ABERTAY GAME DEVELOPMENT SOCIETY

10-Jan-17

CMP105 Games Programming

Introduction and Window

Dr Paul Robertson
p.robertson@abertay.ac.uk

Module overview

- Provide an introduction to the programming concepts and techniques for developing games
- Topics include
  - Input handling
  - 2D sprites and animated sprites
  - Game logic (game related Maths and collision detection)
  - Audio

Assessment

- Single coursework
- Develop a 2D game
- Not pong!

This week

- SFML
  - What is it
- A window
- Some simple rendering
- Building our framework

SFML

- Simple and Fast Multimedia Library
- Designed to ease the development of games and multimedia applications
- Contains five major components
  - System
  - Window
  - Graphics
  - Audio
  - Networking

SFML

- Is multi-platform
  - Windows, Linux and Mac OS X
  - Working on Android and iOS versions
- Is multi-language
  - We will be working with C++
  - But it has bindings for Java, Ruby, Python, etc
- SFML will help us with creating a window, handling keyboard/mouse events and provide some functions for rendering

Game loop

- A “Game Programming Pattern”
- Almost every game has one
  - Rarely used outside of games
- The heart of the game
- Responsible for
  - Processing user **input**
  - **Update** game entities
  - **Render** the game

```
while (true)
{
    handleInput();
    update();
    render();
}
```

Game loop

- One pass through the game loop is a *frame* or *tick*
- The loop isn’t blocking on input
  - Doesn’t stop and wait for user input
  - Just keeps looping
- How fast is the loop going?
  - Different machines, different answers
  - How do we handle that?

