SFML

Simple and Fast Multimedia Library

www.sfml-dev.org

Kim Restad

mail@kimrestad.se

www.bth.se

BLEKINGE INSTITUTE OF TECHNOLOGY

Innehåll

Vad är SFML?
Komma igång med SFML
Ett första program
Spelloopen
Former
Sprites & Texturer
Tid
Animering

Input
- Event
- Polling

Text & Fonter
Utritningsbar klass
Game-klass
Kollisioner
Vad ska ligga var?

Vad är SFML?

- "*SDL fast objektorienterat och skrivet i C++*"
- API för fönsterhantering grafik, ljud, input och nätverk
- Multiplattform
- Använder OpenGL
  - API mot grafikkortet

SFMLs fem moduler

Modulnamn	Användningsområde
System	Behövs alltid – används av alla andra moduler. Här finns bland annat tid och filhantering.
Window	Fönsterhantering och input. Hanterar skapandet och utritningen av ett fönster via OpenGL samt hanterar eventen för fönstret.
Graphics	2D grafik, sprites, texturer, former mm. *Fördjupning: Stödjer även vertisarrayer, flera vyer och att implementera egna OpenGL shaders*.
Audio	Ljudhantering: inladdning och uppspelning av ljud samt manipulation av rådatan för ett ljud.
Networking	Nätverk: TCP och UDP sockets, webförfrågningar (HTML), filöverföring (FTP)

Komma igång med SFML

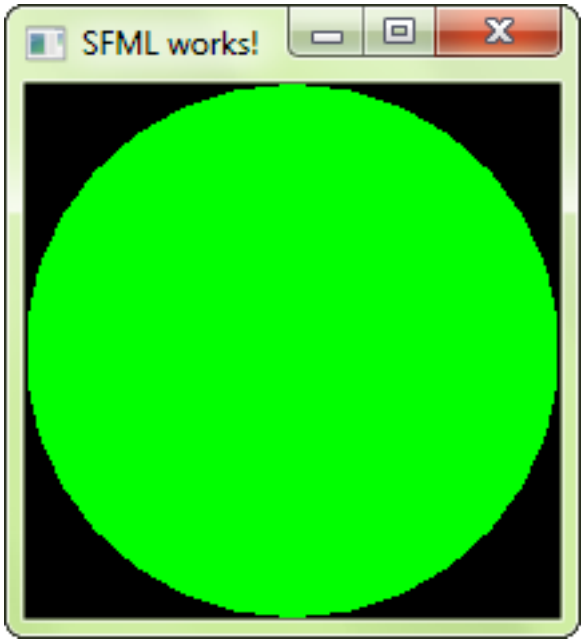
- Länka bibliotek
  - Lägg header-filer och .lib-filer i er projektmapp
  - Ställ in Visual Studio så att det hittar filerna
- Tutorials
  - Visual Studio
  - Linux
- Namespace sf

Det första programmet

- Visual Studio Tutorial…

```
#include <SFML/Graphics.hpp>

int main()
{
    sf::RenderWindow window(sf::VideoMode(200, 200), "SFML works!");
    sf::CircleShape shape(100.f);
    shape.setFillColor(sf::Color::Green);

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        window.clear();
        window.draw(shape);
        window.display();
    }

    return 0;
}
```

http://sfml-dev.org/tutorials/2.1/start-vc.php



http://sfml-dev.org/tutorials/2.1/start-vc.php

Det första programmet, forts.

- Skapa ett fönster att rendera i
  - sf::RenderWindow, tar två argument:
    - VideoMode, tar två argument:
      - Fönstrets bredd
      - Fönstrets höjd
    - sf::string: fönstrets titel

```
sf::RenderWindow window(sf::VideoMode(200, 200), "SFML works!");
sf::CircleShape shape(100.f);
shape.setFillColor(sf::Color::Green);
```

http://sfml-dev.org/tutorials/2.1/start-vc.php

- Skapa en form att rita ut
  - CircleShape tar ett argument: radien
  - Sätt cirkelns fyllningsfärg

Spelloopen

- While-loop
  - Varje varv i loopen motsvarar en *frame*
- Två ansvar:
  1) Fönsterhändelser (events)
  2) Applikationens körning, brukar delas upp i två delar:
     a) Uppdatera applikationen
     b) Rita ut applikationen

```
while (window.isOpen())
{
    sf::Event event;
    while (window.pollEvent(event))
    {
        if (event.type == sf::Event::Closed)
            window.close();
    }

    Update();
    Draw();
}
```

