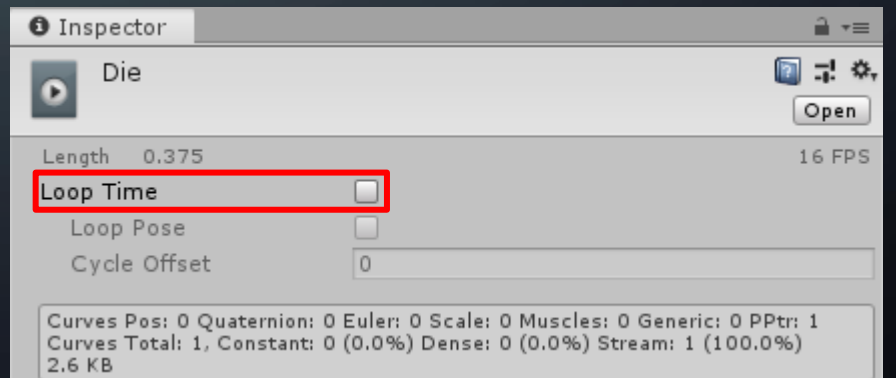


2. 캐릭터 애니메이션 클립 만들기



Die Animation은 **Loop** 반복을 하지 말아야 하기 때문에 **Die Clip**을 선택하고 **Inspector View**에서 **Loop Time** 체크를 해제하자.

