For this tutorial I will be making a proper first person camera on a moving player object

The aim of this tutorial is to make a smooth fully operational first person perspective of a play within a scene that can move around and interact with other things in the future.

To begin this tutorial we need to make a new script. Name this script FPSController or anything that you will remember.

Now we can start inserting the code that will help us move.

First above, the public class inserts the following line of code. This code will insert a character controller into the scene if it is missing. It should look like the image below.

```
[RequireComponent(typeof(CharacterController))]
public class FPSController : MonoBehaviour
```

Next we need to add the variables that will determine the various features of the movement script and how they work. For this we will need how fast the player walks, how fast they are when they sprint, how high they can jump and how much they are affected by gravity.
This also includes how quickly the camera rotates with the mouse. The variables will be laid out as the following.

```
{
    public Camera playerCamera;
    public float walkSpeed = 6f;
    public float runSpeed = 12f;
    public float jumpPower = 7f;
    public float gravity = 10f;

    public float lookSpeed = 2f;
    public float lookXLimit = 45f;

    Vector3 moveDirection = Vector3.zero;
    float rotationX = 0;

    public bool canMove = true;

    CharacterController characterController;
```

After this we need to set the start method for the script. These lines of code will get our character controller and hide the cursor when we play the scene.

```
// Start is called before the first frame update
void Start()
{
    characterController = GetComponent<CharacterController>();
    Cursor.lockState = CursorLockMode.Locked;
    Cursor.visible = false;
}
```

For the actual movement part of the code the lines are as follows. This code will change how the movement works depending on the face of the camera so that wherever the player is looking is what 'forward' is and vice versa for other directions. This will also detect if the player is sprinting and bring up the speed to the variable put into sprint speed earlier.

```
void Update()
{

    Vector3 forward = transform.TransformDirection(Vector3.forward);
    Vector3 right = transform.TransformDirection(Vector3.right);

    // Use Left Shift to Run :D
    bool isRunning = Input.GetKey(KeyCode.LeftShift);
    float curSpeedX = canMove ? (isRunning ? runSpeed : walkSpeed) * Input.GetAxis("Vertical") : 0;
    float curSpeedY = canMove ? (isRunning ? runSpeed : walkSpeed) * Input.GetAxis("Horizontal") :
    float movementDirectionY = moveDirection.y;
    moveDirection = (forward * curSpeedX) + (right * curSpeedY);
```

Next we need to assign the code that allows the player to jump. This code will check for if the player is grounded when they attempt to jump. This happens because if the player attempts to jump when they have already left the floor or are falling, the code will prevent them from being able to jump again on nothing and potentially fly away, which is not something I want to happen with my prototype. This code will also make sure that the direction of jumping is relative to the movement of the player not the camera as this allows the player to keep suitable movement and momentum when looking away from where they are jumping to.

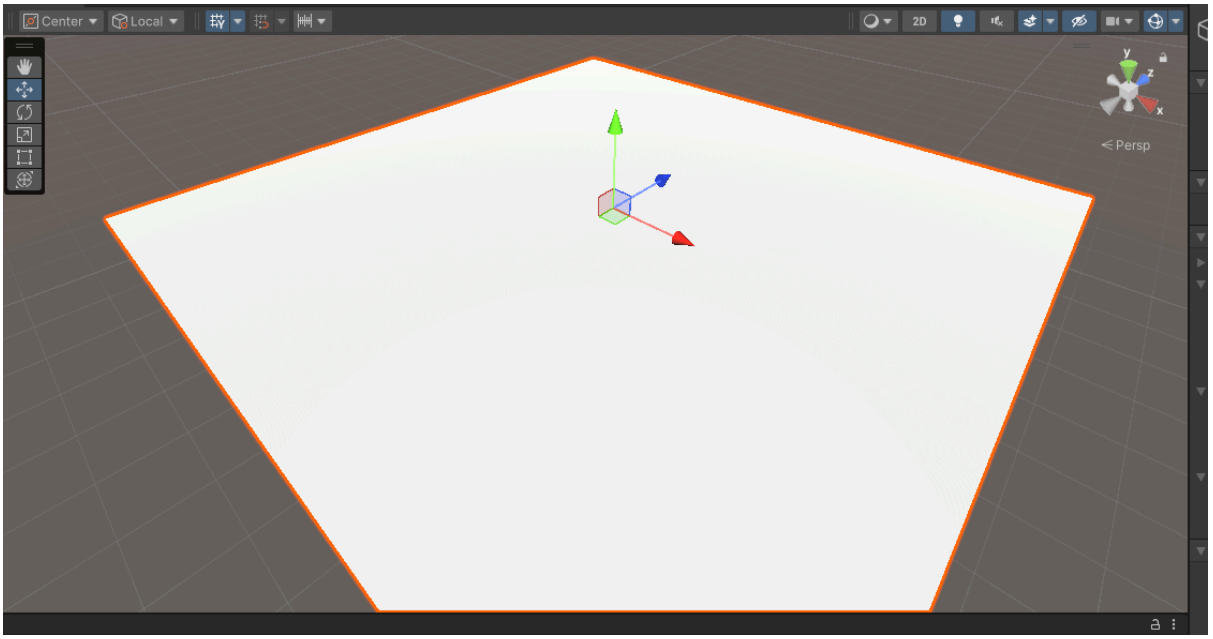
```
if (Input.GetButton("Jump") && canMove && characterController.isGrounded)
{
    moveDirection.y = jumpPower;
}
else
{
    moveDirection.y = movementDirectionY;
}

if (!characterController.isGrounded)
{
    moveDirection.y -= gravity * Time.deltaTime;
}

characterController.Move(moveDirection * Time.deltaTime);
```

