Unity – FSM

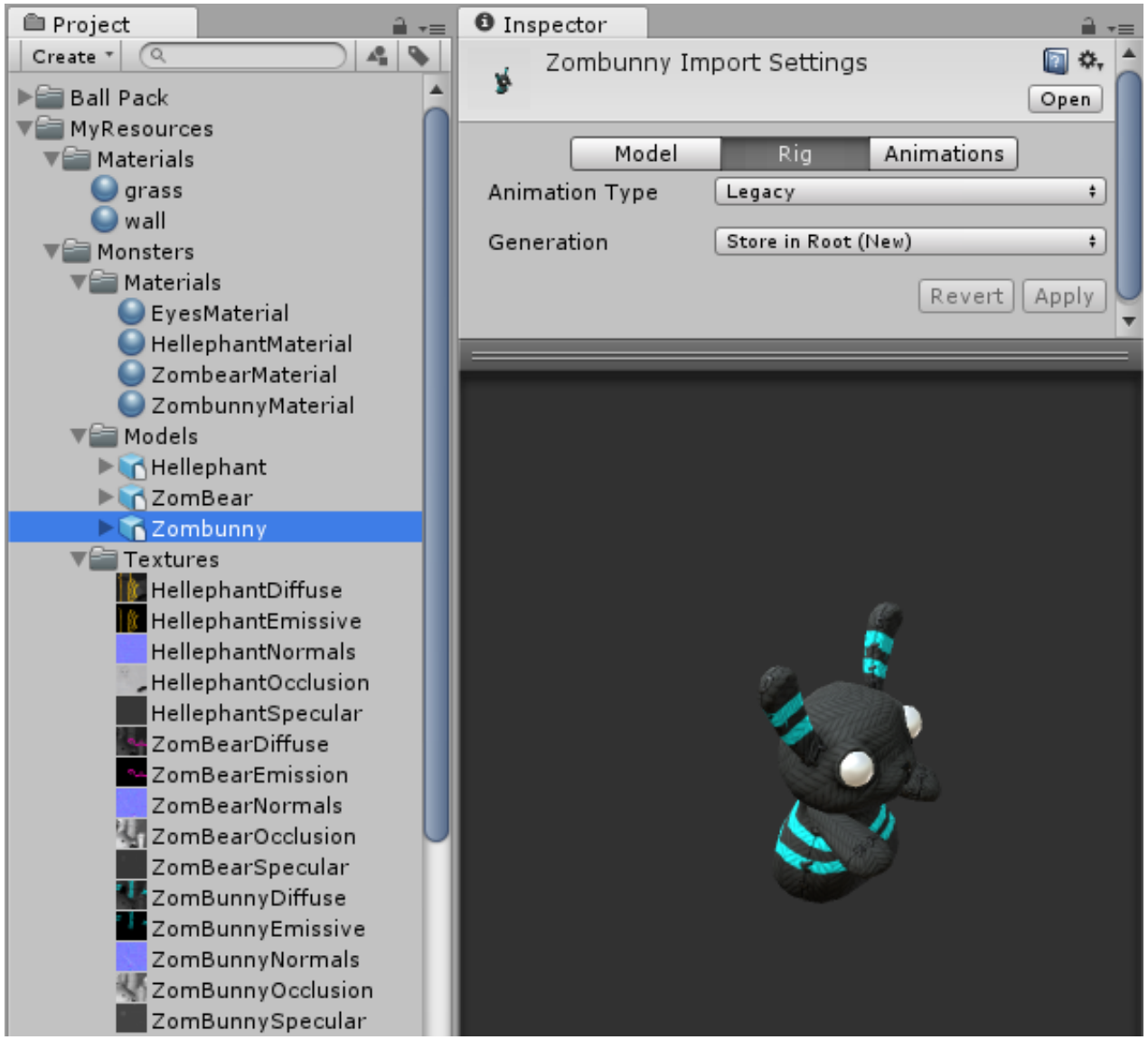
NHN NEXT
서형석

Unity

- Asset Import
  : Zombie Bunny

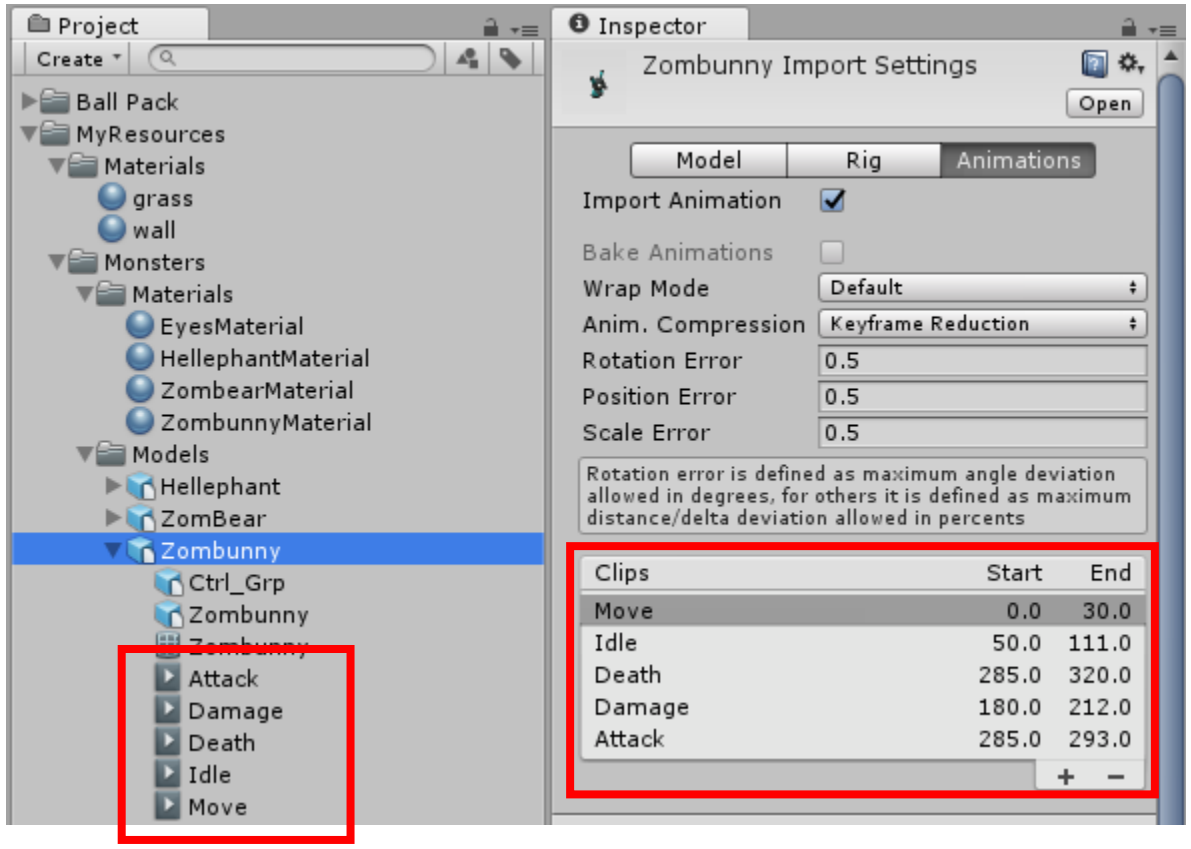
Unity

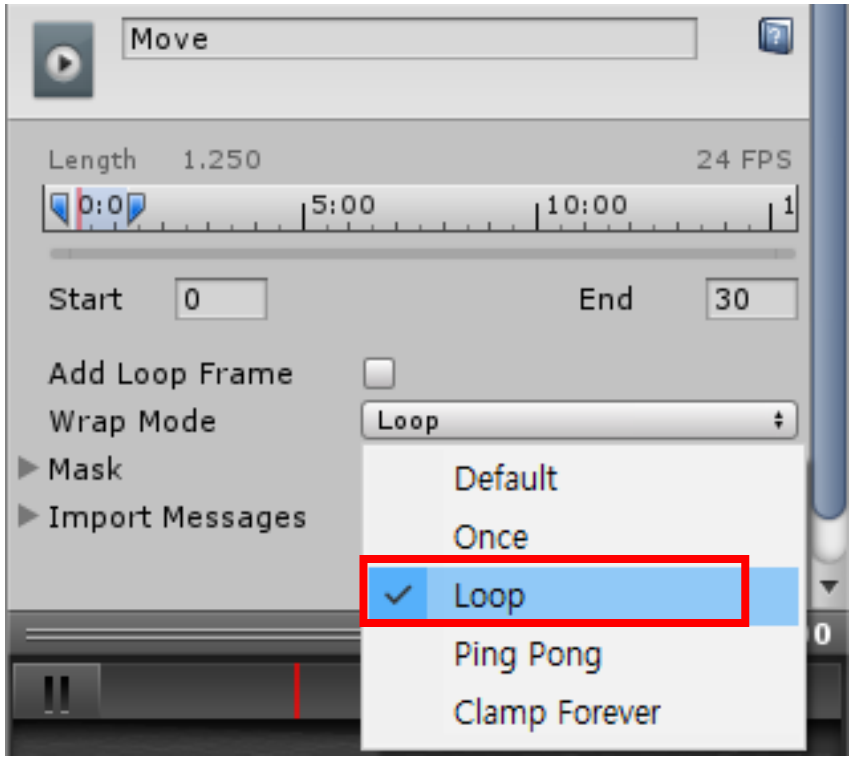
- Animation 확인



Unity

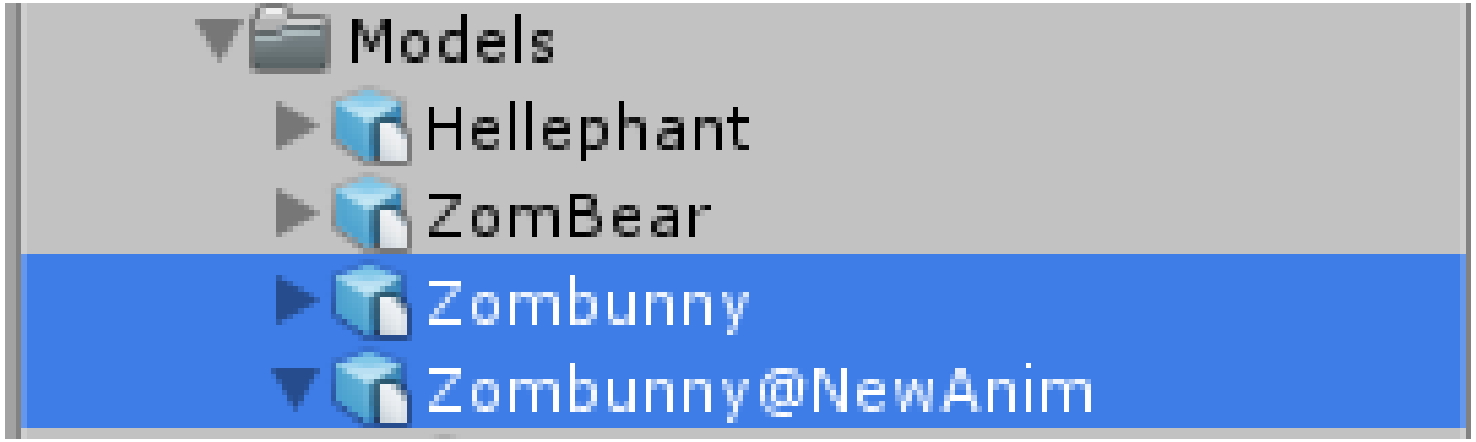
- ZombieBunny 애니메이션 확인





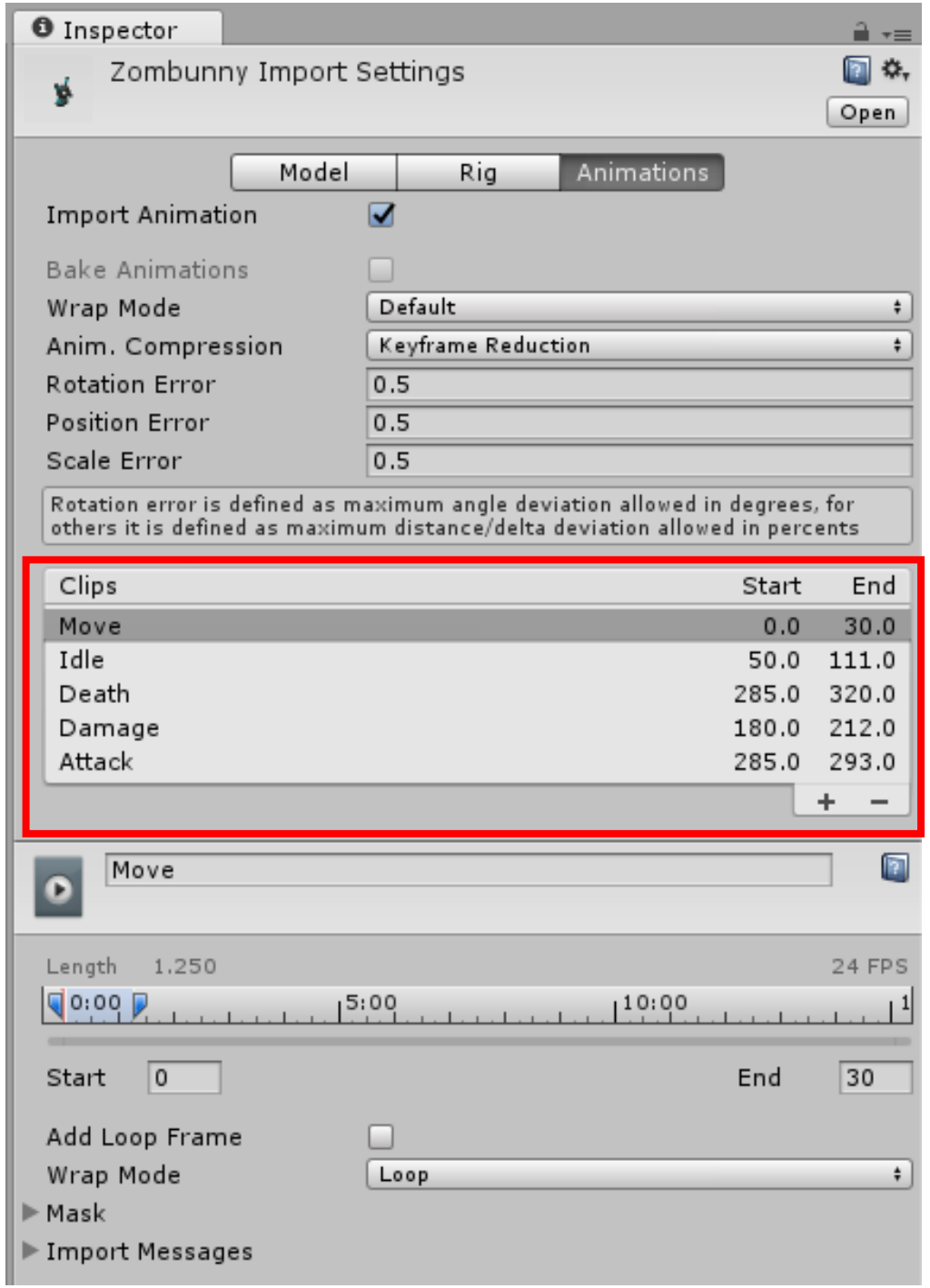
Unity

- 3D Object Import setting
  : Animation Type



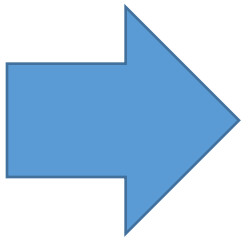
Unity

- Animation clip 확인
- Wrap Mode



Unity

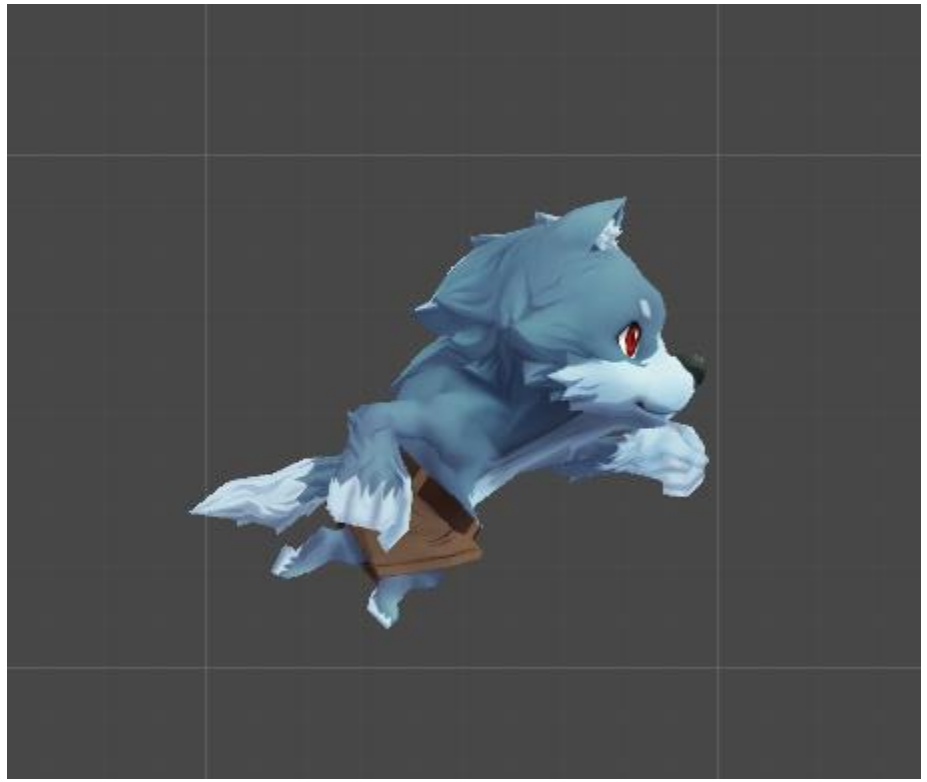
Idle



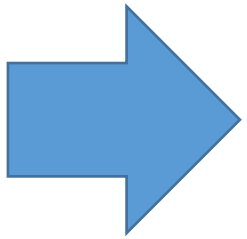
