Class Organization

Classes should be organized with the reader in mind rather than the writer. Since most readers are going to be using the public interface of the class, that should be declared first, followed by the class's protected members, and finally the class's private members. Each section should also be organized in alphabetical order, for a similar reason.

Naming Conventions

- Class methods and variables should use PascalCasing.
- Use camelCasing for method arguments and local variables.
- Member variables in a class should be prefixed by 'm_' to indicate that it is a member variable.
- Template classes should be prefixed by 'T'
- Enums should be prefixed by 'E'

Variable, method and class names should be clear, unambiguous and descriptive. Avoid over-abbreviation.

All functions that return a bool should ask a true/false question.

Comments

Comments are communication and communication is vital. Whilst you don't need to comment every line of code, use them when needed to help readers understand what is being done.

- Place the comment on a separate line, not at the end of a line of code.
- Begin comment text with an uppercase letter.
- End comment text with a period.
- Insert one space between the comment delimiter (//) and the comment text.

Guidelines

- Write self-documenting code
- Comments should be useful and detailed
- Don't comment bad code, rewrite it
- Don't contradict code

Const Correctness

Const is documentation as much as it is a compiler directive, so all code should strive to be const-correct.

This includes:

- Passing function arguments by const pointer or reference if those arguments are not intended to be modified by the function.
- Flagging methods as const if they do not modify the object
- Use const iteration over containers if the loop isn't intended to modify
- Const should be preferred for function parameters and locals. This shows the reader that the variable will not be modified.

Class Formatting

Each class and function should have a comment to explain to the reader what they do.

Class

A class comment should include a description of the problem this class solves.

Function

A function comment should include a description of what problem the function solves, if the function is a return type it should explain what variable will be returned (prefixed by @return), any additional information the writer seems relevant.

Modern C++ Language Syntax

The 'auto' Keyword

You shouldn't use auto in C++ code, except for a few exceptions listed below. Always be explicit about the type you're initializing. This means that the type must be plainly visible to the reader.

When is it acceptable to use auto?

- When you need to bind a lambda to a variable, as lambda types are not expressible in code.
- For iterator variables, but only where the iterator's type is verbose and would impair readability.
- In template code, where the type of an expression cannot easily be discerned. This is an advanced case.

Range-Based for

This is preferred to keep the code easier to understand and more maintainable.

Initialization

If a variable is initialised by the constructor then use an Initializer List to assign the values, rather than inside the body of the constructor. If you know the value of a variable as runtime, initialize the variable in the header file.

Code Formatting

Braces

Braces should be placed on a new line for function, if statements, switch statements, etc. Always include braces in single-statement blocks, this is to prevent editing mistakes.

If - Else

Each block of execution in an if-statement should be in braces. This is to prevent editing mistakes.

Tabs and Indenting

- Indent code by execution block
- Use tabs, not spaces.

Switch Statements

Except for empty cases, switch case statements should explicitly label that a case falls through to the next case. This means that each populated case should either include break, return, or include a falls-through comment.

General Style

- Minimize dependency distance. When code depends on a variable having a certain value, try to set the variables value right before using it. Initializing a variable at the top of an execution block and not using it for a few dozen lines of code, gives a lot of space for someone to accidentally change the value without realizing the dependency.
- Local variables should be initialized when they are defined.
- Split methods into sub-methods if possible.It is easier to understand a few simple methods, rather than a large chunk of code.
- In function declarations or function call sits, do not add a space between the function's name and the parentheses that preceded the argument list.
- Address compiler warnings. Compiler warning messages mean something is wrong, or not optimized. Fix what the compiler is warning you about.
- Avoid repeating the same code multiple times, more duplicate code into a function.
- Use intermediate variables to simplify complicated expressions. If you have a complicated expression, it can be easier to understand if you split it into sub-expressions, that are assigned to intermediate variables, with names describing the meaning of the sub-expression.
- Pointers and references should only have one space, which is to the right of the pointer or reference. Example: Player* m_Player;
- Avoid using pointers, instead use smart pointers.
- Avoid using anonymous literals in function calls. Use named constants which describe their meaning. This makes the intent of the function more obvious to the reader.
- Use a try-catch statement for exception handling.