

Input

CMP105 Games Programming

From the lab

- Vectors
 - Vector2f / 2u / 2i / etc
 - Stores two values **X** and **Y**
 - Represents a 2D vector
 - sf::Vector2f Position;
 - Position.x = 5.0f;

This week

- Window messages
- Building an input class
- Handling input
 - Keyboard
 - Mouse

Input

- We only need to worry about processing input from our main windows
 - Dialogues / Message boxes handle all input from mouse and keyboard behind the scenes
- What sends Window Messages?
 - Keyboard and Mouse Input
 - Key presses
 - Mouse movement
 - Changes made to the window
 - Gained/lost focus
 - Resize
 - etc

Messages

- Every Windows Message has 4 components
 - WHO
 - Handle to the object that sent the message
 - HWND
 - WHAT
 - A message identifier, a positive natural number
 - UINT
 - Data
 - Could be anything and comes in two parts
 - WPARAM = WORD parameter (Uint)
 - LPARAM = LONG parameter (Long)
 - Contents depend on message

Messages

- SFML handles default processing of messages
 - Over 220 Window messages
- Luckily we don't need to process all these messages
 - Focus only on the ones we are interested in
 - Let the default processes handle the rest
 - If we leave the messages unhandled

Basic Message Handling

- We current already handle the window being closed and resized

```
sf::Event event;
while (window.pollEvent(event))
{
    switch (event.type)
    {
        case sf::Event::Closed:
            window.close();
            break;
        case sf::Event::Resized:
            window.setView(sf::View(sf::FloatRect(0.f, 0.f, (float)event.size.width, (float)event.size.height)));
            break;
    }
}
```

Keyboard events/messages

- Two messages
 - Sf::Event::KeyPressed
 - Sf::Event::keyReleased
 - Event data will contain which key the event is related to
- We need to listen for these events and store which key has been pressed/released
- There is another option
 - We can interrogate the hardware to get the current state of a key
 - However this causes issues with some types of input
 - And does not halt on loss of focus

Keyboard

- Add to the switch statement
- Pass key code into an Input class (we will build this)

```
break;
case sf::Event::KeyPressed:
    // update input class
    input.setKeyDown(event.key.code);
    break;
case sf::Event::KeyReleased:
    //update input class
    input.setKeyUp(event.key.code);
    break;
```

Storing keyboard input

- Each key has a corresponding enum
 - This is part of the event
 - `Event.key.code`
 - Each key enum value can be evaluated to an integer
 - `Sf::Key::Space = 57`
- Traditionally, in Windows this is handled as an ASCII value

Storing keyboard input

- We can represent keys as numbers this means it is easy to store information on them
- Store key presses in a Boolean array
 - Bool keys[256];
 - True == pressed/down
 - False == released/up
- When a key is pressed set value to true, when released set to false
- Can check state of the key by querying the array for Boolean values
- Alternative would be a massive collection of *if statements*
 - This is bad programming

0	1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I	J
FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE

