Introduction to ImGui

1) Basics :

- The ImGui library can be used only within the **draw(pDevice)** function, or a function derived from this one.
- From a general point of view, the architecture of ImGui is based on a series of windows nested one inside the other. Each window can accommodate different widgets. The attributes of each element (window or widget) can be redefined in a particular or global way.
- The EOP documentation offers a section dedicated to ImGui (extra : LuaPlugin Gui docs) and its numerous functionalities (only the most essential elements will be covered here).

2) About windows :

- Even if you only want to add a button, you need to create a main window first (and make it invisible next if you wish).
- Being overlays, windows are interfering with the game. This means that even if you create an invisible window, you won't be able to interact with the game through it. On the same way, shortcuts will also be disabled when the game cursor is hovering a window.
- You can display several main windows at the same time. If they overlap, the last appeared will be on the foreground.

3) Creating the main window :

Now let's take the example of a script allowing to display a window when a character is selected, and close it when the action is repeated. This is how ImGui should be used : an event or an action changes the value of a variable, and the draw function then (dis)allow the display of your main window and its content :

```
local open = false

function onCharacterSelected(selectedChar)
    if (open == false) then open = true
        else open = false
    end
end

function draw(pDevice)
    if (open) then -- laconic form of (open == true)
        ImGui.SetNextWindowSize(600, 150)
        open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)

        ImGui.End()
    end
end
```

The block required to create the main window is this part :

```
open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)

ImGui.End()
```

1. First, you declare a variable (open) and give it true as value on a way or another.
2. Next, you use **ImGui.Begin(...)** with this same variable.
3. Finally, you close this window section with **ImGui.End()**.

Note 1 : The string used as a label for your window must be unique (this is true for all other elements). Otherwise, even with a different variable name for two windows (or buttons, etc...), only one of them would appear.

There are two ways to circumvent this problem : either you use ImGui.PushID (see the ID section) or you add an ID to the string to make it unique, with **##**, like this :

```
open, shouldDraw = ImGui.Begin("My Main Window##01", open, ImGuiWindowFlags.NoDecoration)
```

The part of the string starting with **##** will never appear, and you don't even need to have something before. This is particularly useful to have buttons without name displayed.

Note 2 : **ImGuiWindowFlags** accepts different parameters (NoMove, NoResize, etc.). Nevertheless, in the context of the game, you shouldn't need to use something else than **NoDecoration**, which will remove the title bar, the close button as well as the resize grip.

Note 3 : You can nevertheless use several flags at the same time with **bit.bor**, like this :

```
open, shouldDraw = ImGui.Begin("My Main Window", open, bit.bor(ImGuiWindowFlags.NoCollapse,
ImGuiWindowFlags.NoScrollbar))
```

Also, simply specifying a position or size for your window will prevent the player from moving or resizing it. And adding a custom button to close the window or anything else is so easy that you have no reason to keep the upper bar for this.

ImGuiWindowFlags :

NoTitleBar	NoSavedSettings	NoDecoration	AlwaysAutoResize
NoResize	NoInputs	MenuBar	AlwaysVerticalScrollbar
NoMove	NoScrollWithMouse	HorizontalScrollbar	AlwaysHorizontalScrollbar
NoScrollbar	NoFocusOnAppearing	ShowBorders	AlwaysUseWindowPadding
NoCollapse	NoBringToFrontOnFocus		

3) Windows attributes :

Commands affecting the attributes of a window (main or child) can be placed before (**SetNextWindow** commands) or after (**SetWindow** commands) its declaration.

In some cases, the use of **SetNextWindow** commands may be preferable. For example, if you define a position with this type of command, the player won't be able to move the window, while with the **SetWindow** equivalent, it will just restore the position once the window released. Moreover, having all this commands gathered outside will make scripts easier to read.

```
ImGui.SetNextWindowPos(100, 100)

open, shouldDraw = ImGui.Begin("My Main Window##01", open, ImGuiWindowFlags.NoDecoration)
     ImGui.SetWindowSize(600, 150)

ImGui.End()
```

Both these types of commands support optional conditions (**ImGuiCond**) : **Always**, **Once**, **FirstUseEver** and **Appearing**. Their effect is exactly what their name indicates. Unless you have specific needs, there is no reason to use them.

```
ImGui.SetNextWindowPos(300, 100, ImGuiCond.FirstUseEver)
```

Note : To determine the size of the user's screen, and so adapt dimensions and positions depending on it, see section **12**.

