



# Fördjupning i SFML

Simple and Fast Multimedia Library

[www.sfml-dev.org](http://www.sfml-dev.org)

Carl Søyseth  
[caso13@student.bth.se](mailto:caso13@student.bth.se)



# Dagens agenda

- Snabbt rep. På gameloopen.
- Hur man skapar och använder texturer och sprites
- Animering
- Hur man kan hantera input
- Texter och fonts
- Utritningsbarklass
- Game-klass och generell struktur
- Kollisioner
- Struktur



# Den där omtalade gameloopen

- 3 ansvarsområden
  - Kolla events (ska fönstret stängas? Ska det bli större? Mindre? Är det i fokus?)
  - Uppdatera och applicera spel-logiken. Utgörs av att vi anropar `Update(float dt)` på Game objektet.
  - Rendera / rita och presentera ändringarna. Utgörs av att vi anropar `Draw()` på Game objektet.

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

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



# Skillnaden mellan fps och fps

- Vi vill förflytta oss 1 enheter på en sekund, vi har 60 fps
  - `move(direction * 1)`
  - Koden körs 60 gånger under 1 sekund,  $60 * 1 = 60$  enheter
  - 59 enheter för mycket
- Antingen sänker vi fpsen till 1, eller så löser vi det med lite enkel matte.
- Vi inför dt (delta tid), och låter  $dt = 1 / fps$  (dvs skillnaden i hur lång tid det tar mellan varje frame)
  - `move(direction * 1 * dt)`
  - Koden körs 60 gången under 1 sekund,  $60 * 1 * (1 / 60) = 1$  enhet.
- Genom att dela upp förflyttningen i 60 olika steg, som tillsammans blir 1, får vi önskad förflyttning

# Texturer och sprites

- Sprites är en texturerad rektangel.
  - En textur är en bild.
- Sprites är vad vi kommer huvudsakligen använda för att representera föremål i vårt spel.



+



=




Rectangular entity

Texture

Sprite!

<https://www.sfml-dev.org/tutorials/2.4/images/graphics-sprites-definition.png>

- 
- Söker alltid relativt , dvs det är viktigt att du placerat din bild på rätt plats för inläsning
  - `loadFromFile` returnerar Boolean.
  - Vid fel kolla output fönstret i visual studio
  - Kan läsa in bmp, gif, hdr, jpg, pic, png, psd och tga.

Output

Show output from: Debug

'directx11.exe' (Win32): Unloaded 'C:\V  
The thread 0x1f44 has exited with code  
The thread 0x2468 has exited with code  
The thread 0x420 has exited with code  
The thread 0x1c8c has exited with code  
The program '[496] directx11.exe' has

```
1 sf::Texture backgroundTexture;  
2 if (!texture.loadFromFile("../resources/background.png"))  
3 {  
4     // hantera ev. fel här.  
5 }
```

Error List Output Find Symbol Results

../resources/player.png  
Vad innebär detta?

This PC > Documents > Visual Studio 2017 > projects > SFML Project > SFML Project

Name	Date modified	Type
++ Game.cpp	2016-12-01 11:42	C++ Source
Game.hpp	2016-02-19 18:25	C/C++ Header
++ main.cpp	2016-12-02 18:36	C++ Source
++ Player.cpp	2018-01-28 11:42	C++ Source
Player.hpp	2016-12-01 13:58	C/C++ Header
SFML Project.vcxproj	2018-01-28 11:28	VC++ Project
SFML Project.vcxproj.filters	2016-02-19 19:32	

Name	Date modified	Type	Size
.vs	2016-08-28 14:14	File folder	
Bin	2016-12-02 18:22	File folder	
Externals	2016-12-02 18:34	File folder	
Obj	2016-12-02 18:22	File folder	
Resources	2016-08-28 14:14	File folder	
SFML Project	2018-01-28 11:42	File folder	
SFML Project.sdf	2016-02-19 19:42	SDF File	13 376 KB
SFML Project.sln	2016-02-19 19:27	Visual Studio Solu...	2 KB
SFML Project.sdb	2016-12-02 18:40	Data Base File	10 668 KB

This PC > Documents > Visual Studio 2017 > projects > SFML Project > Resources





# Läsa in texturer, vad kan gå fel?

- Kan ibland ha oväntat betende, trots att allt ser rätt ut i koden, funkar det inte.
  - “Unable to open file”
    - Då är sökvägen troligtvis inte rätt, se till att den är korrekt, relativt .vcxproj filen
    - Alt, filen är korrupt / fel format.
  - Texturen är vit / syns bara väldigt kort.
    - Den har gått out of scope, dvs att det inte finns i minnet längre
      - Låt ett objekt som har högre scope hålla i den.