Update() och Draw()

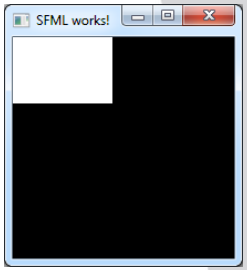
- Update()
  - Spellogik
  - Inputhantering
  - Etc…
- Draw()
  - Utritning
    1. Rensa fönstret
    2. Rita ut alla objekt
    3. Visa resultatet

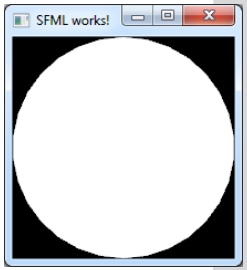
Update() och Draw(), forts.

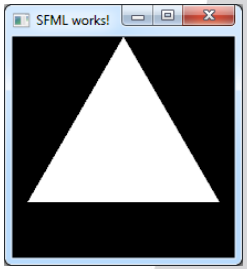
- Hur gör vi med våra variabler window och shape som ligger i main-funktionen?
  - Lägger dem globalt -> Dåligt!
  - Skickar med dem till funktionen -> Bättre
  - Bryter ut all spellogik och lägger det i en egen klass -> BÄST!
    - Kommer längre fram i föreläsningen

Former

- SFML har två fördefinierade former
  - Rektanglar
    - sf::RectangleShape rect(
      sf::Vector2f(90.0, 60.0))
      skapar en rektangel med storlek 90x60
  - Cirklar
    - sf::CircleShape circ(100) skapar en cirkel med radie 100
    - sf::CircleShape circ(100, 3) skapar en cirkel med radie 100 och 3 sidor (dvs. en triangel)







Sprites

- Texturerad rektangel
  - Textur = bild

http://sfml-dev.org/tutorials/2.1/graphics-sprite.php

- Giltiga filformat
  - bmp, gif, hdr, jpg (ej progressive), pic, png, psd och tga

Ladda in en textur

- Skapa en instans av sf::Texture
- loadFromFile*(”filepath”)*
  - *filepath* är sökvägen till filen **relativt** arbetsmappen
    - För Visual Studio är det mappen där filen med filtillägget .vcxproj ligger
  - Returnerar *true* om filen lästes in, annars *false*

```
sf::Texture ballTexture;
if (!ballTexture.loadFromFile("../Assets/ball.png"))
{
    // Error!
}
```

