

## R-Type - Engine

Generated by Doxygen 1.9.1



<b>1 Engine</b>	<b>1</b>
1.1 Compilation	1
1.1.1 Linux	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 Archetypes Class Reference	7
4.2 Color Class Reference	7
4.2.1 Detailed Description	8
4.2.2 Constructor & Destructor Documentation	9
4.2.2.1 Color() [1/2]	9
4.2.2.2 Color() [2/2]	9
4.2.2.3 ~Color()	9
4.2.3 Member Function Documentation	10
4.2.3.1 fromSFMLColor()	10
4.2.3.2 getAlpha()	10
4.2.3.3 getBlue()	11
4.2.3.4 getGreen()	11
4.2.3.5 getRed()	11
4.2.3.6 operator sf::Color()	12
4.2.3.7 setAlpha()	12
4.2.3.8 setBlue()	12
4.2.3.9 setGreen()	13
4.2.3.10 setRed()	13
4.3 Components Class Reference	13
4.3.1 Detailed Description	14
4.3.2 Constructor & Destructor Documentation	14
4.3.2.1 Components()	15
4.3.2.2 ~Components()	15
4.3.3 Member Function Documentation	15
4.3.3.1 getBit()	15
4.3.3.2 init()	16
4.3.3.3 update()	16
4.4 DrawableComponent Class Reference	16
4.4.1 Detailed Description	17
4.4.2 Constructor & Destructor Documentation	17
4.4.2.1 ~DrawableComponent()	17
4.4.3 Member Function Documentation	18

4.4.3.1 draw()	18
4.5 Entity Class Reference	18
4.5.1 Detailed Description	20
4.5.2 Constructor & Destructor Documentation	21
4.5.2.1 Entity() [1/2]	21
4.5.2.2 Entity() [2/2]	21
4.5.2.3 ~Entity()	21
4.5.3 Member Function Documentation	22
4.5.3.1 addComponent()	22
4.5.3.2 addDrawable()	22
4.5.3.3 applyDeferredEntity()	23
4.5.3.4 drawEntity()	23
4.5.3.5 getActive()	23
4.5.3.6 getBit()	24
4.5.3.7 getComponent()	24
4.5.3.8 getComponentArrays()	24
4.5.3.9 getComponentBitset()	25
4.5.3.10 getComponentTypeID()	25
4.5.3.11 getDrawableComponents()	25
4.5.3.12 getName()	26
4.5.3.13 init()	26
4.5.3.14 removeComponent()	27
4.5.3.15 removeDrawable()	27
4.5.3.16 setActive()	27
4.5.3.17 setDeferredEntity()	28
4.5.3.18 setName()	28
4.5.3.19 update()	28
4.6 EntityManager Class Reference	29
4.6.1 Constructor & Destructor Documentation	30
4.6.1.1 EntityManager()	30
4.6.1.2 ~EntityManager()	31
4.6.2 Member Function Documentation	31
4.6.2.1 addEntity()	31
4.6.2.2 getEntities()	32
4.6.2.3 getEntity()	32
4.6.2.4 getEntityMap()	32
4.6.2.5 init()	33
4.7 EntityManagerTest Class Reference	33
4.8 EntityTest Class Reference	35
4.9 EventEngine Class Reference	36
4.9.1 Detailed Description	36
4.9.2 Constructor & Destructor Documentation	37

4.9.2.1 EventEngine()	37
4.9.2.2 ~EventEngine()	37
4.9.3 Member Function Documentation	37
4.9.3.1 addKeyPressed()	37
4.9.3.2 addMouseButtonPressed()	38
4.9.3.3 addMouseMoved()	38
4.9.3.4 getEvent()	39
4.9.3.5 getKeyPressedMap()	39
4.9.3.6 getKeyStatesMap()	39
4.9.3.7 getMouseButtonPressedMap()	40
4.9.3.8 getMouseMovedMap()	40
4.9.3.9 setKeyStatesMap()	40
4.10 EventTest Class Reference	41
4.11 GameEngine Class Reference	42
4.11.1 Detailed Description	45
4.11.2 Constructor & Destructor Documentation	45
4.11.2.1 GameEngine() [1/2]	45
4.11.2.2 GameEngine() [2/2]	45
4.11.2.3 ~GameEngine()	46
4.11.3 Member Function Documentation	46
4.11.3.1 addWorld()	46
4.11.3.2 eventGameEngine()	47
4.11.3.3 getClock()	47
4.11.3.4 getCurrentWorld()	47
4.11.3.5 getDeltaTime()	48
4.11.3.6 getEventEngine()	48
4.11.3.7 getFilesRessources()	48
4.11.3.8 getMapFont()	49
4.11.3.9 getMapMusic()	49
4.11.3.10 getMapSound()	49
4.11.3.11 getMapTexture()	50
4.11.3.12 getWindow()	50
4.11.3.13 getWorld()	50
4.11.3.14 getWorldMap()	51
4.11.3.15 initialize()	51
4.11.3.16 initializeAllFiles()	51
4.11.3.17 initializeFont()	52
4.11.3.18 initializeMusic()	52
4.11.3.19 initializeMusicFunction()	52
4.11.3.20 initializeSound()	53
4.11.3.21 initializeSoundFunction()	53
4.11.3.22 initializeSpriteFunction()	53

4.11.3.23 initializeTextFunction()	54
4.11.3.24 initializeTexture()	54
4.11.3.25 initializeWorldMap()	54
4.11.3.26 isWindowOpen()	55
4.11.3.27 renderGameEngine()	55
4.11.3.28 run()	55
4.11.3.29 setCurrentWorld()	57
4.11.3.30 setDeltaTime()	57
4.11.3.31 updateGameEngine()	58
4.12 GameEngineTest Class Reference	58
4.13 ITransform Class Reference	60
4.13.1 Detailed Description	60
4.13.2 Constructor & Destructor Documentation	60
4.13.2.1 ~ITransform()	60
4.13.3 Member Function Documentation	61
4.13.3.1 getTransform()	61
4.14 Music Class Reference	61
4.14.1 Detailed Description	63
4.14.2 Constructor & Destructor Documentation	63
4.14.2.1 Music()	63
4.14.2.2 ~Music()	63
4.14.3 Member Function Documentation	63
4.14.3.1 applyDeferredMusic()	64
4.14.3.2 getBit()	64
4.14.3.3 getLoop()	64
4.14.3.4 getMusic()	65
4.14.3.5 getStatus()	65
4.14.3.6 getVolume()	65
4.14.3.7 init()	66
4.14.3.8 pause()	66
4.14.3.9 play()	66
4.14.3.10 setDeferredMusic()	67
4.14.3.11 setLoop()	67
4.14.3.12 setMusic()	67
4.14.3.13 setVolume()	68
4.14.3.14 stop()	68
4.14.3.15 update()	68
4.15 MusicTests Class Reference	69
4.16 Rect< T > Class Template Reference	70
4.16.1 Detailed Description	70
4.16.2 Constructor & Destructor Documentation	71
4.16.2.1 Rect()	71

4.16.2.2 ~Rect()	71
4.16.3 Member Function Documentation	72
4.16.3.1 contains()	72
4.16.3.2 getHeight()	72
4.16.3.3 getLeft()	73
4.16.3.4 getRect()	73
4.16.3.5 getTop()	73
4.16.3.6 getWidth()	74
4.17 Script Class Reference	74
4.18 Sound Class Reference	75
4.18.1 Detailed Description	76
4.18.2 Constructor & Destructor Documentation	76
4.18.2.1 Sound()	76
4.18.2.2 ~Sound()	77
4.18.3 Member Function Documentation	77
4.18.3.1 applyDeferredSound()	77
4.18.3.2 getBit()	77
4.18.3.3 getLoop()	78
4.18.3.4 getSound()	78
4.18.3.5 getVolume()	78
4.18.3.6 init()	79
4.18.3.7 isPlaying()	79
4.18.3.8 pause()	79
4.18.3.9 play()	80
4.18.3.10 setDeferredSound()	80
4.18.3.11 setLoop()	80
4.18.3.12 setSound() [1/2]	81
4.18.3.13 setSound() [2/2]	81
4.18.3.14 setVolume()	82
4.18.3.15 stop()	82
4.18.3.16 update()	82
4.19 SoundTest Class Reference	83
4.20 Sprite Class Reference	84
4.20.1 Detailed Description	85
4.20.2 Constructor & Destructor Documentation	85
4.20.2.1 Sprite()	85
4.20.2.2 ~Sprite()	86
4.20.3 Member Function Documentation	86
4.20.3.1 applyDeferredSprite()	86
4.20.3.2 draw()	86
4.20.3.3 getBit()	87
4.20.3.4 getSprite()	87

4.20.3.5 getTransform()	88
4.20.3.6 init()	88
4.20.3.7 setDeferredSprite()	88
4.20.3.8 setSprite() [1/2]	89
4.20.3.9 setSprite() [2/2]	89
4.20.3.10 setTransform()	90
4.20.3.11 update()	90
4.21 SpriteTest Class Reference	91
4.22 TestColor Class Reference	92
4.23 TestRect Class Reference	93
4.24 TesttoSFML Class Reference	94
4.25 TestVector2 Class Reference	95
4.26 TestWorld Class Reference	96
4.27 Text Class Reference	97
4.27.1 Detailed Description	98
4.27.2 Constructor & Destructor Documentation	99
4.27.2.1 Text()	99
4.27.2.2 ~Text()	99
4.27.3 Member Function Documentation	99
4.27.3.1 applyDeferredText()	99
4.27.3.2 draw()	100
4.27.3.3 getBit()	100
4.27.3.4 getColorFill()	101
4.27.3.5 getColorOutline()	101
4.27.3.6 getFont()	101
4.27.3.7 getSize()	102
4.27.3.8 getStringText()	102
4.27.3.9 getText()	102
4.27.3.10 getTransform()	103
4.27.3.11 init()	103
4.27.3.12 setDeferredText()	103
4.27.3.13 setFillColor()	104
4.27.3.14 setFont()	104
4.27.3.15 setOutlineColor()	104
4.27.3.16 setSize()	105
4.27.3.17 setString()	105
4.27.3.18 setText() [1/2]	105
4.27.3.19 setText() [2/2]	106
4.27.3.20 setTransform()	107
4.27.3.21 update()	107
4.28 TextTest Class Reference	108
4.29 toSFML Class Reference	109



4.29.1 Detailed Description	109
4.29.2 Constructor & Destructor Documentation	109
4.29.2.1 toSFML()	109
4.29.2.2 ~toSFML()	110
4.29.3 Member Function Documentation	110
4.29.3.1 toSFMLRect()	110
4.30 Transform Class Reference	111
4.30.1 Detailed Description	112
4.30.2 Constructor & Destructor Documentation	112
4.30.2.1 Transform()	112
4.30.2.2 ~Transform()	113
4.30.3 Member Function Documentation	113
4.30.3.1 applyDeferredTransform()	113
4.30.3.2 getBit()	113
4.30.3.3 getPosition()	115
4.30.3.4 getRotation()	115
4.30.3.5 getScale()	115
4.30.3.6 getTransform()	116
4.30.3.7 init()	116
4.30.3.8 setDeferredTransform()	116
4.30.3.9 setPosition()	117
4.30.3.10 setRotation()	117
4.30.3.11 setScale()	117
4.30.3.12 setTransform()	119
4.30.3.13 update()	119
4.31 TransformTest Class Reference	120
4.32 Vector2< T > Class Template Reference	121
4.32.1 Detailed Description	121
4.32.2 Constructor & Destructor Documentation	121
4.32.2.1 Vector2() [1/2]	121
4.32.2.2 Vector2() [2/2]	122
4.32.2.3 ~Vector2()	122
4.32.3 Member Function Documentation	123
4.32.3.1 getVector2Struct()	123
4.32.3.2 getX()	123
4.32.3.3 getY()	124
4.32.3.4 setX()	124
4.32.3.5 setY()	124
4.33 World Class Reference	125
4.33.1 Detailed Description	127
4.33.2 Constructor & Destructor Documentation	127
4.33.2.1 World()	127

4.33.2.2 ~World()	127
4.33.3 Member Function Documentation	127
4.33.3.1 addEntityManager()	128
4.33.3.2 createEntities()	128
4.33.3.3 getEntitiesManager()	128
4.33.3.4 getEntityManager()	129
4.33.3.5 getEntityManagerMap()	129
4.33.3.6 getNameWorld()	129
4.33.3.7 init()	130
4.33.3.8 setNameWorld()	130
<b>Index</b>	<b>131</b>

# Chapter 1

## Engine

### 1.1 Compilation

#### 1.1.1 Linux

Use the following command to compile the engine:

```
cmake -Bbuild  
make -Cbuild
```

Use the following command to compile the engine and its tests:

```
cmake -Bbuild -DBUILD_TESTS=ON  
make -Cbuild
```

Use the following command for create the package (.tgz or .zip) after compile:

```
cd build  
cpack
```



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Archetypes . . . . .	7
Color . . . . .	7
Components . . . . .	13
Entity . . . . .	18
EntityManager . . . . .	29
World . . . . .	125
GameEngine . . . . .	42
Music . . . . .	61
Sound . . . . .	75
Transform . . . . .	111
Sprite . . . . .	84
Text . . . . .	97
DrawableComponent . . . . .	16
Sprite . . . . .	84
Text . . . . .	97
EventEngine . . . . .	36
GameEngine . . . . .	42
ITransform . . . . .	60
Sprite . . . . .	84
Text . . . . .	97
Rect< T > . . . . .	70
Rect< float > . . . . .	70
Script . . . . .	74
testing::Test	
EntityManagerTest . . . . .	33
EntityTest . . . . .	35
EventTest . . . . .	41
GameEngineTest . . . . .	58
MusicTests . . . . .	69
SoundTest . . . . .	83
SpriteTest . . . . .	91
TestColor . . . . .	92
TestRect . . . . .	93
TestVector2 . . . . .	95

TestWorld . . . . .	96
TesttoSFML . . . . .	94
TextTest . . . . .	108
TransformTest . . . . .	120
toSFML . . . . .	109
Sprite . . . . .	84
Vector2< T > . . . . .	121
Vector2< float > . . . . .	121

## Chapter 3

# Class Index

### 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Archetypes	7
Color	
Color class: <a href="#">Color</a> is a class that use for the color in game	7
Components	
Components class: <a href="#">Components</a> is a class that represents a component in the game	13
DrawableComponent	
DrawableComponent class: <a href="#">DrawableComponent</a> is a class that represents a drawable component in the game	16
Entity	
Entity class: <a href="#">Entity</a> is a class that represents an entity in the game	18
EntityManager	29
EntityManagerTest	33
EntityTest	35
EventEngine	
EventEngine class: <a href="#">EventEngine</a> is a class that represents the event engine of the game	36
EventTest	41
GameEngine	
GameEngine class: <a href="#">GameEngine</a> is a class that represents the game engine	42
GameEngineTest	58
ITransform	
ITransform class: <a href="#">ITransform</a> is a class that represents an interface of the Component <a href="#">Transform</a>	60
Music	
Music class: <a href="#">Music</a> is a class that represents the music in the world	61
MusicTests	69
Rect< T >	
Rect class: <a href="#">Rect</a> is a class that represents a rectangle	70
Script	74
Sound	
Sound class: <a href="#">Sound</a> is a class that represents the sound properties of a Component	75
SoundTest	83
Sprite	
Sprite class: <a href="#">Sprite</a> is a class that represents the rendering properties of a Component	84
SpriteTest	91
TestColor	92
TestRect	93

TesttoSFML	94
TestVector2	95
TestWorld	96
Text	
Text class: <a href="#">Text</a> is a class that represents the text in the world	97
TextTest	108
toSFML	
ToSFML class: <a href="#">toSFML</a> is a class that convert some class into SFML class	109
Transform	
Transform class: <a href="#">Transform</a> is a class that represents the transform of a Component	111
TransformTest	120
Vector2< T >	
Vector class: <a href="#">Vector</a> is a class that represents a vector in 2 dimensions	121
World	
World class: <a href="#">World</a> is a class that represents the world of the game	125



## Chapter 4

# Class Documentation

### 4.1 Archetypes Class Reference

The documentation for this class was generated from the following file:

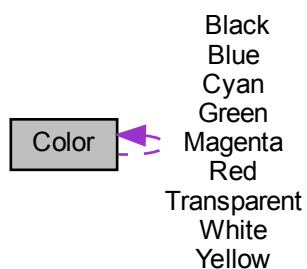
- `src/Archetype/include/Archetypes.h`

### 4.2 Color Class Reference

`Color` class: `Color` is a class that use for the color in game.

```
#include <Color.h>
```

Collaboration diagram for `Color`:



## Public Member Functions

- [Color](#) ()  
*< Represent the Alpha of a color between 0 and 255.*
- [Color](#) (const sf::Color &sfmlColor)  
*Color constructor with sf::Color& as parameter.*
- [~Color](#) ()=default  
*Default override Color destructor.*
- sf::Uint8 [getRed](#) () const  
*getRed(): Get the sf::Uint8 red.*
- sf::Uint8 [getGreen](#) () const  
*getGreen(): Get the sf::Uint8 green.*
- sf::Uint8 [getBlue](#) () const  
*getBlue(): Get the sf::Uint8 blue.*
- sf::Uint8 [getAlpha](#) () const  
*getAlpha(): Get the sf::Uint8 alpha.*
- void [setRed](#) (int newRed)  
*setRed(int): Set the sf::Uint8 red with an int and convert into sf::Unit8 in the function.*
- void [setGreen](#) (int newGreen)  
*setGreen(int): Set the sf::Uint8 green with an int and convert into sf::Unit8 in the function.*
- void [setBlue](#) (int newBlue)  
*setBlue(int): Set the sf::Uint8 blue with an int and convert into sf::Unit8 in the function.*
- void [setAlpha](#) (int newAlpha)  
*setAlpha(int): Set the sf::Uint8 alpha with an int and convert into sf::Unit8 in the function.*
- [operator sf::Color](#) () const  
*operator sf::Color() const: Convert Color classes into sf::Color*

## Static Public Member Functions

- static [Color fromSFMLColor](#) (const sf::Color &sfColor)  
*fromSFMLColor(const sf::Color&): Convert SFML color into Color class.*

## Static Public Attributes

- static const [Color Black](#) = [Color::fromSFMLColor](#)(sf::Color::Black)
- static const [Color White](#) = [Color::fromSFMLColor](#)(sf::Color::White)
- static const [Color Red](#) = [Color::fromSFMLColor](#)(sf::Color::Red)
- static const [Color Green](#) = [Color::fromSFMLColor](#)(sf::Color::Green)
- static const [Color Blue](#) = [Color::fromSFMLColor](#)(sf::Color::Blue)
- static const [Color Yellow](#) = [Color::fromSFMLColor](#)(sf::Color::Yellow)
- static const [Color Magenta](#) = [Color::fromSFMLColor](#)(sf::Color::Magenta)
- static const [Color Cyan](#) = [Color::fromSFMLColor](#)(sf::Color::Cyan)
- static const [Color Transparent](#) = [Color::fromSFMLColor](#)(sf::Color::Transparent)

### 4.2.1 Detailed Description

[Color](#) class: [Color](#) is a class that use for the color in game.

The [Color](#) class manages the color.