Idle -> Run

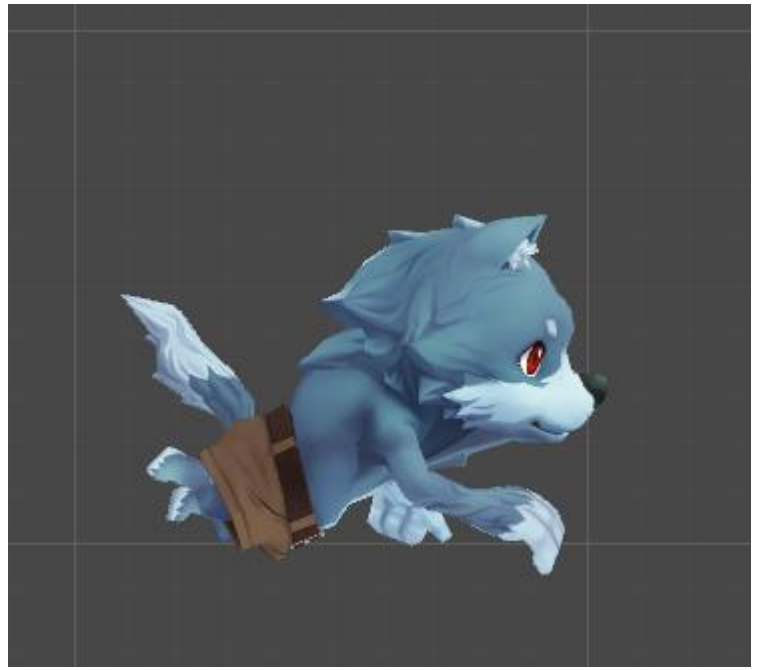
CrossFade

Unity



Run

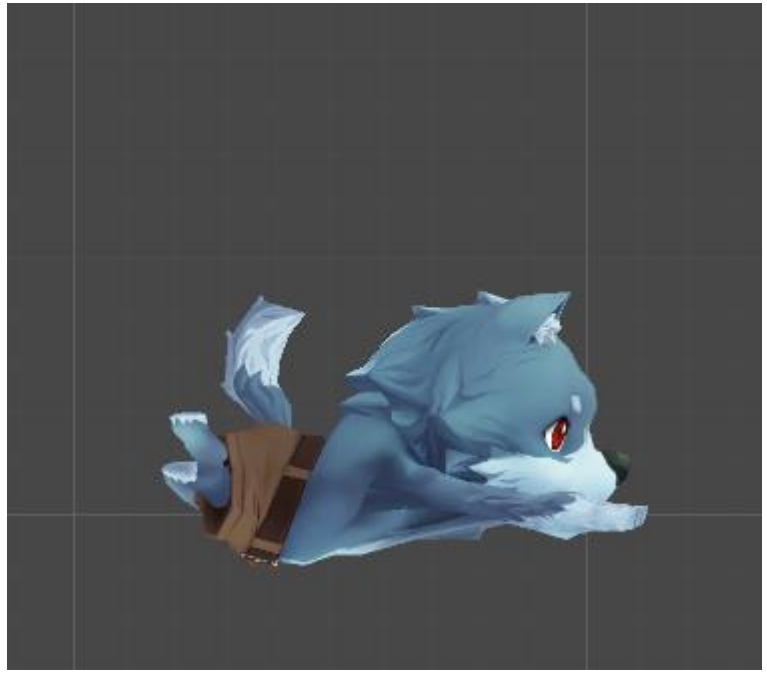




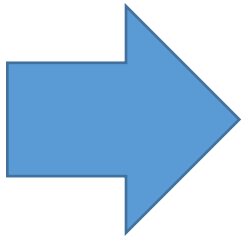
Run -> Sliding

CrossFade

Unity



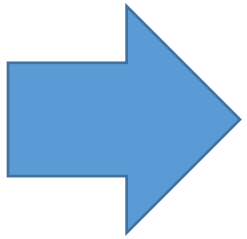
Sliding

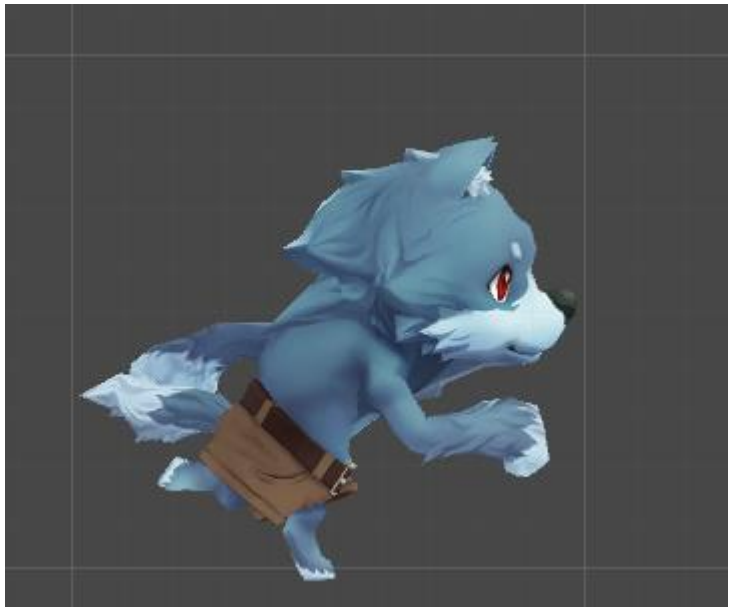


Sliding -> Run

CrossFade

Unity





Sliding

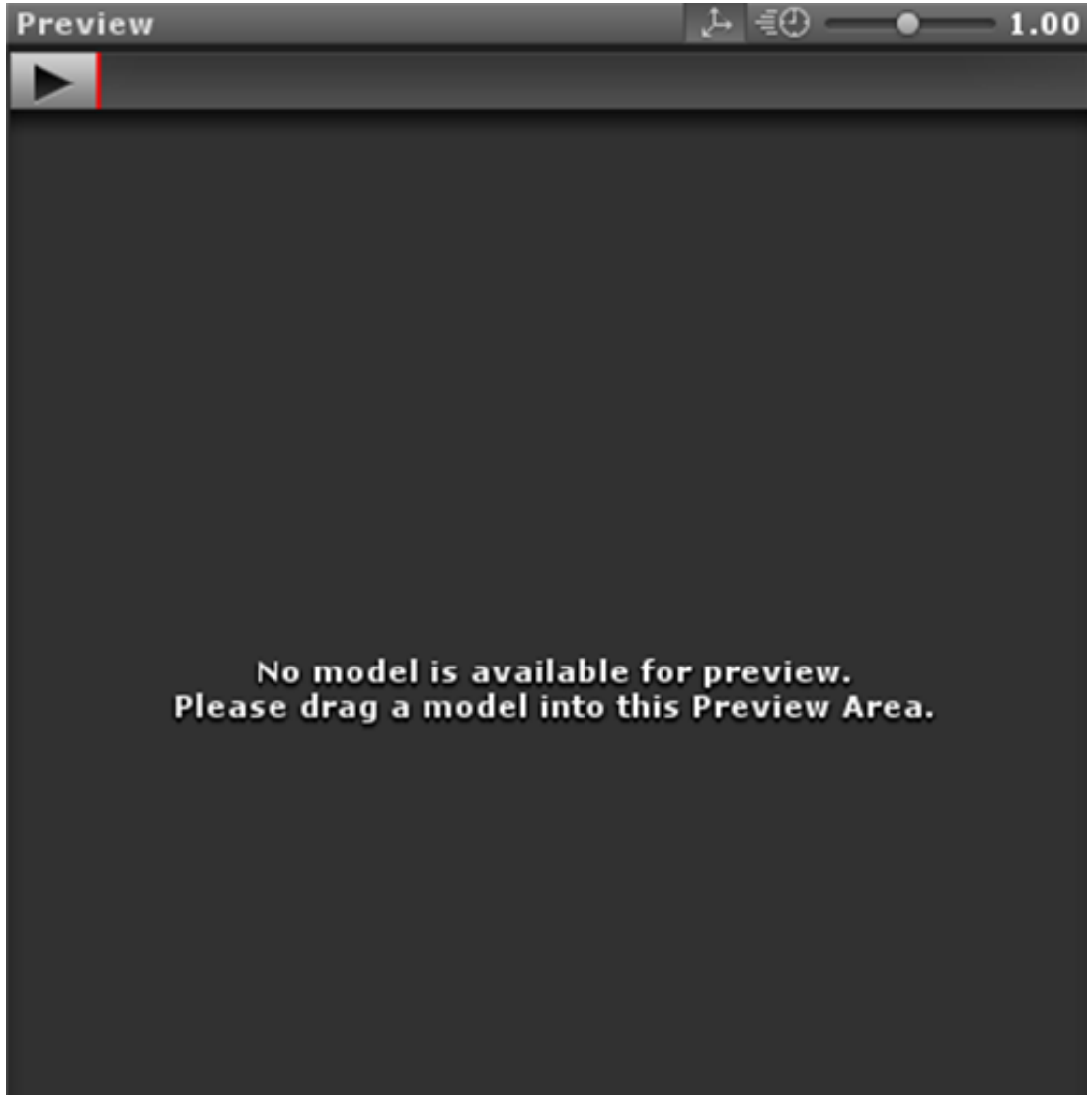
Sliding -> Run

CrossFade

Unity

- Clip 미리보기가 보이지 않는 경우

  : 애니메이션이 가능한 모델(fbx파일) 또는 prefab 파일을 연결

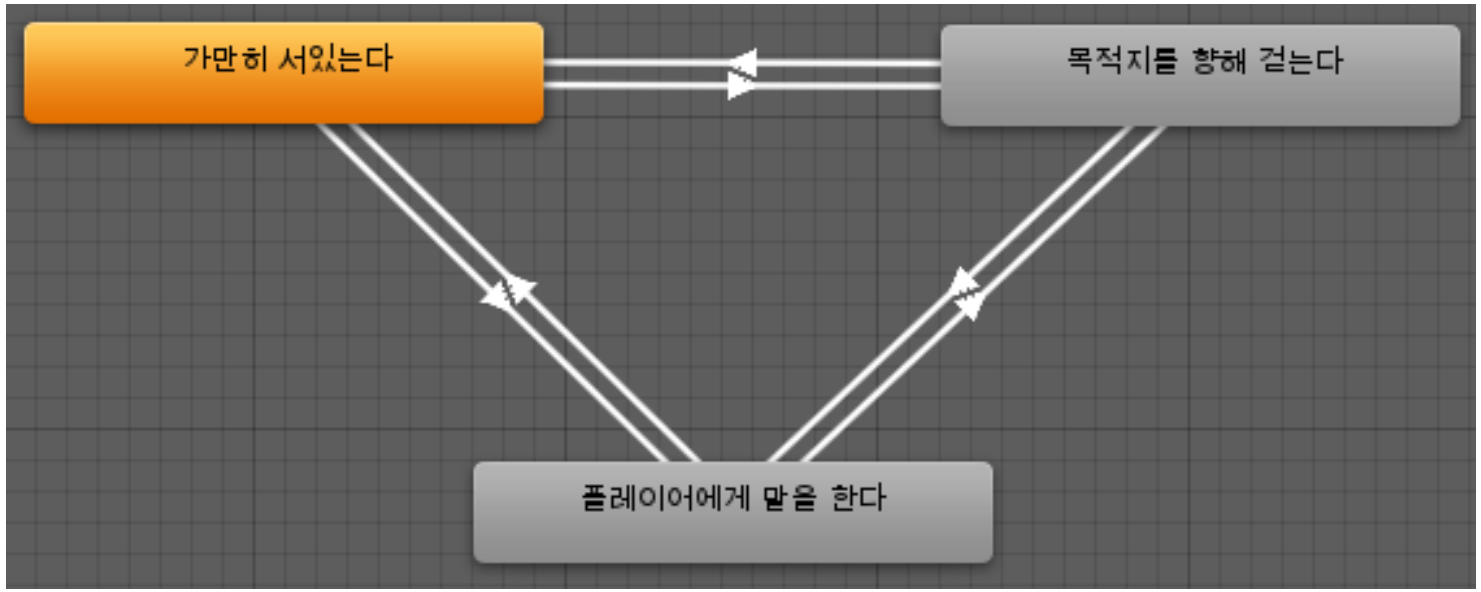


Unity

