CMP105 Games Programming

Double buffering and delta time

This week

- Pointers
- Single buffering
- Double buffering
- Delta time
- Example

References and Pointers

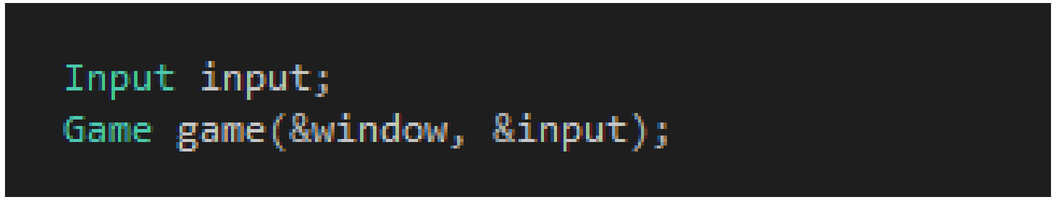
- A reference (&) obtains the address (memory location) of a variable
  - “address of”
  - &myVar;
- A variable that stores an address of another variable is a pointer
  - myPointer = &myVar;

References and Pointers

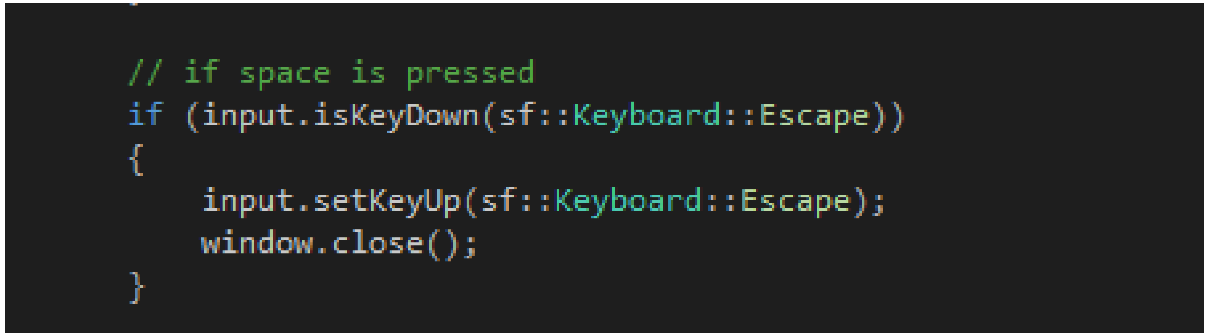
- To access the variable that a pointer points to we need to dereference (*) the pointer
  - “value pointed to by”
  - myNextVar = *myPointer;
- To dereference a function call
  - myPointer->getSome()
  - Not myPointer.getSome()

Practical example

- In main.cpp
  - We work with object directly

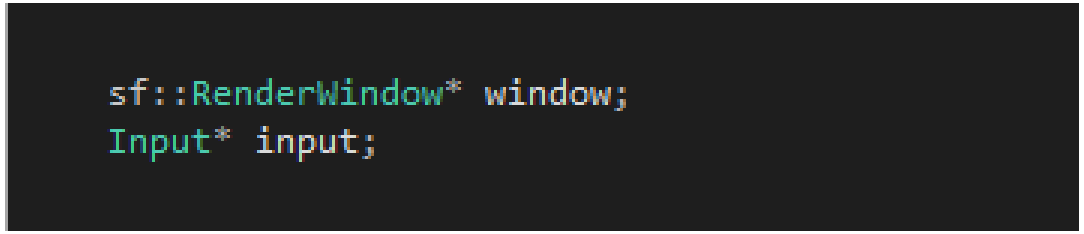


```
Input input;
Game game(&window, &input);
```

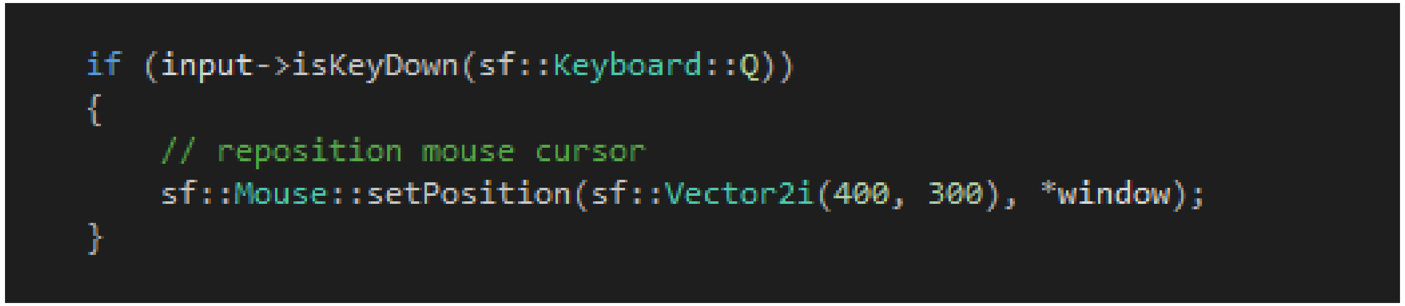


```
    // if space is pressed
    if (input.isKeyDown(sf::Keyboard::Escape))
    {
        input.setKeyUp(sf::Keyboard::Escape);
        window.close();
    }
```