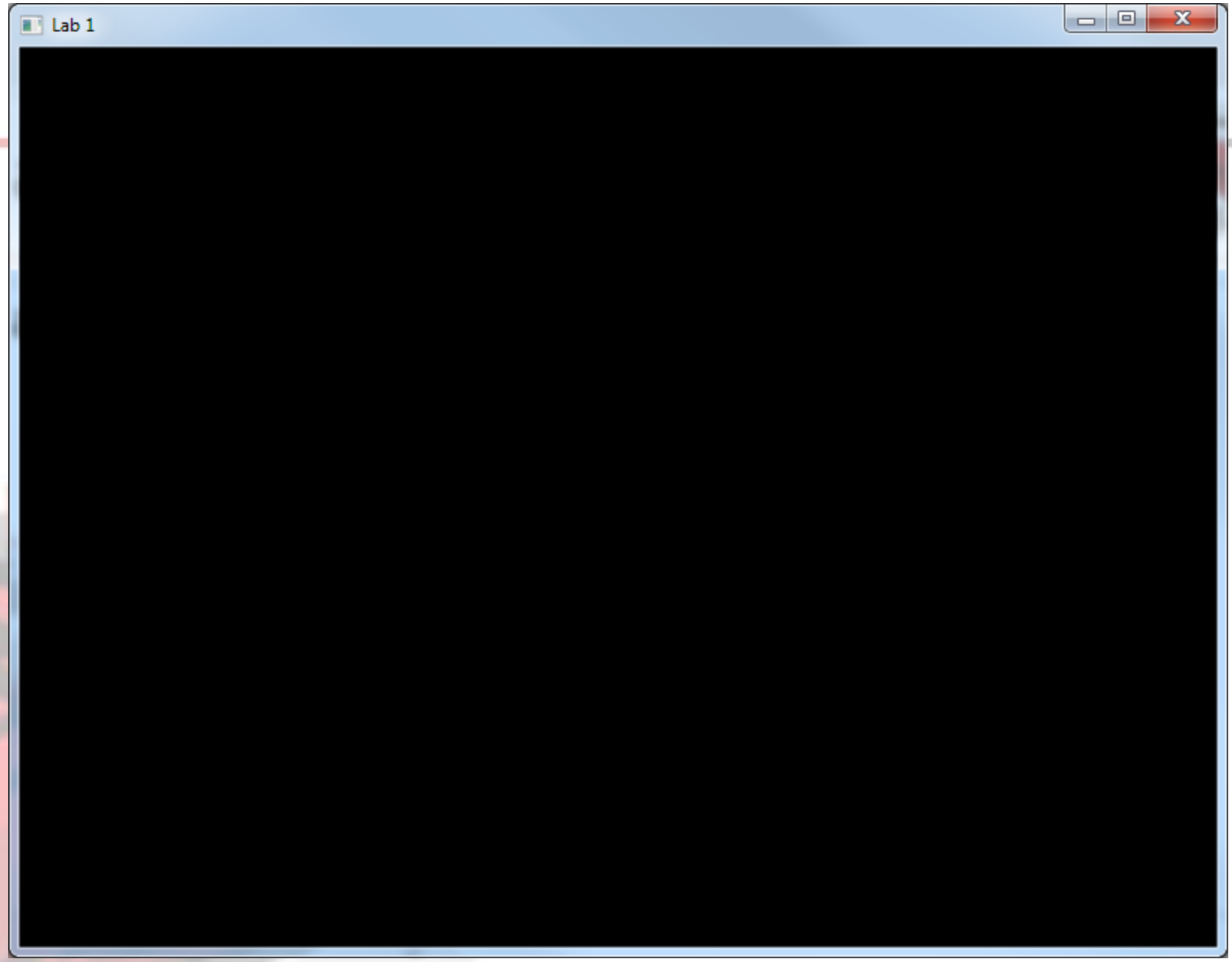
Creating a window

- We need something to contain our graphics / game
- The window will handle communication with the operating system
- The process
  - Create the window
  - Enter the game loop
    - Check for window events
    - Handle user input
    - Update
    - render

```
#include “Game.h”

void main(int argc, char** argv[])
{
sf::RenderWindow window(sf::VideoMode(800, 600), "Lab 1");
Game game(&window);

while (window.isOpen())
{
        sf::Event event;
        while (window.pollEvent(event))
        {
                if (event.type == sf::Event::Closed)
                        window.close();
                if(event.type == sf::Event::Resized)
                        window.setView(sf::View(sf::FloatRect(0, 0, event.size.width, event.size.height)));
        }

        game.handleInput();
        game.update();
        game.render();

}

}
```

Window events

- In order for different parts of the OS to communicate. A messaging system is employed
- In this way separate systems do not need to ‘know’ about each other but only know about the messages they can send and receive
- Message include
  - Window being opened/closed
  - Window being moved/resized
  - Lost/gain of focus
  - Keyboard and mouse events
  - The list goes on


Lab 1

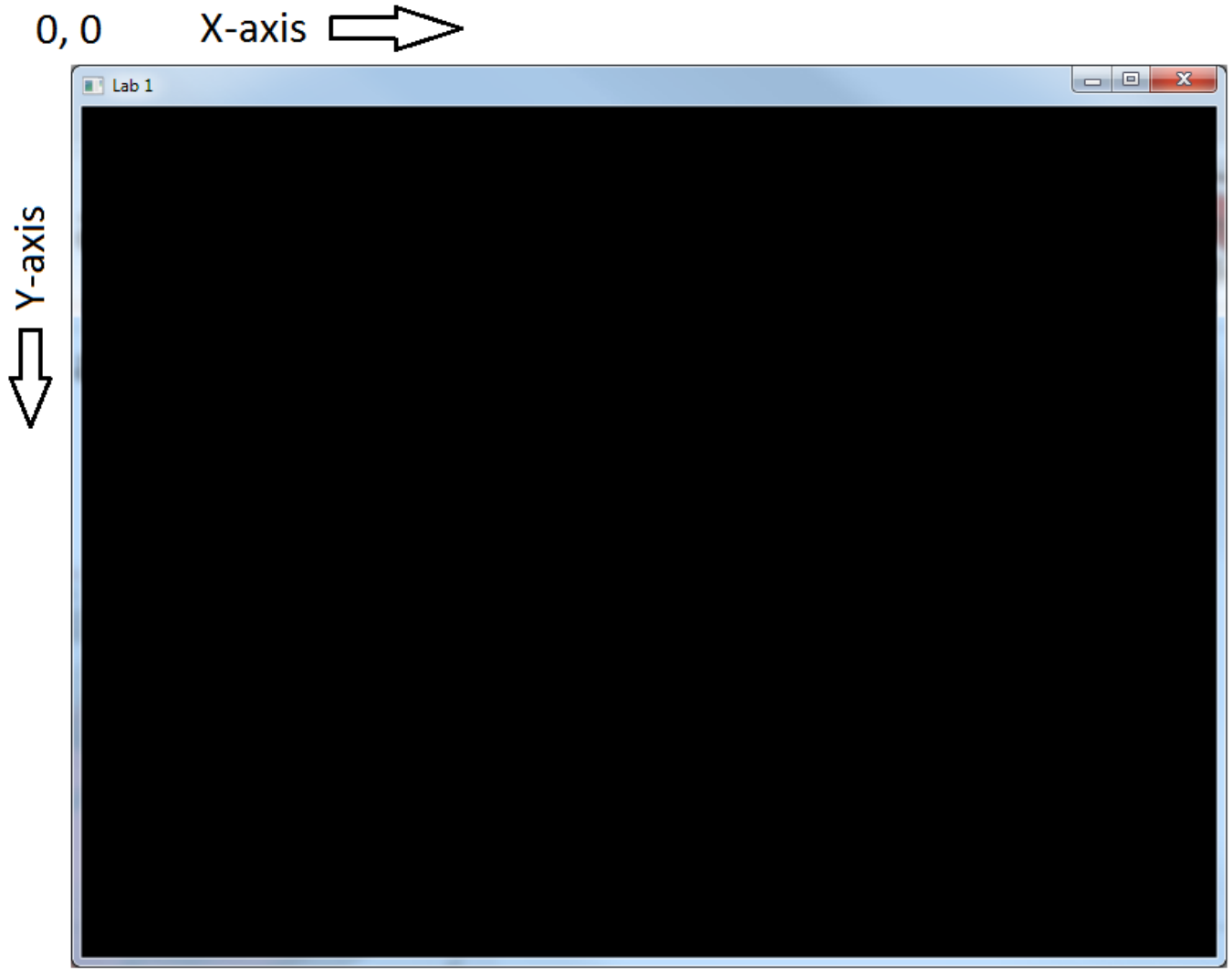
Rendering

- New class that will contain our game/scene/level
- Main.cpp creates/handles the window and game loop
- Game class will handle relevant user input, update game objects and render game objects
- For example rendering some basic geometry
  - A red rectangle
  - A green circle with a magenta outline

