

## Laboratory Document on Simple, Fast Multimedia Library (SFML)

### **Introduction:**

Simple, Fast Multimedia Library (SFML) is a library in C++ for programming graphics based applications. In particular, this library has been used for making games and multimedia applications. It is composed of five modules: system, window, graphics, audio and network.

### **Objectives:**

The following are objectives of this lab:

- To learn Simple Fast Multimedia Library for programming graphics in C++.
- To draw graphics using drawing primitives in SFML.
- To learn about events and event handling using SFML.
- To make simple animations using SFML.
- To make simple animated as well as strategy games in SFML.

### **Structure of the Lab:**

Since this lab is on a new library, particularly graphics library, this lab will consist of several examples as well as exercises. It is important to learn from the examples, since content of several of the exercises relies on insights from these examples.

### **SFML Tutorials and Documentation:**

We will be using SFML 3.0.x (for the exact version, please see the posted installation guide on LMS). Tutorials are available on the SFML Website, (see <https://www.sfml-dev.org/>). For the API documentation, please refer see <https://www.sfml-dev.org/documentation/3.0.0/> )

Note: The version number can be changed within the URL.

**Example 1:** [Basic Window] Compile and execute this program. (Name it example01.cpp)

```
#include <SFML/Graphics.hpp>
#include <iostream>
//This program creates a basic window and closes it on pressing the 'Esc' key

int main() {
    // Create a window with a title and size 800x600
    sf::RenderWindow window(sf::VideoMode(800, 600), "Basic SFML Window");

    // Main loop to keep the window open
    while (window.isOpen()) {
        // Event processing
        sf::Event event;
        while (window.pollEvent(event)) {
            // Close the window when close button or escape key is pressed
            if (event.type == sf::Event::Closed ||
                (event.type == sf::Event::KeyPressed
                 && event.key.code == sf::Keyboard::Escape)) {
                std::cout << "Closed the window";
                window.close();
            }
        }
        // Clear the window with a black color
        window.clear(sf::Color::Black);
        // Display what has been drawn so far
        window.display();
    }
    return 0;
}
```

Some classes in the above Example: `RenderWindow`, `VideoMode`, `sf::Event`, `sf::Color`. Observe the functions of the object `window`, which is an object of type `RenderWindow`: `window.clear()`, `window.display()`, `window.close()`, etc.

**Event Handling:** In event-driven programming, a program waits (loops infinitely) for an event to happen. Examples of events are clicking of a mouse, pressing of a key, etc. In the above program, the window waits for an event to happen within the while loop. If the event is of type `sf::Event::Closed` or `sf::Event::KeyPressed`, the event is triggered. The event is triggered when the 'Close' button of the window is pressed. The action taken by the program is to close the window. From the documentation, we see

```
void sf::Window::close ( )
```

Close the window and destroy all the attached resources.

Example 2: [Basic Circle Shape] Compile and execute this program. (Name it example02.cpp)

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

int main() {
    sf::RenderWindow window(sf::VideoMode(800, 600), "Drawing a Circle");

    // Create a circle shape with radius 50
    sf::CircleShape circle(50);
    circle.setFillColor(sf::Color::Green);

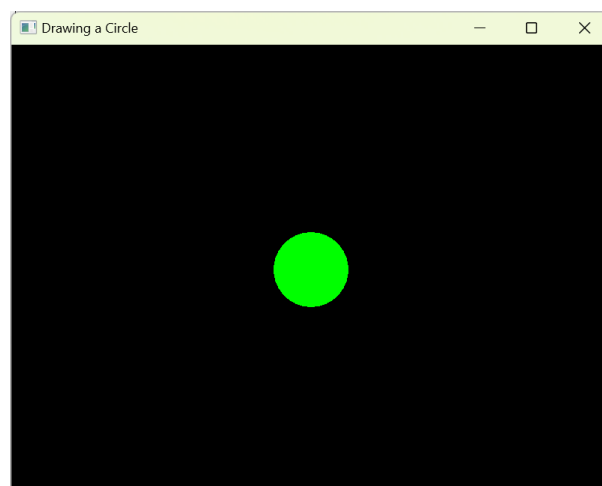
    // Set the position of the circle in the center of the window
    circle.setPosition(400 - 50, 300 - 50); // (window width/2 - radius, window
height/2 - radius)