- 애니메이션에 맞는 Wrap Mode 설정이 필요

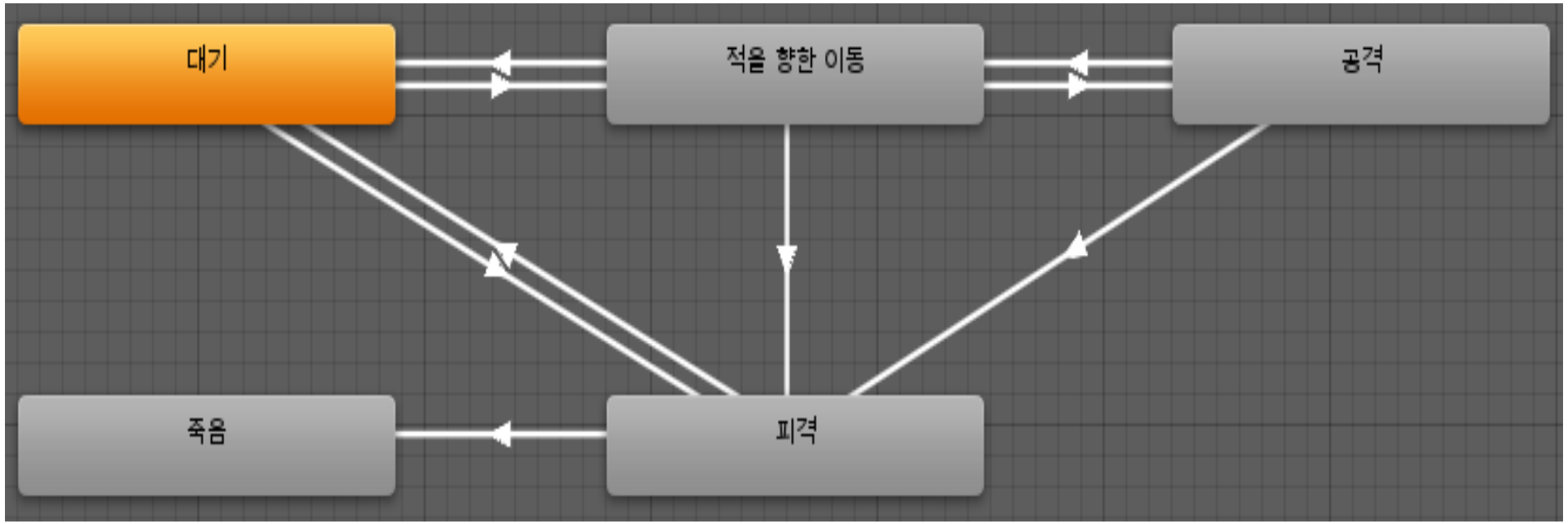
Unity

- 마을 NPC의 FSM 예시



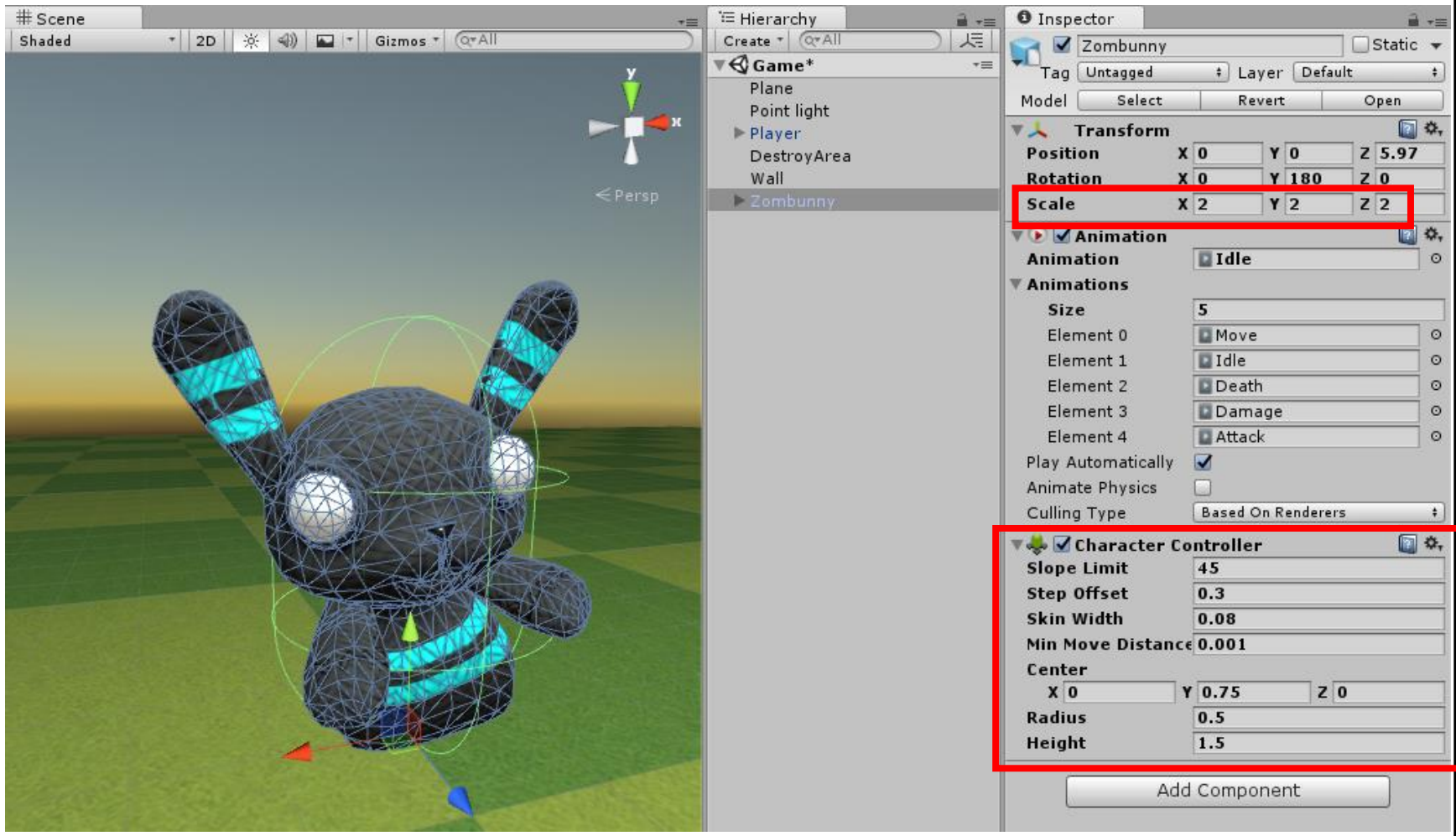
Unity

- 좀비의 FSM



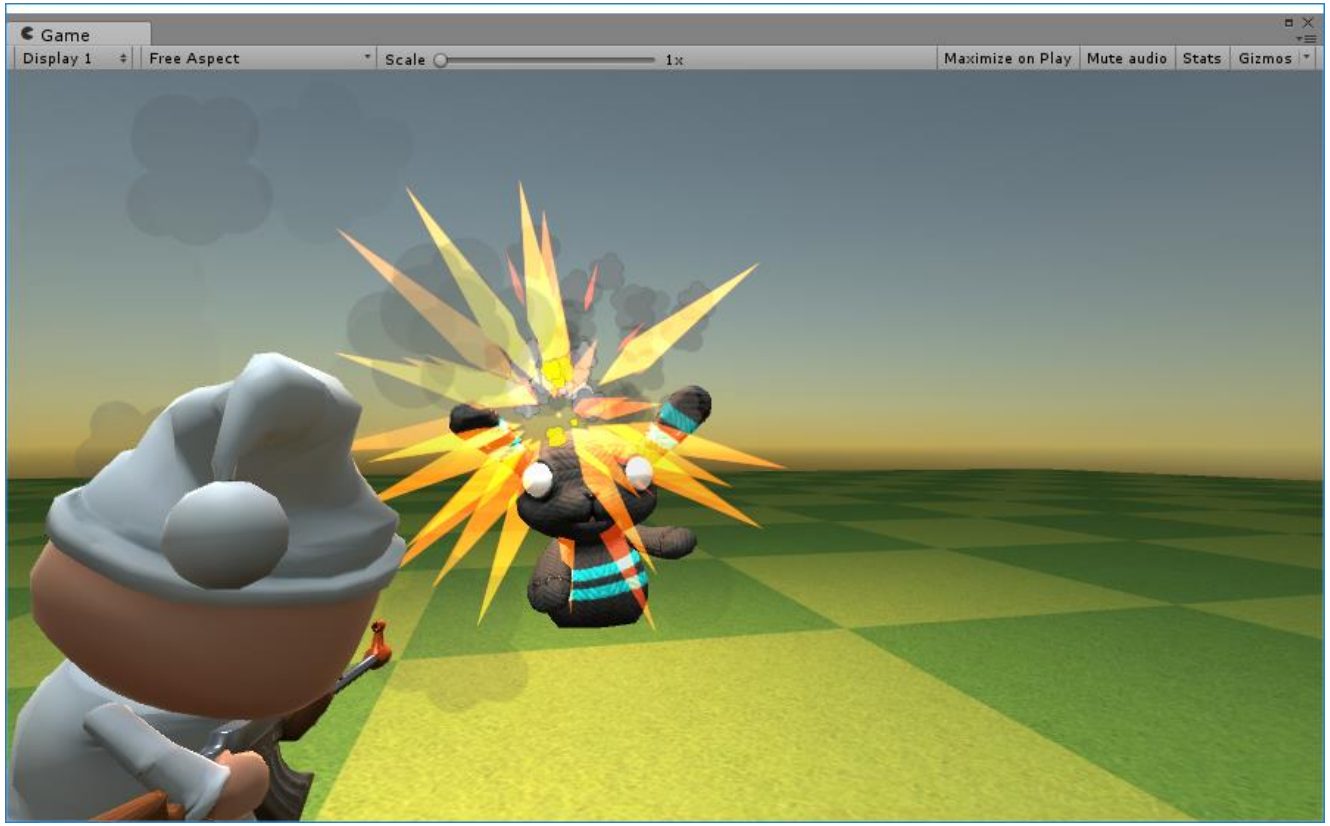
Unity

- CharacterController 추가



Unity

- 충돌영역 테스트



```
public class Zombie : MonoBehaviour
{
    enum  ENEMYSTATE
    {
        IDLE = 0,
        MOVE,
        ATTACK,
        DAMAGE,
        DEAD
    }

    ENEMYSTATE   enemyState   =   ENEMYSTATE.IDLE;
}
```

```
void Update()
{
    switch (  enemyState  )
    {
        case ENEMYSTATE.IDLE:
            {
