Images and textures

We have seen how to draw colorful shapes to the screen, but this is not enough to make a real game!

We need the ability to draw images.

In SFML, this is a process with two parts.

- First, we load the image from a file
- Then, we print it to the screen as part of another object

The thing to note is that images cannot be displayed directly, but rather they are used to paint objects like `RectangleShapes`.

Loading images on memory

An image is represented by the `sf::Texture` class (NOT `sf::Image`, which is a different thing!)

They can be loaded by using the `image.loadFromFile` method, which takes the file name as an argument.

Filenames can be absolute (e.g. `C:/Users/mario/Desktop/someImage.bmp`) or relative (e.g. `someImage.bmp`).

If they are relative, SFML will look for the image in the folder where the program is located, so make sure to put them in the right place!

Drawing images to the screen

A `sf::Texture` object cannot be drawn directly to the screen.

We must use a `sf::RectangleShape` or a `sf::CircleShape` (or a more advanced shape like a `sf::VertexArray`).

This is done by calling the `rectangle.setTexture` method.

See an example in `TexturedRectangle.cpp` - don't forget to make sure the program can find the image.

Some things to consider

Notice that we have manually set the size of the rectangle to be the same as the image. What would happen if we set a different size for the rectangle?

What happens if you use `rect.setFillColor` to change the color of the rectangle?

Try drawing more than one image at the same time - and note that many rects can share the same texture! What happens if they overlap?

As before, we can move textured rectangles around!

Animations

Animating the image in a rectangle is done by quickly switching textures.

This can be done without making a new rectangle.

Animation is usually done on a timer. These can get sophisticated, but the simplest thing is to store an integer and increment it every frame.

Check out `AnimatedTexture.cpp`.

Try to make the animation faster or slower, try to add a new stage.

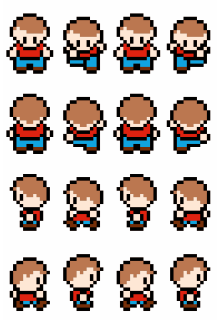
What happens if you want two animations at the same time? (hint: you need two timers).

Sprite sheets

For more complex animations, we could have tens or hundreds of images.

It's cumbersome and slow to load a hundred images into memory.

The traditional solution is using sprite sheets: many images stored into a single file.



Animations and sprites

A sprite sheet is loaded just like a normal image, using `sf::Texture`.

To select which of the sprites to use, we call `rect.setTextureRect`, which allows us to specify the part of the sprite sheet that will be printed by its coordinates.

These coordinates are determined by a rectangle `sf::IntRect(x, y, w, h)` given by an origin `x, y`, width and height `w, h`.

To change the animation frame, we simply select different source coordinates! See `SpriteSheet.cpp`

A small exercise

In the provided sprite sheet, there are four rows, each containing a walking cycle for each direction.

Make it so that the character is static unless the user presses a direction key, then the character starts moving in that direction, until the user releases the key.

Reminder: you can use `sf::Keyboard::isKeyPressed(sf::Keyboard::key)` to check if a key is pressed.

Putting it all together

We will recreate a (very small) part of The Legend of Zelda!

See the file `Game.cpp` to get started.

The code for drawing the scene and creating and animating enemies is provided.

The code for creating and displaying the player is provided as well.

You need to implement code to:

- Animate the player character depending on the direction it's walking
- Prevent the player from walking off-screen
- Hurt the player when it touches a monster
- Make the player able to attack the monsters (tricky!)

Extra! Sound

SFML also allows us to play sounds.

For this, you will need to add `sfml-audio` to the list of libraries in your project.

You will also need to `#include <SFML/Audio.hpp>`

Playing sounds works exactly like using textures. First you load your sound into a `sf::SoundBuffer`, then you create a `sf::Sound` to play it.

Extra! Sound

SFML also allows us to play sounds.

For this, you will need to add `sfml-audio` to the list of libraries in your project.

You will also need to `#include <SFML/Audio.hpp>`

Playing sounds works exactly like using textures. First you load your sound into a `sf::SoundBuffer`, then you create a `sf::Sound` to play it.

Sound example

```
sf::SoundBuffer buffer;
buffer.loadFromFile("mySound.wav");
sf::Sound sound;

sound.setBuffer(buffer);
sound.play();
sound.pause();
sound.stop();
```

See also the example in class `Player`

To know more

Resources on C++

- Programming: Principles and Practice Using C++ (for beginners!)
- C++ Primer (for programmers that are new to C++, good reference)
- Accelerated C++ (like C++ Primer but shorter!)
- The C++ Programming Language (the definitive reference guide)

Resources on SFML

- SFML documentation and tutorials on the web page (REALLY good)
- SFML Blueprints (badly written, but good for games)

Or just drop me an email!