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

        window.clear();
        window.draw(circle); // Draw the circle on the window
        window.display();    // Display the contents
    }

    return 0;
}
```

Output:



Example 3: [Moving Rectangle] Compile and execute this program. (Name it example03.cpp)

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

int main() {
    // Create a video mode object (window size 800x600)
    sf::VideoMode videoMode(800, 600);

    // Create a render window
    sf::RenderWindow window(videoMode, "Move Rectangle with Arrow Keys");

    // Set the frame rate limit to make movement smoother
    window.setFramerateLimit(60);

    // Create a rectangle shape (size 100x50)
    sf::RectangleShape rectangle(sf::Vector2f(100.f, 50.f));

    // Set the rectangle's initial position
    rectangle.setPosition(350.f, 275.f); // Center of the window

    // Set the rectangle's fill color
    rectangle.setFillColor(sf::Color::Green);

    // Movement speed (pixels per frame)
    float movementSpeed = 5.0f;

    // Main game loop
    while (window.isOpen()) {
        // Event polling (for handling window close, etc.)
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        // Real-time input for moving the rectangle
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up)) {
            rectangle.move(0.f, -movementSpeed); // Move up
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down)) {
            rectangle.move(0.f, movementSpeed); // Move down
        }
        if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left)) {
            rectangle.move(-movementSpeed, 0.f); // Move left
        }
    }
}
```

```

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right)) {
        rectangle.move(movementSpeed, 0.f); // Move right
    }

    // Clear the window
    window.clear();

    // Draw the rectangle
    window.draw(rectangle);

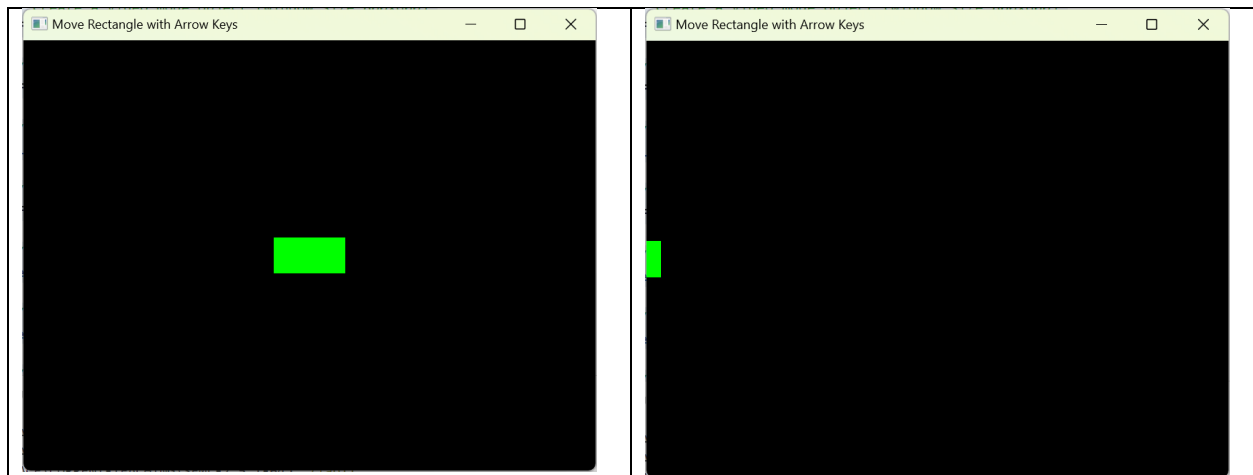
    // Display the rendered frame
    window.display();
}

return 0;
}

```

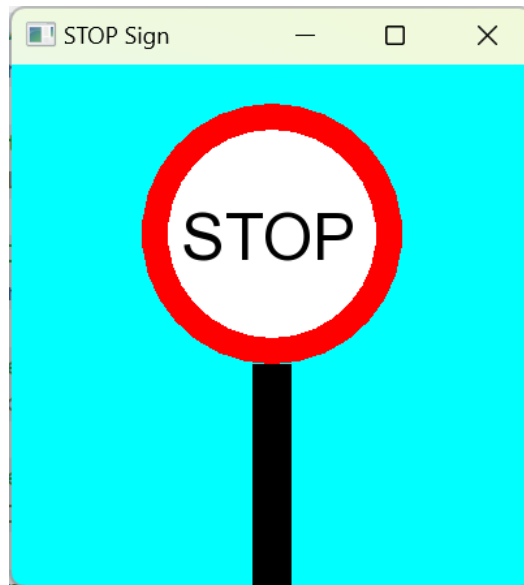
The above program creates a rectangle that can be controlled using arrow keys. Experiment with various combinations (e.g. pressing the up and left key together). Observe that the rectangle vanishes if it crosses the boundary of the window.

Below the output is shown. The left image is the initial configuration, while the right image is when the rectangle is about to vanish from the window.



**Exercise I:** [Wrapped Moving Rectangle] Rewrite Example 03 as `exercise01.cpp`, such that the rectangle wraps around, i.e. when it reaches the end of the screen, it re-emerges on the other side. This behavior should be visible for both horizontal and vertical edges of the screen.