4) Creating child windows :

Once the main window created, instead of directly adding widgets within this one, you can also create child windows. They will allow you to add widgets in some particular areas more easily, as well as give this areas a different background color for example.

```
open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)

    local shouldDraw = ImGui.BeginChild("Child Window##A1", 200, 70, ImGuiWindowFlags.NoDecoration)
        local shouldDraw = ImGui.BeginChild("Child Window##A2", 50, 50, ImGuiWindowFlags.NoDecoration)

        ImGui.EndChild()
    ImGui.EndChild()

    local shouldDraw = ImGui.BeginChild("Child Window##B1", 200, 70, ImGuiWindowFlags.NoDecoration)

    ImGui.EndChild()
ImGui.End()
```

5) Adding Widgets

- **Regular Button** : By default, the state of this button type is false, and true when you click on it.

```
shouldDraw = ImGui.BeginChild("Child Window##A2", 500, 100, ImGuiWindowFlags.NoDecoration)
    local clicked = ImGui.Button("My Button", 100, 50)
    if (clicked == true) then print('Button clicked') end
ImGui.EndChild()
```

A shorter version where the widget is generated directly within the condition :

```
if (ImGui.Button("My Button", 100, 50) == true) then print('Button clicked') end
```

Or in the most laconic form (being the faster this one will be used in this tutorial) :

```
if (ImGui.Button("My Button", 100, 50)) then print('Button clicked') end
```

Note 1 : There are two other forms of regular button : **ImGui.SmallButton** and **ImGui.InvisibleButton**. The first one has just the size to display its label, and the second displays neither texture nor label.

Note 2 : The name of the variable (clicked) is not important in itself. You can have the same variable name for each button (and so use it in a loop), but any effect must then be placed immediately after. And as already said, the label must imperatively be unique.

Note 3 : Having their label centred by default, regular buttons with transparent textures (so not the invisible buttons) also offer an easy way to center the title of a text.

- **Arrow button** : This one work on the same way than regular buttons (true when clicked), you just have to specify a direction (Down, Up, Right or Left). It doesn't display any label, just a small arrow.

```
if (ImGui.ArrowButton("My Arrow", ImGuiDir.Down)) then print('Down') end
```

- **Checkbox** : Automatically (un)checked when you click on it. Only requires a variable for the option (declared outside the draw() function).

```
if (myOption) then myOption, pressed = ImGui.Checkbox("My Option", true)
    else myOption, pressed = ImGui.Checkbox("My Option", false) end
```

- **Radio button** : Two ways to use it. In the first case, there is only one button used on the same way than a checkbox.

```
local myOption = true

function draw(pDevice)
    if (open) then
        ImGui.SetNextWindowSize(1100, 150)
        open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)
           if (myOption) then pressed = ImGui.RadioButton("Click me", pressed == true)
             else pressed = ImGui.RadioButton("Click me", pressed == false)
           end
           if (pressed and myOption) then myOption = false
               elseif (pressed and myOption == false) then  myOption = true
           end
        ImGui.End()
    end
end
```

In the second case, we have different buttons, and only one can be activated at a time. Rather than a boolean, an integer is then used for the option. Simply clicking on a button will change the value of the variable.

```
local myOption = 1

function draw(pDevice)
    if (open) then
        ImGui.SetNextWindowSize(1100, 150)
        open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)

           myOption, pressed = ImGui.RadioButton("Alpha", myOption, 1)
           myOption, pressed = ImGui.RadioButton("Bravo", myOption, 2)
           myOption, pressed = ImGui.RadioButton("Charlie", myOption, 3)

        ImGui.End()
    end
end
```

- **Progress Bar** : Parameters are fraction, length, height and a displayed string (of course, it makes more sense to use a variable for the fraction).

```
ImGui.ProgressBar(0.5, 100, 25, "Loading Failed. Sike. - 50%")
```

Note 1 : You can use the fraction alone. In this case the length of the bar will be determined by the available space in the window, and the height by the font size.

Note 2 : When there is no string, this one is replaced by the percentage derived from the fraction (so "50 %" for 0,5).

