Guía de estilo de C#

Tener convenciones de codificación bien definidas y consistentes es importante para cada proyecto, y Godot no es una excepción a esta regla.

Esta página contiene una guía de estilo de codificación que es seguida por los desarrolladores y colaboradores de Godot. Como tal, está dirigido principalmente a aquellos que quieren contribuir al proyecto, pero dado que las convenciones y directrices mencionadas en este artículo son las de mayor adopción por parte de los usuarios del lenguaje, te animamos a que hagas lo mismo, especialmente si aún no dispones de dicha guía.

Nota

Este artículo no es de ninguna manera una guía exhaustiva sobre cómo seguir las convenciones de codificación estándar o las mejores prácticas. Si se siente inseguro de un aspecto que no se cubre aquí, por favor refiérase a una documentación más completa, tal como C# Coding Conventions o Framework Design Guidelines .

Especificación del lenguaje

Actualmente, Godot utiliza la **versión 7.0 de C#** en su motor y código fuente de ejemplo. Por lo tanto, antes de pasar a una versión más nueva, hay que tener cuidado de no mezclar las funciones de lenguaje sólo disponibles en C# 7.1 o posterior.

Para obtener más información sobre las características de C# en diferentes versiones, consulta What's New in C# .

Formateando

Lineamientos generales

- Utiliza caracteres de salto de línea (**LF**) para separar líneas, no CRLF o CR.
- Usa un carácter de entrada de línea al final de cada archivo, excepto para los archivos "csproj".
- Utilice la codificación **UTF-8** sin Marca de orden de bytes .
- Usa **4 espacios** en lugar de tabuladores para la sangría (lo que se denomina 'tabulaciones suaves').

- Considere la posibilidad de dividir una línea en varias si es más larga de 100 caracteres.

Saltos de línea y líneas en blanco

Para una regla general de identación, sigue `el 'Estilo Allman' <https://en.wikipedia.org/wiki/Identation_style#Allman_style >_ que recomienda poner el corchete asociado con una estructura de control en la siguiente línea, identado al mismo nivel:

```
// Use this style:
if (x > 0)
{
    DoSomething();
}

// NOT this:
if (x > 0) {
    DoSomething();
}
```

Sin embargo, puedes optar por omitir los saltos de línea dentro de llaves:

- Para operadores de miembros simples.
- Para objetos simples, arrays o inicializadores de colección.
- Para abstraciones de propiedades auto, indexaciones, o declaraciones de evento.

```
// You may put the brackets in a single line in following cases:
public interface MyInterface
{
    int MyProperty { get; set; }
}

public class MyClass : ParentClass
{
    public int Value
    {
        get { return 0; }
        set
        {
            ArrayValue = new [] {value};
        }
    }
}
```

Inserte una línea en blanco:

- Después de la lista de sentencias `using`.
- Entre métodos, propiedades y declaraciones de tipo internas.
- Al final de cada archivo.

Las declaraciones de campos y constantes se pueden agrupar según la relevancia. En ese caso, considera insertar una línea en blanco entre los grupos para facilitar la lectura.

Evita insertar una línea en blanco:

- Después de un corchete de apertura `{`.
- Antes de un corchete de cierre ``}`.
- Después de un bloque de comentarios, o un comentario de una sola línea.
- Adyacente a otra línea en blanco.

```
using System;
using Godot;
                                        // Blank line after `using` list.
public class MyClass
{                                       // No blank line after `{`.
    public enum MyEnum
    {
        Value,
        AnotherValue                    // No blank line before `}`.
    }
                                        // Blank line around inner types.
    public const int SomeConstant = 1;
    public const int AnotherConstant = 2;

    private Vector3 _x;                 // Related constants or fields can be
    private Vector3 _y;                 // grouped together.

    private float _width;
    private float _height;

    public int MyProperty { get; set; }
                                        // Blank line around properties.
    public void MyMethod()
    {
        // Some comment.
        AnotherMethod();                // No blank line after a comment.
    }
                                        // Blank line around methods.
    public void AnotherMethod()
    {
    }
}
```

Usando espacios

Inserta un espacio:

- Alrededor de un operador binario y ternario.
- Entre un paréntesis de apertura y las palabras clave `if`, `for`, `foreach`, `catch`, `while`, `lock` o `using`.
- Antes y dentro de un bloque de accesorios de una sola línea.
- Entre los accesorios en un bloque de accesorios de una sola línea.
- Después de una coma que no está al final de una línea.
- Después de un punto y coma en una sentencia `for`.

- Después de una sentencia de dos puntos en un `case` de una sola línea.
- En torno a los dos puntos en una declaración de tipo.
- En torno a una flecha lambda.
- Después de un símbolo de comentario de una sola línea (`//`), y antes de él si se utiliza al final de una línea.