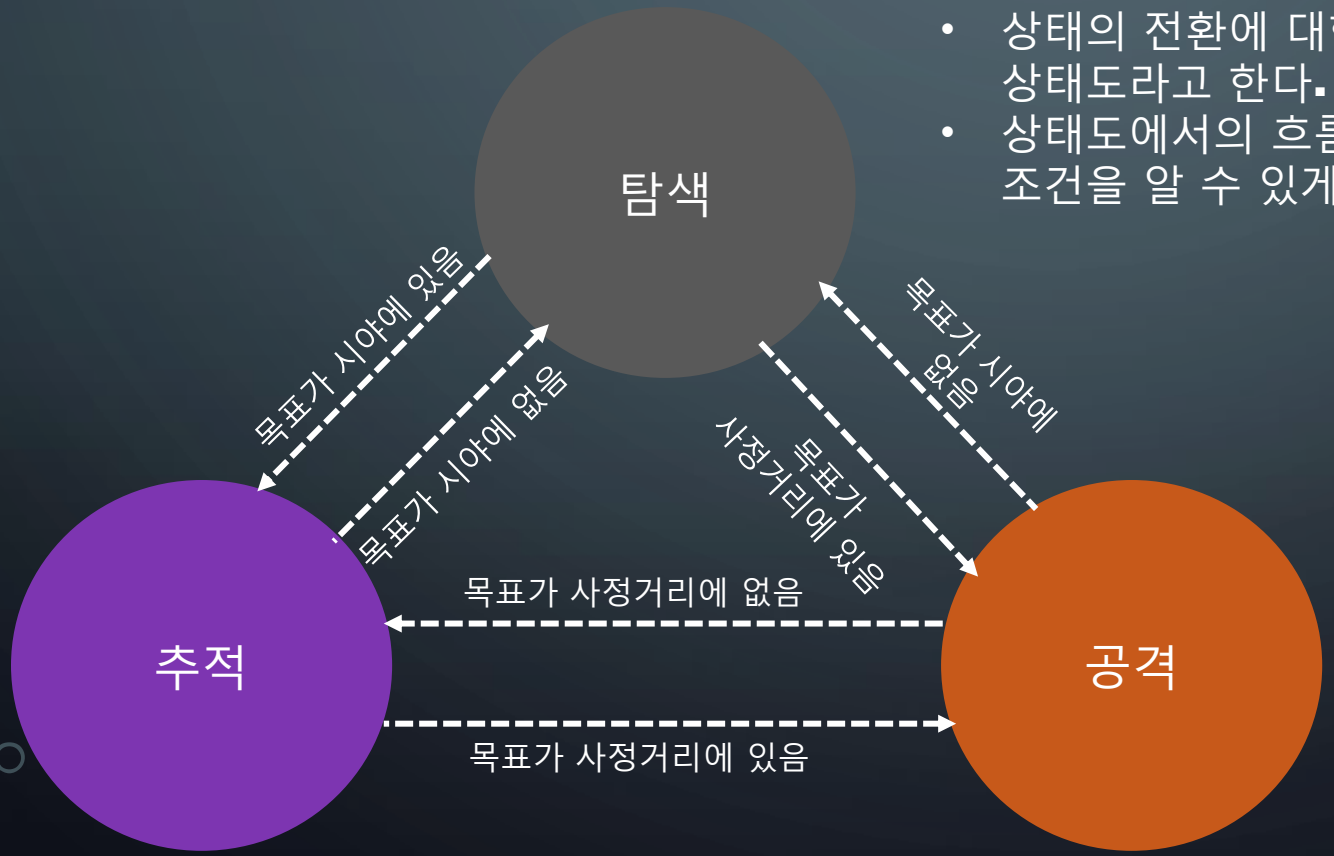




3. FSM(유한 상태 머신)

- **Unity Animator**에서 사용하고 있는 디자인 모델
- 유한한 수의 상태가 존재하며, 한 번에 한 상태만 '현재 상태'가 되도록 프로그램을 설계하는 모델.
  →어떤 상태에서 다른 상태로 전이**(Transition)**하여 현재 상태를 전환할 수 있다.
  →동시에 두 가지 상태를 가질 수 없다.
- 게임에서는 적 **AI** 설계디자인이 대표적인 예.

ex) 적 **AI**가 총 **3**가지(탐색, 추적, 공격) 상태를 가진다고 했을 때.

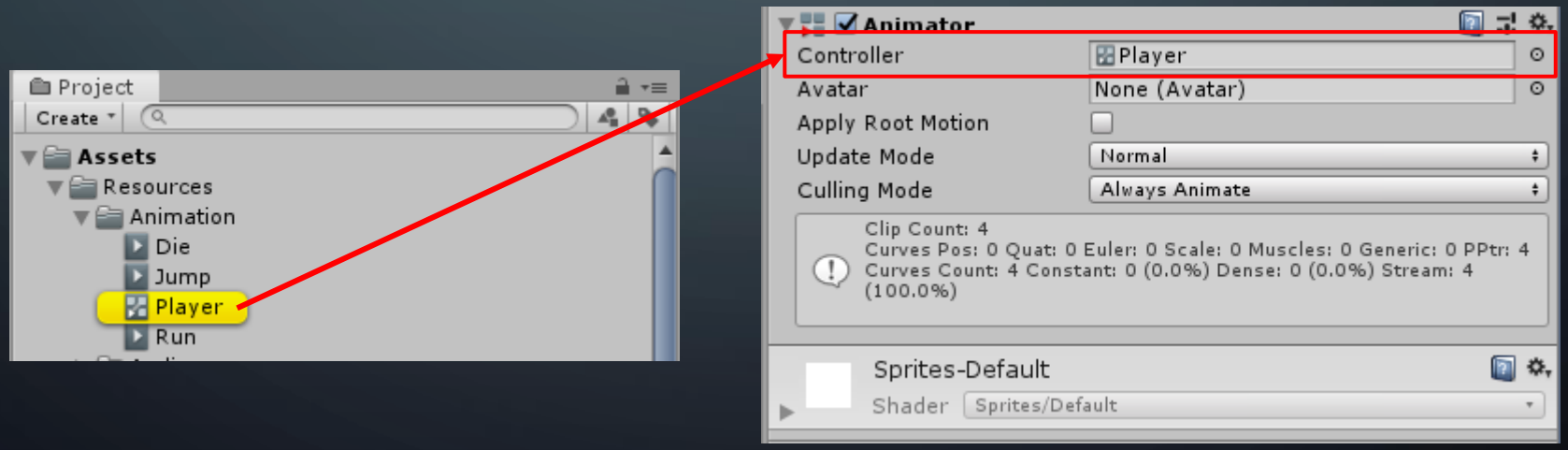


- 상태의 전환에 대한 모습을 그래픽화 한 것을 상태도라고 한다.
- 상태도에서의 흐름은 화살표로 전환 되는 상태의 관계와 조건을 알 수 있게 한다.