- **Combo Box** : There are two ways to use it. The first version, the simplest, is to be used when the selection set is small (2 is the number of items in the table and 5 the max size in items of the pop up (optional) :

```
local current_item = 0

function draw(pDevice)
    if (open) then
        ImGui.SetNextWindowSize(500, 150)
        open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)
        current_item, clicked = ImGui.Combo("Label", current_item, {"Option 1 ", "Option 2"}, 2, 5)
        ImGui.End()
    end
end
```

```
local comboTab = {"Alpha", "Bravo", "Charlie", "Delta", "Echo", "Foxtrot", "Golf", "Hotel", "India"}
local preview = comboTab[1]

function draw(pDevice)
    if (open) then
        ImGui.SetNextWindowSize(500, 150)
        open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)

           if ImGui.BeginCombo("My Combo", preview, ImGuiComboFlags.HeightLargest) then
             for i = 1, #comboTab do
                if (ImGui.Selectable(comboTab[i])) then preview = comboTab[i] end
             end
             ImGui.EndCombo()
            end
        ImGui.End()
    end
end
```

ImGuiComboFlags :

HeightSmall	HeightRegular	HeightLarge	HeightLargest
HeightMask	PopupAlignLeft	NoArrowButton	NoPreview

ImGuiSelectableFlags :

DontClosePopups	SpanAllColumns	AllowDoubleClick	AllowItemOverlap
Disabled			

- **Slider :** There are different versions using floats, integers or angles, each version supporting multiple sliders and values (in a table). The example below is the only thing you could need in the context of the game.

```
local value = 3

function draw(pDevice)
    if (open) then
       ImGui.SetNextWindowSize(500, 150)
       open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)
           value, used = ImGui.SliderInt("My Slider", value, -10, 10)
       ImGui.End()
    end
end
```

- **Input with keyboard** : There are different versions working with texts, multiline texts, floats, multiple values (in a table). Versions using numbers will add plus and minus buttons.

```
local text = "Nevermind"
local value = 3

function draw(pDevice)
    if (open) then
        ImGui.SetNextWindowSize(600, 150)
        open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)
            text, selected = ImGui.InputTextWithHint("Label_1", "Hint", text, 100)
            value, used = ImGui.InputFloat("Label_2", value, 1, 10)
        ImGui.End()
    end
end
```

- **Tab Bar** : Tabs are automatically selected when you click on them. Used with a boolean, ImGui.BeginTabItem will add a button to close the tab.

```
function draw(pDevice)
    if (open) then
        ImGui.SetNextWindowSize(600, 150)
        open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)
            if (ImGui.BeginTabBar("My Tab Bar")) then
                if (ImGui.BeginTabItem("Alpha")) then
                    ImGui.Text("Nevermind 1")
                    ImGui.EndTabItem()
                end
                if (ImGui.BeginTabItem("Bravo")) then
                    ImGui.Text("Nevermind 2")
                    ImGui.EndTabItem()
                end
                if (ImGui.BeginTabItem("Charlie")) then
                    ImGui.Text("Nevermind 3")
                    ImGui.EndTabItem()
                end
            end

        ImGui.End()
    end
end
```

ImGuiTabBarFlags :

Reorderable	NoCloseWithMiddleMouseButton	FittingPolicyScroll	NoTooltip
AutoSelectNewTabs	NoTabListScrollingButtons	FittingPolicyMask	
TabListPopupButton	FittingPolicyResizeDown	FittingPolicyDefault	

ImGuiTabItemFlags :

UnsavedDocument	NoCloseWithMiddleMouseButton	NoTooltip	Leading
SetSelected	NoPushId	NoReorder	Trailing

- **Tree** :

```
function draw(pDevice)
    if (open) then
        ImGui.SetNextWindowSize(600, 150)
        open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)

            local x = ImGui.GetCursorPosX()
            if (ImGui.CollapsingHeader("Header 1")) then
                if (ImGui.TreeNode("Tree Node")) then
                    if (ImGui.Selectable("Alpha")) then print('Alpha') end
                    if (ImGui.Selectable("Bravo")) then print('Bravo') end
                    if (ImGui.Selectable("Charlie")) then print('Charlie') end
                end
            end
            ImGui.SetCursorPosX(x)
            if (ImGui.CollapsingHeader("Header 2")) then
                ImGui.SetNextItemOpen(true)
                if (ImGui.TreeNodeEx("Tree Node Ex", ImGuiTreeNodeFlags.Selected)) then
                    if (ImGui.Selectable("Delta")) then print('Delta') end
                end
            end

        ImGui.End()
    end
end
```

As you can see in the example above, the x coord of the first header is memorized to be restored before the second header (which otherwise would appear staggered when the above node is open).