## Gör inte såhär

```
2 void selectTexture(sf::sprite backgroundSprite)
3 {
4     sf::Texture backgroundTexture;
5
6     if (timeOfDay == time::day)
7     {
8         backgroundTexture.loadFromFile("../resources/dayTimeTexture.jpg")
9     }
10    else
11    {
12        backgroundTexture.loadFromFile("../resources/nightTimeTexture.jpg")
13    }
14
15
16    backgroundSprite.setTexture(backgroundTexture);
17 }
18
19 int main()
20 {
21     sf::sprite backgroundSprite;
22     selectTexture(backgroundSprite);
23 }
```



# Objekt räddar dagen

```
26 class Alien
27 {
28     private:
29         sf::Texture alienTexture;
30         sf::Sprite enemy;
31
32     public:
33         Alien(float x, float y);
34
35     private:
36 };
37
38
39 Alien::Alien(float x, float y)
40 {
41     alienTexture.loadFromFile("../resources/alienTexture.png");
42     enemy.setTexture(alienTexture);
43     enemy.setPosition(sf::Vector2f(x,y));
44     // Etc etc
45 }
```



# Kombinera sprite och textur till något vettigt

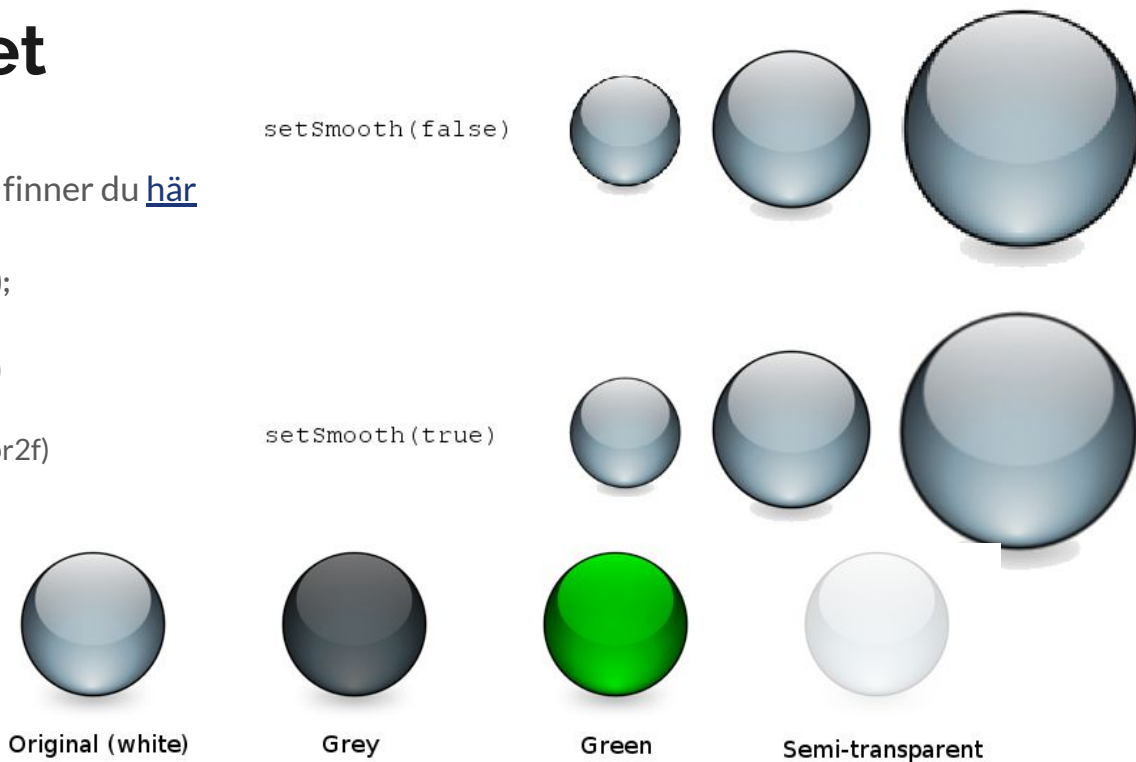
- Texturen laddas in
- Spriten får texturen
  - Återigen måste texturen “leva” lika länge som spriten.
- Spriten ritas ut.

```
1  sf::Texture backgroundImage;  
2  if (!texture.loadFromFile("../resources/background.png"))  
3  {  
4      // hantera ev. fel här.  
5  }  
6  
7  sf::Sprite sprite;  
8  sprite.setTexture(backgroundImage);  
9  window.draw(sprite);  
10
```