Capture input

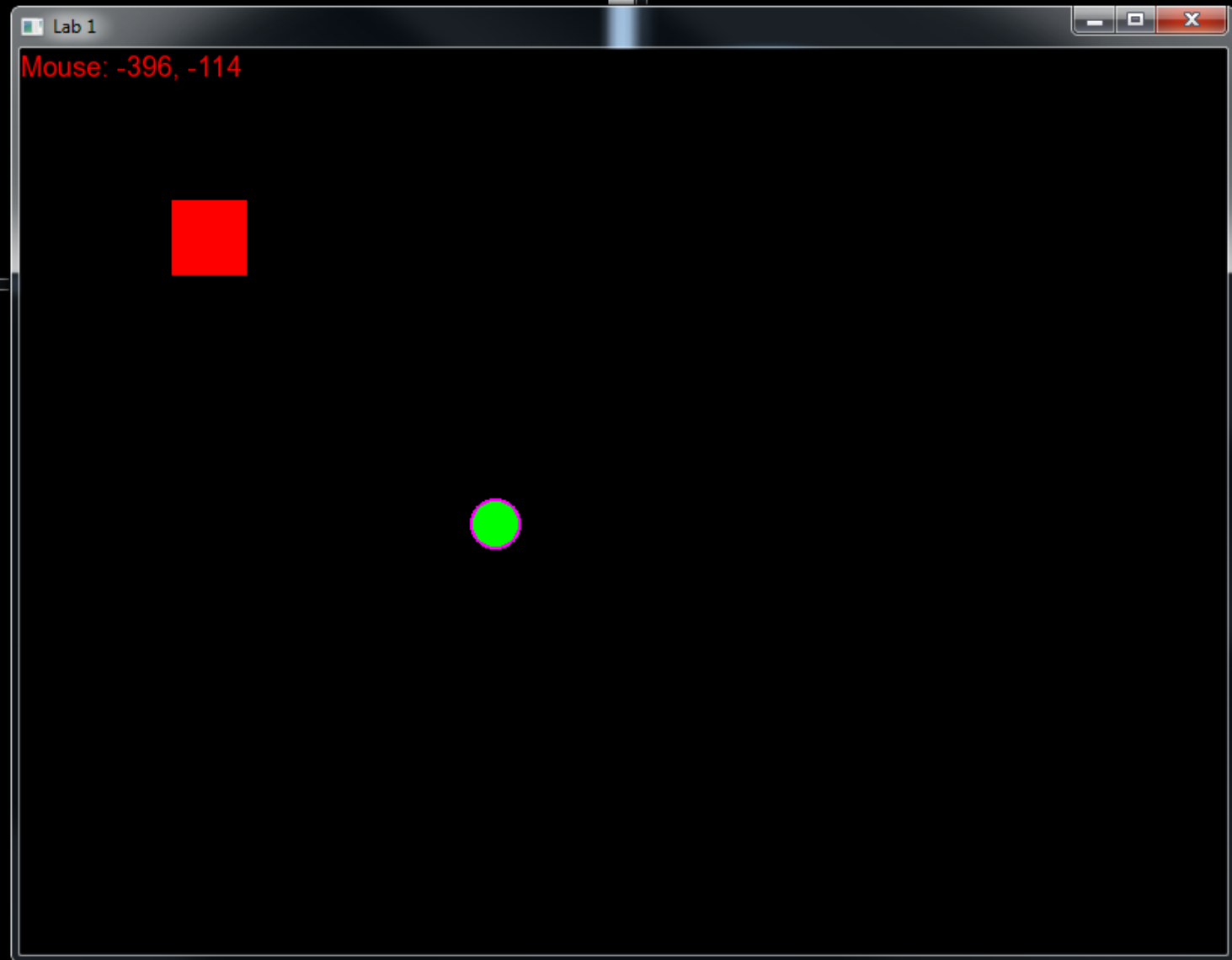
- The main game loop has two processes
 - Window has received a message and needs to process it
 - Little actual processing should be done in here
 - Handle the message/event, storing key presses, mouse movement etc
 - Handled by other classes later
 - Once all messages have been processed
 - We perform all of our game processing /rendering etc
 - Here we can take action on keyboard input and check the “keys” array
 - Pass the input class to other objects for input handling

Key press example

- This takes place after processing all Window messages

```
void Level::handleInput()
{
    // if space is pressed output to console
    if (input->isKeyDown(sf::Keyboard::Space))
    {
        input->setKeyUp(sf::Keyboard::Space);
        std::cout << "Space was pressed\n";
    }
}
```

```
\\vmwfil01\HomeDrives67$\p514846\Teaching\2017-18\CMP105 Games Programming\My Code\...  
Space was pressed
```



Mouse Input

- Mouse events
 - `Sf::Event::MouseButtonPressed`
 - `Sf::Event::MouseButtonReleased`
 - `Sf::Event::MouseMove`
- Event data includes which mouse button or current mouse position
- It is also possible to detect scroll wheel events

Storing Mouse input

- Similar to key presses
 - Boolean values to track if mouse button is pressed or released
 - `bool` left;
 - `bool` right;
 - `bool` middle;
 - Set Boolean variables to true when button is pressed and false when the mouse button is released
 - Process in similar method as key presses, check variable state if true do something (display message box, etc.)

