BEFORE STARTING

Create a new (3D) project with a sample scene with an interactable environment. This should include a floor and terrain like objects for our character to interact with. It is also advisable to have rudimentary lighting set up.

Required packages: Cinemachine

1: CHARACTER MODEL

Create a character model of choice. Creating a simple, sample model is described below:

- Right-click hierarchy > 3D Object > Cylinder (reset, place, and scale the cylinder depending on the context of your sample scene)
- Remove capsule collider component
- Right-click created cylinder > 3D object > Cube
- Place on top of cylinder and transform such that it points in the direction the cylinder is facing
- Remove cube's box collider component

Create an empty object that will act as the parent object for our player object(s). In the prior steps mentioned before, this is easiest done by creating an empty object under the cylinder, resetting position, un-nesting the empty object, and nesting the 3D object under the newly created empty object. It is highly recommended to rename this empty object something memorable like "Third Person Player" for future use.

2: CAMERA

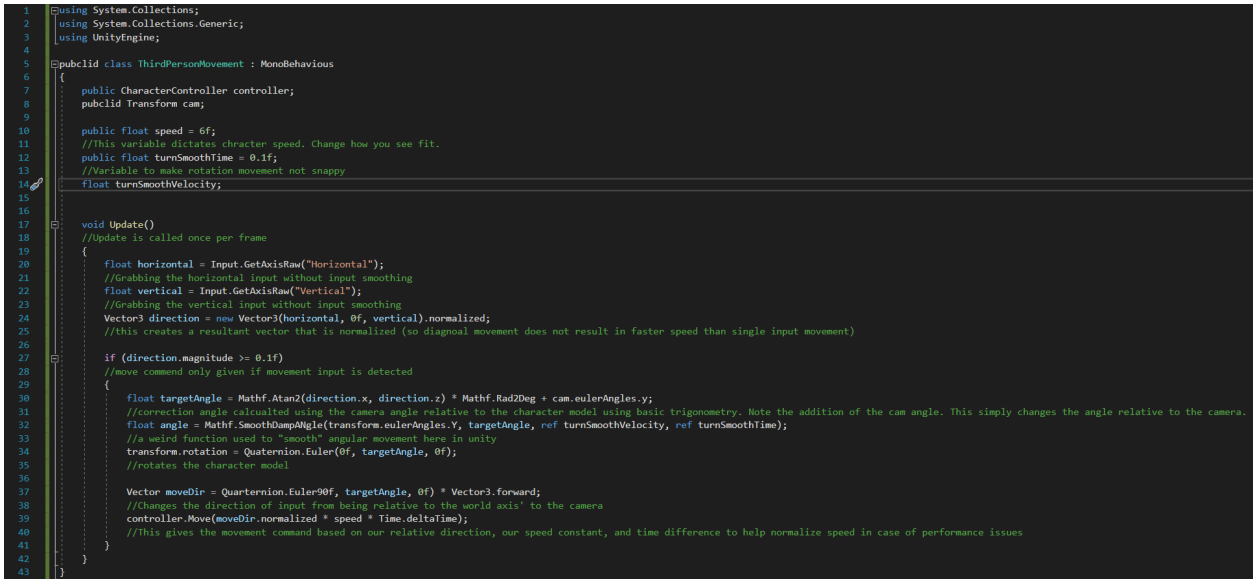
Create a freelook camera. Set the field "Inspector > CinemanchineFreeLook > Follow" and "Inspector > CinemanchineFreeLook > Look At" to our character object. I recommend playing around with the orbit values as, by default, the camera will likely not behave how you would like. The distance between the low, middle, and high rigs will determine how far the player can control the camera vertically while the radius will determine how far the camera distances itself from the model itself.

Binding mode is to be set to "World Space". Camera sensitivity is set via "Axis control >" X Axis" or "Y Axis" > Speed".

3: CHARACTER CONTROLLER AND MOVEMENT

Add a “Character Controller” component to your character model. Set the radius and height respective to your character’s size. If you used the steps provided on the character model, the height should be twice the Y scale and the radius set to half of the X and Z scales.

Add another new, custom component and name it “ThirdPersonMovement” (also under your character model). Code is provided below:



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

pubclid class ThirdPersonMovement : MonoBehavious
{
    public CharacterController controller;
    pubclid Transform cam;

    public float speed = 6f;
    //This variable dictates chracter speed. Change how you see fit.
    public float turnSmoothTime = 0.1f;
    //Variable to make rotation movement not snappy
    float turnSmoothVelocity;


    void Update()
    //Update is called once per frame
    {
        float horizontal = Input.GetAxisRaw("Horizontal");
        //Grabbing the horizontal input without input smoothing
        float vertical = Input.GetAxisRaw("Vertical");
        //Grabbing the vertical input without input smoothing
        Vector3 direction = new Vector3(horizontal, 0f, vertical).normalized;
        //this creates a resultant vector that is normalized (so diagnoal movement does not result in faster speed than single input movement)

        if (direction.magnitude >= 0.1f)
        //move commend only given if movement input is detected
        {
            float targetAngle = Mathf.Atan2(direction.x, direction.z) * Mathf.Rad2Deg + cam.eulerAngles.y;
            //correction angle calcualted using the camera angle relative to the character model using basic trigonometry. Note the addition of the cam angle. This simply changes the angle relative to the camera.
            float angle = Mathf.SmoothDampANgle(transform.eulerAngles.Y, targetAngle, ref turnSmoothVelocity, ref turnSmoothTime);
            //a weird function used to "smooth" angular movement here in unity
            transform.rotation = Quaternion.Euler(0f, targetAngle, 0f);
            //rotates the character model

            Vector moveDir = Quarternion.Euler90f, targetAngle, 0f) * Vector3.forward;
            //Changes the direction of input from being relative to the world axis' to the camera
            controller.Move(moveDir.normalized * speed * Time.deltaTime);
            //This gives the movement command based on our relative direction, our speed constant, and time difference to help normalize speed in case of performance issues
        }
    }
}
```

Note: Code explanation in the comments.

Set cam to “Main Camera” (not “Third Person Camera”).

4: MORE CAMERA STUFF

Under Third Person Camera, add extension “Cinemachine Collider” and change “Collide Against” to all layers you do not want your camera to be intercepted by or clipped into and set “Ignore Tag” to your character Model. Change “Strategy” to “Pull Camera Forward”.

Optional: You can play around with some values you damping and lens clip plane to customize the camera & environment experience.