

## R-Type - Engine

Generated by Doxygen 1.9.1



<b>1 Engine</b>	<b>1</b>
<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 Audio Class Reference	7
4.3 Components Class Reference	7
4.4 DrawableComponent Class Reference	8
4.5 Entity Class Reference	8
4.5.1 Detailed Description	9
4.5.2 Constructor & Destructor Documentation	9
4.5.2.1 Entity() [1/2]	9
4.5.2.2 Entity() [2/2]	9
4.5.2.3 ~Entity()	10
4.5.3 Member Function Documentation	10
4.5.3.1 addComponent()	10
4.5.3.2 getComponent()	10
4.5.3.3 getName()	11
4.5.3.4 init()	11
4.5.3.5 setName()	12
4.6 EntityManager Class Reference	12
4.6.1 Constructor & Destructor Documentation	12
4.6.1.1 EntityManager()	12
4.6.1.2 ~EntityManager()	13
4.6.2 Member Function Documentation	13
4.6.2.1 addEntity()	13
4.6.2.2 getEntities()	14
4.6.2.3 getEntity()	14
4.6.2.4 getEntityMap()	14
4.6.2.5 init()	15
4.7 EntityManagerTest Class Reference	15
4.8 EntityTest Class Reference	16
4.9 EventEngine Class Reference	16
4.9.1 Detailed Description	16
4.9.2 Constructor & Destructor Documentation	16
4.9.2.1 EventEngine()	16
4.9.2.2 ~EventEngine()	17
4.9.3 Member Function Documentation	17

4.9.3.1 addKeyPressed()	17
4.9.3.2 getEvent()	18
4.9.3.3 getKeyPressedMap()	18
4.9.3.4 init()	18
4.10 EventTest Class Reference	19
4.11 GameEngine Class Reference	19
4.12 Sprite Class Reference	20
4.12.1 Detailed Description	21
4.12.2 Constructor & Destructor Documentation	21
4.12.2.1 Sprite() [1/2]	21
4.12.2.2 Sprite() [2/2]	21
4.12.2.3 ~Sprite()	22
4.12.3 Member Function Documentation	22
4.12.3.1 applyDeferredSprite()	22
4.12.3.2 createSprite() [1/3]	22
4.12.3.3 createSprite() [2/3]	23
4.12.3.4 createSprite() [3/3]	23
4.12.3.5 draw()	23
4.12.3.6 getBit()	24
4.12.3.7 getSprite()	24
4.12.3.8 getTexture()	24
4.12.3.9 init()	25
4.12.3.10 isTextureLoaded()	25
4.12.3.11 setDeferredSprite()	25
4.12.3.12 setSprite() [1/2]	26
4.12.3.13 setSprite() [2/2]	26
4.12.3.14 setTexture()	27
4.13 SpriteTest Class Reference	27
4.14 Transform Class Reference	27
4.14.1 Detailed Description	28
4.14.2 Constructor & Destructor Documentation	28
4.14.2.1 Transform() [1/2]	28
4.14.2.2 Transform() [2/2]	28
4.14.2.3 ~Transform()	29
4.14.3 Member Function Documentation	29
4.14.3.1 getBit()	29
4.14.3.2 getPositionVector()	29
4.14.3.3 getRotationVector()	30
4.14.3.4 getScaleVector()	30
4.14.3.5 setTransform()	30
4.15 TransformTest Class Reference	31
4.16 World Class Reference	31

---

4.16.1 Detailed Description . . . . .	32
4.16.2 Constructor & Destructor Documentation . . . . .	32
4.16.2.1 World() . . . . .	32
4.16.2.2 ~World() . . . . .	32
4.16.3 Member Function Documentation . . . . .	32
4.16.3.1 addEntityManager() . . . . .	33
4.16.3.2 createEntities() . . . . .	33
4.16.3.3 getEntityManager() . . . . .	33
4.16.3.4 getEntityManagerMap() . . . . .	34
4.16.3.5 getNameWorld() . . . . .	34
4.16.3.6 init() . . . . .	34
4.16.3.7 setNameWorld() . . . . .	35
<b>Index</b>	<b>37</b>



## Chapter 1

# Engine





## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

Archetypes . . . . .	7
Audio . . . . .	7
Components . . . . .	7
Entity . . . . .	8
EntityManager . . . . .	12
World . . . . .	31
GameEngine . . . . .	19
Sprite . . . . .	20
Transform . . . . .	27
DrawableComponent . . . . .	8
Sprite . . . . .	20
EventEngine . . . . .	16
GameEngine . . . . .	19
testing::Test	
EntityManagerTest . . . . .	15
EntityTest . . . . .	16
EventTest . . . . .	19
SpriteTest . . . . .	27
TransformTest . . . . .	31



## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">Archetypes</a>	7
<a href="#">Audio</a>	7
<a href="#">Components</a>	7
<a href="#">DrawableComponent</a>	8
<a href="#">Entity</a>	
<a href="#">Entity</a> class: <a href="#">Entity</a> is a class that represents an entity in the game	8
<a href="#">EntityManager</a>	12
<a href="#">EntityManagerTest</a>	15
<a href="#">EntityTest</a>	16
<a href="#">EventEngine</a>	
<a href="#">EventEngine</a> class: <a href="#">EventEngine</a> is a class that represents the event engine of the game	16
<a href="#">EventTest</a>	19
<a href="#">GameEngine</a>	19
<a href="#">Sprite</a>	
<a href="#">Sprite</a> class: <a href="#">Sprite</a> is a class that represents the rendering properties of a Component	20
<a href="#">SpriteTest</a>	27
<a href="#">Transform</a>	
<a href="#">Transform</a> class: <a href="#">Transform</a> is a class that represents the transform of a Component	27
<a href="#">TransformTest</a>	31
<a href="#">World</a>	
<a href="#">World</a> class: <a href="#">World</a> is a class that represents the world of the game	31



## Chapter 4

# Class Documentation

### 4.1 Archetypes Class Reference

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

- `src/Archetype/Archetypes.h`

### 4.2 Audio Class Reference

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

- `src/Components/all_components/Audio.h`

### 4.3 Components Class Reference

Inheritance diagram for Components:

#### Public Member Functions

- virtual bool **init** ()
- virtual void **update** ()
- template<typename T >  
ComponentTypeID **GetComponentTypeID** () noexcept

#### Protected Types

- using **ComponentTypeID** = std::size\_t

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

- `src/Components/Components.h`
- `src/Components/Components.cpp`

## 4.4 DrawableComponent Class Reference

Inheritance diagram for DrawableComponent:

### Public Member Functions

- virtual void **draw** (sf::RenderWindow &window) const =0

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

- src/Components/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  
*Default [Entity](#) constructor.*
- [Entity](#) (std::string nameEntity, [Archetypes](#) newArchetype=[Archetypes](#)())  
*[Entity](#) constructor.*
- [~Entity](#) () override=default  
*[Entity](#) destructor.*
- bool [init](#) () override  
*[init\(\)](#): Initialize the entity*
- std::string [getName](#) () const  
*[genName\(\)](#): Get the name of the entity*
- void [setName](#) (std::string newName)  
*[setName\(\)](#): Set the name of the entity*
- void **addDrawable** ([Components](#) \*component)
- void **draw** (sf::RenderWindow &window)
- template<typename T , typename... TArgs>  
T & [addComponent](#) (TArgs &&... args)  
*[addComponent\(\)](#): Add a component to the entity*
- template<typename T >  
T & [getComponent](#) ()  
*[getComponent\(\)](#): Get a component from the entity*
- std::bitset< 3 > **getComponentBitset** () const
- std::vector< [DrawableComponent](#) \* > **getDrawableComponents** () const
- std::array< [Components](#) \*, 3 > **getComponentArrays** () const

## 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 ( ) [default]
```

Default [Entity](#) constructor.

##### Parameters

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

##### Returns

void

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

```
Entity::Entity (
    std::string nameEntity,
    Archetypes newArchetype = Archetypes() ) [inline], [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>
T & Entity::addComponent (
    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&: reference of the component

#### 4.5.3.2 getComponent()

```
template<typename T >
T & Entity::getComponent
```

[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

### 4.5.3.3 getName()

```
std::string Entity::getName ( ) const [inline]
```

getName(): Get the name of the entity

## Parameters

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

## Returns

std::string: name of the entity

### 4.5.3.4 init()

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

init(): Initialize the entity

## Parameters

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

## Returns

bool: true if the entity is initialized, false otherwise

Reimplemented from [Components](#).

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

#### 4.5.3.5 setName()

```
void Entity::setName (
    std::string newName ) [inline]
```

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

##### Parameters

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

##### Returns

void

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

- src/Entity/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](#) ()=default  
*[EntityManager](#) destructor.*
- [Entity](#) & [addEntity](#) (std::string nameEntity, [Archetypes](#) newArchetype=[Archetypes](#)())  
*[addEntity\(\)](#): Create and add a new entity to the entity manager.*
- [Entity](#) & [getEntity](#) (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.*
- bool [init](#) () override  
*[init\(\)](#): Initialize the entity*

### 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 ( ) [default]`

[EntityManager](#) destructor.