Finally the last section of code for the movement will handle the rotation of the camera based on the mouse controls. The movement of the mouse will turn the camera based on the camera speed variable determined earlier and this affects both X and Y translations of the camera. This code also holds where the rotation of the camera and player are as both need to rotate in sync for the point of forward to be the same for both objects.

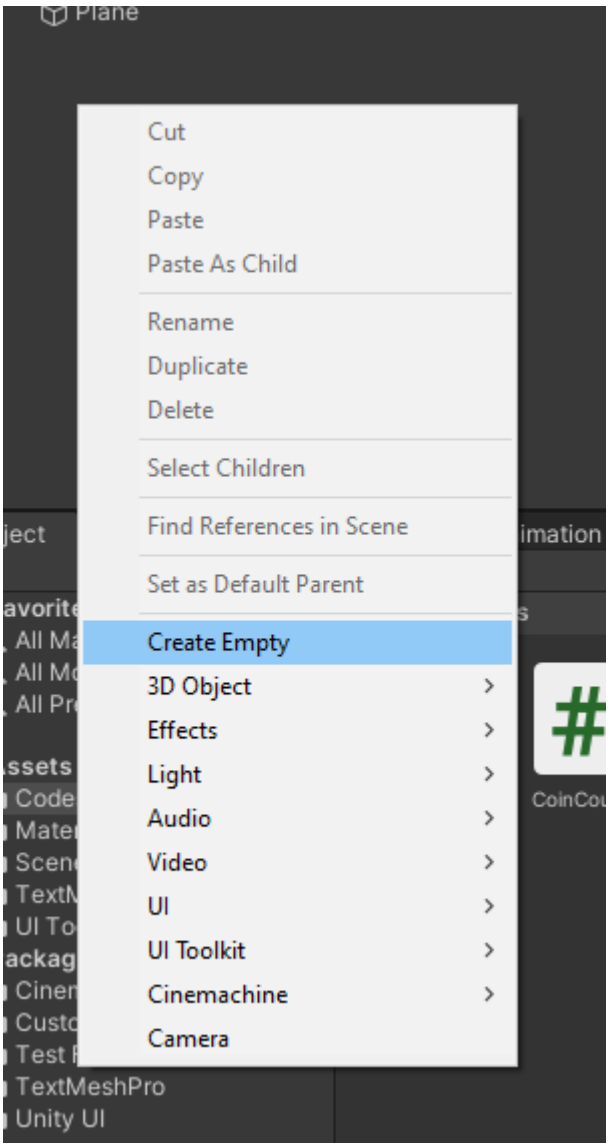
```
characterController.Move(moveDirection * Time.deltaTime);

if (canMove)
{
    rotationX += -Input.GetAxis("Mouse Y") * lookSpeed;
    rotationX = Mathf.Clamp(rotationX, -lookXLimit, lookXLimit);
    playerCamera.transform.localRotation = Quaternion.Euler(rotationX, 0, 0);
    transform.rotation *= Quaternion.Euler(0, Input.GetAxis("Mouse X") * lookSpeed, 0);
}
```

Now we can assign our code to be used in the scene. First however we need to make this scene.

In the hierarchy of the scene create a large plane. This will be the floor that the player will run around on.



Next we can set up the player itself. Create an empty object and name it Player. This will become the parent of the other assets we will be adding layers including the camera and the capsule controller itself.



Next make a capsule object and place it within the Player object for it to be parented and then drag the main camera object in as well so that both objects are inside the empty player object. Making sure that the capsule and the camera's location is 0 in all dimensions will help move the camera higher up within the capsule to about where head height would be instead of viewing from the chest area.

Now add the player script to the parent object player object. Note; if you add this script to the capsule and not the parents the object's camera will not be able to move horizontally.

Finally put the main camera object into the player camera box as seen in the image above and the player should be fully functionable within the screen. (Note; the mouse will not be removed from the screen until you click on the scene window and will reappear again when hitting Esc.)