BOOTSTRAP MY_PAINT

INTRODUCTION TO USER INTERFACE

BOOTSTRAP MY_PAINT

Introduction

There are plenty of ways to interact with the user, the most natural being to use an interface, composed of menus and buttons (GUI).
This is what you will do step by step in this CSFML bootstrap.

This bootstrap will help you to understand how to handle basic UI, feel free to go further and create more of it.

Buttons

Button Structure

In the CSFML library, there is no predefined button, so you must define them by yourself.
A good starting point to create buttons is to define a structure that contains every attribute of the button; at least a size and a position.

CSFML gives us a structure called `sfRectangleShape` that can handle the shape of the button for us.

For now our button structure should look like this:

```
struct button_s {
        sfRectangleShape *rect;
};
```

You should obviously make a function to set everything up.
It must initialize the rect attribute of the given button, and have the following prototype:

```
struct button_s* init_button(sfVector2f position, sfVector2f size);
```

Add a color to your button using the appropriate SFML function.

Callback

Let's add actions to our buttons. To do this, you can either make a function like

```
sfBool is_button_clicked(struct button_s*, sfMouseButtonEvent*);
```

or pass pointers to functions directly from the `struct button_s` structure.

Let's do this for 2 states:

```
struct button_s {
        sfRectangleShape *rect;
        sfBool (*is_clicked)(struct button_s*, sfMouseButtonEvent*);
        sfBool (*is_hover)(struct button_s*, sfMouseMoveEvent*);
};
```

Since the cursor is just a point and buttons are rectangles, it is rather easy. Take a moment to understand how the functions `sfRectangleShape_getGlobalBounds` and `sfFloatRect_contains` work.

Manage event mouse button pressed :

```
...
if (event->type == sfEvtMouseButtonPressed) {
        if (button->is_clicked(button, &event->mouseButton)) {
            puts("Hello");
        }
    }
...
```

You should now see the "Hello" message in your terminal each time you click on the button.

Now, by yourself, manage the `hover` state of the button. The button should be white by default, green if the mouse passes over it.

Let’s take a break

Enum state

Are you familiar with enumerations? If not, please take a moment to learn about them, as we will be using them to manage states.

Let’s imagine several states:
- HOVER,
- PRESSED,
- RELEASED

Let’s declare our enumeration:

```
enum e_gui_state {
    NONE = 0,
    HOVER,
    PRESSED,
    RELEASED
};
```

And let’s modify the structure accordingly:

```
struct button_s {
        sfRectangleShape *rect;
        sfBool (*is_clicked)(struct button_s*, sfMouseButtonEvent*);
        sfBool (*is_hover)(struct button_s*, sfMouseMoveEvent*);
        enum e_gui_state state;
};
```

You now have a button in which the state is located. This may help you for the next step!

I also encourage you to create the corresponding macros: `IS_HOVER(button)`, `IS_PRESSED(button)`, `IS_RELEASED(button)`

Modify your `is_clicked` and `is_hover` functions to change the state of your button.

Drop-down menu

A new world !

More complex, the drop-down menu must have several options. These options must be displayed only when you hover over them.

Let's start by creating two new data structures for dropdown menus and their options. They could look like this:

```
struct s_gui_options {
    struct s_gui_object* option;
    struct s_gui_options* next;
};

struct s_gui_drop_menu {
    struct s_gui_object* button;
    struct s_gui_options* options;
};
```

Consider the `struct button_s` like `struct s_gui_object` from now on

As you can see, we use chained lists to manage the options. But you can use another method if you want.

We already have in the **struct s_gui_object** the function monitors corresponding to the actions. So we will create specific functions for the drop-down menu options, without having to change the structure of our code!

Come on, since you are running, create the following functions:

```c
/**
** @brief create a new drop menu, with position and size
** @param position Position of the menu
** @param size Size of the menu
** @return Address of the new struct s_gui_drop_menu allocated
*/
struct s_gui_drop_menu* create_drop_menu(sfVector2f position, sfVector2f size);

/**
** @brief Add a new option to drop menu, taking position and size of initial button
**        For now, no text, only color
** @param drop_menu Menu where add option
** @return Address of the struct s_gui_drop_menu edited
*/
struct s_gui_drop_menu* add_option_drop_menu(struct s_gui_drop_menu* drop_menu);
```

Create a **struct s_gui_drop_menu** in your window, and add 3 options. Be careful about the position of the options (do you want a dropdown menu that opens at the bottom? or at the side?)

The options should only be displayed if the first menu is in the *HOVER* state.

You should have all the tools to manage your dropdown menu, its states and the corresponding function calls.

Use of pixels

Array of pixel

In graphical programming, each pixels are defined with 4 values between 0 and 255 (red, green, blue, alpha).

Init an array of sfUint8 with the size you need.

You can determinate the number of value you need by the number of pixels on the window.

Draw my array

In CSFML, You already used types to draw what you wanted to. There is also way to draw pixels thanks to those types.

Once you done it, there is time to edit your array thanks to the events.

There is also other type of csfml thay you may not know. Check the sfImage type in the documentation. Make it all easier.

Perfect, you can edit your image the way you want.

A user interface

Now you can combine the previous steps to create a complete (or almost complete) GUI, with actions.

Maybe take some time to structure your code with this new knowledge, and good luck!

v 2.0

EPITECH
LEARN DIFFERENT*

8