캐릭터 애니메이션
(Chapter 5)

Jin-Mo Kim
jinmo.kim@hansung.ac.kr

애니메이션을 활용한 캐릭터 움직임

- 애니메이션을 활용한 캐릭터 움직임 처리
  - 키 입력을 통한 캐릭터 움직임
    - 방향 키 : 이동
    - 스페이스 키 : 점프
  - 자연스러운 애니메이션처리
  - 캐릭터를 따라가는 카메라 이동 처리

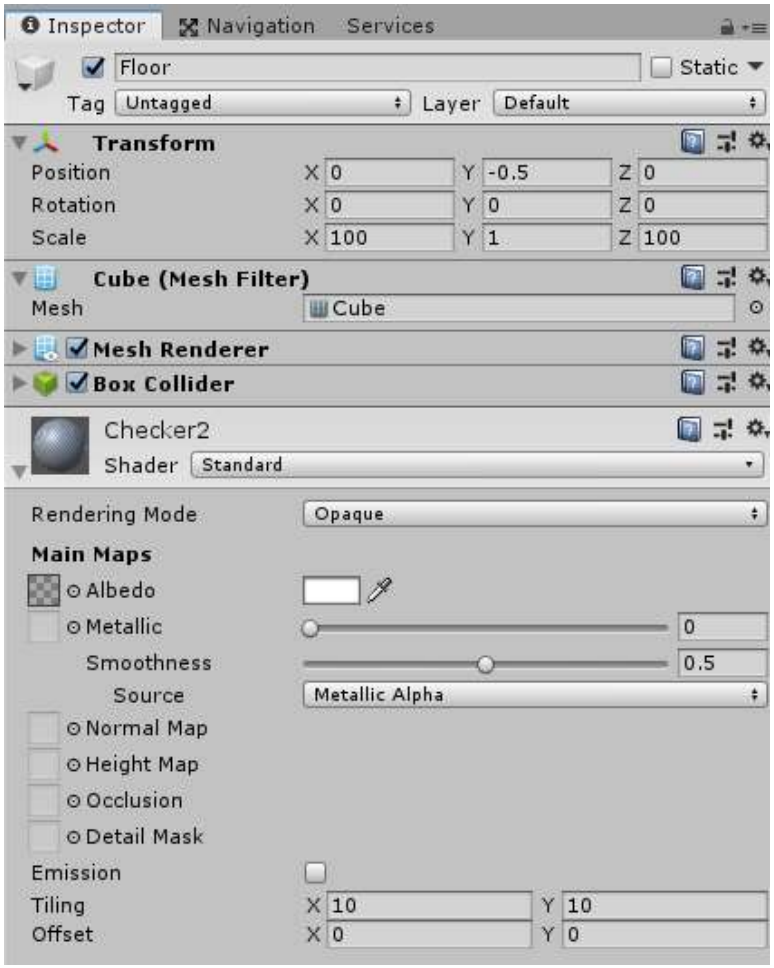
애니메이션을 활용한 캐릭터 움직임

- 스테이지 구성
  - 무대
    - 게임 공간
  - 발판
    - 긴급하게 장애물을 피할 수 있는 수단

  - 무대 생성
    - Hierarchy → Create → Cube
      - 이름 : Floor

      - Scale : 100, 1, 100
      - Position : 0, -0.5, 0

      - Checker2.png 영상을 텍스쳐로 설정
      - Tiling : 10 x 10



Virtual Reality

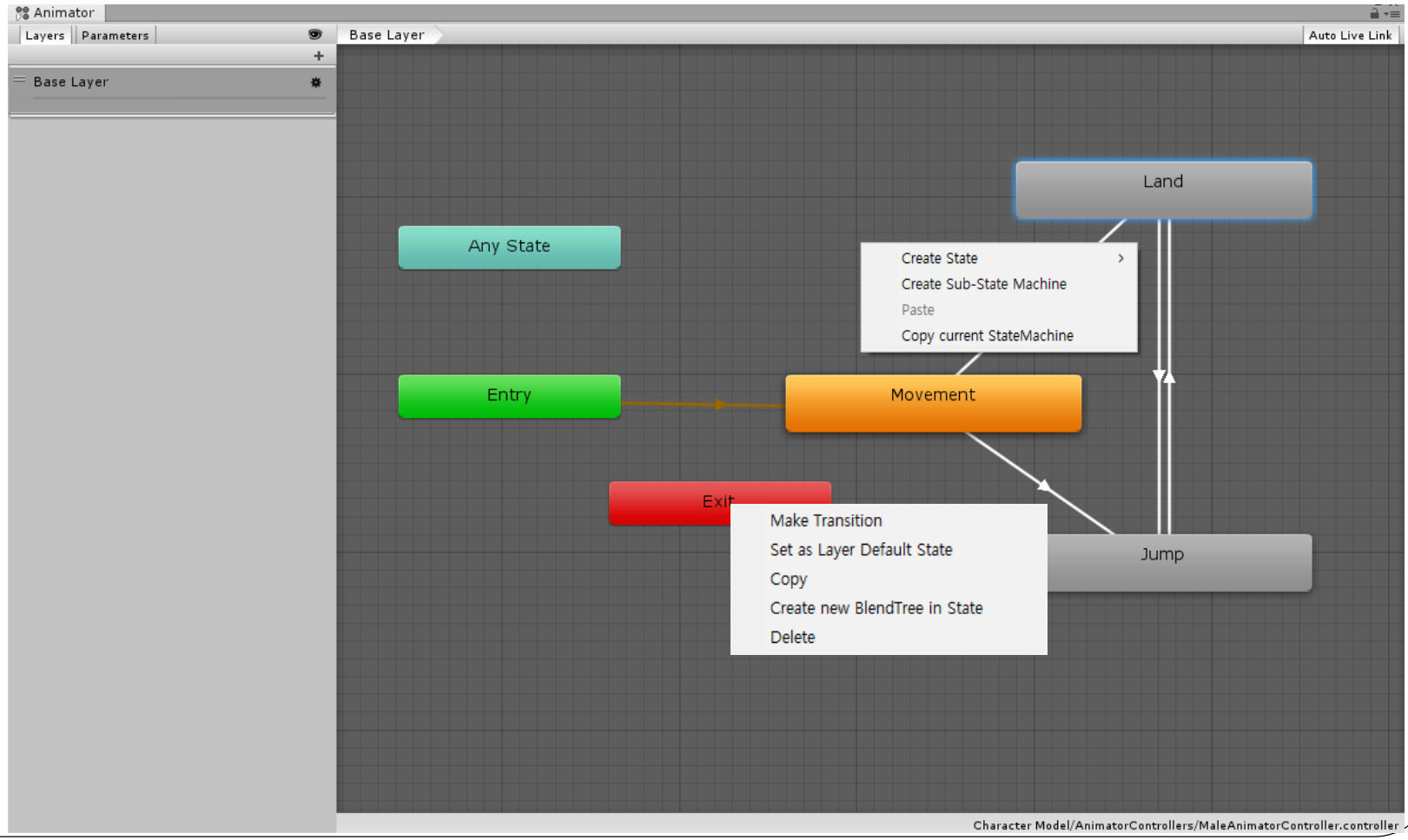
한성대학교 HANSUNG UNIVERSITY

캐릭터를 이용한 미니 게임

