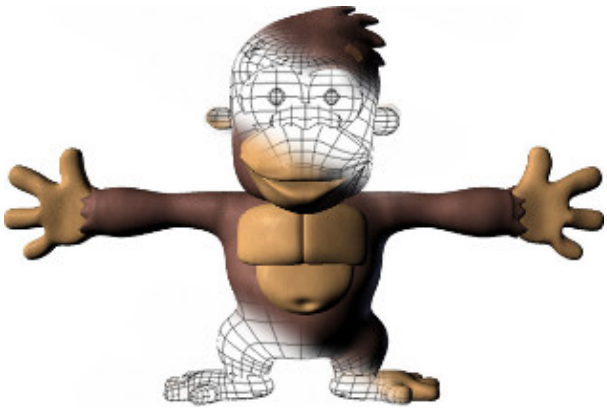
B2 - C Graphical Programming

B-MUL-200

Bootstrap my_defender

Introduction to user interface



1.0

{EPITECH.}

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:

```
void init_button(button_t *button, sfVector2f position, sfVector2f size);
```

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

+ CALLBACK

Let's make our buttons clickable, by binding a function to be triggered (often called a listener or callback) to them.
For testing purpose, we will make it only print "Hellon", and it must comply with the following prototype:

```
void print_hello()
```

We also need a function to check if the mouse cursor is within the button when the player clicks.

```
int button_is_clicked(button_t button, sfVector2f click_position)
```

This function returns 1 when the player clicked on the button, and 0 otherwise.

Since the cursor is just a point and buttons are rectangles, it is rather easy.

Add a function pointer with no parameter, named `callback`, to the button structure, to associate a function with each button.

You can now assign the `print_hello` function created earlier to it during the button's initialization.
To trigger the callback of any button, you just have to do it the following way:

```
if (button_is_clicked(button, click_position)
        button.callback();
```

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

+ IMPROVEMENTS

Here are a few things you can do to improve your buttons from now:

- add different visual states to your button (idle, click, rollover),
- add a text to your button,
- use sprites instead of just a color to your button.

Scenes

+ Scene Structure

In game development, a scene is a way to organize your game objects and user interface: a starting menu is a scene, a game screen is also a scene for instance.
In CSFML, a scene can be viewed as a structure that could look like this:

```
struct scene_s {
        struct game_object_s *objs;
        struct button_s      *buttons;
};
```

with `buttons` and `objs` being an array of the buttons and a list of game objects present inside of your scene.

This way of storing structures allows us to have several different scenes within a single array.

Create a function that initializes a two-scene-array.
The first one is a starting menu with:

- a background image (a castle for example),
- 2 buttons in its center, one to switch scenes, the other to close the game.

The second scene contains:

- a different background image (representing a battlefield for example),
- 3 buttons, all with the same callback prototype, to add different towers.

You should always think about making several functions associated with your structures to handle them.

+ Scene Drawing

Finally, in order to be able to draw any scene, make a function named `drawScene` with the following prototype:

```
void drawScene(sfRenderWindow *window, struct scene_s *scenes, int current_scene);
```

the first parameter is the window to draw into,
the second one is the array of scenes,
the third one is the index to current scene in the array.

This function should, in that order:

- clear the window,
- draw every game object (starting with the background),
- draw every button,
- draw every text (if you have them).

+ IMPROVEMENTS

A good improvement for your scene management could be to load the scenes thanks to XML files, easily customizable, like this:

```
<Scene name="Starting_Menu">
    <Game_Object name="background" image="my/path/to/background.png" x="0" y="0"/>
    <Button name="Start" image="my/path/to/button.png" x="WIDTH/2" y="HEIGHT/2"/>
    <Button name="Quit" image="my/path/to/button.png" x="WIDTH/2" y="HEIGHT/2 + 50"/>
</Scene>
```