**FORMATO DE ESTILO DE PROGRAMACIÓN EN LENGUAJE C++ PARA PROYECTO “*HELLRAISER ENGINE*”.**

**Formato de estilo de código.**

1. **ELEMENTOS BÁSICOS.**
   1. **Indentación**.

El método de Indentación es simple:

* + - Siempre se utilizan dos espacios para indentar declaraciones en C++, excepto para los accesores de clases (**public**, **protected**, **private**) para las cuales se utiliza un espacio.

1. **ARCHIVOS DE ENCABEZADO.**
   1. **Directiva** **#pragma**.

Todos los archivos de encabezados inician con la declaración #pragma once, como alternativa a la macro de protección **#ifndef**.

* 1. **Declaraciones delanteras vs #include**.

Para evitar errores de compilación causados por dependencias circulares y para tener un mejor control del código, se hace uso de declaraciones delanteras, incluyendo solamente el archivo necesario en el archivo fuente (cpp).

* 1. **Orden de inclusión de archivos de encabezados.**

Todos los archivos de encabezados se incluyen en un debido orden:

* + - *Archivos de lenguaje C.*
    - *Archivos de lenguaje C++.*
    - *Archivos de encabezado de terceros.*
    - *Archivos de encabezado propios.*

1. **ÁMBITOS.**
   1. **Espacios de nombres.**

Se debe de evitar usar un espacio de nombres completo en cualquier archivo, invocando solamente mediante la palabra reservada **using**, los elementos del espacio de nombres que se utilizaran.

* 1. **Clases anidadas.**

Evitar clases anidadas, a menos que se tenga un buen control del ámbito de sus miembros y llamadas.

* 1. **Funciones globales, no-miembros y estáticas.**

Evitar las funciones globales para mejorar la jerarquía y el control del código. Usar funciones estáticas o funciones no-miembro (En ámbito de espacio de nombre).

* 1. **Variables locales.**

Se inicializan en su declaración y, en caso de objetos, controlar constructores y destructores.

* 1. **Variables globales y estáticas.**

Evitar uso de alguna de las dos. Pueden causar bugs.

1. **CLASES.**
   1. **Estructuras y clases.**

La estructuras se usan para contenedores de datos (No clases base o interfaces) de las cuales no se usaran instancias, e incluso que contengan funciones estáticas, y se inicializan directamente, no por medio de métodos. Las clases se utilizan de dos maneras: para clases base/interfaces o para uso de miembros privados.

* 1. **Constructores (Por defecto, explícitos, de copia y de movimiento).**

Es preferible utilizar métodos **init()** a constructores, para poder controlar errores por medio de excepciones. Constructores solo se utilizarían para facilitar la escritura.

Todas las clases DEBEN de tener un constructor por defecto (vacío), para evitar errores de definición de miembros y operaciones de memoria dinámica (ej. Cuando se crea una instancia nueva con **new** sin argumentos). Tambien debe de ser definido en caso que la clase no tenga otros constructores. Solo se exceptúa su uso con clases derivadas que no contienen más miembros que los declarados en la clase base.

Los constructores explícitos, que utilizan la palabra reservada **explicit**, solo se usan en constructores con un solo argumento para evitar conversiones y reducir escritura de código.

Los constructores de copia no son tan necesarios pues el compilador de C++ ejecuta una copia de manera implícita.

Debido a la utilidad delos constructores de movimiento, todas las clases lo pueden y deben llevar. En caso de las clases con miembros de acceso private, implementar dichos miembros como punteros inteligentes (C++ 11).

* 1. **Herencia única y multiple.**

**Herencia única.**

Se hace uso de herencias únicas para clases derivadas (Ej. Objetos del editor que derivan de un objeto de juego). Tambien se manejan composiciones y funciones amigas para clases que no dependen entre si y no comparten atributos.

**Herencia multiple**.

No se hace uso frecuente de herencias múltiples, teniendo como alternativa la composición y amistad entre clases y funciones respectivamente.

* 1. Sobrecarga de operadores.

No se recomienda la sobrecarga de algunos operadores matemáticos, puesto que también tienen uso con apuntadores. Se recomienda que los operadores hagan llamada a métodos de la clase que se encarguen de llevar a cabo la operación, para mantener una encapsulación.

