## Git Initiate

### (if unity project doesn't exist)

Start Unity 3D project, ThirdPersonMove
Save and exit the Project
Rename project folder to ThirdPersonMovex
Create Repo named ThirdPersonMove in Git
git clone https://github.com/rayhere/ThirdPersonMove.git
cd ThirdPersonMove
Drag the project files from ThirdPersonMovex into ThirdPersonMove folder
git add .
git commit -a -m "2nd commit, project initiated"
git push

# 第三人稱角色移動 | Unity新手教學

https://www.youtube.com/watch?v=-Q-g44IgX48

## Step1

Required Package
Input System
Cinemachine
ProBuilder

Create

- ☐ Create Empty named Level >
    - ☐ Create 3DObject > Plane > apply Material named Ground
- ☐ Empty named ThirdPersonPlayer >
    - ☐ 3DObject > Capsule named Player > Pos.y 1.5 Scale.y 1.5 > add CharacterController > add PlayerInput > Create Actions [PlayerInput] named PlayerInputAction > apply PlayerInputAction in Action [PlayerInput]
        - ☐ 3DObject > Cube named Eyes > Pos 0, 0.6, 0.2 > Scale 0.6, 0.1, 1
    - ☐ Create Cinemachine > FreeLookCamera named PlayerCam > drag Player [Capsule] to Follow and LookAt [CinemachineFreeLook] > VerticalFOV 60 > Y Axis > InputAxisValue Invert checked > X Axis > InputAxisValue Invert unchecked > Orbits > BindingMode World Space > X Axis > Speed 80 > Orbits MiddleRig Height 1.5 > TopRig Radius 2, MiddleRig Radius 10, BottomRig Radius 2, MiddleRig Body Damping X,Y,Z to 0 > Aim Tracked Object Offset Y to 1.25 > BottomRig Body Damping X,Y,Z to 0 > Aim Tracked Object Offset Y to 1.25
    - ☐

Create 2 script
PlayerController.cs
ControllerMovement3D.cs

Drag
PlayerController.cs
ControllerMovement3D.cs
Into Player [GameObject] in ThirdPersonPlayer
Drag MainCamera into ControllerMovement3D.cs

# Code

## PlayerController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;

public class PlayerController : MonoBehaviour
{
    private ControllerMovement3D _controllerMovement; //To Grab
ControllerMovement3D.cs
    private UnityEngine.Vector3 _moveInput;

    private void Awake()
    {
        //Grab ControllerMovement3D from the Object with this script, so
we don't have to create SerializeField;
        _controllerMovement = GetComponent<ControllerMovement3D>();
    }

    public void OnMove(InputValue value)
    {
        Vector2 input = value.Get<Vector2>();
        _moveInput = new Vector3 (input.x, 0f, input.y);
    }

    private void Update()
    {
        if (_controllerMovement == null) return;

        _controllerMovement.SetMoveInput(_moveInput);
        _controllerMovement.SetLookDirection(_moveInput);
    }
}
```

ControllerMovement3D.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.TextCore.Text;

public class ControllerMovement3D : MonoBehaviour
{
    [Header("Movement")]
    [SerializeField] private float _moveSpeed = 5f;
    [SerializeField] private float _turnSpeed = 10f;
    [SerializeField] private GameObject _mainCamera;
    // Used to synchronize the rotation of the main camera with the
character's position.

    private float _speed = 0f;
    private bool _hasMoveInput;
    private Vector3 _moveInput;
    private Vector3 _lookDirection;

    private CharacterController _characterController;

    private void Start()
    {
        _characterController = GetComponent<CharacterController>();
    }

    // This method is used to set the movement input for the character.
    public void SetMoveInput(Vector3 input)
    {
        // Check if the player presses the key or not.
        // It checks if the player has pressed any keys. If the input
magnitude is greater than 0.1,
        // it sets '_hasMoveInput' to true to avoid stick drag or
floating-point error. Otherwise, it sets it to zero.
        _hasMoveInput = input.magnitude > 0.1f;
        _moveInput = _hasMoveInput ? input : Vector3.zero;
    }

    // to make the character actually rotate
```

```
    public void SetLookDirection(Vector3 direction)
    {
        // We only get axis x and z because the camera only moves
horizontally.
        // Rotate the player.
        _lookDirection = new Vector3(direction.x, 0f,
direction.z).normalized;
    }

    private void FixedUpdate() {
        _speed = 0;

        // If player not moving
        float targetRotation = 0f;

        if (_moveInput.magnitude < 0.1f)
        {
            _moveInput = Vector3.zero; // make movement to zero if
magnitude is too small
            return;
        }

        // Move character
        if (_moveInput != Vector3.zero)
        {
            _speed = _moveSpeed; // If player is moving
        }

        targetRotation =
Quaternion.LookRotation(_lookDirection).eulerAngles.y +
_mainCamera.transform.rotation.eulerAngles.y;
        UnityEngine.Quaternion rotation = UnityEngine.Quaternion.Euler(0,
targetRotation, 0);
        transform.rotation =
UnityEngine.Quaternion.Slerp(transform.rotation, rotation, _turnSpeed *
Time.fixedDeltaTime); // Smooth the rotation

        _moveInput = rotation * Vector3.forward;
        _characterController.Move(_moveInput * _speed *
Time.fixedDeltaTime); // Let CharacterController move the character
```

```
        }
}
```

