Git Initiate

(if unity project doesn't exist)

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

FIRST PERSON MOVEMENT in 10 MINUTES - Unity Tutorial

https://www.youtube.com/watch?v=f473C43s8nE

Summary

This will create a Player in ThirdPerson View
With Move and Jump
With Character Controller
No Animation
No Rigidbody

Step1 Setup a camera

Required Package
Input System
Cinemachine
ProBuilder

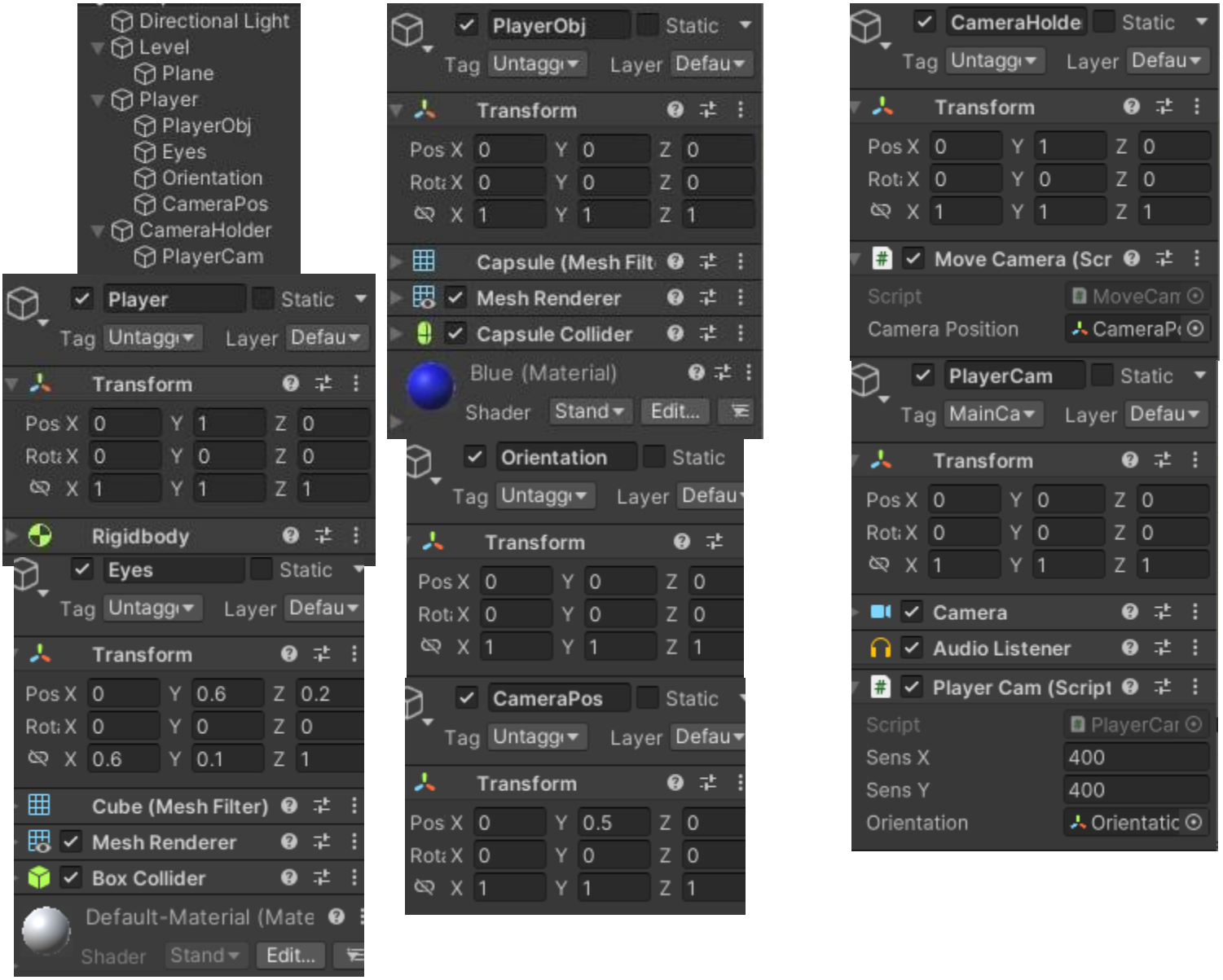
https://youtu.be/f473C43s8nE?si=0GgS2H4KKc4HfyiT&t=131
Create

- ☐ Create Empty named Level >
  - ☐ Create 3DObject > Plane > apply Material named Ground
- ☐ Empty named Player > Pos.y 1 > add Rigidbody > Interpolate Interpolate [Rigidbody] > CollisionDetection Continuous

- [ ] 3DObject > Capsule named PlayerObj > 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 Empty Object named Orientation

- [ ] Create Empty named CameraHolder > add MoveCamera.cs > Transform same as Player [GameObject] > Drag CameraPos child of Player [GameObject] to CameraPosition [MoveCamera.cs]
    - [ ] Drag MainCamera here, named PlayerCam > Pos 0,0,0 > add PlayerCam.cs > set value 400 for SensX, SensY > Drag Orientation child of Player [GameObject] to Orientation [PlayerCam.cs]

P.S. You can directly adjust PlayerCam Pos to have better view



Change:

Empty Object named Orientation is for keeps track of the direction you're facing
Orientation [EmptyObject] stores the direction your facing

Put the camera into a separate camera holder
https://youtu.be/f473C43s8nE?si=KI9Zczq1Wht0kLeX&t=154
Because having a camera on a rigidbody object can be a bit buggy

In order for this to work, you just need this CameraPos [EmptyObject] inside the player.
Drag it up a bit, CameraPos Pos.Y is about Player [GameObject] Pos.Y

Then on the camera holder, you can add this really simple script MoveCamera.cs, to make the camera always move with your player

Create 2 script
MoveCamera.cs
PlayerCam.cs

Code

MoveCamera.cs

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

public class MoveCamera : MonoBehaviour
{
    public Transform cameraPosition;

    private void Update()
    {
        transform.position = cameraPosition.position;
    }
}
```

PlayerCam.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor.Experimental.GraphView;
using UnityEngine;

public class PlayerCam : MonoBehaviour
{
    public float sensX; // Sensitivity for mouse X axis
    public float sensY; // Sensitivity for mouse Y axis

    public Transform orientation; // Reference to the player's orientation
transform

    float xRotation; // Current rotation around the X axis
    float yRotation; // Current rotation around the Y axis

    private void Start()
    {
        // Lock the cursor in the middle of the screen and make it
invisible
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }

    private void Update()
    {
        // Get mouse input
        float mouseX = Input.GetAxisRaw("Mouse X") * Time.deltaTime *
sensX;
        float mouseY = Input.GetAxisRaw("Mouse Y") * Time.deltaTime *
sensY;

        // Update rotation values based on mouse input
        yRotation += mouseX;
        xRotation -= mouseY;

        // Clamp the rotation around the X axis to prevent looking up or
down more than 90 degrees
        xRotation = Mathf.Clamp(xRotation, -90f, 90f);
```

```
        // Rotate the camera and player orientation along both the X and Y
axes
        transform.rotation = Quaternion.Euler(xRotation, yRotation, 0);

        // Rotate the player's orientation along the Y axis
        orientation.rotation = Quaternion.Euler(0, yRotation, 0);
    }
}
```

FIRST PERSON MOVEMENT in 10 MINUTES - Unity Tutorial

https://youtu.be/f473C43s8nE?si=YFn1_gT9Xg2Zde3q&t=200

Step2 Set up a movement

Create PlayerMovement.cs > add PlayerMovement.cs in Player [GameObject] >

Drag Orientation Player[GameObject] into Orientation PlayerMovement.cs Player[GameObject] >

Create Layer named Ground > pick Ground Layer in WhatIsGround PlayerMovement.cs[GameObject]

Drag & Speed Control

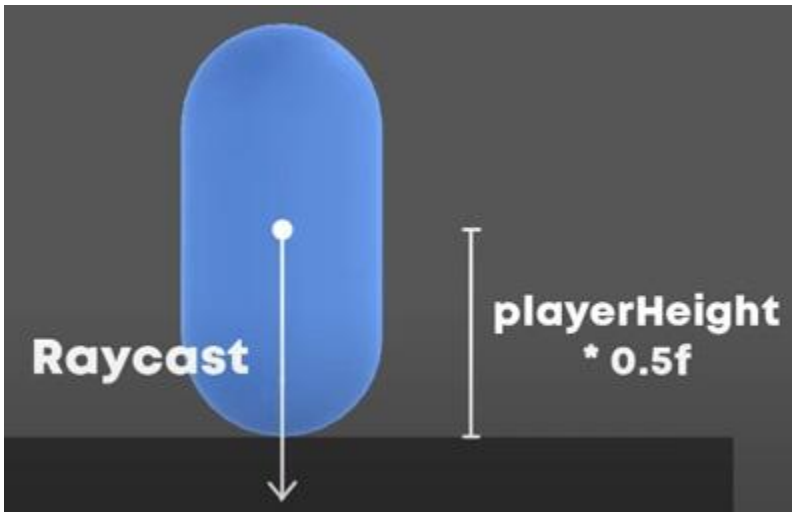
https://youtu.be/f473C43s8nE?si=Ww2eetEaE8hYKGk8&t=305