- In Game
  - We work with a pointer



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

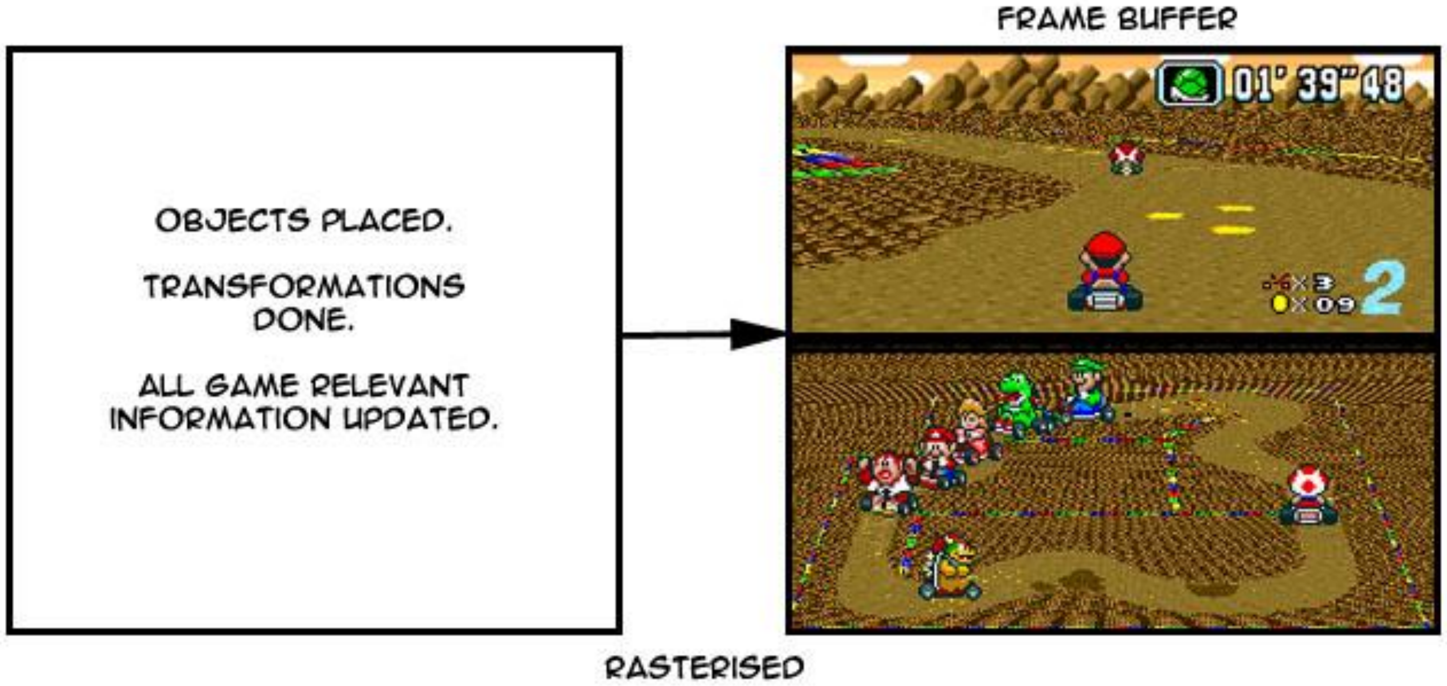


```
if (input->isKeyDown(sf::Keyboard::Q))
{
    // reposition mouse cursor
    sf::Mouse::setPosition(sf::Vector2i(400, 300), *window);
}
```

Buffers

Single buffering

- How do we actually get graphics to the screen?
- Create the scene -> Raster into a frame buffer



OBJECTS PLACED.

TRANSFORMATIONS DONE.

