

Tutorials:

-Player Movement-

With this script I used a person called Dani who used a custom Player Movement script to use physics with the player.

Firstly, you need to create many variables for the script itself and some of them must be public to allow you to edit the code in the inspector in unity than going back to the code to change it.

This Variables include:

- Assignable
- Other (Rigid body component)
- Rotation and Look
- Movement
- Crouch and Slide
- Jumping
- Input (Player Input)
- Sliding (The motion of the slide)

```

//Assingables
public Transform playerCam;
public Transform orientation;

//Other
private Rigidbody rb;

//Rotation and look
private float xRotation;
private float sensitivity = 50f;
private float sensMultiplier = 1f;

//Movement
public float moveSpeed = 4500;
public float maxSpeed = 20;
public bool grounded;
public LayerMask whatIsGround;

public float counterMovement = 0.175f;
private float threshold = 0.01f;
public float maxSlopeAngle = 35f;

//Crouch & Slide
private Vector3 crouchScale = new Vector3(1, 0.5f, 1);
private Vector3 playerScale;
public float slideForce = 400;
public float slideCounterMovement = 0.2f;

//Jumping
private bool readyToJump = true;
private float jumpCooldown = 0.25f;
public float jumpForce = 550f;

//Input
float x, y;
bool jumping, sprinting, crouching;

//Sliding
private Vector3 normalVector = Vector3.up;
private Vector3 wallNormalVector;

```

After creating the variables for the code, the second thing you want to add is an awake function and a start function. The awake function should come before the start because Awake is called before the game is running.

```

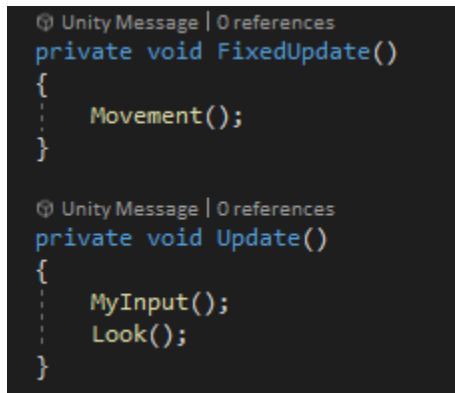
@ Unity Message | 0 references
void Awake()
{
    rb = GetComponent<Rigidbody>();
}

@ Unity Message | 0 references
void Start()
{
    playerScale = transform.localScale;
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
}

```

Inside the awake function you want to get the rigid body of the player and the start function will contain the player scale and should lock the cursor in the center of the screen then hide it.

Next you need to create a fixed update and a regular update and in these pieces of code you need to add your own script and at first you will get errors because you have nothing in that piece of code running anything.



```
Unity Message | 0 references
private void FixedUpdate()
{
    Movement();
}

Unity Message | 0 references
private void Update()
{
    MyInput();
    Look();
}
```

These next pieces of code are a lot longer than the previous ones. Firstly, you need to create the 'MyInput' and in this code you are going to call the Horizontal and Vertical movement which is called in the settings can allows you to move with WSAD and the arrow keys and the same for jumping.

After this you need to create and input that lets you crouch when pressing left control, and inside that you need to create if statements for when the player starts and stops crouching.

After the if statements you must create a 'StartCrouch' and 'StopCrouch' code. Inside the start crouch code, you need to get the players height and make it smaller by halfling it, we will also want to create an if statement to check if the player is grounded and if so, add some force, this will come in handy when you add the sliding. Next you need to obviously have a stop crouch button that will do the opposite of the crouch button.

```

/// <summary>
/// Find user input.
/// </summary>
1 reference
private void MyInput()
{
    x = Input.GetAxisRaw("Horizontal");
    y = Input.GetAxisRaw("Vertical");
    jumping = Input.GetButton("Jump");
    crouching = Input.GetKey(KeyCode.LeftControl);

    //Crouching
    if (Input.GetKeyDown(KeyCode.LeftControl))
        StartCrouch();
    if (Input.GetKeyUp(KeyCode.LeftControl))
        StopCrouch();
}

1 reference
private void StartCrouch()
{
    transform.localScale = crouchScale;
    transform.position = new Vector3(transform.position.x, transform.position.y - 0.5f, transform.position.z);
    if (rb.velocity.magnitude > 0.5f)
    {
        if (grounded)
        {
            rb.AddForce(orientation.transform.forward * slideForce);
        }
    }
}

1 reference
private void StopCrouch()
{
    transform.localScale = playerScale;
    transform.position = new Vector3(transform.position.x, transform.position.y + 0.5f, transform.position.z);
}

