

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

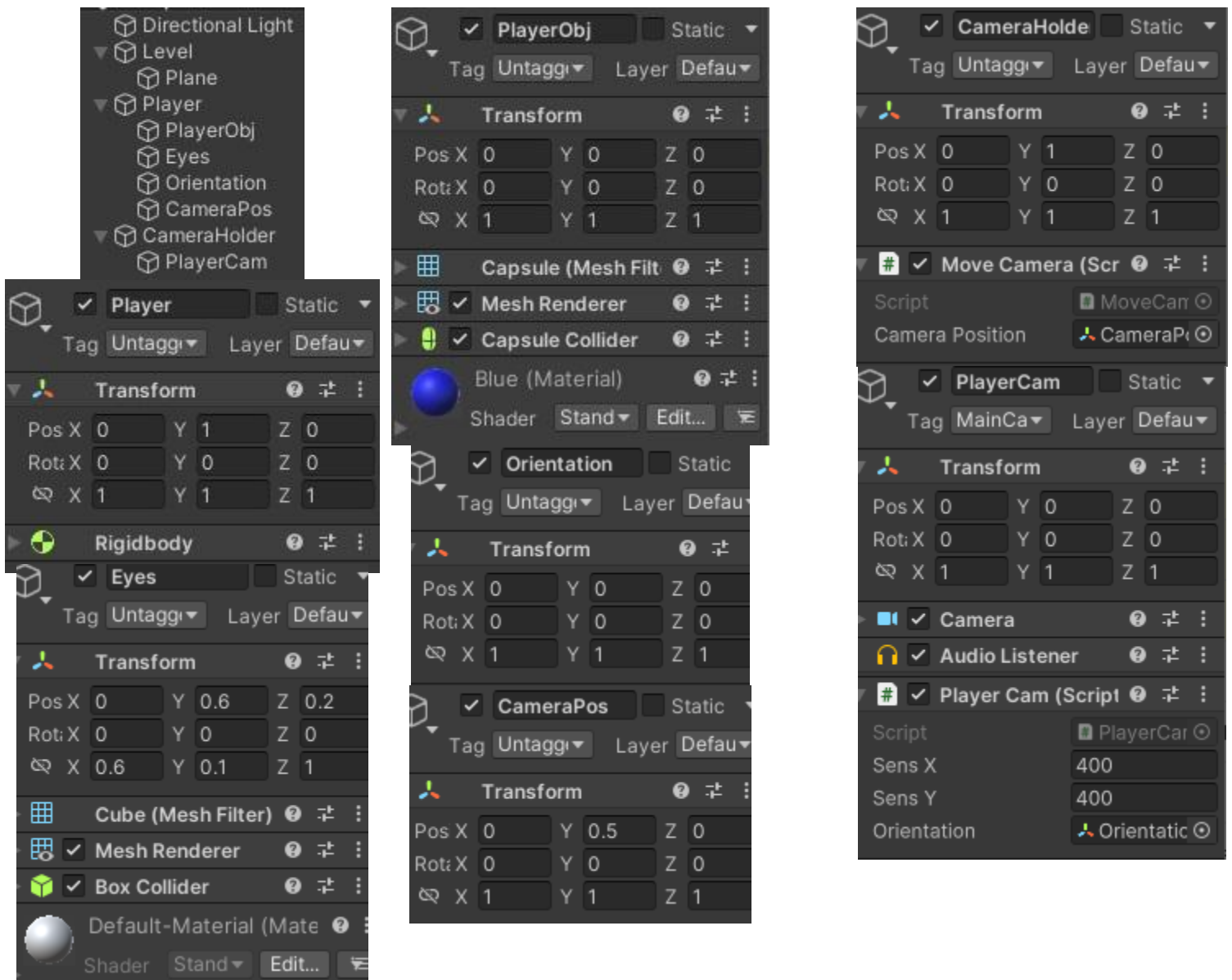
☐ Create Empty named Level >

☐ 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=KI9Zczq1Wh0kLeX&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

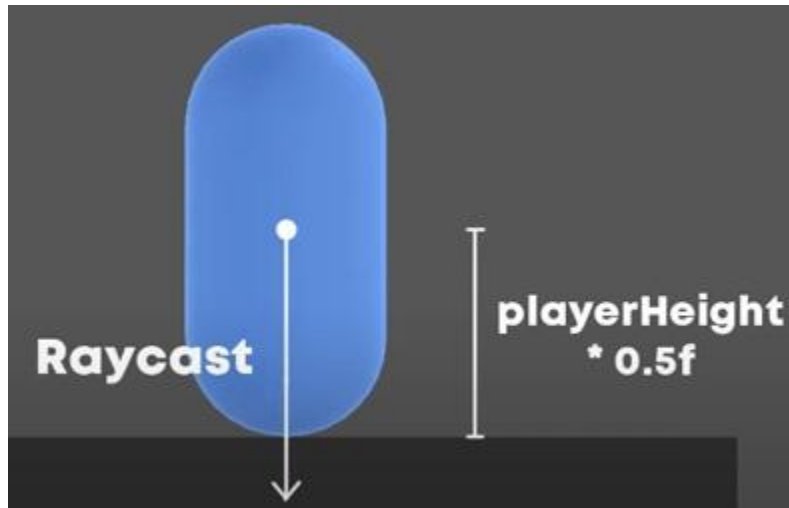
<https://youtu.be/f473C43s8nE?si=Ww2eetEaE8hYKGk8&t=305>

Apply drag to the player's rigidbody, 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 TPro;

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 on slope 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

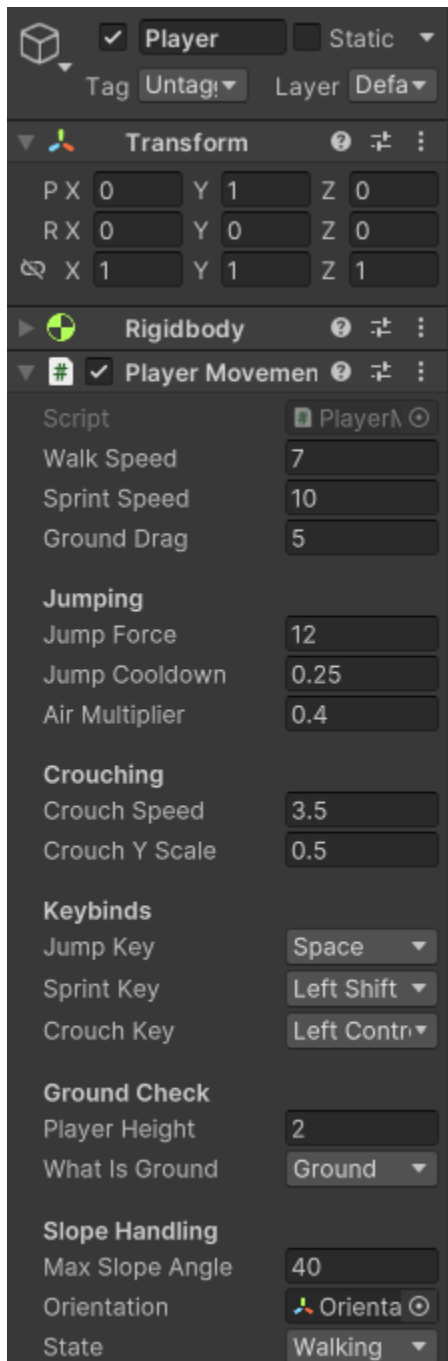
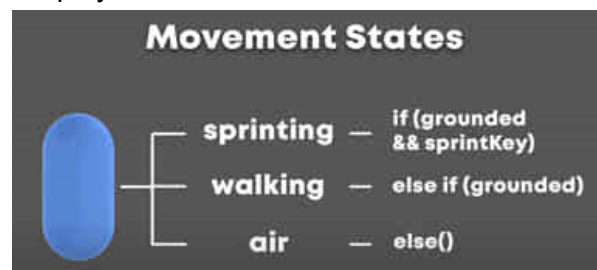
PlayerMovementAdvanced.cs in Player [GameObject]

