# Código C#

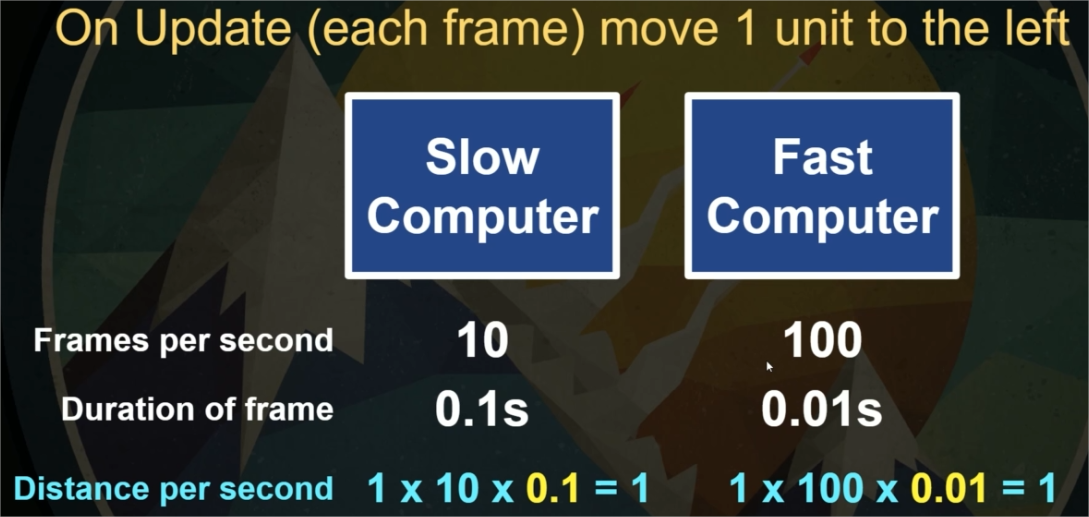
## [SerializeField()]

Se pone la variable disponible en el inspector para modificar cómo si fuera pública, pero no es accesible desde otros scripts.



## Time.deltaTime

* Es la manera de Unity de decirnos cuando tarda en ejecutarse cada frame.
* Cuando multiplicamos algo por Time.deltaTime, hacemos nuestro juego independiente del frame rate. Es decir, el juego se comporta igual sin importar lo rápida o lenta que sea la computadora.



## SceneManager.LoadScene

Método encargado de cargar la escena especificada. La escena especificada es un número que se obtiene mirando el Build Settings

if (other.gameObject.CompareTag("Ground"))

SceneManager.LoadScene(0);

## Crear un delay

Hay dos formas de hacerlo:

### Invoke

Es un poco más fácil de entender, pero no muy poderoso. Es un método que recibe dos parámetros, el nombre del método y el delay.

### Corutinas

## FindObjectOfType<>()/FindObjectsOfType<>()

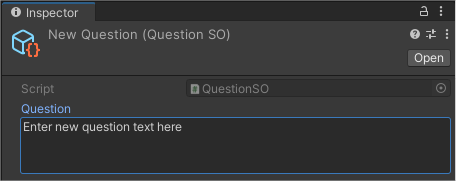
Sirve para encontrar objetos del tipo especificado. Tener cuidado si hay varios componentes del mismo tipo.

## [TextArea()]:

Hace que las variables se puedan definir en un área de texto, definiendo minimo y máximo de lineas.

[TextArea(2, 6)]

[SerializeField] string question = "Enter new question text here";



## ScriptableObject:

Define al script como uno que se encargará de crear scriptable objects.