## 4.2.2 Constructor & Destructor Documentation

### 4.2.2.1 Color() [1/2]

```
Color::Color ( ) [inline]
```

< Represent the Alpha of a color between 0 and 255.

Default [Color](#) constructor.

Set the default value to "Default" and initialize red, green, blue and alpha to 255 for initialize the color white.

#### Parameters

<i>void</i>	
-------------	--

#### Returns

*void*

### 4.2.2.2 Color() [2/2]

```
Color::Color (
    const sf::Color & sfmlColor ) [inline], [explicit]
```

[Color](#) constructor with `sf::Color&` as parameter.

#### Parameters

<i>sfmlColor</i>	Represent a color preset or no from SFML.
------------------	---

#### Returns

*void*

### 4.2.2.3 ~Color()

```
Color::~~Color ( ) [default]
```

Default override [Color](#) destructor.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

*void*

## 4.2.3 Member Function Documentation

### 4.2.3.1 fromSFMLColor()

```
Color Color::fromSFMLColor (
    const sf::Color & sfColor ) [static]
```

[fromSFMLColor\(const sf::Color&\)](#): Convert SFML color into [Color](#) class.

**Parameters**

<i>sfColor</i>	The color from SFML
----------------	---------------------

**Returns**

[Color](#): [Color](#) class.

### 4.2.3.2 getAlpha()

```
sf::Uint8 Color::getAlpha ( ) const
```

[getAlpha\(\)](#): Get the sf::Uint8 alpha.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

sf::Uint8: The value of alpha.

#### 4.2.3.3 getBlue()

```
sf::Uint8 Color::getBlue ( ) const
```

[getBlue\(\)](#): Get the sf::Uint8 blue.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

sf::Uint8: The value of blue.

#### 4.2.3.4 getGreen()

```
sf::Uint8 Color::getGreen ( ) const
```

[getGreen\(\)](#): Get the sf::Uint8 green.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

sf::Uint8: The value of green.

#### 4.2.3.5 getRed()

```
sf::Uint8 Color::getRed ( ) const
```

[getRed\(\)](#): Get the sf::Uint8 red.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

sf::Uint8: The value of red.

#### 4.2.3.6 operator sf::Color()

```
Color::operator sf::Color ( ) const [explicit]
```

operator sf::Color() const: Convert [Color](#) classes into sf::Color

##### Parameters

<i>void</i>	
-------------	--

##### Returns

sf::Color: Get the [Color](#) in sf::Color

#### 4.2.3.7 setAlpha()

```
void Color::setAlpha (
    int newAlpha )
```

[setAlpha\(int\)](#): Set the sf::Unit8 alpha with an int and convert into sf::Unit8 in the function.

##### Parameters

<i>newAlpha</i>	
-----------------	--

##### Returns

void

#### 4.2.3.8 setBlue()

```
void Color::setBlue (
    int newBlue )
```

[setBlue\(int\)](#): Set the sf::Unit8 blue with an int and convert into sf::Unit8 in the function.

##### Parameters

<i>newBlue</i>	
----------------	--

##### Returns

void

#### 4.2.3.9 setGreen()

```
void Color::setGreen (
    int newGreen )
```

[setGreen\(int\)](#): Set the sf::Uint8 green with an int and convert into sf::Unit8 in the function.

##### Parameters

<i>newGreen</i>	
-----------------	--

##### Returns

void

#### 4.2.3.10 setRed()

```
void Color::setRed (
    int newRed )
```

[setRed\(int\)](#): Set the sf::Uint8 red with an int and convert into sf::Unit8 in the function.

##### Parameters

<i>newRed</i>	Number between 0 and 255.
---------------	---------------------------

##### Returns

void

The documentation for this class was generated from the following files:

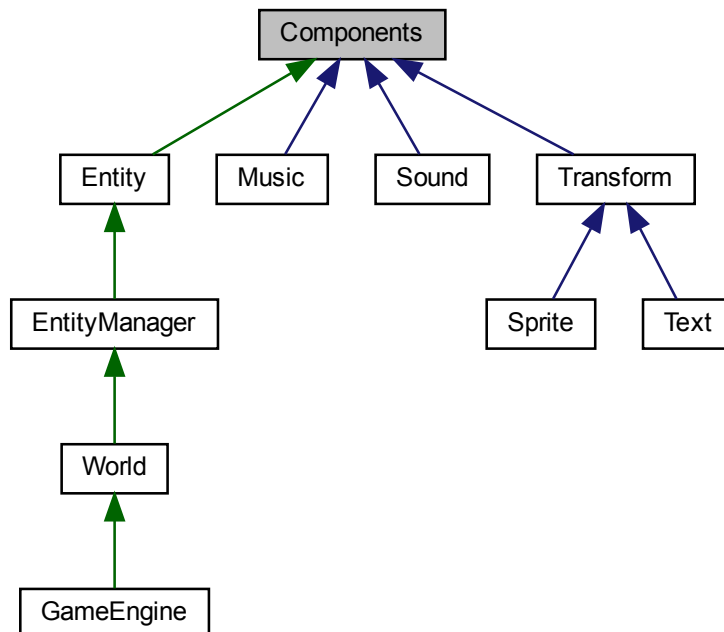
- src/Other/include/Color.h
- src/Other/Color.cpp

## 4.3 Components Class Reference

[Components](#) class: [Components](#) is a class that represents a component in the game.

```
#include <Components.h>
```

Inheritance diagram for Components:



## Public Member Functions

- [Components](#) ()=default  
*Default [Components](#) constructor.*
- virtual [~Components](#) ()=default  
*[Components](#) destructor.*
- virtual bool [init](#) ()=0  
*[init\(\)](#): Initialize the component*
- virtual int [getBit](#) ()=0  
*[getBit\(\)](#): Get the bitmask of the component*
- virtual void [update](#) (sf::Time timeDelta)=0  
*[update\(\)](#): Update the component*

### 4.3.1 Detailed Description

[Components](#) class: [Components](#) is a class that represents a component in the game.

[Components](#) are the building blocks of the game. They are attached to entities and define their behavior.

### 4.3.2 Constructor & Destructor Documentation



#### 4.3.2.1 Components()

```
Components::Components ( ) [default]
```

Default [Components](#) constructor.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

void

#### 4.3.2.2 ~Components()

```
virtual Components::~~Components ( ) [virtual], [default]
```

[Components](#) destructor.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

void

### 4.3.3 Member Function Documentation

#### 4.3.3.1 getBit()

```
virtual int Components::getBit ( ) [pure virtual]
```

[getBit\(\)](#): Get the bitmask of the component

##### Parameters

<i>void</i>	
-------------	--

##### Returns

int: bitmask of the component

Implemented in [Entity](#), [Transform](#), [Text](#), [Sprite](#), [Sound](#), and [Music](#).

#### 4.3.3.2 init()

```
virtual bool Components::init ( ) [pure virtual]
```

[init\(\)](#): Initialize the component

##### Parameters

<i>void</i>	
-------------	--

##### Returns

bool: true if the component is initialized, false otherwise

Implemented in [World](#), [EntityManager](#), [Entity](#), [Transform](#), [Text](#), [Sprite](#), [Sound](#), and [Music](#).

#### 4.3.3.3 update()

```
virtual void Components::update (
    sf::Time timeDelta ) [pure virtual]
```

[update\(\)](#): Update the component

##### Parameters

<i>timeDelta</i>	time elapsed since the last update
------------------	------------------------------------

##### Returns

void

Implemented in [Sound](#), [Music](#), [Entity](#), [Transform](#), [Text](#), and [Sprite](#).

The documentation for this class was generated from the following file:

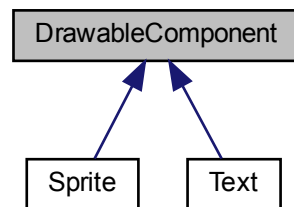
- `src/Components/include/Components.h`

## 4.4 DrawableComponent Class Reference

[DrawableComponent](#) class: [DrawableComponent](#) is a class that represents a drawable component in the game.

```
#include <DrawableComponent.h>
```

Inheritance diagram for DrawableComponent:



## Public Member Functions

- virtual [~DrawableComponent](#) ()=default  
*Default [DrawableComponent](#) constructor.*
- virtual void [draw](#) (sf::RenderWindow &>window) const =0  
*[draw\(\)](#): Draw the component*

### 4.4.1 Detailed Description

[DrawableComponent](#) class: [DrawableComponent](#) is a class that represents a drawable component in the game.

DrawableComponents are components that can be drawn on the screen.

### 4.4.2 Constructor & Destructor Documentation

#### 4.4.2.1 ~DrawableComponent()

```
virtual DrawableComponent::~~DrawableComponent ( ) [virtual], [default]
```

Default [DrawableComponent](#) constructor.

Parameters

<i>void</i>	
-------------	--

**Returns**

void

### 4.4.3 Member Function Documentation

#### 4.4.3.1 draw()

```
virtual void DrawableComponent::draw (  
    sf::RenderWindow & window ) const [pure virtual]
```

[draw\(\)](#): Draw the component

**Parameters**

<i>window</i>	Window to draw the component on
---------------	---------------------------------

**Returns**

void

Implemented in [Text](#), and [Sprite](#).

The documentation for this class was generated from the following file:

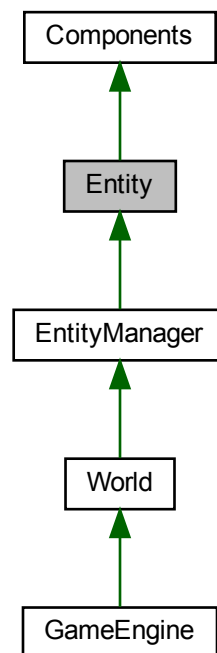
- src/Components/include/DrawableComponent.h

## 4.5 Entity Class Reference

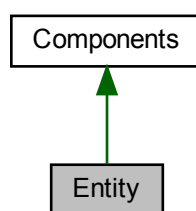
[Entity](#) class: [Entity](#) is a class that represents an entity in the game.

```
#include <entity.h>
```

Inheritance diagram for Entity:



Collaboration diagram for Entity:



## Public Member Functions

- [Entity](#) ()  
*Default [Entity](#) constructor.*
- [Entity](#) (const std::string &nameEntity, [Archetypes](#) newArchetype=[Archetypes](#)())  
*[Entity](#) constructor.*
- [~Entity](#) () override=default

- *Entity* destructor.
- int `getBit` () override  
*getBit(): Get the bit of the Sprite.*
- bool `init` () override  
*init(): Initialize the entity*
- std::string `getName` () const  
*getName(): Get the name of the entity*
- void `setName` (std::string newName)  
*setName(): Set the name of the entity*
- void `update` (sf::Time deltaTime) override  
*update(sf::Time): Update the component Music*
- void `addDrawable` (Components \*component)  
*addDrawable(): Add a drawable component to the entity*
- void `removeDrawable` (Components \*component)  
*removeDrawable(): Remove a drawable component to the entity*
- void `drawEntity` (sf::RenderWindow &window)  
*drawEntity(): Draw the entities*
- template<typename T, typename... TArgs>  
T & `addComponent` (TArgs &&... args)  
*addComponent(): Add a component to the entity*
- template<typename T>  
bool `removeComponent` ()  
*removeComponent(): Remove a component to the entity*
- template<typename T>  
T & `getComponent` ()  
*getComponent(): Get a component from the entity*
- template<typename T>  
std::size\_t `getComponentTypeID` () noexcept  
*getComponentTypeID(): Get a component ID from the entity*
- std::bitset< 6 > `getComponentBitset` () const  
*getComponentBitset(): Get all components bitset from the entity*
- std::vector< DrawableComponent \* > `getDrawableComponents` () const  
*getDrawableComponents(): Get all the drawable components from the entity*
- std::array< Components \*, 6 > `getComponentArrays` () const  
*getComponentArrays(): Get all the components from the entity*
- void `setActive` (bool isActive)  
*setActive(bool): Set the value active for using entity or not*
- bool `getActive` () const  
*getActive(): Get the value active for knowing if entity is using or not.*
- void `setDeferredEntity` (std::function< void()> setter)  
*setDeferredEntity(std::function<void()>): Set the deferred entity.*
- void `applyDeferredEntity` ()  
*setDeferredEntity(std::function<void()>): Set the deferred entity.*

## Additional Inherited Members

### 4.5.1 Detailed Description

`Entity` class: `Entity` is a class that represents an entity in the game.

The `Entity` class manages components associated with the entity.

## 4.5.2 Constructor & Destructor Documentation

### 4.5.2.1 Entity() [1/2]

```
Entity::Entity ( ) [inline]
```

Default [Entity](#) constructor.

#### Parameters

<i>void</i>	
-------------	--

#### Returns

*void*

### 4.5.2.2 Entity() [2/2]

```
Entity::Entity (
    const std::string & nameEntity,
    Archetypes newArchetype = Archetypes() ) [explicit]
```

[Entity](#) constructor.

#### Parameters

<i>nameEntity</i>	name of the entity
<i>newArchetype</i>	archetype of the entity (optional, default = new archetype)

#### Returns

*void*

### 4.5.2.3 ~Entity()

```
Entity::~Entity ( ) [override], [default]
```

[Entity](#) destructor.

#### Parameters

<i>void</i>	
-------------	--

**Returns**

void

### 4.5.3 Member Function Documentation

#### 4.5.3.1 addComponent()

```
template<typename T , typename... TArgs>
template Text & Entity::addComponent< Text > (
    TArgs &&... args )
```

**addComponent()**: Add a component to the entity

**Template Parameters**

<i>T</i>	Type of the component
<i>TArgs</i>	Variadic template for component constructor arguments.

**Parameters**

<i>args</i>	arguments of the component
-------------	----------------------------

**Returns**

T&amp;: reference of the component

#### 4.5.3.2 addDrawable()

```
void Entity::addDrawable (
    Components * component )
```

**addDrawable()**: Add a drawable component to the entity

**Parameters**

<i>component</i>	component to add
------------------	------------------

**Returns**

void



#### 4.5.3.3 applyDeferredEntity()

```
void Entity::applyDeferredEntity ( )
```

[setDeferredEntity\(std::function<void\(\)>\)](#): Set the deferred entity.

##### Parameters

<i>setter</i>	Function that will set the entity.
---------------	------------------------------------

##### Returns

void

#### 4.5.3.4 drawEntity()

```
void Entity::drawEntity (
    sf::RenderWindow & window )
```

[drawEntity\(\)](#): Draw the entities

##### Parameters

<i>window</i>	window where the entities are drawn
---------------	-------------------------------------

##### Returns

void

#### 4.5.3.5 getActive()

```
bool Entity::getActive ( ) const
```

[getActive\(\)](#): Get the value active for knowing if entity is using or not.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

bool: True if the engine use this entity, false otherwise.

#### 4.5.3.6 getBit()

```
int Entity::getBit ( ) [override], [virtual]
```

[getBit\(\)](#): Get the bit of the [Sprite](#).

##### Parameters

<i>void</i>	
-------------	--

##### Returns

int: The bit of the [Sprite](#).

Implements [Components](#).

#### 4.5.3.7 getComponent()

```
template<typename T >
template Text & Entity::getComponent< Text > ( )
```

[getComponent\(\)](#): Get a component from the entity

##### Template Parameters

<i>T</i>	Type of the component
----------	-----------------------

##### Parameters

<i>void</i>	
-------------	--

##### Returns

T&: reference of the component

#### 4.5.3.8 getComponentArrays()

```
std::array< Components *, 6 > Entity::getComponentArrays ( ) const
```

[getComponentArrays\(\)](#): Get all the components from the entity

##### Parameters

<i>void</i>	
-------------	--

**Returns**

`std::array<Components*, 6>`: array of components

**4.5.3.9 GetComponentBitset()**

```
std::bitset< 6 > Entity::GetComponentBitset ( ) const
```

[GetComponentBitset\(\)](#): Get all components bitset from the entity

**Parameters**

<i>void</i>	
-------------	--

**Returns**

`std::bitset<6>`: bitset of the components

**4.5.3.10 GetComponentTypeID()**

```
template<typename T >
template std::size_t Entity::GetComponentTypeID< Text > ( ) [noexcept]
```

[GetComponentTypeID\(\)](#): Get a component ID from the entity

**Template Parameters**

<i>T</i>	Type of the component
----------	-----------------------

**Parameters**

<i>void</i>	
-------------	--

**Returns**

`std::size_t`: id of the component

**4.5.3.11 getDrawableComponents()**

```
std::vector< DrawableComponent * > Entity::getDrawableComponents ( ) const
```

[getDrawableComponents\(\)](#): Get all the drawable components from the entity

**Parameters**

<i>void</i>	
-------------	--

**Returns**

std::vector<DrawableComponent\*>: drawable components of the entity

**4.5.3.12 getName()**

```
std::string Entity::getName ( ) const
```

getName(): Get the name of the entity

**Parameters**

<i>void</i>	
-------------	--

**Returns**

std::string: name of the entity

**4.5.3.13 init()**

```
bool Entity::init ( ) [override], [virtual]
```

init(): Initialize the entity

**Parameters**

<i>void</i>	
-------------	--

**Returns**

bool: true if the entity is initialized, false otherwise

Implements [Components](#).

Reimplemented in [World](#), and [EntityManager](#).

#### 4.5.3.14 removeComponent()

```
template<typename T >
template bool Entity::removeComponent< Text > ( )
```

[removeComponent\(\)](#): Remove a component to the entity

##### Template Parameters

<i>T</i>	Type of the component
----------	-----------------------

##### Returns

T&: reference of the component

#### 4.5.3.15 removeDrawable()

```
void Entity::removeDrawable (
    Components * component )
```

[removeDrawable\(\)](#): Remove a drawable component to the entity

##### Parameters

<i>component</i>	component to remove
------------------	---------------------

##### Returns

void

#### 4.5.3.16 setActive()

```
void Entity::setActive (
    bool isActive )
```

[setActive\(bool\)](#): Set the value active for using entity or not

##### Parameters

<i>isActive</i>	True or false;
-----------------	----------------

##### Returns

void

#### 4.5.3.17 setDeferredEntity()

```
void Entity::setDeferredEntity (
    std::function< void()> setter )
```

[setDeferredEntity\(std::function<void\(\)>\)](#): Set the deferred entity.

##### Parameters

<i>setter</i>	Function that will set the entity.
---------------	------------------------------------

##### Returns

void

#### 4.5.3.18 setName()

```
void Entity::setName (
    std::string newName )
```

[setName\(\)](#): Set the name of the entity

##### Parameters

<i>newName</i>	new name of the entity
----------------	------------------------

##### Returns

void

#### 4.5.3.19 update()

```
void Entity::update (
    sf::Time deltaTime ) [override], [virtual]
```

[update\(sf::Time\)](#): Update the component [Music](#)

##### Parameters

<i>timeDelta</i>	sf::Time of the game.
------------------	-----------------------

**Returns**

void

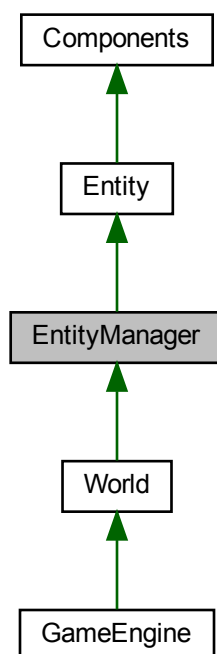
Implements [Components](#).