- 캐릭터 배치
  - Project → Character Model → Prefabs → Base → High Quality → MaleFree1를 Hierarchy 이동
    - Position : 0, 0, 0
    - Rotation : 0, 180, 0
  - Project → Character Model → Models → free_male_1.fbx
    - Inspector → Rig → Animation Type : Generic

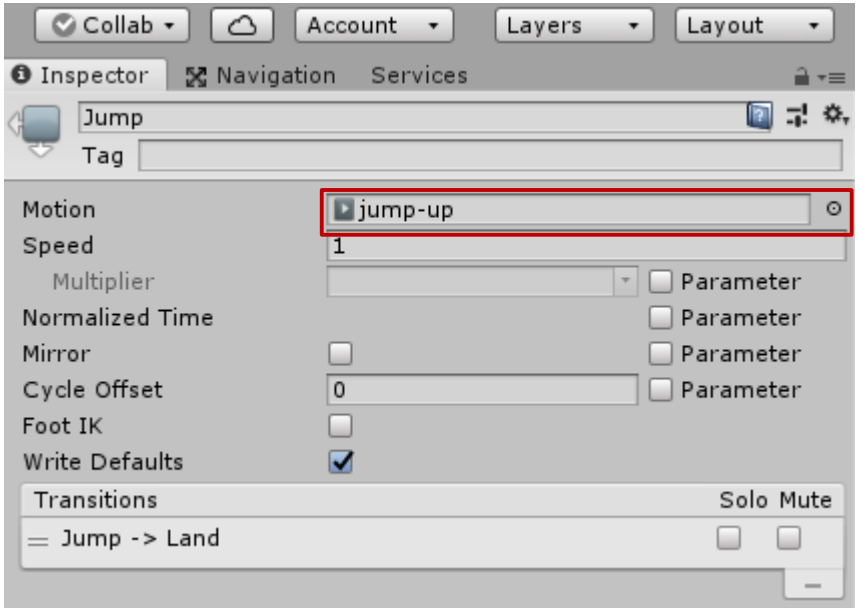
애니메이션 - Animator

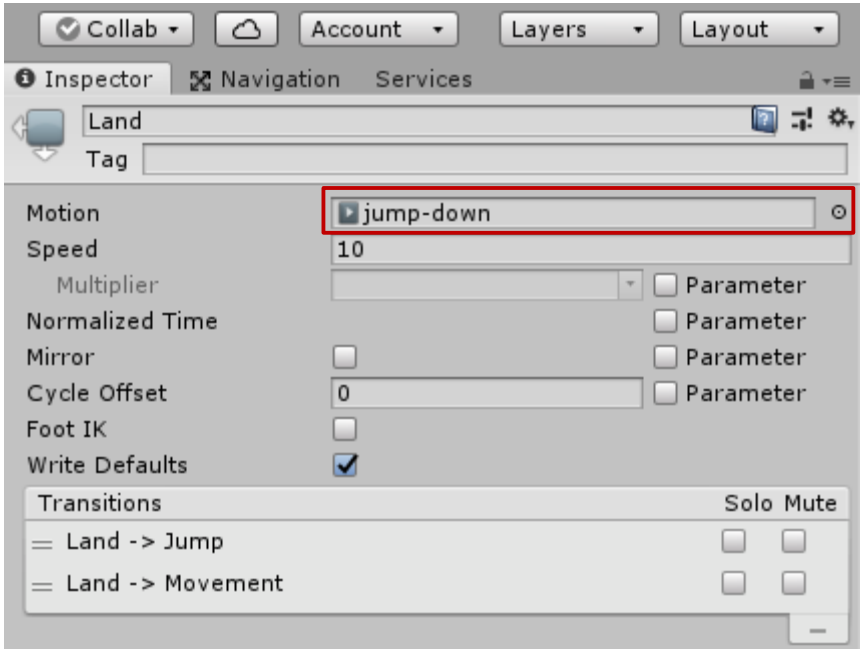
- 애니메이션이 적용된 캐릭터에 대한 처리
  - Project → Create → Animator Controller: MaleAnimatorController



애니메이션 - Animator

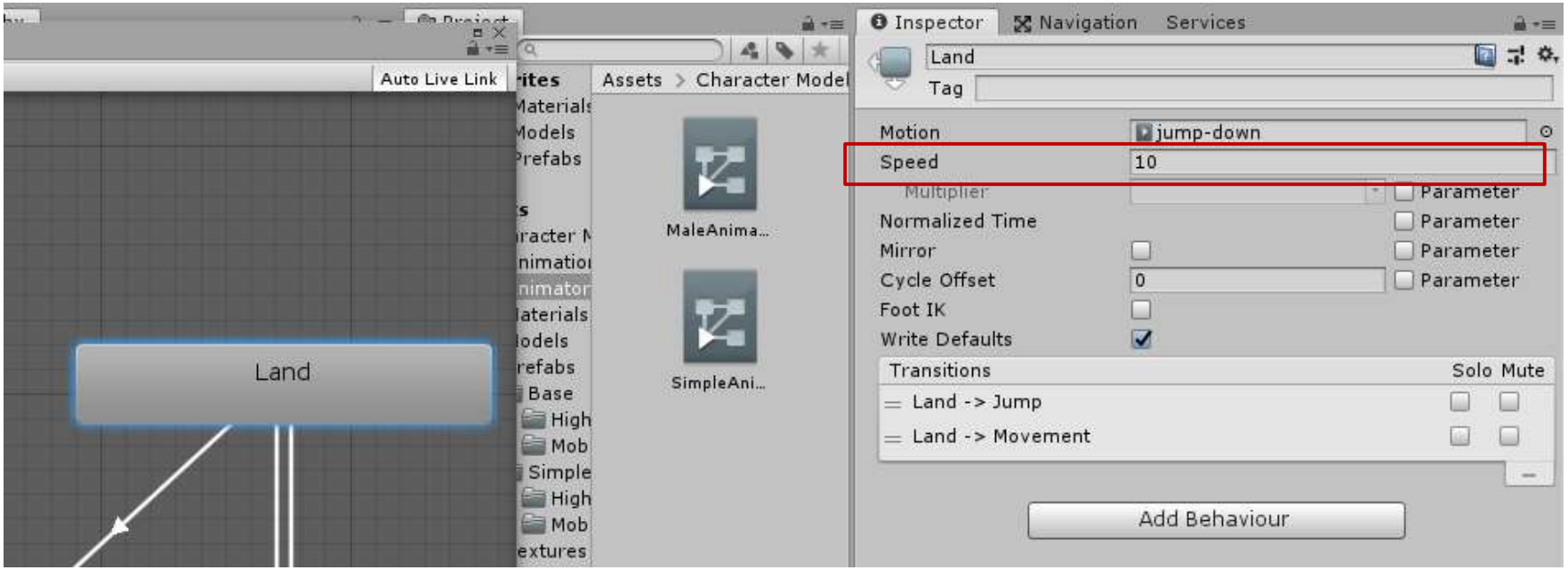
- 애니메이션이 적용된 캐릭터에 대한 처리
  - 애니메이션 동작 등록
    - Land → Motion: jump-down
    - Jump → Motion: jump-up