## Parameters

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

## Returns

void

## 4.6.2 Member Function Documentation

### 4.6.2.1 addEntity()

```
Entity & EntityManager::addEntity (
    std::string nameEntity,
    Archetypes newArchetype = Archetypes() ) [inline]
```

[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 [inline]
```

[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 (
    std::string nameEntity ) [inline]
```

[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 [inline]
```

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

##### Parameters

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

#### Returns

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

#### 4.6.2.5 init()

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

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

#### Parameters

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

#### Returns

bool: true if the entity is initialized, false otherwise

Reimplemented from [Entity](#).

Reimplemented in [World](#).

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

- src/Entity/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`

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 <event.h>
```

Inheritance diagram for EventEngine:

### Public Member Functions

- [EventEngine](#) ()=default  
*Default [EventEngine](#) constructor.*
- virtual [~EventEngine](#) ()=default  
*[EventEngine](#) destructor.*
- bool [init](#) () const  
*[init\(\)](#): Initialize the [EventEngine](#).*
- sf::Event & [getEvent](#) ()  
*[getEvent\(\)](#): Get the SFML Event.*
- void [addKeyPressed](#) (sf::Keyboard::Key keyboard, std::function< void()> function)  
*[addKeyPressed\(\)](#): Add a key pressed to the map.*
- std::map< sf::Keyboard::Key, std::function< void()> > & [getKeyPressedMap](#) ()  
*[getKeyPressedMap\(\)](#): Get the map of the key pressed.*

### 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,
    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 `getEvent()`

```
sf::Event& EventEngine::getEvent ( ) [inline]
```

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

##### Parameters

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

##### Returns

sf::Event: The SFML Event.

#### 4.9.3.3 `getKeyPressedMap()`

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

[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.

#### 4.9.3.4 `init()`

```
bool EventEngine::init ( ) const [inline]
```

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

##### Parameters

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

##### Returns

bool: True if the [EventEngine](#) is initialized, false otherwise.

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

- src/Event/event.h
- src/Event/event.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

Inheritance diagram for GameEngine:

Collaboration diagram for GameEngine:

### Public Member Functions

- **GameEngine** (sf::VideoMode mode, std::string type, sf::String title, sf::Uint32 style=sf::Style::Default, const sf::ContextSettings &settings=sf::ContextSettings())
- void **run** (std::map< std::string, std::unique\_ptr< [World](#) >> mapWorld, std::map< std::string, std::string > pathResources, std::string firstScene)
- void **run** ()
- void **renderGameEngine** ()
- void **eventGameEngine** ()
- bool **isWindowOpen** ()
- void **updateGameEngine** ()
- void **initialize** (std::map< std::string, std::unique\_ptr< [World](#) >> mapWorld, std::map< std::string, std::string > pathResources, std::string firstScene)
- void **initializeSprite** ()
- void **initializeTexture** (std::string path)
- void **initializeWorldMap** (std::map< std::string, std::unique\_ptr< [World](#) >> mapWorld)
- const auto & **getWindow** ()
- void **setWindow** ()
- [EventEngine](#) & **getEventEngine** ()
- void **setCurrentWorld** ([World](#) \*world)
- [World](#) \* **getCurrentWorld** ()
- [World](#) & **addWorld** (std::string nameWorld, std::unique\_ptr< [World](#) > world)
- [World](#) & **getWorld** (std::string nameWorld)
- std::map< std::string, sf::Texture > **getMapTexture** () const
- std::map< std::string, [World](#) \* > **getWorldMap** () const