```

Next is the Movement code, first we want to add some gravity this is because the normal gravity isn't strong enough. After we need to find the velocity of the player in respective of where the player is looking, next we are just going to tweak the sliding so its smooth. The net bit of code is also tweaking the jump. Now we are going to set a max speed so the player doesn't break the sound barrier, after we must create a piece of code that adds some force on the player when they slide down a ramp to give it a more enjoyable mechanics.

Now we need to solve a problem with the max speed and by adding this specific code to stop the player moving at max speed. Now we need to check the movement speed in the air and add a small multiplier to give the player a little speed boost.

```

1 reference
private void Movement()
{
    //Extra gravity
    rb.AddForce(Vector3.down * Time.deltaTime * 10);

    //Find actual velocity relative to where player is looking
    Vector2 mag = FindVelRelativeToLook();
    float xMag = mag.x, yMag = mag.y;

    //Counteract sliding and sloppy movement
    CounterMovement(x, y, mag);

    //If holding jump && ready to jump, then jump
    if (readyToJump && jumping) Jump();

    //Set max speed
    float maxSpeed = this.maxSpeed;

    //If sliding down a ramp, add force down so player stays grounded and also builds speed
    if (crouching && grounded && readyToJump)
    {
        rb.AddForce(Vector3.down * Time.deltaTime * 3000);
        return;
    }

    //If speed is larger than maxspeed, cancel out the input so you don't go over max speed
    if (x > 0 && xMag > maxSpeed) x = 0;
    if (x < 0 && xMag < -maxSpeed) x = 0;
    if (y > 0 && yMag > maxSpeed) y = 0;
    if (y < 0 && yMag < -maxSpeed) y = 0;

    //Some multipliers
    float multiplier = 1f, multiplierV = 1f;

    // Movement in air
    if (!grounded)
    {
        multiplier = 0.5f;
        multiplierV = 0.5f;
    }

    // Movement while sliding
    if (grounded && crouching) multiplierV = 0f;

    //Apply forces to move player
    rb.AddForce(orientation.transform.forward * y * moveSpeed * Time.deltaTime * multiplier * multiplierV);
    rb.AddForce(orientation.transform.right * x * moveSpeed * Time.deltaTime * multiplier);
}

```

Now we must add the jump code. This code is pretty simple all you need to do is add force the player's rigid body and also create a little piece of code to stop the player double jumping. Next, we need to reset the jump.

```

1reference
private void Jump()
{
    if (grounded && readyToJump)
    {
        readyToJump = false;

        //Add jump forces
        rb.AddForce(Vector2.up * jumpForce * 1.5f);
        rb.AddForce(normalVector * jumpForce * 0.5f);

        //If jumping while falling, reset y velocity.
        Vector3 vel = rb.velocity;
        if (rb.velocity.y < 0.5f)
            rb.velocity = new Vector3(vel.x, 0, vel.z);
        else if (rb.velocity.y > 0)
            rb.velocity = new Vector3(vel.x, vel.y / 2, vel.z);

        Invoke(nameof(ResetJump), jumpCooldown);
    }
}

1reference
private void ResetJump()
{
    readyToJump = true;
}