Note : The state of **ImGui.SetNextItemOpen** is permanent, so it could be more interresting to use a variable instead of directly a boolean.

ImGuiTreeNodeFlags :

Selected	AllowItemOverlap	OpenOnArrow	SpanFullWidth
Framed	NoTreePushOnOpen	Bullet	NavLeftJumpsBackHere
DefaultOpen	NoAutoOpenOnLog	FramePadding	CollapsingHeader
Leaf	OpenOnDoubleClick	SpanAvailWidth	

- **List Box** : Two version depending of the number of items. The first one for a small set :

```
local current_item = 0

function draw(pDevice)
    if (open) then
        ImGui.SetNextWindowSize(500, 150)
        open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)

        current_item, clicked = ImGui.ListBox("Label", current_item, {"Item 1", "Item 2", 2}, 2)
        if (clicked) then print(current_item) end

        ImGui.End()
    end
end
```

Note : For the height_in_items (last number), if you use a number higher than reality you will see a "Missing" entry for each supernumerary point.

And the second one, probably more practical :

```
local current_item = 0
function draw(pDevice)
    if (open) then
        ImGui.SetNextWindowSize(600, 150)
        open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)
           if (ImGui.BeginListBox("My List Box", 100.0, 100.0)) then
                if (ImGui.Selectable("Alpha")) then print('Alpha') end
                if (ImGui.Selectable("Bravo")) then print('Bravo') end
                if (ImGui.Selectable("Charlie")) then print('Charlie') end
               ImGui.EndListBox()
           end
        ImGui.End()
    end
end
```

Note : When there is more items than place in the box, a vertical scrollbar appears. But if the items are too long, there is no horizontal scrollbar.

6) Layout

By default, the first element of a window appears in the top-left corner, while the next one will be automatically placed below. You can nevertheless circumvent this by using **ImGui.SameLine()** before each new element (child window or widget) :

```
local shouldDraw = ImGui.BeginChild("Child Window##A1", 200, 70, ImGuiWindowFlags.NoDecoration)
ImGui.EndChild() ImGui.SameLine()
local shouldDraw = ImGui.BeginChild("Child Window##A2", 200, 70, ImGuiWindowFlags.NoDecoration)
ImGui.EndChild()
```

Another way is to use dummies, which as invisible elements will create voids of the specified dimensions. They are thus particularly useful to keep the same space between different buttons.

```
if (ImGui.Button("Button 1", 100, 50)) then print('Button 1 clicked') end
    ImGui.Dummy(100, 25)
if (ImGui.Button("Button 2", 100, 50)) then print('Button 2 clicked') end
```

Finally, you can also directly define the coordinates of the cursor and thus place any element exactly where you want (the top-left corner of your element will then be positioned on the chosen coordinates).

As you can see in the below example, you can use absolute as well as relative coordinates :

```
if (ImGui.Button("Button 1", 100, 50)) then print('Button 1 clicked') end

ImGui.SetCursorPos(200, 50)
if (ImGui.Button("Button 2", 100, 50)) then print('Button 2 clicked') end

x, y = ImGui.GetCursorPos()
ImGui.SetCursorPos(x +75, y +25)
if (ImGui.Button("Button 3", 100, 50)) then print('Button 3 clicked') end
```

7) ID Stack / Scopes

If you want to use the same label for different elements of the same window, you can use **ImGui.PushID("myID")** and **ImGui.PopID()** to indirectly customize these elements.

If you have multiple elements requiring a similar label, this could be in some cases a faster solution than using **##**.

```
ImGui.PushID("ID1")
    if (ImGui.Button("My Button", 100, 50)) then print('Button 1 clicked') end
ImGui.PopID()

ImGui.PushID("ID2")
    if (ImGui.Button("My Button", 100, 50)) then print('Button 2 clicked') end
ImGui.PopID()
```

8) Colors

While it's possible to use a color for a particular string without affecting the other texts, changing the color of another element (e.g. a button) cannot be done individualy .

```
local color_r, color_g, color_b, color_a = ImGui.GetStyleColorVec4(ImGuiCol.Button)
if (ImGui.Button("Button 1", 100, 50)) then print('Button 1 clicked') end

ImGui.PushStyleColor(ImGuiCol.Button, 0.75, 0.25, 1, 1)
if (ImGui.Button("Button 2", 100, 50)) then print('Button 2 clicked') end

ImGui.PushStyleColor(ImGuiCol.Button, color_r, color_g, color_b, color_a)
if (ImGui.Button("Button 3", 100, 50)) then print('Button 3 clicked') end
```

