Guías de estilo de código

Introducción

En el desarrollo de software, la consistencia y claridad del código fuente son cruciales para la mantenibilidad y legibilidad del proyecto. Mantener un estilo de codificación uniforme se vuelve esencial para facilitar la colaboración entre los desarrolladores y asegurar que el código sea accesible y comprensible para todos los miembros del equipo.

Para lograr este objetivo, hemos adoptado `clang-format` como nuestra herramienta estándar para aplicar de manera automática y eficiente el formato a todos los archivos de código que creemos. `Clang-format` es una poderosa herramienta de formateo automático que nos permite definir y mantener un estilo de código coherente, reduciendo la carga cognitiva al revisar el código y permitiéndonos concentrarnos en la lógica y la funcionalidad en lugar de en aspectos estilísticos.

La elección de `clang-format` se debe a su versatilidad y facilidad de integración con la mayoría de los entornos de desarrollo, así como su capacidad para personalizar estilos de formateo para ajustarse a nuestras necesidades específicas. Mediante la definición de un archivo de configuración `.clang-format` en la raíz de nuestro proyecto, podemos especificar detalladamente las reglas de estilo que se aplicarán automáticamente a nuestro código.

Se ha usado la siguiente página web para crear un archivo de configuración: https://clang-format-configurator.site/

En ella, se puede modificar los parámetros de la columna izquierda para conseguir el resultado deseado especificado en el ejemplo de código de la columna derecha con la posibilidad de ver la diferencias ocasionadas entre la selección anterior y la actual.

Finalmente, luego de exportar el archivo de configuración, se importará a Visual Studio 2022 y para aplicar el formato se hará **`CTRL+A`** (seleccionar todo el código) y **`CTRL+K, CTRL+D`**.

Estilo de código

Se ha optado por usar el estilo de formateo de código de `Kernighan & Ritchie` en su mayor parte.

```
while (x == y) {
    something();
    something_else();
}
```

A continuación, se destacarán una serie de especificaciones sobre el estilo:

1. **Uso de macros (en bloques separados por una línea vacía) y de namespaces:** `#pragma once`, `#include`, `#define`, `#ifdef-#endif`, y `namespace std`, `namespace OGRE`...
   Minimizar la inclusión de .h en otros .h, hacer forward declaration (class Object;) siempre que se pueda.

   ** No es posible hacer forward declaration si:
   - La variable de tipo Object no es un puntero
   - Se usan los métodos de la clase Object en el .h
   - La clase Object es una template (por ejemplo, std::vector<>)
   - La variable un typedef (por ejemplo, std::string es un typedef de std::basic_string<char>)

2. **Uso de comentarios en ESPAÑOL y código en INGLÉS:**

```
/*
* COMENTARIOS EN ESPAÑOL, CÓDIGO EN INGLÉS
* ¿Cómo funciona esta clase?
* Hace esto, lo otro...
*/
class ClassExample : public E, public F, public G {
...
```

3. **Formato de una clase (con herencia):**

```
class ClassExample : public E, public F, public G {
private:
protected:
public:
ClassExample(); // Constructora
~ClassExample(); // Destructora

void otherFunction(); // Otras funciones
};
```

4. **Máxima longitud de las líneas:** 120 caracteres.
5. **Máxima longitud de las clases y/o funciones:** ilimitadas.

6. **Formato en el nombre de…**

   - Clases: `class ClassName`
   - Funciones: `functionName()`
   - Constantes: `SHORT_NAME`
   - Variables públicas y privadas: `varName`

7. **Espaciado**

   - Comas: `a, a`
   - Operadores: `a + a`
   - Funciones: `a()`
   - Bloques de control: `if (a == b)`

8. **Identación al dividir en diferentes líneas:**

```
int myFunction(int aaaaaaaaaaaaa, int bbbbbbbbbbbbbbbbbbbbbb,
               int ccccccccccccc, int d, int e);
```

9. **Indentación en bloques switch-case:**

```
switch (e) {
case 1: return e;
case 3: a = e; break;
case 4:
  a = d;
  // Más código...
break;
default: a = 1; break;
};
```

10. **Indentación en lambdas:**

```
[]() -> void {
    // Código
}
```

11. **Definiciones del mismo tipo de variable en diferentes líneas:**

```
int a = 1; // Mismo tipo de variable
int longComplicatedName = 4; // en diferentes líneas
```

12. **Definiciones de Getters-Setters y de funciones cortas de una línea en el archivo .h:**

```
// Getters-Setters y funciones cortas son definidas en .h
inline int getA() const { return a; }
inline void setA(int a) { a = a; } // this.a = a;
inline float magnitude(Vector3 const& a) { return sqrt(v.x *
v.x + v.y * v.y + v.z * v.z); }
```

13. **Comprobación de punteros nulos siempre igualando a `nullptr`**
14. **Uso de `override` cuando se pueda**
15. **Uso de `virtual` únicamente en la clase base**
16. **Uso de `auto` únicamente con iteradores y en recorridos con `for`:**

```
for (auto it = inputVector.begin(); it != inputVector.end();
it++) // ...

for (auto it : inputVector) // ...
```

17. **Uso de `if-else` en una sola línea sin llaves:**

```
if (a) anotherVariable = "baz";
else anotherVariable = "bar";
```