The documentation for this class was generated from the following files:

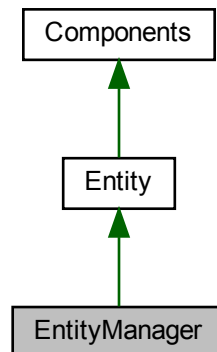
- src/Entity/include/entity.h
- src/Entity/entity.cpp

## 4.6 EntityManager Class Reference

Inheritance diagram for EntityManager:



Collaboration diagram for EntityManager:



## Public Member Functions

- [EntityManager](#) ()=default  
*Default [EntityManager](#) constructor.*
- [~EntityManager](#) () override=default  
*[EntityManager](#) destructor.*
- bool [init](#) () override  
*[initEntityManager\(\)](#): Initialize the [EntityManager](#).*
- [Entity](#) & [addEntity](#) (const std::string &nameEntity, [Archetypes](#) newArchetype=[Archetypes](#)())  
*[addEntity\(\)](#): Create and add a new entity to the entity manager.*
- [Entity](#) & [getEntity](#) (const std::string &nameEntity)  
*[getEntity\(\)](#): Get an entity from the entity manager by its name.*
- std::map< std::string, [Entity](#) \* > [getEntities](#) () const  
*[getEntities\(\)](#): Get the [EntityManager](#)'s entities.*
- std::map< std::string, [Entity](#) \* > [getEntityMap](#) () const  
*[getEntityMap\(\)](#): Get the [EntityManager](#)'s entity map.*

## Additional Inherited Members

### 4.6.1 Constructor & Destructor Documentation

#### 4.6.1.1 EntityManager()

```
EntityManager::EntityManager ( ) [default]
```

Default [EntityManager](#) constructor.



## Parameters

<i>void</i>	
-------------	--

## Returns

void

### 4.6.1.2 ~EntityManager()

```
EntityManager::~~EntityManager ( ) [override], [default]
```

[EntityManager](#) destructor.

## Parameters

<i>void</i>	
-------------	--

## Returns

void

## 4.6.2 Member Function Documentation

### 4.6.2.1 addEntity()

```
Entity & EntityManager::addEntity (
    const std::string & nameEntity,
    Archetypes newArchetype = Archetypes() )
```

[addEntity\(\)](#): Create and add a new entity to the entity manager.

## Template Parameters

<i>T</i>	Type of the entity.
<i>TArgs</i>	Type of the arguments.

## Parameters

<i>args</i>	Arguments of the entity.
-------------	--------------------------

#### 4.6.2.2 getEntities()

```
std::map< std::string, Entity * > EntityManager::getEntities ( ) const
```

[getEntities\(\)](#): Get the [EntityManager](#)'s entities.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

std::map<std::string, Entity \*>: Entities.

#### 4.6.2.3 getEntity()

```
Entity & EntityManager::getEntity (
    const std::string & nameEntity )
```

[getEntity\(\)](#): Get an entity from the entity manager by its name.

##### Template Parameters

<i>T</i>	Type of the entity.
----------	---------------------

##### Parameters

<i>nameEntity</i>	Name of the entity.
-------------------	---------------------

##### Returns

T&: Reference of the entity.

#### 4.6.2.4 getEntityMap()

```
std::map< std::string, Entity * > EntityManager::getEntityMap ( ) const
```

[getEntityMap\(\)](#): Get the [EntityManager](#)'s entity map.

##### Parameters

<i>void</i>	
-------------	--

**Returns**

Entity::EntityMap: [Entity](#) map.

**4.6.2.5 init()**

```
bool EntityManager::init ( ) [override], [virtual]
```

initEntityManager(): Initialize the [EntityManager](#).

**Parameters**

<i>void</i>	
-------------	--

**Returns**

bool: true if the [EntityManager](#) is initialized, false otherwise.

Reimplemented from [Entity](#).

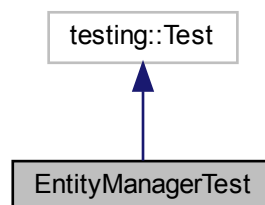
Reimplemented in [World](#).

The documentation for this class was generated from the following files:

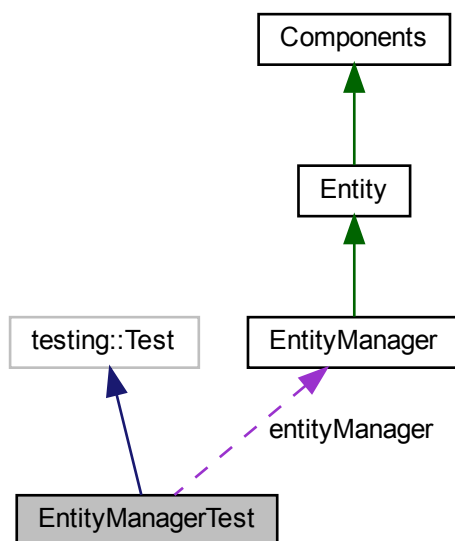
- src/Entity/include/entityManager.h
- src/Entity/entityManager.cpp

## 4.7 EntityManagerTest Class Reference

Inheritance diagram for EntityManagerTest:



Collaboration diagram for EntityManagerTest:



### Protected Member Functions

- void **SetUp** () override
- void **TearDown** () override

### Protected Attributes

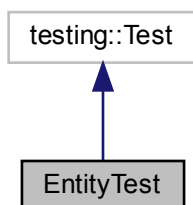
- [EntityManager](#) entityManager {}

The documentation for this class was generated from the following file:

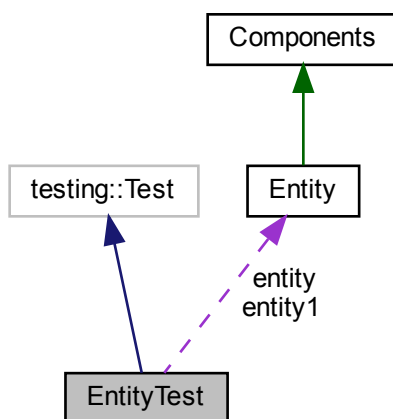
- tests/Entity/TestEntityManager.cpp

## 4.8 EntityTest Class Reference

Inheritance diagram for EntityTest:



Collaboration diagram for EntityTest:



### Protected Attributes

- [Entity](#) entity
- [Entity](#) entity1

The documentation for this class was generated from the following file:

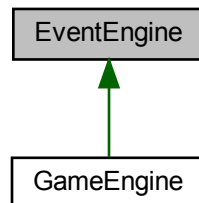
- tests/Entity/TestEntity.cpp

## 4.9 EventEngine Class Reference

[EventEngine](#) class: [EventEngine](#) is a class that represents the event engine of the game.

```
#include <eventEngine.h>
```

Inheritance diagram for EventEngine:



### Public Member Functions

- [EventEngine](#) ()=default  
*Default [EventEngine](#) constructor.*
- virtual [~EventEngine](#) ()=default  
*[EventEngine](#) destructor.*
- sf::Event & [getEvent](#) ()  
*[getEvent\(\)](#): Get the SFML Event.*
- void [addKeyPressed](#) (sf::Keyboard::Key keyboard, const std::function< void()> &function)  
*[addKeyPressed\(\)](#): Add a key pressed to the map.*
- void [addMouseButtonPressed](#) (sf::Mouse::Button mouse, const std::function< void()> &function)  
*[addMouseButtonPressed\(\)](#): Add a mouse button pressed to the map.*
- void [addMouseMoved](#) (const std::string &nameEntity, const std::function< void()> &function)  
*[addMouseMoved\(\)](#): Add a mouse moved to the map.*
- std::map< sf::Keyboard::Key, std::function< void()> > & [getKeyPressedMap](#) ()  
*[getKeyPressedMap\(\)](#): Get the map of the key pressed.*
- std::map< sf::Mouse::Button, std::function< void()> > & [getMouseButtonPressedMap](#) ()  
*[getMouseButtonPressedMap\(\)](#): Get the map of the mouse button pressed.*
- std::map< std::string, std::function< void()> > & [getMouseMovedMap](#) ()  
*[getMouseMovedPressedMap\(\)](#): Get the map of the key pressed.*
- std::map< sf::Keyboard::Key, bool > & [getKeyStatesMap](#) ()  
*[getKeyStatesMap\(\)](#): Get the map of the key states.*
- void [setKeyStatesMap](#) (sf::Keyboard::Key key)  
*[setKeyStatesMap\(sf::Keyboard::Key\)](#): Initialize the map of the key states for the parameter value to false*

### 4.9.1 Detailed Description

[EventEngine](#) class: [EventEngine](#) is a class that represents the event engine of the game.

The [EventEngine](#) class manages the events of the game.

## 4.9.2 Constructor & Destructor Documentation

### 4.9.2.1 EventEngine()

```
EventEngine::EventEngine ( ) [default]
```

Default [EventEngine](#) constructor.

#### Parameters

<i>void</i>	
-------------	--

#### Returns

*void*

### 4.9.2.2 ~EventEngine()

```
virtual EventEngine::~~EventEngine ( ) [virtual], [default]
```

[EventEngine](#) destructor.

#### Parameters

<i>void</i>	
-------------	--

#### Returns

*void*

## 4.9.3 Member Function Documentation

### 4.9.3.1 addKeyPressed()

```
void EventEngine::addKeyPressed (
    sf::Keyboard::Key keyboard,
    const std::function< void()> & function )
```

[addKeyPressed\(\)](#): Add a key pressed to the map.

**Parameters**

<i>keyboard</i>	SFML Keyboard::Key of the key pressed.
<i>function</i>	Function to execute when the key is pressed.

**Returns**

void

**4.9.3.2 addMouseButtonPressed()**

```
void EventEngine::addMouseButtonPressed (
    sf::Mouse::Button mouse,
    const std::function< void()> & function )
```

[addMouseButtonPressed\(\)](#): Add a mouse button pressed to the map.

**Parameters**

<i>mouse</i>	SFML Mouse::Button of the mouse button pressed.
<i>function</i>	Function to execute when the mouse button is pressed.

**Returns**

void

**4.9.3.3 addMouseMoved()**

```
void EventEngine::addMouseMoved (
    const std::string & nameEntity,
    const std::function< void()> & function )
```

[addMouseMoved\(\)](#): Add a mouse moved to the map.

**Parameters**

<i>nameEntity</i>	: Name of the <a href="#">Entity</a> you want.
<i>function</i>	Function to execute when the mouse moved on entity.

**Returns**

void



#### 4.9.3.4 `getEvent()`

```
sf::Event & EventEngine::getEvent ( )
```

[`getEvent\(\)`](#): Get the SFML Event.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

sf::Event: The SFML Event.

#### 4.9.3.5 `getKeyPressedMap()`

```
std::map< sf::Keyboard::Key, std::function< void()> > & EventEngine::getKeyPressedMap ( )
```

[`getKeyPressedMap\(\)`](#): Get the map of the key pressed.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

std::map<sf::Keyboard::Key, std::function<void()>>: The map of the key pressed.

#### 4.9.3.6 `getKeyStatesMap()`

```
std::map< sf::Keyboard::Key, bool > & EventEngine::getKeyStatesMap ( )
```

[`getKeyStatesMap\(\)`](#): Get the map of the key states.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

std::map<sf::Keyboard::Key, bool>&: The map of the key states.

#### 4.9.3.7 getMouseButtonPressedMap()

```
std::map< sf::Mouse::Button, std::function< void()> > & EventEngine::getMouseButtonPressedMap
( )
```

[getMouseButtonPressedMap\(\)](#): Get the map of the mouse button pressed.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

`std::map<sf::Mouse::Button, std::function<void()>>`: The map of the mouse button pressed.

#### 4.9.3.8 getMouseMovedMap()

```
std::map< std::string, std::function< void()> > & EventEngine::getMouseMovedMap ( )
```

[getMouseMovedPressedMap\(\)](#): Get the map of the key pressed.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

`std::map<std::string, std::function<void()>>`: The map of the mouse moved.

#### 4.9.3.9 setKeyStatesMap()

```
void EventEngine::setKeyStatesMap (
    sf::Keyboard::Key key )
```

[setKeyStatesMap\(sf::Keyboard::Key\)](#): Initialize the map of the key states for the parameter value to false

##### Parameters

<i>key</i>	The touch of the keyboard with using SFML.
------------	--

##### Returns

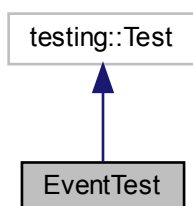
`void`

The documentation for this class was generated from the following files:

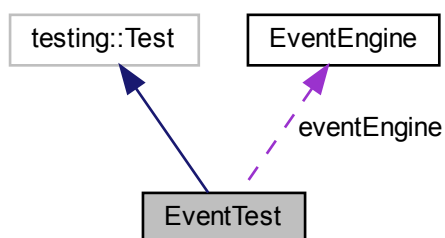
- `src/Event/include/eventEngine.h`
- `src/Event/eventEngine.cpp`

## 4.10 EventTest Class Reference

Inheritance diagram for EventTest:



Collaboration diagram for EventTest:



### Protected Attributes

- [EventEngine](#) `eventEngine`

The documentation for this class was generated from the following file:

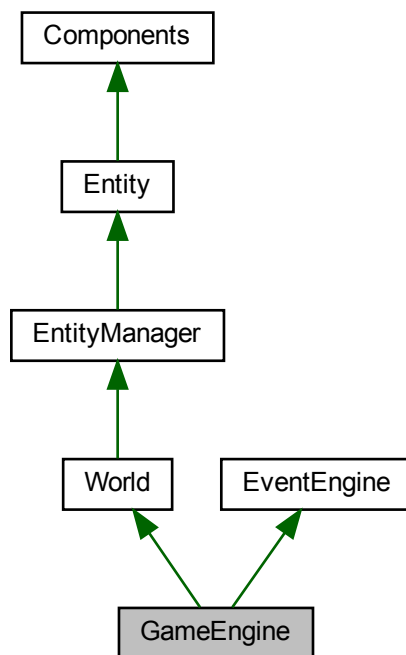
- `tests/Event/TestEvent.cpp`

## 4.11 GameEngine Class Reference

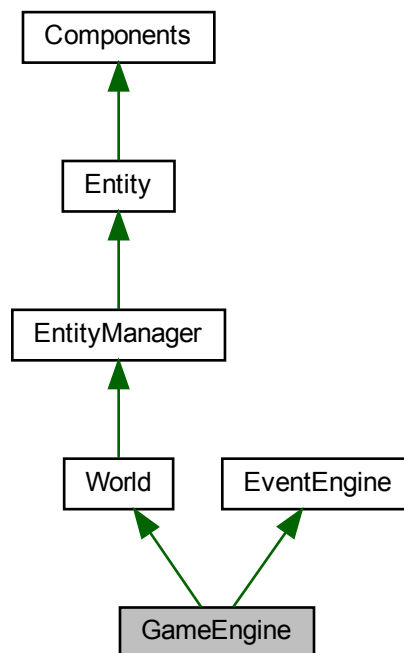
[GameEngine](#) class: [GameEngine](#) is a class that represents the game engine.

```
#include <gameEngine.h>
```

Inheritance diagram for GameEngine:



Collaboration diagram for GameEngine:



## Public Member Functions

- `GameEngine()` = default  
*< Time of the game. Using with the Clock.*
- `GameEngine(sf::VideoMode mode, const sf::String &title, sf::Uint32 style=sf::Style::Default, const sf::ContextSettings &settings=sf::ContextSettings())`  
*GameEngine constructor with parameters.*
- `~GameEngine()` override = default  
*GameEngine destructor.*
- `void run(std::map< std::string, std::unique_ptr< World >> mapWorld, const std::map< std::string, std::string > &pathRessources, const std::string &firstScene)`  
*run(): Run the game engine (with parameters).*
- `void renderGameEngine()`  
*renderGameEngine(): Render the game engine.*
- `void eventGameEngine()`  
*eventGameEngine(): Manage the events of the game engine.*
- `void updateGameEngine()`  
*updateGameEngine(): Update the game engine.*
- `bool isWindowOpen()`  
*isWindowOpen(): Check if the window is open.*
- `void initialize(std::map< std::string, std::unique_ptr< World >> mapWorld, const std::map< std::string, std::string > &pathRessources, const std::string &firstScene)`  
*initialize(): Initialize the game engine.*

- void [initializeSpriteFunction](#) () const  
*initializeSpriteFunction(): Initialize the sprites function.*
- void [initializeSoundFunction](#) () const  
*initializeSoundFunction(): Initialize the sound function.*
- void [initializeMusicFunction](#) () const  
*initializeMusicFunction(): Initialize the music function.*
- void [initializeTextFunction](#) () const  
*initializeFontFunction(): Initialize the font function.*
- void [initializeAllFiles](#) (const std::map< std::string, std::string > &pathResources)  
*initializeAllFiles(): Initialize all the ressources files the engine need.*
- void [initializeTexture](#) (std::string path)  
*initializeTexture(): Initialize the textures with their path.*
- void [initializeSound](#) (std::string path)  
*initializeSound(): Initialize the sound with their path.*
- void [initializeMusic](#) (std::string path)  
*initializeMusic(): Initialize the music with their path.*
- void [initializeFont](#) (std::string path)  
*initializeFont(): Initialize the font with their path.*
- void [initializeWorldMap](#) (std::map< std::string, std::unique\_ptr< [World](#) >> mapWorld)  
*initializeWorldMap(): Initialize the world map.*
- sf::RenderWindow & [getWindow](#) ()  
*getWindow(): Get the window.*
- [EventEngine](#) & [getEventEngine](#) ()  
*getEventEngine(): Get the event engine.*
- void [setCurrentWorld](#) ([World](#) \*world)  
*setCurrentWorld(): Set [GameEngine](#)'s current world.*
- [World](#) \* [getCurrentWorld](#) () const  
*getCurrentWorld(): Get [GameEngine](#)'s current world.*
- [World](#) & [addWorld](#) (const std::string &nameWorld, std::unique\_ptr< [World](#) > world)  
*addWorld(): Add a world to the world map.*
- [World](#) & [getWorld](#) (const std::string &nameWorld)  
*getWorld(): Get a world from the world map with its name.*
- std::map< std::string, std::shared\_ptr< sf::Texture > > [getMapTexture](#) () const  
*getMapTexture(): Get [GameEngine](#)'s map of the textures.*
- std::map< std::string, [World](#) \* > [getWorldMap](#) () const  
*getWorldMap(): Get [GameEngine](#)'s map of the worlds.*
- std::map< std::string, std::shared\_ptr< sf::Music > > [getMapMusic](#) () const  
*getMapMusic(): Get [GameEngine](#)'s map of the music.*
- std::map< std::string, std::shared\_ptr< sf::SoundBuffer > > [getMapSound](#) () const  
*getMapSound(): Get [GameEngine](#)'s map of the sound.*
- std::map< std::string, std::shared\_ptr< sf::Font > > [getMapFont](#) () const  
*getMapFont(): Get [GameEngine](#)'s map of the font.*
- sf::Clock [getClock](#) () const  
*getClock(): Get [GameEngine](#)'s clock.*
- sf::Time [getDeltaTime](#) () const  
*getDeltaTime(): Get [GameEngine](#)'s deltaTime.*
- void [setDeltaTime](#) (sf::Time newTimeDelta)  
*setDeltaTime(): Set [GameEngine](#)'s deltaTime.*

## Static Public Member Functions

- static std::vector< std::string > [getFilesRessources](#) (const std::string &pathDirectory)  
[getFilesRessources\(\)](#): *Get all the ressources type files in the given directory.*

## Additional Inherited Members

### 4.11.1 Detailed Description

[GameEngine](#) class: [GameEngine](#) is a class that represents the game engine.