            }
            break;
        case ENEMYSTATE.MOVE:
            {
            }
            break;
        case ENEMYSTATE.ATTACK:
            {
            }
            break;
        case ENEMYSTATE.DAMAGE:
            {
            }
            break;
        case ENEMYSTATE.DEAD:
            {
            }
            break;
    }
}
```

```
void Update()
{
    switch (  enemyState  )
    {
        case ENEMYSTATE.IDLE:
            {
            }
            break;
        case ENEMYSTATE.MOVE:
            {
            }
            break;
        case ENEMYSTATE.ATTACK:
            {
            }
           break;
       case ENEMYSTATE.DAMAGE:
            {
            }
            break;
       case ENEMYSTATE.DEAD:
            {
            }
            break;
    }
}
```

상태별 처리 구조 구현

```
public class Zombie : MonoBehaviour
{
    .....
     float    stateTime = 0.0f;
     public  float  idleStateMaxTime = 2.0f;
     public  Animation   anim;

     void  Awake()
     {
          InitZombie();
     }

    void InitZombie()
    {
        enemyState = ENEMYSTATE.IDLE;
        PlayIdleAnim();
    }
```

```
void PlayIdleAnim()
{
        anim[“Idle”].speed = 3.0f;
        anim.Play("Idle");
}
```

```
    void Update()
    {
        switch (enemyState)
        {
            case ENEMYSTATE.IDLE:
            {
                stateTime  +=  Time.deltaTime;
                if( stateTime > idleStateMaxTime )
                {
                    stateTime   = 0.0f;
                    enemyState = ENEMYSTATE.MOVE;
                }
            }
            break;
            .....
    }    // Update End
}
```

