Mercury Theme System Charles Lohr

Concept:

Mercury contains a Theme system. Its theme system is designed to hold variables specific to the game. Multiple themes can be loaded on top of each other and can only change certain portions of the game's variables. This theme system provides for multiple inheritance as well. This means, different sections of the theme can inherit all of the values from one or more other sections of the theme. Mercury's theme system is based around the metrics file, "metrics.ini." Mercury encourages use of a theme system for a number of reasons:

- Allows customizing and designing to be free of code

  Graphic artists and designers may not want to recompile a program every time they want to slide some image, or shape somewhere on the screen. The Theme system allows them to modify a logically laid out file to get what they want done, done.
- Allows for heavy code/screen re-use

  If you write a game where there are many different screens that all act very similar with subtle differences, like different maps, graphics, details about the game play, etc. Without creating a new screen that abstracts from the base one, you could use metrics to define new screens. This cuts down on code base size as well as allows for more scalability in the end product.
- Allows for themeing by third parties at a later time

  It is particularly useful for any program that is expected to have any decent sized following to provide the users with a non-destructive manner to develop new skins for the product. In this case, users can develop themes to modify sections of game play, and all of the themes can be overlaid (in an order of choice and priority) to the user's desire.

Implementation:

Mercury has this theme system in "MercuryTheme.h" it contains a global singleton called `THEME`. The following functions are accessible via `THEME`.

- `void ReloadAllThemes()` – Reload all themes in the order found in Mercury.ini, under section [Options], value Themes. The first theme is the primary in search order. The default theme should be last.
- `bool GetMetric(CString sKey, CString sValue, CString &sData)` – Get a specific metric value from metrics, it returns the data via sData, and returns true if the value was found, false otherwise.
- `CString GetMetricS( CString sKey, CString sValue, CString Default = "" )` – Get a specific value from metrics and return it as a string, using Default as the default return.
- `int GetMetricI( CString & sKey, CString sValue, int Default = 0 )` – Get a specific value from metrics and return it as a integer, using Default as the default return.
- `float GetMetricF( CString sKey, CString sValue, float Default = 0 )` – Get a specific value from metrics and return it as a float, using Default as the default return.

- `bool GetMetricB( CString sKey, CString sValue, bool Default = false )` – Get a specific value from metrics and return it as a boolean, using Default as the default return.
- `bool GetPathToFile( const CString & File, CString & Path )` – Get path to a file, in any theme. This will check to see if any of the themes have the file in order. If the file exists, it will put the path to the file (including the file) in Path and return true, else it will return false.
- `CString GetPathToGraphic( const CString &Name )` – Get the path to a specific graphic, check all themes for the graphic, and if none is found, return the path to the default missing image.

MercuryTheme also contains a macro for easy image loading. It checks metrics first to find the name of the image to load. This is useful so that themes can choose which image to put where and to allow for easy image re-use. This macro is called:

```
GET_GRAPHIC_BY_NAME( GraphicName )
```

It can be used whenever you want to load an image and allow themes to change what image to use. For example in our hello world program, instead of using a hard-coded name for the .png, we could make it depend on metrics by using the following line:

```
m_sprHello.LoadPNG( GET_GRAPHIC_BY_NAME( "HelloImage" ) );
```

It would look in metrics for what image to use. When it finds the following in metrics.ini:

```
[ScreenHelloWorld]
HelloImage=HelloWorld.png
```

It will output: “Themes/default/Graphics/HelloWorld.png”

All graphics should be held in the Graphics folder of a theme to help with organization.

This can also use falling back to obtain the graphic. A high level theme can tell the screen to use an image that it does not contain, but that exists in the default theme and Mercury will find it.

Using fallbacks:

If any one section utilizes many elements common to another it is permitted to fallback to another section by setting a Fallback value in the section. For example, if we had multiple Hello World screens, each used the same image, they could fall back to a single hello world base. You can see this below:

```
[ScreenHelloCommon]
HelloImage=HelloWorld.png

[ScreenHelloWorld1]
Fallback=ScreenHelloCommon
```

```
HelloText=This is the first set of values

[ScreenHelloWorld2]
HelloText=This is the second screen
Fallback=ScreenHelloCommon
```

It is also possible to support multiple fallbacks (multiple inheritance by separating each fallback section with a comma, like:

```
Fallback=ScreenHelloCommon,SomeOtherSectionWithStuffWeWant
```

When looking at a fallback class, it will go through each theme to try to find that given fallback, this means that you do not have to have a copy of `[ScreenHelloCommon]` locally if you wish to use the one in the default, or other theme.