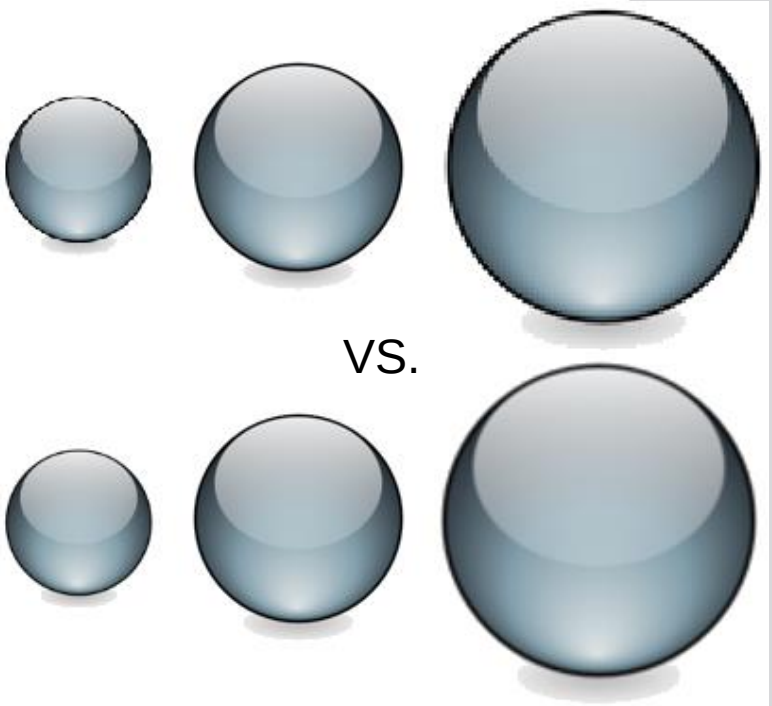
Skapa sprites

- Sprites använder klassen sf::Sprite
  - sf::Sprite ballSprite;
- setTexture() specificerar texturen
  - ballSprite.setTexture(ballTexture);
  - Variabeln *ballTexture* måste finnas så länge spriten ska ritas ut (dvs. ingen lokal variabel!)
- Rita ut
  - window.draw(ballSprite);

```
sf::Sprite ballSprite;
ballSprite.setTexture(ballTexture);

window.draw(ballSprite);
```

Fler funktioner

http://sfml-dev.org/tutorials/2.1/graphics-sprite.php

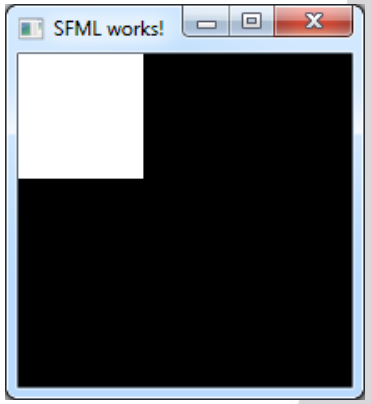


- Textur
  - setSmooth(bool) anger om SFML ska släta ut texturen (true) eller inte (false)
- Sprite
  - setColor anger en färg att tona spriten i

http://sfml-dev.org/tutorials/2.1/graphics-sprite.php

Sprites, vanliga problem

- loadFromFile() misslyckas
  - Har du rätt sökväg till filen?
  - SFML skriver ut ett felmeddelande till konsolen, kolla det
- Vit rektangel istället för texturen
  - Du har sparat texturen i en lokal variabel som försvinner när funktionen är klar. Då försvinner även texturen.



Vit rektangel

Förflytta sprites

- move(float, float)
  - Två argument: rörelse i x- respektive y-led
- move(sf::Vector2f)
  - Ett argument: en float-vektor som anger rörelsen

```
void Update()
{
    ballSprite.move(0.01f, 0.02f);
}
```

```
void Update()
{
    sf::Vector2f velocity(0.01f, 0.02f);
    ballSprite.move(velocity);
}
```

Tid

- Hur lång tid en frame tar beror på:
  - Hårdvaran i datorn
  - Hur mycket kod som exekveras
- Det ger ett inkonsekvent beteende!
- Lösning: multiplicera med framens tid

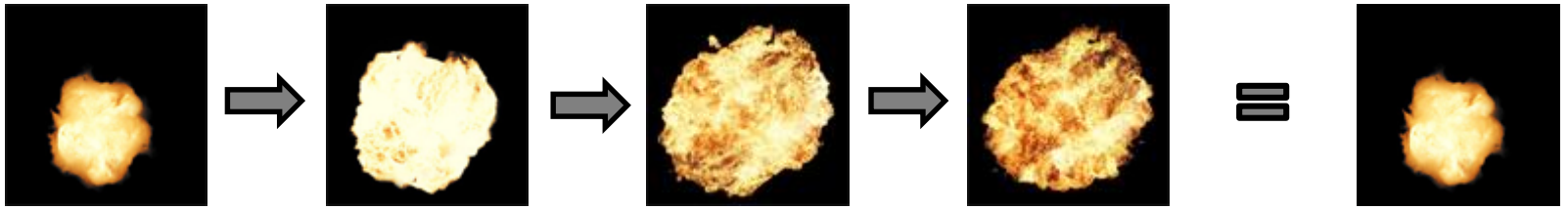
```
sf::Clock gameTime;
```

```
Update(gameTime.restart().asSeconds());
Draw();
```

```
void Update(float dt)
{
    sf::Vector2f velocity(20.0f, 45.0f);
    ballSprite.move(velocity * dt);
}
```