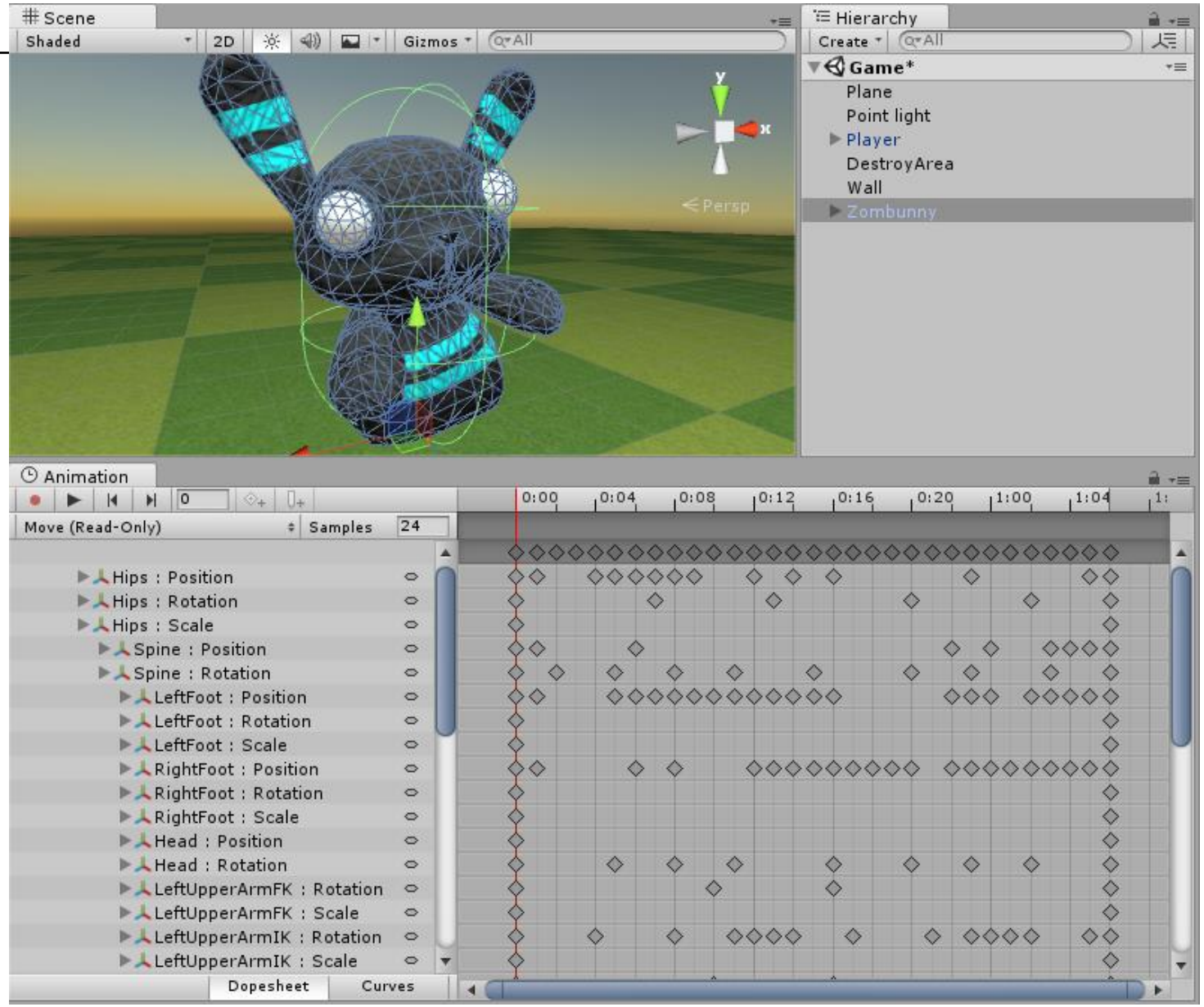
```
public class Zombie : MonoBehaviour
{
    .....
    Transform      target = null;
    CharacterController      characterController = null;

    public    float    moveSpeed    =    5.0f;
    public    float    rotationSpeed   =   10.0f;
    public    float    attackRange    =    2.5f;

    void    Start()
    {
        target    =    GameObject.Find( "Player" ).transform;
        characterController    =    GetComponent< CharacterController >();
    }
}
```

```
case ENEMYSTATE.MOVE:
{
        anim["Move"].speed = 2.0f;
        anim.CrossFade( "Move" );

        float distance = (target.position - transform.position).magnitude;
        if( distance < attackRange )
        {
            enemyState = ENEMYSTATE.ATTACK;
        }
        else
        {
            Vector3 dir = target.position - transform.position;
            dir.y = 0.0f;
            dir.Normalize();
            characterController.SimpleMove( dir * moveSpeed );

            transform.rotation = Quaternion.Lerp( transform.rotation,
                                                  Quaternion.LookRotation( dir ),
                                                  rotationSpeed * Time.deltaTime );
        }
}
break;
```

```
public class Zombie : MonoBehaviour
{
    .....
    public   float    attackStateMaxTime   =  2.0f;

    void Update()
    {
        .......
          case ENEMYSTATE.ATTACK:
          {
               stateTime += Time.deltaTime;
               if( stateTime > attackStateMaxTime )
               {
                    stateTime = 0.0f;
                    anim["Attack"].speed = -0.5f;
                    anim["Attack"].time = anim["Attack"].length;
                    anim.Play( "Attack" );
               }
          }
          break;
    .......
```