3. ANIMATOR & ANIMATOR CONTROLLER

- **FSM(**유한 상태 머신**)**을 사용해 재생할 **Animation**을 결정하는 상태도를 표현하는 **Default Asset**이다
- **Animator Controller**를 참고하여 **GameObject**의 **Animation**을 적용하는 **Component**가 **Animator**이다**.**
- **GameObject**의 **Animation Clip**을 생성하기 위해 **Animation Component**를 생성하면 자동적으로 이를 컨트롤러를 적용하기 위한 **Animator Controller**가 생성되고 **GameObject**에는 **Animatior Component**가 추가된다**.**
- 새로운 **Animator Controller**는 **Create > Animator Controller**로 생성할 수 있다**.**

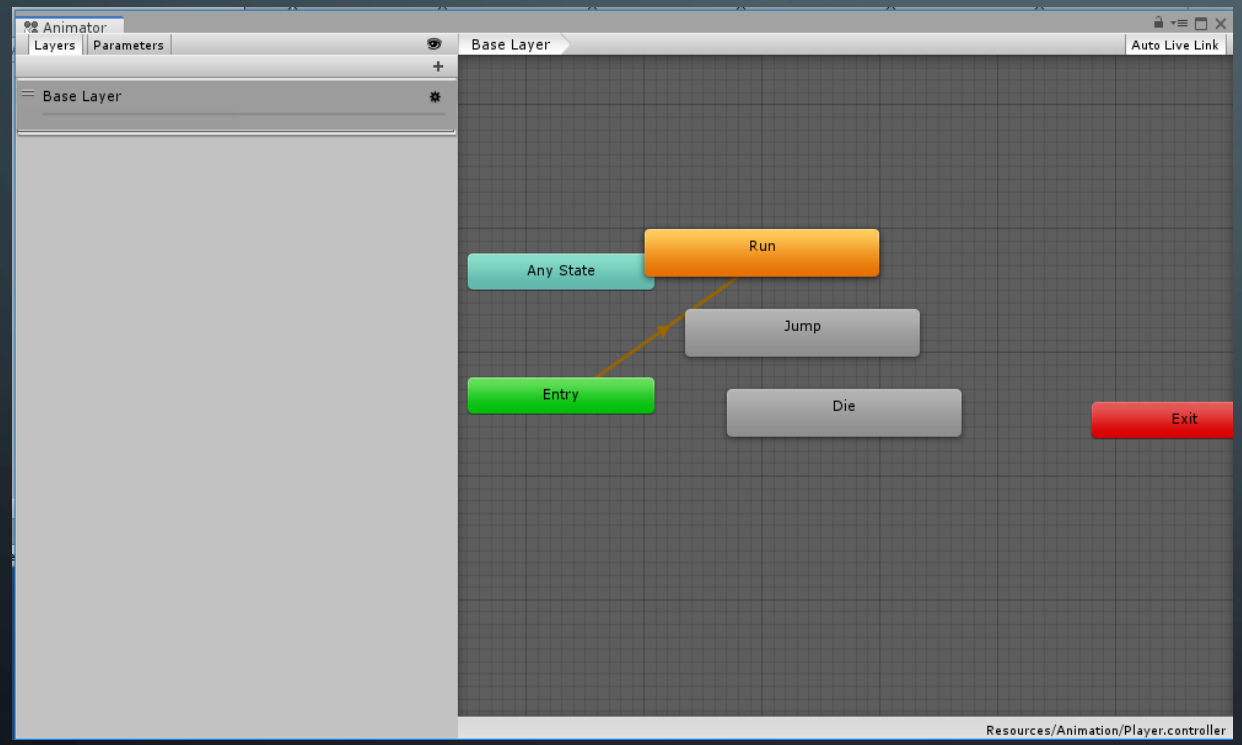
Player Animation Clip들을 가지고 있는 **Animator Controller**를 **Animator**에 추가해서 **Animator**의 전이**(Transition)**을 구성할 준비를 하자**.**



3. ANIMATOR & ANIMATOR CONTROLLER

Player의 **Animator** 작성

1. **Player Object**를 선택한 상태에서 **Window > Animation > Animator** 클릭
2. 열린 **Animator View**을 적절한 위치로 **Drag**하여 배치
3. **Scene View**와 똑같이 마우스 조작 컨트롤이 가능하다.