This implies that if you want to change the color of only one element, you need to first memorize the current color, next use your custom color, and next restore the original one, like in the following example :

Names of elements for ImGuiCol (mainly used with ImGui.PushStyleColor)**:**

Text	TitleBgActive	Button	ResizeGripActive
TextDisabled	TitleBgCollapsed	ButtonHovered	CloseButton
WindowBg	MenuBarBg	ButtonActive	CloseButtonHovered
ChildWindowBg	ScrollbarBg	Header	CloseButtonActive
PopupBg	ScrollbarGrab	HeaderHovered	PlotLines
Border	ScrollbarGrabHovered	HeaderActive	PlotLinesHovered
BorderShadow	ScrollbarGrabActive	Separator	PlotHistogram
FrameBg	ComboBg	SeparatorHovered	PlotHistogramHovered
FrameBgHovered	CheckMark	SeparatorActive	TextSelectedBg
FrameBgActive	SliderGrab	ResizeGrip	ModalWindowDarkening
TitleBg	SliderGrabActive	ResizeGripHovered	

9) Images

ImGui also allows to display images as background for windows and widgets :

```
local open = false
local myImage = {x = 0, y = 0, img = nil}

function onCharacterSelected(selectedChar)
    if (open == false) then open = true
        else open = false
    end
    myImage.x, myImage.y, myImage.img = M2TWEOP.loadTexture(M2TWEOP.getModPath().."/youneuoy_Data/plugins/
images/myImage.dds")
end

function draw(pDevice)
    if (open) then
        ImGui.SetNextWindowSize(1000, 500)
        open, shouldDraw = ImGui.Begin("My Main Window", open, ImGuiWindowFlags.NoDecoration)

         local xPos, yPos = ImGui.GetCursorPos()
         ImGui.Image(myImage.img, 1000, 500)
         ImGui.SetCursorPos(xPos, yPos)

        ImGui.End()
    end
end
```

1. First you declare a table for each image you want to use, always with the same content :

```
local myImage = {x = 0, y = 0, img = nil}
```

2. Next, the keys of this table are redefined with the path of the image. This must be done before its display, but don't do it immediately after the declaration of the table (when files are loaded), otherwise you will get an error.

```
myImage.x, myImage.y, myImage.img = M2TWEOP.loadTexture(M2TWEOP.getModPath().."/youneuoy_Data/plugins/
images/myImage.dds")
```

As you can see, a custom folder ('images') is used for myImage. Here dds is used as format, but tga, png and jpg also work, and certainly some others.

3. Lastly, we just need to add the image like any other element :

```
local xPos, yPos = ImGui.GetCursorPos()
ImGui.Image(myImage.img, 1000, 500)
ImGui.SetCursorPos(xPos, yPos)
```

Here, it's just a background for the window, and the very first element, so the position of the cursor is not directly important.

Nevertheless, the image being itself an element, the cursor would be moved if we only used ImGui.Image (line 2). That's why we need to memorize the original cursor position (line 1), and restore it after the addition of the image (line 3).

10) Width

When a widget in itself doesn't directly allow to define its size, you can nevertheless change its width, like this :

```
ImGui.SetNextItemWidth(100)
value, used = ImGui.SliderInt("My Slider", value, -10, 10)
```

Or also on this way for more widgets :

```
ImGui.PushItemWidth(300)
    value1, used = ImGui.SliderInt("My Slider 1", value1, -10, 10)
    value2, used = ImGui.SliderInt("My Slider 2", value2, -10, 10)
ImGui.PopItemWidth()
```

Of course increasing the font size will also increase the heigth (but the sharpness will be lower).

11) Shortcuts :

It's also possible to reproduce the vanilla ShortcutTriggered on the following way :

```
function draw(pDevice)
    if (ImGui.IsKeyPressed(ImGuiKey.Backspace)) then
        print('Backspace pressed')
    end
end
```

Some of the keys supported by ImGuiKey :

Tab	DownArrow	End	Escape	LeftShift
LeftArrow	PageUp	Delete	GraveAccent	RightShift
RightArrow	PageDown	Backspace	LeftAlt	LeftCtrl
UpArrow	Home	Enter	RightAlt	RightCtrl

12) Size and position of items depending on display size :

First of all, you need to integrate the function and variables of the next page in your files. It will allow you to use two global variables **_wd** (for width) and **_hg** (for height), which you will add as multiplier coefficients to the dimensions and positions of your items.