- Render order is important

Position

- The origin of the window is the top left corner
- When rendering objects will be placed in relation to this



Game.h

```h
#pragma once

#include <SFML/Graphics.hpp>

class Game {
public:
    Game(sf::RenderWindow* hwnd);
    ~Game();

    void handleInput();
    void update();
    void render();

private:
    sf::RenderWindow* window;
    void beginDraw();
    void endDraw();

    // Game Variables
    sf::RectangleShape rect;
    sf::CircleShape circle;

};
```

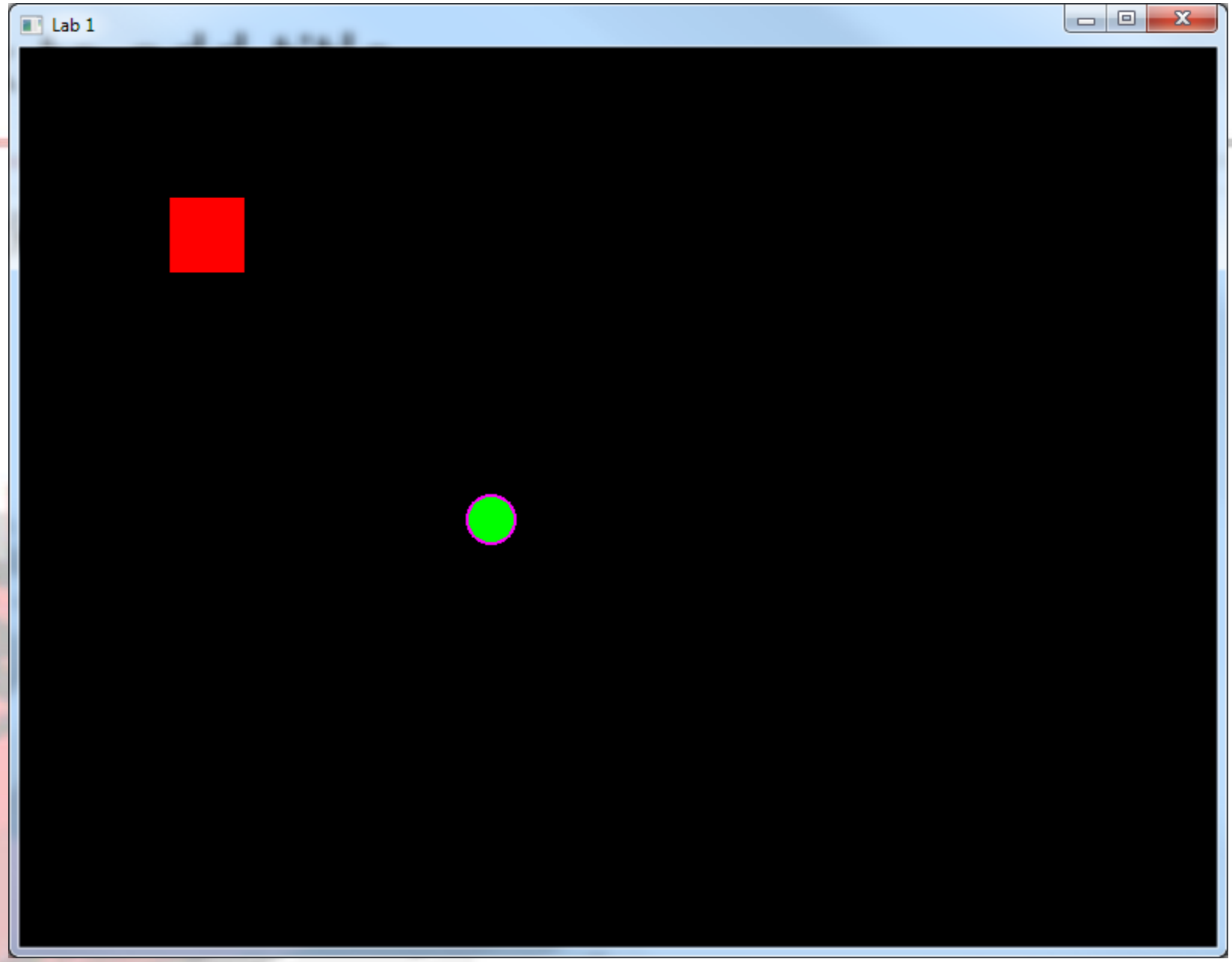
Game.cpp

```cpp
#include "Game.h"

Game::Game(sf::RenderWindow* hwnd)
{
    window = hwnd;

    rect.setSize(sf::Vector2f(50, 50));
    rect.setPosition(100, 100);
    rect.setFillColor(sf::Color::Red);

    circle.setRadius(15);
    circle.setPosition(300, 300);
    circle.setFillColor(sf::Color::Green);
    circle.setOutlineColor(sf::Color::Magenta);
    circle.setOutlineThickness(2.f);

}
```

