Tutorial 1 – 2D Player Movement

This tutorial will go through creating a 2D player movement script, with variable speed, acceleration and jump height, perfect for platformers.

The first step is to create a player object and a platform for the player to stand on. I do this using a white square .png and dragging it into the scene; one for the player and one for the ground, resizing them until they look appropriate. *I recommend renaming them to avoid confusion.*

Add a Rigidbody2D to the player and a BoxCollider2D to both the player and the floor. Now when you press play the player should drop to the ground but not fall through.

Create a new script called PlayerController.cs and add it to the player object. Start the script by defining four public floats (the player's acceleration, the player's deceleration, the player's maximum speed and the player's jump force) and one public layer mask (the layer for all ground objects). We also add three private variables: a private float to store the player's current speed, a private Rigidbody2D to store the player's rigidbody and a private bool to store whether or not the player has jumped.

```
public float accelSpeed = 100f, decelSpeed = 50f, maxSpeed = 10f, jumpForce = 370f;
public LayerMask groundLayer;

Rigidbody2D rb;
float currentSpeed = 0;
bool hasJumped = false;
```

In the start function, add the following code to get the player's rigidbody and assign it to the variable.

```
rb = GetComponent<Rigidbody2D>();
```

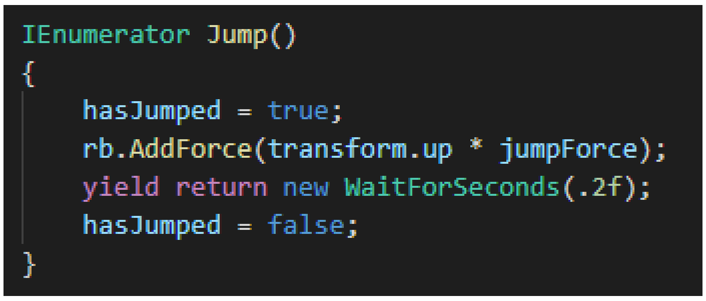
In the update function, create two new variables and assign them the raw vertical axis and the raw horizontal axis. This gets inputs for horizontal movement as well as jumping.

```
float v = Input.GetAxisRaw("Vertical");
float h = Input.GetAxisRaw("Horizontal");
```

Below this, write the following code. This code adds or subtracts the current acceleration but prevents the speed going over the max speed.

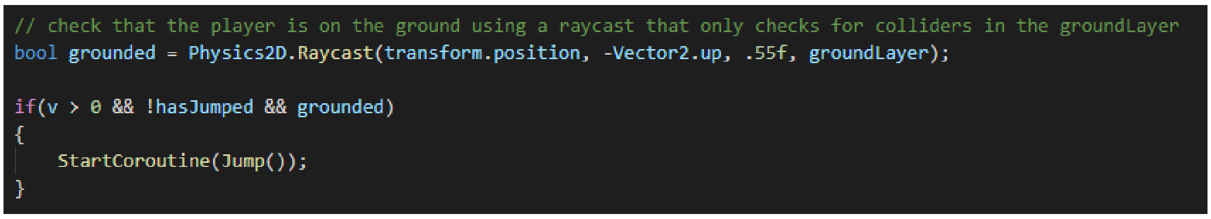
```
// add acceleration in the correct direction
currentSpeed += accelSpeed * h * Time.deltaTime;

// add deceleration in the opposite direction (to stop infinite movement)
if(currentSpeed > 0.01f) currentSpeed -= decelSpeed * Time.deltaTime;
else if (currentSpeed < -0.01f) currentSpeed += decelSpeed * Time.deltaTime;
else currentSpeed = 0;

// ensure the player isn't moving too fast then set the speed to the rigidbody's velocity
currentSpeed = Mathf.Clamp(currentSpeed, -maxSpeed, maxSpeed);
rb.velocity = new Vector2(currentSpeed, rb.velocity.y);
```

Now, create a new IEnumerator method called Jump(). Inside this we need to set hasJumped to true, add jumpForce to the player upwards, wait a short amount of time and then finally set hasJumped back to false. This ensures the player only gets the force added once per jump.



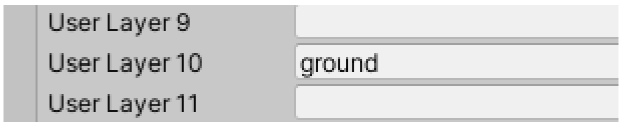
```
IEnumerator Jump()
{
    hasJumped = true;
    rb.AddForce(transform.up * jumpForce);
    yield return new WaitForSeconds(.2f);
    hasJumped = false;
}
```

Finally, we need to call this method when three parameters are met: the player wants to jump (v > 0), the player hasn't already jumped too recently (hasJumped == false) and finally the player is grounded. This can be done using the following code.



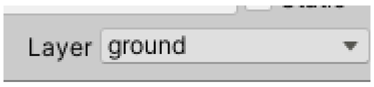
```
// check that the player is on the ground using a raycast that only checks for colliders in the groundLayer
bool grounded = Physics2D.Raycast(transform.position, -Vector2.up, .55f, groundLayer);

if(v > 0 && !hasJumped && grounded)
{
    StartCoroutine(Jump());
}
```

Now that all the code is finished, all we have to do is assign the groundLayer. First, go to the ground layer, click the layer box and then add a new layer called ground.



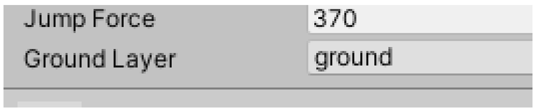
User Layer 9	
User Layer 10	ground
User Layer 11	

Go back to the ground object and set it's layer to ground.



Layer ground

Finally, go to the player, find the playerController script and set the groundLayer variable to 'ground'



Jump Force	370
Ground Layer	ground