To that end, you just have (if need be) to replace **1920** and **1080** with the dimensions you use when modding the UI.

For most of elements, you will simply use this variables like this :

```
ImGui.Image(scroll.img, 953*_wd, 823*_hg)
```

For windows though, the situation can be a little different since the size of their borders is a constant included in the window size. So, unless you want to show the borders, you will then proceed like this for a full screen image (where **8** is in fact 4+4, the addition of borders of both sides) :

```
ImGui.SetNextWindowSize((1920*_wd)+8, (1080*_hg)+8)
ImGui.SetNextWindowPos(-4, -4)
```

```
local ffi = require("ffi")
ffi.cdef [[
typedef long LONG;
typedef void* HANDLE;
typedef HANDLE HWND;
typedef struct RECT {
      LONG left;
      LONG top;
      LONG right;
      LONG bottom;
} RECT;
typedef int BOOL;
typedef RECT *LPRECT;
BOOL GetClientRect(HWND hWnd,LPRECT lpRect);
HWND GetActiveWindow(void);
]]

function onCampaignMapLoaded()
      local screenHeight, screenWidth = 0, 0
      local window = ffi.C.GetActiveWindow()
      local rect = ffi.new("RECT")
      ffi.C.GetClientRect(window, rect)
      screenWidth = rect.right-rect.left
      screenHeight = rect.bottom-rect.top
      _wd = screenWidth/1920 ; _hg = screenHeight/1080
end
```

13) Fonts :

Custom fonts simply requires to be defined as global variables in the onLoadingFonts() function (/event).

```
function onLoadingFonts()
     _descrDemiboldFont_1 = ImGui.GetIO().Fonts:AddFontFromFileTTF(M2TWEOP.getModPath().."/data/fonts/
data/descr-demibold.ttf", 18, nil, nil)
end
```

Next, their use is as simple as this :

```
ImGui.PushFont(_descrDemiboldFont)
ImGui.Text("Alpha")
ImGui.Text("Bravo")
ImGui.PopFont(_descrDemiboldFont)
```

The simplest way to resize a font is to use one of the multiplier coefficients (**_wd** preferably), but depending on the display ratio, the font could be in smaller or bigger than normal in the end.

Also, when you define a font, its default size must be specified (18 in the example above). You can next modify it in your script with **ImGui.SetWindowFontScale**, like this :

```
ImGui.SetWindowFontScale(0.8*_wd)
```

However, the further the display size is from the default size, the poorer the rendering quality will be (the font will be blurred if oversized, or pixelated if undersized). To avoid this, it's best to use several sizes (one being automatically selected depending on the the display ratio), as well as a new variable (**_fn**) as multiplier for fonts, as shown below:

```
function onLoadingFonts()
	_descrDemiboldFont_1 = ImGui.GetIO().Fonts:AddFontFromFileTTF(M2TWEOP.getModPath().."/data/fonts/
data/descr-demibold.ttf", 18, nil, nil)
	_descrDemiboldFont_2 = ImGui.GetIO().Fonts:AddFontFromFileTTF(M2TWEOP.getModPath().."/data/fonts/
data/descr-demibold.ttf", 27, nil, nil)
	_descrDemiboldFont_3 = ImGui.GetIO().Fonts:AddFontFromFileTTF(M2TWEOP.getModPath().."/data/fonts/
data/descr-demibold.ttf", 36, nil, nil)
end

function onCampaignMapLoaded()
	local screenHeight, screenWidth = 0, 0
	local window = ffi.C.GetActiveWindow()
	local rect = ffi.new("RECT")
	ffi.C.GetClientRect(window, rect)
	screenWidth = rect.right-rect.left
	screenHeight = rect.bottom-rect.top
	_wd = screenWidth/1920 ; _hg = screenHeight/1080

	if (screenWidth >= 1600) then
	_fn = _wd
	_descrDemiboldFont = _descrDemiboldFont_3
		elseif (screenWidth < 1300) then
			_fn = _wd*2.2
			_descrDemiboldFont = _descrDemiboldFont_1
		else
			_fn = _wd*1.5
			_descrDemiboldFont = _descrDemiboldFont_2
	end
end
```