To code the sprinting ability

<https://youtu.be/xCxSjgYTw9c?si=7lqvPFZ6LCN2tr71&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/xCxSjgYT9c?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()

```

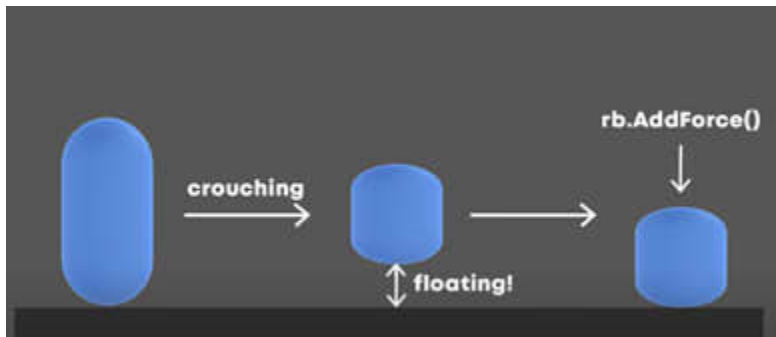
```

{
    // 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
    }
}

```

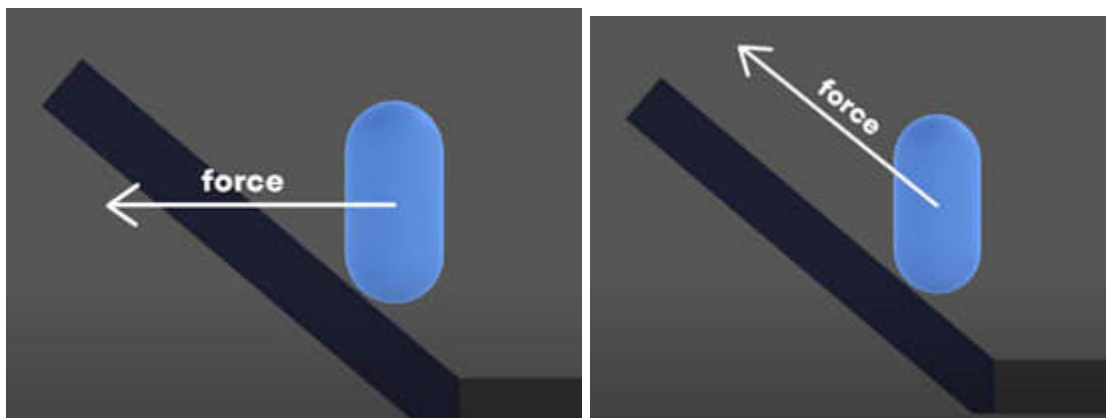
```
// Mode - Sprinting  
else if(grounded && Input.GetKey(sprintKey))  
{
```

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

Slope Movement

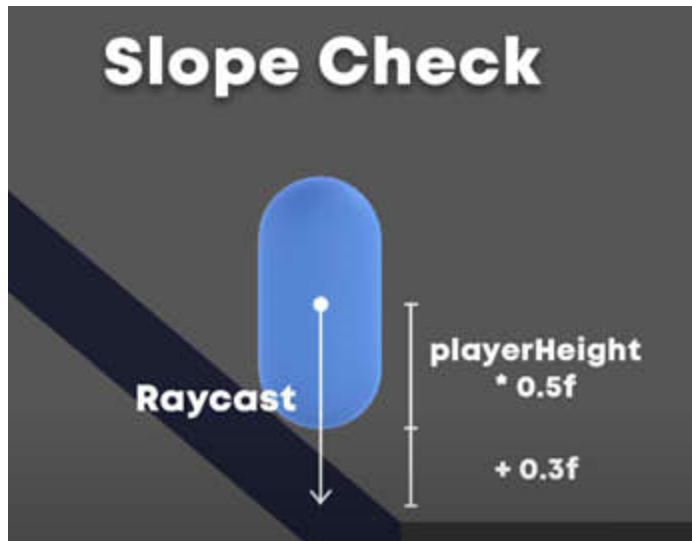
<https://youtu.be/xCxSjgYTw9c?si=8Tn5m0T5gBZo9kSa&t=225>

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/xCxSigYTw9c?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;    }
```

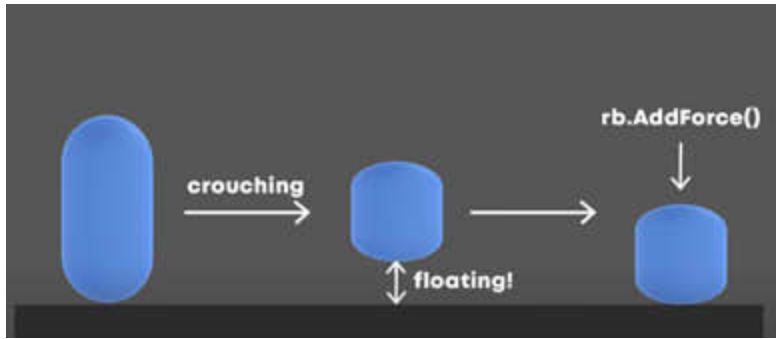
```
private void Update()
{
    horizontalInput = Input.GetAxisRaw("Horizontal");
    verticalInput = Input.GetAxisRaw("Vertical");

    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=dL4A24lQl61oMEMK&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_hbl&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
```

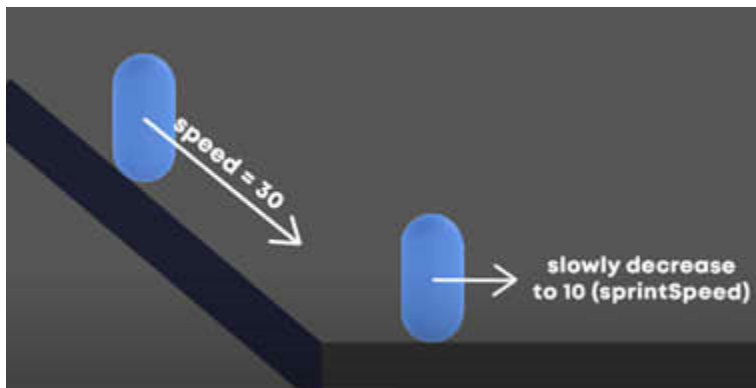
Change moveSpeed variables in PlayerMovementAdvanced.cs to desiredMovement.
The reason for change is that we're now implementing momentum 冲力 into our game.