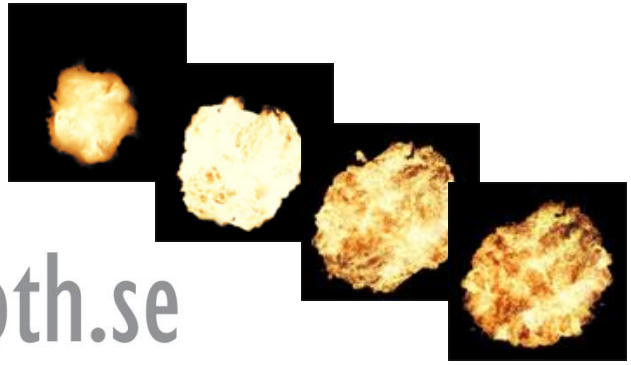
2D-animering

- Ett antal bilder som ritas ut i följd
  - *Key Frame*: En bild i animationen
  - *Animationshastighet*: hur ofta bilden byts

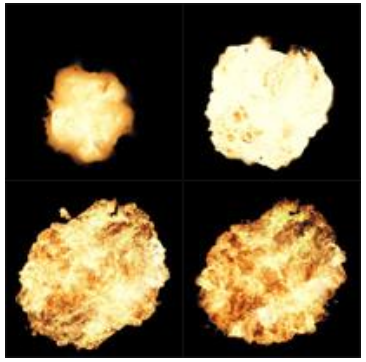


2D-animering, forts.

- Byta bild kan hanteras på två sätt:
  - Ladda in en ny textur
    - Blir snabbt väldigt många texturer
  - Sprite sheet
    - Kan bli stora bilder som tar mycket minne
    - Kan ta onödigt mycket plats om det inte är fullt
    - Lättare att hantera

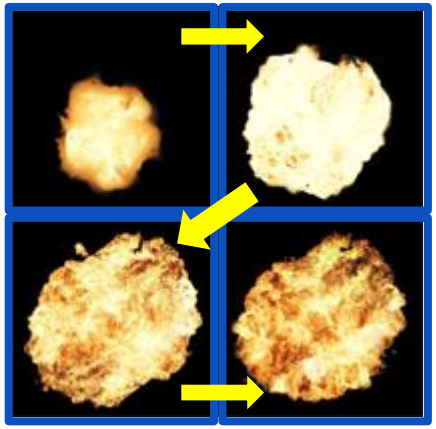


VS



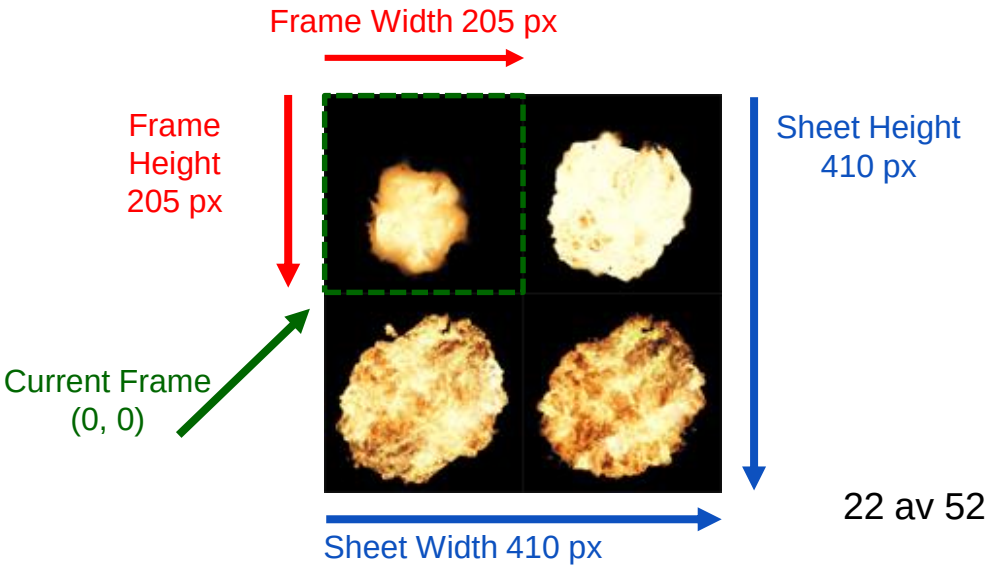
Sprite Sheet

- Alla bilder i animationen ligger i en och samma bildfil.
- Rita ut endast en del av en textur
  - sf::Sprite.setTextureRect(sf::IntRect)
  - För att visa nästa key frame uppdaterar vi bara vilken del som ritas ut!