The [GameEngine](#) class manages the game engine.

### 4.11.2 Constructor & Destructor Documentation

#### 4.11.2.1 [GameEngine\(\)](#) [1/2]

```
GameEngine::GameEngine ( ) [default]
```

< Time of the game. Using with the Clock.

Default [GameEngine](#) constructor.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

void

#### 4.11.2.2 [GameEngine\(\)](#) [2/2]

```
GameEngine::GameEngine (
    sf::VideoMode mode,
    const sf::String & title,
    sf::Uint32 style = sf::Style::Default,
    const sf::ContextSettings & settings = sf::ContextSettings() )
```

[GameEngine](#) constructor with parameters.

## Parameters

<i>mode</i>	Video mode.
<i>type</i>	Type of the graphics ("2D" or "3D").
<i>title</i>	Title of the window.
<i>style</i>	Style of the window (sf::Style::Default by default).
<i>settings</i>	Settings of the window.

## Returns

void

**4.11.2.3   ~GameEngine()**

```
GameEngine::~GameEngine ( ) [override], [default]
```

[GameEngine](#) destructor.

## Parameters

<i>void</i>	
-------------	--

## Returns

void

**4.11.3   Member Function Documentation****4.11.3.1   addWorld()**

```
World & GameEngine::addWorld (
    const std::string & nameWorld,
    std::unique_ptr< World > world )
```

[addWorld\(\)](#): Add a world to the world map.

## Parameters

<i>nameWorld</i>	Name of the world.
<i>world</i>	<a href="#">World</a> to add.



## Returns

[World&](#): The world.

### 4.11.3.2 eventGameEngine()

```
void GameEngine::eventGameEngine ( )
```

[eventGameEngine\(\)](#): Manage the events of the game engine.

## Parameters

<i>void</i>	
-------------	--

## Returns

void

### 4.11.3.3 getClock()

```
sf::Clock GameEngine::getClock ( ) const
```

[getClock\(\)](#): Get [GameEngine](#)'s clock.

## Parameters

<i>void</i>	
-------------	--

## Returns

sf::Clock: [GameEngine](#)'s clock.

### 4.11.3.4 getCurrentWorld()

```
World * GameEngine::getCurrentWorld ( ) const
```

[getCurrentWorld\(\)](#): Get [GameEngine](#)'s current world.

## Parameters

<i>void</i>	
-------------	--

**Returns**

World\*: [GameEngine](#)'s current world.

**4.11.3.5 getDeltaTime()**

```
sf::Time GameEngine::getDeltaTime ( ) const
```

[getDeltaTime\(\)](#): Get [GameEngine](#)'s deltaTime.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

sf::Time: [GameEngine](#)'s deltaTimes.

**4.11.3.6 getEventEngine()**

```
EventEngine & GameEngine::getEventEngine ( )
```

[getEventEngine\(\)](#): Get the event engine.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

[EventEngine](#)&: [GameEngine](#)'s [EventEngine](#).

**4.11.3.7 getFilesRessources()**

```
std::vector< std::string > GameEngine::getFilesRessources (
    const std::string & pathDirectory ) [static]
```

[getFilesRessources\(\)](#): Get all the ressources type files in the given directory.

**Parameters**

<i>pathDirectory</i>	Path of the directory.
----------------------	------------------------

**Returns**

`std::vector<std::string>`: Vector of the ressources type files' names.

**4.11.3.8 getMapFont()**

```
std::map< std::string, std::shared_ptr< sf::Font > > GameEngine::getMapFont ( ) const
```

[getMapFont\(\)](#): Get [GameEngine](#)'s map of the font.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

`std::map<std::string, std::shared_ptr<sf::Font>>`: [GameEngine](#)'s map of the musics.

**4.11.3.9 getMapMusic()**

```
std::map< std::string, std::shared_ptr< sf::Music > > GameEngine::getMapMusic ( ) const
```

[getMapMusic\(\)](#): Get [GameEngine](#)'s map of the music.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

`std::map<std::string, std::shared_ptr<sf::Music>>`: [GameEngine](#)'s map of the musics.

**4.11.3.10 getMapSound()**

```
std::map< std::string, std::shared_ptr< sf::SoundBuffer > > GameEngine::getMapSound ( ) const
```

[getMapSound\(\)](#): Get [GameEngine](#)'s map of the sound.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

`std::map<std::string, std::shared_ptr<sf::SoundBuffer>>`: [GameEngine](#)'s map of the musics.

**4.11.3.11 getMapTexture()**

```
std::map< std::string, std::shared_ptr< sf::Texture > > GameEngine::getMapTexture ( ) const
```

[getMapTexture\(\)](#): Get [GameEngine](#)'s map of the textures.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

`std::map<std::string, std::shared_ptr<sf::Texture>>`: [GameEngine](#)'s map of the textures.

**4.11.3.12 getWindow()**

```
sf::RenderWindow & GameEngine::getWindow ( )
```

[getWindow\(\)](#): Get the window.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

`sf::RenderWindow&`: [GameEngine](#)'s window.

**4.11.3.13 getWorld()**

```
World & GameEngine::getWorld (
    const std::string & nameWorld )
```

[getWorld\(\)](#): Get a world from the world map with its name.

**Parameters**

<i>nameWorld</i>	Name of the world.
------------------	--------------------

## Returns

[World](#)&: [GameEngine](#)'s world.

4.11.3.14 `getWorldMap()`

```
std::map< std::string, World * > GameEngine::getWorldMap ( ) const
```

`getWorldMap()`: Get [GameEngine](#)'s map of the worlds.

## Parameters

<code>void</code>	
-------------------	--

## Returns

`std::map<std::string, World*>`: [GameEngine](#)'s map of the worlds.

4.11.3.15 `initialize()`

```
void GameEngine::initialize (
    std::map< std::string, std::unique_ptr< World >> mapWorld,
    const std::map< std::string, std::string > & pathRessources,
    const std::string & firstScene )
```

`initialize()`: Initialize the game engine.

## Parameters

<i>mapWorld</i>	Map of <a href="#">World</a> classes' unique pointers.
<i>pathRessources</i>	Map of the path of the ressources (assets).
<i>firstScene</i>	Name of the first scene.

## Returns

`void`

4.11.3.16 `initializeAllFiles()`

```
void GameEngine::initializeAllFiles (
    const std::map< std::string, std::string > & pathRessources )
```

`initializeAllFiles()`: Initialize all the ressources files the engine need.

## Parameters

<i>pathResources</i>	Map of the path of the ressources (assets).
----------------------	---

## Returns

void

**4.11.3.17 initializeFont()**

```
void GameEngine::initializeFont (
    std::string path )
```

[initializeFont\(\)](#): Initialize the font with their path.

## Parameters

<i>path</i>	Path of the font file.
-------------	------------------------

## Returns

void

**4.11.3.18 initializeMusic()**

```
void GameEngine::initializeMusic (
    std::string path )
```

[initializeMusic\(\)](#): Initialize the music with their path.

## Parameters

<i>path</i>	Path of the music file.
-------------	-------------------------

## Returns

void

**4.11.3.19 initializeMusicFunction()**

```
void GameEngine::initializeMusicFunction ( ) const
```

[initializeMusicFunction\(\)](#): Initialize the music function.

## Parameters

<i>void</i>	
-------------	--

## Returns

void

**4.11.3.20 initializeSound()**

```
void GameEngine::initializeSound (
    std::string path )
```

[initializeSound\(\)](#): Initialize the sound with their path.

## Parameters

<i>path</i>	Path of the sound file.
-------------	-------------------------

## Returns

void

**4.11.3.21 initializeSoundFunction()**

```
void GameEngine::initializeSoundFunction ( ) const
```

[initializeSoundFunction\(\)](#): Initialize the sound function.

## Parameters

<i>void</i>	
-------------	--

## Returns

void

**4.11.3.22 initializeSpriteFunction()**

```
void GameEngine::initializeSpriteFunction ( ) const
```

[initializeSpriteFunction\(\)](#): Initialize the sprites function.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

void

**4.11.3.23 initializeTextFunction()**

```
void GameEngine::initializeTextFunction ( ) const
```

initializeFontFunction(): Initialize the font function.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

void

**4.11.3.24 initializeTexture()**

```
void GameEngine::initializeTexture (
    std::string path )
```

[initializeTexture\(\)](#): Initialize the textures with their path.

**Parameters**

<i>path</i>	Path of the texture.
-------------	----------------------

**Returns**

void

**4.11.3.25 initializeWorldMap()**

```
void GameEngine::initializeWorldMap (
    std::map< std::string, std::unique_ptr< World >> mapWorld )
```

[initializeWorldMap\(\)](#): Initialize the world map.



## Parameters

<i>mapWorld</i>	Map of <a href="#">World</a> classes' unique pointers.
-----------------	--

## Returns

void

**4.11.3.26 isWindowOpen()**

```
bool GameEngine::isWindowOpen ( )
```

[isWindowOpen\(\)](#): Check if the window is open.

## Parameters

<i>void</i>	
-------------	--

## Returns

bool: True if the window is open, false otherwise.

**4.11.3.27 renderGameEngine()**

```
void GameEngine::renderGameEngine ( )
```

[renderGameEngine\(\)](#): Render the game engine.

## Parameters

<i>void</i>	
-------------	--

## Returns

void

**4.11.3.28 run()**

```
void GameEngine::run (
    std::map< std::string, std::unique_ptr< World >> mapWorld,
```

```
const std::map< std::string, std::string > & pathResources,  
const std::string & firstScene )
```

[run\(\)](#): Run the game engine (with parameters).

## Parameters

<i>mapWorld</i>	Map of <a href="#">World</a> classes' unique pointers.
<i>pathResources</i>	Map of the path of the ressources (assets).
<i>firstScene</i>	Name of the first scene.

## Returns

void

**4.11.3.29 setCurrentWorld()**

```
void GameEngine::setCurrentWorld (
    World * world )
```

[setCurrentWorld\(\)](#): Set [GameEngine](#)'s current world.

## Parameters

<i>world</i>	<a href="#">World</a> to set.
--------------	-------------------------------

## Returns

void

**4.11.3.30 setDeltaTime()**

```
void GameEngine::setDeltaTime (
    sf::Time newTimeDelta )
```

[setDeltaTime\(\)](#): Set [GameEngine](#)'s deltaTime.

## Parameters

<i>newTimeDelta</i>	New deltaTime for <a href="#">GameEngine</a> 's deltaTime.
---------------------	--

## Returns

void

#### 4.11.3.31 updateGameEngine()

```
void GameEngine::updateGameEngine ( )
```

[updateGameEngine\(\)](#): Update the game engine.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

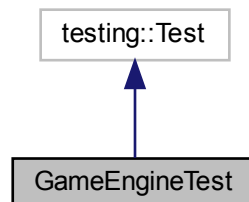
void

The documentation for this class was generated from the following files:

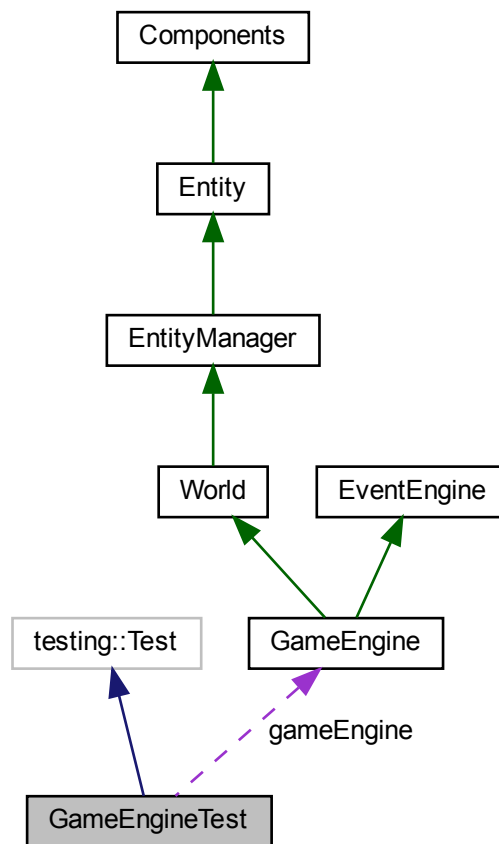
- src/GameEngine/include/gameEngine.h
- src/GameEngine/gameEngine.cpp

## 4.12 GameEngineTest Class Reference

Inheritance diagram for GameEngineTest:



Collaboration diagram for GameEngineTest:



### Protected Member Functions

- `void TearDown ()` override

### Protected Attributes

- `GameEngine * gameEngine`

The documentation for this class was generated from the following file:

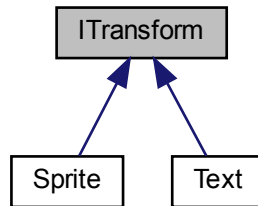
- `tests/GameEngine/TestGameEngine.cpp`

## 4.13 ITransform Class Reference

**ITransform** class: **ITransform** is a class that represents an interface of the Component **Transform**.

```
#include <ITransform.h>
```

Inheritance diagram for ITransform:



### Public Member Functions

- virtual `~ITransform()`=default  
Default Virtual **ITransform** destructor.
- virtual `Transform * getTransform()`=0  
*getTransform(): Get the reference of the component **Transform** of the same **Entity***

#### 4.13.1 Detailed Description

**ITransform** class: **ITransform** is a class that represents an interface of the Component **Transform**.

The **ITransform** interface give to components which need to have a reference to **Transform**

#### 4.13.2 Constructor & Destructor Documentation

##### 4.13.2.1 ~ITransform()

```
virtual ITransform::~~ITransform ( ) [virtual], [default]
```

Default Virtual **ITransform** destructor.

Parameters

<code>void</code>	
-------------------	--

**Returns**

void

### 4.13.3 Member Function Documentation

#### 4.13.3.1 getTransform()

```
virtual Transform* ITransform::getTransform ( ) [pure virtual]
```

**getTransform():** Get the reference of the component [Transform](#) of the same [Entity](#)

Virtual function which get the reference of the [Transform](#) component from the same [Entity](#) when a component need to use [Transform](#). If [Transform](#) don't exist **getTransform()** return nullptr.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

Transform\*: The reference of [Transform](#) or nullptr.

Implemented in [Text](#), and [Sprite](#).

The documentation for this class was generated from the following file:

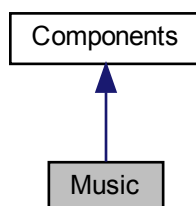
- src/Components/all\_components/include/ITransform.h

## 4.14 Music Class Reference

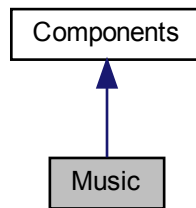
**Music** class: [Music](#) is a class that represents the music in the world.

```
#include <Music.h>
```

Inheritance diagram for Music:



Collaboration diagram for Music:



## Public Member Functions

- **Music** ()=default  
*< Bit of the **Music***
- **~Music** () override=default  
*Default override **Music** destructor.*
- int **getBit** () override  
***getBit()**: Get the bit of the **Music**.*
- void **update** (sf::Time timeDelta) override  
***update(sf::Time)**: Update the component **Music***
- bool **init** () override  
***init()**: Initialize the component.*
- void **setMusic** (std::map< std::string, std::shared\_ptr< sf::Music >> mapMusic, const std::string &name↵  
Music)  
***setMusic(std::map<std::string, std::shared\_ptr<sf::Music>>, const std::string&)**: Initialize the sf::Music of the class.*
- void **setDeferredMusic** (std::function< void()> setter)  
***setDeferredMusic(std::function<void()>)**: Set the deferred function for **Music**.*
- void **applyDeferredMusic** ()  
***applyDeferredMusic()**: Apply the deferred function for **Music***
- std::shared\_ptr< sf::Music > **getMusic** () const  
***getMusic()**: Get the music.*
- void **play** ()  
***play()**: Play the music.*
- void **pause** ()  
***pause()**: Pause the music.*
- void **stop** ()  
***stop()**: Stop the music.*
- void **setLoop** (bool loop)  
***setLoop(bool)**: Set the loop of the music.*
- bool **getLoop** () const  
***getLoop()**: Get if the loop is set to True or False.*
- void **setVolume** (float volume)  
***setVolume(float)**: Set the volume of the music.*
- float **getVolume** () const  
***getVolume()**: Get the volume of the music.*
- sf::SoundSource::Status **getStatus** () const  
***getStatus()**: Get the status of the music. Playing, pause or stop.*



### 4.14.1 Detailed Description

**Music** class: **Music** is a class that represents the music in the world.

The music class manages the music from an **Entity** using SFML.

### 4.14.2 Constructor & Destructor Documentation

#### 4.14.2.1 Music()

```
Music::Music ( ) [default]
```

< Bit of the **Music**

Default **Music** constructor.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

void

#### 4.14.2.2 ~Music()

```
Music::~Music ( ) [override], [default]
```

Default override **Music** destructor.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

void

### 4.14.3 Member Function Documentation

#### 4.14.3.1 `applyDeferredMusic()`

```
void Music::applyDeferredMusic ( )
```

[`applyDeferredMusic\(\)`](#): Apply the deferred function for [Music](#)

##### Parameters

<i>void</i>	
-------------	--

##### Returns

`void`

#### 4.14.3.2 `getBit()`

```
int Music::getBit ( ) [override], [virtual]
```

[`getBit\(\)`](#): Get the bit of the [Music](#).

##### Parameters

<i>void</i>	
-------------	--

##### Returns

`int`: The bit of the [Music](#).

Implements [Components](#).

#### 4.14.3.3 `getLoop()`

```
bool Music::getLoop ( ) const
```

[`getLoop\(\)`](#): Get if the loop is set to True or False.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

`bool`: True or False. If no music set, return false.

#### 4.14.3.4 `getMusic()`

```
std::shared_ptr< sf::Music > Music::getMusic ( ) const
```

`getMusic()`: Get the music.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

`std::shared_ptr<sf::Music>`: The shared ptr of the music.

#### 4.14.3.5 `getStatus()`

```
sf::SoundSource::Status Music::getStatus ( ) const
```

`getStatus()`: Get the status of the music. Playing, pause or stop.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

`sf::SoundSource::Status`: Enumerator of `sf::SoundSource::Status` which is (Stopped, Paused, Playing). If no music set, return Stopped.

#### 4.14.3.6 `getVolume()`

```
float Music::getVolume ( ) const
```

`getVolume()`: Get the volume of the music.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

`float`: Float number that represents the volume between 0 and 100 of the music. If no music set, return -100.

#### 4.14.3.7 init()

```
bool Music::init ( ) [override], [virtual]
```

[init\(\)](#): Initialize the component.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

bool: true if the component is initialized, false otherwise

Implements [Components](#).