Apply drag to the player's ridgebody, which will make the movement less slippery and to limit the player's velocity to its movement speed

Apply Drag when Player on Ground

Ground Check

To perform the ground check, you want to shoot the raycast from your current position down, and see if it hits something, the length of this ray will be half of your player's height + a bit more.




```
    [Header("Movement")]
    public float groundDrag;
    [Header("Ground Check")]
    public float playerHeight;
    public LayerMask whatIsGround;
    bool grounded;

    private void Update()
    {
        GroundDrag();
    }

    private void GroundDrag()
    {
        // Perform ground check
        grounded = Physics.Raycast(transform.position, Vector3.down,
playerHeight * 0.5f + 0.3f, whatIsGround);

        // Apply drag when grounded
        rb.drag = grounded ? groundDrag : 0;
    }
```

Jumping & Air Control

https://youtu.be/f473C43s8nE?si=pxDbN1yNIVLISXS5&t=442

```
    public float jumpForce;
    public float jumpCooldown;
    public float airMultiplier;
    bool readyToJump;
```

```
private void MyInput()
    {
        // Check for jump input and conditions for jumping
        if (Input.GetKey(jumpKey) && readyToJump && grounded)
        {
            readyToJump = false;
            Debug.Log("Jump!");
            Jump();
            Invoke(nameof(ResetJump), jumpCooldown); // Allow to
continuously jump if the jump key is held down
        }
    }
```

```
private void Jump()
    {
        // Reset vertical velocity before applying jump force to avoid
accumulating jump force
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

        // Apply impulse force for jumping
        rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
    }

    private void ResetJump()
    {
        readyToJump = true;
    }
```

Code

PlayerMovement.cs

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

public class PlayerMovement : MonoBehaviour
{
    [Header("Movement")]
    public float moveSpeed;

    public float groundDrag;

    public float jumpForce;
    public float jumpCooldown;
    public float airMultiplier;
    bool readyToJump;

    [HideInInspector] public float walkSpeed;
    [HideInInspector] public float sprintSpeed;

    [Header("Keybinds")]
    public KeyCode jumpKey = KeyCode.Space;

    [Header("Ground Check")]
    public float playerHeight;
    public LayerMask whatIsGround;
    bool grounded;

    public Transform orientation;

    // To hold your input
    float horizontalInput;
    float verticalInput;

    Vector3 moveDirection;
```

```
    Rigidbody rb;

    private void Start()
    {
        // Assign your Rigidbody and freeze its rotation
        rb = GetComponent<Rigidbody>();
        rb.freezeRotation = true;

        readyToJump = true;
    }

    private void Update()
    {
        GroundDrag();
        MyInput(); // This will keep checking allowed input for all
movement
        SpeedControl();
    }

    private void FixedUpdate()
    {
        MovePlayer(); // To apply force on the player Rigidbody
    }

    // This method will handle your input, including movement and jumping,
and verify if jumping is allowed
    private void MyInput()
    {
        horizontalInput = Input.GetAxisRaw("Horizontal");
        verticalInput = Input.GetAxisRaw("Vertical");

        // Check for jump input and conditions for jumping
        if (Input.GetKey(jumpKey) && readyToJump && grounded)
        {
            readyToJump = false;

            Jump();

            Invoke(nameof(ResetJump), jumpCooldown); // Allow to
continuously jump if the jump key is held down
```

```
        }
    }

    private void MovePlayer()
    {
        // Calculate movement direction based on orientation
        moveDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;

        // Apply force for movement based on ground or air
        if (grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f,
ForceMode.Force);
        else if (!grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f *
airMultiplier, ForceMode.Force);
    }

    private void SpeedControl()
    {
        // Get the horizontal velocity
        Vector3 flatVel = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

        // Limit velocity if needed to prevent exceeding maximum speed
        if (flatVel.magnitude > moveSpeed)
        {
            Vector3 limitedVel = flatVel.normalized * moveSpeed;
            rb.velocity = new Vector3(limitedVel.x, rb.velocity.y,
limitedVel.z);
        }
    }

    private void Jump()
    {
        // Reset vertical velocity before applying jump force to avoid
accumulating jump force
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

        // Apply impulse force for jumping
        rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
```

```
    }

    private void ResetJump()
    {
        readyToJump = true;
    }

    private void GroundDrag()
    {
        // Perform ground check
        grounded = Physics.Raycast(transform.position, Vector3.down,
playerHeight * 0.5f + 0.3f, whatIsGround);

        // Apply drag when grounded
        rb.drag = grounded ? groundDrag : 0;
    }
}
```

Problem: Rigidbody won’t do rotation follow the direction of PlayerCam

To Fix:
First, create a serialized field for the PlayerCam object:

```
[SerializeField] private Transform playerCam;
```

Then, in your `Update()` or `FixedUpdate()` method, update the rotation of the Rigidbody to match the rotation of the PlayerCam object:

```
private void Update()
    {
        GroundDrag();
        MyInput(); // This will keep checking allowed input for all
movement
        SpeedControl();
        RotatePlayer();
    }
private void RotatePlayer()
    {
        if (playerCam != null)
        {
            // Set the Rigidbody's rotation to match the PlayerCam's
rotation
            rb.rotation = Quaternion.Euler(0f, playerCam.eulerAngles.y,
0f);
        }
    }
```

Code

PlayerMovement.cs Fix update rb rotation follow the roataion of separate object

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

public class PlayerMovement : MonoBehaviour
{
    [Header("Movement")]
    public float moveSpeed;

    public float groundDrag;

    public float jumpForce;
    public float jumpCooldown;
    public float airMultiplier;
    bool readyToJump;

    [HideInInspector] public float walkSpeed;
    [HideInInspector] public float sprintSpeed;

    [Header("Keybinds")]
    public KeyCode jumpKey = KeyCode.Space;

    [Header("Ground Check")]
    public float playerHeight;
    public LayerMask whatIsGround;
    bool grounded;

    public Transform orientation;

    [SerializeField] private Transform playerCam;

    // To hold your input
    float horizontalInput;
    float verticalInput;
```

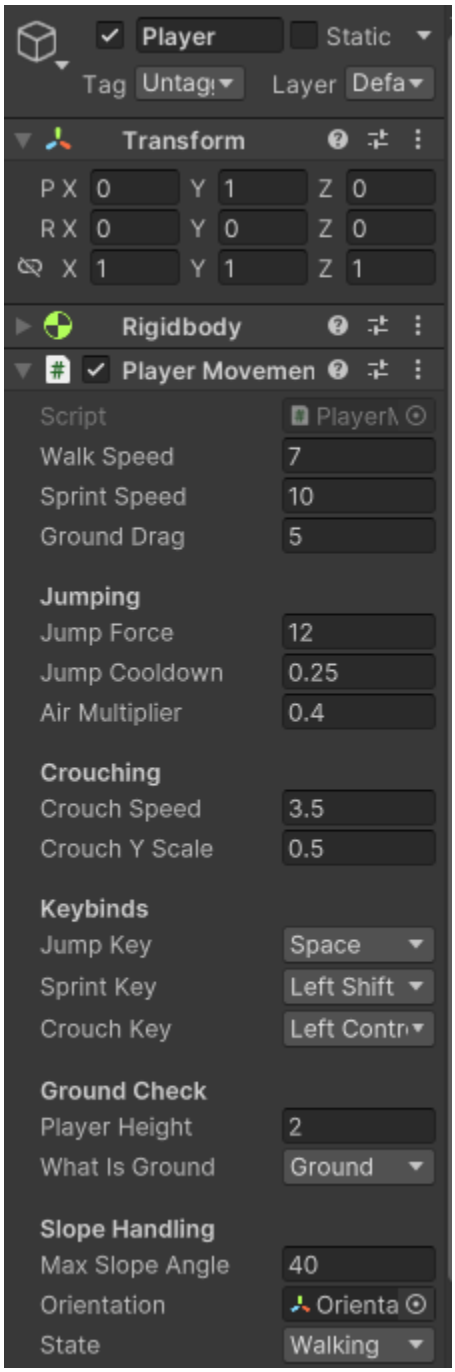
```
    Vector3 moveDirection;

    Rigidbody rb;

    private void Start()
    {
        // Assign your Rigidbody and freeze its rotation
        rb = GetComponent<Rigidbody>();
        rb.freezeRotation = true;

        readyToJump = true;
    }

    private void Update()
    {
        GroundDrag();
        MyInput(); // This will keep checking allowed input for all
movement
        SpeedControl();
        RotatePlayer();
    }

    private void FixedUpdate()
    {
        MovePlayer(); // To apply force on the player Rigidbody
    }

    // This method will handle your input, including movement and jumping,
and verify if jumping is allowed
    private void MyInput()
    {
        horizontalInput = Input.GetAxisRaw("Horizontal");
        verticalInput = Input.GetAxisRaw("Vertical");

        // Check for jump input and conditions for jumping
        if (Input.GetKey(jumpKey) && readyToJump && grounded)
        {
            readyToJump = false;
            Debug.Log("Jump!");
```

```
            Jump();

            Invoke(nameof(ResetJump), jumpCooldown); // Allow to
continuously jump if the jump key is held down
        }
    }

    private void MovePlayer()
    {
        // Calculate movement direction based on orientation
        moveDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;

        // Apply force for movement based on ground or air
        if (grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f,
ForceMode.Force);
        else if (!grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f *
airMultiplier, ForceMode.Force);
    }

    private void SpeedControl()
    {
        // Get the horizontal velocity
        Vector3 flatVel = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

        // Limit velocity if needed to prevent exceeding maximum speed
        if (flatVel.magnitude > moveSpeed)
        {
            Vector3 limitedVel = flatVel.normalized * moveSpeed;
            rb.velocity = new Vector3(limitedVel.x, rb.velocity.y,
limitedVel.z);
        }
    }

    private void Jump()
    {
        // Reset vertical velocity before applying jump force to avoid
accumulating jump force
```

```
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

        // Apply impulse force for jumping
        rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
    }

    private void ResetJump()
    {
        readyToJump = true;
    }

    private void GroundDrag()
    {
        // Perform ground check
        grounded = Physics.Raycast(transform.position, Vector3.down,
playerHeight * 0.5f + 0.3f, whatIsGround);

        // Apply drag when grounded
        rb.drag = grounded ? groundDrag : 0;
    }

    private void RotatePlayer()
    {
        if (playerCam != null)
        {
            // Set the Rigidbody's rotation to match the PlayerCam's
rotation
            rb.rotation = Quaternion.Euler(0f, playerCam.eulerAngles.y,
0f);
        }
    }
}
```