Sprite Sheet, forts.

- Vi måste hålla reda på flera saker:
  - Storleken på en key frame (bredd, höjd)
  - Antalet key frames på vårt sprite sheet (x, y)
  - Vilken key frame som ska ritas ut (x, y)
  - Animationshastigheten (antal sekunder)



Sprite Sheet, kod

```
sf::Texture spriteSheet;
sf::Sprite explosionSprite;
sf::Vector2i spriteSheetSize(5, 5);
sf::Vector2i spriteFrameSize(205, 205);
sf::Vector2i animationFrame(0, 0);
const float animationSpeed = 0.1f;
float frameDuration = 0.0f;
```

1. Skapa och initialisera variablerna

```
spriteSheet.loadFromFile("../Assets/explosion.jpg");
    explosionSprite.setTexture(spriteSheet);
    explosionSprite.setTextureRect(
        sf::IntRect(0, 0, spriteFrameSize.x, spriteFrameSize.y));
```

2. Ladda in spritesheet och ställ in spriten

```
void Update(float dt)
{
    sf::Vector2f velocity(20.0f, 45.0f);
    ballSprite.move(velocity * dt);

    frameDuration += dt;

    if (frameDuration > animationSpeed)
    {
        animationFrame.x += 1;
        if (animationFrame.x >= spriteSheetSize.x)
        {
            animationFrame.x = 0;
            animationFrame.y += 1;
            if (animationFrame.y >= spriteSheetSize.y)
                animationFrame.y = 0;
        }

        frameDuration = 0;
        explosionSprite.setTextureRect(sf::IntRect(animationFrame.x * spriteFrameSize.x,
                                                   animationFrame.y * spriteFrameSize.y,
                                                   spriteFrameSize.x,
                                                   spriteFrameSize.y));
    }
}
```

3. Uppdatera animationen

Input

- Två sätt att hantera det:
  - Event
    - Använder fönstrets egna event (från operativsystemet)
    - Meddelande när något händer
  - Polling
    - Vanligast
    - Fråga om en specifik tangent är nedtryckt

Event

- Operativsystemet skickar events
- Programmet bör ta hand om dessa
- De hanteras i spelloopen, innan Update() och Draw()
  - Antalet är okänt, så med while loopar vi så länge det finns event vi inte tittat på.
  - Om vi vill kan vi reagera på ett event

```
while (window.isOpen())
{
    sf::Event event;
    while (window.pollEvent(event))
    {
        if (event.type == sf::Event::Closed)
            window.close();
    }

    Update();
    Draw();
}
```

Event forts.

- Olika typer av event
  - Fönstret
    - Ex. Closed, Resized, LostFocus, GainedFocus, etc…
  - Tangentbordet
    - Ex. KeyPressed, KeyReleased, etc…
  - Musen
    - Ex. MouseWheelMoved, MouseButtonPressed, MouseButtonReleased, MouseMoved, etc…

InputEvent

- Vi kan reagera på tangentbordsevent
  - event.type == sf::Event::KeyPressed
    - Händer precis då vi trycker ner tangenten
    - I detta fall kan vi använda event.key.code
      - sf::Keyboard::A, …, sf::Keyboard::Z osv.

```
if (event.type == sf::Event::KeyPressed)
{
    if(event.key.code == sf::Keyboard::A)
    {
            std::cout << "A key pressed" << std::endl;
    }
}
```

Input Polling

- Events -> Om vi alltid vill reagera på input innan något annat händer
  - Ex. Skapa en egen eventhanterare
- Polling -> Om vi vill ha kontroll på när vi reagerar på input
  - Ex. Först ska alla fiender flyttas, därefter kollar vi om användaren har tryckt på någon tangent och flyttar spelaren i så fall
  - Vi vill ha kontroll!

Input Polling: keyboard

- sf::Keyboard
  - Ta reda på om en tangent är nedtryckt
  - isKeyPressed(Key)
    - Key: sf::Keyboard::A, …, sf::Keyboard::Z osv.

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::A))
        // Move player character left
```

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::P))
        std::cout << "Tangent P är nedtryckt" << std::endl;
```

Input Polling: keyboard, forts.