<https://www.sfml-dev.org/tutorials/2.4/images/graphics-sprites-smooth.png>

# Lite funktionalitet

- Allt du kan göra med en sprite finner du [här](#)
- Textur
  - `sf::texture.setSmooth(true);`
- Sprites
  - `sf::sprite.setColor(sf::color)`
  - Kan transformeras
    - `setPosition(sf::vector2f)`
    - `move(sf::vector2f)`
    - `rotate(float)`
    - `scale(float)`
    - Och mycket mer





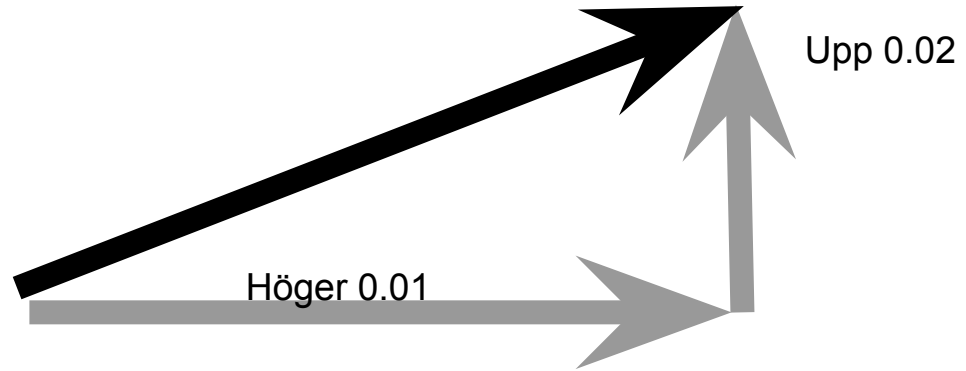
## Lite eksempel hur det kan se ut

```
void Alien::update(float dt)
{
    alien.move(currentDirection * dt);
    alien.setScale(alien.scale() + sf::vector2f(dt, dt));
    alien.rotate(dt);

    if (alien.getPosition().x > 800)
        currentDirection = currentDirection * -1.0;
    else if (alien.getPosition().x < 0)
        currentDirection = currentDirection * -1.0;
}
```

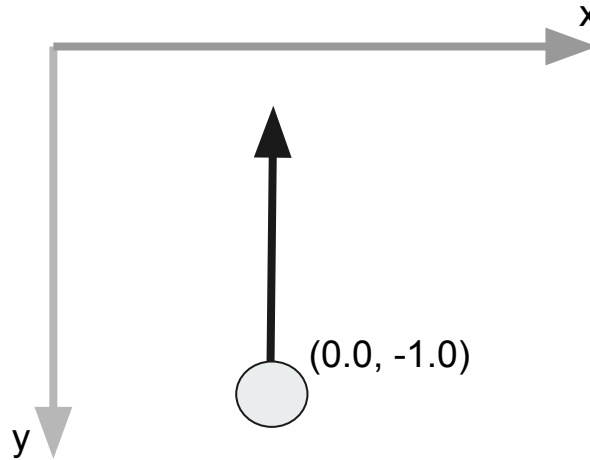
# Funktionalitet - exempel

- All rörelse utgörs av vektorer
  - `sprite1.move(0.01f, 0.02f);`
- Alt
  - `sf::Vector2f velocity(0.01f, 0.02f);`
  - `sprite.move(velocity);`
  - `sprite.move(0.01f, 0.02f);`