#### 4.14.3.8 pause()

```
void Music::pause ( )
```

[pause\(\)](#): Pause the music.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

void

#### 4.14.3.9 play()

```
void Music::play ( )
```

[play\(\)](#): Play the music.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

void

#### 4.14.3.10 setDeferredMusic()

```
void Music::setDeferredMusic (
    std::function< void()> setter )
```

[setDeferredMusic\(std::function<void\(\)>\)](#): Set the deferred function for [Music](#).

##### Parameters

<i>setter</i>	Function that will use <a href="#">Music</a> .
---------------	--

##### Returns

void

#### 4.14.3.11 setLoop()

```
void Music::setLoop (
    bool loop )
```

[setLoop\(bool\)](#): Set the loop of the music.

##### Parameters

<i>loop</i>	True or False.
-------------	----------------

##### Returns

void

#### 4.14.3.12 setMusic()

```
void Music::setMusic (
    std::map< std::string, std::shared_ptr< sf::Music >> mapMusic,
    const std::string & nameMusic )
```

[setMusic\(std::map<std::string, std::shared\\_ptr<sf::Music>>, const std::string&\)](#): Initialize the sf::Music of the class.

##### Parameters

<i>mapMusic</i>	Map of all the music loaded.
<i>nameMusic</i>	Name of the music loaded.

**Returns**

void

**4.14.3.13 setVolume()**

```
void Music::setVolume (
    float volume )
```

[setVolume\(float\)](#): Set the volume of the music.

**Parameters**

<i>volume</i>	Float number that represents the volume between 0 and 100 of the music.
---------------	---

**Returns**

void

**4.14.3.14 stop()**

```
void Music::stop ( )
```

[stop\(\)](#): Stop the music.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

void

**4.14.3.15 update()**

```
void Music::update (
    sf::Time timeDelta ) [override], [virtual]
```

[update\(sf::Time\)](#): Update the component [Music](#)

## Parameters

<i>timeDelta</i>	sf::Time of the game.
------------------	-----------------------

## Returns

void

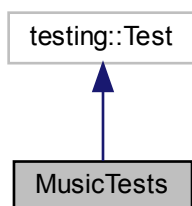
Implements [Components](#).

The documentation for this class was generated from the following files:

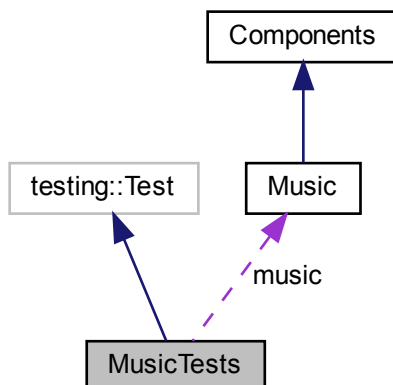
- src/Components/all\_components/include/Music.h
- src/Components/all\_components/Music.cpp

## 4.15 MusicTests Class Reference

Inheritance diagram for MusicTests:



Collaboration diagram for MusicTests:



## Protected Member Functions

- void **SetUp** () override
- void **TearDown** () override

## Protected Attributes

- **Music** music

The documentation for this class was generated from the following file:

- tests/Components/all\_components/TestMusic.cpp

## 4.16 Rect< T > Class Template Reference

**Rect** class: **Rect** is a class that represents a rectangle.

```
#include <Rect.h>
```

### Public Member Functions

- **Rect** (T left, T top, T width, T height)  
*< Rect is the variable you can use for change the data in RectStruct.*
- **~Rect** ()=default  
*Rect destructor.*
- RectStruct **getRect** () const  
*getRect(): Get the using RectStruct.*
- T **getLeft** () const  
*getLeft(): Get the using RectStruct left.*
- T **getTop** () const  
*getTop(): Get the using RectStruct top.*
- T **getWidth** () const  
*getWidth(): Get the using RectStruct width.*
- T **getHeight** () const  
*getHeight(): Get the using RectStruct height.*
- bool **contains** (T x, T y) const  
*contains(): Check if a point is in the rectangle.*

### 4.16.1 Detailed Description

```
template<typename T>
class Rect< T >
```

**Rect** class: **Rect** is a class that represents a rectangle.

This create a rectangle and using for what you want.



## 4.16.2 Constructor & Destructor Documentation

### 4.16.2.1 Rect()

```
template<typename T >
Rect< T >::Rect (
    T left,
    T top,
    T width,
    T height ) [inline]
```

< Rect is the variable you can use for change the data in RectStruct.

Rect constructor with parameters.

#### Template Parameters

<i>T</i>	Type of the rect.
----------	-------------------

#### Parameters

<i>left</i>	Position x.
<i>top</i>	Position y.
<i>width</i>	Width of your rectangle.
<i>height</i>	Height of your rectangle.

#### Returns

void

### 4.16.2.2 ~Rect()

```
template<typename T >
Rect< T >::~~Rect ( ) [default]
```

Rect destructor.

#### Template Parameters

<i>T</i>	Type of the rect.
----------	-------------------

#### Parameters

<i>void</i>	
-------------	--

**Returns**

void

### 4.16.3 Member Function Documentation

#### 4.16.3.1 contains()

```
template<typename T >
template bool Rect< T >::contains (
    T x,
    T y ) const
```

**contains()**: Check if a point is in the rectangle.

**Template Parameters**

<i>T</i>	Type of the rect.
----------	-------------------

**Parameters**

<i>x</i>	: Position x of the point.
<i>y</i>	: Position y of the point.

**Returns**

*T* : *T* is the type you want (float, int,...).

#### 4.16.3.2 getHeight()

```
template<typename T >
template int Rect< T >::getHeight ( ) const
```

**getHeight()**: Get the using RectStruct height.

**Template Parameters**

<i>T</i>	Type of the rect.
----------	-------------------

**Parameters**

<i>void</i>	
-------------	--

**Returns**

T : T is the type you want (float, int,...).

**4.16.3.3 getLeft()**

```
template<typename T >
template int Rect< T >::getLeft ( ) const
```

[getLeft\(\)](#): Get the using RectStruct left.

**Template Parameters**

<i>T</i>	Type of the rect.
----------	-------------------

**Parameters**

<i>void</i>	
-------------	--

**Returns**

T : T is the type you want (float, int,...).

**4.16.3.4 getRect()**

```
template<typename T >
RectStruct Rect< T >::getRect ( ) const [inline]
```

[getRect\(\)](#): Get the using RectStruct.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

[Rect](#)

**4.16.3.5 getTop()**

```
template<typename T >
template int Rect< T >::getTop ( ) const
```

[getTop\(\)](#): Get the using RectStruct top.

#### Template Parameters

<i>T</i>	Type of the rect.
----------	-------------------

#### Parameters

<i>void</i>	
-------------	--

#### Returns

T : T is the type you want (float, int,...).

### 4.16.3.6 getWidth()

```
template<typename T >
template int Rect< T >::getWidth ( ) const
```

[getWidth\(\)](#): Get the using RectStruct width.

#### Template Parameters

<i>T</i>	Type of the rect.
----------	-------------------

#### Parameters

<i>void</i>	
-------------	--

#### Returns

T : T is the type you want (float, int,...).

The documentation for this class was generated from the following files:

- src/Other/include/Rect.h
- src/Other/Rect.cpp

## 4.17 Script Class Reference

### Public Member Functions

- virtual void **execute** ()=0

The documentation for this class was generated from the following file:

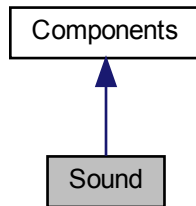
- src/Script/include/Script.h

## 4.18 Sound Class Reference

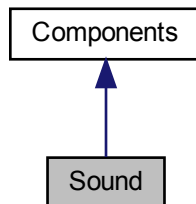
**Sound** class: **Sound** is a class that represents the sound properties of a Component.

```
#include <Sound.h>
```

Inheritance diagram for Sound:



Collaboration diagram for Sound:



### Public Member Functions

- **Sound** ()=default  
    < Bit of the **Sound**.
- **~Sound** () override=default  
    Default override **Sound** destructor.
- int **getBit** () override  
    **getBit()**: Get the bit of the **Sound**.
- void **update** (sf::Time timeDelta) override  
    **update(sf::Time)**: Update the component **Sound**
- bool **init** () override  
    **init()**: Initialize the component.
- void **setSound** (const sf::Sound &sound)

- [`setSound\(const sf::Sound&\)`](#): Set the sound with an existing one. Automatically set the component sound buffer.
- void [`setSound`](#) (std::map< std::string, std::shared\_ptr< sf::SoundBuffer >> mapSound, const std::string &nameSound)

[`setSound\(std::map<std::string, std::shared\_ptr<sf::SoundBuffer>>, const std::string&\)`](#): Initialize the sf::Sound of the class.
- void [`setDeferredSound`](#) (std::function< void()> setter)

[`setDeferredSound\(std::function<void\(\)>\)`](#): Set the deferred function for [Sound](#).
- void [`applyDeferredSound`](#) ()

[`applyDeferredSound\(\)`](#): Apply the deferred function for [Sound](#)
- const sf::Sound & [`getSound`](#) () const

[`getSound\(\)`](#): Get the sound.
- void [`play`](#) ()

[`play\(\)`](#): Play the sound.
- void [`pause`](#) ()

[`pause\(\)`](#): Pause the sound.
- void [`stop`](#) ()

[`stop\(\)`](#): Stop the sound.
- void [`setLoop`](#) (bool loop)

[`setLoop\(bool\)`](#): Set the loop of the sound.
- bool [`getLoop`](#) () const

[`getLoop\(\)`](#): Get if the loop is set to True or False.
- void [`setVolume`](#) (float volume)

[`setVolume\(float\)`](#): Set the volume of the sound.
- float [`getVolume`](#) () const

[`getVolume\(\)`](#): Get the volume of the sound.
- bool [`isPlaying`](#) () const

[`isPlaying\(\)`](#): Check if the sound is currently playing.

### 4.18.1 Detailed Description

[Sound](#) class: [Sound](#) is a class that represents the sound properties of a Component.

The [Sound](#) class manages the sound representation of a Component using SFML.

### 4.18.2 Constructor & Destructor Documentation

#### 4.18.2.1 [Sound\(\)](#)

```
Sound::Sound ( ) [default]
```

< Bit of the [Sound](#).

Default [Sound](#) constructor.

#### Parameters

<code>void</code>	
-------------------	--

## Returns

void

#### 4.18.2.2 ~Sound()

```
Sound::~~Sound ( ) [override], [default]
```

Default override [Sound](#) destructor.

## Parameters

<i>void</i>	
-------------	--

## Returns

void

### 4.18.3 Member Function Documentation

#### 4.18.3.1 applyDeferredSound()

```
void Sound::applyDeferredSound ( )
```

[applyDeferredSound\(\)](#): Apply the deferred function for [Sound](#)

## Parameters

<i>void</i>	
-------------	--

## Returns

void

#### 4.18.3.2 getBit()

```
int Sound::getBit ( ) [override], [virtual]
```

[getBit\(\)](#): Get the bit of the [Sound](#).

**Parameters**

<i>void</i>	
-------------	--

**Returns**

int: The bit of the [Sound](#).

Implements [Components](#).

**4.18.3.3 getLoop()**

```
bool Sound::getLoop ( ) const
```

[getLoop\(\)](#): Get if the loop is set to True or False.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

bool: True or False.

**4.18.3.4 getSound()**

```
const sf::Sound & Sound::getSound ( ) const
```

[getSound\(\)](#): Get the sound.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

const sf::Sound&: The shared ptr of the sound.

**4.18.3.5 getVolume()**

```
float Sound::getVolume ( ) const
```

[getVolume\(\)](#): Get the volume of the sound.



**Parameters**

<i>void</i>	
-------------	--

**Returns**

float: Float number that represents the volume between 0 and 100 of the sound.

**4.18.3.6 init()**

```
bool Sound::init ( ) [override], [virtual]
```

[init\(\)](#): Initialize the component.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

bool: true if the component is initialized, false otherwise

Implements [Components](#).

**4.18.3.7 isPlaying()**

```
bool Sound::isPlaying ( ) const
```

[isPlaying\(\)](#): Check if the sound is currently playing.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

bool: True if the sound is playing, false otherwise.

**4.18.3.8 pause()**

```
void Sound::pause ( )
```

[pause\(\)](#): Pause the sound.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

void

**4.18.3.9 play()**

```
void Sound::play ( )
```

[play\(\)](#): Play the sound.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

void

**4.18.3.10 setDeferredSound()**

```
void Sound::setDeferredSound (
    std::function< void()> setter )
```

[setDeferredSound\(std::function<void\(\)>\)](#): Set the deferred function for [Sound](#).

**Parameters**

<i>setter</i>	Function that will use <a href="#">Sound</a> .
---------------	--

**Returns**

void

**4.18.3.11 setLoop()**

```
void Sound::setLoop (
    bool loop )
```

[setLoop\(bool\)](#): Set the loop of the sound.

## Parameters

<i>loop</i>	True or False.
-------------	----------------

## Returns

void

**4.18.3.12 setSound()** [1/2]

```
void Sound::setSound (
    const sf::Sound & sound )
```

[setSound\(const sf::Sound&\)](#): Set the sound with an existing one. Automatically set the component sound buffer.

## Parameters

<i>sound</i>	SFML <a href="#">Sound</a> for sound.
--------------	---------------------------------------

## Returns

void

**4.18.3.13 setSound()** [2/2]

```
void Sound::setSound (
    std::map< std::string, std::shared_ptr< sf::SoundBuffer >> mapSound,
    const std::string & nameSound )
```

[setSound\(std::map<std::string, std::shared\\_ptr<sf::SoundBuffer>>, const std::string&\)](#): Initialize the sf::Sound of the class.

## Parameters

<i>mapSound</i>	Map of all the sound loaded.
<i>nameSound</i>	Name of the sound loaded.

## Returns

void

#### 4.18.3.14 **setVolume()**

```
void Sound::setVolume (
    float volume )
```

[setVolume\(float\)](#): Set the volume of the sound.

##### Parameters

<i>volume</i>	Float number that represents the volume between 0 and 100 of the sound.
---------------	---

##### Returns

void

#### 4.18.3.15 **stop()**

```
void Sound::stop ( )
```

[stop\(\)](#): Stop the sound.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

void

#### 4.18.3.16 **update()**

```
void Sound::update (
    sf::Time timeDelta ) [override], [virtual]
```

[update\(sf::Time\)](#): Update the component [Sound](#)

##### Parameters

<i>timeDelta</i>	sf::Time of the game.
------------------	-----------------------

##### Returns

void

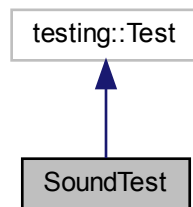
Implements [Components](#).

The documentation for this class was generated from the following files:

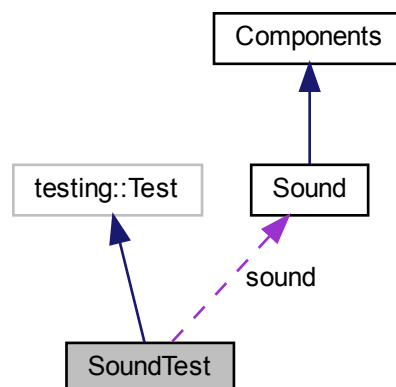
- src/Components/all\_components/include/Sound.h
- src/Components/all\_components/Sound.cpp

## 4.19 SoundTest Class Reference

Inheritance diagram for SoundTest:



Collaboration diagram for SoundTest:



### Protected Member Functions

- void **SetUp** () override
- void **TearDown** () override

## Protected Attributes

- [Sound](#) `sound`

The documentation for this class was generated from the following file:

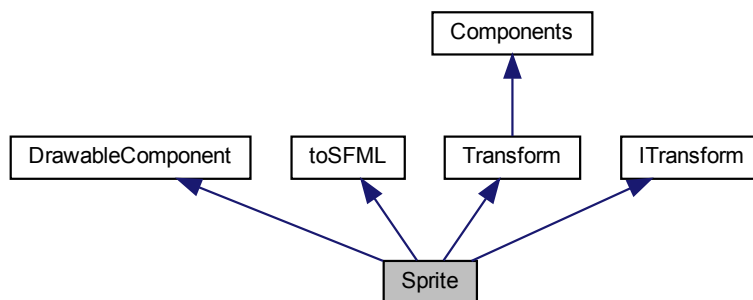
- `tests/Components/all_components/TestSound.cpp`

## 4.20 Sprite Class Reference

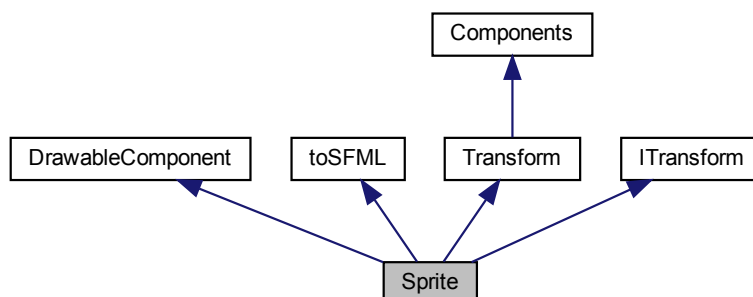
[Sprite](#) class: [Sprite](#) is a class that represents the rendering properties of a Component.

```
#include <Sprite.h>
```

Inheritance diagram for [Sprite](#):



Collaboration diagram for [Sprite](#):



## Public Member Functions

- [Sprite](#) ()  
*< Doing the animation.*
- [~Sprite](#) () override=default  
*Default override [Sprite](#) destructor.*
- [Transform](#) \* [getTransform](#) () override  
*[getTransform\(\)](#): Get the reference to the component [Transform](#).*
- bool [init](#) () override  
*[init\(\)](#): Initialize the component.*
- int [getBit](#) () override  
*[getBit\(\)](#): Get the bit of the [Music](#).*
- void [draw](#) (sf::RenderWindow &>window) const override  
*[draw\(\)](#): Draw the [Sprite](#).*
- void [update](#) (sf::Time deltaTime) override  
*[update\(sf::Time\)](#): Update the component [Music](#)*
- sf::Sprite [getSprite](#) () const  
*[getSprite\(\)](#): Get the SFML [Sprite](#) for rendering.*
- void [setSprite](#) (const sf::Sprite &sprite)  
*[setSprite\(sf::Sprite&\)](#): Set the SFML [Sprite](#) with an existing one for rendering.*
- void [setSprite](#) (std::map< std::string, std::shared\_ptr< sf::Texture >> mapTexture, const std::string &name← Texture, bool animate=false, const std::vector< [Rect](#)< int >> &newFrames=std::vector< [Rect](#)< int >>(), int durationOfFrame=100)  
*Sets the sprite of the component.*
- void [setDeferredSprite](#) (std::function< void()> setter)  
*[setDeferredSprite\(std::function< void\(\)>\)](#): Set the deferred sprite.*
- void [applyDeferredSprite](#) ()  
*[applyDeferredSprite\(\)](#): Apply the deferred sprite.*
- void [setTransform](#) ([Transform](#) &newTransform)  
*[setTransform\(Transform&\)](#): Set the reference of the [Transform](#) component.*

### 4.20.1 Detailed Description

[Sprite](#) class: [Sprite](#) is a class that represents the rendering properties of a Component.