- Trycktes tangenten ner denna frame?
  - Spara om tangenten var nedtryckt föregående frame
  - Om den inte var det då och är det nu så har den precis tryckts ner

```
bool wasAPressed = false;
```

```
bool isAPressed = sf::Keyboard::isKeyPressed(sf::Keyboard::A);

if (isAPressed && !wasAPressed)
    std::cout << "Tangent A är nedtryckt" << std::endl;

wasAPressed = isAPressed; // SIST!
```

Input Polling: mouse

- sf::Mouse
  - isButtonPressed(Button)
    - Button: sf::Mouse::Left, sf::Mouse::Right, osv.
  - getPosition()
  - setPosition(sf::Vector2i)

```
if (sf::Mouse::isButtonPressed(sf::Mouse::Button::Left))
        sf::Mouse::setPosition(sf::Mouse::getPosition() + sf::Vector2i(1, 1));
```

Om vänster musknapp är nedtryckt, flytta musens position 1 pixel åt höger och 1 pixel nedåt.

```
if (sf::Mouse::isButtonPressed(sf::Mouse::Button::Right))
        sf::Mouse::setPosition(sf::Vector2i(100, 100), window);
```

Om höger musknapp är nedtryckt, flytta musens till position (100, 100) räknat från fönstrets övre vänstra hörn.

Text

- I SFML består text av ett font-objekt och ett text-objekt
  - sf::Font och sf::Text
  - Jämför med sprites som består av en textur och en sprite (sf::Texture och sf::Sprite)
- Giltiga filformat:
  - BDF, CFF, OpenType, PFR, SFNT, TrueType, Type 1, Type 42, Windows FNT och X11 PCF

Font

- Skapa en instans av sf::Font
- loadFromFile*(”filepath”)*
  - *filepath* är sökvägen till filen **relativt** arbetsmappen
    - För Visual Studio är det mappen där filen med filtillägget .vcxproj ligger
  - Returnerar *true* om filen lästes in, annars *false*

```
sf::Font gameFont;
if (!gameFont.loadFromFile("../Assets/BuxtonSketch.ttf"))
{
    // Error!
}
```

Font, forts.

- Teckensnitt måste laddas in via en sökväg till filen
  - Kan inte laddas in direkt från systemet
    - ~~Ex. loadFromFile(”Arial”); fungerar INTE~~
    - Istället måste de kopieras till projektmappen
      - I Windows ligger teckensnitten i C:\Windows\Fonts
      - I Unix-system kan sökvägarna vara ex. /usr/shared/fonts /usr/local/share/fonts eller /home/<username>/.fonts

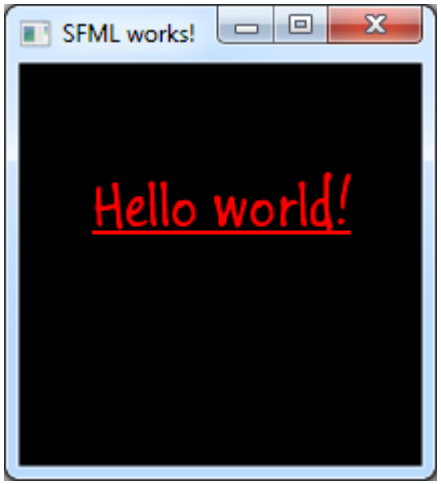
Textutritning

- Skapa en instans av sf::Text
  - Textens utseende definieras mha funktioner
    - setFont(sf::Font) – sätter teckensnittet
    - setString(sf::String) – sätter texten
    - setCharacterSize(int) – sätter textens storlek i pixlar (dvs inte den vanliga teckensnittsstorleken)
    - setColor(sf::Color) – sätter färgen
    - setStyle(uint) – sätter stilen, ex fetstil, kursiv. Flera kan anges genom att avgränsa dem med |
      - Ex. text.setStyle(sf::Text::Italic | sf::Text::Bold);

Textutritning, forts.

- Positionen anges med funktionen text.setPosition(sf::Vector2f);
- Texten ritas ut genom att skicka in instansen till fönstrets draw()-funktion

```
sf::Text text;
text.setFont(gameFont);
text.setString("Hello world!");
text.setCharacterSize(30);
text.setColor(sf::Color::Red);
text.setStyle(sf::Text::Underlined);
text.setPosition(sf::Vector2f(36, 50));
window.draw(text);
```



Utritningsbart objekt

- Alla objekt i SFML som kan ritas ut ärver från klassen sf::Drawable
- sf::Drawable har endast en funktion, draw(sf::RenderTarget, sf::RenderStates)
  - pure virtual = abstrakt funktion
    - Dvs funktionen måste implementeras av den ärvande klassen
  - OBS! Första bokstaven, ’d’, måste vara en gemen (annars kraschar exekveringen)

Utritningsbart objekt, forts.