## Additional Inherited Members

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

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

## 4.12 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** ()=default  
*Default **Sprite** constructor.*
- **Sprite** (const std::string &texturePath)  
***Sprite** constructor with an existing texture path.*
- ~**Sprite** () override=default  
***Sprite** destructor.*
- bool **init** () const  
*init(): Initialize the **Sprite**.*
- int **getBit** () const  
*getBit(): Get the bit of the **Sprite**.*
- void **draw** (sf::RenderWindow &window) const override  
*draw(): Draw the **Sprite**.*
- void **createSprite** (const std::string &texturePath)  
*createSprite(): Create the SFML **Sprite** with a texture path for rendering.*
- void **createSprite** (const sf::Texture &existingTexture)  
*createSprite(): Create the SFML **Sprite** with an existing texture for rendering.*
- void **createSprite** ()  
*createSprite(): Create the SFML **Sprite** with the component's texture for rendering.*
- sf::Sprite **getSprite** () const  
*getSprite(): Get the SFML **Sprite** for rendering.*
- sf::Texture **getTexture** () const  
*getTexture(): Get the SFML Texture for the sprite.*
- bool **isTextureLoaded** () const  
*isTextureLoaded(): Check if the texture is loaded.*
- void **setSprite** (const sf::Sprite &sprite)  
*setSprite(): Set the SFML **Sprite** with an existing one for rendering.*
- void **setSprite** (std::map< std::string, sf::Texture > mapTexture, std::string nameTexture, std::map< std::string, std::vector< float >> &mapTransform)  
*setSprite(): Set the SFML **Sprite** with a map of string and textures, a texture name and a map of string and vector of floats.*
- void **setDeferredSprite** (std::function< void()> setter)  
*setDeferredSprite(): Set the deferred sprite.*
- void **applyDeferredSprite** ()  
*applyDeferredSprite(): Apply the deferred sprite.*
- void **setTexture** (const sf::Texture &existingTexture)  
*setTexture(): Set the texture with an existing one for the sprite.*

## Additional Inherited Members

### 4.12.1 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.

### 4.12.2 Constructor & Destructor Documentation

#### 4.12.2.1 [Sprite\(\)](#) [1/2]

```
Sprite::Sprite ( ) [default]
```

Default [Sprite](#) constructor.

##### Parameters

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

##### Returns

void

#### 4.12.2.2 [Sprite\(\)](#) [2/2]

```
Sprite::Sprite (
    const std::string & texturePath ) [inline], [explicit]
```

[Sprite](#) constructor with an existing texture path.

##### Parameters

<i>texturePath</i>	Path to the texture file for the sprite.
--------------------	--

##### Returns

void

#### 4.12.2.3 ~Sprite()

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

[Sprite](#) destructor.

##### Parameters

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

##### Returns

void

### 4.12.3 Member Function Documentation

#### 4.12.3.1 applyDeferredSprite()

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

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

##### Parameters

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

##### Returns

void

#### 4.12.3.2 createSprite() [1/3]

```
void Sprite::createSprite ( )
```

[createSprite\(\)](#): Create the SFML [Sprite](#) with the component's texture for rendering.

##### Parameters

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

##### Returns

void

#### 4.12.3.3 createSprite() [2/3]

```
void Sprite::createSprite (
    const sf::Texture & existingTexture )
```

**createSprite()**: Create the SFML [Sprite](#) with an existing texture for rendering.

##### Parameters

<i>existingTexture</i>	SFML Texture for the sprite
------------------------	-----------------------------

##### Returns

void

#### 4.12.3.4 createSprite() [3/3]

```
void Sprite::createSprite (
    const std::string & texturePath )
```

**createSprite()**: Create the SFML [Sprite](#) with a texture path for rendering.

##### Parameters

<i>texturePath</i>	Path to the texture file for the sprite.
--------------------	--

##### Returns

void

#### 4.12.3.5 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](#).

**4.12.3.6 getBit()**

```
int Sprite::getBit ( ) const [inline]
```

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

**Parameters**

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

**Returns**

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

**4.12.3.7 getSprite()**

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

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

**Parameters**

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

**Returns**

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

**4.12.3.8 getTexture()**

```
sf::Texture Sprite::getTexture ( ) const
```

[getTexture\(\)](#): Get the SFML Texture for the sprite.

## Parameters

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

## Returns

sf::Texture: SFML Texture for the sprite

**4.12.3.9 init()**

```
bool Sprite::init ( ) const [inline]
```

init(): Initialize the [Sprite](#).

## Parameters

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

## Returns

bool: True if the [Sprite](#) is initialized, false otherwise.

**4.12.3.10 isTextureLoaded()**

```
bool Sprite::isTextureLoaded ( ) const [inline]
```

[isTextureLoaded\(\)](#): Check if the texture is loaded.

## Parameters

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

## Returns

bool: True if the texture is loaded, false otherwise.

**4.12.3.11 setDeferredSprite()**

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

[setDeferredSprite\(\)](#): Set the deferred sprite.