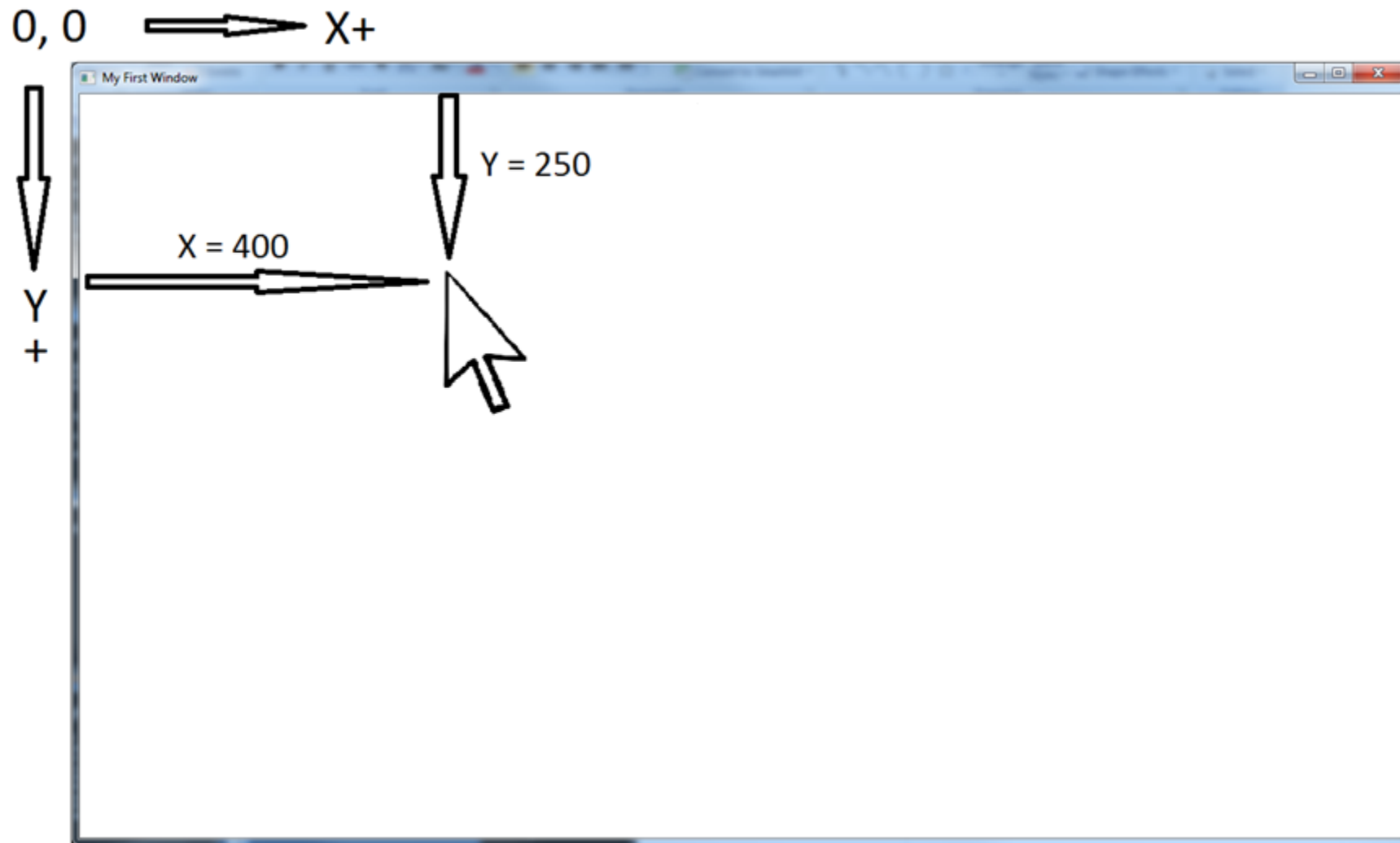
Storing Mouse input

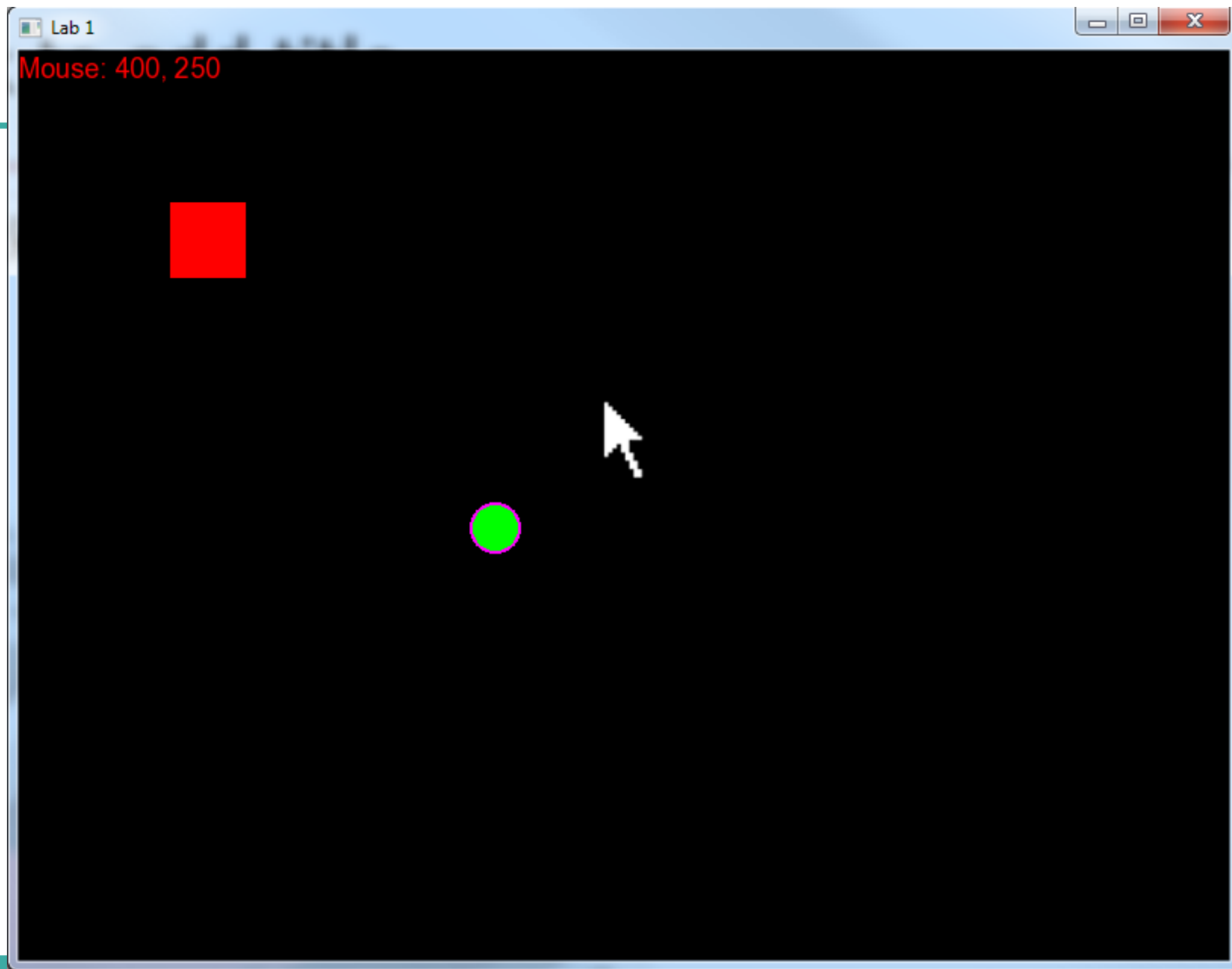
- Mouse Moved
 - Called every time the mouse has moved
 - Event data stores current mouse position
 - Can track the mouse position either in relation to the window or the desktop
- Input class needs to store X and Y position of mouse (integer)
 - We are interested in the mouse's position in relation to the window