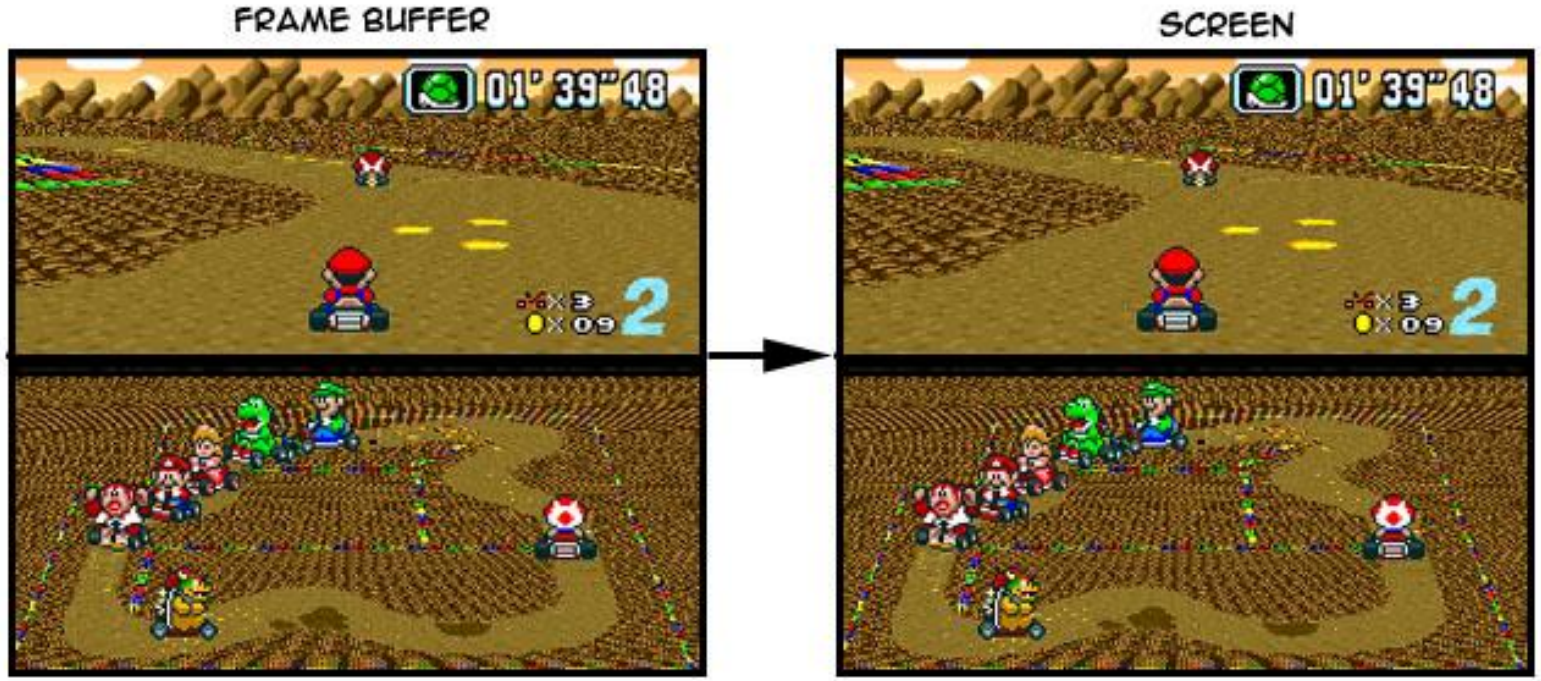
ALL GAME RELEVANT INFORMATION UPDATED.

FRAME BUFFER

RASTERISED

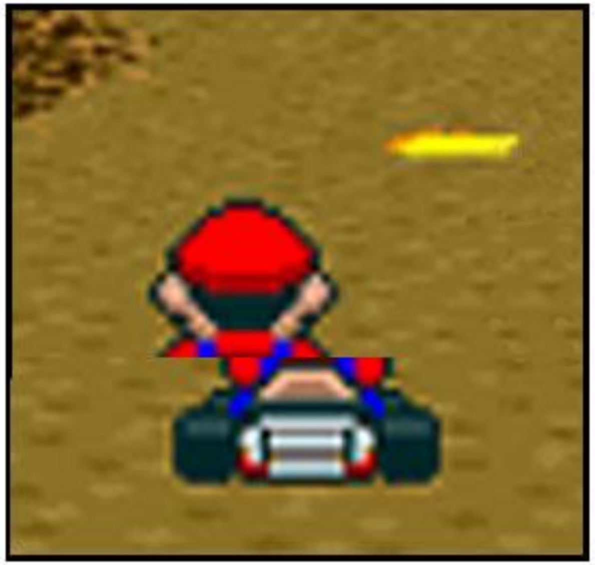
Single buffering

- The frame buffer



Single buffering

- Problems
  - What if the contents of the buffer are changed (being updated) as the scene is displayed on screen?
  - This is known as "tearing"



