First Person Character Controller: Player and Camera Movement

By Nathan Stalley

In this tutorial, we will be learning how to make a basic 1st Person Character Controller, this character controller is highly customisable and easy to add new features to.

To start off with, we will add the variables that we need to be used to get our character controller working. As these are public variables, they can all be changed within the inspector in Unity and we do not need to define any vales here.

```
private Vector3 Velocity;
private Vector3 PlayerMovementInput;
private Vector2 PlayerMouseInput;
private float xRot;

public Transform PlayerCamera;
public CharacterController Controller;

public float WalkSpeed;
public float SprintSpeed;
public float CrouchSpeed;
public float JumpForce;
public float Sensitivity;
public float Gravity;

public float CrouchHeight;
public float StandingHeight;
public float TimeToCrouch;
public Vector3 CrouchingCentre = new Vector3(0, 0.5f, 0);
public Vector3 StandingCentre = new Vector3(0, 0, 0);
private bool isCrouching;
private bool duringCrouchingAnim;
```

Underneath these variables, we will make a void Awake() method, this will be used to hide the cursor when the game is loaded.

```
void Awake()
{
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
}
```

The next thing we will do is to setup the Players movement and mouse inputs, inside of void Update() we will add in the following pieces of code.

```
void Update()
{
    PlayerMovementInput = new Vector3 (Input.GetAxis("Horizontal"), 0f, Input.GetAxis("Vertical"));
    PlayerMouseInput = new Vector2 (Input.GetAxis("Mouse X"), Input.GetAxis("Mouse Y"));
```

These are easy and efficient ways to add movement into your game, the use of “Horizontal” and “Vertical” will consider the use of WASD as well as the arrow keys and Mouse X & Y will use the players mouse to determine movement.

Next thing we will do is to create a new private void and we will call this Moveplayer(), this will be used and referenced to move the player.

```
private void MovePlayer()
{
    Vector3 MoveVector = transform.TransformDirection(PlayerMovementInput);

    if (Controller.isGrounded)
    {
        Velocity.y = -1f;

        if (Input.GetKeyDown(KeyCode.Space))
        {
            Velocity.y = JumpForce;
        }

    }
    else
    {
        Velocity.y -= Gravity * -2f * Time.deltaTime;
    }

    Controller.Move(MoveVector * (isCrouching ? CrouchSpeed : IsSprinting ? SprintSpeed : WalkSpeed) *Time.deltaTime);
    Controller.Move(Velocity * Time.deltaTime);
}
```

We then use some if statements to make the player jump, the first if statement checks if the player is on the ground and the second if statement checks if the player has pressed the Spacebar, if the player has pressed the Spacebar, the character will jump. IsGrounded is used so that the character can only jump while on the ground and so that they cannot double jump.

Next, we will create a new private void called MoveCamer() this will be used so that the player can use the mouse to make the character look around in the game.

```
private void MoveCamera()
{
    xRot -= PlayerMouseInput.y * Sensitivity;

    xRot = Mathf.Clamp(xRot, -90f, 90f);
    transform.Rotate(0f, PlayerMouseInput.x * Sensitivity, 0f);
    PlayerCamera.transform.localRotation = Quaternion.Euler(xRot, 0f, 0f);
}
```

We start by setting the rotation and multiplying the mouse input by the Sensitivity variable, which we will set in the inspector when we are back in Unity.

We then use mathf.Clamp to lock the rotation of the mouse, we do this so that the mouse stops rotating when it gets to a certain angle and so that the player cannot keep continuously rotating.

Next, we will add in a Coroutine that will handle the crouching for the player.

```
private void HandleCrouch()
{
    if (ShouldCrouch)
    {
        StartCoroutine(CrouchStand());
    }
}

1 reference
private IEnumerator CrouchStand()
{
    if (isCrouching && Physics.Raycast(PlayerCamera.transform.position, Vector3.up, 2f))
        yield break;

    duringCrouchingAnim = true;

    float timeElapsed = 0;
    float targetHeight = isCrouching ? StandingHeight : CrouchHeight;
    float currentHeight = Controller.height;
    Vector3 targetCentre = isCrouching ? StandingCentre : CrouchingCentre;
    Vector3 currentCentre = Controller.center;

    while (timeElapsed < TimeToCrouch)
    {
        Controller.height = Mathf.Lerp(currentHeight, targetHeight, timeElapsed / TimeToCrouch);
        Controller.center = Vector3.Lerp(currentCentre, targetCentre, timeElapsed / TimeToCrouch);
        timeElapsed += Time.deltaTime;
        yield return null;
    }

    Controller.height = targetHeight;
    Controller.center = targetCentre;

    isCrouching = !isCrouching;

    duringCrouchingAnim = false;
```

We will start by making a new private void called HandleCrouch(), this will be used later on to make the player crouch when a key is pressed.