SLOPE MOVEMENT, SPRINTING & CROUCHING - Unity Tutorial

https://www.youtube.com/watch?v=xCxSjgYTw9c



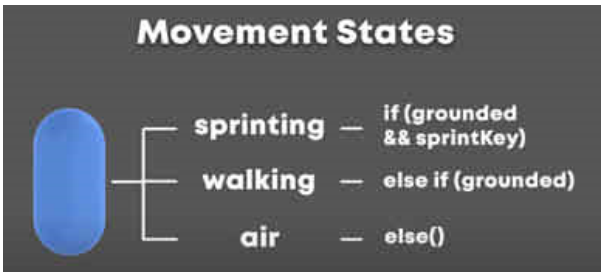
Summary

Have different movement states include:
Walking, Sprinting, Jumping, Crouching
Have basic On slope movement
Cons:
Crouching is rescale function, just squeeze the transform Scale
On slope movement will turn off rb.gravity while rb on Slope, however jump movespeed have no different between onslope or off slope, because it always apply air movement speed for jump
Missing
Cam rotation apply on Rig rotation still missing
Animation didn't apply on move
Cinemachine missing

PlayerMovementAdvanced.cs in Player [GameObject]

To code the sprinting ability
https://youtu.be/xCxSjgYTw9c?si=7IqvPFZ6LCN2tr71&t=53

To Create movement states for our player, depending on which keys you're pressing.
the player will enter a different state



Sprinting

```
    [Header("Keybinds")]
    public KeyCode jumpKey = KeyCode.Space;
    public KeyCode sprintKey = KeyCode.LeftShift;
    public KeyCode crouchKey = KeyCode.LeftControl;
```

```
    public MovementState state; // To store the current state of the
player
    public enum MovementState
    {
        walking,
        sprinting,
        crouching,
        air
    }
```

To set your movement state to different state depend on input

```
StateHandler()
```

```
private void StateHandler()  // Change MovementState and value depend on
input
    {
        // Mode - Crouching
        if (Input.GetKey(crouchKey))  // If crouchKey down
        {
            state = MovementState.crouching; // Change crouch State
            moveSpeed = crouchSpeed;  // Set the speed
        }

        // Mode - Sprinting
        else if(grounded && Input.GetKey(sprintKey)) // Do Sprinting here
        {
            state = MovementState.sprinting;
            moveSpeed = sprintSpeed;
        }

        // Mode - Walking
        else if (grounded)
        {
            state = MovementState.walking; // Stop Sprint
```

```
            moveSpeed = walkSpeed;
        }

        // Mode - Air
        else
        {
            state = MovementState.air;
        }
    }
```

Last,

```
private void Update()
    {
        MyInput();
        SpeedControl();
        StateHandler(); // always update statehandler, tracking the input
    }
```

https://youtu.be/xCxSjgYTw9c?si=GHeNVmfZOknt7564&t=118

Crouching

```
    [Header("Crouching")]
    public float crouchSpeed;
    public float crouchYScale;
    private float startYScale; // to store the original yScale
```

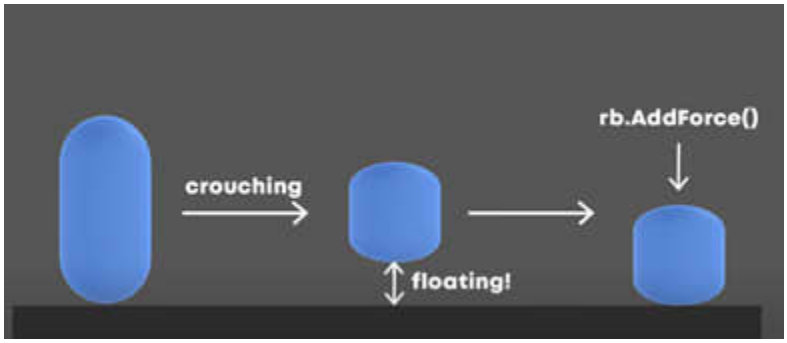
```
private void Start()
    {
        rb = GetComponent<Rigidbody>();
        rb.freezeRotation = true;

        readyToJump = true;

        startYScale = transform.localScale.y;
    }
```

Check crouchKey input, invoke Crouch Scale

```
    private void MyInput()
```

```
    {
        // start crouch
        if (Input.GetKeyDown(crouchKey)) // check if put down crouchKey
        {  // To shrink your player down by setting your local scale to a new vector3, keep the x and z scale the same, but change the y scale to your crouch y scale
            transform.localScale = new Vector3(transform.localScale.x, crouchYScale, transform.localScale.z);
            rb.AddForce(Vector3.down * 5f, ForceMode.Impulse); // push rb down
        }

        // stop crouch
        if (Input.GetKeyUp(crouchKey))
        {
            transform.localScale = new Vector3(transform.localScale.x, startYScale, transform.localScale.z);
        }
    }
```

There have problem if changed player scale down, it will floating in the air, so need to add downward force to quickly push the player on the ground



https://youtu.be/xCxSjgYTw9c?si=zRWvpuZw-Bf5LopL&t=162

Change the state with StateHandler according Input

```
    private void StateHandler()
    {
        // Mode - Crouching
        if (Input.GetKey(crouchKey)) // if key pressed
        {
            state = MovementState.crouching; // Change the state
            moveSpeed = crouchSpeed; // change the speed, reduced
        }
```

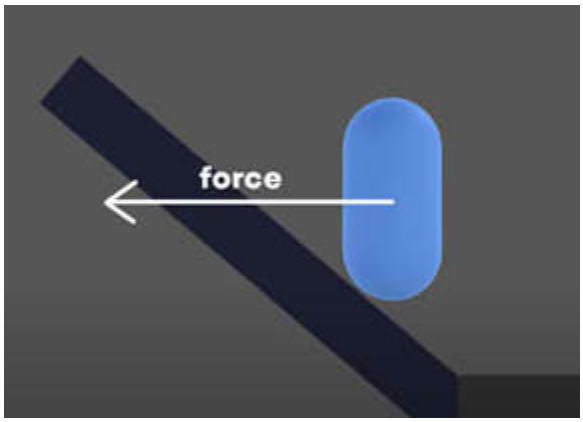
```
        // Mode - Sprinting
        else if(grounded && Input.GetKey(sprintKey))
    }
```

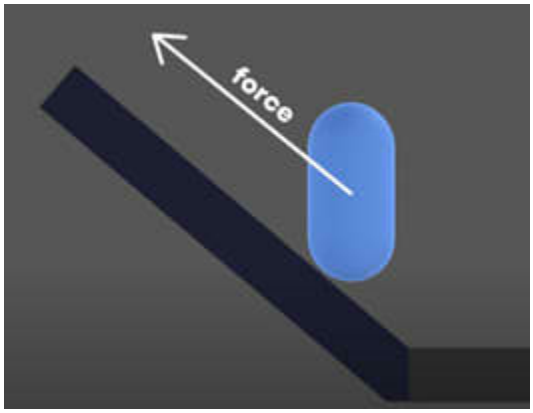
https://youtu.be/xCxSjgYTw9c?si=uezzoPqO5iqKj4K2&t=214

