Rotación Quaternians.

```
if (movementDirections != Vector3.zero)
      {
         Quaternion toRotation = Quaternion.LookRotation(movementDirections, Vector3.up);

         transform.rotation = Quaternion.RotateTowards(transform.rotation, toRotation, rotationSpeed *
Time.deltaTime);
      }
   }
```

Primero creamos una variable publica para designar la velocidad de rotación que queremos implementar.

public float rotationSpeed;

Seguido tenemos que darle destino a esta variable, así como suavizar el sistema de transiciones entre los movimientos que requieren cambio de poscición, en nuestro caso una rotación.

Rotación Quaternians.

```
if (movementDirections != Vector3.zero)
    {
        Quaternion toRotation = Quaternion.LookRotation(movementDirections, Vector3.up);

        transform.rotation = Quaternion.RotateTowards(transform.rotation, toRotation, rotationSpeed *
Time.deltaTime);
    }
  }
```

Quaternion toRotation Hacemos uso de los **Quaternion,** que son una variable de tipo especifico para almacenar rotaciones.

Rotación Quaternians.

Quaternion toRotation = Quaternion.LookRotation(movementDirections, Vector3.up);

Quaternion.LookRotation-

Crea una rotación con las direcciones forward y upwards especificadas.

La función **LookRotation** de la clase Quaternion, lo que hace es rotar con respecto al Vector3 que le pasemos como parámetro, devolviendo el Quaternion resultante. El Vector3 que le pasaremos será movementDirections, así que lo que hará LookRotation es rotar a nuestro personaje de manera que quede mirando en el sentido de la dirección del movimiento de dicho vector.

Vector3.up Forma corta de escribir Vector3(0, 1, 0).

transform.rotation = Quaternion.RotateTowards(transform.rotation, toRotation, rotationSpeed * Time.deltaTime);

Quaternion.RotateTowards- Rotamos desde la posicion en la que nos encontramos hacia la que queremos dirigirnos, esto lo conseguimos mediante el **transform.rotation,** rotación, velocidad y todo multiplicado por **Time.deltaTime.**

Movimiento básico.

El personaje puede moverse por nuestra escena en función de la entrada del teclado o el **gamepad** .El problema es que el personaje siempre se mueve a la misma velocidad por mucho que movamos el joystick del gamepad. No hay diferencia entre mover el joystick completamente en una dirección y moverlo una fracción.

En la mayoría de los juegos, la velocidad del personaje reflejará el movimiento del joystick. **Ejemplo**, el personaje puede caminar lentamente cuando el joystick se mueve una fracción y correr cuando se mueve por completo.

Rotación Quaternians.

```
void Update()
{
    float horizontalInput = Input.GetAxis("Horizontal");
    float verticalInput = Input.GetAxis("Vertical");

    Vector3 movementDirection = new Vector3(horizontalInput, 0, verticalInput);
    movementDirection.Normalize();

    transform.Translate(movementDirection speed * Time.deltaTime, Space.World);

    if (movementDirection != Vector3.zero)
    {
        Quaternion toRotation = Quaternion.LookRotation(movementDirection, Vector3.up);
        transform.rotation = Quaternion.RotateTowards(transform.rotation, toRotation, rotationAnglePerSecond *
Time.deltaTime)
```

Sensibilidad Joystick.

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

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

Estamos recibiendo la entrada del eje horizontal y el eje vertical.

Estos valores de entrada oscilarán entre -1 y 1 y reflejarán correctamente el movimiento del joystick.

Si el joystick está medio hacia la derecha, el valor de la entrada horizontal será 0,5.Luego creamos un vector a partir de estos valores de entrada para nuestra dirección de movimiento. Luego normalizamos este vector antes de mover el personaje en esta dirección. El problema en realidad se introduce en la línea donde normalizamos el vector.