TEARING

Single buffering

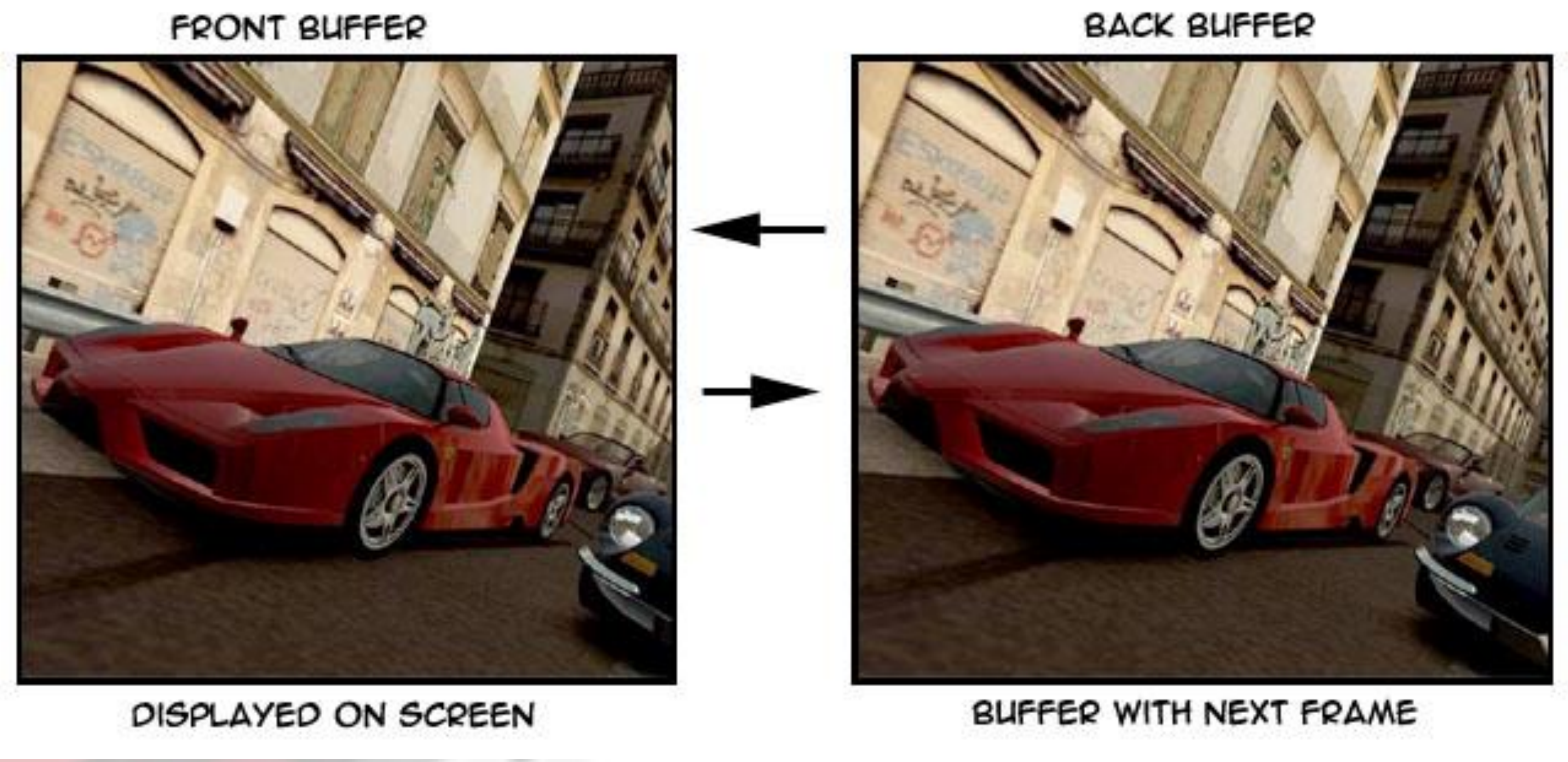
- Solution
  - Do not display the frame until it has been fully created
  - Do not begin creating the frame until the previous one has been displayed
  - This is really slow as the next operation is constantly waiting on the previous one to be completed
  - Can cause trouble if there is a lot in the scene

Double buffering

- Fixes the problems of single buffering
- Almost still universally used today in 2D and 3D games
  - Supported in virtually every type of graphics hardware available
- There is also triple/multi buffering
  - Not needed right now
  - Double buffering still predominantly used