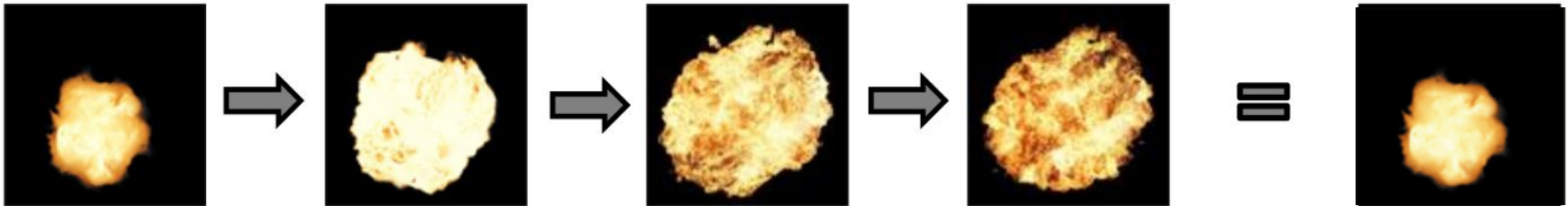
# SFMLs koordinatsystem

- Y är positiv åt “nedåt”
- X är positiv åt “höger”
- Dvs, för att röra oss uppåt
  - `sprite.move(0.0f, -1.0f);`



# Animering, i 2D

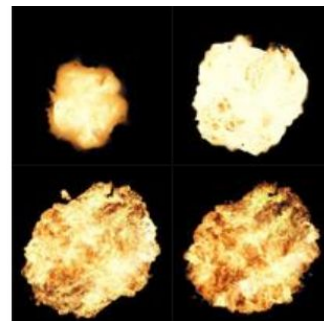
- Våldigt tacksamt jämfört med 3D, vi behöver bara byta ut bild
  - Key frame: En bild av animationen
  - Animationshastighet: Hur snabbt / ofta vi byter bild.





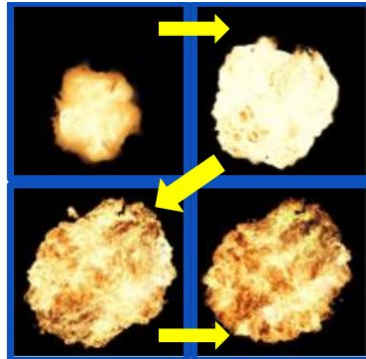
# Animering, i 2D

- Kan göras på två sätt
  - Ha 4 st texturer inladdad för 4 st keyframes
    - Blir onödigt mycket att hantera / omständigt, 4 st objekt
  - Använda sprite-sheet
    - Stora bilder, som kan ta mycket minne, men minne är “gratis”.
    - Kan vara slöseri, om inte “allt” används
    - Lättare att hantera, bara ett objekt att hålla koll på.



# Sprite sheet

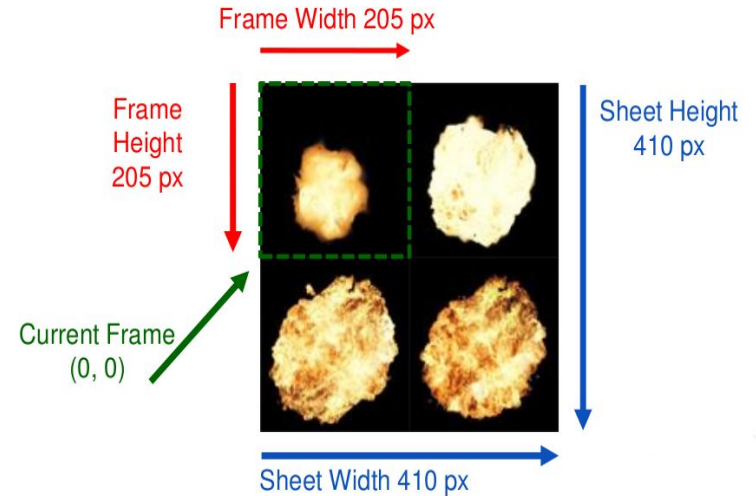
- Alla keyframes ligger i en och samma bild
- Vi ritar bara ut en bit av bilden
  - Använder `sf::Sprite.setTextureRect(sf::IntRect)`
  - För att visa nästa bit av animationen uppdaterar vi bara vilken del av vår sprite som ska ritas ut



# 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 bilden visas)



```
sf::Texture spriteSheet;  
sf::Sprite explosionSprite;  
sf::Vector2i spriteSheetSize(2, 2);  
sf::Vector2i spriteFrameSize(205, 205);  
sf::Vector2i animationFrame(0, 0);  
const float animationSpeed = 0.1f;  
float frameDuration = 0.0f;
```

1. Skapa och  
initialisera variablerna

```
void Update(float 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));  
    }  
}
```