Store Mouse input

```
break;
case sf::Event::MouseMove:
    //update input class
    input.setMousePosition(event.mouseMove.x, event.mouseMove.y);
    break;
case sf::Event::MouseButtonPressed:
    if (event.mouseButton.button == sf::Mouse::Left)
    {
        //update input class
        input.setMouseLeftDown(true);
    }
    break;
case sf::Event::MouseButtonReleased:
    if (event.mouseButton.button == sf::Mouse::Left)
    {
        //update input class
        input.setMouseLeftDown(false);
    }
    break;
```

Coordinates reminder





References and Pointers

- A reference (&) obtains the address (memory location) of a variable
 - “address of”
 - `&myVar;`
- A variable that stores an address of another variable is a pointer
 - `myPointer = &myVar;`

References and Pointers

- To access the variable that a pointer points to we need to dereference operator (*) the pointer
 - “value pointed to by”
 - `myNextVar = *myPointer;`
- To dereference a function call
 - `myPointer->getSome()`
 - Not `myPointer.getSome()`

Practical example

- In main.cpp
 - We work with object directly
- In Level
 - We work with a pointer

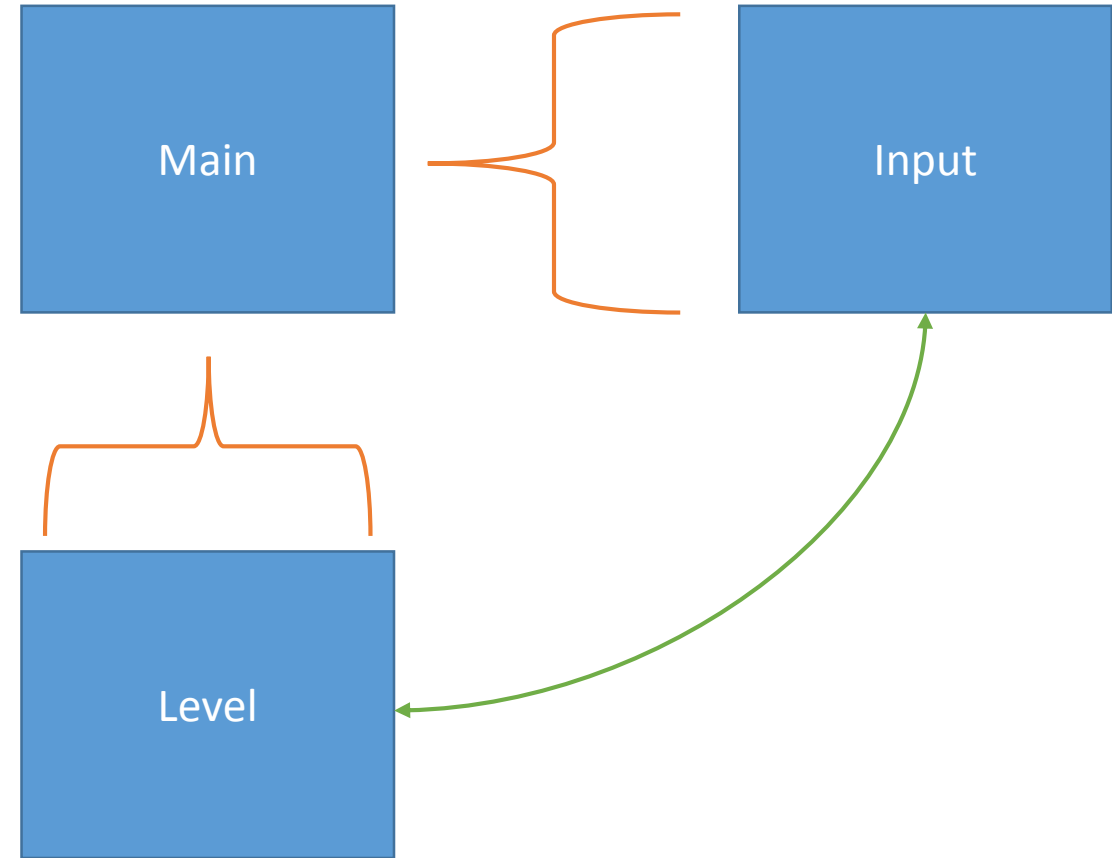
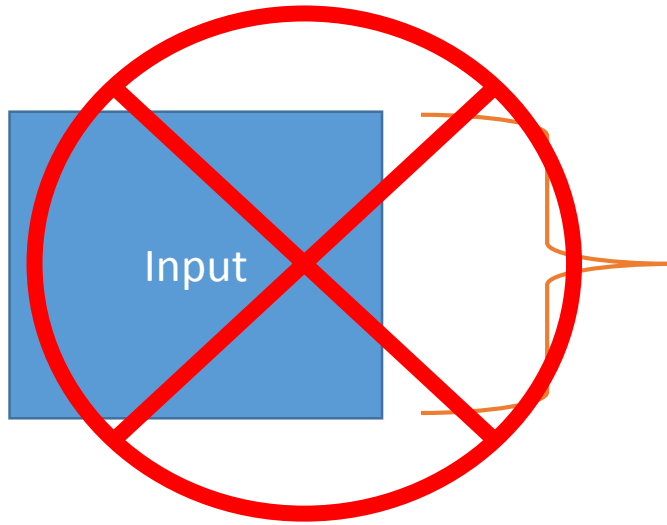
```
Input input;  
Game game(&window, &input);
```

```
sf::RenderWindow* window;  
Input* input;
```

```
if (input->isKeyDown(sf::Keyboard::Q))  
{  
    // reposition mouse cursor  
    sf::Mouse::setPosition(sf::Vector2i(400, 300), *window);  