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=KMFfsj4jHZNQycpA&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>

to learn how to code wall running that feels great,
can be controlled in multiple directions works with or without gravity
and even on curved walls well search no more in this tutorial

need two layer masks

to define what is ground and what is wall

```
[Header("Wallrunning")]  
  
public LayerMask whatIsWall;  
public LayerMask whatIsGround;  
  
public float wallRunForce;  
public float wallClimbSpeed;  
public float maxWallRunTime;  
private float wallRunTimer;
```

for the keyboard inputs all you need is

floats for the horizontal and vertical axis

```
[Header("Input")]  
  
public KeyCode upwardsRunKey = KeyCode.LeftShift;  
public KeyCode downwardsRunKey = KeyCode.LeftControl;  
private bool upwardsRunning;  
private bool downwardsRunning;  
private float horizontalInput;  
private float verticalInput;
```

for the wall detection

create floats for the wall check distance

minimal jump height

as well as raycast hit variables

and booleans for the left and right side raycasts

```
[Header("Detection")]  
  
public float wallCheckDistance;  
public float minJumpHeight;  
private RaycastHit leftWallhit;  
private RaycastHit rightWallhit;  
private bool wallLeft;  
private bool wallRight;
```

to get a reference to the rigid body
and orientation of the player
as well as one to your player movement script

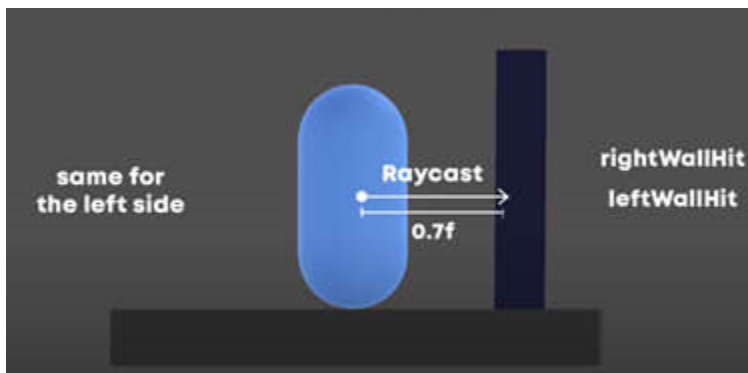
```
[Header("References")]  
public Transform orientation;  
private PlayerMovementAdvanced pm;  
private Rigidbody rb;
```

in void start

you can assign these references using the get component function

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

<https://youtu.be/gNt9wBOrQO4?si=c6Ju-lkzszAYUSg9&t=91>
now before the player should start any wall running movement
you of course need **to check if there's even a wall in range**
we'll do this by shooting out raycasts
to the left and right side of the player
the distance is going to be the wall check distance
and we're going to store the information of the raycast
in the variables we created



create a function called CheckForWall()
and use physics.raycast like this to perform the wall check

```
Physics.Raycast(Start point, Direction, out rightWallhit, Distance,  
whatIsWall);
```

```
private void CheckForWall()  
{
```



```

        wallRight = Physics.Raycast(transform.position, orientation.right,
out rightWallhit, wallCheckDistance, whatIsWall);
        wallLeft = Physics.Raycast(transform.position, -orientation.right,
out leftWallhit, wallCheckDistance, whatIsWall);
    }

```

notice that this part stores the information of the object we hit for later

```

out rightWallhit
out RaycastHit hitInfo

```

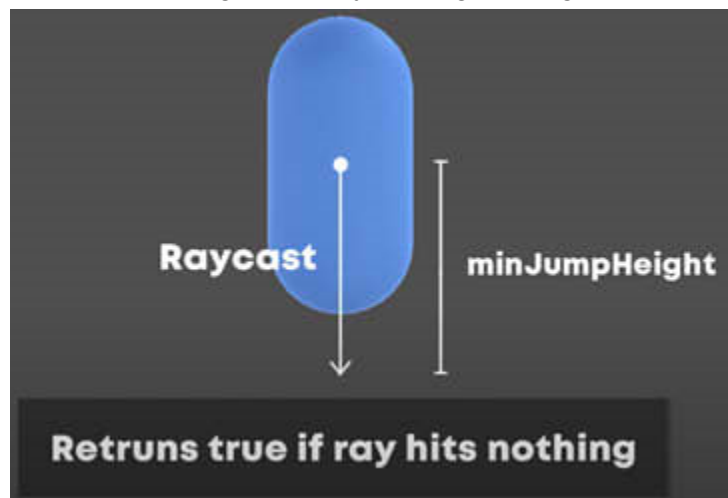
also don't forget to call this CheckForWall() function
in void update

```

private void Update()
{
    CheckForWall();
    StateMachine();
}

```

there's one more check we need to perform
which is checking if the player is high enough in the air to start wall running



so create this spool and it's the same as before
but this time the ray goes downwards and
don't forget that this time you want to return true if the ray hits nothing

```

private bool AboveGround()
{

```

```

        return !Physics.Raycast(transform.position, Vector3.down,
minJumpHeight, whatIsGround);
    }

```

next create a function called StateMachine()
 and we'll start off by getting the horizontal and vertical keyboard inputs
 and now we're basically going
to define when the player should enter the wall running state
 the conditions for this are

1. there has to be a wall on the left or right side
2. you need to be pressing the w key
3. and the player has to be above the ground

if this is all true we want to start a wall run

```

private void StateMachine()
{
    // Getting Inputs
    horizontalInput = Input.GetAxisRaw("Horizontal");
    verticalInput = Input.GetAxisRaw("Vertical");

    upwardsRunning = Input.GetKey(upwardsRunKey);
    downwardsRunning = Input.GetKey(downwardsRunKey);

    // State 1 - Wallrunning
    if((wallLeft || wallRight) && verticalInput > 0 && AboveGround())
    {
        if (!pm.wallrunning)
            StartWallRun();
    }

    // State 3 - None
    else
    {
        if (pm.wallrunning)
            StopWallRun();
    }
}

```

but for this we're going to need a few more functions
 so create a StartWallRun() function

a StopWallRun() function and
also a function for the WallRunningMovement()

```
private void StartWallRun() {}  
private void WallRunningMovement() {}  
private void StopWallRun() {}
```

<https://youtu.be/gNt9wBOrQO4?si=2vCOxAprqEKuNr4B&t=196>

now before we continue you have to understand that

i usually handle all the speed limitations in the player movement script
and then add forces in separate scripts



so quickly open the movement script

And add a float for the wall run speed

```
[Header("Movement")]  
private float moveSpeed;  
public float groundDrag;  
public float walkSpeed;  
public float sprintSpeed;  
  
public float slideSpeed;  
private float desiredMoveSpeed;  
private float lastDesiredMoveSpeed;  
public float speedIncreaseMultiplier;  
public float slopeIncreaseMultiplier;  
  
public float wallrunSpeed; // new
```

A state called wall running

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

```
        air
    }
```

and a bool with the same name

```
public bool crouching;
public bool sliding;
public bool wallrunning; // new
```

now in the state handler you can easily
add this wall running state

```
private void StateHandler()
{
    // Mode - Wallrunning
    if (wallrunning)
    {
        state = MovementState.wallrunning;
        desiredMoveSpeed = wallrunSpeed;
    }
    // TL;DR
```

Now that the speed limit is set up

in StartWallRun()

you can just set the wall running bool of the player movement script to true

```
private void StartWallRun()
{
    pm.wallrunning = true;
}
```

and now let's call the wall running movement

<https://youtu.be/gNt9wBOrQO4?si=H-STyx4OcAC9BSKb&t=242>

The hardest part here is
to find the forward direction of the wall
Because this has to work no matter how your wall is rotated

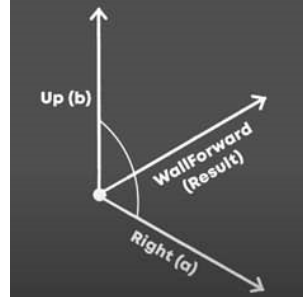
For this we're going to use `vector3.cross`
a function that takes in the right and upwards
direction
and then returns the forward direction

now the right direction is also called the `WallNormal`

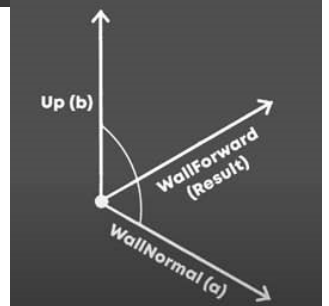
it's just a direction pointing away from the wall

and this one is easy to get
because we already stored the raycast hit information