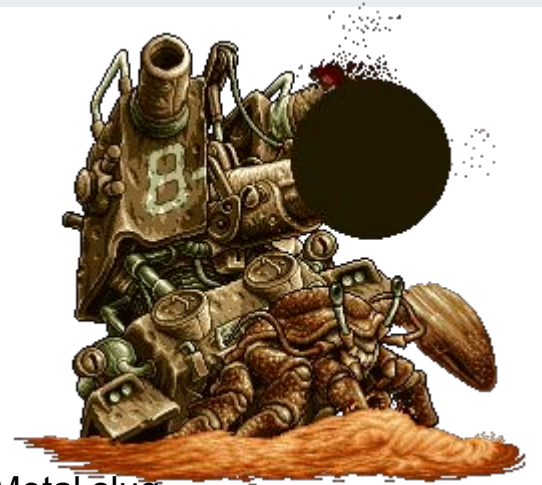
```
spriteSheet.loadFromFile("../Assets/explosion.jpg");  
explosionSprite.setTexture(spriteSheet);  
explosionSprite.setTextureRect(  
    sf::IntRect(0, 0, spriteFrameSize.x, spriteFrameSize.y));
```

2. Ladda in spritesheet  
och ställ in spriten

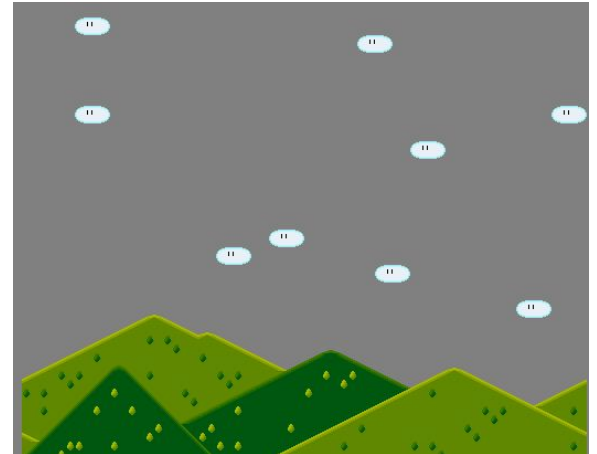
3. Uppdatera  
animationen

# När använder vi vad?

- Spritesheets
  - När vi har ett objekt som rör på sig, och vi vill ge spelaren en visuell Representation
- Sprite, när föremålet är statiskt, bakgrundsbild



Metal slug. troligtvis det bästa spelet, någonsin





# Input

- Events
  - Operativsystemet berättar för oss vad som hänt (vilka knappar som tryckts ned, om muspekaren flyttats), Om fönstret ändrat storlek, är i fokus, om det stängts.
    - `Event.type == sf::Event::KeyPressed`
      - För att sedan kolla om `event.key.code == sf::Keyboard::A`
- Dock är vi inte intresserade av detta.
  - Omständigt, skapar dålig struktur

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

    // Update()
    game.Update(gameTime.restart().asSeconds());

    // Draw()
    window.clear();
    window.draw(game);
    window.display();
}
```



# Input

- Polling
  - Är en viss tangent nedtryckt
  - `isKeyPressed(Key)`
    - Där key kan vara `sf::Keyboard::A.... Keyboard::Z`
  - Görs inte i main - utan i objektet där det är relevant.
    - Blir troligtvis bara i “Player” i vårt fall, för vad annars vill kontrollera med input?



# Exempel på polling

Vi bygger alltså en vektor som beskriver hur vi ska

Röra oss, utefter input

```
sf::Vector2f direction(0.0f, 0.0f);

// Handle input from arrow keys and update direction and animation
if (sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
{
    direction.x = -1.0f;
    mKeyFrameDuration += dt;
    mCurrentKeyFrame.y = 1;
}
else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Right))
{
    direction.x = 1.0f;
    mKeyFrameDuration += dt;
    mCurrentKeyFrame.y = 2;
}
else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Down))
{
    direction.y = 1.0f;
    mKeyFrameDuration += dt;
    mCurrentKeyFrame.y = 0;
}
else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Up))
{
    direction.y = -1.0f;
    mKeyFrameDuration += dt;
    mCurrentKeyFrame.y = 3;
}

