Gui

Overview

There are currently two GUI-systems in the engine. The normal one which is called “retained” or just the “normal” GUI. This works by creating all the assets up-front and then works by having special callbacks for all of the GUI elements. For instance when a button is clicked a callback is sent to some piece of code that does the logic for that. It is easy to make an editor where you can build the element for this type of GUI (unfortunately we presently do not have one).

The other one is the immediate mode gui or just ImGui. This is a bit special and works by drawing graphics and doing logic at the same time. None of the GUI elements are saved by defined any created every update. This makes it very easy to make changes and extremely fast to iterate (especially when working in script). It does not work very well for more complex interfaces though and is mainly used to make stuff like HUD, Game menus and in-game interfaces.

The game also support to draw the GUI on rendered surfaces and how to achieve this will be brought up at the end of this document.

Retained GUI

Info coming later!

Immediate GUI

Summary

The Immediate Mode Gui (ImGui) is a bit special to how you normally do GUI and works by essentially recreating the entire interface every update. On top of this also handles logic and graphical update in the very same function. Normally this is a sort of sin, but for this system it works really well and make it very easy to set up interfaces. The ImGui have been designed around this concept and the main focus have been to have a system that make GUI creation and iteration really fast.

Anyone who has used Unity's older GUI should get the basics of this and it is actually worth looking through their manual for some extra information. This GUI has some extra tricks though and do somethings a bit differently so do not take this as a copy of the Unity one. One of the biggest changes is that it contains some state saving making it even easier to main things.

Another important feature of the ImGui is that it works seamlessly with both mouse, keyboard and gamepad input. The most important part of this is a system that calculates navigate with keys/stick on the fly so it is really easy to set up.

Names and Ids

In order to keep track of the state of the various GUI elements, the ImGui must save some states. In order to do this it creates 64-bit Ids from the names of the widgets, variables, etc. This makes it very important that the Name of the gui elements are all unique, or else the interface will not work.

Internally the actually string names are not used at all, but for speed reasons they are instead represented by 64 bit numbers (that are generated from the strings).

Widgets

Widgets are the different elements that make up an interface. Some of these are possible to interact with, while other are not. This is important to note so that you use the right kind of widget when you want a specific behavior. Worth noting is also that all widgets have one function that use the default data settings and one where a data class is needed as argument. All of the widget functions start with ImGui_Do*.

Some of the widgets have default values arguments, such as the ToggleButton, Slider, etc, which behave in a special way. The default value will only be used to set the initial value, and then as the value change through interaction, an internally saved value will be used instead. However, the default value is saved too, and if it changes then the changed default value will be used instead of the saved internal one.

Widget functions usually come in two version. One that uses the the default settings, which have been set up using ImGui_SetDefault*. The other comes with the data settings as an argument and have the suffix "Ext" (in helper functions, when directly using the ImGui class).

Finally it is only widgets that are affected by the layout, so never draw anything directly unless you know what you are doing. It is almost always much better to for instance use Label instead of DrawText to print a message.

Button
This is a normal button that returns true if it was pressed this update.

RepeatButton
Like Button, but will return true as long as it is being pressed.

ToggleButton
Another version of the button that returns can be be turn on/off. It returns the current state.

SliderHorizontal
A slider button, which has values between two numbers. A step-size value can also be set to specify in which interval between min and max values can take. It returns the current value.

SliderVertical
Just like SliderHorizontal, but placed vertically.

Label
The label does not have any interaction logic, but only displays a text string. If you need a text string

with interaction use a button instead.

Image
This displays an image. Like Label it has no interaction.

MultiToggle
A grid of toggle buttons, where only one can be active at a time. It returns the index (0=first) of the currently selected. In order to add items use ImGui_AddItem* functions.

CheckBox
A text next to a checkbox that can be on or off. Returns the current value.

TextFrame
A scrollable text of frame. It will return, in rows, the amount of text not outside of the window. For instance, if the text contains 5 rows, and only 4 are visible, 1 is the return value. This value is very useful when making a slider that can scroll the text.

MultiSelect
Here several different items can be selected. It returns the index of the currently selected one. In order to add items use ImGui_AddItem* functions.

Modifiers

Modifiers is a bunch of states that can be set in order to change the function and appearance of the widget. Once called, any widget defined afterwards will be affected by it. In order to remove the effects, one can use ImGui_ResetModifiers to set everything back to default values. Another way of handling it is to use the modifier stack. By ImGui_PushModifiers the current state of all the modifiers will be pushed on to the stack. One can then set a bunch of new modifier values and simple use ImGui_PopModifiers when the last recently pushed state of values should be used again.

Previous State

The previous state is a set of properties that are set after a widget function have been called. They can be used to get all sort of information on the state of the most recently defined widget. Mostly, these are useful when doing custom Widgets, (see below) and can then be used to see if the widget just became into focus, or more importantly what as the absolute size and position used to draw it.

Layout

State Variables

Fading

Timers

Custom Widgets

In-Scene GUI Rendering

In order for the a game entity to have a GUI surfaces the following criteria must be met:

- The surface that is to contain the GUI must be flat, consist of 4 vertices and have UV coords going from 0→1 where (0,0) is the upper left corner.
- The material of this surface must be translucent. It can be any blend mode though, making it possible to have transperant in-game GUI.
- The material must have a diffuse texture set, although this texture will never be used in-game (it will only show up in editors and such).
- Any other textures may also be set, so it is possible to give the GUI screen refractive patterns, etc.

Currently, the only entity type that support GUI is “Prop_Terminal”, so set the entity to that type and then in the ent-file class settings, set up the properties required. Most important is to set the ConnectedSubmesh variables which is the name of the flat surface where the GUI will be displayed. Also remember to set up a ScreenSize (in pixels) that has the same ratio has the ConnectedSubmesh.