# 第三人稱角色移動 | Unity新手教學

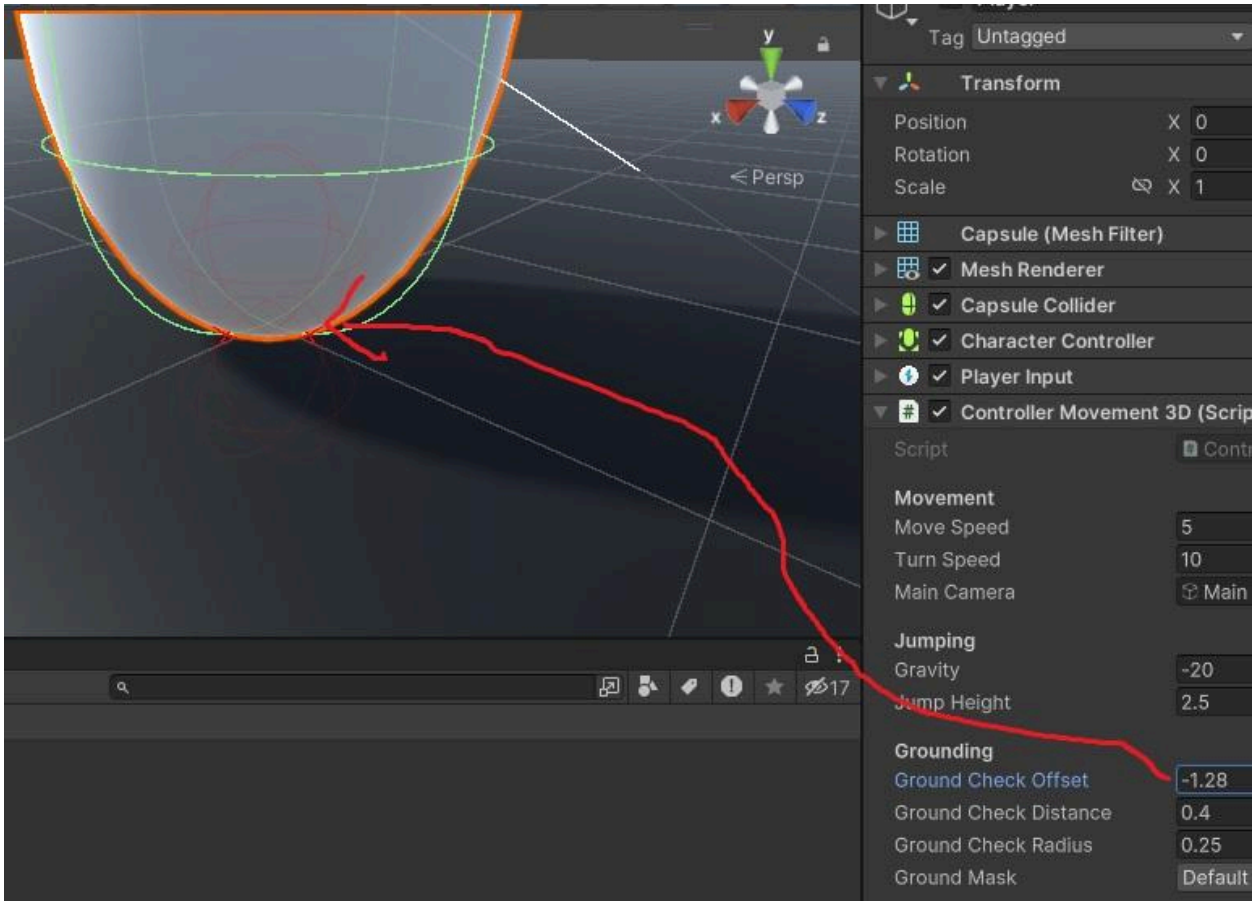
https://youtu.be/-Q-g44IgX48?si=QkH7__LkWqc5J170&t=374