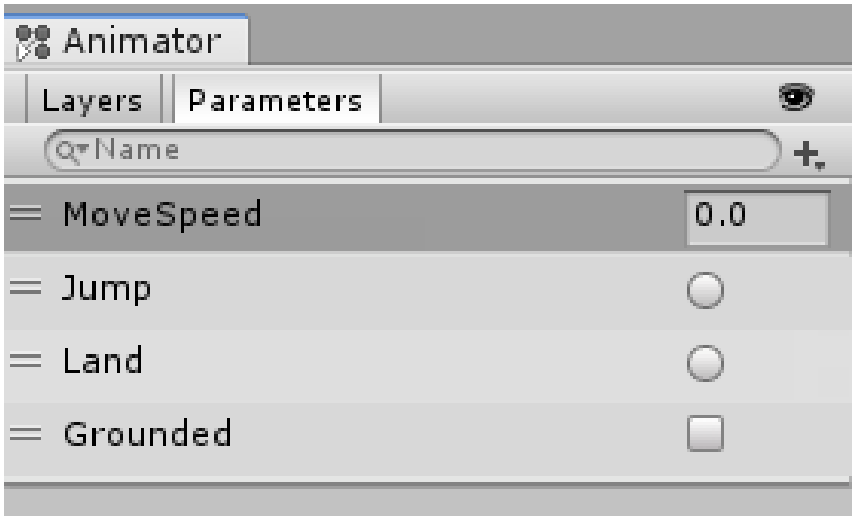
애니메이션 - Animator

- 애니메이션이 적용된 캐릭터에 대한 처리
  - 애니메이션 재생 속도 조절: Speed



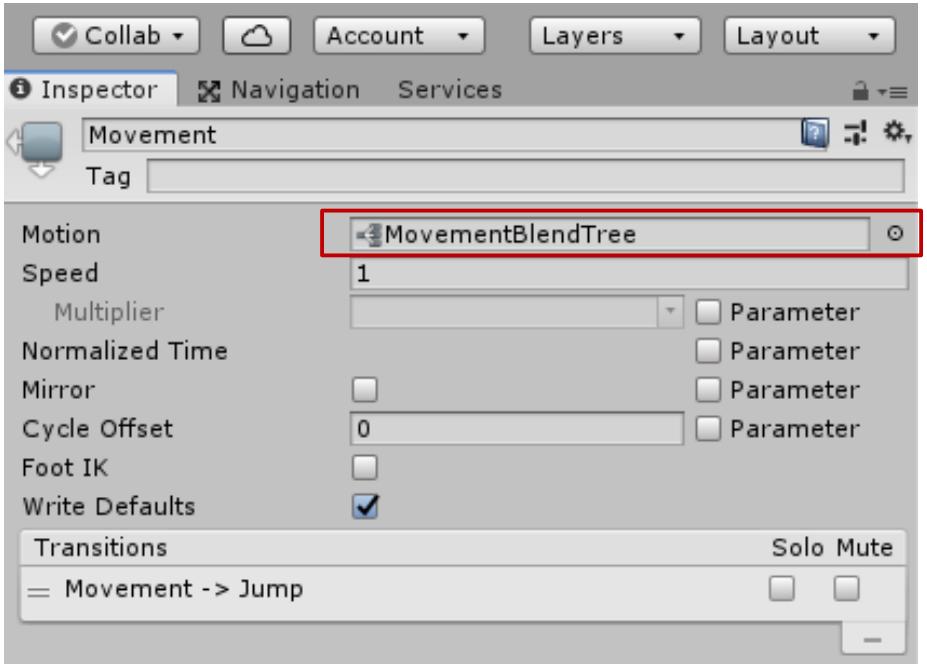
애니메이션 - Animator

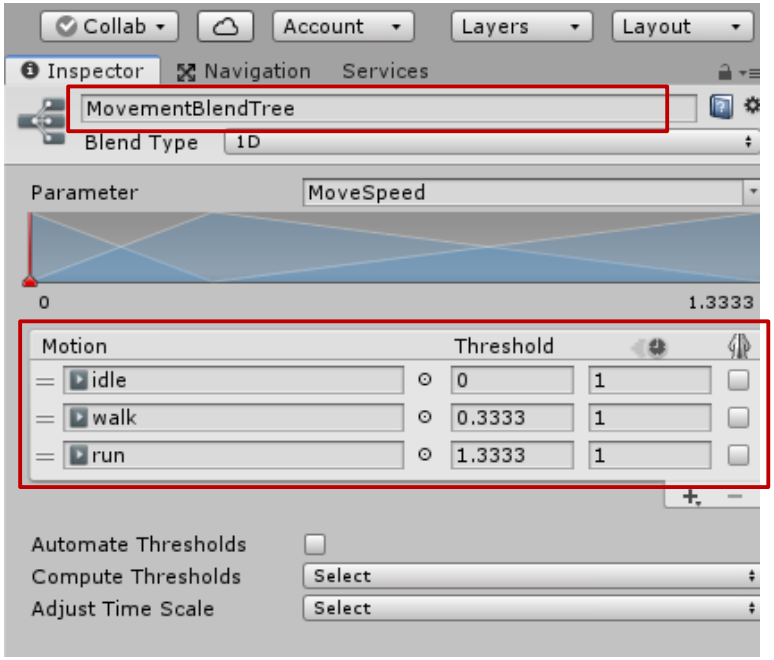
- 애니메이션이 적용된 캐릭터에 대한 처리
  - Parameters 추가
    - MoveSpeed : Float
    - Jump : Trigger
    - Land : Trigger
    - Grounded : Bool