}
```

Structure

- Main **contains** input object
- Main **contains** level object
- Level has a **reference** to input



Bring it all together

- We have seen how to store and process keyboard and mouse events
- Now we need to bring this all together into a working application
- The best way to do this is to create an Input Class
 - An object that will handle all our input needs
 - This object can be passed into multiple objects which can handle their specific input needs
 - This means creating two new files (and new class)
 - Input.h
 - Input.cpp

Input.h

- Requires
 - Mouse struct
 - Boolean array for tracking keys
 - Bool keys[256];
 - Functions
 - Getters and setters for key press/release
 - Return if specific key is pressed
 - Similar for mouse + mouse position

```
#pragma once
```

```
class Input
```

```
{
```

```
private:
```

```
    struct Mouse
```

```
{
```

```
    int x, y;
```

```
    bool left;
```

```
};
```

```
public:
```

```
    void setKeyDown(int key);
```

```
    void setKeyUp(int key);
```

```
    bool isKeyDown(int key);
```

```
    void setMouseX(int lx);
```

```
    void setMouseY(int ly);
```

```
    void setMousePosition(int lx, int ly);
```

```
    int getMouseX();
```

```
    int getMouseY();
```

```
    // Some functions missing. You will need to add these.
```

```
private:
```

```
    bool keys[256]{ false };
```

```
    Mouse mouse;
```

```
};
```