* 1. Control de accesos.

Todos los miembros de una clase tienen acceso privado, otorgando acceso por medio de métodos de tipo **m\_SetVar( )** o **m\_GetVar( )**. Los miembros variables de una clase se denotan por la abreviación “a\_” (atributo). En cambio todos los miembros de una estructura tienen acceso público.

* 1. Orden de declaraciones.

Existe un orden de declaraciones de miembros para las clases y estructuras dentro de los archivos de encabezado. El orden y jerarquía es la siguiente:

* + - Acceso público.
    - Acceso protegido.
    - Acceso privado.
      1. Typedefs y enumeradores.
      2. Constantes estáticas.
      3. Constructores.
      4. Destructor.
      5. Miembros variables.
      6. Funciones miembros y estáticas.
    - Funciones **friend** son de acceso private.
    - Definiciones de funciones en archivos .cpp y definiciones no muy largas.

1. **APUNTADORES.**

El uso de “raw pointers” para objetos independientes de las clases a las que pertenecen (Ej. Una clase que contiene un apuntador de otra clase, pero que su ciclo de vida no depende de la clase que la contiene.)

* 1. Apuntadores inteligentes.

El uso de los “smart pointers” depende del tipo de smart pointer utilizado:

* + - **unique\_ptr**.

Se utilizan punteros únicos cuando la clase a la que pertenece el puntero tiene el control de su ciclo de vida

**Formato de estilo de comentarios, nomenclatura de variables y formato de texto.**

1. **NOMENCLATURA.**
   1. Nombre de variables y abreviaciones.
      * Todos los miembros de datos (variables) son precedidos por la abreviatura “m\_” de *miembro*. Los nombres son concisos y tienen un límite de 50 caracteres como nombre.
      * Todos las funciones miembros no tienen abreviatura, solo nombre. Funciones globales son precedidas por “*g\_*”de global. Los nombres de las funciones son concisos y tienen un límite de 50 caracteres.
      * Todas las clases son precedidas por la abreviatura “C\_” de *clase* y tiene un límite de 40 caracteres como nombre.
      * Todas las estructuras son precedidas por la abreviatura “S\_” de *struct/estructura*. Las estructuras tienen un límite de 40 caracteres de nombre.
      * Enumeradores son precedidos por la abreviatura “E\_” de *enumerador*.
   2. Archivos.

Todos los archivos (encabezado, fuente, imágenes, etc.) son precedidos por la abreviatura “*HR\_*”. Para el caso de archivos de Visual Studio, seguido de la abreviatura va el nombre del proyecto al que pertenece, seguido del nombre del archivo (NO DEBE SUPERAR LOS 80 CARACTERES). Para evitar errores, los archivos SIEMPRE utilizan la separación por guiones bajos “\_”.

* 1. Tipos.
  2. Macros, espacios de nombres, constantes, funciones y enumeradores.

1. **COMENTARIOS.**
   1. Estilo general.

El estilo de comentarios del motor es el formato DOXYGEN. Todo bloque de código está escrito como: “**/\*! \*/**” Y todos contienen el comando de “brief” de DOXYGEN. Las líneas de código están escritas de la siguiente manera: “**//!** ”.

* 1. Clases y estructuras.

Los bloques de código de las clases contienen los comandos “class” y el nivel de acceso que tiene “public”, “protected” y “private”.

Los enumeradores contienen, en sus bloques de comentarios, el comando “enum”. Los elementos del enumerador se comentan dentro de su ámbito con una simple línea.

* 1. Funciones.

Las funciones contienen dentro de su bloque de comentarios los comandos “param”, “exception” y “return”. El comando “param” se repite dependiendo el número de parámetros que entren en la función (Lo mismo ocurre con el comando “exception”).

* 1. Variables.

1. **FORMATO DE PROGRAMACIÓN.**
   1. Formato de funciones (Declaración y definición).
   2. Apuntadores.
   3. Condicionales.
   4. Expresiones booleanas.
   5. Clases.
   6. Variables.
   7. Espacios en blanco.