The [Sprite](#) class manages the graphical representation of a Component using SFML.

### 4.20.2 Constructor & Destructor Documentation

#### 4.20.2.1 [Sprite](#)()

```
Sprite::Sprite ( ) [inline]
```

*< Doing the animation.*

Default [Sprite](#) constructor.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

void

**4.20.2.2 ~Sprite()**

```
Sprite::~~Sprite ( ) [override], [default]
```

Default override [Sprite](#) destructor.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

void

**4.20.3 Member Function Documentation****4.20.3.1 applyDeferredSprite()**

```
void Sprite::applyDeferredSprite ( )
```

[applyDeferredSprite\(\)](#): Apply the deferred sprite.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

void

**4.20.3.2 draw()**

```
void Sprite::draw (
    sf::RenderWindow & window ) const [override], [virtual]
```



[draw\(\)](#): Draw the [Sprite](#).

#### Parameters

<i>window</i>	SFML RenderWindow where the <a href="#">Sprite</a> will be drawn.
---------------	---

#### Returns

void

Implements [DrawableComponent](#).

### 4.20.3.3 [getBit\(\)](#)

```
int Sprite::getBit ( ) [override], [virtual]
```

[getBit\(\)](#): Get the bit of the [Music](#).

#### Parameters

<i>void</i>	
-------------	--

#### Returns

int: The bit of the [Music](#).

Implements [Components](#).

### 4.20.3.4 [getSprite\(\)](#)

```
sf::Sprite Sprite::getSprite ( ) const
```

[getSprite\(\)](#): Get the SFML [Sprite](#) for rendering.

#### Parameters

<i>void</i>	
-------------	--

#### Returns

sf::Sprite: SFML [Sprite](#) for rendering

#### 4.20.3.5 getTransform()

```
Transform * Sprite::getTransform ( ) [override], [virtual]
```

[getTransform\(\)](#): Get the reference to the component [Transform](#).

##### Parameters

<i>void</i>	
-------------	--

##### Returns

Transform\*: Reference of [Transform](#)

Implements [ITransform](#).

#### 4.20.3.6 init()

```
bool Sprite::init ( ) [override], [virtual]
```

[init\(\)](#): Initialize the component.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

bool: true if the component is initialized, false otherwise

Implements [Components](#).

#### 4.20.3.7 setDeferredSprite()

```
void Sprite::setDeferredSprite (
    std::function< void()> setter )
```

[setDeferredSprite\(std::function<void\(\)>\)](#): Set the deferred sprite.

##### Parameters

<i>setter</i>	Function that will set the sprite.
---------------	------------------------------------

**Returns**

void

**4.20.3.8 setSprite() [1/2]**

```
void Sprite::setSprite (
    const sf::Sprite & sprite )
```

setSprite(sf::Sprite&): Set the SFML [Sprite](#) with an existing one for rendering.

**Parameters**

<i>sprite</i>	SFML <a href="#">Sprite</a> for rendering
---------------	---

**Returns**

void

**4.20.3.9 setSprite() [2/2]**

```
void Sprite::setSprite (
    std::map< std::string, std::shared_ptr< sf::Texture >> mapTexture,
    const std::string & nameTexture,
    bool animate = false,
    const std::vector< Rect< int >> & newFrames = std::vector<Rect<int>>(),
    int durationOfFrame = 100 )
```

Sets the sprite of the component.

This function sets the sprite of the component using the provided texture map and texture name. Optionally, it can enable animation by providing a vector of frames and the duration of each frame.

**Parameters**

<i>mapTexture</i>	A map of texture names and their corresponding shared pointers to sf::Texture objects.
<i>nameTexture</i>	The name of the texture to set as the sprite.
<i>animate</i>	Flag indicating whether to enable animation or not. Default is false.
<i>newFrames</i>	A vector of frames to use for animation. Default is an empty vector.
<i>durationOfFrame</i>	The duration of each frame in milliseconds. Default is 100 milliseconds.

**Returns**

void

#### 4.20.3.10 setTransform()

```
void Sprite::setTransform (
    Transform & newTransform )
```

[setTransform\(Transform&\)](#): Set the reference of the [Transform](#) component.

##### Parameters

<i>newTransform</i>	Reference of <a href="#">Transform</a> .
---------------------	--

##### Returns

void

#### 4.20.3.11 update()

```
void Sprite::update (
    sf::Time deltaTime ) [override], [virtual]
```

[update\(sf::Time\)](#): Update the component [Music](#)

##### Parameters

<i>timeDelta</i>	sf::Time of the game.
------------------	-----------------------

##### Returns

void

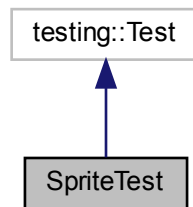
Implements [Components](#).

The documentation for this class was generated from the following files:

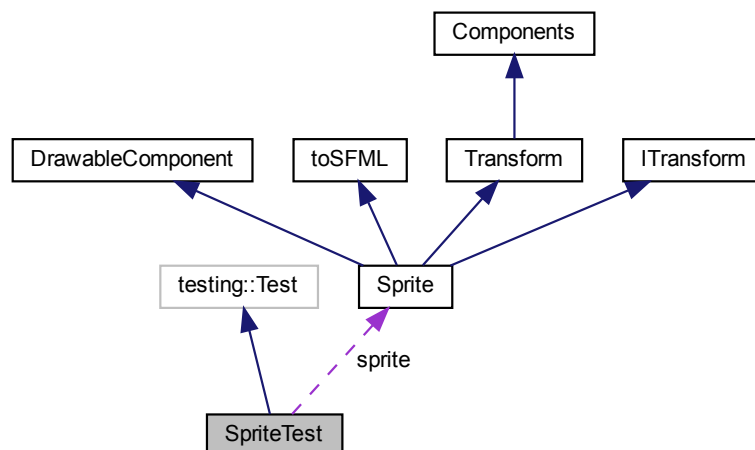
- src/Components/all\_components/include/Sprite.h
- src/Components/all\_components/Sprite.cpp

## 4.21 SpriteTest Class Reference

Inheritance diagram for SpriteTest:



Collaboration diagram for SpriteTest:



### Protected Attributes

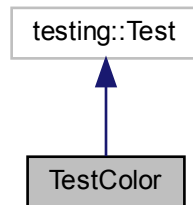
- [Sprite](#) `sprite`

The documentation for this class was generated from the following file:

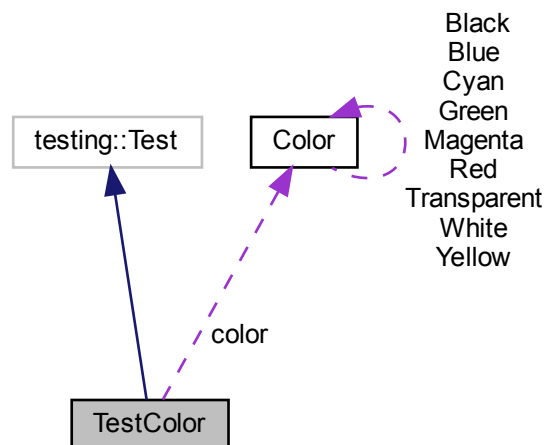
- `tests/Components/all_components/TestSprite.cpp`

## 4.22 TestColor Class Reference

Inheritance diagram for TestColor:



Collaboration diagram for TestColor:



### Protected Member Functions

- void **SetUp** () override
- void **TearDown** () override

### Protected Attributes

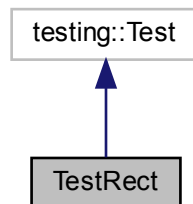
- `Color` `color`

The documentation for this class was generated from the following file:

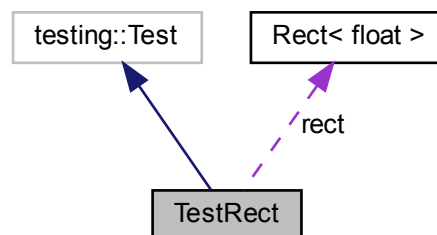
- `tests/Other/TestColor.cpp`

## 4.23 TestRect Class Reference

Inheritance diagram for TestRect:



Collaboration diagram for TestRect:



### Protected Attributes

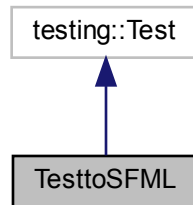
- `Rect< float > rect = Rect<float>(0, 0, 0, 0)`

The documentation for this class was generated from the following file:

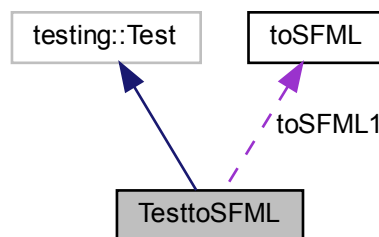
- `tests/Other/TestRect.cpp`

## 4.24 TesttoSFML Class Reference

Inheritance diagram for TesttoSFML:



Collaboration diagram for TesttoSFML:



### Protected Attributes

- `toSFML toSFML1 = toSFML()`

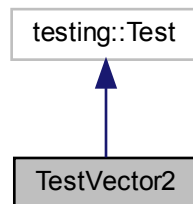
The documentation for this class was generated from the following file:

- `tests/toSFML/TesttoSFML.cpp`

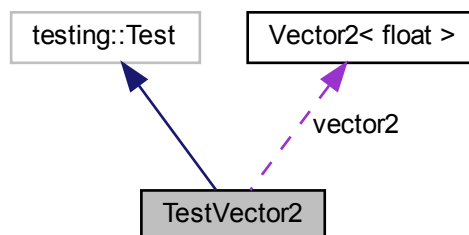


## 4.25 TestVector2 Class Reference

Inheritance diagram for TestVector2:



Collaboration diagram for TestVector2:



### Protected Attributes

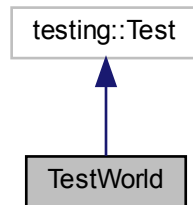
- `Vector2< float > vector2 = Vector2<float>(0, 0)`

The documentation for this class was generated from the following file:

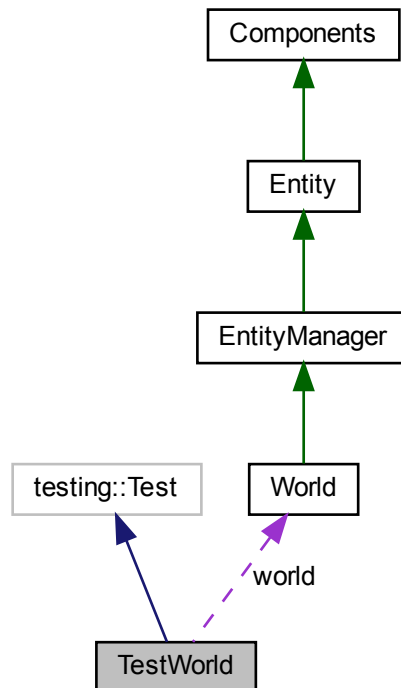
- tests/Other/TestVector2.cpp

## 4.26 TestWorld Class Reference

Inheritance diagram for TestWorld:



Collaboration diagram for TestWorld:



### Protected Attributes

- [World](#) world

The documentation for this class was generated from the following file:

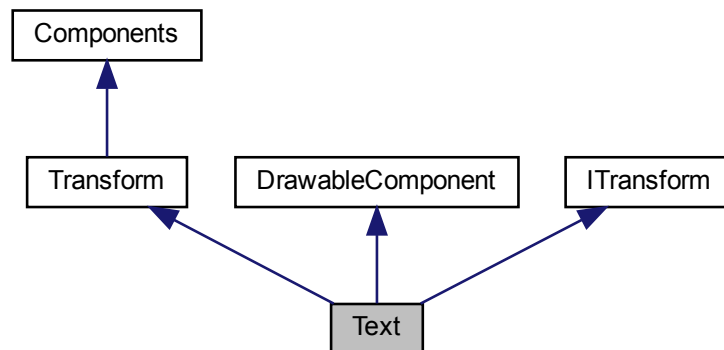
- tests/World/TestWorld.cpp

## 4.27 Text Class Reference

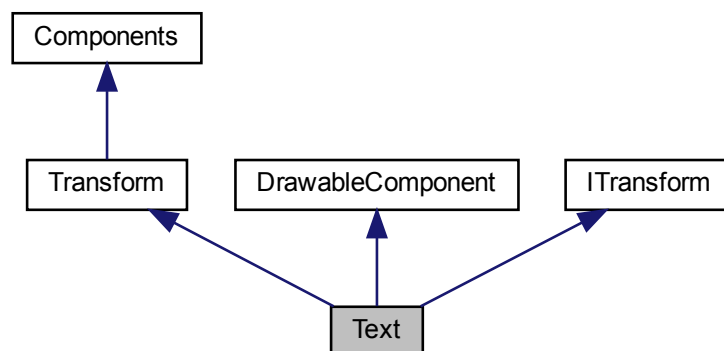
**Text** class: **Text** is a class that represents the text in the world.

```
#include <Text.h>
```

Inheritance diagram for Text:



Collaboration diagram for Text:



### Public Member Functions

- **Text** ()  
    *< Bit of the **Text**.*
- **~Text** () override=default  
    *Default override **Text** destructor.*

- int `getBit` () override  
*getBit(): Get the bit of the `Text`.*
- void `draw` (sf::RenderWindow &window) const override  
*draw(): Draw the `Text`.*
- void `update` (sf::Time deltaTime) override  
*update(sf::Time): Update the component `Text`*
- bool `init` () override  
*init(): Initialize the component.*
- void `setText` (std::map< std::string, std::shared\_ptr< sf::Font >> mapFont, const std::string &nameFont, const std::string &newStringText, int sizeText, `Color` fillColor)  
*Sets the text of the component.*
- void `setText` (std::map< std::string, std::shared\_ptr< sf::Font >> mapFont, const std::string &nameFont, const std::string &newStringText, int sizeText, `Color` fillColor, `Color` outlineColor)  
*Sets the text of the component.*
- void `setFont` (std::map< std::string, std::shared\_ptr< sf::Font >> mapFont, const std::string &nameFont)  
*setFont(std::map<std::string, std::shared\_ptr<sf::Font>>, const std::string&): Set the font of `Text`.*
- void `setString` (const std::string &newStringText)  
*setString(const std::string&): Set the string of `Text`.*
- void `setSize` (int sizeText)  
*setSize(int): Set the size of `Text`.*
- void `setOutlineColor` (`Color` outlineColor)  
*setOutlineColor(Color): Set the outline color of `Text`.*
- void `setFillColor` (`Color` fillColor)  
*setFillColor(Color): Set the fill color of `Text`.*
- sf::Text `getText` () const  
*getText(): Get the `Text`.*
- sf::Font `getFont` () const  
*getFont(): Get the `Font`.*
- std::string `getStringText` () const  
*getStringText(): Get the string.*
- int `getSize` () const  
*getSize(): Get the size.*
- `Color` `getColorFill` () const  
*getColorFill(): Get the fill color.*
- `Color` `getColorOutline` () const  
*getColorOutline(): Get the outline color.*
- `Transform` \* `getTransform` () override  
*getTransform(): Get the reference to the component `Transform`.*
- void `setTransform` (`Transform` &newTransform)  
*setTransform(Transform&): Set the reference of the `Transform` component.*
- void `setDeferredText` (std::function< void()> setter)  
*setDeferredText(std::function<void()>): Set the deferred text.*
- void `applyDeferredText` ()  
*applyDeferredText(): Apply the deferred text.*

### 4.27.1 Detailed Description

`Text` class: `Text` is a class that represents the text in the world.

The text class manages the text from an `Entity` using SFML.

## 4.27.2 Constructor & Destructor Documentation

### 4.27.2.1 Text()

```
Text::Text ( ) [inline]
```

< Bit of the [Text](#).

Default [Text](#) constructor.

#### Parameters

<i>void</i>	
-------------	--

#### Returns

void

### 4.27.2.2 ~Text()

```
Text::~Text ( ) [override], [default]
```

Default override [Text](#) destructor.

#### Parameters

<i>void</i>	
-------------	--

#### Returns

void

## 4.27.3 Member Function Documentation

### 4.27.3.1 applyDeferredText()

```
void Text::applyDeferredText ( )
```

[applyDeferredText\(\)](#): Apply the deferred text.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

void

**4.27.3.2 draw()**

```
void Text::draw (
    sf::RenderWindow & window ) const [override], [virtual]
```

[draw\(\)](#): Draw the [Text](#).

**Parameters**

<i>window</i>	SFML <a href="#">RenderWindow</a> where the <a href="#">Text</a> will be drawn.
---------------	---

**Returns**

void

Implements [DrawableComponent](#).

**4.27.3.3 getBit()**

```
int Text::getBit ( ) [override], [virtual]
```

[getBit\(\)](#): Get the bit of the [Text](#).

**Parameters**

<i>void</i>	
-------------	--

**Returns**

int: The bit of the [Text](#).

Implements [Components](#).

#### 4.27.3.4 getColorFill()

```
Color Text::getColorFill ( ) const
```

[getColorFill\(\)](#): Get the fill color.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

[Color](#): Fill color of the text.

#### 4.27.3.5 getColorOutline()

```
Color Text::getColorOutline ( ) const
```

[getColorOutline\(\)](#): Get the outline color.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

[Color](#): Outline color of the text.

#### 4.27.3.6 getFont()

```
sf::Font Text::getFont ( ) const
```

[getFont\(\)](#): Get the Font.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

[sf::Font](#): Font of the [Text](#).

#### 4.27.3.7 getSize()

```
int Text::getSize ( ) const
```

[getSize\(\)](#): Get the size.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

int: int number that represents size of the text.

#### 4.27.3.8 getStringText()

```
std::string Text::getStringText ( ) const
```

[getStringText\(\)](#): Get the string.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

std::string: String of the text.

#### 4.27.3.9 getText()

```
sf::Text Text::getText ( ) const
```

[getText\(\)](#): Get the [Text](#).

##### Parameters

<i>void</i>	
-------------	--

##### Returns

sf::Text: [Text](#) for draw.



#### 4.27.3.10 getTransform()

```
Transform * Text::getTransform ( ) [override], [virtual]
```

[getTransform\(\)](#): Get the reference to the component [Transform](#).

##### Parameters

<i>void</i>	
-------------	--

##### Returns

Transform\*: Reference of [Transform](#)

Implements [ITransform](#).

#### 4.27.3.11 init()

```
bool Text::init ( ) [override], [virtual]
```

[init\(\)](#): Initialize the component.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

bool: true if the component is initialized, false otherwise. If no [Transform](#) is set, returns false.

Implements [Components](#).

#### 4.27.3.12 setDeferredText()

```
void Text::setDeferredText (
    std::function< void()> setter )
```

[setDeferredText\(std::function<void\(\)>\)](#): Set the deferred text.

##### Parameters

<i>setter</i>	Function that will set the text.
---------------	----------------------------------

**Returns**

void

**4.27.3.13 setFillColor()**

```
void Text::setFillColor (
    Color fillColor )
```

[setFillColor\(Color\)](#): Set the fill color of [Text](#).