Slope Movement

https://youtu.be/xCxSjgYTw9c?si=8Tn5m0T5gBZo9kSa&t=225

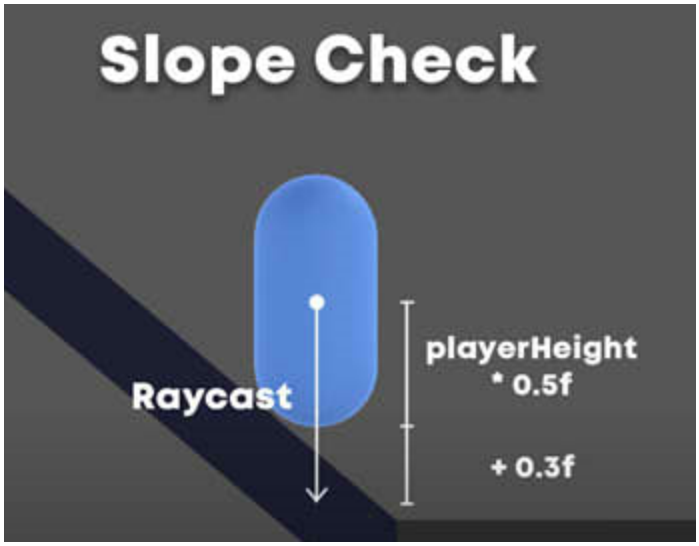
To get better slope movement
Don't adding force directly into the slope
Apply force relative to the angle of the slope





First, check if the player is even standing on the slope

```
    [Header("Slope Handling")]
    public float maxSlopeAngle;
    private RaycastHit slopeHit;
    private bool exitingSlope;
```



Shoot raycast downwards, and the length will be half of our player's height + a bit more (like the ground check)

To find the correct direction relative to our slope

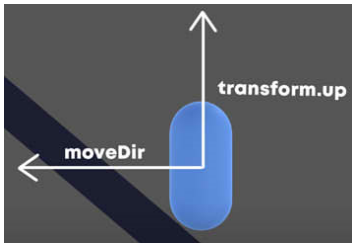
```
    private bool OnSlope()
    {
        if(Physics.Raycast(transform.position, Vector3.down, out slopeHit,
playerHeight * 0.5f + 0.3f))
        {
// Calculate the angle between the player's direction and the surface
normal
            float angle = Vector3.Angle(Vector3.up, slopeHit.normal);
            // Determine if the angle is within the acceptable slope range
            return angle < maxSlopeAngle && angle != 0;
        }
        return false;
    }//
// slopeHit stores the information of the object we hit in the slope hit
variable
// with Vector3.Angle we can calculate how steep the slope per standard is
// and we want the bool to return true if the angle is smaller than our
max slope angle and not zero.
// if the raycast doesn't hit anything, the bool should return false
```

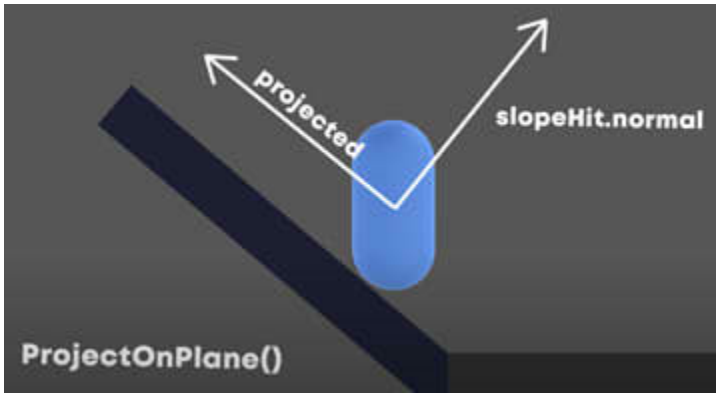
Find the correct direction relative to our slope

```
private Vector3 GetSlopeMoveDirection()
    {
```

```
        return Vector3.ProjectOnPlane(moveDirection,
slopeHit.normal).normalized;
    }
// use the project on plane function passing in your move direction and
the slopeHit.normal
```

Now
We projected our normal move direction onto the slope






```
private void MovePlayer()
    {
        // calculate movement direction
        moveDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;
        // on slope
        if (OnSlope() && !exitingSlope)
        {
            rb.AddForce(GetSlopeMoveDirection() * moveSpeed * 20f,
ForceMode.Force); // it will put extra force when rb on slope

            if (rb.velocity.y > 0)
                rb.AddForce(Vector3.down * 80f, ForceMode.Force);
        }
        // on ground
        else if(grounded)

    }
```

https://youtu.be/xCxSjgYTw9c?si=tmD_UQGpb2DUGRs3&t=355

Now, when rb is on slope, it will slide down the slope because of gravity

https://youtu.be/xCxSjgYTw9c?si=Lqp9hj9jFaXS3M89&t=360
Turn off the rb's gravity while we're standing on a slope. (not a good way)

```
 private void MovePlayer()
    {
        // turn gravity off while on slope
        rb.useGravity = !OnSlope();
    }
```

Weird bumping movement if gravity off on slope while moving up on slope
https://youtu.be/xCxSjgYTw9c?si=LRjHaYxcZqUzKUwb&t=391

```
    private void MovePlayer()
    {
        // calculate movement direction
        moveDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;

        // on slope
        if (OnSlope() && !exitingSlope)
        {
            rb.AddForce(GetSlopeMoveDirection() * moveSpeed * 20f,
ForceMode.Force);

            if (rb.velocity.y > 0)
                rb.AddForce(Vector3.down * 80f, ForceMode.Force);
        }
```

Moving too fast on slope
https://youtu.be/xCxSjgYTw9c?si=7H5PG_T8ksDRDOy-&t=414

Because of SpeedControl()
Limit the player's velocity to our move speed, while player is on slope and not exist, even it is jumping on slope, no matter in which direction the player is going

```
    private void SpeedControl()
    {
        // limiting speed on slope
        if (OnSlope() && !exitingSlope)
        {
            if (rb.velocity.magnitude > moveSpeed)
                rb.velocity = rb.velocity.normalized * moveSpeed;
        }
```

Can’t jump
https://youtu.be/xCxSjgYTw9c?si=nGVZVjt8-7FTU74d&t=459

```
    [Header("Slope Handling")]
    public float maxSlopeAngle;
    private RaycastHit slopeHit;
    private bool exitingSlope; // add this line
```

```
    private void Jump()
    {
        exitingSlope = true; // if you are jumping, set it to true
        // reset y velocity
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);
        rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
    }
```

```
    private void ResetJump() // for continuous jump
    {
        readyToJump = true;
        exitingSlope = false; // set it to false to reset your jump
    }
```

Only apply the limitation and slope movement if you’re not trying to exit the slope

```
    private void SpeedControl()
    {
        // limiting speed on slope
        if (OnSlope() && !exitingSlope) // Here, now won’t do speed limit
while jumping on slope
        {
            if (rb.velocity.magnitude > moveSpeed)
                rb.velocity = rb.velocity.normalized * moveSpeed;
        }
```

```
    private void MovePlayer()
    {
        // calculate movement direction
        moveDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;
```

```
        // on slope
        if (OnSlope() && !exitingSlope) // Here
        {
            rb.AddForce(GetSlopeMoveDirection() * moveSpeed * 20f,
ForceMode.Force);

            // since we turn off the gravity on slope
            // if the player is moving upwards which means its y velocity
is greater than zero
            if (rb.velocity.y > 0)
            // we add a bit of downward force to keep the player
constantly on the slope
                rb.AddForce(Vector3.down * 80f, ForceMode.Force);
        }
```

Code

PlayerMovementAdvanced.cs

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

public class PlayerMovementAdvanced : MonoBehaviour
{
    [Header("Movement")]
    private float moveSpeed;
    public float walkSpeed;
    public float sprintSpeed;

    public float groundDrag;

    [Header("Jumping")]
    public float jumpForce;
    public float jumpCooldown;
    public float airMultiplier;
    bool readyToJump;
```

```
    [Header("Crouching")]
    public float crouchSpeed;
    public float crouchYScale;
    private float startYScale;

    [Header("Keybinds")]
    public KeyCode jumpKey = KeyCode.Space;
    public KeyCode sprintKey = KeyCode.LeftShift;
    public KeyCode crouchKey = KeyCode.LeftControl;

    [Header("Ground Check")]
    public float playerHeight;
    public LayerMask whatIsGround;
    bool grounded;

    [Header("Slope Handling")]
    public float maxSlopeAngle;
    private RaycastHit slopeHit;
    private bool exitingSlope;


    public Transform orientation;

    float horizontalInput;
    float verticalInput;

    Vector3 moveDirection;