애니메이션 - Animator

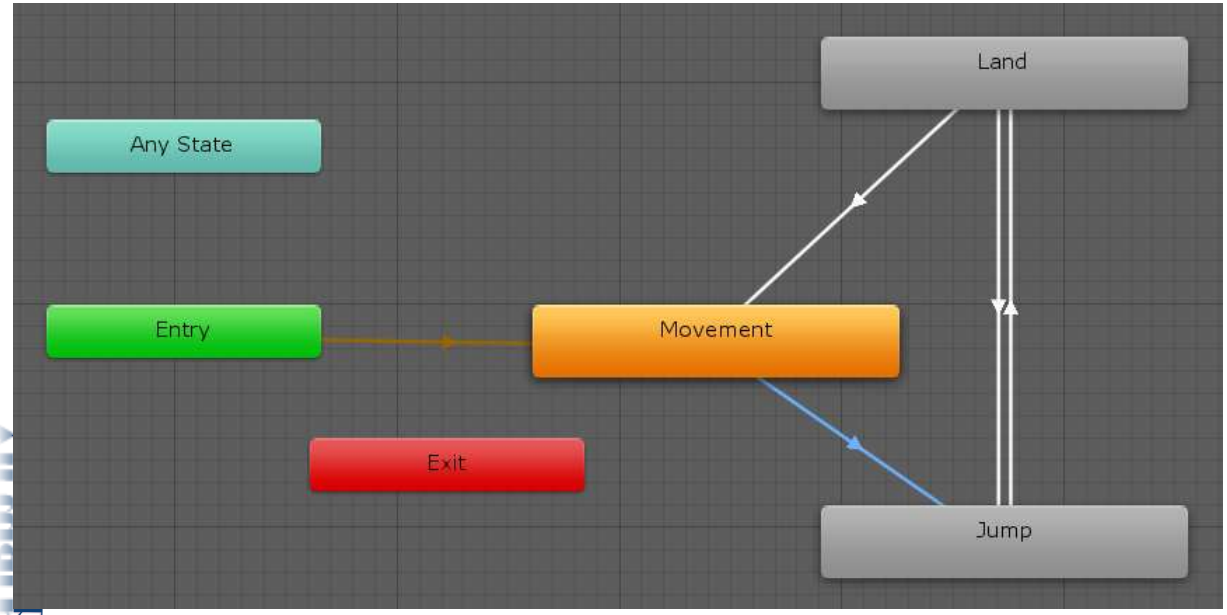
- 애니메이션이 적용된 캐릭터에 대한 처리
  - Movement 선택 → 마우스 우클릭 : Create new BlendTree in State
    - Motion 이름: MovementBlendTree

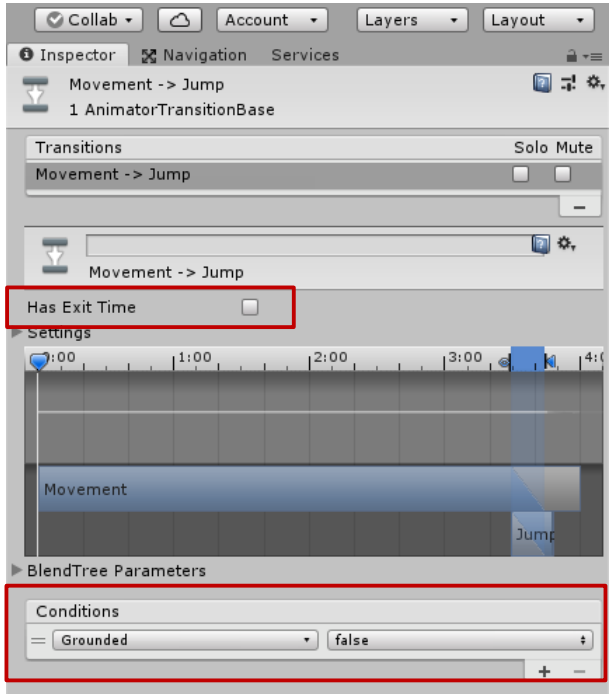




애니메이션 - Animator

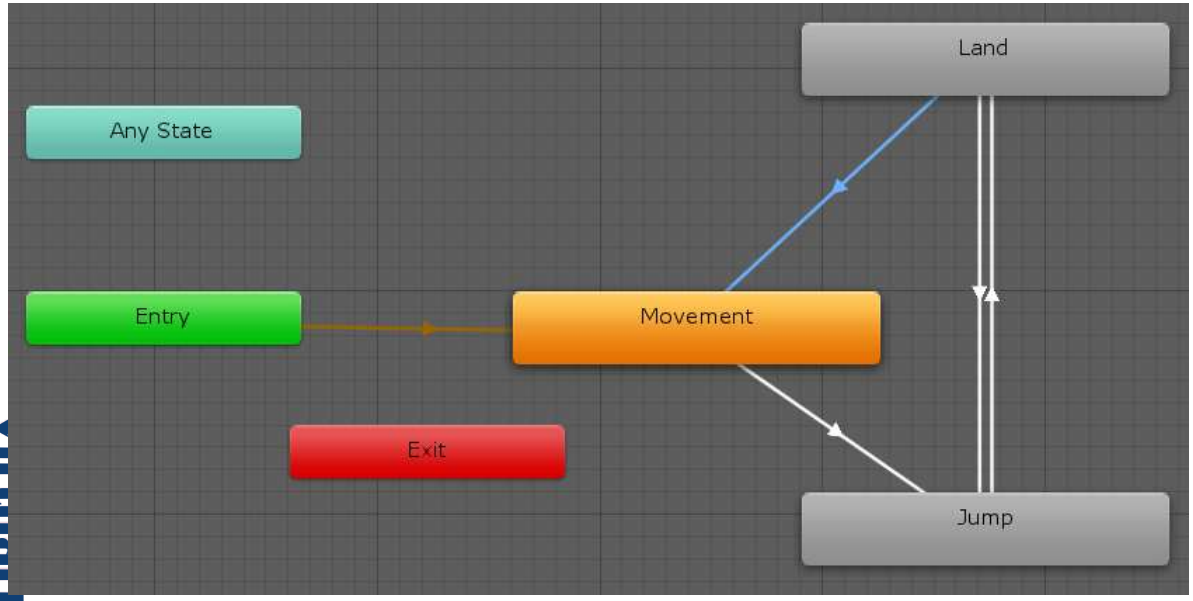
- 애니메이션이 적용된 캐릭터에 대한 처리
  - 애니메이션 사이의 전환 처리 : Transition