기본 포함된 상태

Entry : 현재 상태가 진입하는 '입구'
Exit : 상태 머신의 동작이 종료되는 '출구'
Any State : 현재 상태가 무엇이든 상관없이 특정 상태로 즉시 전이하게 허용하는 상태

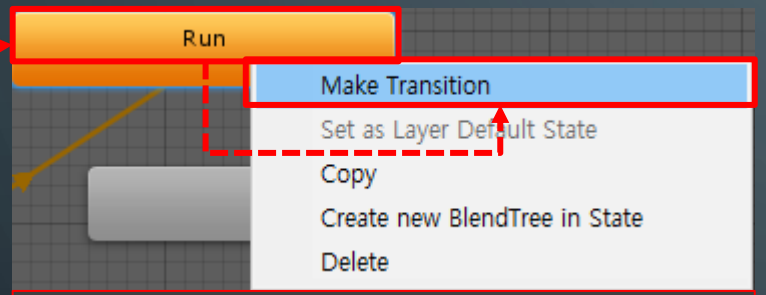
직접 추가한 상태

Run : Run Animation Clip 재생
Jump : Jump Animation Clip 재생
Die : Die Animation Clip 재생

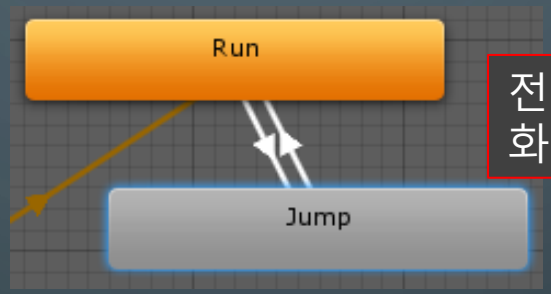
3. ANIMATOR & ANIMATOR CONTROLLER

전이(Transition)구성하기

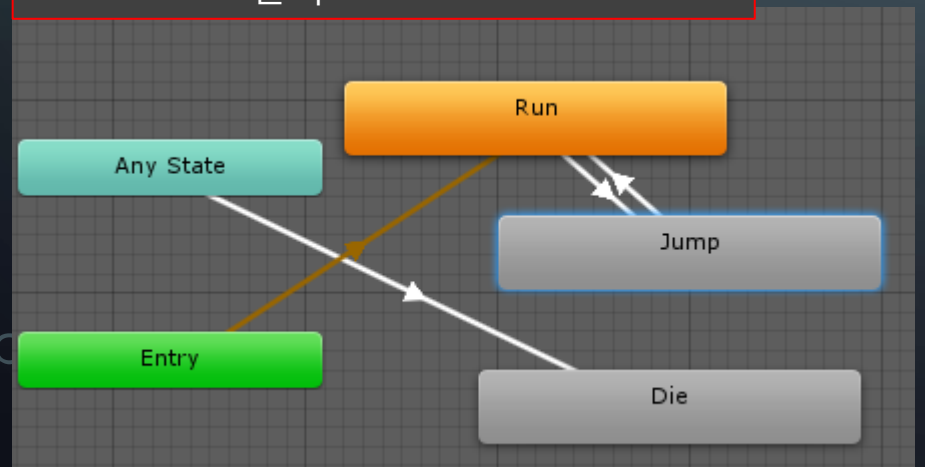
1. **Run**상태에서 마우스 오른쪽 클릭 **> Make Transition** 클릭**,** 전이 화살표를 **Jump** 상태에 연결(전이 화살표를 끌어다 **Jump** 상태에 클릭)
2. **Jump** 상태에서 똑같이 **Run**으로 연결
3. **Any State** 상태에서 마우스 오른쪽 클릭 **> Make Transition** 클릭**, Die**와 연결
   → **Die** 상태는 중간 전이와 상관없이 모든 상태를 중지하고 강제로 변환되어야 하기 때문에 **Any State**와 연결**.**