mSpriteSheet.move(direction * mSpeed * dt);
```





# Polling vs events

Ordningen spelar roll

- Events kollas först. Om vi alltid vill reagera på något innan något annat har hänt
- Polling kollas när vi själva bestämmer det.
  - Först ska fiender flyttas, därefter kollar vi om användaren tryckt någon tangent, och flyttar spelaren i så fall
  - Reagera på input i spelarklassen.
  - Vanligast och enklare



# Text

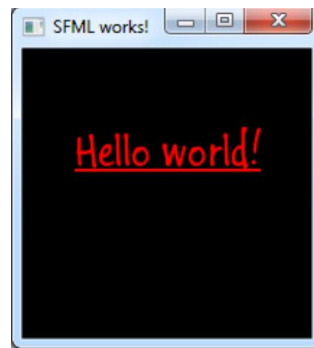
- Utgörs av att man kombinerar sf::Font och sf::Text
  - Font är hur typsnittet ser ut
  - Text är själva innehållet, storlek, färg och stil
- Font laddas in i text, det är alltså text vi arbetar primärt med
  - Text ärver ifrån Transformable
    - Vi kan alltså flytta texten
    - Ge den ny position
    - Skala
    - Etc
  - Sprite och texture är en bra liknelse

```
sf::Font gameFont;  
if (!gameFont.loadFromFile("BuxtonSketch.ttf"))  
{  
    // Något gick fel, hantera felet  
}
```

# Hur används texten?

- Void sf::Text.setString(String) - Sätter texten
- Void sf::Text.setFont(sf::Font)- anger vilken font som ska användas (tex. Den vi läste in tidigare)
- Void sf::Text.setCharacterSize(unsigned int) - Storlek
- Void sf::Text.setColor(sf::Color) - Enum typ som anger färgen.
- Void sf::Text.setStyle(unit) - Anger stil, **fet**, kursiv
  - Text.setStyle(sf::Text::Italic | sf::Text::Bold)

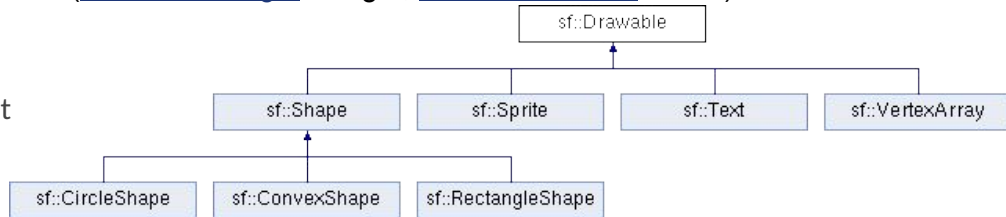
[https://www.sfml-dev.org/documentation/2.4.2/classsf\\_1\\_1Text.php](https://www.sfml-dev.org/documentation/2.4.2/classsf_1_1Text.php)



```
7  sf::Font gameFont;
8  if (!gameFont.loadFromFile("BuxtonSketch.ttf"))
9  {
10     // Något gick fel, hantera felet
11 }
12
13 sf::Text text;
14 text.setFont(gameFont);
15 text.setString("Hello world");
16 text.setCharacterSize(30);
17 text.setColor(sf::Color::Red);
18 text.setStyle(sf::Text::Underlined);
19 text.setPosition(sf::Vector2f(36.0f, 50.0f));
20 window.draw(text);
```

# Våra egna klasser - drawable

- Allt som kan renderas i SFML har någon relation till sf::Drawable
- sf::Drawable är en Abstract class -> void draw([sf::RenderTarget& target](#), [sf::RenderStates states](#)) const
- Används istället för window.draw()
  - Istället skriver vi target.draw(circle);
  - States innehåller information om objektet
    - Transformationer
    - Blendmodes
    - Sprites



[https://www.sfml-dev.org/documentation/2.4.2/classsf\\_1\\_1Drawable.png](https://www.sfml-dev.org/documentation/2.4.2/classsf_1_1Drawable.png)



# Exempelkod

```
class MyDrawable : public sf::Drawable
{
private:
    sf::Font gameFont;
    sf::Text gameText;

public:
    MyDrawable(std::string& fontName const);