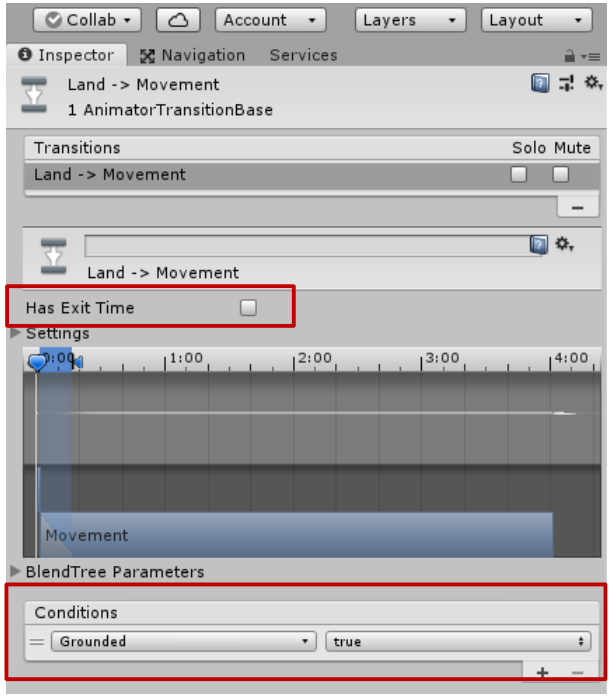




애니메이션 - Animator

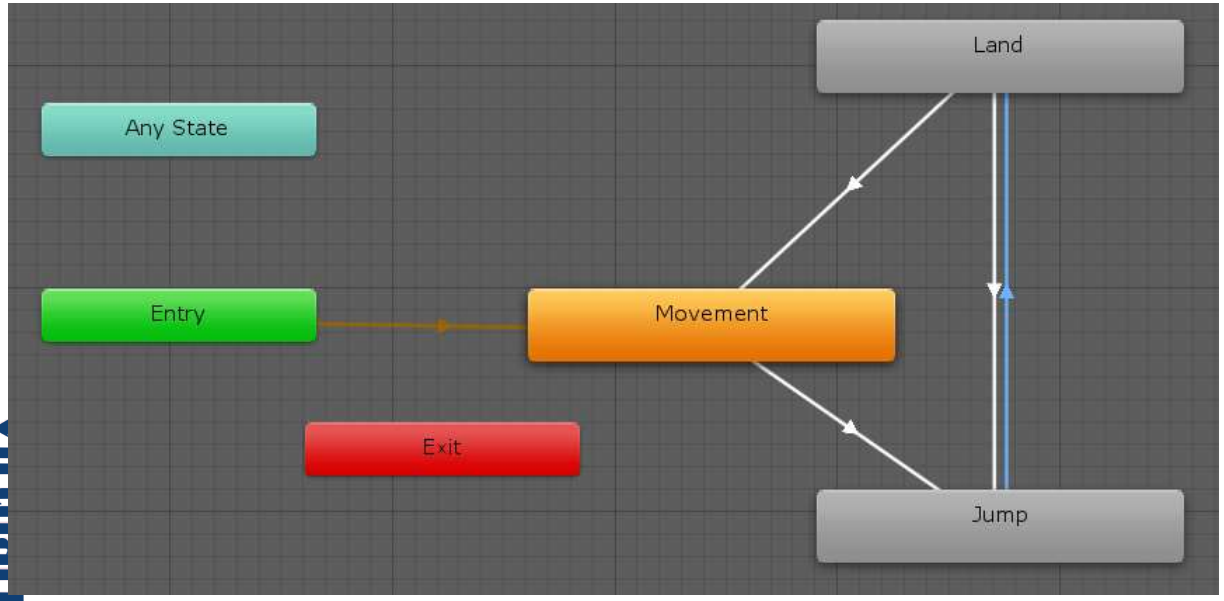
- 애니메이션이 적용된 캐릭터에 대한 처리
  - 애니메이션 사이의 전환 처리 : Transition