Double buffering

- TWO frame buffers instead of one



Double buffering

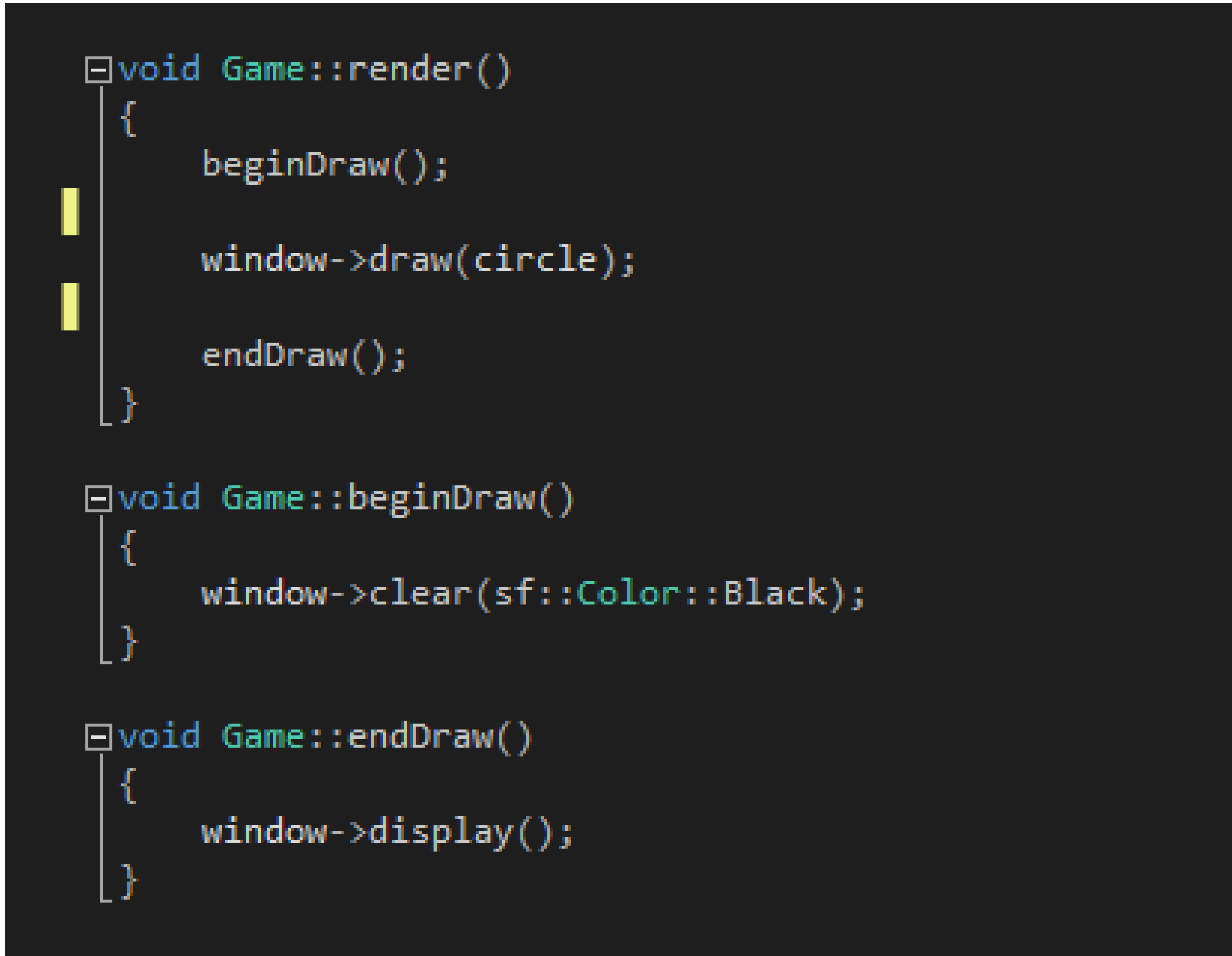
- Concept
  - Frame is rendered into the back buffer
  - Contents of front and back buffers are swapped
  - Frame is displayed from the front buffer while the next frame is created in the back buffer
  - Front buffer is cleared and the Buffers are swapped again
  - “Frame Rate” is the amount of time we swap buffers per-second

Double buffering

- Benefits and Drawbacks
  - Tearing can still occur
    - but is much reduced
    - can be removed by synching the buffer swap with the monitor refresh. Usually a setting for this on most GFX cards
  - Only other drawback is that double the memory is required for the frame buffers (trivial on today's graphics cards)

SFML and double buffering

- The SFML RenderWindow class handles the double buffering
  - Window.clear(colour)
    - Clears the back buffer ready for drawing
  - Window.draw(object)
    - Renders an object on the back buffer
  - Window.display()
    - Swap buffers
    - Pushes the back buffer onto the frame/front buffer



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

    window->draw(circle);

    endDraw();
}

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

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

Delta time

The problem

- Controlling animation, movement etc
- Moving an object every frame can be unpredictable
  - Not all hardware is created equally
  - Some computers will generate more frames in the same amount of time
  - Moving 1 pixel every frame would be find if everything was 60FPS
  - But if it runs at 30 FPS our game is now at half speed
    - Or twice speed at 120 FPS

Delta time

- The time interval between frames
- Not constant, has random variation
- Its use can ensure that movement/animation/etc are consistent
  - Not dependent on system performance
- To do movement
  - Speed = change in distance over time (per second)
  - Delta = Time between frames
  - Movement = speed * delta
- Careful of the unit of measure second vs millisecond

Some maths

- A rough example
- Desired speed is 100 pixel per second
- We manage 50 frames per second
  - If delta was constant it would be 1/50
  - 0.02 seconds
- So for every frame we want to move
  - 100 pixels * 0.02 seconds = 2 pixels per frame (ish)