- I draw()-funktionen måste vi rita ut allt som klassen ska rita ut
  - Istället för window.draw() använder vi den *target*-parameter som skickas in
    - target.draw(textHelloWorld);
  - *states*-parametern skickar vi med så att vi inte blir av med information
    - Innehåller ex. BlendModes, Transformationer etc
    - target.draw(textHelloWorld, states);

Utritningsbart objekt, kod

```
#ifndef MY_DRAWABLE_HPP
#define MY_DRAWABLE_HPP

#include <SFML/Graphics.hpp>

class MyDrawable : public sf::Drawable
{
public:
    MyDrawable();

private:
    sf::Font gameFont;
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
};
#endif
```

1. Ärv från sf::Drawable

2. Implementera draw()

```
#include "MyDrawable.hpp"

MyDrawable::MyDrawable()
{
    if (gameFont.loadFromFile("../Assets/BuxtonSketch.ttf"))
    {
        // Error!
    }
}

void MyDrawable::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
    sf::Text text;
    text.setFont(gameFont);
    text.setString("Hello world!");
    text.setCharacterSize(30);

    target.draw(text, states);
}
```

3. Använd target-*parametern för att rita ut*

Transformationer

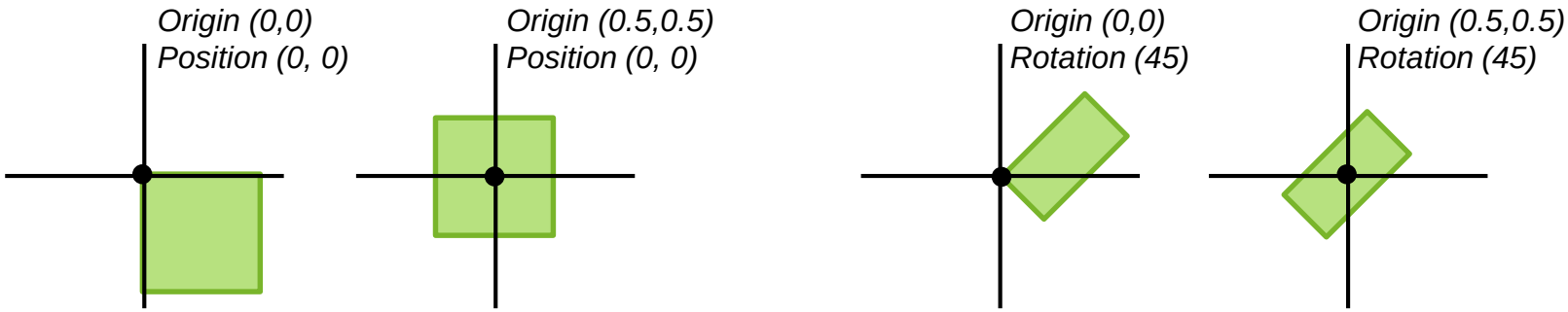
- Många objekt i SFML kan transformeras på olika sätt
  - Ex. setPosition, setRotation, move
- Alla dess objekt ärver från klassen sf::Transformable
- Genom att även låta egna klasser ärva från sf::Transformable får vi dessa funktioner gratis!

sf::Transformable

- Position
  - move, setPosition, getPosition
- Rotation
  - rotate, setRotation, getRotation
- Skala
  - scale, setScale, getScale
- Dokumentation

Origin

- Den punkt varifrån alla transformationer utgår
  - Default är (0, 0) dvs övre vänstra hörnet
  - setOrigin, getOrigin
- Ofta i mitten av objekt



Använda egna klasser

- Använda klassen
  - Skapa en instans av klassen i main.cpp
  - Anropa klassens Update() från Update() i main.cpp
  - Rita ut klassen i Draw()

```
MyDrawable drawable;
```

```
void Update(float dt)
{
    drawable.Update();
}
```