Vector3.Cross(a, b)



Vector3.Cross(a, b)



https://youtu.be/gNt9wBOrQO4?si=L7DSxZsomJ2exU_z&t=273

```
private void WallRunningMovement()
{
    rb.useGravity = false;
    rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

    Vector3 wallNormal = wallRight ? rightWallhit.normal :
leftWallhit.normal;

    Vector3 wallForward = Vector3.Cross(wallNormal, transform.up);

    if ((orientation.forward - wallForward).magnitude >
(orientation.forward - -wallForward).magnitude)
        wallForward = -wallForward;

    // forward force
    rb.AddForce(wallForward * wallRunForce, ForceMode.Force);

    // upwards/downwards force
```

```

        if (upwardsRunning)
            rb.velocity = new Vector3(rb.velocity.x, wallClimbSpeed,
rb.velocity.z);
        if (downwardsRunning)
            rb.velocity = new Vector3(rb.velocity.x, -wallClimbSpeed,
rb.velocity.z);

        // push to wall force
        if (!(wallLeft && horizontalInput > 0) && !(wallRight &&
horizontalInput < 0))
            rb.AddForce(-wallNormal * 100, ForceMode.Force);
    }

```

and for now we're just gonna turn the gravity off
and set the rigidbody's y velocity to zero

so just add the new vector3 called **wallNormal**
and **if the wall is on the right** we want to use the **rightWallHit.normal**
otherwise the **leftWallHit.normal**

and as explained for the **wallForward** we're just gonna use the **cross product** of the
wallNormal and the upwards direction which is **transform.up**
now you can add force with **rigidbody.addforce** in the **wall forward direction**
Multiplied with your **walRunForce** and using **ForceMode.Force**

<https://youtu.be/gNt9wBOrQO4?si=vqLjVHCO7ITywnb&t=316>
also in stopwall run you want
to set the wall running bool to false again

```

private void StopWallRun()
{
    pm.wallrunning = false;
}

```

now obviously you need to call these functions somewhere
so go back to your state machine
and in the wall running state
Call StartWallRun()
and if you're not in this state
Call StopWallRun()

```

private void StateMachine()

```

```

{
// TD;DR
    // State 1 - Wallrunning
    if((wallLeft || wallRight) && verticalInput > 0 && AboveGround())
    {
        if (!pm.wallrunning)
            StartWallRun();
    }

    // State 3 - None
    else
    {
        if (pm.wallrunning)
            StopWallRun();
    }
}

```

and now you can open FixedUpdate()
And while you're wall running
you can call the wall running movement function

```

private void FixedUpdate()
{
    if (pm.wallrunning)
        WallRunningMovement();
}

```

also don't forget to call the StateMachine() in void Update

```

private void Update()
{
    CheckForWall();
    StateMachine();
}

```

<https://youtu.be/gNt9wBOrQO4?si=fI04WOj9xwK2qxyd&t=344>

now switch to unity assign a wall run
speed add the wall running script and
also set the rest of the variables and
to keep it simple i'm just using whatisground for both layers
just make sure that your walls have the what is ground layer selected

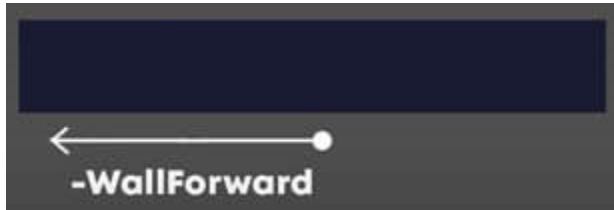
<https://youtu.be/gNt9wBOrQO4?si=ji6mKyD2k1r4-evr&t=365>

if you're now at play you can see the wall run is already working
but if you try from the other side i'm wall running backwards

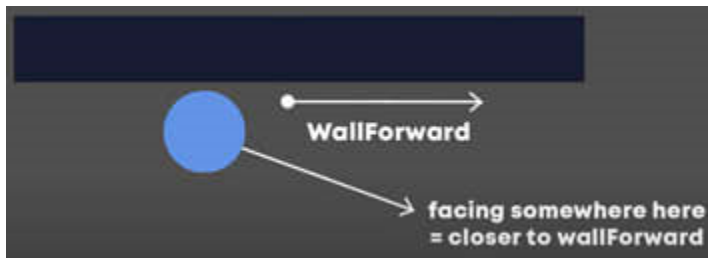
finding the forward direction is great



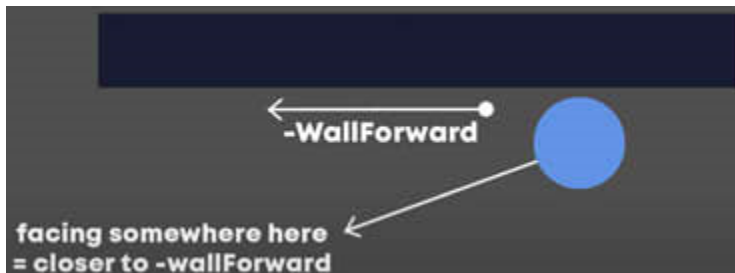
<https://youtu.be/gNt9wBOrQO4?si=Y0uJGUHgw2CtD4&t=370>
but sometimes we want to use the exact opposite



so basically if the player is facing somewhere here
we want to use the forward direction



whereas when the player is facing here
we want to use the backwards or minus forward direction



So backwards or minus forward direction

So add these two lines

to find out which direction is closer to where the player is facing

```
private void WallRunningMovement()
```



```

    {
// TL;DR
        if ((orientation.forward - wallForward).magnitude >
(orientation.forward - -wallForward).magnitude)
            wallForward = -wallForward;
// TL;DR
    }

```

and there you go you can now wall run in both directions
and this even works on curved walls

however if you try to wall run on the
outside of a curved wall you lose contact

so quickly head back to your script and inside of the wall running movement function
you want to push your player towards the wall
by using `rigidbody.addforce` the opposite of the `wallNormal` times 100 and using
`ForceMode.Force`

```

private void WallRunningMovement()
{
// TL;DR
    // push to wall force
    if (!(wallLeft && horizontalInput > 0) && !(wallRight &&
horizontalInput < 0))
        rb.AddForce(-wallNormal * 100, ForceMode.Force);
}

```

and you only want to add this force
If the player is not currently trying to get away from the wall
which would mean there's a wall on the left
and he's pressing d or there's a wall on the right and he's pressing a

<https://youtu.be/gNt9wBOrQO4?si=FXo1bguUUW-e3ET2&t=444>

how to create diagonal wall running
which is really important because it gives the player a lot more control
so there's different ways how to code this
but what i recommend is to create key codes for upwards and downwards while running
in my case that would be shift and control
and then you also need two bools with similar names

```

[Header("Input")]
public KeyCode upwardsRunKey = KeyCode.LeftShift;
public KeyCode downwardsRunKey = KeyCode.LeftControl;
private bool upwardsRunning;