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

```
#include "MyDrawable.h"

MyDrawable::MyDrawable(std::string& fontName const)
{
    if (!gameFont.loadFromFile(fontName))
    {
        // Något gick fel, hantera felet
    }

    gameText.setFont(gameFont);
    gameText.setString("Hello world");
    gameText.setCharacterSize(30);
    gameText.setColor(sf::Color::Red);
    gameText.setStyle(sf::Text::Underlined);
    gameText.setPosition(sf::Vector2f(36.0f, 50.0f));
}

void MyDrawable::draw(sf::RenderTarget& target, sf::RenderStates states)
{
    target.draw(gameText states);
}
```

```
89 int main()
90 {
91     RenderWindow window(VideoMode(300, 300, 32), "Hello");
92     MyDrawable entity;
93
94     window.draw(entity);
95 }
```



## Varför ärva ifrån drawable?

- Alternativet är att vi blir tvungna att göra följande

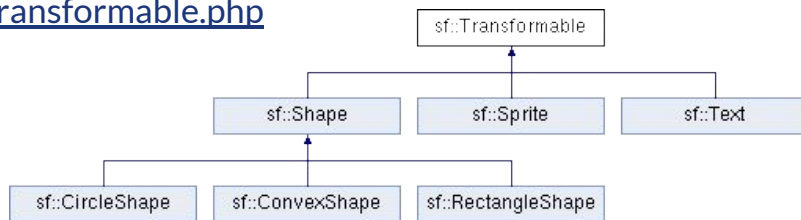
```
79 virtual void draw(sf::RenderWindow& window) const
80 {
81     window.draw(gameText);
82 }
```

```
84 int main()
85 {
86     RenderWindow window(VideoMode(300, 300, 32), "Hello");
87     MyDrawable entity;
88
89     entity(window);
90 }
```

# Vår egna klasser - transformable

- Transformable innehåller allt för att förflytta och manipulera objekt
- Det är transformable som ger Sprite, text etc möjligheten att använda move etc
- Genom att låta våra objekt ärva från Transformable får vi all funktionalitet “gratis”
- Dock behövs inte transformable i alla lägen, om ditt objekt består av en sprite, behövs det inte.
  - Det är så Shape, Sprite och Text funkar

[https://www.sfml-dev.org/documentation/2.4.2/classsf\\_1\\_1Transformable.php](https://www.sfml-dev.org/documentation/2.4.2/classsf_1_1Transformable.php)





## Exempel när det är värt att ärva från transformable

- Ett objekt består av flera st sprites, alla ska roteras, och flyttas relativt varandra.
  - Istället för att anropa rotate och move på alla spritesen individuellt.
    - Anropar vi rotate och move på objektet



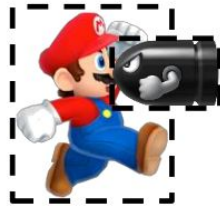
```
78 class MyEntity : public sf::Transformable, public sf::Drawable
79 {
80     virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
81     {
82         states.transform *= getTransform();
83         for (auto& const sprite : sprites)
84         {
85             target.draw(sprite, states);
86         }
87     }
88 };
89 int main()
90 {
91     MyEntity entity;
92     entity.setPosition(10, 20);
93     entity.setRotation(45);
94     window.draw(entity);
95 }
```

# Kollisionshantering

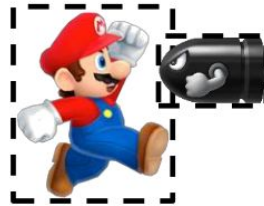
- Viktig del av vårt spel
- Görs med hjälp av rektanglar, sk. *Boundingboxes*.
  - Inte 100% korrekt, men bra nog
- Varje objekt som kan kollidera med något, ska ha en egen boundingbox.
- Vi kollar efter överlappning
- Dyrt, mycket att kolla mot



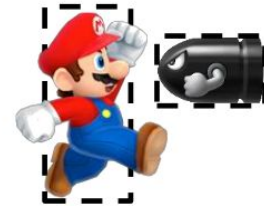
Ingen kollision



Kollision!



Falsk kollision



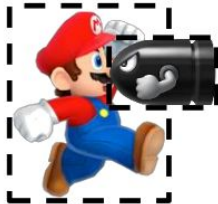
Lösning: mindre bounding box

# Kollisionshantering forts

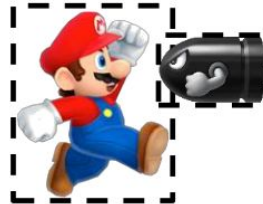
- För att få boundingboxen för vårt objekt - `sf::sprite::getGlobalBounds()`
  - Anropas på både Mario och kulan
- För att avgöra om bounding boxarna har kolliderat, anropar vi `På någon av boundingboxarna` - spelar ingen roll vilken.
  - `sf::Rect::Intersects(sf::Rect)`
- `mario.getGlobalBounds().Intersects(kula.getGlobalBounds())`
  - Returnerar bool