## Parameters

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

## Returns

void

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

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

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

## Parameters

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

## Returns

void

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

```
void Sprite::setSprite (
    std::map< std::string, sf::Texture > mapTexture,
    std::string nameTexture,
    std::map< std::string, std::vector< float >> & mapTransform )
```

**setSprite()**: Set the SFML [Sprite](#) with a map of string and textures, a texture name and a map of string and vector of floats.

## Parameters

<i>mapTexture</i>	Map of string and textures.
<i>nameTexture</i>	Name of the texture.
<i>mapTransform</i>	Map of string and vector of floats.

## Returns

void



4.12.3.14 `setTexture()`

```
void Sprite::setTexture (
    const sf::Texture & existingTexture )
```

`setTexture()`: Set the texture with an existing one for the sprite.

## Parameters

<i>existingTexture</i>	SFML Texture for the sprite
------------------------	-----------------------------

## Returns

void

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

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

## 4.13 SpriteTest Class Reference

Inheritance diagram for SpriteTest:

## 4.14 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  
*Default `Transform` constructor.*
- `bool init ()` const
- `Transform (const std::map< std::string, std::vector< float >> &mapTransform)`  
*`Transform` constructor.*
- `~Transform ()` override=default  
*`Transform` destructor.*
- `int getBit ()` const  
*`getBit()`: Get the bitmask of the component*
- `std::vector< float > getPositionVector ()` const  
*`getPositionVector()`: Get the position vector of the component;*
- `std::vector< float > getRotationVector ()` const  
*`getRotationVector()`: Get the rotation vector of the component;*
- `std::vector< float > getScaleVector ()` const  
*`getScaleVector()`: Get the scale vector of the component;*
- `void setTransform (const std::map< std::string, std::vector< float >> &mapTransform)`  
*`setTransform()`: Set the transformation properties of the component*

## Additional Inherited Members

### 4.14.1 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.

### 4.14.2 Constructor & Destructor Documentation

#### 4.14.2.1 Transform() [1/2]

```
Transform::Transform ( ) [default]
```

Default [Transform](#) constructor.

##### Parameters

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

##### Returns

void

#### 4.14.2.2 Transform() [2/2]

```
Transform::Transform (
    const std::map< std::string, std::vector< float >> & mapTransform ) [inline],
[explicit]
```

[Transform](#) constructor.

##### Parameters

<i>mapTransform</i>	Map containing transformation properties (std::string, std::vector<float>).
---------------------	---

##### Returns

void

#### 4.14.2.3 ~Transform()

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

[Transform](#) destructor.

##### Parameters

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

##### Returns

void

### 4.14.3 Member Function Documentation

#### 4.14.3.1 getBit()

```
int Transform::getBit ( ) const
```

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

##### Parameters

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

##### Returns

int: bitmask of the component

#### 4.14.3.2 getPositionVector()

```
std::vector< float > Transform::getPositionVector ( ) const
```

[getPositionVector\(\)](#): Get the position vector of the component;

##### Parameters

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

##### Returns

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

#### 4.14.3.3 getRotationVector()

```
std::vector< float > Transform::getRotationVector ( ) const
```

[getRotationVector\(\)](#): Get the rotation vector of the component;

##### Parameters

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

##### Returns

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

#### 4.14.3.4 getScaleVector()

```
std::vector< float > Transform::getScaleVector ( ) const
```

[getScaleVector\(\)](#): Get the scale vector of the component;

##### Parameters

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

##### Returns

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

#### 4.14.3.5 setTransform()

```
void Transform::setTransform (
    const std::map< std::string, std::vector< float >> & mapTransform )
```

[setTransform\(\)](#): Set the transformation properties of the component

##### Parameters

<i>mapTransform</i>	Map containing transformation properties (std::string, std::vector<float>).
---------------------	---

## Returns

void

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

- src/Components/all\_components/Transform.h
- src/Components/all\_components/Transform.cpp

## 4.15 TransformTest Class Reference

Inheritance diagram for TransformTest:

Collaboration diagram for TransformTest:

### Protected Attributes

- [Transform](#) transform

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

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

## 4.16 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  
    < Name of the world.
- [~World](#) () override=default  
    [World](#) destructor.
- void [createEntities](#) (std::map< std::string, std::pair< std::unique\_ptr< [EntityManager](#) >, std::vector< std::string >>> &mapEntityManager, std::string keyEntityManager)  
    [createEntities\(\)](#): Create the entities.
- [EntityManager](#) & [addEntityManager](#) (std::string NameEntityManager)  
    [addEntityManager\(\)](#): Add an entity manager to the map.
- [EntityManager](#) & [getEntityManager](#) (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.
- bool [init](#) () override  
    [init\(\)](#): Initialize the [World](#).