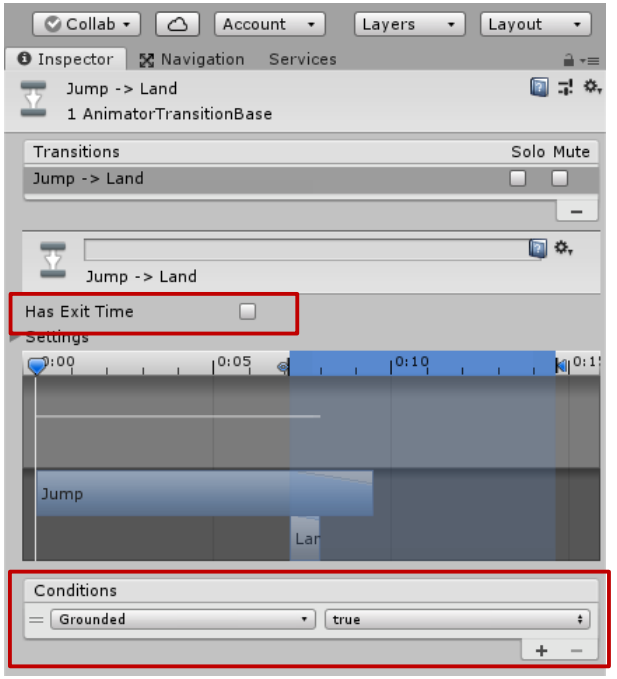




애니메이션 - Animator

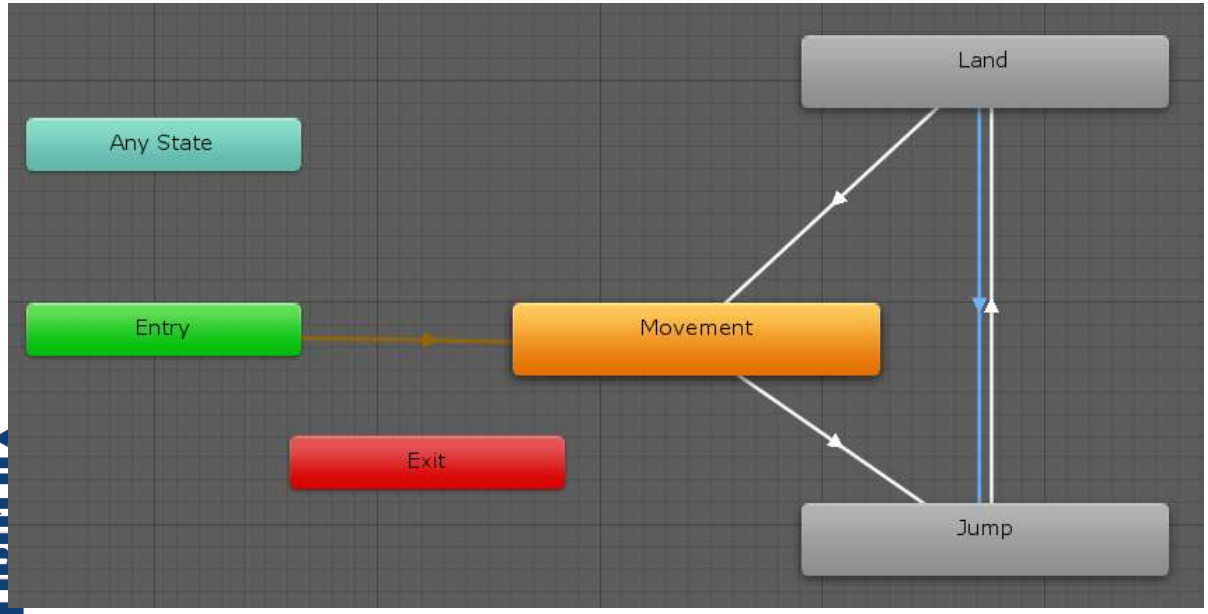
- 애니메이션이 적용된 캐릭터에 대한 처리
  - 애니메이션 사이의 전환 처리 : Transition

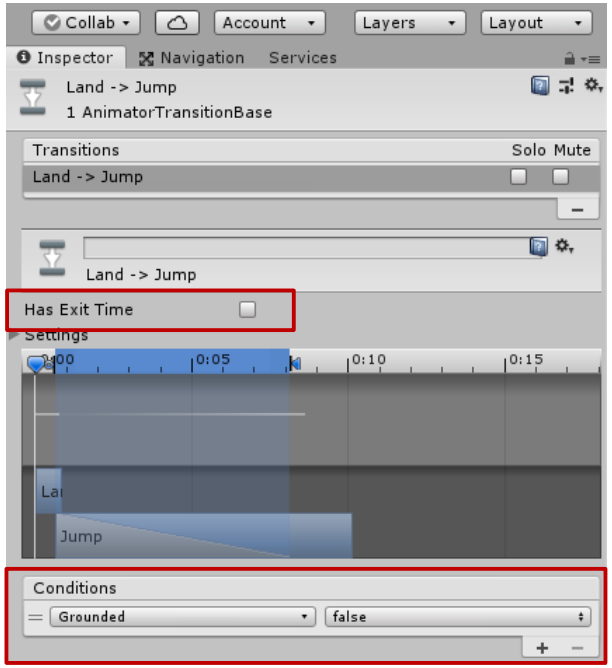




애니메이션 - Animator

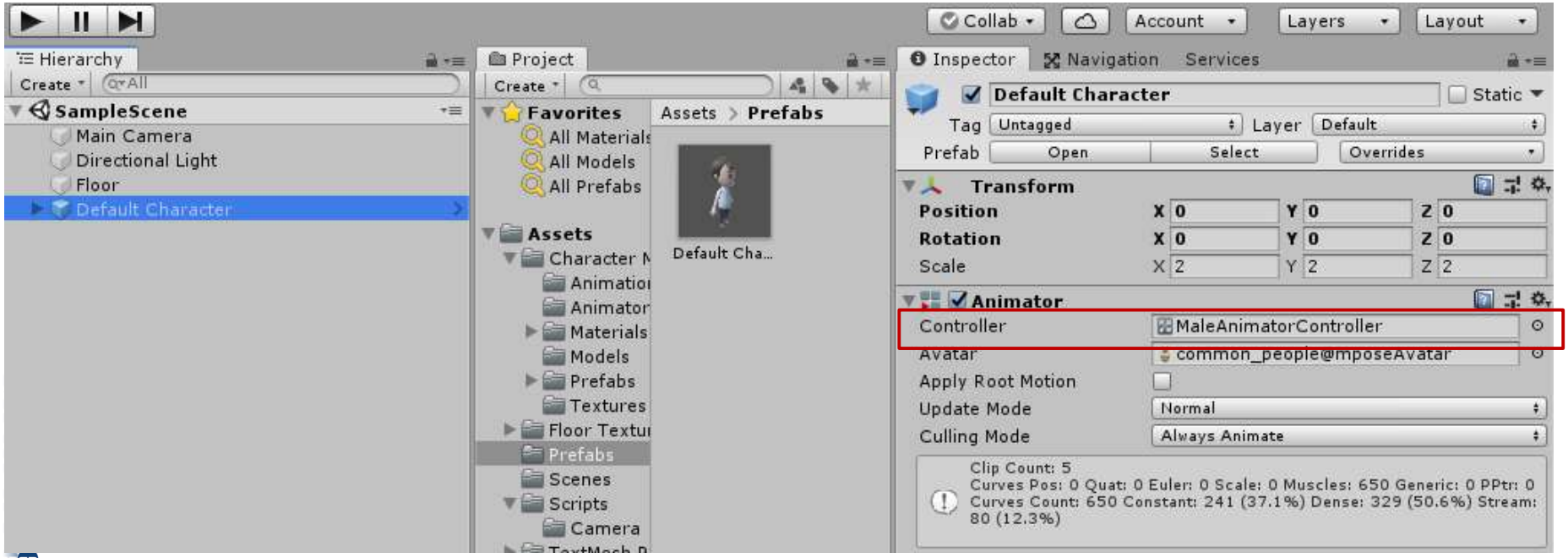
- 애니메이션이 적용된 캐릭터에 대한 처리
  - 애니메이션 사이의 전환 처리 : Transition





