Speed up animation on player touch.

Before void start you need to call the animator fuction. Ex: Animator anim;

And recall it on the void Start: anim = GetComponent<Animator>();

Now you have to create a new void that will activate when the player will enter in concact whit the game object. So:

void OnTriggerEnter(Collider other)

{

if (other.tag == "Player")

{  
}}

The if will so compare the tag of the game object. For make the if function work we will need only to tag the player whit the Player tag.   
Now we just need to call the animator speed in the if function and increase it. The basic speed of the animator is 1, so:

if (other.tag == "Player")

{

anim.speed = 4f;}

If we want the speed to go back to the original speed after few second (5 in the example) we will need to create a new void whit the animator speed back to 1, and invoke it in the if function.

if (other.tag == "Player")

{

anim.speed = 4f;

Invoke("SlowDown", 5f);

}

void SlowDown()

{

anim.speed = 1f;

}

The f number inside the parenthesis of the invoke function will determinate the delay in second.

Play a musical scale on different trigger.

If we want to play the different sound of a musical scale when the player touch different trigger we need to code a list inside the player. First of all we will need to create a list and call it back in the void Start:

List<AudioSource> bouncesounds = new List<AudioSource>();public AudioSource sound1;

public AudioSource sound2;

public AudioSource sound3;

public AudioSource sound4;

public AudioSource sound5;

void Start()

{

bouncesounds.Add(sound1);

bouncesounds.Add(sound2);

bouncesounds.Add(sound3);

bouncesounds.Add(sound4);

bouncesounds.Add(sound5);

}

We will also need to set a number on the list, for be scure that the sounds will be play in the right order. We will do it using a int.

public int bouncecount = 0;

This will be linked to the list.

Now we have to create the on trigger void. It will be like this:

void OnTriggerEnter(Collider other)

{

if (other.CompareTag("Bounce")){

bouncesounds[bouncecount].Play ();

bouncecount += 1;

}}

We just need to remember to tag the game object whit the tag we prefer (Bounce in the example) and the game will play the first sound of the list (sound1 in the example, that is the number 0 of the list) and then add 1 to the bouncecount int, giving the possibility to play the next sound on the next collision. But when the number will grow up to the length of the list this will stop play sound. For fix this problem we will need to ad this:

if (bouncecount > 4){

bouncecount = 0;

}

Where the first number is the length of our list -1, because the list always start whit 0.

Turn light on when player collide whit the game object.

First of all, we need to call the light, the intensity of the light, create a private bool for turn the light on and off, and create a constant float. We will also need a public float. So:

public Light lgt;

public float area;

private const float coef = 100.2f;

public float intensity;

private bool lightOn;

void Start()

{

lgt = GetComponent<Light>();

lightOn = false;

}

Now we need the OnTrigger void. This will set the bool lightOn on true and will turn on the light.

private void OnTriggerEnter(Collider other)

{

if (other.CompareTag("Player")&& lightOn == false)

{

area = 450;

lightOn = true;

Invoke(“lightOff”, 5f);

}

}

So we’ll also have a void for light off.

void lightOff()

{

lightOn = false;

}

Now we can move to void Update. What we need is the light to go down whit the time, so we have to call the area float and make it decrease with the time.

void Update()

{

area -= coef \* Time.deltaTime;

lgt.intensity = area;

}

If we change the number linked to coef the speed of decreasing will change.

Player movement

First of all we have to fix some public float and call the rigid body of the player, so:

public Rigidbody rb;

public float movementSpeed;

public float jumpForce;

private bool isGrounded;  
 private float inputHorizontal, inputVertical;

void Start()

{

rb = GetComponent<Rigidbody>();

}

Now we have to call the axis and to prepare the void for the movement of the player. So:

void Update()

{

inputHorizontal = Input.GetAxis("Horizontal");

inputVertical = Input.GetAxis("Vertical");

PlayerMovement();

And we create a new void call void PlayerMovement. This void will use a translate function for move the player and will call the public float movementSpeed.

void PlayerMovement()

{

float hor = Input.GetAxis("Horizontal");

float ver = Input.GetAxis("Vertical");

Vector3 playerMovment = new Vector3(hor, 0f, ver) \* movementSpeed \* Time.deltaTime;

transform.Translate(playerMovment, Space.Self);

}

Now we have to create the jump function. We will go back to void Update and we will add an if function.

if (Input.GetKeyDown(KeyCode.Space) && isGrounded)

{

Jump();

}

But when isGrounded is working? We will need to create 2 void more for fix the value of this component. So:

private void OnCollisionExit(Collision collision)

{

if (collision.gameObject.tag == "Floor")

{

isGrounded = false;

}

}

private void OnCollisionEnter(Collision collision)

{

if (collision.gameObject.tag == "Floor")

{

isGrounded = true;

}

}

And so we will only to tag the floor as Floor.   
Now we only need to code the jump function. We will use an add force whit as force the public float jupForce. So:

void Jump()

{

rb.AddForce(Vector3.up \* jumpForce, ForceMode.Impulse);

}

And now the player will be able to move and jump in a 3D space.