    Rigidbody rb;

    public MovementState state;
    public enum MovementState
    {
        walking,
        sprinting,
        crouching,
        air
    }

    private void Start()
```

```
    {
        rb = GetComponent<Rigidbody>();
        rb.freezeRotation = true;

        readyToJump = true;

        startYScale = transform.localScale.y;
    }

    private void Update()
    {
        // ground check
        grounded = Physics.Raycast(transform.position, Vector3.down,
playerHeight * 0.5f + 0.2f, whatIsGround);

        MyInput();
        SpeedControl();
        StateHandler();

        // handle drag
        if (grounded)
            rb.drag = groundDrag;
        else
            rb.drag = 0;
    }

    private void FixedUpdate()
    {
        MovePlayer();
    }

    private void MyInput()
    {
        horizontalInput = Input.GetAxisRaw("Horizontal");
        verticalInput = Input.GetAxisRaw("Vertical");

        // when to jump
        if(Input.GetKey(jumpKey) && readyToJump && grounded)
        {
            readyToJump = false;
```

```
            Jump();

            Invoke(nameof(ResetJump), jumpCooldown);
        }

        // start crouch
        if (Input.GetKeyDown(crouchKey))
        {
            transform.localScale = new Vector3(transform.localScale.x,
crouchYScale, transform.localScale.z);
            rb.AddForce(Vector3.down * 5f, ForceMode.Impulse);
        }

        // stop crouch
        if (Input.GetKeyUp(crouchKey))
        {
            transform.localScale = new Vector3(transform.localScale.x,
startYScale, transform.localScale.z);
        }
    }

    private void StateHandler()
    {
        // Mode - Crouching
        if (Input.GetKey(crouchKey))
        {
            state = MovementState.crouching;
            moveSpeed = crouchSpeed;
        }

        // Mode - Sprinting
        else if(grounded && Input.GetKey(sprintKey))
        {
            state = MovementState.sprinting;
            moveSpeed = sprintSpeed;
        }

        // Mode - Walking
        else if (grounded)
```

```
        {
            state = MovementState.walking;
            moveSpeed = walkSpeed;
        }

        // Mode - Air
        else
        {
            state = MovementState.air;
        }
    }

    private void MovePlayer()
    {
        // calculate movement direction
        moveDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;

        // on slope
        if (OnSlope() && !exitingSlope)
        {
            rb.AddForce(GetSlopeMoveDirection() * moveSpeed * 20f,
ForceMode.Force);

            // since we turn off the gravity on slope
            // if the player is moving upwards which means its y velocity
is greater than zero
            if (rb.velocity.y > 0)
            // we add a bit of downward force to keep the player
constantly on the slope
                rb.AddForce(Vector3.down * 80f, ForceMode.Force);
        }

        // on ground
        else if(grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f,
ForceMode.Force);

        // in air
        else if(!grounded)
```

```
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f *
airMultiplier, ForceMode.Force);

        // turn gravity off while on slope
        rb.useGravity = !OnSlope();
    }

    private void SpeedControl()
    {
        // limiting speed on slope
        if (OnSlope() && !exitingSlope)
        {
            if (rb.velocity.magnitude > moveSpeed)
                rb.velocity = rb.velocity.normalized * moveSpeed;
        }

        // limiting speed on ground or in air
        else
        {
            Vector3 flatVel = new Vector3(rb.velocity.x, 0f,
rb.velocity.z);

            // limit velocity if needed
            if (flatVel.magnitude > moveSpeed)
            {
                Vector3 limitedVel = flatVel.normalized * moveSpeed;
                rb.velocity = new Vector3(limitedVel.x, rb.velocity.y,
limitedVel.z);
            }
        }
    }

    private void Jump()
    {
        exitingSlope = true;

        // reset y velocity
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

        rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
```

```
    }
    private void ResetJump()
    {
        readyToJump = true;

        exitingSlope = false;
    }

    private bool OnSlope()
    {
        if(Physics.Raycast(transform.position, Vector3.down, out slopeHit,
playerHeight * 0.5f + 0.3f))
        {
            // Calculate the angle between the player's direction and the
surface normal
            float angle = Vector3.Angle(Vector3.up, slopeHit.normal);

            // Determine if the angle is within the acceptable slope range
            return angle < maxSlopeAngle && angle != 0;
        }
        return false;
    }

    private Vector3 GetSlopeMoveDirection()
    {
        return Vector3.ProjectOnPlane(moveDirection,
slopeHit.normal).normalized;
    }
}
```

ADVANCED SLIDING IN 9 MINUTES - Unity Tutorial

https://www.youtube.com/watch?v=SsckrYYxcuM

To make your players slide in any direction,
As well as how to slide down slopes
Build up speed while doing so

Sliding.cs

```
    [Header("References")]
    public Transform orientation; // just an empty game object that keeps track of where the player is looking
    public Transform playerObj; // transform of playerObj
    private Rigidbody rb;
    private PlayerMovementAdvanced pm; // Also reference your movement script
```

```
    [Header("Sliding")]
    public float maxSlideTime; // for maximum time you're allowed to slide
    public float slideForce; // the slide force
    private float slideTimer; // a timer to check how long you've been sliding already

    public float slideYScale; // to shrink the player down while sliding
    private float startYScale; // reset the slide y scale after slide
```

```
    [Header("Input")]
    public KeyCode slideKey = KeyCode.LeftControl; // define key code for slide key
    private float horizontalInput; // also direction input
    private float verticalInput;
```

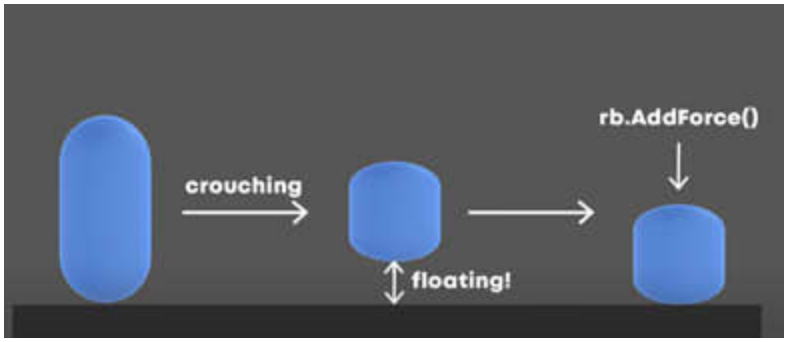
```
    private void Start()
    {
        rb = GetComponent<Rigidbody>(); // get rigidbody component
        pm = GetComponent<PlayerMovementAdvanced>(); // get movement script
// save y scale of player for crouch and sliding
```

```
        startYScale = playerObj.localScale.y;     }
```

```
    private void Update()
    {
        horizontalInput = Input.GetAxisRaw("Horizontal");
        verticalInput = Input.GetAxisRaw("Vertical");

        if (Input.GetKeyDown(slideKey) && (horizontalInput != 0 ||
verticalInput != 0))
            StartSlide(); // call slide if slide key down with direction
down

        if (Input.GetKeyUp(slideKey) && pm.sliding)
            StopSlide(); // stop slide if slide key up and in slide state
    }
```

```
    private void FixedUpdate()
    {
        if (pm.sliding)
            SlidingMovement(); // while is sliding, call function
    }
```

```
    private void StartSlide()
    { // when do slide
        pm.sliding = true; // set the bool sliding in Movement.cs true
// only change the y scale while leaving x and z scale as they are
        playerObj.localScale = new Vector3(playerObj.localScale.x,
slideYScale, playerObj.localScale.z);
        rb.AddForce(Vector3.down * 5f, ForceMode.Impulse);
// add down force to push rb down, because of floating
        slideTimer = maxSlideTime; // reset the slide timer
    }
```




```
    private void SlidingMovement() // apply sliding force here
    { // calculate the input direction, forward direction of the player * your vertical input + right direction of your player * your horizontal input
        Vector3 inputDirection = orientation.forward * verticalInput + orientation.right * horizontalInput;
// this way you can slide in all directions depending on which keys you're pressing

        // sliding normal
        if(!pm.OnSlope() || rb.velocity.y > -0.1f)
        { // apply force in the calculated direction
// use normalized input direction
            rb.AddForce(inputDirection.normalized * slideForce, ForceMode.Force);
// while sliding, count down your slide timer
            slideTimer -= Time.deltaTime;
        }

        // sliding down a slope
        else
        {
            rb.AddForce(pm.GetSlopeMoveDirection(inputDirection) * slideForce, ForceMode.Force);
        }
// call stop slide function if slidetimer reaches zero
        if (slideTimer <= 0)
            StopSlide(); // call for set the bool pm.sliding to false
    }
```

```
    private void StopSlide()
```

```
    {
        pm.sliding = false; // call for set the bool pm.sliding to false
// reset player y scale back to normal
        playerObj.localScale = new Vector3(playerObj.localScale.x,
startYScale, playerObj.localScale.z);
    }
```

https://youtu.be/SsckrYYxcuM?si=ewtY7aU6V9Kd5FvD&t=228

To fix sliding down slopes bumping movement
https://youtu.be/SsckrYYxcuM?si=dL4A24IQl61oMEMK&t=260

