Documento de estructura de código para Black Engine.

Por Carlos Eduardo Ortiz Pizaña.

**Variables:**

Para las variables se utilizará notación húngara. En las variables globales, y miembras de clase se usará el prefijo 'g' y 'm' seguido de un '\_'. Después, y antes del nombre de la variable se pone la notación hungara para el tipo de variable, f, i, b, etc.

global int g\_iGlobalExample;

class MyClass : public CGameObject

{

public:

int m\_iExample;

float m\_fExample;

}

Para variables locales, la primer letra de cada palabra nueva, va con mayúsculas, la primera letra de la variable va en minúsculas.

Lleva notación húngara dependiendo del tipo de variable que sea.

void GetKnowledge()

{

int iVar;

bool bVar;

}

**Funciones:**

En las funciones todas las primeras letras de su nombre van en mayúsculas. Las variables de los parámetros, deben ser cortas y que tengan sentido con lo que piden. En caso de no recibir parámetros, se dejarán los paréntesis vacíos, sin "void". Hacer templates cuando se vea necesaria la posibilidad de usarla para diferentes objetos.

Siempre que se repita algo de código, hacer una función que se encargue de eso, para no repetir lo mismo muchas veces, y mantener el código más ordenado y legible.

template <typename T>

std::list GetObjectsInArea(int Radio, T\* Object)

{

//\*\*

HARDCORE CODING GOING ON.

..\*//

}

**Tamaño de linea:**

Siguiendo el límite estandard de c++, las lineas no deberán tener más de 80 caracteres. Esto para mantener el código legible.

**Headers:**

Antes que nada en todos los archivos .h, debe haber un #pragma once. Esto para reducir el tiempo de compilación, y evitar errores.

Ya que sólo lee una vez el archivo que se quiera incluir. En los .h sólo se implementan funciones que no tomen más de 3 lineas, y que sea muy sencillo. Como regresar un valor, o asignar un valor a una variable. Y si no hace que se extienda mucho la linea, se hace en la misma.

No usar #includes si lo que se busca ya está en la cabecera precompilada, o ya se ha usado en un archivo que ya se ha incluido.

class.h

Class CMyClass

{

int GetVar() { return m\_iVar; }

//important variable.

int m\_iVar;

}

**CPP's:**

En los CPP's, lo primero que debe haber es la cabecera precompilada, si es que hay una. También es necesario incluir el .h al que pertenece el CPP, si es que lo hay. Aquí es donde se hacen todas las implementaciones de funciones mayores a 3 lineas de código.

**Posición de llaves {}:**

El inicio de las llaves va una linea abajo de la función, clase o cualquier cosa que haya desencadenado la acción de abrir éstas.

El cierre de las llaves es una linea abajo de, la última linea que necesitaba estar dentro de ese scope.

struct MyStruct

{

//Do something.

}

**Orden de los archivos de inclusión:**

En los archivos .h van en el siguiente orden de inclusión los archivos:

1. librerias de C.

2. librerias de C++.

3. librerias externas.

4. archivos de nuestro programa.

En los archivos .cpp están estructurados de la siguiente manera:

1. Cabecera precompilada.

2. archivo .h al que pertenece.

3. librerias de C.

4. librerias de C++.

5. librerias externas.

6. archivos de nuestro programa.

**Clases:**

las clases siempre empiezan con una 'C' mayúscula, de clase. Y luego la primer letra del nombre en mayúscula, así como cualquier otra primer letra de cada palabra que se pueda arreglar. si hereda de una clase, va en la misma linea que el nombre de ésta.

Se usan para cualquier objeto nuevo que se necesite implementación de funciones y variables miembras. Procurar no manejar en una clase más de un objeto, para mantener el orden y la lógica.

**Estructuras:**

Las struct se usan cuando sólo se deben de almacenar pocos datos y no tenga funciones. En cuanto a la nomenclatura tiene el nombre de la estructura con cada primer letra de cada palabra que se agregue en mayúsculas.

**Comentarios:**

Todos los comentarios descriptivos de una función o variable, debe de ir en la linea anterior a la del código por comentar. Todas las variables, a parte de tener un nombre descriptivo, deben llevar un comentario de para que se utilizan. Las funciones llevan una breve descripción de lo que hacen, y que son los parámetros que pide. Todos estos elementos de la función deberán de igual manera tener un nombre que describa bien lo que hace.

**Operadores:**

Se sobrecargan los operadores aritméticos básicos, como la suma, resta, multiplicación, y división. También los signos <, >, >=. <=, ==, con el fin de hacer comparaciones menos largas, y que se entienda mejor el código.