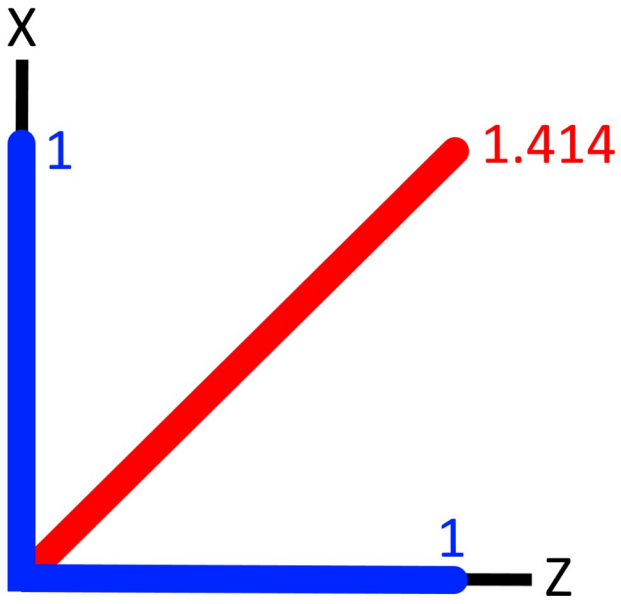
Vector3 movementDirection = new Vector3(horizontalInput, 0, verticalInput);

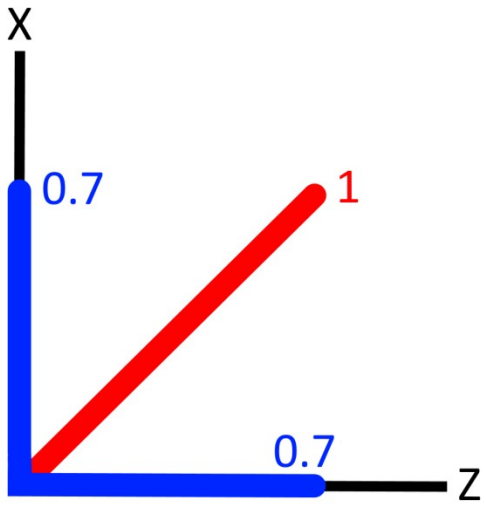
movementDirection.Normalize();

Sensibilidad Joystick.

Si nuestro vector es 1 en los ejes X y Z, entonces la magnitud de nuestro vector es 1.414.

Normalize mantiene la dirección pero establece la magnitud en 1.
Por lo tanto, en este escenario, la normalización de nuestro vector reducirá el tamaño en los ejes X y Z a 0,7, haciendo que la magnitud sea 1 mientras se mantiene la dirección.





Sensibilidad Joystick.

magnitude = Mathf.Clamp01(magnitude);

Todavía queremos normalizar el vector, ya que un vector de dirección debe tener una magnitud de 1.

- Pero antes de hacerlo, almacenaremos la magnitud.
- Luego multiplicamos por esta magnitud al mover nuestro personaje.
- Para limitar la magnitud a un rango de 0 a 1, usamos el método Clamp01.

Mathf.Clamp

Fija el valor dado entre los valores de flotación mínimo y máximo dados. Devuelve el valor dado si está dentro del rango mínimo y máximo.

Uso -para restringir un valor a un rango definido por los valores mínimo y máximo.

Nota: si el valor mínimo es mayor que el valor máximo, el método devuelve el valor mínimo.

Sensibilidad Joystick.

```
void Update()
  {
    float horizontalInput = Input.GetAxis("Horizontal");
    float verticalInput = Input.GetAxis("Vertical");

    Vector3 movementDirections = new Vector3(horizontalInput, 0, verticalInput);
    float magnitude = movementDirections.magnitude;
    magnitude = Mathf.Clamp01(magnitude);

    movementDirections.Normalize();

    transform.Translate(movementDirections * magnitude* speed * Time.deltaTime, Space.World);

    if (movementDirections != Vector3.zero)
    {
      Quaternion toRotation = Quaternion.LookRotation(movementDirections, Vector3.up);

      transform.rotation = Quaternion.RotateTowards(transform.rotation, toRotation, rotationSpeed * Time.deltaTime);
```

```
public class Player : MonoBehaviour
{
    public float speed;
    public float rotationSpeed;

    // Update is called once per frame
    void Update()
    {
        float horizontalInput = Input.GetAxis("Horizontal");
        float verticalInput = Input.GetAxis("Vertical");

        Vector3 movementDirections = new Vector3(horizontalInput, 0, verticalInput);
        float magnitude = movementDirections.magnitude;
        magnitude = Mathf.Clamp01(magnitude);

        movementDirections.Normalize();

        transform.Translate(movementDirections * magnitude* speed * Time.deltaTime, Space.World);

        if (movementDirections != Vector3.zero)
        {
            Quaternion toRotation = Quaternion.LookRotation(movementDirections, Vector3.up);

            transform.rotation = Quaternion.RotateTowards(transform.rotation, toRotation, rotationSpeed * Time.deltaTime);
        }
    }
```