Game.cpp

```cpp
void Game::render()
{
    beginDraw();

    window->draw(rect);
    window->draw(circle);

    endDraw();
}
```

```cpp
void Game::beginDraw()
{
    window->clear(sf::Color::Black);
}

void Game::endDraw()
{
    window->display();
}
```


Lab 1

The framework

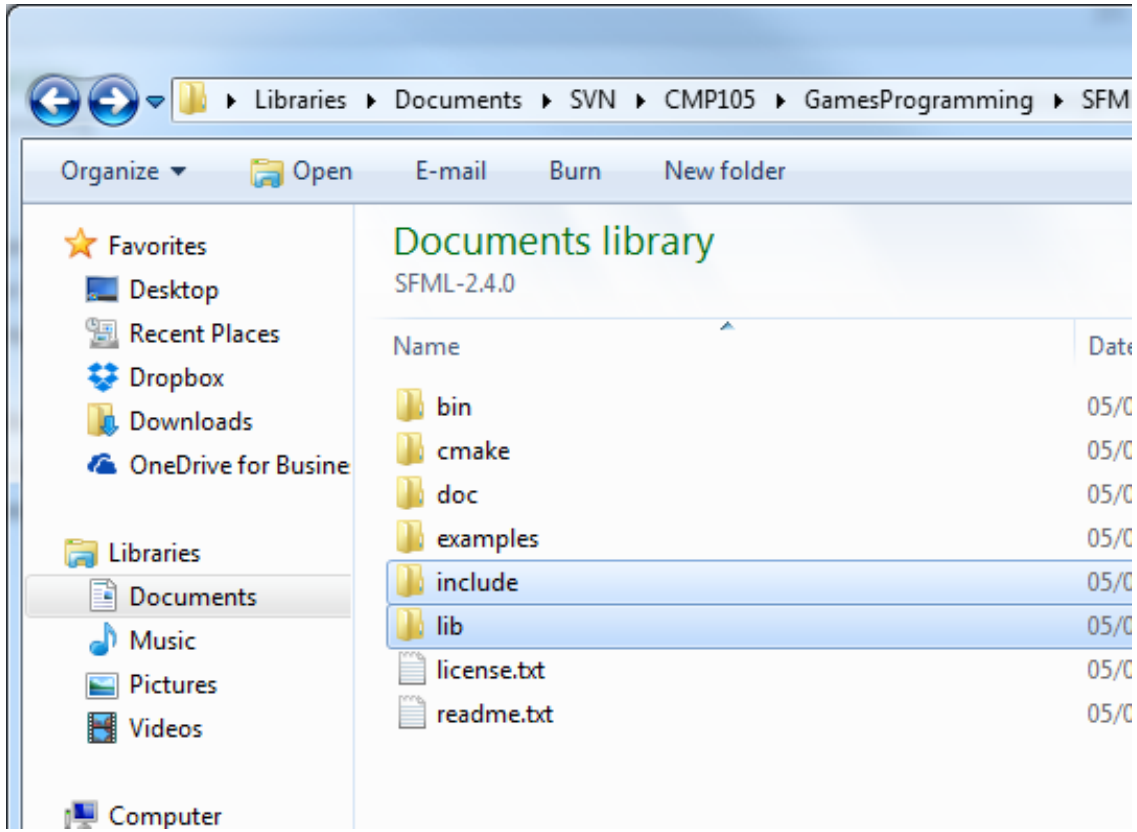
- This is the beginning of our framework
  - Main (game loop)
  - Game (scene/level)
  - Link to SFML
- We will be adding to it as the module progresses

