Code Guidelines

New coding guidelines have been applied in the big refactor. If you want your code to be consistent you should follow up these simple rules.

# Members

No prefix is needed in front of a new member.

Window\* window;

Members are always written in lower camel case and spacing is defined by an underscore

float my\_new\_float\_value;

When referring to a member inside the source file always use “this->” in front of it. Except when you find yourself in the initializer list.

Example()

:my\_new\_float\_value(0.0f)

{

float value = this->my\_new\_float\_value;

}

# Locals

No prefix is needed in front of a new local variable.

float value;

Local variables are always written in lower case and spacing is defined by an underscore

float my\_new\_float\_value;

# Parameters

No prefix is needed in front of a new parameter.

float value;

Parameters are always written in lower camel case

Example(float myNewValue);

# Methods

Methods are always defined in lower camel case.

void thisIsSomeMethod();

# Classes

Classes are always defined in upper camel case.

class ThisIsMyNewClass

# Enumerations

Enumerations are always defined in upper camel case and using the new syntax of class enumerations

enum class MyEnumeration

Enumeration items are always defined in capital letters

enum class LogType

{

INFO\_LOG,

WARNING\_LOG,

ERROR\_LOG,

TODO\_LOG

};

# Selections

When writing an if-statement we will drop the “ { “ and “ } “ when there is only one statement underneath the if. Same will be done for an else. Also known as the simple statement.

if (statement)

return true;

if (statement)

{

// some code ...

// some more code ...

return true;

}

Whenever we would right an if/else-statement we will always write it in such a way that the code can early out as soon as possible. For example:

TransformComponent\* object\_transform = object->getComponent<TransformComponent>();

if (object\_transform != nullptr)

{

// Do some stuff …

// Do some other stuff …

}

TransformComponent\* object\_transform = object->getComponent<TransformComponent>();

if (object\_transform == nullptr)

return nullptr;

# Iterations

Iterations will have the same functionality as if statements when writing a “ for–loop “, “ while–loop “ or a “ do-while-loop “ having only one statement inside the loop will result in removing the “ { “ and the “ } “

for (GameObject\* object : this->vec\_objects)

object->render();

for (GameObject\* object : this->vec\_objects)

{

if (object->getInitialized())

continue;

object->setupInput(input);

if (!object->initialize())

return false;

}

Whenever we would right an iteration we will always write it in such a way that the code can early out as soon as possible. But we should try to avoid the “ break “ keyword. For example:

for (Light\* light : lights)

{

if (light->getLight()->getType() != LightType::Directional)

{

// Do some stuff ...

// Do some other stuff ...

}

}

for (Light\* light : lights)

{

if (light->getLight()->getType() == LightType::Directional)

continue;

}