```

```
private bool downwardsRunning;
```

and a float for the wall climb speed

```
[Header("Wallrunning")]  
public float wallClimbSpeed;
```

in your state machine you can now assign the inputs like this

```
private void StateMachine()  
{  
// TL;DR  
    upwardsRunning = Input.GetKey(upwardsRunKey);  
    downwardsRunning = Input.GetKey(downwardsRunKey);  
// TL;DR  
    // State 1 - Wallrunning  
}
```

and in the wall running movement function when you want to run upwards
just set the y velocity of your rigidbody to your wall climb speed
and the opposite applies to when you want to run downwards

```
private void WallRunningMovement()  
{  
// TL;DR  
    // upwards/downwards force  
    if (upwardsRunning)  
        rb.velocity = new Vector3(rb.velocity.x, wallClimbSpeed,  
rb.velocity.z);  
    if (downwardsRunning)  
        rb.velocity = new Vector3(rb.velocity.x, -wallClimbSpeed,  
rb.velocity.z);  
    // push to wall force  
// TL;DR  
}
```

now set the wall climb speed to something like 3 and there you go

Value

Player

Static

Tag Player Layer Default

Transform

Rigidbody

Player Movement Advance

Script

PlayerMovement

Ground Drag

4

Walk Speed

7

Sprint Speed

10

Slide Speed

30

Speed Increase Multi

1.5

Slope Increase Multi

2.5

Wallrun Speed

8.5

Jumping

Jump Force

12

Jump Cooldown

0.25

Air Multiplier

0.4

Crouching

Crouch Speed

3.5

Crouch Y Scale

0.5

Keybinds

Jump Key

Space

Sprint Key

Left Shift

Crouch Key

C

Ground Check

Player Height

2

What Is Ground

Ground

Slope Handling

Max Slope Angle

80

Orientation

Orientation (Tran

State

Walking

Crouching

Sliding

Wallrunning

Ground Check Distan

0.7

Num Raycasts

5

Slope Threshold

80

Sliding (Script)

Wall Running (Script)

Player

Static

Tag Player Layer Default

Transform

Rigidbody

Player Movement Adv

Sliding (Script)

Wall Running (Script)

Script

WallRunning

Wallrunning

What Is Wall

Ground

What Is Ground

Ground

Wall Run Force

200

Wall Climb Speed

3

Max Wall Run Time

1.5

Input

Upwards Run Key

Left Shift

Downwards Run Key

Left Control

Detection

Wall Check Distance

0.7

Min Jump Height

2

References

Orientation

Orientation (T

Code

WallRunning.cs

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

public class WallRunning : MonoBehaviour
{
    [Header("Wallrunning")]
    public LayerMask whatIsWall;
    public LayerMask whatIsGround;
    public float wallRunForce;
    public float wallClimbSpeed;
    public float maxWallRunTime;
    private float wallRunTimer;

    [Header("Input")]
    public KeyCode upwardsRunKey = KeyCode.LeftShift;
    public KeyCode downwardsRunKey = KeyCode.LeftControl;
    private bool upwardsRunning;
    private bool downwardsRunning;
    private float horizontalInput;
    private float verticalInput;

    [Header("Detection")]
    public float wallCheckDistance;
    public float minJumpHeight;
    private RaycastHit leftWallhit;
    private RaycastHit rightWallhit;
    private bool wallLeft;
    private bool wallRight;

    [Header("References")]
    public Transform orientation;
    private PlayerMovementAdvanced pm;
    private Rigidbody rb;

    private void Start()
```

```

{
    rb = GetComponent<Rigidbody>();
    pm = GetComponent<PlayerMovementAdvanced>();
}

private void Update()
{
    CheckForWall();
    StateMachine();
}

private void FixedUpdate()
{
    if (pm.wallrunning)
        WallRunningMovement();
}

private void CheckForWall()
{
    wallRight = Physics.Raycast(transform.position, orientation.right,
out rightWallhit, wallCheckDistance, whatIsWall);
    wallLeft = Physics.Raycast(transform.position, -orientation.right,
out leftWallhit, wallCheckDistance, whatIsWall);
}

private bool AboveGround()
{
    return !Physics.Raycast(transform.position, Vector3.down,
minJumpHeight, whatIsGround);
}

private void StateMachine()
{
    // Getting Inputs
    horizontalInput = Input.GetAxisRaw("Horizontal");
    verticalInput = Input.GetAxisRaw("Vertical");

    upwardsRunning = Input.GetKey(upwardsRunKey);
    downwardsRunning = Input.GetKey(downwardsRunKey);
}

```

```

        // State 1 - Wallrunning
        if((wallLeft || wallRight) && verticalInput > 0 && AboveGround())
        {
            if (!pm.wallrunning)
                StartWallRun();
        }

        // State 3 - None
        else
        {
            if (pm.wallrunning)
                StopWallRun();
        }
    }

    private void StartWallRun()
    {
        pm.wallrunning = true;
    }

    private void WallRunningMovement()
    {
        rb.useGravity = false;
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);

        Vector3 wallNormal = wallRight ? rightWallhit.normal :
leftWallhit.normal;

        Vector3 wallForward = Vector3.Cross(wallNormal, transform.up);

        if ((orientation.forward - wallForward).magnitude >
(orientation.forward - -wallForward).magnitude)
            wallForward = -wallForward;

        // forward force
        rb.AddForce(wallForward * wallRunForce, ForceMode.Force);

        // upwards/downwards force
        if (upwardsRunning)

```

```

        rb.velocity = new Vector3(rb.velocity.x, wallClimbSpeed,
rb.velocity.z);
        if (downwardsRunning)
            rb.velocity = new Vector3(rb.velocity.x, -wallClimbSpeed,
rb.velocity.z);

        // push to wall force
        if (!(wallLeft && horizontalInput > 0) && !(wallRight &&
horizontalInput < 0))
            rb.AddForce(-wallNormal * 100, ForceMode.Force);
    }

    private void StopWallRun()
    {
        pm.wallrunning = false;
    }
}

```

PlayerMovementAdvanced.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TPro;

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

    public float slideSpeed;
    private float desiredMoveSpeed;
    private float lastDesiredMoveSpeed;
    public float speedIncreaseMultiplier;
    public float slopeIncreaseMultiplier;

    public float wallrunSpeed; // new

```

```

[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,
        wallrunning, // new
        crouching,
        sliding,
        air
    }

    public bool crouching;
    public bool sliding;
    public bool wallrunning; // new

    // Parameters for ground check
    public float groundCheckDistance = 0.5f; // The distance to check for
ground
    public int numRaycasts = 5; // Number of raycasts to cast
    public float slopeThreshold = 30f; // The maximum slope angle that is
considered as walkable

    // Ground check variables
    //bool grounded;
    bool onSteepGround;

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

        readyToJump = true;

        startYScale = transform.localScale.y;
    }

    private void Update()
    {
        // Perform ground check
        GroundCheck();
        // 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.GetKeyUp(jumpKey) && !grounded) {
        Debug.Log("Jumped, but not ground");
        Debug.Log("Jumped, onSteepGround is " + onSteepGround);
    }
    else if (Input.GetKey(jumpKey) && readyToJump && grounded)
    {
        readyToJump = false;
        //SDebug.Log("Jumped, on ground");
        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 - Wallrunning
    if (wallrunning)
    {
        state = MovementState.wallrunning;
        desiredMoveSpeed = wallrunSpeed;
    }

    // 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;
    }

    private void GroundCheck2()
    {
        grounded = false;
        onSteepGround = false;

        // Cast multiple rays downward from the player's position
        for (int i = 0; i < numRaycasts; i++)
        {
            float angle = i * (360f / numRaycasts); // Calculate angle for
raycast direction
            Vector3 direction = Quaternion.AngleAxis(angle, transform.up)
* -transform.forward; // Calculate raycast direction

            //groundCheckDistance = playerHeight * 0.5f + 0.2f;

            RaycastHit hit;
            if (Physics.Raycast(transform.position, Vector3.down, out hit,
playerHeight * 0.5f + 0.2f, whatIsGround))
                //if (Physics.Raycast(transform.position, direction, out hit,
groundCheckDistance, whatIsGround))
            {
                grounded = true;

                // Check if the slope angle exceeds the threshold
                float slopeAngle = Vector3.Angle(hit.normal, Vector3.up);
                if (slopeAngle > slopeThreshold)
                {
                    onSteepGround = true;
                    break; // Exit loop if a steep slope is detected
                }
            }
        }
    }

    private void GroundCheck() // work
    {