Input.cpp (not all functions)

```
#include "Input.h"

void Input::setKeyDown(int key)
{
    if (key > 0)
    {
        keys[key] = true;
    }
}

void Input::setKeyUp(int key)
{
    if (key > 0)
    {
        keys[key] = false;
    }
}

bool Input::isKeyDown(int key)
{
    return keys[key];
}
```

```
void Input::setMouseX(int lx)
{
    mouse.x = lx;
}

void Input::setMouseY(int ly)
{
    mouse.y = ly;
}
```

Adding the Input class

- Need to include a few files (in Level.h)
 - `#include "Input.h"`
- Create Input object in Main.cpp
- Pass a reference to the input object into Game object (requires updating game object)

```
// Define game objects
Input input;
Level level(&window, &input);
```

Process events

- Add to the events loop and capture keyboard and mouse events
- Updating the input class as needed

```
sf::Event event;
while (window.pollEvent(event))
{
    switch (event.type)
    {
        case sf::Event::Closed:
            window.close();
            break;
        case sf::Event::Resized:
            window.setView(sf::View(sf::FloatRect(0.f, 0.f, (float)event.size.width, (float)event.size.height)));
            break;
        case sf::Event::KeyPressed:
            // update input class
            input.setKeyDown(event.key.code);
            break;
        case sf::Event::KeyReleased:
            //update input class
            input.setKeyUp(event.key.code);
            break;
        case sf::Event::MouseMoved:
            //update input class
            input.setMousePosition(event.mouseMove.x, event.mouseMove.y);
            break;
    }
}
```

Handle Input

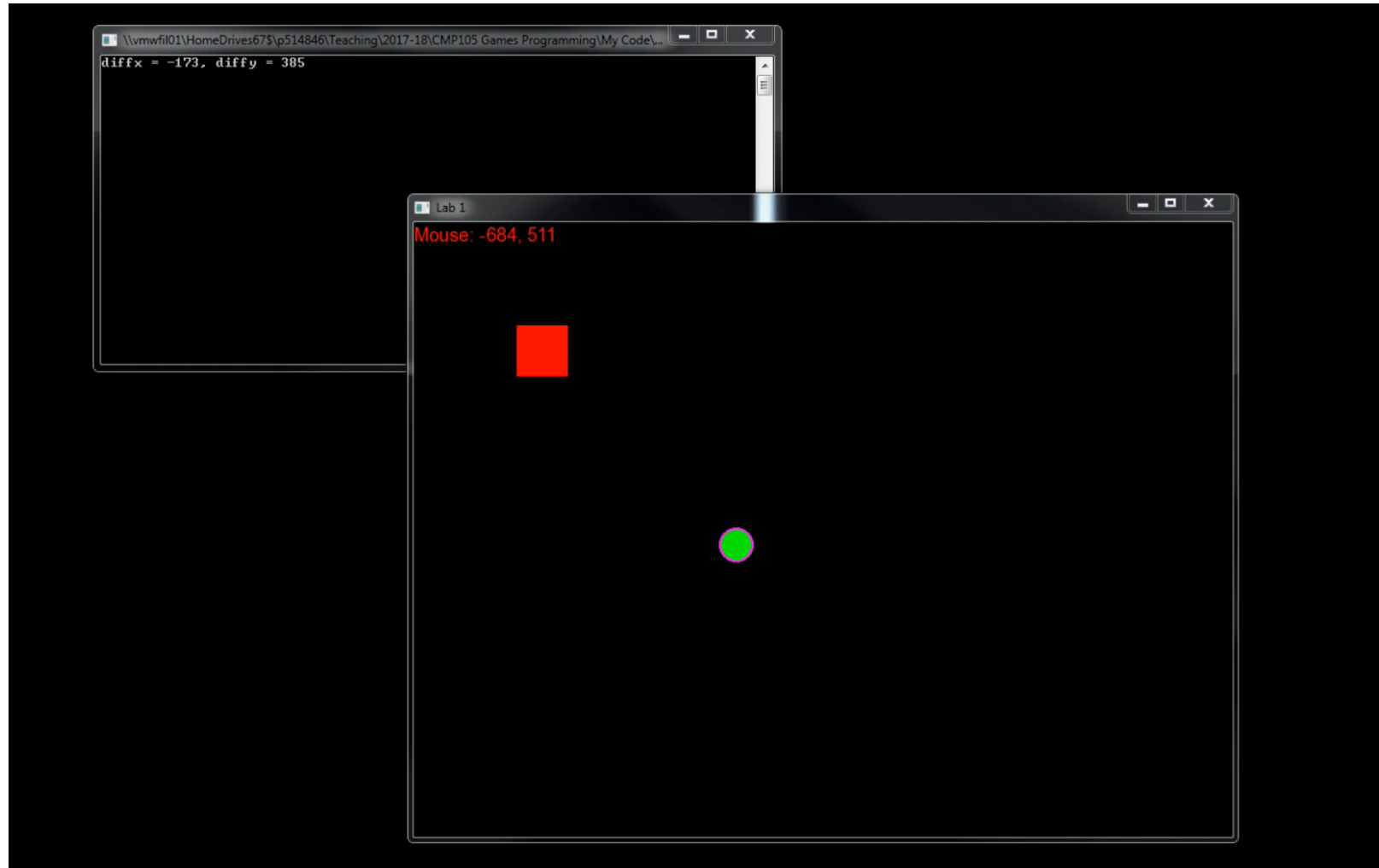
- Where
 - Level class, `handleInput()` function
 - Level/Scene related inputs
- Additionally
 - Pass a reference to the input class to another object for further processing
 - E.g. a player class would contain code on handling inputs related to player control