**Exercise 2:** [Stop Sign] On a window of size 400x300 construct the following picture. Let's name this program as `exercise02.cpp`. Note that you will have to explore various library functions for drawing Graphics primitives.

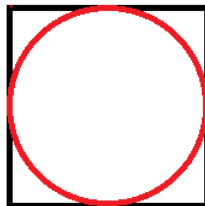


Some details:

- Circle Radius: 80
- Circle Border Thickness: 20
- Post dimensions: 30x200.
- Font type: Arial.ttf
- Font size: 50
- Background Color: Cyan

About the function `setPosition`. You have seen the function `setPosition` in previous programs. Since this program requires you to accurately compute pixel positions, let's describe the function `setPosition(x, y)` describes the top-left corners of the bounding box in which the shape is to be placed. Thus `circle.setPosition(x, y)` places the circle in a bounding box with coordinates as follows:

*(x, y) - top left  
coordinates of  
the bounding  
box*



However, for fonts the `setPosition` function does *not* work this way. Check it out yourself!

**Example 4: [Bouncing Ball Animation]** Compile and execute this program. Call it example04.cpp

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

int main() {
    // Create a window of size 400x300
    sf::RenderWindow window(sf::VideoMode(400, 300), "Bouncing Ball");

    // Create a ball (circle) with radius 50
    sf::CircleShape ball(20);
    ball.setFillColor(sf::Color::White);

    // Initial position of the ball (starting near the center)
    ball.setPosition(100, 100);

    // Ball velocity (initially moving in both X and Y directions)
    sf::Vector2f velocity(0.05f, 0.05f); // X and Y velocities

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

        // Get the current position of the ball
        sf::Vector2f position = ball.getPosition();

        // Check for collision with the walls and bounce back
        if (position.x <= 0 || position.x + ball.getRadius() * 2 >= 400) {
            velocity.x = -velocity.x; // Reverse direction in X axis
        }
        if (position.y <= 0 || position.y + ball.getRadius() * 2 >= 300) {
            velocity.y = -velocity.y; // Reverse direction in Y axis
        }

        // Move the ball by the velocity
        ball.move(velocity);

        // Clear the screen
        window.clear();

        // Draw the ball
        window.draw(ball);
    }
}
```

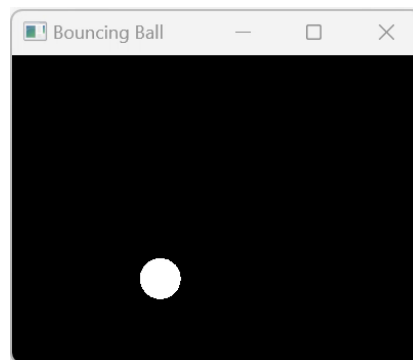
```

    // Display the contents
    window.display();
}

return 0;
}

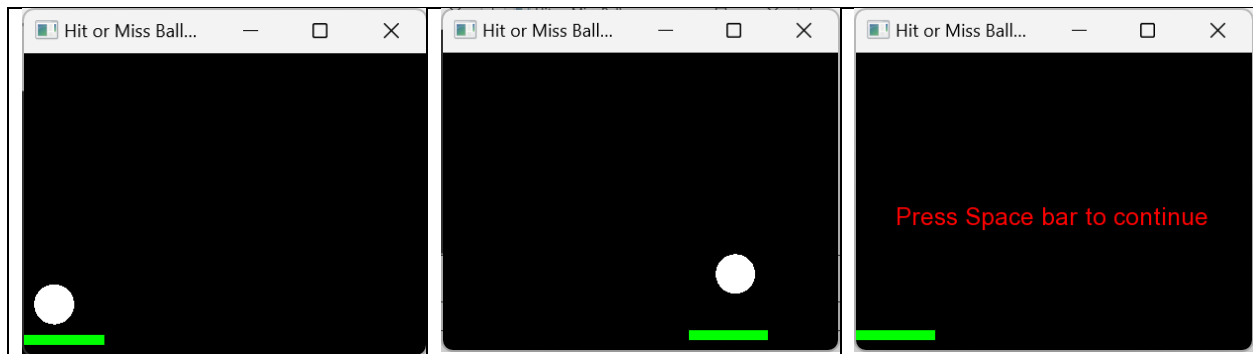
```

The output will be:



**Exercise 3:** Use the above program to create a paddle ball game. Call this program `exercise03.cpp`. The paddle moves left and right using keyboard controls. If the ball hits the paddle it bounces back. If the ball misses the paddle, the game pauses and a message 'Press Space Bar to continue' is displayed. Once the game continues, the ball restarts from its initial position.

Hint: You may use a class called **Player** to emulate a **Player**.



**OPTIONAL:** Once done, consider adding features to the game such as: increasing the speed as the game progresses, adding a scoreboard, having multiple balls in the game, etc.