Run상태에 마우스 오른쪽 클릭 **> Make Transition** 클릭



전이 화살표를 **Jump** 상태에 연결(전이 화살표를 끌어다 **Jump** 상태를 클릭)

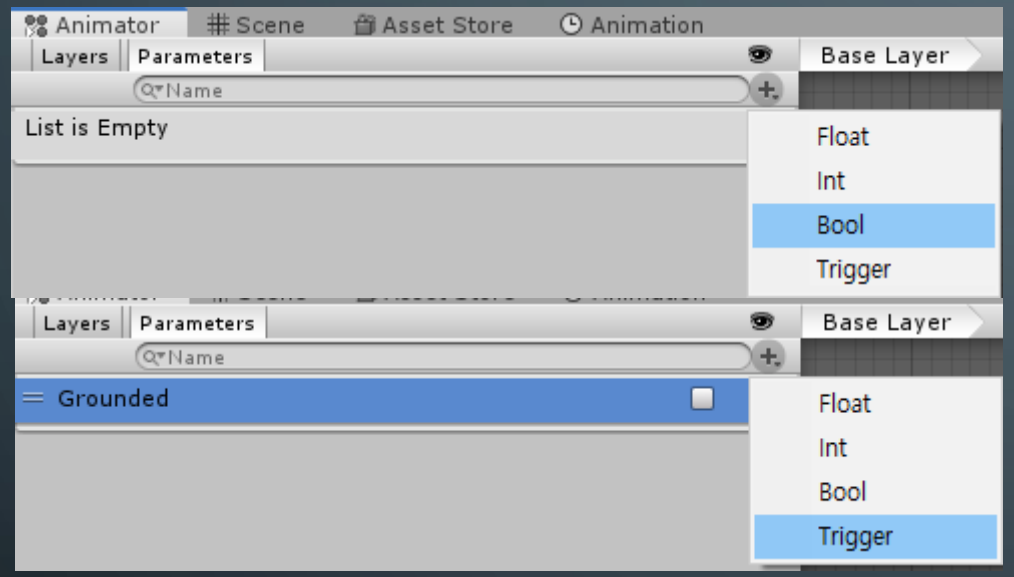


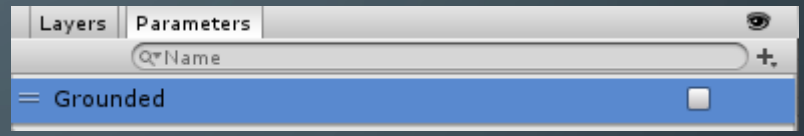
다른 전이 상태도 연결해 주자**.**

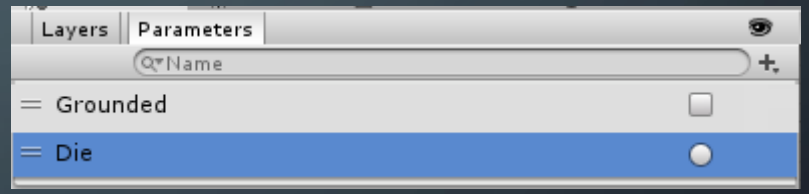
3. ANIMATOR & ANIMATOR CONTROLLER

전이(Transition) 조건 Parameter 추가

1. **Animator View**에서 **Parameters Tab** 클릭
2. **+** 버튼을 클릭 **> Bool** 클릭 **>** 생성된 파라미터 이름을 **Grounded**로 변경
3. **+** 버튼을 클릭 **> Trigger** 클릭 **>** 생성된 파라미터 이름을 **Die**로 변경