Handle input

- Inside the level class
 - Detects the space key being pressed
 - Forces the key up
 - Outputs message

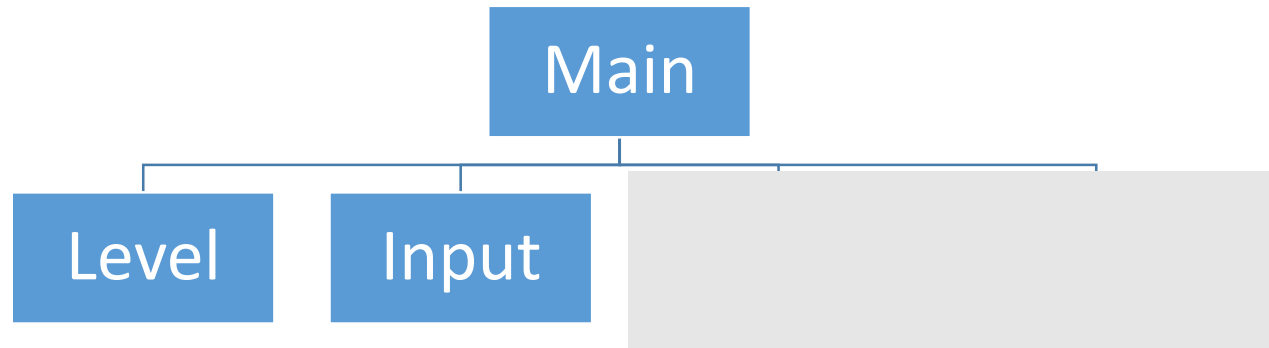
```
void Level::handleInput()
{
    // if space is pressed output to console
    if (input->isKeyDown(sf::Keyboard::Space))
    {
        input->setKeyUp(sf::Keyboard::Space);
        std::cout << "Space was pressed\n";
    }
}
```


Example



The framework

- Added input
 - Main
 - Level
 - Could be menu, splash screen, credits, etc
 - **Input**
 - Track keyboard and mouse input
- Link to SFML



In the labs

- Building input class
- Doing stuff with keyboard and mouse
 - Detecting key presses
 - Mouse movement and button presses
- Some reading on pointers
 - <http://101.lv/learn/C++/htm/ch08.htm>