Jason's C++ Style Guide

Updated: 1/8/2020

Most development companies (at least those that I've worked for) have some kind of style guide. Sometimes it's just suggestions – other times, your code will be rejected by supervisors if it doesn't adhere to their style. To new programmers, worrying about style seems pointless (admittedly, the style almost never changes the resulting executable). But…it does make navigating the source code much easier.

On some of these points, I'm willing to negotiate. Others, you'll just be stuck with 😊*. But make sure you stick to what we end up here!*

1. Curly braces

a. no hanging indents. Plus, tab everything under the block.

Bad

```
void func(int x) {
   // do stuff
}
```

Good

```
void func(int x)
{
   // do stuff
}
```

Why? I think the initial brace gets lost on long lines. Keep in mind, ~50% of the C++ community disagrees with me.

b. On one-line blocks, curly braces are option.

```
for (int i = 0; i < n; i++)
        cout << "I is " << i << endl;
```

2. Classes

a. **One class per .h** (and .cpp) file unless there is a very good reason not to (e.g. templates)

b. **Naming conventions**:
   i. Name the file the same as the class, but all lower-case, and with underscores (e.g. my_data_descriptor.h and my_data_descriptor.cpp)
   ii. Class names are camel-case, initial cap (e.g. class MyDataDescriptor)
   iii. Attribute names are camel-case with an initial m (e.g. mIndexPosition)
   iv. Static attributes are similar, but with an ms prefix.
   v. Method names are all lower-case, with separating underscore (e.g. void find_val)

c. **Order of methods**. Put all members into groups, in this order, with the access-modifier repeated for each (even if not necessary from an above section
   i. Enum classes
   ii. Nested classes
   iii. Attributes
   iv. Constructors / destructor
   v. Getters / setters
   vi. Other methods (these can be broken out for long classes)

d. **Cpp/h-split**.

i. If possible (templates are one place it's not), separate the class into a .h and a .cpp file.
ii. For any non-trivial (1-2 statements) method, put the body in the .cpp

e. Documentation

i. Use doxygen-style comments (/// for line comments or /** … */ for block) in these places
    1. Above the class prototype in the .h file. Indicate what the class is and how it is used.
    2. Above each attribute. Try to keep these to ~ 1 sentence.
    3. Above each method.
ii. Make the doxygen comments descriptive – your audience is someone that's familiar with C++, but not the use of our engine.
iii. Use normal comments within your implementation to provide insight to understanding difficult sections of code. Don't document obvious statements (like adding to a variable or reading a line of text).

f. Minimize unnecessary stuff

i. Only make a variable an attribute if you need it in multiple places (and won't be re-setting it)
    1. The name of a game object makes sense – we'll likely use that multiple places.
    2. A counter variable, like i, doesn't make sense. It is used multiple places, but we are re-setting it each time. Just make this a temporary.
ii. When including header files (either from a .h or a .cpp), only include those .h files that you need.
    1. This can help avoid circulare include-dependencies
    2. It's also helps readability.
    3. Remember: sometimes a forward declaration is enough – you only need the .h if you're going to actually use a class (not just declare pointers to it).

3. Readability Aids

a. Put a few lines of white space between function bodies in a .cpp file.
b. Within a function, put a line of white-space to distinguish related chunks of code.

4. Use dOxygen-style documentation comments.

a. Two major methods:

```
/**
First line of documentation
Second line of documentation*/
[thing I want to document]

/// This is a brief documentation – one line only

/// first line of documentation – this one is a in-depth documentation
/// second line of documentation – has to be more than one line.
```

b. Here's the minimal amount of stuff to document:
    i. The namespace – this appears before the namespace (in any file). It becomes the main documentation page
    ii. A class – put it in the .h, before the class

iii. A member (method or attribute) – put it before the thing.

c. After we've written some code (remind me before we start lab2), I'll show you how to generate the documentation.