No uses un espacio:

- Después de un paréntesis en conversión de tipo.
- inicialización de llaves de una sola línea.

El siguiente ejemplo muestra un uso adecuado de los espacios, de acuerdo con algunas de las convenciones antes mencionadas:

```
public class MyClass<A, B> : Parent<A, B>
{
    public float MyProperty { get; set; }

    public float AnotherProperty
    {
        get { return MyProperty; }
    }

    public void MyMethod()
    {
        int[] values = {1, 2, 3, 4}; // No space within initializer brackets.
        int sum = 0;

        // Single line comment.
        for (int i = 0; i < values.Length; i++)
        {
            switch (i)
            {
                case 3: return;
                default:
                    sum += i > 2 ? 0 : 1;
                    break;
            }
        }

        i += (int)MyProperty; // No space after a type cast.
    }
}
```

Convenciones para la definición de nombres

Utiliza *PascalCase* para todos los espacios de nombres, nombre de tipos e identificadores miembro (es decir, métodos, propiedades, constantes, eventos), excepto para campos privados:

```
namespace ExampleProject
{
    public class PlayerCharacter
    {
        public const float DefaultSpeed = 10f;

        public float CurrentSpeed { get; set; }

        protected int HitPoints;

        private void CalculateWeaponDamage()
        {
        }
    }
}
```

Utiliza *camelCase* para todos los demás identificadores (como ser variables locales, argumentos de métodos), y utiliza guión bajo (`_`) como prefijo para los campos privados (pero no para los métodos o propiedades, como se explicó anteriormente):

```
private Vector3 _aimingAt; // Use a `_` prefix for private fields.

private void Attack(float attackStrength)
{
    Enemy targetFound = FindTarget(_aimingAt);

    targetFound?.Hit(attackStrength);
}
```

Hay una excepción con los acrónimos que consiste en dos letras como `UI`, que debe escribirse en mayúsculas cuando se usa en PascalCase, y en minúscula en caso contrario.

Tenga en cuenta que ``id``**no** es un acrónimo, por lo que debe tratarse como un identificador normal:

```
public string Id { get; }

public UIManager UI
{
    get { return uiManager; }
}
```

En general, se desaconseja utilizar un nombre de tipo como prefijo de un identificador como `string strText` o `float fPower`, por ejemplo. Sin embargo, hay una excepción acerca de las interfaces, en cuyo caso **deberían** ser nombradas usando una `I` mayúscula de prefijo, como `IInventoryHolder` o `IDamageable`.

Por último, considera elegir nombres descriptivos y no trates de acortarlos demasiado si eso afecta la legibilidad.

Por ejemplo, si quieres escribir código para encontrar un enemigo cercano y golpearlo con un arma, se prefiere:

```
FindNearbyEnemy()?.Damage(weaponDamage);
```

En lugar de:

```
FindNode()?.Change(wpnDmg);
```

Varaibles miembro

No declares variables miembro si sólo serán usadas localmente en un método, esto hace el código más difícil de seguir. En lugar de esto declaralas como varaibles locales en el cuerpo del método.

Variables locales

Declara las variables locales tan cerca como puedas de su primer uso. Esto hace que sea más fácil de seguir el código sin tener que desplazar el código demasiado para encontrar dónde fue declarada la variable.

Variables locales tipadas implícitamente

Considera usar tipado implícito (`var`) para la declaración de variables locales, pero hazlo **sólo cuando el tipo sea evidente** desde el lado derecho de la asignación:

```
// You can use `var` for these cases:

var direction = new Vector2(1, 0);

var value = (int)speed;

var text = "Some value";

for (var i = 0; i < 10; i++)
{
}

// But not for these:

var value = GetValue();

var velocity = direction * 1.5;

// It's generally a better idea to use explicit typing for numeric values, especially
with
// the existence of the `real_t` alias in Godot, which can either be double or float
// depending on the build configuration.

var value = 1.5;
```

Otras consideraciones

- Usa modificadores de acceso explícito.
- Use propiedades en lugar de campos no privados.
- Use modificadores en este orden: `public` / `protected` / `private` / `internal` / `virtual` / `override` / `abstract` / `new` / `static` / `readonly`
- Evita el uso de nombres completamente calificados o prefijos `this.` para miembros cuando no sea necesario.
- Elimina las sentencias `using` no utilizadas y paréntesis innecesarios.
- Considera omitir el valor inicial predeterminado para un tipo.
- Considere usar operadores condicion nula(null-conditional) o inicializadores de tipo(type initializers) para hacer el código más compacto.
- Usa un método seguro cuando exista la posibilidad de que el valor sea de otro tipo, y usa el método directo en caso contrario.