```



```
        // Cast a single raycast straight down from the player's position
        RaycastHit hit;
        if (Physics.Raycast(transform.position, Vector3.down, out hit,
playerHeight * 0.5f + 0.2f, whatIsGround))
        {
            grounded = true;

            // Check if the slope angle exceeds the threshold
            float slopeAngle = Vector3.Angle(hit.normal, Vector3.up);
            if (slopeAngle > maxSlopeAngle)
            {
                onSteepGround = true;
            }
            else
            {
                onSteepGround = false;
            }
        }
        else
        {
            grounded = false;
            onSteepGround = false;
        }
    }
}
```

WALL JUMPING & CAMERA EFFECTS - Unity Tutorial

<https://www.youtube.com/watch?v=WfW0k5qENxM>

add wall jumping and a few other things while running on wall

add wall jumping while running on wall

open up your wall running script

and add two floats for the wall jump up force

and wall jump side force

```
[Header("Wallrunning")]
public LayerMask whatIsWall;
public LayerMask whatIsGround;
public float wallRunForce;
public float wallJumpUpForce;
public float wallJumpSideForce;
public float wallClimbSpeed;
public float maxWallRunTime;
private float wallRunTimer;
```

then you can create a new function called WallJump()

<https://youtu.be/WfW0k5qENxM?si=Z8Ezd5soBNhZgAsW&t=21>

```
private void WallJump()
{
    // enter exiting wall state
    exitingWall = true;
    exitWallTimer = exitWallTime;
// save the wallNormal depending on whether the wall is on the left or
right side
    Vector3 wallNormal = wallRight ? rightWallhit.normal :
leftWallhit.normal;
// calculate the force to apply
// by multiplying the upwards direction with the wallJumpForce
// and adding the wallNormal multiplied with the wallJumpSideForce
    Vector3 forceToApply = transform.up * wallJumpUpForce + wallNormal
* wallJumpSideForce;
// reset y velocity before add force
```

```

        // reset y velocity and add force
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);
// apply force
        rb.AddForce(forceToApply, ForceMode.Impulse);
    }

```

as with any jump i would recommend you
to reset the y velocity of the player before you add the force

this way the jumping will feel clean even if you're currently falling

also you need to call this function somewhere

so define a jump key

```

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

```

and while you're in the wall running state just call the wall jump function if you press this key

```

private void StateMachine()
{
// TL;DR
    // State 1 - Wallrunning
    if((wallLeft || wallRight) && verticalInput > 0 && AboveGround()
&& !exitingWall)
    {
// TL;DR
        // wall jump
        if (Input.GetKeyDown(jumpKey)) WallJump(); // Here
    }
}

```

okay now you can switch back to unity
set the values of the variables you created and hit play
<https://youtu.be/WfW0k5qENxM?si=XVWYHzuLPsIC5U1x&t=80>

Problem:

Can't exist from the wall while wall jumping

the reason for this is that

WallRun -1Frame- **WallJump -1Frame-** WallRun -1Frame-

currently you can do a **wall jump**

but then one frame later

if you're still close enough to a wall

You'll automatically enter the **wall running state** again

<https://youtu.be/WfW0k5qENxM?si=wt6THvpmWhxOG49q&t=102>

To fix this let's create an exiting wall state for this

you're going to need a bool called exiting

wall as well as 2 floats for the exit wall time and exit wall timer

```
[Header("Exiting")]
private bool exitingWall;
public float exitWallTime;
private float exitWallTimer;
```

now just as you created the other states

open the state machine add an elseif statement

and the condition is going to be the bool you just created

```
private void StateMachine()
{
    // Getting Inputs
// TL;DR
    // State 1 - Wallrunning
// Should not be able to start a wallrun while you're trying to exit the
wall
// will bool check !exitingWall
    if((wallLeft || wallRight) && verticalInput > 0 && AboveGround()
&& !exitingWall)
    {
        if (!pm.wallrunning){}
        // wallrun timer
        if (wallRunTimer > 0){}
        if(wallRunTimer <= 0 && pm.wallrunning){}

        // wall jump
        if (Input.GetKeyDown(jumpKey)) WallJump();
    }

    // State 2 - Exiting // Start from Here
    else if (exitingWall) //
    {
        if (pm.wallrunning)
            StopWallRun(); // to cancel any active wallrun
    }
}
```

```

// only want to be in this state for short amount of time
// countdown exitWallTimer until it reach zero
    if (exitWallTimer > 0)
        exitWallTimer -= Time.deltaTime;
// as soon as it reach zero, set exitingWall to false
    if (exitWallTimer <= 0)
        exitingWall = false;
}

// State 3 - None
else
{
    if (pm.wallrunning)
        StopWallRun();
}
}

```

ok and now you can go back to your WallJump function
and exit the wall by setting exiting wall to true and exit wall timer to exit wall time

```

private void WallJump()
{
    // enter exiting wall state
    exitingWall = true; // Here
    exitWallTimer = exitWallTime; // Here
// TL;DR
}

```

<https://youtu.be/WfW0k5qENxM?si=ZAAkcsTIEYcFdzy&t=161>

now back in unity set the exit wall time

2:45

to something like this 0.2 and there you go

Improve your wall run ability

<https://youtu.be/WfW0k5qENxM?si=wPQ4HZ9TlpKwbLK4&t=178>

some more things to improve your wall run ability

the first one would be **limiting your wall run time**

because right now it's a bit **weird** since you **can just wall run forever**

so let's implement that go back to the wall running script

And add floats for the max wall run time

And the wall run timer

```
[Header("Wallrunning")]
public LayerMask whatIsWall;
public LayerMask whatIsGround;
public float wallRunForce;
public float wallJumpUpForce;
public float wallJumpSideForce;
public float wallClimbSpeed;
public float maxWallRunTime; // Here
private float wallRunTimer; // Here
```

now when you start the wall run

just set the wall run timer to the maximum

```
private void StartWallRun()
{
    pm.wallrunning = true;

    wallRunTimer = maxWallRunTime; // Here
}
```

and while you're in the wall run state

make sure that the timer is counting down

```
private void StateMachine()
{
    // Getting Inputs
// TL;DR
    // State 1 - Wallrunning
    if((wallLeft || wallRight) && verticalInput > 0 && AboveGround()
&& !exitingWall)
    {
        if (!pm.wallrunning)
            StartWallRun();
// while you're in the wall run state
// make sure that the timer is counting down
        // wallrun timer
        if (wallRunTimer > 0)
            wallRunTimer -= Time.deltaTime;
// whenever the timer reaches zero and you're currently wall running,
// you want to set exitingWall to true,
```

```
// and start the exitWallTimer
    if(wallRunTimer <= 0 && pm.wallrunning)
    {
        exitingWall = true;
        exitWallTimer = exitWallTime;
    }
// TL;DR
}
```