Unity

- 애니메이션 이벤트 추가

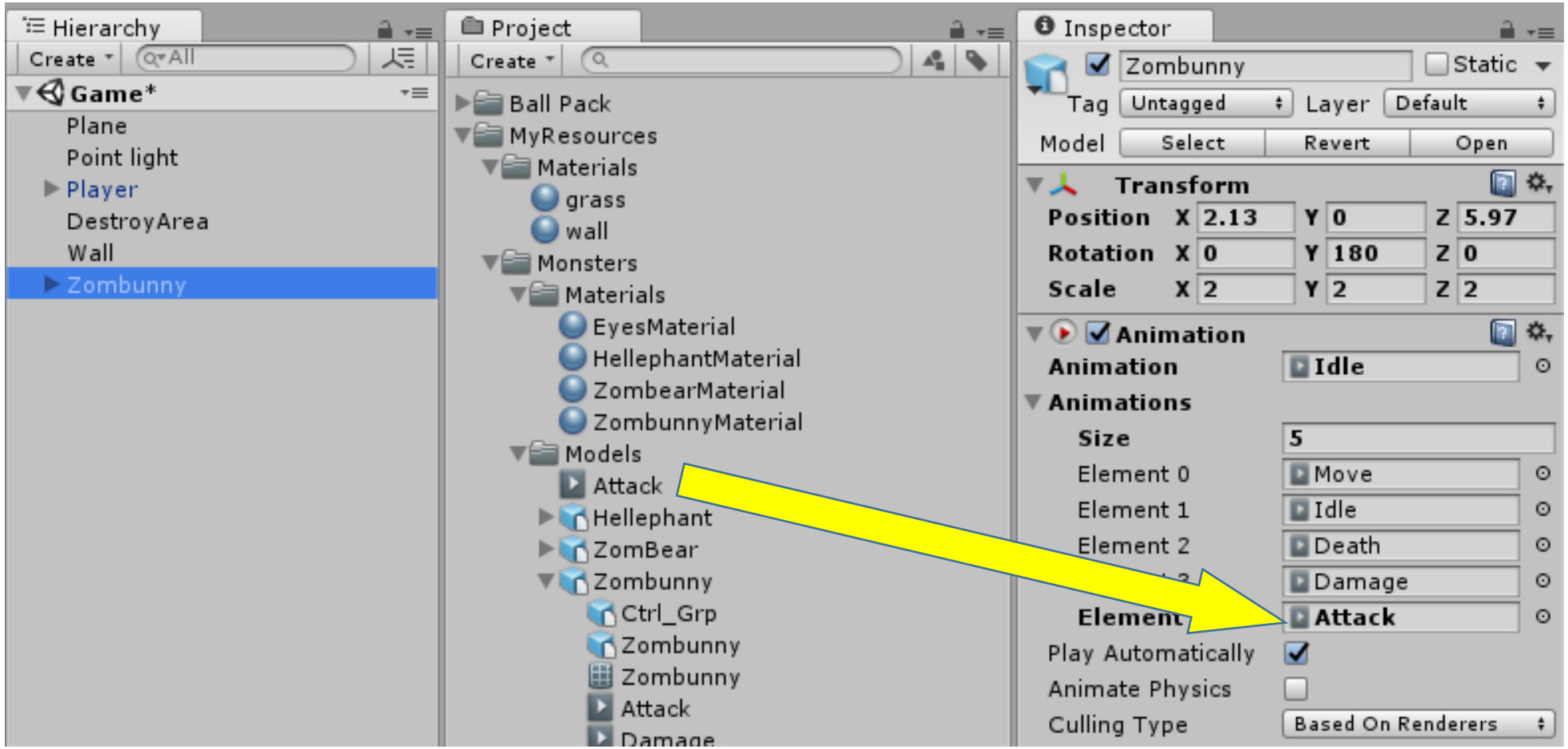


Unity

- Attack 애니메이션 복제

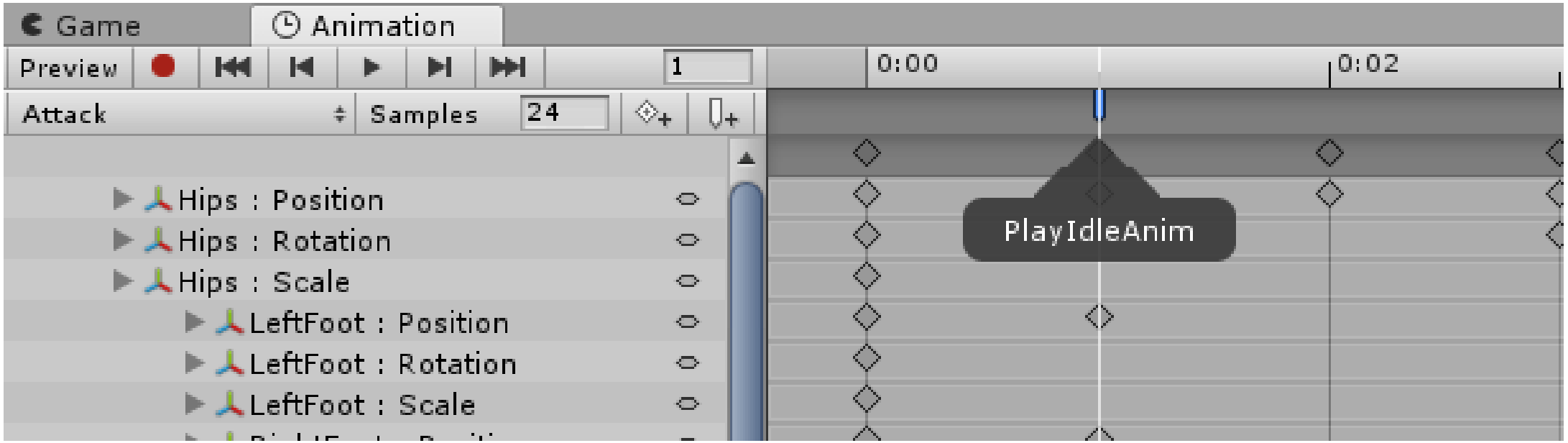
Unity

- Attack 애니메이션 설정



Unity

- Event 설정



Unity – ATTACK 구현

```
case ENEMYSTATE.MOVE:
{
        anim[“run”].speed = 1.5f;
        anim.CrossFade( “run” );

        float distance = (target.position - transform.position).magnitude;
        if( distance < attackRange )
        {
            enemyState = ENEMYSTATE.ATTACK;

            stateTime = attackStateMaxTime;
        }
        ……
}
break;
```

Unity

- 거리가 멀어지면 따라오지 않는 문제가 있다.
  간단히 수정해 볼 것.

Unity – ATTACK 구현

```
case ENEMYSTATE.ATTACK:
{
        ……
        float   distance   =   (target.position - transform.position).magnitude;
        if( distance   >   attackRange )
        {
              enemyState   =   ENEMYSTATE.IDLE;
        }
}
break;
```

Unity

- 구현된 Zombie AI 확인

Unity

- 포탄에 대한 충돌 처리 구현
  Tag 또는 Layer로도 구현해 볼것.

```
public class Zombie : MonoBehaviour
{
    ……
    void OnCollisionEnter( Collision other  )
    {
        Debug.Log(  other.gameObject.name );
        if(  other.gameObject.name.Contains( “Ball” ) == false  )
            return;

        enemyState = ENEMYSTATE.DAMAGE;
    }
}
```

Unity

- 체력 처리를 위한 변수 선언 및 처리

```
public class Zombie : MonoBehaviour
{
    public int   healthPoint = 5;

    void PlayIdleFromDamage()
    {
        anim.CrossFade("Idle");
    }
```

```
case ENEMYSTATE.DAMAGE:
{
    healthPoint -= 1;

    AnimationState animState = anim.PlayQueued("Damage",
                                    QueueMode.PlayNow);
    animState.speed = 2.0f;

    float   animLength = anim["Damage"].length / animState.speed;
    CancelInvoke();
    Invoke("PlayIdleFromDamage",   animLength);

    stateTime = 0.0f;
    enemyState = ENEMYSTATE.IDLE;

    if( healthPoint <= 0 )
        enemyState = ENEMYSTATE.DEAD;
}
break;
```

Unity

- 죽음 처리

```
case ENEMYSTATE.DEAD:
{
    Destroy( gameObject );
}
break;
```

Unity

- 충돌 감지 방법 : Layer 방식

```
void OnCollisionEnter( Collision   other   )
{
    int   layerIndex   =   other.gameObject.layer;
    if(   LayerMask.LayerToName( layerIndex ) != “Ball”    )
          return;

    enemyState   =   ENEMYSTATE.DAMAGE;
}
```

Unity

- 충돌 감지 방법 : Tag 방식

```
void OnCollisionEnter( Collision   other   )
{
    if(   other.gameObject.tag   !=   “Ball”   )
        return;

    enemyState   =   ENEMYSTATE.DAMAGE;
}
```

Unity

- 과제 :
  AI를 구현하기에는 switch~case 문은 복잡할 수 있다.
  Dictionary 와 delegate를 활용하여 구조를 변경해 보자.

Unity

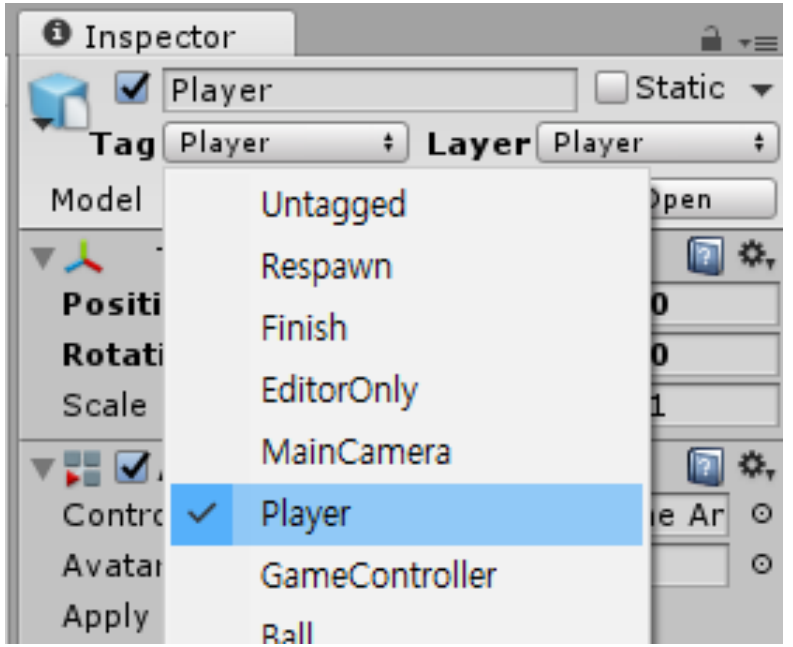
- 적을 계속해서 생성하는 매니저 구현

  EnemyManager 오브젝트 생성

- n 초마다 적 생성
- 적 생성 제한
- 위치는 맵 위에서 랜덤으로 배치

Unity

```
public class EnemyManager : MonoBehaviour
{
    Transform    playerTransform;
    void    Start()
    {
        playerTransform
                = GameObject.FindGameObjectWithTag("Player").transform;
    }
}
```



```
public class EnemyManager : MonoBehaviour
{
    public  GameObject  enemy;
    public  float  spawnTime  =  2.0f;

    float   deltaSpawnTime  =  0.0f;

    void Update()
    {
        deltaSpawnTime  +=  Time.deltaTime;
        if (deltaSpawnTime > spawnTime)
        {
            deltaSpawnTime  =  0.0f;

            GameObject  enemyObj  =  Instantiate( enemy )  as  GameObject;
            Vector3  spawnPos  =  playerTransform.forward * Random.Range(5.0f, 10.0f);
            spawnPos.x  +=  Random.Range(-10.0f, 10.0f);
            spawnPos.z  +=  Random.Range(0.0f, 5.0f);
            spawnPos.y  =  0.1f;
            enemyObj.transform.position  =  spawnPos;
        }
    }
}
```

Unity

- 문제점 : 적이 너무 많이 생성되는 문제가 있다.
  적의 생성에 제한을 두도록 하자.