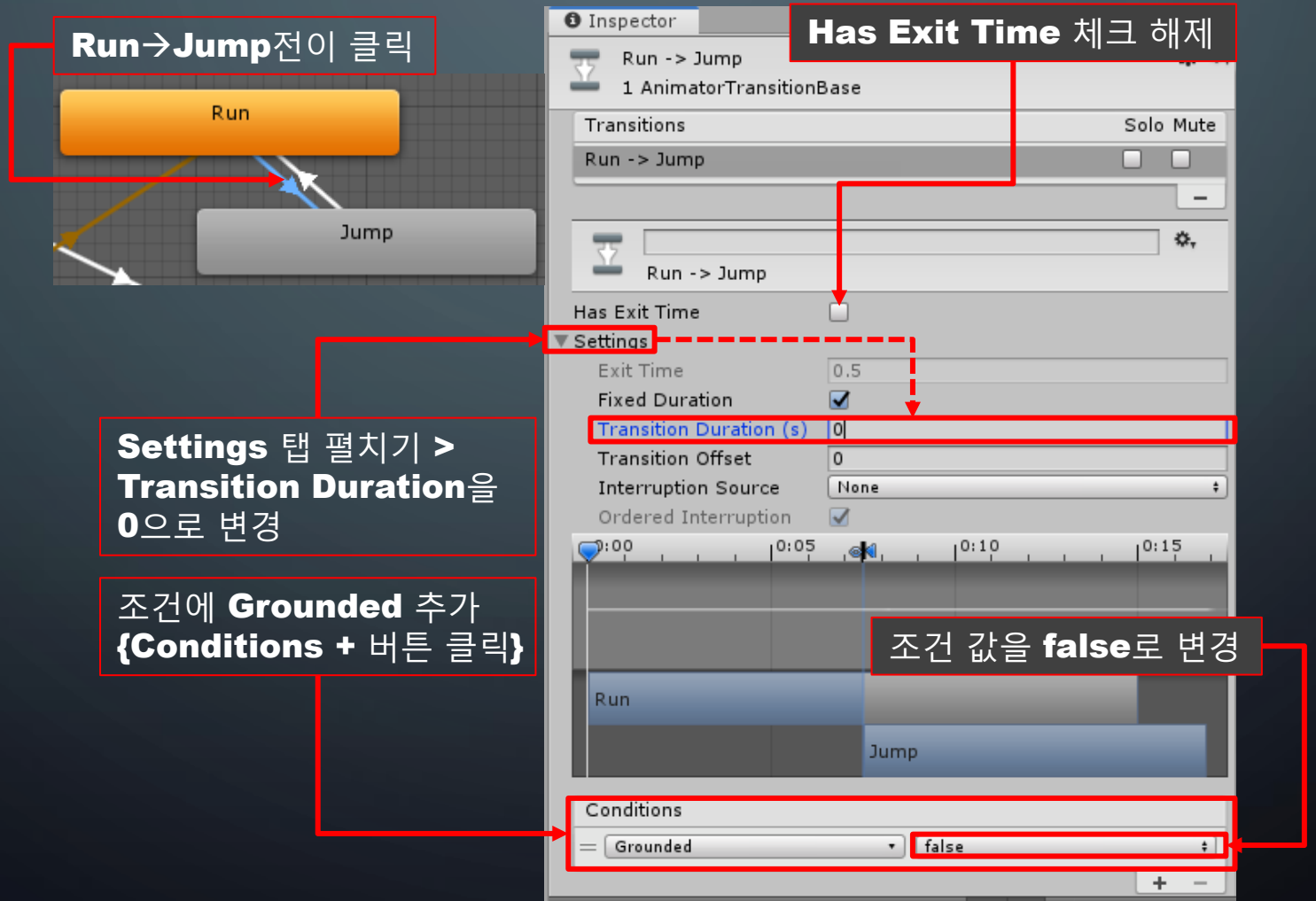
Parameter

- 전이 조건으로 사용할 수 있는 수치**,** 실수**(float),** 정수**(int),** 불리언**(bool),** 트리거**(trigger)**가 있다**.**
- **bool** 타입은 **true, false** 상태를 지정해서 해당상태에서 조건이 만족하면 전이하게 된다**.**
- **trigger** 타입은 특정 값의 할당 없이 현재 상태의 트리거 조건을 지정하는 순간 전이하게 된다**.**