Some maths

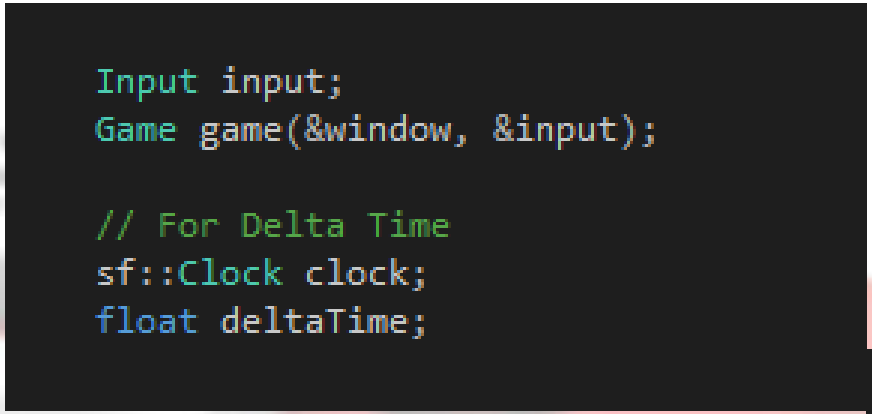
- Now the frame rate changes to 5 FPS
  - As each frame takes longer to calc, delta time increases
  - The distance an object moves per frame changes
  - At 5 FPS the object moves 20 pixels to keep up
- Frame rate independence

Helpful

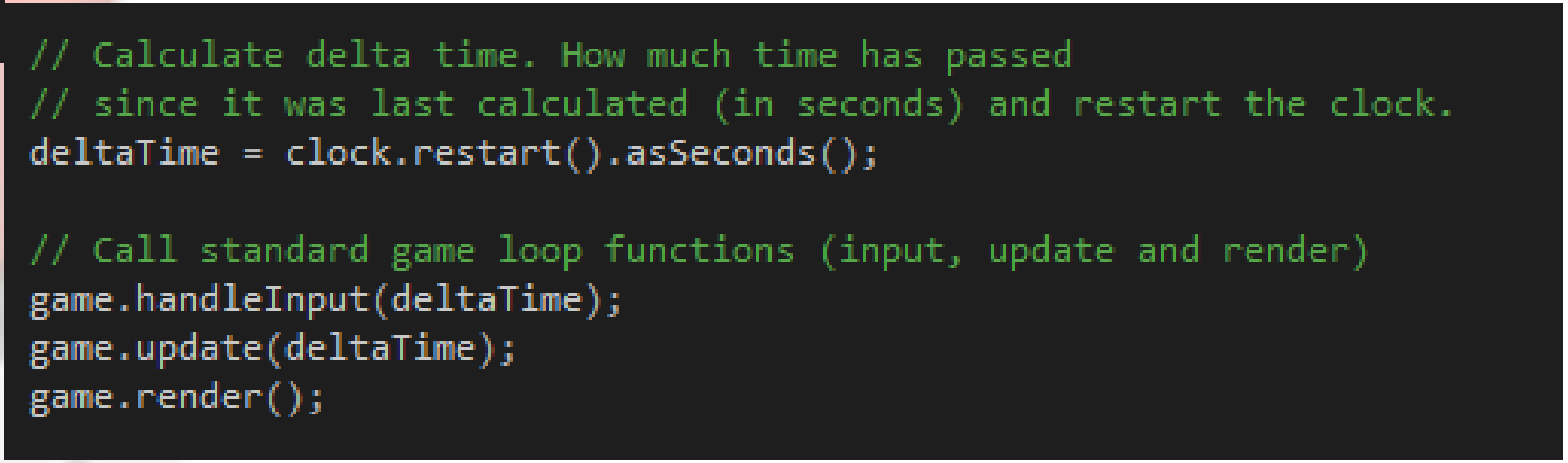
- Calculate movement in floats (not ints)
  - While position will be handled by int (pixel based) delta can be very small
  - If
    - Speed = 1 (int)
    - Delta = 0.02 (float)
    - Movement (int) = speed * delta = 0
  - Resulting in no movement every frame
    - Speed = 1.0f (float)
    - Delta = 0.02 (float)
    - Movement (float) = speed * delta = 0.02
  - No movement until the value is above 1.0 but then movement

Setup delta time

- Inside the game loop (in main.cpp) we need to calculate how much time has passed, since the last time round the loop
- Then pass delta time to the game when handling update and input



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

// For Delta Time
sf::Clock clock;
float deltaTime;
```



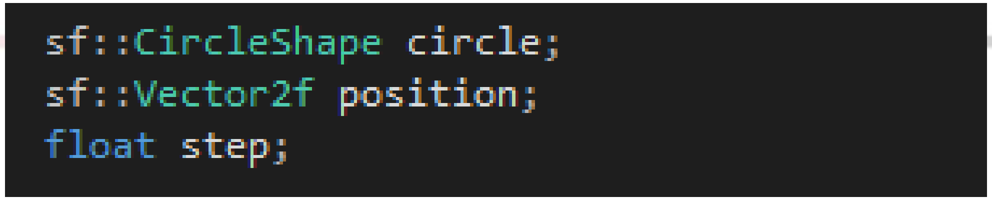
```
// Calculate delta time. How much time has passed
// since it was last calculated (in seconds) and restart the clock.
deltaTime = clock.restart().asSeconds();