<https://youtu.be/WfW0k5qENxM?si=uYX03Z2QSFqWcaO&t=218>
 now if you go back to unity set the variables to something like this
 MaxWallRunTime 0.7
 and hit play you can see that after a short time the wall run stops

how to use gravity when wall running

https://youtu.be/WfW0k5qENxM?si=0djzvFhxUJ_BDboy&t=227
 so let's implement that
 create a bool called useGravity and a float called gravity counterforce

```
[Header("Gravity")]
public bool useGravity;
public float gravityCounterForce;
```

4:02

now whenever you set the rigidbody's gravity
 don't just set it to false and instead
 set it to use gravity

```
private void WallRunningMovement()
{
    rb.useGravity = useGravity; // Here
```

this way you can easily control in the inspector
 whether or not you want to use it

also move this line up into the StartWallRun function

```
private void StartWallRun()
{
    pm.wallrunning = true;

    wallRunTimer = maxWallRunTime;
```

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

because we don't want this to be called every frame

and also go back to your player movement script and make sure that
this line only gets executed
if you're not currently while running

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

<https://youtu.be/WfW0k5qENxM?si=JUdHPrl5ckxhWu0&t=268>

if you turn it on and hit play it does work but it's a bit too strong

so to weaken the gravity a bit

go back to your wall running movement function
and if you are using gravity
apply a bit of counter force

```
private void WallRunningMovement()  
{ // TL;DR  
    // push to wall force  
    if (!(wallLeft && horizontalInput > 0) && !(wallRight &&  
horizontalInput < 0))  
        rb.AddForce(-wallNormal * 100, ForceMode.Force);  
  
    // weaken gravity  
    if (useGravity) // Here  
        rb.AddForce(transform.up * gravityCounterForce,  
ForceMode.Force);  
}
```

<https://youtu.be/WfW0k5qENxM?si=g4aDpulONh8mSt84&t=286>

now the higher you set the counterforce
the lower the effect of gravity will be
just don't set it too high
unless you want to create a spaceship simulation

Fov changes and Camera Tilt

https://youtu.be/WfW0k5qENxM?si=-GJWYvTxZ_nDVoPK&t=297

Open up your PlayerCam script

whichever one you're using and

Create functions for the Fov changes and Tilting

also both functions should take in a float for the end value

```
public void DoFov(float endValue) {}  
public void DoTilt(float zTilt) {}
```

Fade in and out Effect

<https://youtu.be/WfW0k5qENxM?si=ZH8hFE5G13zAR2cf&t=313>

use a great asset called DOTween

So import it to unity go through the setup steps

and then you can add using dg.tweening to your Cam script

```
using DG.Tweening;
```

now thanks to this asset the rest is super easy

just get the component of the camera

and use the DOFieldOfView function passing in the endValue and 0.25 for the transition time

and for the tilting you can use DOLocalRotate and then just pass in detailed on the z axis and again a transition time of 0.25

```
public void DoFov(float endValue)  
{  
    GetComponent<Camera>().DOFieldOfView(endValue, 0.25f);  
}  
  
public void DoTilt(float zTilt)  
{  
    transform.DOLocalRotate(new Vector3(0, 0, zTilt), 0.25f);  
}
```

The **DoFov** method animates the camera's field of view change using DOTween's **DOFieldOfView** function. It smoothly transitions the camera's FOV to the specified **endValue** over a duration of 0.25 seconds.

The **DoTilt** method animates an object's tilt around its z-axis using DOTween's **DOLocalRotate** function. It smoothly rotates the object to the specified **zTilt** value around its local z-axis over a duration of 0.25 seconds.

and now one important change that you have to do is

to create a transform for the Cam holder

```
public Transform camHolder;
```

and then here you want to rotate the camHolder instead of the camera

```
private void Update()
{ // TL;DR
    // Rotate cam and orientation
    camHolder.rotation = Quaternion.Euler(xRotation, yRotation, 0);
```

otherwise you're rotating the camera from two points in the script and it would be overriding itself

<https://youtu.be/WfW0k5qENxM?si=1v4fKKvvpOzK29md&t=372>

and back in your WallRunning script

you can now get a reference to your camera script

```
[Header("References")]
public Transform orientation;
public PlayerCam cam; // Here
private PlayerMovementAdvanced pm;
private Rigidbody rb;
```

and when you start the wall run

you can use the functions you just created to set the fov and tilt to something like this

```
private void StartWallRun()
{
    pm.wallrunning = true;

    wallRunTimer = maxWallRunTime;

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

    // apply camera effects
    cam.DoFov(90f); // Here
    if (wallLeft) cam.DoTilt(-5f); // Here
    if (wallRight) cam.DoTilt(5f); // Here
}
```

and don't forget to set them back to normal when you stop the wall run

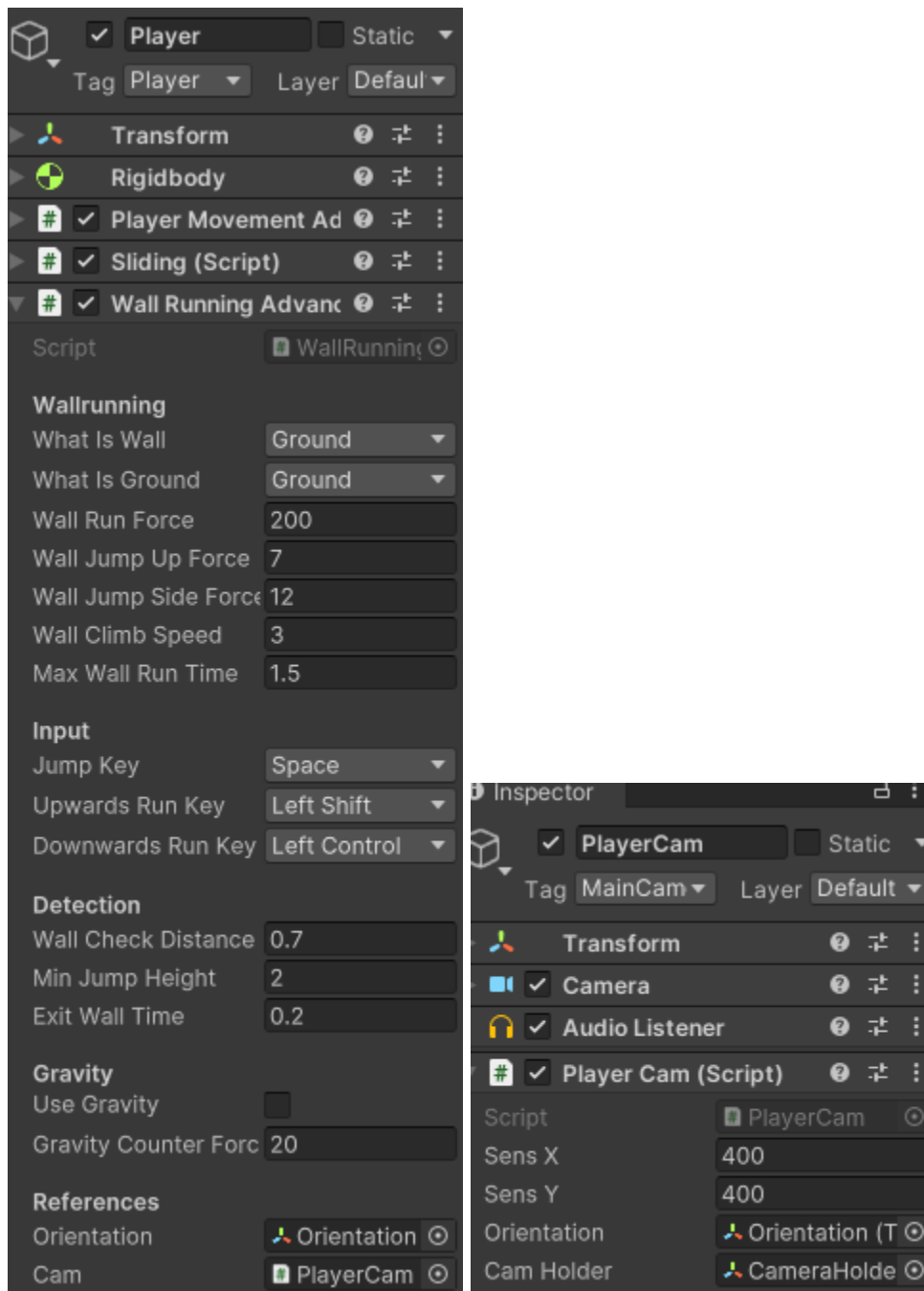
```
private void StopWallRun()
{
    pm.wallrunning = false;
```

```
// reset camera effects  
cam.DoFov(80f); // Here  
cam.DoTilt(0f); // Here  
}
```