3. ANIMATOR & ANIMATOR CONTROLLER

Run → Jump 전이 설정

1. **Animator View**에서 **Run → Jump** 전이 클릭, **Inspector View**에서 **Has Exit Time** 체크 해제
2. **Settings** 에서 **> Transition Duration**을 **0**으로 변경
3. 조건에 **Grounded** 추가(**Conditions**의 **+** 클릭), **Grounded**의 조건 값을 **false**로 변경



3. ANIMATOR & ANIMATOR CONTROLLER

Has Exit Time

종료 시점을 활성화 한다.
→활성화된 종료 시점의 값은 **Exit Time**필드에서 변경할 수 있다.
→종료 시점이란 전이에서 현재 상태를 탈출하여 다음 상태로 넘어가는 시점이다.
체크 : 활성화, 체크 해제 : 비활성화
→활성화 되어있으면 종료 시점이 존재하게 되어 전이의 조건이 만족해도 즉시 다음 상태로 전이하지 않고 한 루프의 **Animation**이 끝났을 때 전이 하고 비활성화시 그와 반대로 즉시 전이 된다.

Transition Duration

전환 지속 시간은 전이가 이루어지는 동안 현재 **Animation Clip**과 다음 **Animation Clip**을 섞어서 **(Blending)** 부드럽게 이어주는 역할
3D Bone Animation에서 뛰다가 점프하려고 동작을 바꾸는 것을 자연스럽게 연결해준다.
→**2D**는 이런 블렌딩이 필요 없기 때문에 **0**초로 한다.

나머지 전이 상황도 설정해 보자.

4. PLAYERCONTROLLER SCRIPT

Script를 작성하는데 필요한 변수는 미리 준비해두었다.

```
public AudioClip deathClip; // 사망시 재생할 오디오 클립
public float jumpForce = 700f; // 점프 힘
```

```
private int jumpCount = 0; // 누적 점프 횟수
private bool isGrounded = false; // 바닥에 닿았는지 나타냄
private bool isDead = false; // 사망 상태
```

```
private Rigidbody2D playerRigidbody; // 사용할 리지드바디 컴포넌트
private Animator animator; // 사용할 애니메이터 컴포넌트
private AudioSource playerAudio; // 사용할 오디오 소스 컴포넌트
```

Start() Method에서 **Component**에 대한 할당을 하자.

```
private void Start()
{
    // GameObject로부터 사용할 Component들을 가져와 변수에 할당
    playerRigidbody = GetComponent<Rigidbody2D>();
    animator = GetComponent<Animator>();
    playerAudio = GetComponent<AudioSource>();
}
```

4. PLAYERCONTROLLER SCRIPT

Update() isDead처리를 위한 구문을 만들자**.**

```
private void Update()
{
    if(isDead)
    {
        //사망 시 처리를 더 이상 진행하지 않고종료
        return;
    }
```

마우스 왼쪽 클릭을 통해 점프를 구현하고 오디오를 플레이한다**.**

- 점프 사이에 충분한 시간 간격을 두고 이단 점프를 실행**(**마우스 왼쪽 버튼을 두 번 클릭**)**
- 매우 짧은 간격으로 이단 점프 실행**(**마우스 왼쪽 버튼을 빠르게 두 번 클릭**)**

```
//마우스 왼쪽 버튼을 눌렀으며 && 최대 점프 횟수(2)에 도달하지 않았다면
if(Input.GetMouseButtonDown(0) && jumpCount < 2)
{
    //점프 횟수 증가
    jumpCount++;
    //점프 직전에 속도를 순간적으로 제로(0, 0)로 변경
    playerRigidbody.velocity = Vector2.zero;
    //리지드바디에 위쪽으로 힘 주기
    playerRigidbody.AddForce(new Vector2(0, jumpForce));
    //오디오 소스 재생
    playerAudio.Play();
}
```

4. PLAYERCONTROLLER SCRIPT

마우스 좌클릭을 했지만 조건에 맞으면 **Gounded Parameter**를 변경해서 **Jump**상태로 전이하자.

```
else if(Input.GetMouseButton(0) && playerRigidbody.velocity.y > 0)
{
    //마우스 왼쪽 버튼에서 손을 떼는 순간 && 속도의 y값이 양수라면(위로 상승 중)
    //현재 속도를 절반으로 변경
    playerRigidbody.velocity = playerRigidbody.velocity * 0.5f;
}
//애니메이터의 Grounded 파라미터를 isGrounded 값으로 갱신
animator.SetBool("Grounded", isGrounded);
```

Input.GetMouseButtonDown() : 마우스 버튼을 '누르는 순간'
Input.GetMouseButton() : 마우스 버튼을 '누르고 있는 동안'
Input.GetMouseButtonUp() : 마우스 버튼에서 '손을 떼는 순간'

If ~ else if() 구문에서 조건은..