Adding libraries

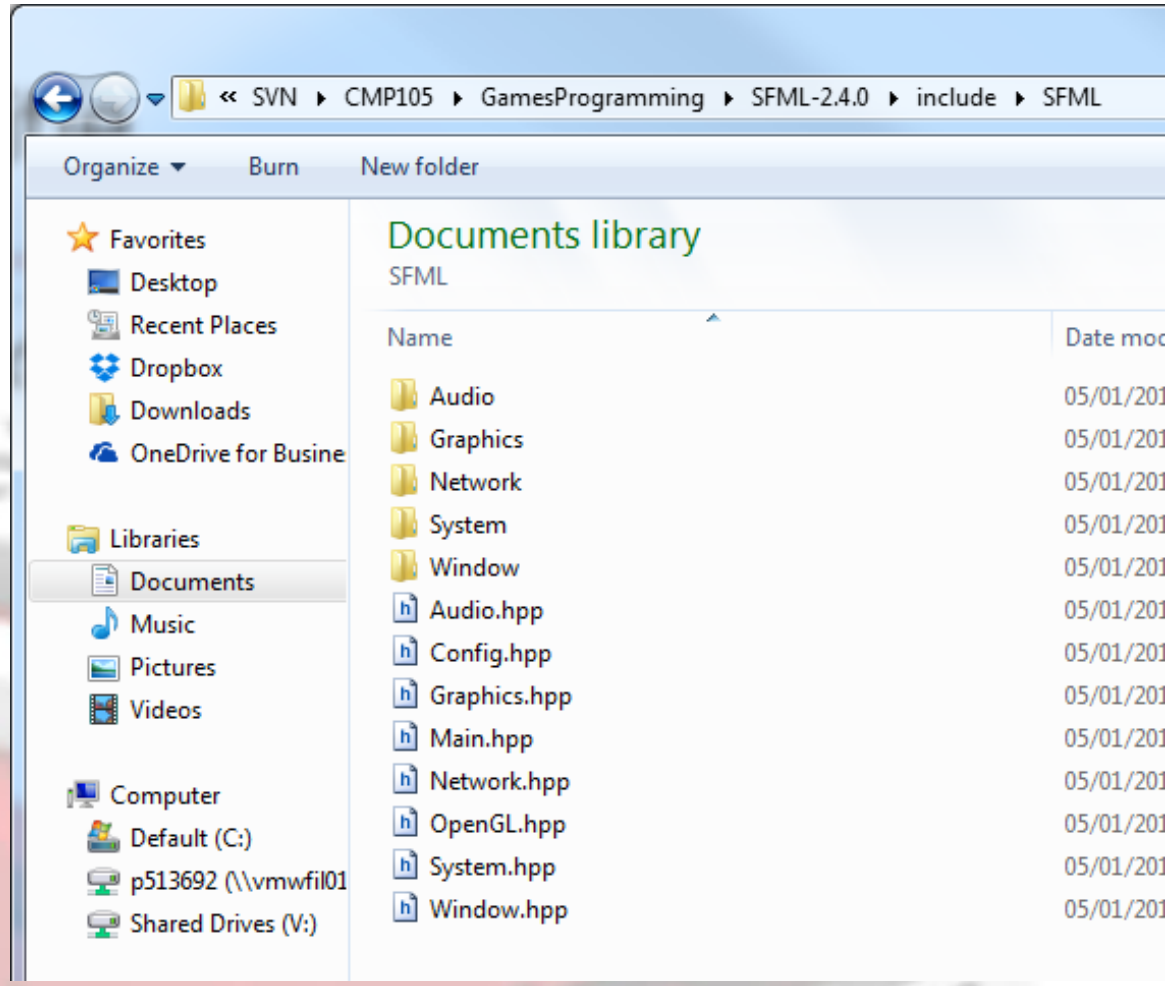
- Linking to SFML
  - Our application needs access to SFML functions, objects, etc
- **Standard** libraries are provided by the OS
  - We can safely assume other machines with the same OS have the same libraries
- For additional libraries we will have to inform the compiler about them
  - And provide a copy of the library for the compiled executable