We then make an Enumerator called CrouchStand(), as you can see, this is referenced within HandleCrouch() so do not be alarmed if you get an error straight away.

We then make an if statement that references a Physics.Raycast, this is used to shoot a Raycast straight up in the air from the characters head, it will detect a surface and will not allow the player to un-crouch if there is a surface within range of the Raycast.

We then set up a few variables that will be used in the while() statement. This while() statement is used to move the character from a standing position down to a crouched position and then back to a standing position when the crouch key is pressed.

Next, we need to go back to our void Update() and add the following under the code that we already have there.

These are used as references to the methods that we just created and are placed here so that they can be checked once per frame to see if they need to be run.

```
if (canCrouch)
{
    HandleCrouch();
}

MovePlayer();
MoveCamera();
```

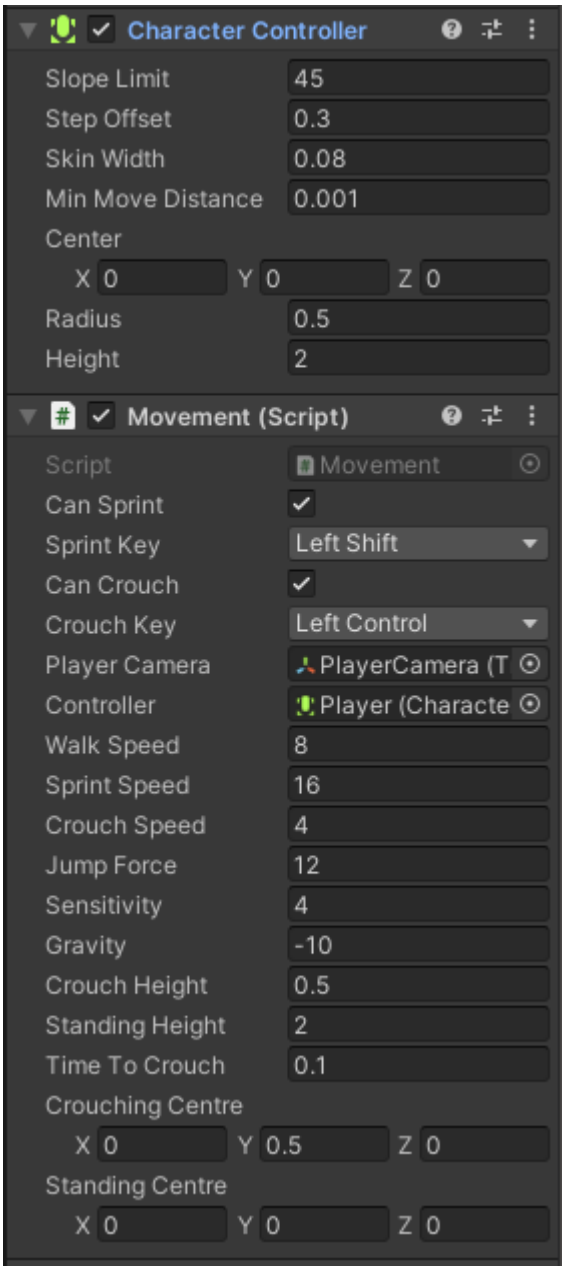
At this point, you may have some red lines under some of your code, this is solved by adding in the following variables at the top of your script.

```
private bool IsSprinting => canSprint && Input.GetKey(sprintKey);
1 reference
private bool ShouldCrouch => Input.GetKeyDown(crouchKey) && !duringCrouchingAnim && Controller.isGrounded;

public bool canSprint = true;
public KeyCode sprintKey = KeyCode.LeftShift;
public bool canCrouch = true;
public KeyCode crouchKey = KeyCode.LeftControl;
```

These are used to set the sprint and crouch keys and to also set them to true, this will allow them to be used within the game and should solve our red line errors.

Finally, we need to head back into Unity and apply this script as well as the Character Controller to our character.



As you can see, I have added the script and the character controller to my character and it works perfectly. These are the settings that I have used for my character, but feel free to play around to get something that suits your needs.

You will also see that there are options for Player Camera and Controller, by default, these will be empty. For the Controller, you simply drag in the player game object from the hierarchy into this slot and if you have the character controller added to your Player, this will work fine.

For the Camera, you will need to place the Camera object to where you want it on the Player. So as this is a 1st person character controller, we will place it at the head of the character, as if we were looking through the eyes of the character. Once you have the camera set up, make it a child of your Player, and then drag the Camera from the hierarchy into the Player Camera slot.

You should now have a basic 1st person character controller that allows your player to walk, sprint, crouch, jump and use the mouse to look around. You can easily add to this script to make your player do more actions if you want.