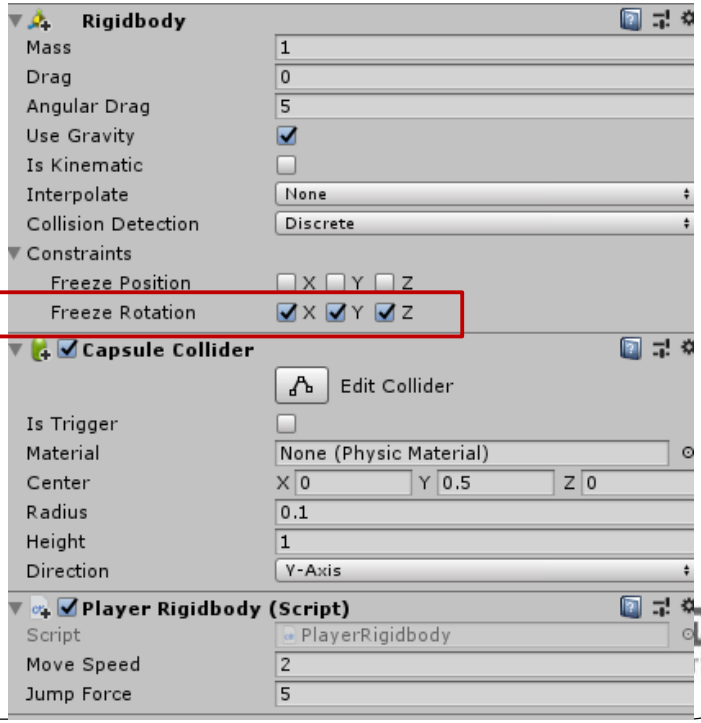
애니메이션 - Animator

- 애니메이션이 적용된 캐릭터에 대한 처리
  - 애니메이터 등록



캐릭터 이동 – Translate/Rigidbody

- 플레이어 캐릭터 구축
  - Rigidbody & Collider 컴포넌트 추가
    - Hierarchy → MaleFree1 선택
    - Component → Physics → Rigidbody
    - Component → Physics → Capsule Collider
      - 캐릭터와 대략적으로 일치시킬 수 있도록 수치 조절



캐릭터 이동 – Translate/Rigidbody

- 스크립트에 의한 캐릭터의 움직임 제어
  - Project → C# Script → PlayerRigidbody.cs 생성

```
private Animator m_animator;

private Rigidbody m_rigidBody;
private bool m_wasGrounded;
private bool m_isGrounded;
private List<Collider> m_collisions = new List<Collider>();

public float m_moveSpeed = 2.0f;
public float m_jumpForce = 5.0f;
private float m_jumpTimeStamp = 0;
private float m_minJumpInterval = 0.25f;

void Start()
{
    m_rigidBody = GetComponent<Rigidbody>();
    m_animator = GetComponent<Animator>();
}

void Update()
{
    m_animator.SetBool("Grounded", m_isGrounded);

    PlayerMove();
    JumpingAndLanding();

    m_wasGrounded = m_isGrounded;
}
```

캐릭터 이동 – Translate/Rigidbody

- 스크립트에 의한 캐릭터의 움직임 제어
  - Project → C# Script → PlayerRigidbody.cs 생성

```
private void PlayerMove()
{
    float h = Input.GetAxis("Horizontal");
    float v = Input.GetAxis("Vertical");

    Vector3 moveHorizontal = Vector3.right * h;
    Vector3 moveVertical = Vector3.forward * v;
    Vector3 velocity = (moveHorizontal + moveVertical).normalized;

    transform.LookAt(transform.position + velocity);

    if (Input.GetKey(KeyCode.LeftShift))
    {
        velocity *= 2.0f;
    }
    transform.Translate(velocity * m_moveSpeed * Time.deltaTime, Space.World);
    //m_rigidBody.MovePosition(transform.position + velocity * m_moveSpeed * Time.deltaTime);

    m_animator.SetFloat("MoveSpeed", velocity.magnitude);
}
```

캐릭터 이동 – Translate/Rigidbody

- 스크립트에 의한 캐릭터의 움직임 제어
  - Project → C# Script → PlayerRigidbody.cs 생성

```
private void JumpingAndLanding()
{
    bool jumpCooldownOver = (Time.time - m_jumpTimeStamp) >= m_minJumpInterval;

    if (jumpCooldownOver && m_isGrounded && Input.GetKey(KeyCode.Space))
    {
        m_jumpTimeStamp = Time.time;
        m_rigidBody.AddForce(Vector3.up * m_jumpForce, ForceMode.Impulse);
    }

    if (!m_wasGrounded && m_isGrounded)
    {
        m_animator.SetTrigger("Land");
    }

    if (!m_isGrounded && m_wasGrounded)
    {
        m_animator.SetTrigger("Jump");
    }
}
```

캐릭터 이동 – Translate/Rigidbody

①
```
private void OnCollisionEnter(Collision collision)
    {
        ContactPoint[] contactPoints = collision.contacts;
        for (int i = 0; i < contactPoints.Length; i++)
        {
            if (Vector3.Dot(contactPoints[i].normal, Vector3.up) > 0.5f)
            {
                if (!m_collisions.Contains(collision.collider))
                {
                    m_collisions.Add(collision.collider);
                }
                m_isGrounded = true;
            }
        }
    }
```

②
```
private void OnCollisionStay(Collision collision)
    {
        ContactPoint[] contactPoints = collision.contacts;
        bool validSurfaceNormal = false;
        for (int i = 0; i < contactPoints.Length; i++)
        {
            if (Vector3.Dot(contactPoints[i].normal, Vector3.up) > 0.5f)
            {
                validSurfaceNormal = true; break;
            }
        }

        if (validSurfaceNormal)
        {
            m_isGrounded = true;
            if (!m_collisions.Contains(collision.collider))
            {
                m_collisions.Add(collision.collider);
            }
        }
        else
        {
            if (m_collisions.Contains(collision.collider))
            {
                m_collisions.Remove(collision.collider);
            }
            if (m_collisions.Count == 0) { m_isGrounded = false; }
        }
    }
```