[CreateAssetMenu(menuName = "Quiz Question", fileName = "New Question")]

public class QuestionSO : ScriptableObject

{

[TextArea(2, 6)]

[SerializeField] string question = "Enter new question text here";

}

## CreateAssetObject():

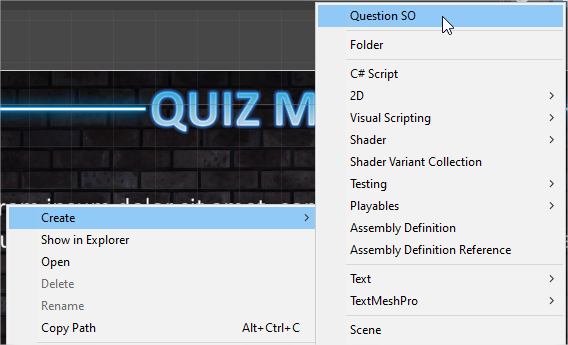
Hace que se pueda crear un objeto del script:

[CreateAssetMenu()]

public class QuestionSO : ScriptableObject

{

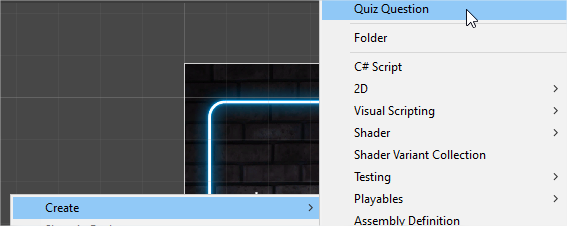
}



### Opciones

* MenuName: define el nombre con el que aparecerá en el menú de creación.
* fileName: define un nombre por defecto de los objetos creados.

[CreateAssetMenu(menuName = "Quiz Question", fileName = "New Question")]



## Cambiar sprite de un objeto:

Image buttonImage = answersButtons[index].GetComponent<Image>();

buttonImage.sprite = correctAnswerSprite;

## [Header()]

Sirve para agregarle una cabecera a las propiedades cuando se muestren en el inspector.

[Header("Questions")]

[SerializeField] TextMeshProUGUI questionText;

[SerializeField] QuestionSO question;

[Header("Answers")]

[SerializeField] GameObject[] answersButtons;

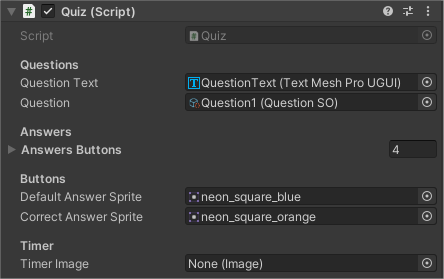
[Header("Buttons")]

[SerializeField] Sprite defaultAnswerSprite;

[SerializeField] Sprite correctAnswerSprite;

[Header("Timer")]

[SerializeField] Image timerImage;



## Movimiento del jugador

Se puede utilizar el siguiente método:

void OnMove(InputValue value)

{

moveInput = value.Get<Vector2>();

}

void Run()

{

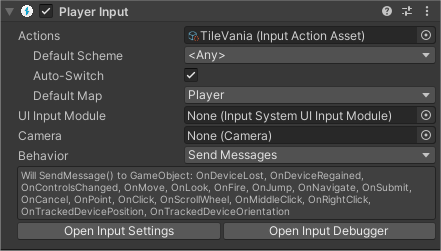
float movement = moveInput.x \* runSpeed;

Vector2 playerVelocity = new Vector2(movement, rigidBody.velocity.y);

rigidBody.velocity = playerVelocity;

}

El move input se obtendría con el objeto Player Input:



## Dar vuelta el sprite del jugador:

void FlipSprite()

{

bool playerHasHorizontalSpeed = Mathf.Abs(rigidBody.velocity.x) > Mathf.Epsilon;

if(playerHasHorizontalSpeed)

transform.localScale = new Vector2(Mathf.Sign(rigidBody.velocity.x), 1f);

}

Mathf.Abs se usa para obtener el absoluto de la velocidad horizontal.

Mathf.Epsilon es un número muy pequeño. Se usa en lugar del 0 ya que es más limpio.

Mathf.Sign se utiliza para decidir si el número dado por parámetro es positivo (1) o negativo (-1).

## Collider.IsTouchingLayers()

Sirve para verificar si el collider está tocando una layer especificada.

if (!collider.IsTouchingLayers(LayerMask.GetMask("Ground"))) return;