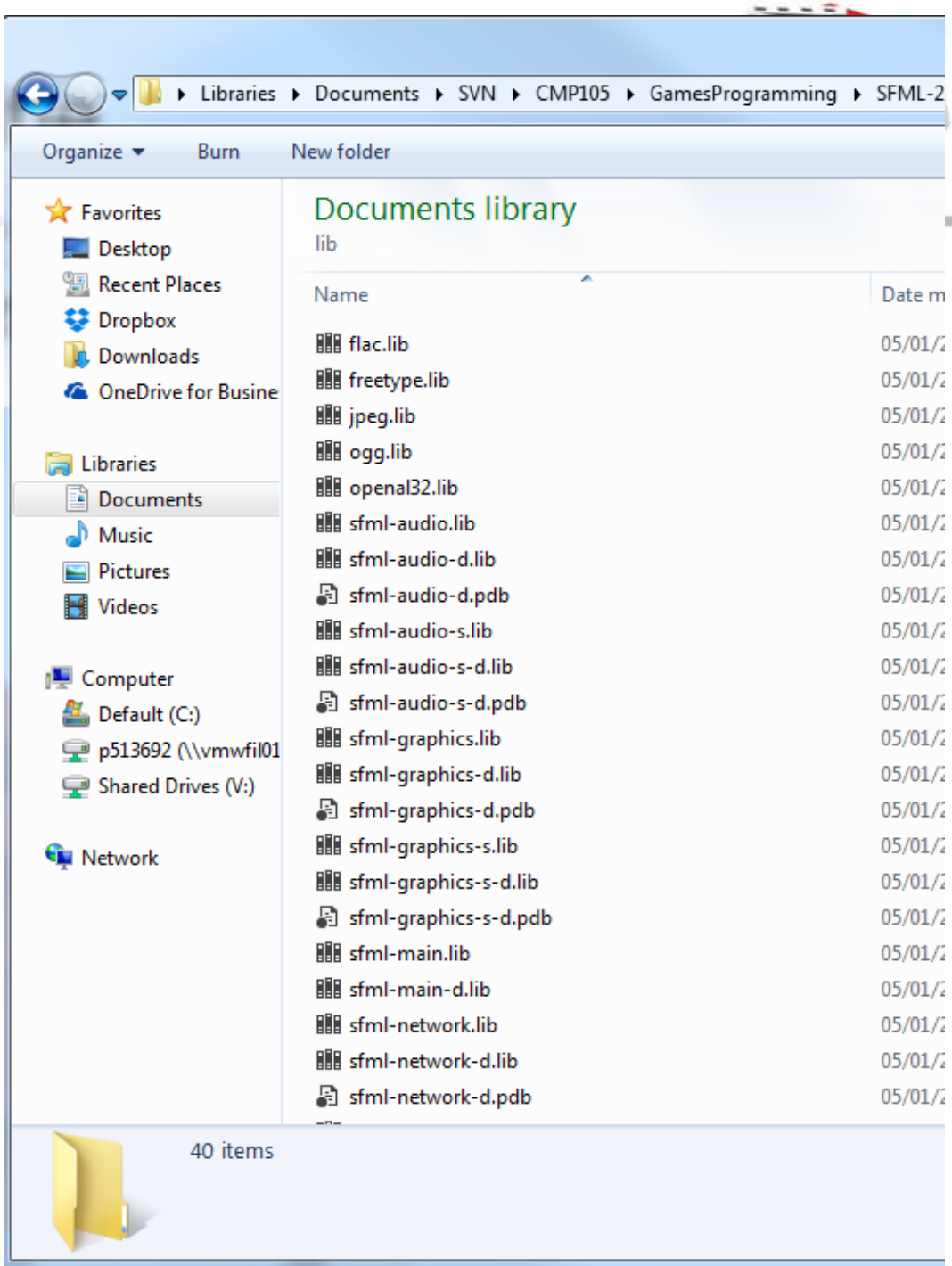
SFML structure

- Important directories are
  - Bin
    - Dll files for runtime
  - Include
    - Headers file for development
  - Lib
    - Libraries for development
- Other directories include
  - Examples, documentation