```

Now this is a lot of code for the looking of the player, in short you need to find mouse x & y and multiply the sensitivity and use a little piece of code that stops the player from over or under rotating, then we need to create a counter movement this is to again slow down the player.

```

1 reference
private void Look()
{
    float mouseX = Input.GetAxis("Mouse X") * sensitivity * Time.fixedDeltaTime * sensMultiplier;
    float mouseY = Input.GetAxis("Mouse Y") * sensitivity * Time.fixedDeltaTime * sensMultiplier;

    //Find current look rotation
    Vector3 rot = playerCam.transform.localRotation.eulerAngles;
    desiredX = rot.y + mouseX;

    //Rotate, and also make sure we dont over- or under-rotate.
    xRotation -= mouseY;
    xRotation = Mathf.Clamp(xRotation, -90f, 90f);

    //Perform the rotations
    playerCam.transform.localRotation = Quaternion.Euler(xRotation, desiredX, 0);
    orientation.transform.localRotation = Quaternion.Euler(0, desiredX, 0);
}

1 reference
private void CounterMovement(float x, float y, Vector2 mag)
{
    if (!grounded || jumping) return;

    //Slow down sliding
    if (crouching)
    {
        rb.AddForce(moveSpeed * Time.deltaTime * -rb.velocity.normalized * slideCounterMovement);
        return;
    }

    //Counter movement
    if (Math.Abs(mag.x) > threshold && Math.Abs(x) < 0.05f || (mag.x < -threshold && x > 0) || (mag.x > threshold && x < 0))
    {
        rb.AddForce(moveSpeed * orientation.transform.right * Time.deltaTime * -mag.x * counterMovement);
    }
    if (Math.Abs(mag.y) > threshold && Math.Abs(y) < 0.05f || (mag.y < -threshold && y > 0) || (mag.y > threshold && y < 0))
    {
        rb.AddForce(moveSpeed * orientation.transform.forward * Time.deltaTime * -mag.y * counterMovement);
    }

    //Limit diagonal running. This will also cause a full stop if sliding fast and un-crouching, so not optimal.
    if (Mathf.Sqrt((Mathf.Pow(rb.velocity.x, 2) + Mathf.Pow(rb.velocity.z, 2))) > maxSpeed)
    {
        float fallspeed = rb.velocity.y;
        Vector3 n = rb.velocity.normalized * maxSpeed;
        rb.velocity = new Vector3(n.x, fallspeed, n.z);
    }
}

```

Now in this piece of code we are again stopping the player from over and under looking, basically limiting their movement. To do this it uses a lot of 'Mathf' and 'Vectors'. After this we need to make an 'IsFloor' code this will check if the player is on the floor.

```

/// <summary>
/// Find the velocity relative to where the player is looking
/// Useful for vectors calculations regarding movement and limiting movement
/// </summary>
/// <returns></returns>
1reference
public Vector2 FindVelRelativeToLook()
{
    float lookAngle = orientation.transform.eulerAngles.y;
    float moveAngle = Mathf.Atan2(rb.velocity.x, rb.velocity.z) * Mathf.Rad2Deg;

    float u = Mathf.DeltaAngle(lookAngle, moveAngle);
    float v = 90 - u;

    float magnitue = rb.velocity.magnitude;
    float yMag = magnitue * Mathf.Cos(u * Mathf.Deg2Rad);
    float xMag = magnitue * Mathf.Cos(v * Mathf.Deg2Rad);

    return new Vector2(xMag, yMag);
}

1reference
private bool IsFloor(Vector3 v)
{
    float angle = Vector3.Angle(Vector3.up, v);
    return angle < maxSlopeAngle;
}

private bool cancellingGrounded;

```

Finally, we are going to have better ground detection. And we will do this with collision code and this is where the 'IsFloor' code will come in.


```

/// <summary>
/// Handle ground detection
/// </summary>
@ Unity Message | 0 references
private void OnCollisionStay(Collision other)
{
    //Make sure we are only checking for walkable layers
    int layer = other.gameObject.layer;
    if (whatIsGround != (whatIsGround | (1 << layer))) return;

    //Iterate through every collision in a physics update
    for (int i = 0; i < other.contactCount; i++)
    {
        Vector3 normal = other.contacts[i].normal;
        //FLOOR
        if (IsFloor(normal))
        {
            grounded = true;
            cancellingGrounded = false;
            normalVector = normal;
            CancelInvoke(nameof(StopGrounded));
        }
    }

    //Invoke ground/wall cancel, since we can't check normals with CollisionExit
    float delay = 3f;
    if (!cancellingGrounded)
    {
        cancellingGrounded = true;
        Invoke(nameof(StopGrounded), Time.deltaTime * delay);
    }
}

2 references
private void StopGrounded()
{
    grounded = false;
}