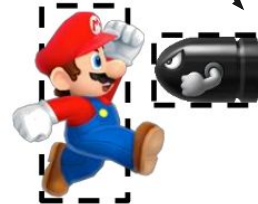
*Ingen kollision*



*Kollision!*



*Falsk kollision*



*Lösning: mindre*



# Allt är inte rektanglar, hur gör vi med cirklar?

- Två alternativ, vi fuskar, och hanterar det som boundingboxes
- Eller så använder vi matte och google, dock inget krav, det ligger utanför kursen.



# Hur bör kontrollen utföras

- Kollisionshantering
  - Där informationen finns
    - Exempelvis, projektil - fiende
      - Game-klassen vet om alla projektiler, och alla fiender
      - Game kan då hämta ut alla Fiendens boundingboxes och utföra kontrollen.



## Om game hanterar kollission

```
63 void Game::update(float)
64 {
65
66     for (int i = 0; i < nrOfProjectiles; ++i)
67     {
68         sf::Rect tempRect = projectiles[i]->getGlobalBounds();
69         for (int j = 0; j < nrOfAliens; ++j)
70             if (aliens[j]->getGlobalBounds().Intersects(tempRect))
71             {
72                 aliens[j]->kill();
73                 projectiles[i]->destroy();
74             }
75     }
```



# Att ha allt i main blir väldigt rörigt

- Det bättre / ett måste är att ha en sk. Game-class
- Game kan liknas vid “Scene” i spelmotorer
  - Vars ansvar är att Uppdatera objekt och rendera
  - Håller i alla objekt som berör spelet
    - Våra egendefinierade objekt
    - Allt
  - Ansvarar även för att skapa och initialisera alla objekt
    - Och att frigöra dem, om de allokerats på heapen.
- Börja inte koda i main för att sedan “flytta” över koden till Game. Börja i game.

# GameClass kod

```
7 class Game : public sf::Drawable
8 {
9     public:
10         Game();
11
12         void Update(float dt);
13
14     private:
15         sf::Texture mBackgroundTex;
16         sf::Sprite mBackgroundSprite;
17         Player mPlayer;
18
19         void draw(sf::RenderTarget &target, sf::RenderStates states) const;
20 };
```

```
4 Game::Game() :mPlayer(1)
5 {
6     if (!mBackgroundTex.loadFromFile("../Resources/background.jpg"))
7     {
8         // Handle error: Print error message.
9         std::cout << "ERROR: Background image could not be loaded.\n---" << std::endl;
10    }
11    mBackgroundSprite.setTexture(mBackgroundTex);
12 }
13
14 void Game::Update(float dt)
15 {
16     // Make sure everything in the game is updated (if needed).
17     mPlayer.Update(dt);
18 }
19
20 void Game::draw(sf::RenderTarget &target, sf::RenderStates states) const
21 {
22     // Make sure everything in the game is drawn.
23     target.draw(mBackgroundSprite, states);
24     target.draw(mPlayer, states);
25 }
```





# Game-klass, main

```
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();
        }

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

    return 0;
}
```



# Hur bör koden struktureras

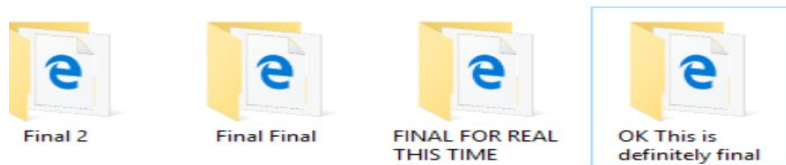
- Main
  - Så lite som möjligt, helst *bara* game klassen, och gameloopen samt fönsterhantering
- Objekt
  - Allt som berör objektet, dess logik, attribut.
    - Inputhantering
    - Hur objektet rör sig
- Input
  - I objektet, om det bara är ett objekt som påverkas, tex spelarobjektet.



I prefer the real version control



I said the *real* version control



Perfection



# Github - ingår inte kursen, men väldigt bra.

<https://guides.github.com/activities/hello-world/>

[https://www.youtube.com/results?search\\_query=github+desktop+tutorial](https://www.youtube.com/results?search_query=github+desktop+tutorial)



**Frågor?**