https://youtu.be/WfW0k5qENxM?si=_-ko0p4-8ffV4wi4&t=405

now assign the cam holder to the camera script
and the camera to the wall running script

Value



Code

WallRunningAdvanced.cs

```
using System.Collections;
```

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

public class WallRunningAdvanced : MonoBehaviour
{
    [Header("Wallrunning")]
    public LayerMask whatIsWall;
    public LayerMask whatIsGround;
    public float wallRunForce;
    public float wallJumpUpForce;
    public float wallJumpSideForce;
    public float wallClimbSpeed;
    public float maxWallRunTime;
    private float wallRunTimer;

    [Header("Input")]
    public KeyCode jumpKey = KeyCode.Space;
    public KeyCode upwardsRunKey = KeyCode.LeftShift;
    public KeyCode downwardsRunKey = KeyCode.LeftControl;
    private bool upwardsRunning;
    private bool downwardsRunning;
    private float horizontalInput;
    private float verticalInput;

    [Header("Detection")]
    public float wallCheckDistance;
    public float minJumpHeight;
    private RaycastHit leftWallhit;
    private RaycastHit rightWallhit;
    private bool wallLeft;
    private bool wallRight;

    [Header("Exiting")]
    private bool exitingWall;
    public float exitWallTime;
    private float exitWallTimer;

    [Header("Gravity")]
    public bool useGravity;
    public float gravityCounterForce;
```

```

[Header("References")]
public Transform orientation;
public PlayerCam cam;
private PlayerMovementAdvanced pm;
private Rigidbody rb;

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

private void Update()
{
    CheckForWall();
    StateMachine();
}

private void FixedUpdate()
{
    if (pm.wallrunning)
        WallRunningMovement();
}

private void CheckForWall()
{
    wallRight = Physics.Raycast(transform.position, orientation.right,
out rightWallhit, wallCheckDistance, whatIsWall);
    wallLeft = Physics.Raycast(transform.position, -orientation.right,
out leftWallhit, wallCheckDistance, whatIsWall);
}

private bool AboveGround()
{
    return !Physics.Raycast(transform.position, Vector3.down,
minJumpHeight, whatIsGround);
}

private void StateMachine()

```

```

{
    // Getting Inputs
    horizontalInput = Input.GetAxisRaw("Horizontal");
    verticalInput = Input.GetAxisRaw("Vertical");

    upwardsRunning = Input.GetKey(upwardsRunKey);
    downwardsRunning = Input.GetKey(downwardsRunKey);

    // State 1 - Wallrunning
    if((wallLeft || wallRight) && verticalInput > 0 && AboveGround()
&& !exitingWall)
    {
        if (!pm.wallrunning)
            StartWallRun();

        // wallrun timer
        if (wallRunTimer > 0)
            wallRunTimer -= Time.deltaTime;

        if(wallRunTimer <= 0 && pm.wallrunning)
        {
            exitingWall = true;
            exitWallTimer = exitWallTime;
        }

        // wall jump
        if (Input.GetKeyDown(jumpKey)) WallJump();
    }

    // State 2 - Exiting
    else if (exitingWall)
    {
        if (pm.wallrunning)
            StopWallRun();

        if (exitWallTimer > 0)
            exitWallTimer -= Time.deltaTime;

        if (exitWallTimer <= 0)
            exitingWall = false;
    }
}

```

```

    }

    // State 3 - None
    else
    {
        if (pm.wallrunning)
            StopWallRun();
    }
}

private void StartWallRun()
{
    pm.wallrunning = true;

    wallRunTimer = maxWallRunTime;

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

    // apply camera effects
    cam.DoFov(90f);
    if (wallLeft) cam.DoTilt(-5f);
    if (wallRight) cam.DoTilt(5f);
}

private void WallRunningMovement()
{
    rb.useGravity = useGravity;

    Vector3 wallNormal = wallRight ? rightWallhit.normal :
leftWallhit.normal;

    Vector3 wallForward = Vector3.Cross(wallNormal, transform.up);

    if ((orientation.forward - wallForward).magnitude >
(orientation.forward - -wallForward).magnitude)
        wallForward = -wallForward;

    // forward force
    rb.AddForce(wallForward * wallRunForce, ForceMode.Force);
}

```



```

        // upwards/downwards force
        if (upwardsRunning)
            rb.velocity = new Vector3(rb.velocity.x, wallClimbSpeed,
rb.velocity.z);
        if (downwardsRunning)
            rb.velocity = new Vector3(rb.velocity.x, -wallClimbSpeed,
rb.velocity.z);

        // push to wall force
        if (!(wallLeft && horizontalInput > 0) && !(wallRight &&
horizontalInput < 0))
            rb.AddForce(-wallNormal * 100, ForceMode.Force);

        // weaken gravity
        if (useGravity)
            rb.AddForce(transform.up * gravityCounterForce,
ForceMode.Force);
    }

    private void StopWallRun()
    {
        pm.wallrunning = false;

        // reset camera effects
        cam.DoFov(80f);
        cam.DoTilt(0f);
    }

    private void WallJump()
    {
        // enter exiting wall state
        exitingWall = true;
        exitWallTimer = exitWallTime;

        Vector3 wallNormal = wallRight ? rightWallhit.normal :
leftWallhit.normal;

        Vector3 forceToApply = transform.up * wallJumpUpForce + wallNormal
* wallJumpSideForce;

```

```

        // reset y velocity and add force
        rb.velocity = new Vector3(rb.velocity.x, 0f, rb.velocity.z);
        rb.AddForce(forceToApply, ForceMode.Impulse);
    }
}

```

PlayerCam.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using DG.Tweening;

public class PlayerCam : MonoBehaviour
{
    public float sensX;
    public float sensY;

    public Transform orientation;
    public Transform camHolder;

    float xRotation;
    float yRotation;

    private void Start()
    {
        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;

        yRotation += mouseX;
    }
}

```

```
xRotation -= mouseY;
xRotation = Mathf.Clamp(xRotation, -90f, 90f);

// rotate cam and orientation
camHolder.rotation = Quaternion.Euler(xRotation, yRotation, 0);
orientation.rotation = Quaternion.Euler(0, yRotation, 0);
}

public void DoFov(float endValue)
{
    GetComponent<Camera>().DOFieldOfView(endValue, 0.25f);
}

public void DoTilt(float zTilt)
{
    transform.DOLocalRotate(new Vector3(0, 0, zTilt), 0.25f);
}
}
```

Full CLIMBING SYSTEM in 10 MINUTES - Unity Tutorial

<https://www.youtube.com/watch?v=tAJLiOEfbQg>

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>