```

-Camera Movement-

This is all the code that is used for the camera movement which is very small this is because it's an FPS game so I only want the camera to go to the players head position.

```

public Transform player;

@ Unity Message | 0 references
void Update()
{
    transform.position = player.transform.position;
}

```

-Grappling Gun-

For the grappling gun we want to make a line render to show where the grappling gun is going, you need to create these variables which will all come in handy later on.

```

private LineRenderer lr;
private Vector3 grapplePoint;
public LayerMask whatIsGrappleable;
public Transform gunTip, camera, player;
private float maxDistance = 100f;
private SpringJoint joint;

```

For this we need an awake to get the line renderer. Then an update function which gets the mouse input and draw the line to the players mouse position.

```

@ Unity Message | 0 references
void Awake()
{
    lr = GetComponent<LineRenderer>();
}

@ Unity Message | 0 references
void Update()
{
    if (Input.GetMouseButtonDown(0))
    {
        StartGrapple();
    }
    else if (Input.GetMouseButtonUp(0))
    {
        StopGrapple();
    }
}

//Called after Update
@ Unity Message | 0 references
void LateUpdate()
{
    DrawRope();
}

```

Now we are going to call the grapple code and create a ray cast and, in this ray cast we want to check the hit point and a joint between the player and the ray cast and also make the actual line render come from the tip of the gun.

```
/// <summary>
/// Call whenever we want to start a grapple
/// </summary>
1reference
void StartGrapple()
{
    RaycastHit hit;
    if (Physics.Raycast(camera.position, camera.forward, out hit, maxDistance, whatIsGrappleable))
    {
        grapplePoint = hit.point;
        joint = player.gameObject.AddComponent<SpringJoint>();
        joint.autoConfigureConnectedAnchor = false;
        joint.connectedAnchor = grapplePoint;

        float distanceFromPoint = Vector3.Distance(player.position, grapplePoint);

        //The distance grapple will try to keep from grapple point.
        joint.maxDistance = distanceFromPoint * 0.8f;
        joint.minDistance = distanceFromPoint * 0.25f;

        //Adjust these values to fit your game.
        joint.spring = 4.5f;
        joint.damper = 7f;
        joint.massScale = 4.5f;

        lr.positionCount = 2;
        currentGrapplePosition = gunTip.position;
    }
}
```

Finally, we are going to stop the grapple after it has been called and to do this you want to destroy the joint, and we are also going to check if the line has been destroyed.

```

/// <summary>
/// Call whenever we want to stop a grapple
/// </summary>
1reference
void StopGrapple()
{
    lr.positionCount = 0;
    Destroy(joint);
}

private Vector3 currentGrapplePosition;

1reference
void DrawRope()
{
    //If not grappling, don't draw rope
    if (!joint) return;

    currentGrapplePosition = Vector3.Lerp(currentGrapplePosition, grapplePoint, Time.deltaTime * 8f);

    lr.SetPosition(0, gunTip.position);
    lr.SetPosition(1, currentGrapplePosition);
}

1reference
public bool IsGrappling()
{
    return joint != null;
}

1reference
public Vector3 GetGrapplePoint()
{
    return grapplePoint;
}

```

-Gun Rotation-

This is short code that lets the gun of the grappling gun rotate at the point of the grapple and this is because it would look odd if the gun didn't rotate and made the game look unprofessional.

```
public GrapplingGun grappling;  
  
private Quaternion desiredRotation;  
private float rotationSpeed = 5f;  
  
@ Unity Message | 0 references  
void Update()  
{  
    if (!grappling.IsGrappling())  
    {  
        desiredRotation = transform.parent.rotation;  
    }  
    else  
    {  
        desiredRotation = Quaternion.LookRotation(grappling.GetGrapplePoint() - transform.position);  
    }  
  
    transform.rotation = Quaternion.Lerp(transform.rotation, desiredRotation, Time.deltaTime * rotationSpeed);  
}
```