- Om ett objekt påverkas av input, lägg inputhanteringen i dess klass
  - Update()-funktionen

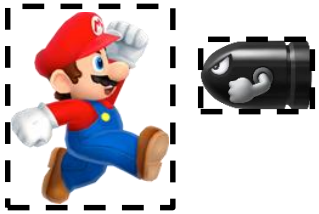
```
void Draw()
{
    window.clear();
    window.draw(drawable);
    window.display();
}
```

Kollisionshantering

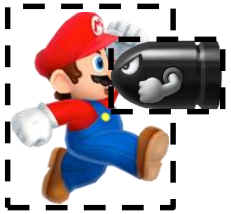
- Kolla om ett objekt, *O*, kolliderar med något
  1. Loopa igenom alla objekt, K_1, … , K_n som *O* kanske kan kollidera mot
  2. För varje objekt K_i, kolla om det kolliderar med *O*.
  3. Om en kollision har inträffat, reagera!
- Om vi vill kolla många objekt, O_1, …, O_n kan vi lägga det i en yttre loop

Kollisionshantering, forts.

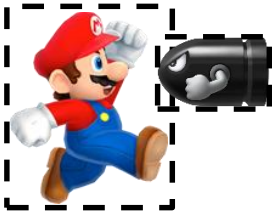
- Kolla om en kollision inträffat
  - Lättast är att använda *bounding boxes*
    - En rektangel som inte (nödvändigtvis) ritas ut men som omger hela objektet som kan kollidera
    - Med bounding boxes behöver vi bara se om dessa två rektanglar överlappar -> kollision!
    - Kan ge ”falska kollisioner”



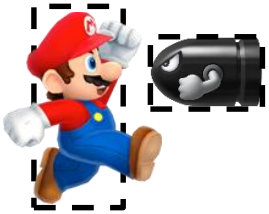
Ingen kollision



Kollision!



Falsk kollision



Lösning: mindre bounding box

Game-klass

- Det blir snabbt mycket kod i main
  - Svårt att överse
  - Många globala variabler = dåligt!
- Bryt ut all kod som har med spelet att göra till en egen utritningsbar klass
- Kvar i main
  - Fönsterhantering
  - Eventhantering

Game-klass, kod

Game.hpp

```
#ifndef GAME_HPP
#define GAME_HPP

#include <SFML/Graphics.hpp>

class Game : public sf::Drawable
{
public:
    Game();
    ~Game();

    void Update(float dt);

private:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;

};
#endif
```

```
#include "Game.hpp"

Game::Game()
{
    // Initialise all game variables here
}

Game::~Game()
{
    // Clean up game
}

void Game::Update(float dt)
{
    // All logic goes here (as well as Input handling)
}

void Game::draw(sf::RenderTarget& target, sf::RenderStates states) const
{
    // Draw everything in the game
}
```

Game.cpp

Game-klass, kod: main

```
#include <SFML/Graphics.hpp>
#include "Game.hpp"

int main()
{
    Game game;
    sf::RenderWindow window(sf::VideoMode(200, 200), "GameWindow");
    sf::Clock gameTime;

    while (window.isOpen())
    {
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
            // Handle other events if necessary
        }

        game.Update(gameTime.restart().asSeconds());
        window.clear();
        window.draw(game);
        window.display();
    }

    return 0;
}
```

main.cpp

Tips: Var ska koden ligga?

- main.cpp
  - Så lite som möjligt, ex spelloopen och fönsterhantering (+ Game-instans)
- Objektet
  - All logik som ändrar objektets status
    - T.ex. Input, förflyttning, kollision
- Input
  a) I Game-klassen
  b) I ett objekt om endast ett objekt påverkas

Tips: Var ska koden ligga?

- Kollisionshantering
  - Kort svar: Det beror på
  - Tumregel: *Lägg kollisionskoden där större delen av informationen som behövs för kollisionen finns*
    - Kollision mellan spelare och skott kan ligga i en skotthanterare (många fler skott än spelare)
    - Kollision mellan fiender kan ligga i fiende-klassen (eller en fiende-hanterare)

Questions?

Länkar

- SFML home
- SFML tutorials
- SFML keyboard documentation
- SFML mouse documentation

- Ladda ner SFML
- Ytterligare versioner (ex. VS2013)