Project Guidelines

Remus Africanus

University of North Southern

1. Use of C++11 (and later) features

As of yet, we have used:

inline (C++17)

The *clip_rect* function in the Collidable class is noted with this keyword.

auto (C++11)

Certainly saves a good deal of typing, at the expense of clarity. An example appears at the start of the *clip_dir* function definition in the file *Collidable.cpp*, where the local variables *isect* and *center* are declared and initialized.

enum class (C++11)

Used in the Game class for the various game states.

nullptr (C++11)

Provides a type-safe value to assign to unused pointers.

Range-based for loop (C++11)

Line 150 in *game.cpp*

2. STL Containers

Vectors to hold players' and the level tiles' sprite maps, amongst other things, of course.

Dynamic sizing provided by vectors enabled convenient addition of new assets as the project progressed, such as inserting the second player character's sprite map once it was finally ready to go.

3. Use of Namespaces

We have one major namespace, **fxx** which encompasses all of the classes used in the game. This guards against potential name conflicts with 3rd party libraries (e.g. SFML). Each branch of the includes has its own respective namespace to further protect against name conflicts.

For example, "Game.h" and "Menu.h" are both under the "directors" directory. Both of these classes also fall under the **directors** namespace.

4. Exception Handling

A rather elementary example, but exception handling can be seen in the constructor of the menu class. The code which attempts to open a font file is encompassed in a try block, which upon failure will throw a C-string error message which is then caught by the following catch block. At this point in development, this is not believed to be a catastrophic error, so the program continues on after displaying an error message to be noted by the engineers.