What is SFML?

SFML stands for “Simple Fast Media Layer”

It is a set of convenient functions that let you:

- Draw 2d graphics on the screen
- Get information from the mouse and keyboard
- Play sounds and music
- Interact with other computers over the network

All of these can be achieved without SFML, but it becomes much harder, and it is done differently on different operating systems!

Your first SFML program

We will create a window and wait until the user presses Escape

See `BasicWindow.cpp`

Drawing to the screen

Animation in games is based on the same principle as movies. It is an illusion based on drawing static images (frames) very fast!

On every frame, we clear the screen completely, then redraw the entire scene.

See `BouncingBall.cpp`

Coordinate systems

Coordinates in SFML are, by default, relative to the upper-left corner. I.e. the upper-left corner of the window is point $(0,0)$.

Movement goes down and to the right, i.e. moving something $+3$ units on the x axis will move it to the right, while moving it -3 units will move it to the left.

Similarly, moving something $+3$ units on the y axis will move it down!

Exercises

Play around with `BouncingBall.cpp` until you're sure you understand what's going on!

Some exercises that you should be able to do:

- Make the ball bigger or smaller (make sure it's still centered!)
- Make the ball fall faster or slower
- Make the ball move to the sides as well (bonus points if you make it bounce against the walls)
- Allow the user to change the ball size or color or speed by pressing keys.
- Make it so the ball slows down on every collision so it eventually stops bouncing.

Some issues

By now you should have noticed two quirks:

One, the window sometimes freezes and becomes unresponsive.

- This is due to the *event queue*.
- Every time you click on the window, or click on minimize or close, a request is sent to the window
- But we aren't processing these requests, so they accumulate and eventually the window becomes unresponsive

Two, the program runs at different speeds on different machines.

- This is because we are processing each frame as fast as we can, so faster computers do it more often.

Emptying the event queue

To solve the first issue, we just need to make sure that we go through the event queue regularly.

We don't need to process these events, just remove them from the queue!

This is done with the `window.pollEvent` function, which pops the next event off the queue.

See `BouncingBallPolling.cpp` to see an example.

Framerate limiting

Solving the speed issue is just as simple.

All we need to do is standarize on a frame rate (normally 60 frames per second).

Then we calculate how long each frame should be displayed for (1/60 of a second).

After the program is done processing a frame, if it took less than this time, we make it wait.

Or, alternatively...just use `window.setFramerateLimit(60)`

See `BouncingBallComplete.cpp`

Collision detection

When making games, we often want to check when two objects are touching.

We can get the position of a shape with `shape.getPosition()`, and this is good enough to check when the shape collides with the screen.

It is, in general, very hard to know when two shapes touch, so instead we approximate a shape by a rectangle, the so-called "bounding box" - we merely check when bounding boxes are touching.

See `Paddles.cpp` for an example.

Putting everything together

Now that you've come this far, you can try turning `Paddles.cpp` into a playable Pong game!

There are three steps to this.

Moving the ball

The ball currently only moves from side to side - that is hardly a challenging game!

Make it so the ball moves diagonally and bounces against the top and bottom edges of the screen!

If the ball goes too far off the left or right side, make it re-set back to the center. You can also keep track of the score of each player.

Moving the player

One of the paddles should be controllable by the player.

Add code to the main loop so it checks whether the up or down keys are being pressed (using `sf::Keyboard.isKeyPressed` and `sf::Keyboard::Up`) and move the player paddle acordingly.

What should happen when the player hits the top or the bottom? Should they stop? Should they come out the other side? Your choice.

Moving the opponent

Here you have two options.

One: you can have another player control the opponent with the keyboard so two people can play the game at the same time.

Two: you can make the opponent move on its own. To do this, in each iteration the code should check where the ball is and move the opponent accordingly (i.e. if the ball is moving up, the opponent's paddle should move up, and vice versa).

It's easy to make a perfect opponent, but it's harder to make an opponent that can lose!

Other things to try

Keep count of both player's scores. When one of them reaches some limit, close the window and print a victory message.

Make the ball go faster as time goes on.

To make things more even, if one player is winning for too long make their paddle smaller.

Add some randomness to the game. This can range from randomizing the speed of the ball on each bounce to changing the angle at which it bounces.

Add another ball! Bonus points if you write a `Ball` class to avoid repeating the same code.