Tutorial de Independiente

Sumario

This is our first basic C++ tutorial. It also shows how to write a stand alone executable using orx and how to use the localization module (`orxLOCALE`).

As we are **NOT** using the default executable anymore for this tutorial, its code will be directly compiled into the executable and not into an external library.

This implies that we will **NOT** have the default hardcoded behavior we had in the previous tutorials:

- F11 will not affect vertical sync toggler
- Escape won't automatically exit
- F12 won't capture a screenshot
- Backspace won't reload configuration files
- the [Main] section in the config file won't be used to load a plugin (“GameFile” key)

A program based directly on orx [1)], by default, will also **NOT** exit if it receives the orxSYSTEM_EVENT_CLOSE event.
To do so, we will either have to use the helper `orx_Execute()` function (see below) or handle it ourselves.

See previous basic tutorials for more info about basic object creation, clock handling, frames hierarchy, animations, cameras & viewports, sounds & musics, FXs, physics and scrolling.

As we're on our own here, we need to write the main function and initialize orx manually.
The good thing is that we can then specify which modules we want to use, and deactivates display or any other module at will, if needed.

If we still want a semi-automated initialization of orx, we can use the `orx_Execute()` function.
This tutorial will cover the use of orx with this helper function, but you can decide not to use it if its behavior doesn't suit your needs.

This helper function will take care of initializing everything correctly and exiting properly.
It will also make sure the clock module is constantly ticked (as it's part of orx's core) and that we exit if the orxSYSTEM_EVENT_CLOSE event is sent.
This event is sent when closing the windows, for example, but it can also be sent under your own criteria (escape key pressed, for example).

This code is also a basic C++ example to show how to use orx without having to write C code.
This tutorial could have been architectured in a better way (cutting it into pieces with headers files, for example) but we wanted to keep a single file per *basic* tutorial.

This stand alone executable also creates a console (as does the default orx executable), but you can have you own console-less program if you wish.
In order to achieve that, you only need to provide an argc/argv style parameter list that contains the executable name.
If you don't, the default loaded config file will be orx.ini instead of being based on our executable

name (ie. 10_StandAlone.ini).

For visual studio users (windows), it can easily be achieved by writing a `WinMain()` function instead of `main()`, and by getting the executable name (or hardcoding it, as it's shamelessly done in this tutorial).

This tutorial simply display orx's logo and a localized legend. Press space or click left mouse button to cycle through all the availables languages for the legend's text.

Some explanations about core elements that you can find in this tutorial:

- `Run function`: Don't put *ANY* logic code here, it's only a backbone where you can handle default core behaviors (tracking exit or changing locale, for example) or profile some stuff. As it's directly called from the main loop and not part of the clock system, time consistency can't be enforced. For all your main game execution, please create (or use an existing) clock and register your callback to it.
- `Event handlers`: When an event handler returns orxSTATUS_SUCCESS, no other handler will be called after it for the same event. On the other hand, if orxSTATUS_FAILURE is returned, event processing will continue for this event if other handlers are listening this event type. We'll monitor locale events to update our legend's text when the selected language is changed.
- `orx_Execute()`: Inits and executes orx using our self-defined functions (Init, Run and Exit). We can of course not use this helper and handles everything manually if its behavior doesn't suit our needs. You can have a look at the content of `orx_Execute()` [2] to have a better idea on how to do this.

Detalles

Recursos

[1]
ie. without the help of orx's launcher
[2]
which is implemented in `orx.h`