```
    private void SlidingMovement()
    {
        Vector3 inputDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;

        // sliding normal
        if(!pm.OnSlope() || rb.velocity.y > -0.1f)
        {
// this will only be executed when the player is not on a slope or moving
upwards
            rb.AddForce(inputDirection.normalized * slideForce,
ForceMode.Force);

            slideTimer -= Time.deltaTime;
        }
        // sliding down a slope
        else
        {
// when the player is on a slope and moving downwards, you want to apply
the force in the slope movement direction
            rb.AddForce(pm.GetSlopeMoveDirection(inputDirection) *
slideForce, ForceMode.Force);
        }

        if (slideTimer <= 0)
            StopSlide();
    }
```

https://youtu.be/SsckrYYxcuM?si=--kekHN1K8Bn_hbI&t=337

Build up speed over time

```
    [Header("Movement")]
    private float moveSpeed;
    public float walkSpeed;
    public float sprintSpeed;

    public float slideSpeed; // new
    private float desiredMoveSpeed; // new
    private float lastDesiredMoveSpeed; // new
```

```
    public enum MovementState
    {
        walking,
        sprinting,
        crouching,
        sliding, // new
        air
    }

    public bool sliding; // new
```

```
    private void StateHandler()
    {
        // Mode - Sliding
        if (sliding) // new
        {
            state = MovementState.sliding;
// if player is on slope and move downwards, set desiredMoveSpeed to
slideSpeed
            if (OnSlope() && rb.velocity.y < 0.1f)
                desiredMoveSpeed = slideSpeed;

            else
                desiredMoveSpeed = sprintSpeed;
        }

        // Mode - Crouching
        else if
```

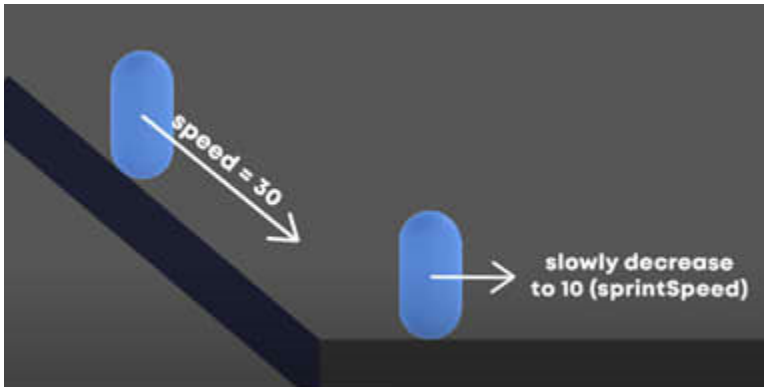
Change moveSpeed variables in PlayerMovementAdvanced.cs to desiredMovement.
The reason for change is that we're now implementing momentum冲力 into our game.

To handle Speed Limitations differently

https://youtu.be/SsckrYYxcuM?si=G7qPM6wd0ddf7iJ8&t=398

For example

If the player builds up a speed of 30 on a slope, and then hits the ground.
You don't want the speed to instantly drop to 10.
Instead it should slowly decrease.



For this, we're going to use Mathf.Lerp inside of this simple quarantine隔离
This script **changing** the **movespeed** variable to **desiredMoveSpeed** (**overtime**)

```
    private IEnumerator SmoothlyLerpMoveSpeed()
    {
        // smoothly lerp movementSpeed to desired value
        float time = 0;
        float difference = Mathf.Abs(desiredMoveSpeed - moveSpeed);
        float startValue = moveSpeed;

        while (time < difference)
        {
            moveSpeed = Mathf.Lerp(startValue, desiredMoveSpeed, time /
difference);

            if (OnSlope())
            {
```

```
                float slopeAngle = Vector3.Angle(Vector3.up,
slopeHit.normal);
                float slopeAngleIncrease = 1 + (slopeAngle / 90f);

                time += Time.deltaTime * speedIncreaseMultiplier *
slopeIncreaseMultiplier * slopeAngleIncrease;
            }
            else
                time += Time.deltaTime * speedIncreaseMultiplier;

            yield return null;
        }

        moveSpeed = desiredMoveSpeed;
    }
```

Save the last desired move speed at the end of the state handler,
And check if the difference of the desiredMovespeed to the last desired movespeed is greater than 4.
If so, start coroutine
If not, set the value directly

```
    private void StateHandler()
    {// TL;DR
        // Mode - Air
        else
        {
            state = MovementState.air;
        }

        // check if desiredMoveSpeed has changed drastically
        if(Mathf.Abs(desiredMoveSpeed - lastDesiredMoveSpeed) > 4f &&
moveSpeed != 0)
        {
            StopAllCoroutines();
            StartCoroutine(SmoothlyLerpMoveSpeed());
        }
        else
        {
```

```
            moveSpeed = desiredMoveSpeed;
        }

        lastDesiredMoveSpeed = desiredMoveSpeed;
    }
```

https://youtu.be/SsckrYYxcuM?si=pTW7nkEPoFcFUUkW&t=457

Why only change it if the difference is Greater than 4f?

check if the difference of the desiredMovespeed to the last desired movespeed is greater than 4.

```
        // check if desiredMoveSpeed has changed drastically
        if(Mathf.Abs(desiredMoveSpeed - lastDesiredMoveSpeed) > 4f &&
moveSpeed != 0)
```

If you're changing from walking to sprinting, the speed difference is only 3.
Therefore the speed changes instantly.

But if you build up a speed of 30, and you're changing to sprinting, the difference is 20, which is greater than 4, which means the speed will now slowly decrease.

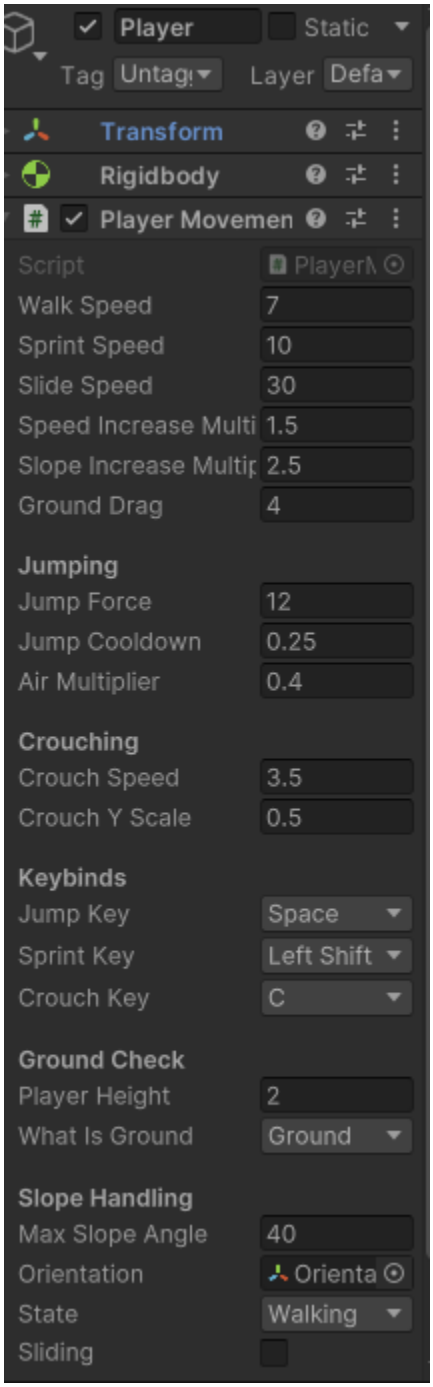
This way you have it both.
On one side, you can quickly change between sprinting and walking.
On the other side, you slowly change between going really fast and really slow.
You're able to keep your momentum动量 冲力

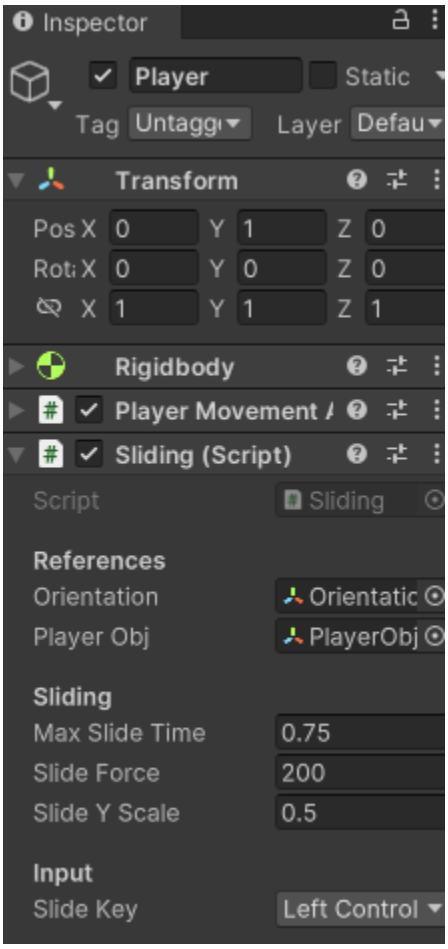
Set your value
https://youtu.be/SsckrYYxcuM?si=9DPOQxtaA32N6jYK&t=505

https://youtu.be/SsckrYYxcuM?si=KMVfsj4jHZNQycpA&t=516
Build up more speed depending on how steep the slope is

