CMPT 433 How To Guide

Sepehr Ahmadipourshirazi
Andrew Lam

Overview

In this guide, you'll learn how to:

- Set up SFML on your Linux system (or embedded Linux board if supported)
- Write a simple C++ program that opens a green window
- Use arrow keys to move a box on the screen
- Compile and run your application using CMake

This tutorial assumes you're familiar with basic Linux terminal usage and C++ programming but not with SFML.

Part 1: Installing SFML and Development Tools

Install Dependencies:

Open a terminal and run the following commands to install the required packages:

```
$ sudo apt update
$ sudo apt install libsfml-dev g++ make
```

NOTE: Make sure the installation finishes without errors.

Part 2: Writing the Code

Step 1: Include the Required Header

To start using SFML, you need to include its graphics module:

```
#include <SFML/Graphics.hpp>
```

This includes the entire SFML graphics library, which internally pulls in modules like window, system, and graphics. It's all you need for basic 2D apps.

Step 2: Create the Main Window

Copy the following code to start the program and create the main application window:

```
int main() {
   sf::RenderWindow window(sf::VideoMode(800, 600), "SFML Window");
   return 0;
}
```

Step 3: Drawing the Red Box

SFML makes it very intuitive to draw shapes using its built-in **shape classes**. In this case, we're using '*sf::RectangleShape*', which is part of the **SFML Graphics module**.

Here is an example of how we draw our rectangle box and set properties for it:

```
// Create a red square of size 50x50
sf::RectangleShape player(sf::Vector2f(50, 50));
player.setFillColor(sf::Color::Red);

// Start near the center
player.setPosition(375, 275);
```

Step 4: Game Loop

Now that we have the player box, we can move on to handling the game loop and the keyboard inputs.

In SFML, the main logic of your program runs inside a loop like this:

```
while (window.isOpen()) {
    ...
}
```

This is called the **game loop** or **main loop**, and it continues running until the user closes the window. Let's break down everything that happens inside it.

Within the game loop, we can start by adding logic for closing the window if the user clicks the ‘x’ at the top right of the window:

```
sf::Event event;
while (window.pollEvent(event)) {
    if (event.type == sf::Event::Closed)
        window.close();
}
```

`sf::Event event` → declares a variable that will store different types of events, like key presses, mouse movements, or closing the window.

`window.pollEvent(event)` → checks if there are any new events, and processes them one by one.

`event.type == sf::Event::Closed` → checks if the window's close button (X) was clicked. If so, `**window.close()**` shuts down the app.

Step 5: Input Handling

Below is an example of how you can listen for keyboard inputs and move the box with the arrow keys inside the game loop:

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
    player.move(-1, 0);
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
```

```
    player.move(1, 0);
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
    player.move(0, -1);
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
    player.move(0, 1);
```

`sf::Keyboard::isKeyPressed(...)` → checks if a key is currently being held down.

`player.move(x, y)` → function shifts the red box by the specified amount:

- (-1, 0) = move 1 pixel left
- (1, 0) = move 1 pixel right
- (0, -1) = move 1 pixel up
- (0, 1) = move 1 pixel down

Why not use events for this? SFML lets you check if a key is held down **outside of events** for smooth movement (like holding down an arrow key), instead of reacting to each individual key press event.

Step 5: Finalizing Application

At the end of the main game loop, all you need to do is to clear the screen, draw the red square in its position, and display the updated frame:

```
window.clear(sf::Color::Green);  // Set background to green
window.draw(player);             // Draw the red box
window.display();                // Show the final frame
```

Here is an overview of what your entire code should look like so far:

```
#include <SFML/Graphics.hpp>

int main() {
   sf::RenderWindow window(sf::VideoMode(800, 600), "SFML Window");


   // Create a red square of size 50x50
   sf::RectangleShape player(sf::Vector2f(50, 50));
   player.setFillColor(sf::Color::Red);
```

```
    // Start near the center
    player.setPosition(375, 275);


    while (window.isOpen()) {

        /*
        declares a variable that will store different types of events,
        like key presses, mouse movements, or closing the window.
        */
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
            player.move(-1, 0);
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
            player.move(1, 0);
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
            player.move(0, -1);
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
            player.move(0, 1);


        window.clear(sf::Color::Green);  // Set background to green
        window.draw(player);             // Draw the red box
        window.display();                // Show the final frame
    }

    return 0;
}
```

Step 5: Building & Execution using CMake

In order to build and execute the application, start by creating a CMakeList.txt in the same directory as your main.cpp (the code that we just wrote for the game). Then, copy paste the code below into your CMakeList.txt:

```
cmake_minimum_required(VERSION 3.10)
project(box_game)

set(CMAKE_CXX_STANDARD 20)
```

```
# Find SFML
find_package(SFML 2.5 REQUIRED COMPONENTS graphics window system)

# Create the executable
add_executable(box_game main.cpp)

# Link SFML to your actual executable target
target_link_libraries(box_game sfml-graphics sfml-window sfml-system)
```

Once you have the cmake ready, run the following command to build you application:

```
 $ cmake -B build && cmake --build build
```

Now all you have to do to run the application is to go inside the build directory and execute the **box_game** executable:

```
$ cd build/
$ ./box_game
```

Troubleshooting

SFML Not Found by CMake

Error:
`Could not find SFML (missing: SFML_GRAPHICS_LIBRARY SFML_WINDOW_LIBRARY SFML_SYSTEM_LIBRARY)'

Fix:
Make sure SFML is installed:

```
$ sudo apt install libsfml-dev
```

Then delete the build folder and re-run the commands:

```
rm -rf build
cmake -B build
cmake --build build
```

Linking Error

Error:
`undefined reference to `sf::RenderWindow::RenderWindow(...)`

Fix:
This means the linker isn't finding SFML. Make sure your **CMakeLists.txt** links the required components:

```
target_link_libraries(box_game sfml-graphics sfml-window sfml-system)
```