조건 **Input.GetMouseButtonDown(0) && jumpCount < 2**의 결과가 **false**
Input.GetMouseButtonUp(0) : 마우스 왼쪽 버튼에서 손을 때는 순간
playerRigidbody.velocity.y > 0 : y 방향 속도가 **0**보다 큼

Parameter 관리
SetBool(string name, bool value);
SetInt(string name, int value);
SetFloat(string name, float value);
SetTrigger(string name);

4. PLAYERCONTROLLER SCRIPT

Die() Method를 만들어 보자.

1. 죽었을 때 **Trigger**로 **Die** 상태 전이를 적용하자.
2. 죽었을 때 할당된 오디오 클립을 적용해서 사운드를 플레이 하자.
3. 죽은 상황에서 물리작용을 더 받지 않아도 되기 때문에 속도 값을 초기화 하자.
4. **Dead** 상태를 활성화 하자.

```
private void Die()
{
    //애니메이터의 Die 트리거 파라미터를 셋팅한다.
    animator.SetTrigger("Die");

    //오디오 소스에 할당된 오디오 클립을 deathClip으로 변경
    playerAudio.clip = deathClip;
    //사망 효과음 재생
    playerAudio.Play();

    //속도를 제로(0, 0)로 변경
    playerRigidbody.velocity = Vector2.zero;
    //사망 상태를 true로 변경
    isDead = true;
}
```

4. PLAYERCONTROLLER SCRIPT

OnTriggerEnter2D() Method를 이용해 **Die** 체크 이벤트만 발생 시키자.

```
private void OnTriggerEnter2D(Collider2D other)
{
    if(other.tag == "Dead" && !isDead)
    {
        //충돌한 상대방의 태그가 Dead이며 아직 사망하지 않았다면 Die() 실행
        Die();
    }
}
```

점프를 시도할 때 이미 지면이랑 충돌 중이라는 걸 가정하고 빠져 나갈 때와 다시 충돌할 때를 체크하자.

```
private void OnCollisionEnter2D(Collision2D collision)
{
    //어떤 콜라이터와 닿았으며, 충돌 표면이 위쪽을 보고 있으면
    if(collision.contacts[0].normal.y > 0.7f)
    {
        // isGround를 true로 변경하고, 누적 점프 횟수를 0으로 리셋
        isGrounded = true;
        jumpCount = 0;
    }
}
```

```
private void OnCollisionExit2D(Collision2D collision)
{
    //어떤 콜라이더에서 빠져나온 경우 isGround를 false로 변경
    isGrounded = false;
}
```

4. PLAYERCONTROLLER SCRIPT

OnCollision 계열의 충돌 이벤트에 **Normal Vector**값을 사용하자**!**

여러 충돌 정보를 담는 **Collision Type**의 **Data**를 입력 받는다**.**
→충돌 지점들의 정보를 담는 **ConstactPoint Type**의 **Data**를 **constacts**라는 배열 변수로 제공한다**.**
→**constacts** 배열의 길이는 충돌 지점의 개수와 일치한다**.**

앞선 코드에서 **collision.contacts[0]**은 두 물체 사이의 여러 충돌지점 중에서 첫 번째 충돌 지점의 정보를 가져온 것이다**. ContactPoint**와 **ContactPoint2D Type**은 충돌 지점에서 충돌 표면의 방향 벡터**(**노멀 벡터**)**을 알려주는 **normal**을 제공한다**.**
→방향을 직관적으로 알려주고 있기 때문에 적절한 값으로 비스듬하게 충돌되는지 여부도 판별할 수 있다**.**