```
    [Header("Movement")]
// TL;DL
    public float speedIncreaseMultiplier; // new
    public float slopeIncreaseMultiplier; // new
```

Value for Script





Problem:

Rig does not rotate as PlayerCam's rotation.
Cannot jump on a very steep slope in any direction.
It won’t count steep slopes as being on the ground.
Always in the air on steep slopes.

Code

Sliding.cs

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

public class Sliding : MonoBehaviour
{
    [Header("References")]
    public Transform orientation;
    public Transform playerObj;
    private Rigidbody rb;
    private PlayerMovementAdvanced pm;

    [Header("Sliding")]
    public float maxSlideTime;
    public float slideForce;
    private float slideTimer;

    public float slideYScale;
    private float startYScale;

    [Header("Input")]
    public KeyCode slideKey = KeyCode.LeftControl;
    private float horizontalInput;
    private float verticalInput;


    private void Start()
    {
        rb = GetComponent<Rigidbody>();
        pm = GetComponent<PlayerMovementAdvanced>();

        startYScale = playerObj.localScale.y;
    }

    private void Update()
    {
```

```
        horizontalInput = Input.GetAxisRaw("Horizontal");
        verticalInput = Input.GetAxisRaw("Vertical");

        if (Input.GetKeyDown(slideKey) && (horizontalInput != 0 ||
verticalInput != 0))
            StartSlide();

        if (Input.GetKeyUp(slideKey) && pm.sliding)
            StopSlide();
    }

    private void FixedUpdate()
    {
        if (pm.sliding)
            SlidingMovement();
    }

    private void StartSlide()
    {
        pm.sliding = true;

        playerObj.localScale = new Vector3(playerObj.localScale.x,
slideYScale, playerObj.localScale.z);
        rb.AddForce(Vector3.down * 5f, ForceMode.Impulse);

        slideTimer = maxSlideTime;
    }

    private void SlidingMovement()
    {
        Vector3 inputDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;

        // sliding normal
        if(!pm.OnSlope() || rb.velocity.y > -0.1f)
        {
            rb.AddForce(inputDirection.normalized * slideForce,
ForceMode.Force);

            slideTimer -= Time.deltaTime;
```

```
        }

        // sliding down a slope
        else
        {
            rb.AddForce(pm.GetSlopeMoveDirection(inputDirection) *
slideForce, ForceMode.Force);
        }

        if (slideTimer <= 0)
            StopSlide();
    }

    private void StopSlide()
    {
        pm.sliding = false;

        playerObj.localScale = new Vector3(playerObj.localScale.x,
startYScale, playerObj.localScale.z);
    }
}
```

PlayerMovementAdvanced.cs

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

public class PlayerMovementAdvanced : MonoBehaviour
{
    [Header("Movement")]
    private float moveSpeed;
    public float walkSpeed;
    public float sprintSpeed;

    public float slideSpeed; // new
    private float desiredMoveSpeed; // new
    private float lastDesiredMoveSpeed; // new
```

```
    public float speedIncreaseMultiplier; // new
    public float slopeIncreaseMultiplier; // new

    public float groundDrag;

    [Header("Jumping")]
    public float jumpForce;
    public float jumpCooldown;
    public float airMultiplier;
    bool readyToJump;

    [Header("Crouching")]
    public float crouchSpeed;
    public float crouchYScale;
    private float startYScale;

    [Header("Keybinds")]
    public KeyCode jumpKey = KeyCode.Space;
    public KeyCode sprintKey = KeyCode.LeftShift;
    public KeyCode crouchKey = KeyCode.LeftControl;

    [Header("Ground Check")]
    public float playerHeight;
    public LayerMask whatIsGround;
    bool grounded;

    [Header("Slope Handling")]
    public float maxSlopeAngle;
    private RaycastHit slopeHit;
    private bool exitingSlope;


    public Transform orientation;

    float horizontalInput;
    float verticalInput;

    Vector3 moveDirection;

    Rigidbody rb;
```

```
    public MovementState state;
    public enum MovementState
    {
        walking,
        sprinting,
        crouching,
        sliding, // new
        air
    }

    public bool sliding; // new

    private void Start()
    {
        rb = GetComponent<Rigidbody>();
        rb.freezeRotation = true;

        readyToJump = true;

        startYScale = transform.localScale.y;
    }

    private void Update()
    {
        // ground check
        grounded = Physics.Raycast(transform.position, Vector3.down,
playerHeight * 0.5f + 0.2f, whatIsGround);

        MyInput();
        SpeedControl();
        StateHandler();

        // handle drag
        if (grounded)
            rb.drag = groundDrag;
        else
            rb.drag = 0;
    }
```

```
    private void FixedUpdate()
    {
        MovePlayer();
    }

    private void MyInput()
    {
        horizontalInput = Input.GetAxisRaw("Horizontal");
        verticalInput = Input.GetAxisRaw("Vertical");

        // when to jump
        if(Input.GetKey(jumpKey) && readyToJump && grounded)
        {
            readyToJump = false;

            Jump();

            Invoke(nameof(ResetJump), jumpCooldown);
        }

        // start crouch
        if (Input.GetKeyDown(crouchKey))
        {
            transform.localScale = new Vector3(transform.localScale.x,
crouchYScale, transform.localScale.z);
            rb.AddForce(Vector3.down * 5f, ForceMode.Impulse);
        }

        // stop crouch
        if (Input.GetKeyUp(crouchKey))
        {
            transform.localScale = new Vector3(transform.localScale.x,
startYScale, transform.localScale.z);
        }
    }

    private void StateHandler()
    {
        // Mode - Sliding
        if (sliding) // new
```

```
        {
            state = MovementState.sliding;

            if (OnSlope() && rb.velocity.y < 0.1f)
                desiredMoveSpeed = slideSpeed;

            else
                desiredMoveSpeed = sprintSpeed;
        }

        // Mode - Crouching
        else if (Input.GetKey(crouchKey)) // change to else if
        {
            state = MovementState.crouching;
            desiredMoveSpeed = crouchSpeed; // moveSpeed to
desiredMoveSpeed
        }

        // Mode - Sprinting
        else if(grounded && Input.GetKey(sprintKey))
        {
            state = MovementState.sprinting;
            desiredMoveSpeed = sprintSpeed; // moveSpeed to
desiredMoveSpeed
        }

        // Mode - Walking
        else if (grounded)
        {
            state = MovementState.walking;
            desiredMoveSpeed = walkSpeed;
        }

        // Mode - Air
        else
        {
            state = MovementState.air;
        }

        // check if desiredMoveSpeed has changed drastically
```

```
        if(Mathf.Abs(desiredMoveSpeed - lastDesiredMoveSpeed) > 4f &&
moveSpeed != 0)
        {
            StopAllCoroutines();
            StartCoroutine(SmoothlyLerpMoveSpeed());
        }
        else
        {
            moveSpeed = desiredMoveSpeed;
        }

        lastDesiredMoveSpeed = desiredMoveSpeed;
    }

    private IEnumerator SmoothlyLerpMoveSpeed()
    {
        // smoothly lerp movementSpeed to desired value
        float time = 0;
        float difference = Mathf.Abs(desiredMoveSpeed - moveSpeed);
        float startValue = moveSpeed;

        while (time < difference)
        {
            moveSpeed = Mathf.Lerp(startValue, desiredMoveSpeed, time /
difference);

            if (OnSlope())
            {
                float slopeAngle = Vector3.Angle(Vector3.up,
slopeHit.normal);
                float slopeAngleIncrease = 1 + (slopeAngle / 90f);

                time += Time.deltaTime * speedIncreaseMultiplier *
slopeIncreaseMultiplier * slopeAngleIncrease;
            }
            else
                time += Time.deltaTime * speedIncreaseMultiplier;

            yield return null;
        }
```

```
        moveSpeed = desiredMoveSpeed;
    }

    private void MovePlayer()
    {
        // calculate movement direction
        moveDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;

        // on slope
        if (OnSlope() && !exitingSlope)
        {
            rb.AddForce(GetSlopeMoveDirection(moveDirection) * moveSpeed *
20f, ForceMode.Force);

            // since we turn off the gravity on slope
            // if the player is moving upwards which means its y velocity
is greater than zero
            if (rb.velocity.y > 0)
            // we add a bit of downward force to keep the player
constantly on the slope
                rb.AddForce(Vector3.down * 80f, ForceMode.Force);
        }

        // on ground
        else if(grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f,
ForceMode.Force);

        // in air
        else if(!grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f *
airMultiplier, ForceMode.Force);

