CSC 481/581 Fall 22 Homework 1: SFML Foundations

Due: 09/07/22 by the start of class

Overview

Your task for this assignment is to explore the basics of constructing a simple game environment using SFML. While this code may not appear to be related to a game engine, it is laying the foundations upon which you will build out your game engine throughout the course of this semester. This is an ***individual assignment***, you are to work alone. As always, you are expected to abide by the University's Academic Integrity Policy (`http://policies.ncsu.edu/policy/pol-11-35-01`), which includes providing appropriate attribution for all external sources of information consulted while working on this assignment.

There are 125 points available on this assignment. Students enrolled in 481 are required to complete the first 100 points (Sections 1–4), but may also choose to complete Section 5. Students enrolled in 481 earning scores above 100 will receive a 100 on the assignment (*i.e.*, extra points will not be given as extra credit). Students enrolled in 591 are required to complete all 125 points, and will receive a grade as the percentage of the 125 points they earn. All students are required to submit a writeup addressing the sections of the assignment they have completed.

Development vs. Grading Environments

In general, you are free to develop your code using any tools and any environment you wish; however, with such a large course it will be crucial that we standardize on an environment to facilitate grading. Regardless of what you choose to use for developing your code, you will have to:

1. Use and submit a `Makefile` that will allow your code to be compiled in a standard grading environment.
2. Ensure that Makefile works in a standardized environment for compiling your code without errors, and that the resulting binary executes without errors.

The grading environment will be Ubuntu 20.04, with very little in the way of additional libraries installed. *Note: the instructor may opt to add libraries for future assignments.* As one part of this assignment, you will have to setup a grading environment using the instructions provided on Moodle.

You may choose to develop your code in a different environment than you validate grading in. For example, macOS can easily install SFML via homebrew (even on Apple Silicon). You may choose to use a fully-featured code editor like VS Code. This is all completely up to you; however, keep in mind that ***it is your responsibility to validate that your code compiles and runs in the grading environment as specified in this assignment and the instructions provided for setting it up.***

Part 0: Setup Grading Environment (0 points)

The standardized grading environment will be Ubuntu 20.04.

- If you are already running Ubuntu 20.04 on the machine you plan to use for this course, I strongly suggest you follow the instructions and setup a VM to test with to ensure there are no accidental dependencies you aren't aware of.

- If you're using macOS, I recommend Ubuntu's Multipass as a fast and free solution. That being said, Multipass recently has not been as stable as in the past, so you may choose to go with a more full-featured virtualisation solution from VMWare or Parallels for development purposes. UTM is another free option designed specifically for MacOS.
- If you're using Windows, there are several options depending on your Windows version (10 vs. 11, insider preview or not). Multipass does work on Windows, but you will need X11 software to display windows. You can also follow the instructions on Moodle for installing via WSL2 to get around this requirement.

When you have completed this part of the assignment, you should have a functioning Ubuntu 20.04 installation with minimal packages installed.

The packages that will be installed on top of the base system are: `x11-apps`, `build-essential`, and `libsfml-dev`. If you depend on anything else, you run the risk of not getting credit for any implementation.

Section 1: Your First SFML Project (25 points)

It's time to create a window so that you can draw to the screen. Follow the SFML tutorial on "Opening and managing a SFML window": `https://www.sfml-dev.org/tutorials/2.5/window-window.php`. There are many things you can do with a window in SFML. For this part of the assignment, you must:

- Write a `main()` function in your main.cpp file that opens a `sf::Window`.
- Demonstrate a window that is at least able to be closed (`sf::Style::Close`), if not also able to be resized (either `sf::Style::Resize` or `sf::Style::Default`).
- The window must appear on the screen, without going over the edge (800x600, as in the tutorial, is a reasonable size).

Section 2: Drawing Objects (40 points)

For this part of the assignment, you will be starting to draw and move objects across the screen. In particular, you have to create:

1. a "*platform*" which is a shape (let's assume a rectangle)
2. a "*character*" which is a special type of shape that we'll be controlling using inputs (requirements in Section 4).
3. a "*moving platform*" which is a special kind of platform that moves around in a regular pattern.

