Домашнее задание: Библиотека SFML

Подключение библиотеки

Библиотека SFML (Simple and Fast Multimedia Library) - простая и быстрая библиотека для работы с мультимедиа. Кроссплатформенная (т. е. одна программа будет работать на операционных системах Linux, Windows и MacOS). Позволяет создавать окно, рисовать в 2D и 3D, проигрывать музыку и передавать информацию по сети. Для подключения библиотеки вам нужно скачать нужную версию с сайта: sfml-dev.org. Если вы работаете на Windows с компилятором MinGW, то рекомендуется скачать версию **GCC 5.1.0 TDM**.

Подключение вручную:

Для подключения библиотеки вручную через опции `g++` нужно задать путь до папок `include/` и `lib/` и названия файлов библиотеки, используя опции `-I`, `-L` или `-l`.

```
g++ .\main.cpp -I<путь до include> -L<путь до lib> -lsfml-graphics -lsfml-window -lsfml-system
```

Например так:

```
g++ .\main.cpp -I./SFLL-2.5.1/include -L./SFLL-2.5.1/lib -lsfml-graphics -lsfml-window -lsfml-system
```

Подключение с помощью make (файл Makefile):

Если вы программируйте на Linux/MacOS или на Windows с компилятором MinGW, то можно создать файл под названием `Makefile` в текущей директории. и написать в нём:

```
main.exe:
g++ .\main.cpp -o main.exe -o -I<путь до include> -L<путь до lib> -lsfml-graphics -lsfml-window
-lsfml-system
```

После этого можно будет скомпилировать программу вызвав `make` (на Linux) или `mingw32-make` (на Windows - MinGW)
Пример make-файла можно посмотреть в `classroom_tasks/2sfml/3makefile`

Подключение с помощью cmake (файл CMakeLists.txt):

Система автоматической сборки cmake позволяет собирать большие проекты. Чтобы работать с ней вам нужно её скачать по адресу cmake.org и установить переменную среды `PATH`. Затем нужно создать файл `CMakeLists.txt` в директории вашего проекта и написать в нём:

```
cmake_minimum_required(VERSION 2.8.0)
project(simple_sfml)

# Создадим исполняемый файл по имени simple_sfml из исходного файла main.cpp
add_executable(simple_sfml main.cpp)

# Найдём библиотеку SFML автоматически с компонентами graphics, system и window
find_package(SFML 2.5 COMPONENTS graphics system window)
# Подключим эту библиотеку
target_link_libraries(simple_sfml sfml-graphics sfml-system sfml-window)
```

После этого, проект можно собрать так:

```
cmake -G<генератор> <путь до CMakeLists.txt>
```

Пример make-файла можно посмотреть в папке `classroom_tasks/2sfml/4cmake` и `classroom_tasks/2sfml/5cmake_find_package`

- Соберите проект в папке `0basics`, используя один из приведённых выше способов (предпочтительно - make или cmake).

Работа с библиотекой:

Документация и туториалы по библиотеке SFML можно найти на оффициальном сайте: `https://www.sfml-dev.org/`sfml-dev.org. Пример простой программы, для работы с SFML в папке `1sfml_basics`. Основные классы SFML и их методы:

- `sf::Vector3f, sf::Vector2f, sf::Vector2i` и т. д. Классы для математического вектора с перегруженными операциями. (аналогичные тем, что мы писали на предыдущих занятиях). sfml-dev.org/documentation/2.5.1/classsf_1_1Vector2.php
- `sf::RenderWindow` - класс для окна. sfml-dev.org/documentation/2.5.1/classsf_1_1RenderWindow.php
- `sf::CircleShape` - класс для фигуры - круг. sfml-dev.org/documentation/2.5.1/classsf_1_1CircleShape.php

Задачи:

- **Шарики:**
  В папке `2sfml_balls` лежит пример простой программы, которая рисует множество движующихся шаров. Скомпилируйте и запустите эту программу.
- **Граничные условия - тор:**
  Видоизмените программу так, чтобы шарики при выходе за границы экрана телепортировались к противоположной границе (сохраняя скорость).
- **Граничные условия - стенки:**
  Видоизмените программу так, чтобы шарики отражались от границ.
- **Задача N тел**
  Добавьте гравитационное взаимодействие между шариками. Считайте что масса всех шариков равна 1. Один тонкий момент - если шарики подойдут очень близко к друг другу, то они могут нереалистично разлететься. Чтобы это избежать, ограничьте снизу дистанцию гравитационного взаимодействия.
- **Задача N тел с массой**
  Добавьте разную массу шарикам. При создании шарика масса должна задаваться случайным образом. Масса шарика должна быть пропорциональна площади (квадрату радиуса).
- **Электрические заряды**
  Смоделируйте взаимодействие заряженных частиц. Для этого нужно добавить поле в структуру `Ball`, которое будет определять величину заряда. Эта величина может быть как положительной, так и отрицательной. В начале работы программы заряд должен задаваться случайно. Заряды должны взаимодействовать по закону Кулона. Гравитацией можно пренебречь. Цвета зарядов должны быть различными (красный для положительного зарада и синий для отрицательного, интенсивность цвета - пропорциональна величине заряда).
- **Нажатие мыши**
  События нажатия мыши можно обработать с помощью следующего синтаксиса:

```
if (event.type == sf::Event::MouseButtonPressed)
{
    if (event.mouseButton.button == sf::Mouse::Right)
    {
        std::cout << "the right button was pressed" << std::endl;
        std::cout << "mouse x: " << event.mouseButton.x << std::endl;
        std::cout << "mouse y: " << event.mouseButton.y << std::endl;
    }
}
```

  Внутри цикла `while (window.pollEvent(event))`.
  Видоизмените вашу программу так, чтобы при нажатии левой кнопки мыши в том месте, где находится мышь, создавался бы шарик со средними массой и средним положительным зарядом зарядом. При нажатии правой кнопки мыши должен создаваться шарик с очень большой массой и очень большим положительным зарядом. При аналогичных нажатиях, но с зажатой клавишой Shift, должны создаваться отрицательные заряды.

В качестве решения нужно прислать финальную версию программы (1 файл).