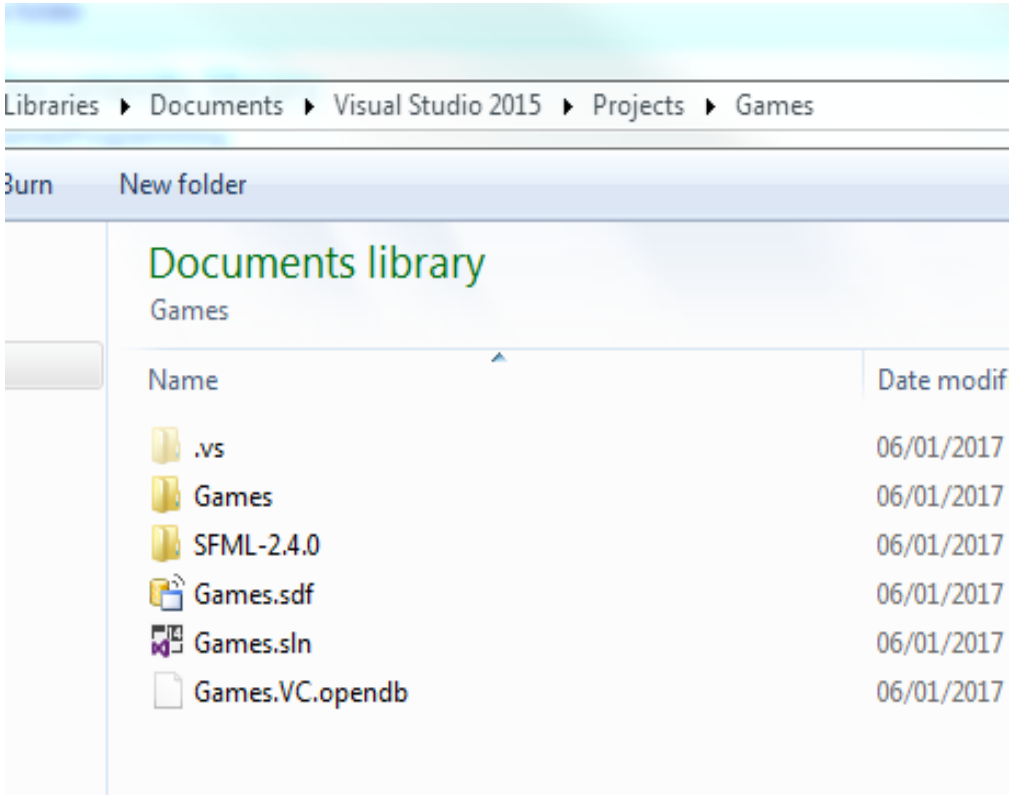
SFML structure



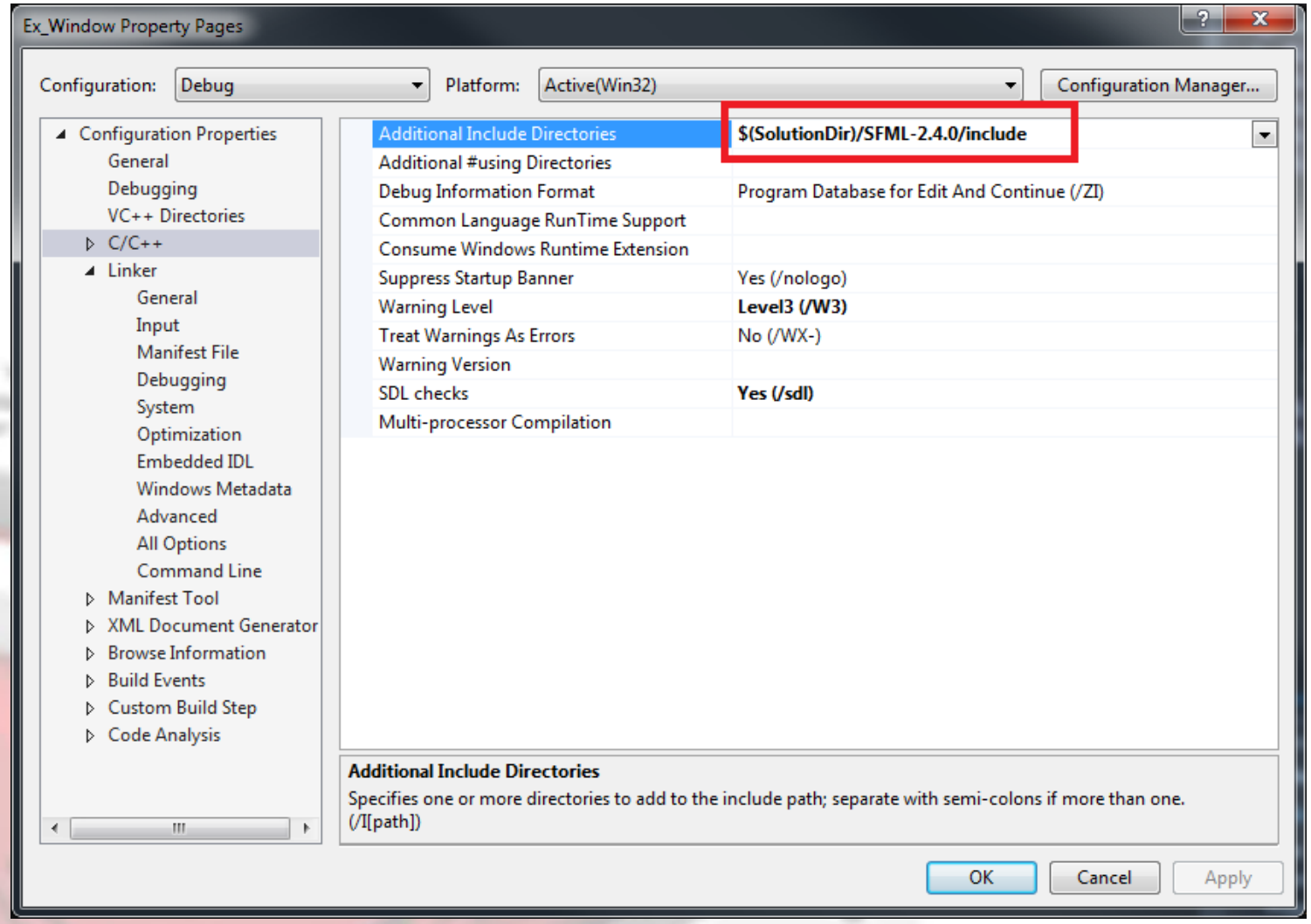


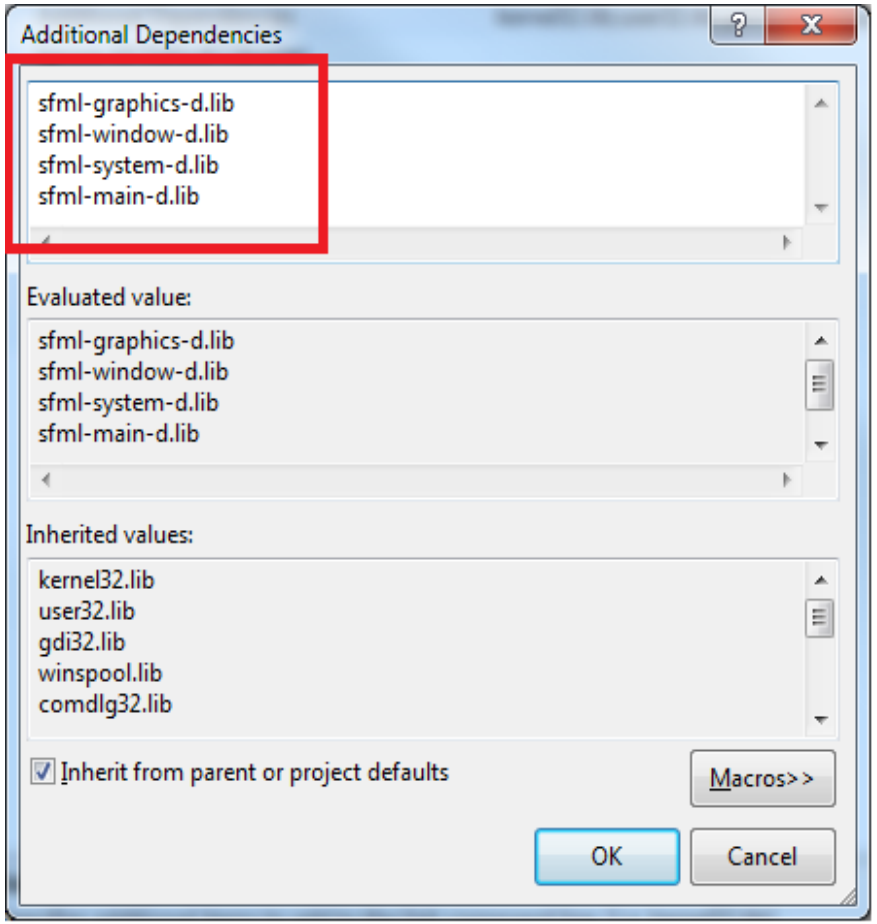
Adding SFML to the project

- Copy the SFML folder into the workspace / solution directory
  - If we had multiple projects they could all access the libraries
- Then we need to tell the compiler / visual studio where the libraries etc can be found
  - And which ones we are using



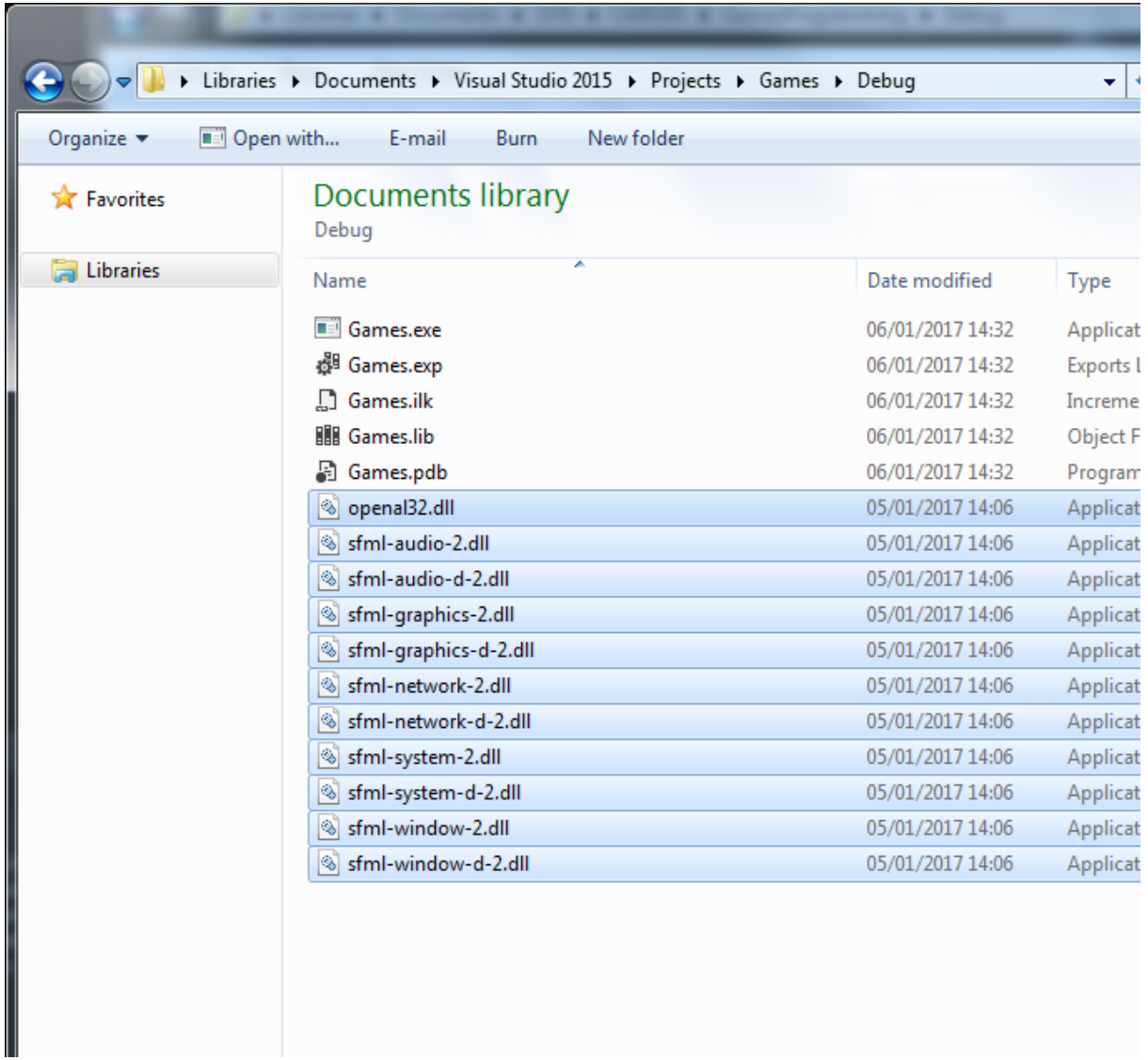
Adding SFML to the project





Add SFML to the project

- Copy the dlls into the Debug folder
  - This allows the executable to access the libraries used in development
  - Machine independent
- Similar will need done for the release version



In the labs

- Setting up the project
- The game loop
- Rendering basic geometry

- Helper – Christopher Acornley

- SFML documentation
  - http://www.sfml-dev.org/documentation/2.4.1/