There are three tutorials for SFML I would base your work on:

- The "Drawing 2D" tutorial:
  `https://www.sfml-dev.org/tutorials/2.5/graphics-draw.php`
- The "Shape" tutorial:
  `https://www.sfml-dev.org/tutorials/2.5/graphics-shape.php`
- The "Transforming Entities" tutorial:
  `https://www.sfml-dev.org/tutorials/2.5/graphics-transform.php`

To create your shapes, you ***must create a class that extends the `sf::Shape`*** (either the `sf::CircleShape` or `sf::RectangleShape` will suffice). The requirement for this assignment is to use inheritance. While there is no explicit requirement regarding classes beyond that, it may be wise to consider at least three separate classes in some relationship for your platform, character, and moving platform.

All three objects must appear on the screen at the same time, and at least one of them should be textured. For now, the moving object should move in some regular pattern (*e.g.*, left to right then back again, top to bottom then back again, *etc.*). The movement should be continuous (with the exception of brief pauses as part of the pattern), and should repeat once the pattern is completed. Feel free to go beyond these minimum requirements and create arbitrarily complex patterns!

Section 3: Handling Inputs (25 points)

For this part of the assignment, your task to make a shape that is controlled by inputs, either keyboard or mouse. When you're done, some combination of keyboard inputs and/or mouse movements and clicks should enable you to control the movement of your character shape in two dimensions. Hint: We'll be turning these building blocks into a 2D platformer game, so implementing left-right movement and a "jump" may be a good way to get ahead for future assignments.

The SFML tutorial that covers inputs can be found here: `https://www.sfml-dev.org/tutorials/2.5/window-inputs.php`. For this assignment, please use "real-time input". The tutorial states "Real-time input allows you to query the global state of keyboard, mouse and joysticks at any time ('is this button currently pressed?', 'where is the mouse currently?') while events notify you when something happens ('this button was pressed', 'the mouse has moved').". We will cover events later in the semester.

Section 4: Collision Detection (10 points)

The final required section for all students is to implement some form of collision detection so your shapes can interact. Thinking ahead to what a 2D platformer will require, you need a way to have your character land on platforms. That's your goal for this part of the assignment.

While there is no official tutorial for collision detection in SFML, there are elements of the API that make this relatively straightforward. In particular, the `sf::Shape` has a function to get the bounding rectangles (`sf::ConvexShape::getGlobalBounds()` and `sf::ConvexShape::getLocalBounds()`), and `sf:Rect` objects have an `intersects` function. This will allow you to determine if your shapes intersect.

Section 5: Scaling (25 points, 581 required/481 optional)

For students enrolled in 581, and for those enrolled in 481 who choose to complete it, the task for this part of the assignment is to implement two different modes of scaling.

1. Constant size: this is likely the default behavior, where the shapes on the screen occupy the same number of pixels independent of the size of the window.

2. Proportional: in this mode, the shapes will increase and decrease in size proportional to the change in size of the window.

You must make your window have the `sf::Style::Resize` property. You must also make scaling mode toggle between constant and proportional with a keypress.

Writeup

Now that you have designed and implemented a number of features, write a 1–2 page paper summarizing your design. That is a minimum of 1 FULL single-spaced page. Think creatively about what you have done. Why did you make the design decisions you made? What did they enable for this assignment? What will they enable in the future? What didn't work out the way you had hoped, and why? The most successful writeups will contain evidence that you have thought deeply about these decisions and implementations and what they can produce and have gone beyond what is written in the assignment.

As an appendix to your paper, please include all relevant screenshots to support your discussion. The appendix does not count toward your 1–2 page requirement. Your points for the writeup will represent 2/5 of the points allocation for each of the above sections (*i.e.*, 16 of the 40 points for the "Drawing Objects" section will come from your writeup).

What to Submit

By the start of class on 09/07/22, please upload to Moodle a .zip file containing your visual studio project (including your source code and build files, in multiple subdirectories if you have different versions for different Sections of the assignment), a README file with compilation and execution instructions, and a pdf of your writeup.