**Parameters**

<i>fillColor</i>	<a href="#">Color</a> for the text.
------------------	-------------------------------------

**Returns**

void

**4.27.3.14 setFont()**

```
void Text::setFont (
    std::map< std::string, std::shared_ptr< sf::Font >> mapFont,
    const std::string & nameFont )
```

[setFont\(std::map<std::string, std::shared\\_ptr<sf::Font>>, const std::string&\)](#): Set the font of [Text](#).

**Parameters**

<i>mapFont</i>	Map of all the font loaded.
<i>nameFont</i>	Name of the font loaded.

**Returns**

void

**4.27.3.15 setOutlineColor()**

```
void Text::setOutlineColor (
    Color outlineColor )
```

[setOutlineColor\(Color\)](#): Set the outline color of [Text](#).

## Parameters

<i>outlineColor</i>	Color for the border of the text.
---------------------	-----------------------------------

## Returns

void

**4.27.3.16 setSize()**

```
void Text::setSize (
    int sizeText )
```

[setSize\(int\)](#): Set the size of [Text](#).

## Parameters

<i>sizeText</i>	Size of the text.
-----------------	-------------------

## Returns

void

**4.27.3.17 setString()**

```
void Text::setString (
    const std::string & newStringText )
```

[setString\(const std::string&\)](#): Set the string of [Text](#).

## Parameters

<i>newStringText</i>	String text for draw.
----------------------	-----------------------

## Returns

void

**4.27.3.18 setText()** [1/2]

```
void Text::setText (
    std::map< std::string, std::shared_ptr< sf::Font >> mapFont,
```

```

const std::string & nameFont,
const std::string & newStringText,
int sizeText,
Color fillColor )

```

Sets the text of the component.

This function sets the [Text](#) of the component using the provided font map, the font name, a string for set the [Text](#), the size for the size of character and fill color for color the text.

#### Parameters

<i>mapFont</i>	Map of all the font loaded.
<i>nameFont</i>	Name of the font loaded.
<i>newStringText</i>	String text for draw.
<i>sizeText</i>	Size of the text.
<i>fillColor</i>	<a href="#">Color</a> for the text.

#### Returns

void

### 4.27.3.19 setText() [2/2]

```

void Text::setText (
    std::map< std::string, std::shared_ptr< sf::Font >> mapFont,
    const std::string & nameFont,
    const std::string & newStringText,
    int sizeText,
    Color fillColor,
    Color outlineColor )

```

Sets the text of the component.

This function sets the [Text](#) of the component using the provided font map, the font name, a string for set the [Text](#), the size for the size of character, fill color for color the text and outline color for the border of the text.

#### Parameters

<i>mapFont</i>	Map of all the font loaded.
<i>nameFont</i>	Name of the font loaded.
<i>newStringText</i>	String text for draw.
<i>sizeText</i>	Size of the text.
<i>fillColor</i>	<a href="#">Color</a> for the text.
<i>outlineColor</i>	<a href="#">Color</a> for the border of the text.

#### Returns

void

#### 4.27.3.20 setTransform()

```
void Text::setTransform (
    Transform & newTransform )
```

[setTransform\(Transform&\)](#): Set the reference of the [Transform](#) component.

##### Parameters

<i>newTransform</i>	Reference of <a href="#">Transform</a> .
---------------------	--

##### Returns

void

#### 4.27.3.21 update()

```
void Text::update (
    sf::Time deltaTime ) [override], [virtual]
```

[update\(sf::Time\)](#): Update the component [Text](#)

##### Parameters

<i>timeDelta</i>	sf::Time of the game.
------------------	-----------------------

##### Returns

void

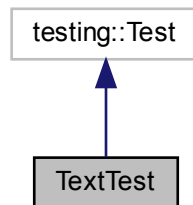
Implements [Components](#).

The documentation for this class was generated from the following files:

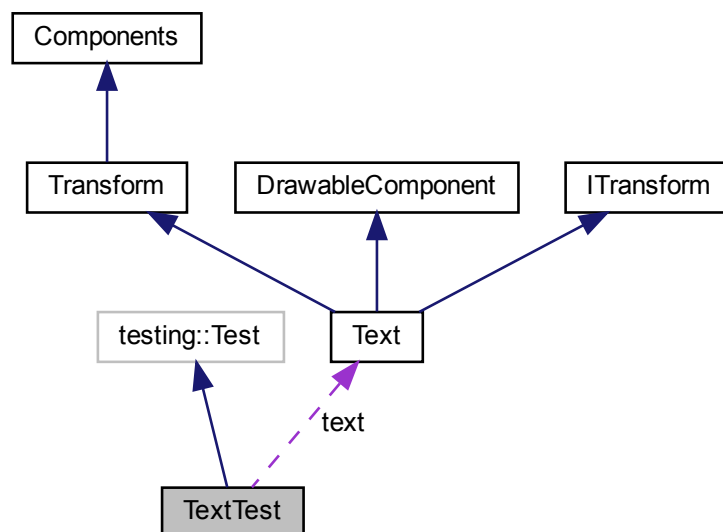
- src/Components/all\_components/include/Text.h
- src/Components/all\_components/Text.cpp

## 4.28 TextTest Class Reference

Inheritance diagram for TextTest:



Collaboration diagram for TextTest:



### Protected Member Functions

- void **SetUp** () override
- void **TearDown** () override

### Protected Attributes

- [Text](#) `text`

The documentation for this class was generated from the following file:

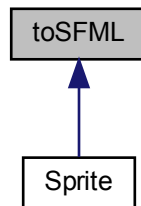
- tests/Components/all\_components/TestText.cpp

## 4.29 toSFML Class Reference

`toSFML` class: `toSFML` is a class that convert some class into SFML class.

```
#include <toSFML.h>
```

Inheritance diagram for `toSFML`:



### Public Member Functions

- `toSFML()`=default  
*Default `toSFML` constructor.*
- `~toSFML()`=default  
*`toSFML` destructor.*
- `template<typename T>`  
`sf::Rect< T > toSFMLRect (Rect< T > rect)`  
*`toSFMLRect()`: Convert your `Rect<T>` into `sf::Rect<T>`.*

### 4.29.1 Detailed Description

`toSFML` class: `toSFML` is a class that convert some class into SFML class.

Convert some class in SFML class.

### 4.29.2 Constructor & Destructor Documentation

#### 4.29.2.1 toSFML()

```
toSFML::toSFML ( ) [default]
```

Default `toSFML` constructor.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

*void*

**4.29.2.2 ~toSFML()**

```
toSFML::~~toSFML ( ) [default]
```

[toSFML](#) destructor.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

*void*

**4.29.3 Member Function Documentation****4.29.3.1 toSFMLRect()**

```
template<typename T >
template sf::Rect< float > toSFML::toSFMLRect (
    Rect< T > rect )
```

[toSFMLRect\(\)](#): Convert your Rect<T> into sf::Rect<T>.

**Template Parameters**

<i>T</i>	Type of the rect.
----------	-------------------

**Parameters**

<i>rect</i>	The rect you want to convert.
-------------	-------------------------------



## Returns

sf::Rect<T>: SFML rect.

The documentation for this class was generated from the following files:

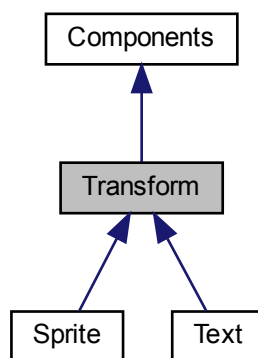
- src/toSFML/include/toSFML.h
- src/toSFML/toSFML.cpp

## 4.30 Transform Class Reference

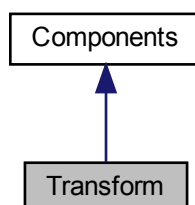
[Transform](#) class: [Transform](#) is a class that represents the transform of a Component.

```
#include <Transform.h>
```

Inheritance diagram for Transform:



Collaboration diagram for Transform:



## Public Member Functions

- [Transform](#) ()  
*Default [Transform](#) constructor.*
- bool [init](#) () override  
*[init\(\)](#): Initialize the component*
- [~Transform](#) () override=default  
*[Transform](#) destructor.*
- void [update](#) (sf::Time deltaTime) override  
*[update\(sf::Time\)](#): Update the component [Music](#)*
- int [getBit](#) () override  
*[getBit\(\)](#): Get the bitmask of the component*
- [Vector2](#)< float > [getPosition](#) () const  
*[getPositionVector\(\)](#): Get the position vector of the component;*
- float [getRotation](#) () const  
*[getRotationVector\(\)](#): Get the rotation vector of the component;*
- [Vector2](#)< float > [getScale](#) () const  
*[getScaleVector\(\)](#): Get the scale vector of the component;*
- TransformStruct [getTransform](#) () const  
*[getTransform\(\)](#): Get the the transform of the component;*
- void [setTransform](#) ([Vector2](#)< float > newPosition, float newRotation, [Vector2](#)< float > newScale)  
*[setTransform\(\)](#): Set the transform of the component;*
- void [setPosition](#) ([Vector2](#)< float > newPosition)  
*[setPosition\(\)](#): Set the transform position of the component;*
- void [setRotation](#) (float newRotation)  
*[setRotation\(\)](#): Set the transform rotation of the component;*
- void [setScale](#) ([Vector2](#)< float > newScale)  
*[setScale\(\)](#): Set the transform scale of the component;*
- void [setDeferredTransform](#) (const std::function< void()> &setter)  
*[setDeferredTransform\(\)](#): Set the deferred transform.*
- void [applyDeferredTransform](#) ()  
*[applyDeferredTransform\(\)](#): Apply the deferred transform.*

### 4.30.1 Detailed Description

[Transform](#) class: [Transform](#) is a class that represents the transform of a Component.

The [Transform](#) class manages the position, rotation and scale of a Component.

### 4.30.2 Constructor & Destructor Documentation

#### 4.30.2.1 Transform()

```
Transform::Transform ( ) [inline]
```

Default [Transform](#) constructor.

## Parameters

<i>void</i>	
-------------	--

## Returns

void

**4.30.2.2 ~Transform()**

```
Transform::~Transform ( ) [override], [default]
```

[Transform](#) destructor.

## Parameters

<i>void</i>	
-------------	--

## Returns

void

**4.30.3 Member Function Documentation****4.30.3.1 applyDeferredTransform()**

```
void Transform::applyDeferredTransform ( )
```

[applyDeferredTransform\(\)](#): Apply the deferred transform.

## Parameters

<i>void</i>	
-------------	--

## Returns

void

**4.30.3.2 getBit()**

```
int Transform::getBit ( ) [override], [virtual]
```

[getBit\(\)](#): Get the bitmask of the component

## Parameters

<i>void</i>	
-------------	--

## Returns

int: bitmask of the component

Implements [Components](#).

### 4.30.3.3 getPosition()

```
Vector2< float > Transform::getPosition ( ) const
```

getPositionVector(): Get the position vector of the component;

## Parameters

<i>void</i>	
-------------	--

## Returns

std::vector<float>: position vector of the component

### 4.30.3.4 getRotation()

```
float Transform::getRotation ( ) const
```

getRotationVector(): Get the rotation vector of the component;

## Parameters

<i>void</i>	
-------------	--

## Returns

std::vector<float>: rotation vector of the component

### 4.30.3.5 getScale()

```
Vector2< float > Transform::getScale ( ) const
```

getScaleVector(): Get the scale vector of the component;

**Parameters**

<i>void</i>	
-------------	--

**Returns**

std::vector<float>: scale vector of the component

**4.30.3.6 getTransform()**

```
Transform::TransformStruct Transform::getTransform ( ) const
```

[getTransform\(\)](#): Get the the transform of the component;

**Parameters**

<i>void</i>	
-------------	--

**Returns**

TransformStruct: struct of the [Transform](#).

**4.30.3.7 init()**

```
bool Transform::init ( ) [override], [virtual]
```

[init\(\)](#): Initialize the component

**Parameters**

<i>void</i>	
-------------	--

**Returns**

bool: true if the component is initialized, false otherwise

Implements [Components](#).

**4.30.3.8 setDeferredTransform()**

```
void Transform::setDeferredTransform (
    const std::function< void()> & setter )
```

[setDeferredTransform\(\)](#): Set the deferred transform.

## Parameters

<i>setter</i>	Function that will set the transform.
---------------	---------------------------------------

## Returns

void

**4.30.3.9 setPosition()**

```
void Transform::setPosition (
    Vector2< float > newPosition )
```

[setPosition\(\)](#): Set the transform position of the component;

## Parameters

<i>newPosition</i>	: the new <a href="#">Vector2&lt;float&gt;</a> position.
--------------------	--

## Returns

void

**4.30.3.10 setRotation()**

```
void Transform::setRotation (
    float newRotation )
```

[setRotation\(\)](#): Set the transform rotation of the component;

## Parameters

<i>newRotation</i>	: the new float rotation.
--------------------	---------------------------

## Returns

void

**4.30.3.11 setScale()**

```
void Transform::setScale (
    Vector2< float > newScale )
```

[setScale\(\)](#): Set the transform scale of the component;



## Parameters

<i>newScale</i>	: the new <a href="#">Vector2&lt;float&gt;</a> scale.
-----------------	---

## Returns

void

**4.30.3.12 setTransform()**

```
void Transform::setTransform (
    Vector2< float > newPosition,
    float newRotation,
    Vector2< float > newScale )
```

[setTransform\(\)](#): Set the transform of the component;

## Parameters

<i>newPosition</i>	: the new <a href="#">Vector2&lt;float&gt;</a> position.
<i>newRotation</i>	: the new float rotation.
<i>newScale</i>	: the new <a href="#">Vector2&lt;float&gt;</a> scale.

## Returns

void

**4.30.3.13 update()**

```
void Transform::update (
    sf::Time deltaTime ) [override], [virtual]
```

[update\(sf::Time\)](#): Update the component [Music](#)

## Parameters

<i>timeDelta</i>	sf::Time of the game.
------------------	-----------------------

## Returns

void

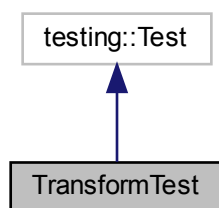
Implements [Components](#).

The documentation for this class was generated from the following files:

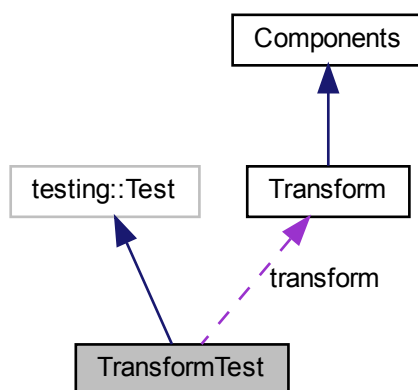
- src/Components/all\_components/include/Transform.h
- src/Components/all\_components/Transform.cpp

## 4.31 TransformTest Class Reference

Inheritance diagram for TransformTest:



Collaboration diagram for TransformTest:



### Protected Member Functions

- void **SetUp** () override
- void **TearDown** () override

## Protected Attributes

- [Transform](#) transform

The documentation for this class was generated from the following file:

- tests/Components/all\_components/TestTransform.cpp

## 4.32 Vector2< T > Class Template Reference

Vector class: Vector is a class that represents a vector in 2 dimensions.

```
#include <Vector2.h>
```

### Public Member Functions

- [Vector2](#) ()  
*< Variable for using the value of the Vector2Struct.*
- [Vector2](#) (T x, T y)  
*Vector2 constructor with parameters.*
- [~Vector2](#) ()=default  
*Vector2 destructor.*
- Vector2Struct [getVector2Struct](#) () const  
*getVector2Struct(): Get the using Vector2Struct.*
- T [getX](#) () const  
*getX(): Get x of Vector2Struct.*
- T [getY](#) () const  
*getY(): Get y of Vector2Struct.*
- void [setX](#) (T newX)  
*setX(): Set x of Vector2Struct.*
- void [setY](#) (T newY)  
*setY(): Set y of Vector2Struct.*

### 4.32.1 Detailed Description

```
template<typename T>
class Vector2< T >
```

Vector class: Vector is a class that represents a vector in 2 dimensions.

This create a vector with 2 value.

### 4.32.2 Constructor & Destructor Documentation

#### 4.32.2.1 Vector2() [1/2]

```
template<typename T >
Vector2< T >::Vector2 ( ) [inline]
```

< Variable for using the value of the Vector2Struct.

[Vector2](#) constructor with parameters.

#### Template Parameters

<i>T</i>	Type of the vector.
----------	---------------------

#### Parameters

<i>x</i>	Position x.
<i>y</i>	Position y.

#### Returns

void

#### 4.32.2.2 Vector2() [2/2]

```
template<typename T >
Vector2< T >::Vector2 (
    T x,
    T y ) [inline]
```

[Vector2](#) constructor with parameters.

#### Template Parameters

<i>T</i>	Type of the vector.
----------	---------------------

#### Parameters

<i>x</i>	Position x.
<i>y</i>	Position y.

#### Returns

void

#### 4.32.2.3 ~Vector2()

```
template<typename T >
Vector2< T >::~~Vector2 ( ) [default]
```

[Vector2](#) destructor.

## Template Parameters

<i>T</i>	Type of the vector.
----------	---------------------

## Parameters

<i>void</i>	
-------------	--

## Returns

void

### 4.32.3 Member Function Documentation

#### 4.32.3.1 `getVector2Struct()`

```
template<typename T >
template Vector2< int >::Vector2Struct Vector2< T >::getVector2Struct ( ) const
```

`getVector2Struct()`: Get the using Vector2Struct.

## Parameters

<i>void</i>	
-------------	--

## Returns

Vector2Struct

#### 4.32.3.2 `getX()`

```
template<typename T >
template int Vector2< T >::getX ( ) const
```

`getX()`: Get x of Vector2Struct.

## Template Parameters

--	--

#### 4.32.3.3 getY()

```
template<typename T >
template int Vector2< T >::getY ( ) const
```

[getY\(\)](#): Get y of Vector2Struct.

##### Template Parameters

--	--

#### 4.32.3.4 setX()

```
template<typename T >
template void Vector2< T >::setX (
    T newX )
```

[setX\(\)](#): Set x of Vector2Struct.

##### Template Parameters

<i>T</i>	Type of the <a href="#">Vector2</a>
----------	-------------------------------------

##### Parameters

<i>newX</i>	The new value of x.
-------------	---------------------

##### Returns

void

#### 4.32.3.5 setY()

```
template<typename T >
template void Vector2< T >::setY (
    T newY )
```

[setY\(\)](#): Set y of Vector2Struct.