## Step2

Jump Move
Select PlayerInputAction [InputActionAsset] > Edit > Create a Jump [Actions] in Player [ActionMaps] > ActionType Button > BindingPath Space[Keyboard] > set Keyboard&Mouse UseInControlScheme > SaveAsset

Update Code
PlayerController.cs
ControllerMovement3D.cs

ControllerMovement3D.cs
Adjust GroundCheckOffset -2 > GroundMask Default
https://youtu.be/-Q-g44IgX48?si=leRmN6tjW9iiiSBX&t=495



### Calculate jump velocity from jump height and gravity

https://youtu.be/-Q-g44IgX48?si=4D-Uplmn9Np9l1rU&t=520

We use SphereCast to determine if a player has landed on the ground.

### Prefix “On” for OnMove() method

We use Prefix “On” for the method OnMove(), OnJump for(), to understand which methods are bound to input actions using NewInputSystem

Prefix “On” for the function like OnMove(), OnJump() in PlayerController.cs because it follows the naming convention specified by the Unity Input System.

When you bind an action to a method in a script using the Unity Input System, you typically use the "On" prefix followed by the name of the action. This naming convention is used to automatically wire up the method to be called when the associated input action is triggered.

So, in the case of `OnMove(InputValue value)`, it means that this method will be called whenever there is input detected for the "Move" action defined in the input actions asset. This is just a convention used in the Unity Input System to help developers understand which methods are bound to input actions.

## Code

### PlayerController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.InputSystem;

public class PlayerController : MonoBehaviour
{
    private ControllerMovement3D _controllerMovement; //To Grab
ControllerMovement3D.cs
    private UnityEngine.Vector3 _moveInput;

    private void Awake()
    {
```

```
        //Grab ControllerMovement3D from the Object with this script, so
we don't have to create SerializeField;
        _controllerMovement = GetComponent<ControllerMovement3D>();
    }

    public void OnMove(InputValue value)
    {
        Vector2 input = value.Get<Vector2>();
        _moveInput = new Vector3 (input.x, 0f, input.y);
    }

    public void OnJump(InputValue value)
    {
        _controllerMovement.Jump();
    }

    private void Update()
    {
        if (_controllerMovement == null) return;