// Call standard game loop functions (input, update and render)
game.handleInput(deltaTime);
game.update(deltaTime);
game.render();
```

Example

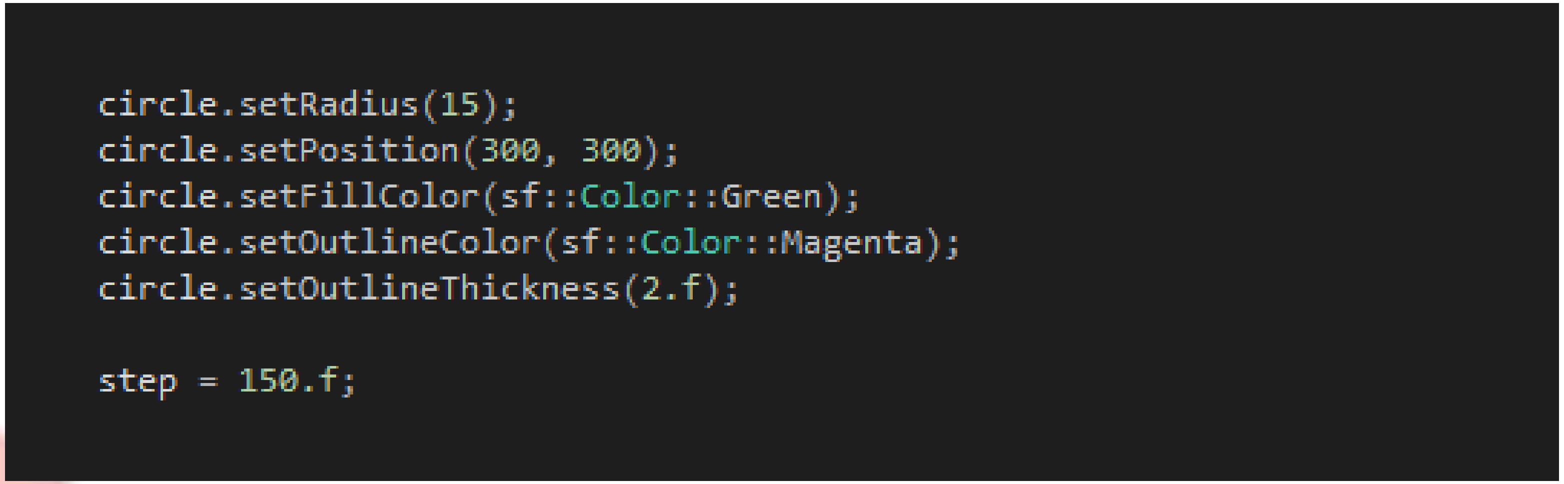
- Program a circle to move across the screen, automatically, using delta time

Moving box example

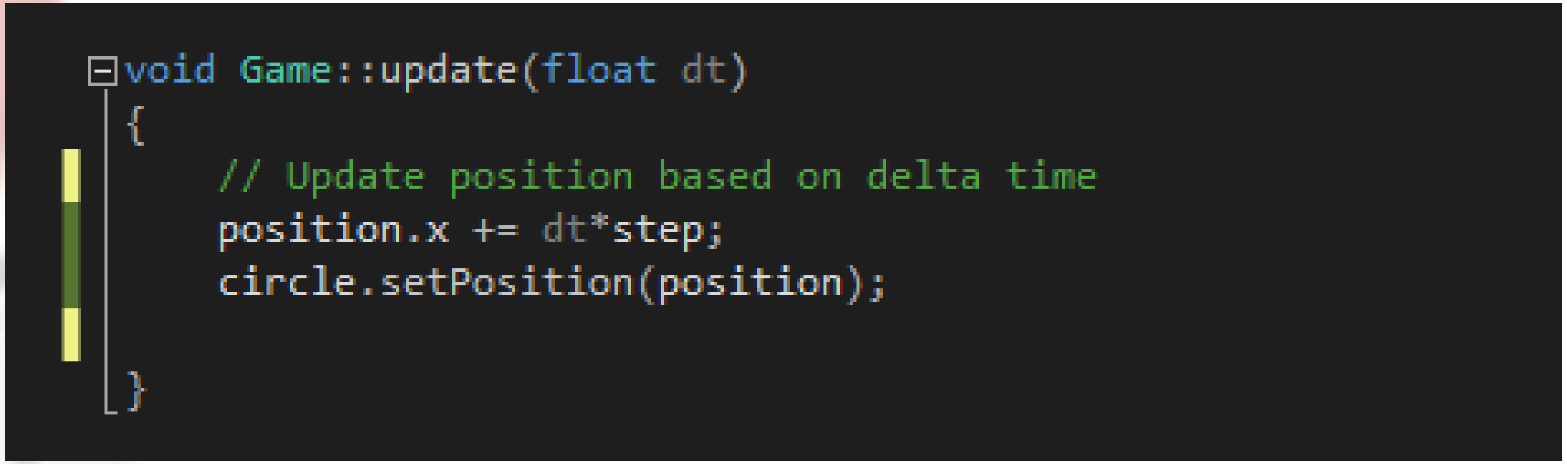
- Required variable
  - Circle
  - Position
  - step
- Circle setup
- Step
  - Speed object will move at
  - Pixels per second



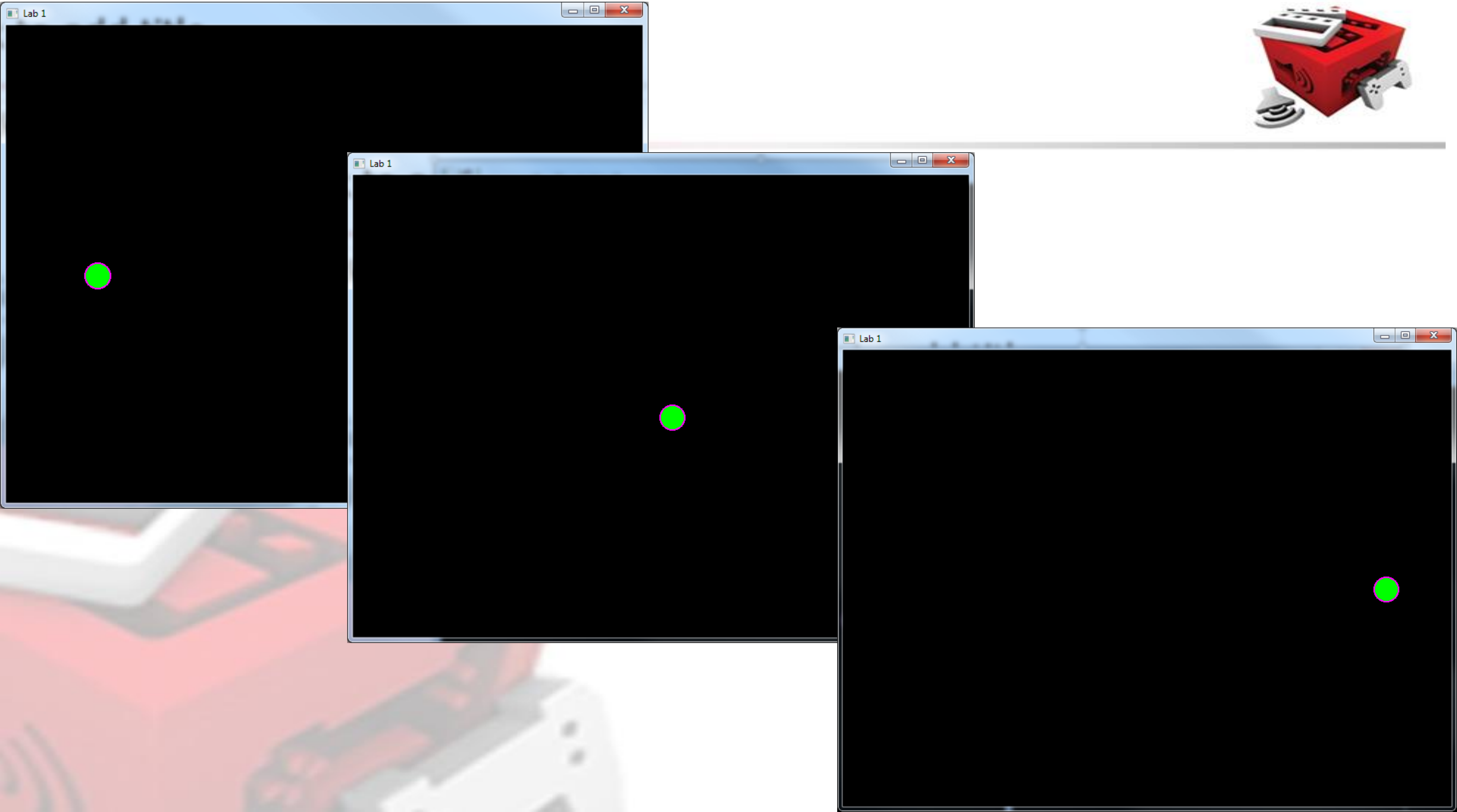
```
sf::CircleShape circle;
sf::Vector2f position;
float step;
```



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

step = 150.f;
```



```
void Game::update(float dt)
{
    // Update position based on delta time
    position.x += dt*step;
    circle.setPosition(position);

}
```


Lab 1
Lab 1
Lab 1

Live demo

- Circe movement with and without delta time

In the labs

- Moving objects with delta time
- User controlled objects with delta time

- Reminder, labs
  - Pods A-G (left side of the room)
    - Another class in the other side of the room