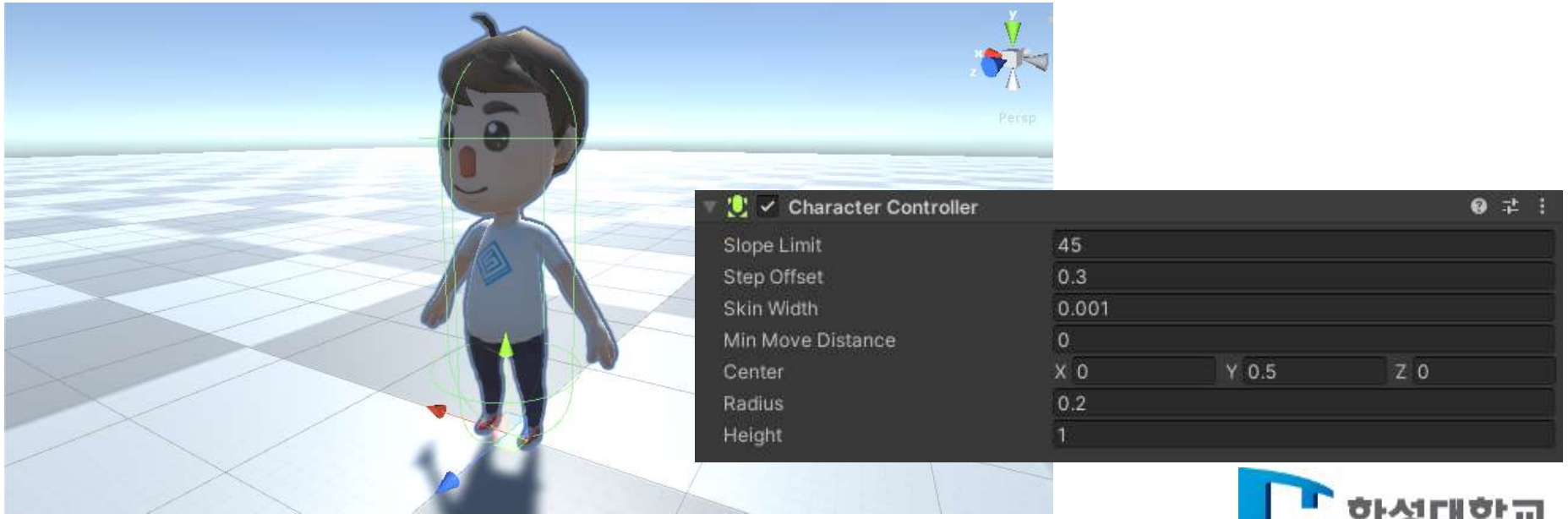
③
```
private void OnCollisionExit(Collision collision)
    {
        if (m_collisions.Contains(collision.collider))
        {
            m_collisions.Remove(collision.collider);
        }
        if (m_collisions.Count == 0) { m_isGrounded = false; }
    }
```

캐릭터를 이용한 액션 게임

- Input 키 설정
  - Edit → Project Settings → Input'
  - 사용자가 미리 지정해 놓은 Input 값을 활용

캐릭터 이동 – Character Controller

- 플레이어 캐릭터 구축
  - CharacterController 컴포넌트 추가
    - Hierarchy → MaleFree1 선택
    - Component → Physics → Character Controller
    - Capsule Collider 같이 캡슐 모양
    - 캐릭터와 대략적으로 일치시킬 수 있도록 수치 조절



한성대학교 HANSUNG UNIVERSITY

캐릭터 이동 – Character Controller

- 스크립트에 의한 캐릭터의 움직임 제어
  - Project → C# Script → PlayerController.cs 생성

```
private Animator m_animator;

private Vector3 m_velocity;

private bool m_wasGrounded;
private bool m_isGrounded = true;

public float m_moveSpeed = 2.0f;
public float m_jumpForce = 5.0f;

void Start()
{
    m_animator = GetComponent<Animator>();
}

void Update()
{
    m_animator.SetBool("Grounded", m_isGrounded);
    PlayerMove();
    JumpingAndLanding();

    m_wasGrounded = m_isGrounded;
}
```

캐릭터 이동 – Character Controller

```
private void PlayerMove()
{
    CharacterController controller = GetComponent<CharacterController>();
    float gravity = 20.0f;

    if (controller.isGrounded)
    {
        m_velocity = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
        m_velocity = m_velocity.normalized;

        if (Input.GetKey(KeyCode.LeftShift))
        {
            m_velocity *= 2.0f;
        }
        m_animator.SetFloat("MoveSpeed", m_velocity.magnitude);

        if (Input.GetButtonDown("Jump"))
        {
            m_velocity.y = m_jumpForce;
        }
        else if (m_velocity.magnitude > 0.5)
        {
            transform.LookAt(transform.position + m_velocity);
        }
    }
    m_velocity.y -= gravity * Time.deltaTime;
    controller.Move(m_velocity * m_moveSpeed * Time.deltaTime);
    m_isGrounded = controller.isGrounded;
}
```

캐릭터 이동 – Character Controller

```
private void JumpingAndLanding()
{
    if (!m_wasGrounded && m_isGrounded)
    {
        m_animator.SetTrigger("Land");
    }

    if (!m_isGrounded && m_wasGrounded)
    {
        m_animator.SetTrigger("Jump");
    }
}
```

카메라 이동

- 캐릭터를 따라 움직이는 카메라에 대한 처리
  - Loot At Camera
    - 카메라의 이동 없이 캐릭터를 항상 바라보도록 설정
  - Follow Camera
    - 캐릭터의 뒤에서 일정한 간격으로 따라가도록 설정
    - Mario Galaxy와 같은 게임에서 주로 사용
    - 캐릭터가 바라 보는 방향과 카메라 방향을 일치하기 위하여 회전을 적용
  - Mouse Aim Camera
    - Follow Camera의 기능에 마우스 움직임에 따른 캐릭터 회전을 추가
    - Follow Camera와 Mouse Aim Camera를 위해서는 스크립트 수정 필요

카메라 이동

- 캐릭터를 따라 움직이는 카메라에 대한 처리
  - Follow Camera와 Mouse Aim Camera를 위해서는 스크립트 수정 필요

```
float h = Input.GetAxis("Horizontal");
float v = Input.GetAxis("Vertical");

Vector3 moveHorizontal = transform.right * h;
Vector3 moveVertical = transform.forward * v;
Vector3 velocity = (moveHorizontal + moveVertical).normalized;

//transform.LookAt(transform.position + velocity); LookAt를 삭제
```

  - 나를 중심으로 캐릭터가 이동할 수 있도록 설정

카메라 이동

- 캐릭터를 따라 움직이는 카메라에 대한 처리

  - The Dungeon Crawler Camera
    - 디아블로와 같은 전형적인 3인칭 시점 카메라
    - 일정한 간격을 두고 캐릭터와 함께 이동하는 카메라 (회전 적용 없음)
    - 캐릭터가 카메라로부터 멀어지면 자연스럽게 간격을 좁혀갈 수 있도록 선형 보간 함수(Lerp)를 사용

    - public static float **Lerp**(float **a**, float **b**, float **t**);
      - /t/에 의해 /a/와 /b/사이를 보간, /t/는 0-1사이의 값으로 고정

      - The parameter t is clamped to the range [0, 1].

      - When t = 0 returns a, When t = 1 return b.
      - t = 0.5 일 때, a와 b 의 평균값을 반환