        _controllerMovement.SetMoveInput(_moveInput);
        _controllerMovement.SetLookDirection(_moveInput);
    }
}
```

ControllerMovement3D.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.TextCore.Text;

public class ControllerMovement3D : MonoBehaviour
{
    [Header("Movement")]
    [SerializeField] private float _moveSpeed = 5f;
    [SerializeField] private float _turnSpeed = 10f;
    [SerializeField] private GameObject _mainCamera;
    // Used to synchronize the rotation of the main camera with the
character's position.

    private float _speed = 0f;
    private bool _hasMoveInput;
    private Vector3 _moveInput;
    private Vector3 _lookDirection;

    [Header("Jumping")]
    [SerializeField] private float _gravity = -20f;
    [SerializeField] private float _jumpHeight = 2.5f;
    private Vector3 _velocity;


    [Header("Grounding")]
    [SerializeField] private float _groundCheckOffset = 0f;
    [SerializeField] private float _groundCheckDistance = 0.4f;
    [SerializeField] private float _groundCheckRadius = 0.25f;
    [SerializeField] private LayerMask _groundMask;
    private bool _isGrounded;
    private Vector3 _groundNormal;

    private CharacterController _characterController;

    private void Start()
    {
        _characterController = GetComponent<CharacterController>();
    }
```

```
    // This method is used to set the movement input for the character.
    public void SetMoveInput(Vector3 input)
    {
        // Check if the player has pressed any keys. If the input
magnitude is greater than 0.1,
        // set '_hasMoveInput' to true to avoid stick drag or
floating-point error. Otherwise, set it to zero.
        _hasMoveInput = input.magnitude > 0.1f;
        _moveInput = _hasMoveInput ? input : Vector3.zero;
    }

    // This method is used to set the look direction for the character.
    public void SetLookDirection(Vector3 direction)
    {
        // We only get axis x and z because the camera only moves
horizontally.
        // Rotate the player.
        _lookDirection = new Vector3(direction.x, 0f,
direction.z).normalized;
    }

    // This method is called when the player wants to jump.
    public void Jump()
    {
        // Check if the player is grounded before allowing the jump.
        if (!_isGrounded) return;

        // Calculate jump velocity from jump height and gravity.
        float jumpVelocity = Mathf.Sqrt(2f * -_gravity * _jumpHeight);
        _velocity = new Vector3(0, jumpVelocity, 0);
    }
    private void FixedUpdate()
    {
        // Check if the player is grounded.
        _isGrounded = CheckGround();

        // Apply gravity over time.
        _velocity.y += _gravity * Time.fixedDeltaTime;
```

```
        _characterController.Move(_velocity * Time.fixedDeltaTime); // Use
calculated in Jump()

        //_speed = 0; // Reset speed to 0 at the beginning of each
FixedUpdate.

        // If player is not moving, set movement to zero.
        if (_moveInput.magnitude < 0.1f)
        {
            _moveInput = Vector3.zero; // make movement to zero if
magnitude is too small
            return; // Having this line, we may not need to Reset speed to
0
        }

        // Move character.
        if (_moveInput != Vector3.zero)
        {
            _speed = _moveSpeed; // If player is moving
        }

        float targetRotation =
Quaternion.LookRotation(_lookDirection).eulerAngles.y +
_mainCamera.transform.rotation.eulerAngles.y;
        //UnityEngine.Quaternion rotation =
UnityEngine.Quaternion.Euler(0, targetRotation, 0);
        //transform.rotation =
UnityEngine.Quaternion.Slerp(transform.rotation, rotation, _turnSpeed *
Time.fixedDeltaTime); // Smooth the rotation

        Quaternion rotation = Quaternion.Euler(0, targetRotation, 0);
        transform.rotation = Quaternion.Slerp(transform.rotation,
rotation, _turnSpeed * Time.fixedDeltaTime); // Smooth rotation.

        _moveInput = rotation * Vector3.forward;
        _characterController.Move(_moveInput * _speed *
Time.fixedDeltaTime); // Let CharacterController move the character.
    }

    // Check if the player is grounded using a sphere cast.
```

```
    private bool CheckGround()
    {
        // Start position for the sphere cast.
        Vector3 start = transform.position + Vector3.up *
_groundCheckOffset;

        // Perform sphere cast.
        if (Physics.SphereCast(start, _groundCheckRadius, Vector3.down,
out RaycastHit hit, _groundCheckDistance, _groundMask))
        {
            // If the player is grounded, save the ground normal and
return true.
            _groundNormal = hit.normal;
            return true;
        }
        _groundNormal = Vector3.up;
        return false;
    }

    // Draw debug spheres for ground checking.
    private void OnDrawGizmosSelected()
    {
        // Set gizmos color.
        //Gizmos.color = Color.red;
        //if (_isGrounded) Gizmos.color = Color.green;
        Gizmos.color = _isGrounded ? Color.green : Color.red;

        // Find start/end positions of sphere cast.
        Vector3 start = transform.position + Vector3.up *
_groundCheckOffset;
        Vector3 end = start + Vector3.down * _groundCheckDistance;

        // Draw wire spheres.
        Gizmos.DrawWireSphere(start, _groundCheckRadius);
        Gizmos.DrawWireSphere(end, _groundCheckRadius);
    }
}
```