##### Template Parameters

<i>T</i>	Type of the <a href="#">Vector2</a>
----------	-------------------------------------

## Parameters

<i>newY</i>	The new value of y.
-------------	---------------------

## Returns

void

The documentation for this class was generated from the following files:

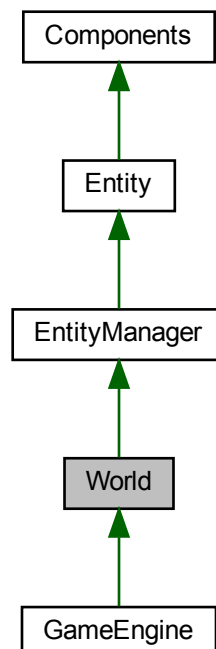
- src/Other/include/Vector2.h
- src/Other/Vector2.cpp

## 4.33 World Class Reference

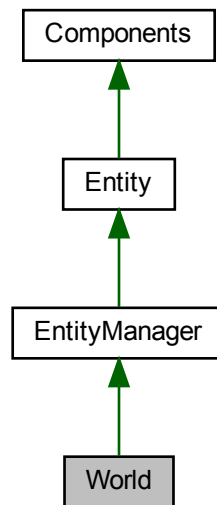
[World](#) class: [World](#) is a class that represents the world of the game.

```
#include <world.h>
```

Inheritance diagram for World:



Collaboration diagram for World:



## Public Member Functions

- [World](#) ()=default  
*Default [World](#) constructor.*
- [~World](#) () override=default  
*[World](#) destructor.*
- bool [init](#) () override  
*[init\(\)](#): Initialize the world.*
- void [createEntities](#) (std::map< std::string, std::pair< std::unique\_ptr< [EntityManager](#) >, std::vector< std::string >>> &mapEntityManager)  
*[createEntities\(\)](#): Create the entities.*
- [EntityManager](#) & [addEntityManager](#) (const std::string &NameEntityManager)  
*[addEntityManager\(\)](#): Add an entity manager to the map.*
- [EntityManager](#) & [getEntityManager](#) (const std::string &NameEntityManager)  
*[getEntityManager\(\)](#): Get the entity manager.*
- void [setNameWorld](#) (std::string newName)  
*[setNameWorld\(\)](#): Set the name of the world.*
- std::string [getNameWorld](#) () const  
*[getNameWorld\(\)](#): Get the name of the world.*
- std::map< std::string, [EntityManager](#) \* > [getEntityManagerMap](#) () const  
*[getEntityManagerMap\(\)](#): Get the map of the entity manager.*
- std::map< std::string, [EntityManager](#) \* > [getEntitiesManager](#) () const  
*[getEntitiesManager\(\)](#): Get the entities*



## Additional Inherited Members

### 4.33.1 Detailed Description

[World](#) class: [World](#) is a class that represents the world of the game.

The [World](#) class manages the world of the game.

### 4.33.2 Constructor & Destructor Documentation

#### 4.33.2.1 World()

```
World::World ( ) [default]
```

Default [World](#) constructor.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

void

#### 4.33.2.2 ~World()

```
World::~World ( ) [override], [default]
```

[World](#) destructor.

##### Parameters

<i>void</i>	
-------------	--

##### Returns

void

### 4.33.3 Member Function Documentation

#### 4.33.3.1 addEntityManager()

```
EntityManager & World::addEntityManager (
    const std::string & NameEntityManager )
```

[addEntityManager\(\)](#): Add an entity manager to the map.

##### Parameters

<i>NameEntityManager</i>	Name of the entity manager.
--------------------------	-----------------------------

##### Returns

[EntityManager&](#): The entity manager.

#### 4.33.3.2 createEntities()

```
void World::createEntities (
    std::map< std::string, std::pair< std::unique_ptr< EntityManager >, std::vector<
std::string >>> & mapEntityManager )
```

[createEntities\(\)](#): Create the entities.

##### Parameters

<i>mapEntityManager</i>	Map of the entities manager's unique pointers.
<i>keyEntityManager</i>	Key of the entities manager.

##### Returns

void

#### 4.33.3.3 getEntitiesManager()

```
std::map< std::string, EntityManager * > World::getEntitiesManager ( ) const
```

[getEntitiesManager\(\)](#): Get the entities

##### Parameters

<i>void</i>	
-------------	--

**Returns**

`std::map<std::string, EntityManager*>`: Get the entities.

**4.33.3.4 getEntityManager()**

```
EntityManager & World::getEntityManager (
    const std::string & NameEntityManager )
```

`getEntityManager()`: Get the entity manager.

**Parameters**

<code>NameEntityManager</code>	Name of the entity manager.
--------------------------------	-----------------------------

**Returns**

`EntityManager&`: The entity manager.

**4.33.3.5 getEntityManagerMap()**

```
std::map< std::string, EntityManager * > World::getEntityManagerMap ( ) const
```

`getEntityManagerMap()`: Get the map of the entity manager.

**Parameters**

<code>void</code>	
-------------------	--

**Returns**

`std::map<std::string, EntityManager*>`: The map of the entity manager.

**4.33.3.6 getNameWorld()**

```
std::string World::getNameWorld ( ) const
```

`getNameWorld()`: Get the name of the world.

**Parameters**

<code>void</code>	
-------------------	--

**Returns**

std::string: The name of the world.

**4.33.3.7 init()**

```
bool World::init ( ) [override], [virtual]
```

[init\(\)](#): Initialize the world.

**Parameters**

<i>void</i>	
-------------	--

**Returns**

bool: True if the world is initialized, false otherwise.

Reimplemented from [EntityManager](#).

**4.33.3.8 setNameWorld()**

```
void World::setNameWorld (
    std::string newName )
```

[setNameWorld\(\)](#): Set the name of the world.

**Parameters**

<i>newName</i>	New name of the world.
----------------	------------------------

**Returns**

void

The documentation for this class was generated from the following files:

- src/World/include/world.h
- src/World/world.cpp

# Index

- ~Color
  - Color, [9](#)
- ~Components
  - Components, [15](#)
- ~DrawableComponent
  - DrawableComponent, [17](#)
- ~Entity
  - Entity, [21](#)
- ~EntityManager
  - EntityManager, [31](#)
- ~EventEngine
  - EventEngine, [37](#)
- ~GameEngine
  - GameEngine, [46](#)
- ~ITransform
  - ITransform, [60](#)
- ~Music
  - Music, [63](#)
- ~Rect
  - Rect< T >, [71](#)
- ~Sound
  - Sound, [77](#)
- ~Sprite
  - Sprite, [86](#)
- ~Text
  - Text, [99](#)
- ~Transform
  - Transform, [113](#)
- ~Vector2
  - Vector2< T >, [122](#)
- ~World
  - World, [127](#)
- ~toSFML
  - toSFML, [110](#)
- addComponent
  - Entity, [22](#)
- addDrawable
  - Entity, [22](#)
- addEntity
  - EntityManager, [31](#)
- addEntityManager
  - World, [127](#)
- addKeyPressed
  - EventEngine, [37](#)
- addMouseButtonPressed
  - EventEngine, [38](#)
- addMouseMove
  - EventEngine, [38](#)
- addWorld
  - GameEngine, [46](#)
- applyDeferredEntity
  - Entity, [22](#)
- applyDeferredMusic
  - Music, [63](#)
- applyDeferredSound
  - Sound, [77](#)
- applyDeferredSprite
  - Sprite, [86](#)
- applyDeferredText
  - Text, [99](#)
- applyDeferredTransform
  - Transform, [113](#)
- Archetypes, [7](#)
- Color, [7](#)
  - ~Color, [9](#)
  - Color, [9](#)
  - fromSFMLColor, [10](#)
  - getAlpha, [10](#)
  - getBlue, [10](#)
  - getGreen, [11](#)
  - getRed, [11](#)
  - operator sf::Color, [11](#)
  - setAlpha, [12](#)
  - setBlue, [12](#)
  - setGreen, [12](#)
  - setRed, [13](#)
- Components, [13](#)
  - ~Components, [15](#)
  - Components, [14](#)
  - getBit, [15](#)
  - init, [16](#)
  - update, [16](#)
- contains
  - Rect< T >, [72](#)
- createEntities
  - World, [128](#)
- draw
  - DrawableComponent, [18](#)
  - Sprite, [86](#)
  - Text, [100](#)
- DrawableComponent, [16](#)
  - ~DrawableComponent, [17](#)
  - draw, [18](#)
- drawEntity
  - Entity, [23](#)
- Entity, [18](#)

- ~Entity, 21
- addComponent, 22
- addDrawable, 22
- applyDeferredEntity, 22
- drawEntity, 23
- Entity, 21
- getActive, 23
- getBit, 23
- getComponent, 24
- getComponentArrays, 24
- getComponentBitset, 25
- getComponentTypeID, 25
- getDrawableComponents, 25
- getName, 26
- init, 26
- removeComponent, 26
- removeDrawable, 27
- setActive, 27
- setDeferredEntity, 28
- setName, 28
- update, 28
- EntityManager, 29
  - ~EntityManager, 31
  - addEntity, 31
  - EntityManager, 30
  - getEntities, 31
  - getEntity, 32
  - getEntityMap, 32
  - init, 33
- EntityManagerTest, 33
- EntityTest, 35
- EventEngine, 36
  - ~EventEngine, 37
  - addKeyPressed, 37
  - addMouseButtonPressed, 38
  - addMouseMove, 38
  - EventEngine, 37
  - getEvent, 38
  - getKeyPressedMap, 39
  - getKeyStatesMap, 39
  - getMouseButtonPressedMap, 39
  - getMouseMoveMap, 40
  - setKeyStatesMap, 40
- eventGameEngine
  - GameEngine, 47
- EventTest, 41
- fromSFMLColor
  - Color, 10
- GameEngine, 42
  - ~GameEngine, 46
  - addWorld, 46
  - eventGameEngine, 47
  - GameEngine, 45
  - getClock, 47
  - getCurrentWorld, 47
  - getDeltaTime, 48
  - getEventEngine, 48
  - getFilesResources, 48
  - getMapFont, 49
  - getMapMusic, 49
  - getMapSound, 49
  - getMapTexture, 50
  - getWindow, 50
  - getWorld, 50
  - getWorldMap, 51
  - initialize, 51
  - initializeAllFiles, 51
  - initializeFont, 52
  - initializeMusic, 52
  - initializeMusicFunction, 52
  - initializeSound, 53
  - initializeSoundFunction, 53
  - initializeSpriteFunction, 53
  - initializeTextFunction, 54
  - initializeTexture, 54
  - initializeWorldMap, 54
  - isWindowOpen, 55
  - renderGameEngine, 55
  - run, 55
  - setCurrentWorld, 57
  - setDeltaTime, 57
  - updateGameEngine, 57
- GameEngineTest, 58
- getActive
  - Entity, 23
- getAlpha
  - Color, 10
- getBit
  - Components, 15
  - Entity, 23
  - Music, 64
  - Sound, 77
  - Sprite, 87
  - Text, 100
  - Transform, 113
- getBlue
  - Color, 10
- getClock
  - GameEngine, 47
- getColorFill
  - Text, 100
- getColorOutline
  - Text, 101
- getComponent
  - Entity, 24
- getComponentArrays
  - Entity, 24
- getComponentBitset
  - Entity, 25
- getComponentTypeID
  - Entity, 25
- getCurrentWorld
  - GameEngine, 47
- getDeltaTime
  - GameEngine, 48

- getDrawableComponents
  - Entity, [25](#)
- getEntities
  - EntityManager, [31](#)
- getEntitiesManager
  - World, [128](#)
- getEntity
  - EntityManager, [32](#)
- getEntityManager
  - World, [129](#)
- getEntityManagerMap
  - World, [129](#)
- getEntityMap
  - EntityManager, [32](#)
- getEvent
  - EventEngine, [38](#)
- getEventEngine
  - GameEngine, [48](#)
- getFilesRessources
  - GameEngine, [48](#)
- getFont
  - Text, [101](#)
- getGreen
  - Color, [11](#)
- getHeight
  - Rect< T >, [72](#)
- getKeyPressedMap
  - EventEngine, [39](#)
- getKeyStatesMap
  - EventEngine, [39](#)
- getLeft
  - Rect< T >, [73](#)
- getLoop
  - Music, [64](#)
  - Sound, [78](#)
- getMapFont
  - GameEngine, [49](#)
- getMapMusic
  - GameEngine, [49](#)
- getMapSound
  - GameEngine, [49](#)
- getMapTexture
  - GameEngine, [50](#)
- getMouseButtonPressedMap
  - EventEngine, [39](#)
- getMouseMoveMap
  - EventEngine, [40](#)
- getMusic
  - Music, [64](#)
- getName
  - Entity, [26](#)
- getNameWorld
  - World, [129](#)
- getPosition
  - Transform, [115](#)
- getRect
  - Rect< T >, [73](#)
- getRed
  - Color, [11](#)
- getRotation
  - Transform, [115](#)
- getScale
  - Transform, [115](#)
- getSize
  - Text, [101](#)
- getSound
  - Sound, [78](#)
- getSprite
  - Sprite, [87](#)
- getStatus
  - Music, [65](#)
- getStringText
  - Text, [102](#)
- getText
  - Text, [102](#)
- getTop
  - Rect< T >, [73](#)
- getTransform
  - ITransform, [61](#)
  - Sprite, [87](#)
  - Text, [102](#)
  - Transform, [116](#)
- getVector2Struct
  - Vector2< T >, [123](#)
- getVolume
  - Music, [65](#)
  - Sound, [78](#)
- getWidth
  - Rect< T >, [74](#)
- getWindow
  - GameEngine, [50](#)
- getWorld
  - GameEngine, [50](#)
- getWorldMap
  - GameEngine, [51](#)
- getX
  - Vector2< T >, [123](#)
- getY
  - Vector2< T >, [123](#)
- init
  - Components, [16](#)
  - Entity, [26](#)
  - EntityManager, [33](#)
  - Music, [65](#)
  - Sound, [79](#)
  - Sprite, [88](#)
  - Text, [103](#)
  - Transform, [116](#)
  - World, [130](#)
- initialize
  - GameEngine, [51](#)
- initializeAllFiles
  - GameEngine, [51](#)
- initializeFont
  - GameEngine, [52](#)
- initializeMusic

- GameEngine, 52
- initializeMusicFunction
  - GameEngine, 52
- initializeSound
  - GameEngine, 53
- initializeSoundFunction
  - GameEngine, 53
- initializeSpriteFunction
  - GameEngine, 53
- initializeTextFunction
  - GameEngine, 54
- initializeTexture
  - GameEngine, 54
- initializeWorldMap
  - GameEngine, 54
- isPlaying
  - Sound, 79
- isWindowOpen
  - GameEngine, 55
- ITransform, 60
  - ~ITransform, 60
  - getTransform, 61
- Music, 61
  - ~Music, 63
  - applyDeferredMusic, 63
  - getBit, 64
  - getLoop, 64
  - getMusic, 64
  - getStatus, 65
  - getVolume, 65
  - init, 65
  - Music, 63
  - pause, 66
  - play, 66
  - setDeferredMusic, 66
  - setLoop, 67
  - setMusic, 67
  - setVolume, 68
  - stop, 68
  - update, 68
- MusicTests, 69
- operator sf::Color
  - Color, 11
- pause
  - Music, 66
  - Sound, 79
- play
  - Music, 66
  - Sound, 80
- Rect
  - Rect< T >, 71
- Rect< T >, 70
  - ~Rect, 71
  - contains, 72
  - getHeight, 72
  - getLeft, 73
  - getRect, 73
  - getTop, 73
  - getWidth, 74
  - Rect, 71
- removeComponent
  - Entity, 26
- removeDrawable
  - Entity, 27
- renderGameEngine
  - GameEngine, 55
- run
  - GameEngine, 55
- Script, 74
- setActive
  - Entity, 27
- setAlpha
  - Color, 12
- setBlue
  - Color, 12
- setCurrentWorld
  - GameEngine, 57
- setDeferredEntity
  - Entity, 28
- setDeferredMusic
  - Music, 66
- setDeferredSound
  - Sound, 80
- setDeferredSprite
  - Sprite, 88
- setDeferredText
  - Text, 103
- setDeferredTransform
  - Transform, 116
- setDeltaTime
  - GameEngine, 57
- setFillColor
  - Text, 104
- setFont
  - Text, 104
- setGreen
  - Color, 12
- setKeyStatesMap
  - EventEngine, 40
- setLoop
  - Music, 67
  - Sound, 80
- setMusic
  - Music, 67
- setName
  - Entity, 28
- setNameWorld
  - World, 130
- setOutlineColor
  - Text, 104
- setPosition
  - Transform, 117
- setRed



- Color, 13
- setRotation
  - Transform, 117
- setScale
  - Transform, 117
- setSize
  - Text, 105
- setSound
  - Sound, 81
- setSprite
  - Sprite, 89
- setString
  - Text, 105
- setText
  - Text, 105, 106
- setTransform
  - Sprite, 89
  - Text, 106
  - Transform, 119
- setVolume
  - Music, 68
  - Sound, 81
- setX
  - Vector2< T >, 124
- setY
  - Vector2< T >, 124
- Sound, 75
  - ~Sound, 77
  - applyDeferredSound, 77
  - getBit, 77
  - getLoop, 78
  - getSound, 78
  - getVolume, 78
  - init, 79
  - isPlaying, 79
  - pause, 79
  - play, 80
  - setDeferredSound, 80
  - setLoop, 80
  - setSound, 81
  - setVolume, 81
  - Sound, 76
  - stop, 82
  - update, 82
- SoundTest, 83
- Sprite, 84
  - ~Sprite, 86
  - applyDeferredSprite, 86
  - draw, 86
  - getBit, 87
  - getSprite, 87
  - getTransform, 87
  - init, 88
  - setDeferredSprite, 88
  - setSprite, 89
  - setTransform, 89
  - Sprite, 85
  - update, 90
- SpriteTest, 91
- stop
  - Music, 68
  - Sound, 82
- TestColor, 92
- TestRect, 93
- TesttoSFML, 94
- TestVector2, 95
- TestWorld, 96
- Text, 97
  - ~Text, 99
  - applyDeferredText, 99
  - draw, 100
  - getBit, 100
  - getColorFill, 100
  - getColorOutline, 101
  - getFont, 101
  - getSize, 101
  - getStringText, 102
  - getText, 102
  - getTransform, 102
  - init, 103
  - setDeferredText, 103
  - setFillColor, 104
  - setFont, 104
  - setOutlineColor, 104
  - setSize, 105
  - setString, 105
  - setText, 105, 106
  - setTransform, 106
  - Text, 99
  - update, 107
- TextTest, 108
- toSFML, 109
  - ~toSFML, 110
  - toSFML, 109
  - toSFMLRect, 110
- toSFMLRect
  - toSFML, 110
- Transform, 111
  - ~Transform, 113
  - applyDeferredTransform, 113
  - getBit, 113
  - getPosition, 115
  - getRotation, 115
  - getScale, 115
  - getTransform, 116
  - init, 116
  - setDeferredTransform, 116
  - setPosition, 117
  - setRotation, 117
  - setScale, 117
  - setTransform, 119
  - Transform, 112
  - update, 119
- TransformTest, 120
- update

- Components, [16](#)
- Entity, [28](#)
- Music, [68](#)
- Sound, [82](#)
- Sprite, [90](#)
- Text, [107](#)
- Transform, [119](#)
- updateGameEngine
  - GameEngine, [57](#)
- Vector2
  - Vector2< T >, [121](#), [122](#)
- Vector2< T >, [121](#)
  - ~Vector2, [122](#)
  - getVector2Struct, [123](#)
  - getX, [123](#)
  - getY, [123](#)
  - setX, [124](#)
  - setY, [124](#)
  - Vector2, [121](#), [122](#)
- World, [125](#)
  - ~World, [127](#)
  - addEntityManager, [127](#)
  - createEntities, [128](#)
  - getEntitiesManager, [128](#)
  - getEntityManager, [129](#)
  - getEntityManagerMap, [129](#)
  - getNameWorld, [129](#)
  - init, [130](#)
  - setNameWorld, [130](#)
  - World, [127](#)