        // turn gravity off while on slope
        rb.useGravity = !OnSlope();
    }

    private void SpeedControl()
```

```
    {
        // limiting speed on slope
        if (OnSlope() && !exitingSlope)
        {
            if (rb.velocity.magnitude > moveSpeed)
                rb.velocity = rb.velocity.normalized * moveSpeed;
        }

        // limiting speed on ground or in air
        else
        {
            Vector3 flatVel = new Vector3(rb.velocity.x, 0f,
rb.velocity.z);

            // limit velocity if needed
            if (flatVel.magnitude > moveSpeed)
            {
                Vector3 limitedVel = flatVel.normalized * moveSpeed;
                rb.velocity = new Vector3(limitedVel.x, rb.velocity.y,
limitedVel.z);
            }
        }
    }

    private void Jump()
    {
        exitingSlope = true;

        // reset y velocity
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

        rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
    }
    private void ResetJump()
    {
        readyToJump = true;

        exitingSlope = false;
    }
```

```
    public bool OnSlope()
    {
        if(Physics.Raycast(transform.position, Vector3.down, out slopeHit,
playerHeight * 0.5f + 0.3f))
        {
            // Calculate the angle between the player's direction and the
surface normal
            float angle = Vector3.Angle(Vector3.up, slopeHit.normal);

            // Determine if the angle is within the acceptable slope range
            return angle < maxSlopeAngle && angle != 0;
        }
        return false;
    }

    public Vector3 GetSlopeMoveDirection(Vector3 direction)
    {
        return Vector3.ProjectOnPlane(direction,
slopeHit.normal).normalized;
    }
}
```

PlayerMovemetAdvanced.cs Another solution

```
using System.Collections;
using UnityEngine;

public class PlayerMovementAdvanced : MonoBehaviour
{
    [Header("Movement")]
    private float moveSpeed;
    public float walkSpeed;
    public float sprintSpeed;
    public float slideSpeed;
    private float desiredMoveSpeed;
    private float lastDesiredMoveSpeed;
    public float speedIncreaseMultiplier;
    public float slopeIncreaseMultiplier;
    public float groundDrag;
```

```
    [Header("Jumping")]
    public float jumpForce;
    public float jumpCooldown;
    public float airMultiplier;
    private bool readyToJump = true;

    [Header("Crouching")]
    public float crouchSpeed;
    public float crouchYScale;
    private float startYScale;

    [Header("Keybinds")]
    public KeyCode jumpKey = KeyCode.Space;
    public KeyCode sprintKey = KeyCode.LeftShift;
    public KeyCode crouchKey = KeyCode.LeftControl;

    [Header("Ground Check")]
    public float playerHeight;
    public LayerMask whatIsGround;
    private bool grounded;
    private bool onSteepGround;

    [Header("Slope Handling")]
    public float maxSlopeAngle;
    private RaycastHit slopeHit;
    private bool exitingSlope;

    public Transform orientation;
    private Rigidbody rb;
    private Vector3 moveDirection;
    public MovementState state;
    public enum MovementState
    {
        walking,
        sprinting,
        crouching,
        sliding,
        air
    }
    public bool sliding;
```

```
private void Start()
{
    rb = GetComponent<Rigidbody>();
    rb.freezeRotation = true;
    startYScale = transform.localScale.y;
}

private void Update()
{
    GroundCheck();
    MyInput();
    SpeedControl();
    StateHandler();
    rb.drag = grounded ? groundDrag : 0;
}

private void FixedUpdate()
{
    MovePlayer();
}

private void MyInput()
{
    horizontalInput = Input.GetAxisRaw("Horizontal");
    verticalInput = Input.GetAxisRaw("Vertical");

    if (Input.GetKeyUp(jumpKey) && !grounded && onSteepGround)
        Jump();

    if (Input.GetKey(jumpKey) && readyToJump && grounded)
    {
        readyToJump = false;
        Jump();
        Invoke(nameof(ResetJump), jumpCooldown);
    }

    if (Input.GetKeyDown(crouchKey))
    {
```

```
            transform.localScale = new Vector3(transform.localScale.x,
crouchYScale, transform.localScale.z);
            rb.AddForce(Vector3.down * 5f, ForceMode.Impulse);
        }

        if (Input.GetKeyUp(crouchKey))
            transform.localScale = new Vector3(transform.localScale.x,
startYScale, transform.localScale.z);
    }

    private void StateHandler()
    {
        if (sliding)
        {
            state = MovementState.sliding;
            desiredMoveSpeed = OnSlope() && rb.velocity.y < 0.1f ?
slideSpeed : sprintSpeed;
        }
        else if (Input.GetKey(crouchKey))
        {
            state = MovementState.crouching;
            desiredMoveSpeed = crouchSpeed;
        }
        else if (grounded && Input.GetKey(sprintKey))
        {
            state = MovementState.sprinting;
            desiredMoveSpeed = sprintSpeed;
        }
        else if (grounded)
        {
            state = MovementState.walking;
            desiredMoveSpeed = walkSpeed;
        }
        else
            state = MovementState.air;

        if (Mathf.Abs(desiredMoveSpeed - lastDesiredMoveSpeed) > 4f &&
moveSpeed != 0)
        {
            StopAllCoroutines();
```

```
            StartCoroutine(SmoothlyLerpMoveSpeed());
        }
        else
            moveSpeed = desiredMoveSpeed;

        lastDesiredMoveSpeed = desiredMoveSpeed;
    }

    private IEnumerator SmoothlyLerpMoveSpeed()
    {
        float time = 0;
        float difference = Mathf.Abs(desiredMoveSpeed - moveSpeed);
        float startValue = moveSpeed;

        while (time < difference)
        {
            moveSpeed = Mathf.Lerp(startValue, desiredMoveSpeed, time /
difference);

            if (OnSlope())
            {
                float slopeAngle = Vector3.Angle(Vector3.up,
slopeHit.normal);
                float slopeAngleIncrease = 1 + (slopeAngle / 90f);
                time += Time.deltaTime * speedIncreaseMultiplier *
slopeIncreaseMultiplier * slopeAngleIncrease;
            }
            else
                time += Time.deltaTime * speedIncreaseMultiplier;

            yield return null;
        }

        moveSpeed = desiredMoveSpeed;
    }

    private void MovePlayer()
    {
        moveDirection = orientation.forward * verticalInput +
orientation.right * horizontalInput;
```

```
        if (OnSlope() && !exitingSlope)
        {
            rb.AddForce(GetSlopeMoveDirection(moveDirection) * moveSpeed *
20f, ForceMode.Force);
            if (rb.velocity.y > 0)
                rb.AddForce(Vector3.down * 80f, ForceMode.Force);
        }
        else if (grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f,
ForceMode.Force);
        else if (!grounded)
            rb.AddForce(moveDirection.normalized * moveSpeed * 10f *
airMultiplier, ForceMode.Force);

        rb.useGravity = !OnSlope();
    }

    private void SpeedControl()
    {
        if (OnSlope() && !exitingSlope && rb.velocity.magnitude >
moveSpeed)
            rb.velocity = rb.velocity.normalized * moveSpeed;
        else
        {
            Vector3 flatVel = new Vector3(rb.velocity.x, 0f,
rb.velocity.z);
            if (flatVel.magnitude > moveSpeed)
            {
                Vector3 limitedVel = flatVel.normalized * moveSpeed;
                rb.velocity = new Vector3(limitedVel.x, rb.velocity.y,
limitedVel.z);
            }
        }
    }

    private void Jump()
    {
        exitingSlope = true;
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);
```

```
        rb.AddForce(transform.up * jumpForce, ForceMode.Impulse);
    }

    private void ResetJump()
    {
        readyToJump = true;
        exitingSlope = false;
    }

    public bool OnSlope()
    {
        if (Physics.Raycast(transform.position, Vector3.down, out
slopeHit, playerHeight * 0.5f + 0.3f))
            return Vector3.Angle(Vector3.up, slopeHit.normal) <
maxSlopeAngle && slopeHit.normal != Vector3.up;
        return false;
    }

    public Vector3 GetSlopeMoveDirection(Vector3 direction)
    {
        return Vector3.ProjectOnPlane(direction,
slopeHit.normal).normalized;
    }

    private void GroundCheck()
    {
        RaycastHit hit;
        if (Physics.Raycast(transform.position, Vector3.down, out hit,
playerHeight * 0.5f + 0.2f, whatIsGround))
        {
            grounded = true;
            onSteepGround = Vector3.Angle(hit.normal, Vector3.up) >
maxSlopeAngle;
        }
        else
        {
            grounded = false;
            onSteepGround = false;
        }
    }
```

```
}
```

ADVANCED WALL RUNNING - Unity Tutorial (Remastered)

https://www.youtube.com/watch?v=gNt9wBOrQO4

THIRD PERSON MOVEMENT in 11 MINUTES - Unity Tutorial

https://www.youtube.com/watch?v=UCwwn2q4Vys

Different Orbits value for CinemachineFreeLook
https://youtu.be/UCwwn2q4Vys?si=WsGNXDFBfr4Ftr4o&t=145