## Additional Inherited Members

### 4.16.1 Detailed Description

[World](#) class: [World](#) is a class that represents the world of the game.

The [World](#) class manages the world of the game.

### 4.16.2 Constructor & Destructor Documentation

#### 4.16.2.1 World()

```
World::World ( ) [default]
```

< Name of the world.

Default [World](#) constructor.

##### Parameters

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

##### Returns

void

#### 4.16.2.2 ~World()

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

[World](#) destructor.

##### Parameters

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

##### Returns

void

### 4.16.3 Member Function Documentation

#### 4.16.3.1 addEntityManager()

```
EntityManager & World::addEntityManager (
    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.

#### 4.16.3.2 createEntities()

```
void World::createEntities (
    std::map< std::string, std::pair< std::unique_ptr< EntityManager >, std::vector<
std::string >>> & mapEntityManager,
    std::string keyEntityManager )
```

**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

#### 4.16.3.3 getEntityManager()

```
EntityManager & World::getEntityManager (
    std::string NameEntityManager )
```

**getEntityManager()**: Get the entity manager.

##### Parameters

<i>NameEntityManager</i>	Name of the entity manager.
--------------------------	-----------------------------

**Returns**

[EntityManager](#)&: The entity manager.

**4.16.3.4 getEntityManagerMap()**

```
std::map<std::string, EntityManager\*> World::getEntityManagerMap ( ) const [inline]
```

[getEntityManagerMap\(\)](#): Get the map of the entity manager.

**Parameters**

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

**Returns**

std::map<std::string, [EntityManager\\*](#)>: The map of the entity manager.

**4.16.3.5 getNameWorld()**

```
std::string World::getNameWorld ( ) const [inline]
```

[getNameWorld\(\)](#): Get the name of the world.

**Parameters**

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

**Returns**

std::string: The name of the world.

**4.16.3.6 init()**

```
bool World::init ( ) [inline], [override], [virtual]
```

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

**Parameters**

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



**Returns**

bool: True if the world is initialized, false otherwise.

Reimplemented from [EntityManager](#).

**4.16.3.7 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/world.h
- src/World/world.cpp



# Index

- ~Entity
  - Entity, 9
- ~EntityManager
  - EntityManager, 13
- ~EventEngine
  - EventEngine, 17
- ~Sprite
  - Sprite, 21
- ~Transform
  - Transform, 28
- ~World
  - World, 32

- addComponent
  - Entity, 10
- addEntity
  - EntityManager, 13
- addEntityManager
  - World, 32
- addKeyPressed
  - EventEngine, 17
- applyDeferredSprite
  - Sprite, 22
- Archetypes, 7
- Audio, 7

- Components, 7
- createEntities
  - World, 33
- createSprite
  - Sprite, 22, 23

- draw
  - Sprite, 23
- DrawableComponent, 8

- Entity, 8
  - ~Entity, 9
  - addComponent, 10
  - Entity, 9
  - getComponent, 10
  - getName, 11
  - init, 11
  - setName, 11
- EntityManager, 12
  - ~EntityManager, 13
  - addEntity, 13
  - EntityManager, 12
  - getEntities, 13
  - getEntity, 14

- getEntityMap, 14
- init, 15
- EntityManagerTest, 15
- EntityTest, 16
- EventEngine, 16
  - ~EventEngine, 17
  - addKeyPressed, 17
  - EventEngine, 16
  - getEvent, 17
  - getKeyPressedMap, 18
  - init, 18
- EventTest, 19

- GameEngine, 19
- getBit
  - Sprite, 24
  - Transform, 29
- getComponent
  - Entity, 10
- getEntities
  - EntityManager, 13
- getEntity
  - EntityManager, 14
- getEntityManager
  - World, 33
- getEntityManagerMap
  - World, 34
- getEntityMap
  - EntityManager, 14
- getEvent
  - EventEngine, 17
- getKeyPressedMap
  - EventEngine, 18
- getName
  - Entity, 11
- getNameWorld
  - World, 34
- getPositionVector
  - Transform, 29
- getRotationVector
  - Transform, 30
- getScaleVector
  - Transform, 30
- getSprite
  - Sprite, 24
- getTexture
  - Sprite, 24
- init
  - Entity, 11

- EntityManager, [15](#)
- EventEngine, [18](#)
- Sprite, [25](#)
- World, [34](#)
- isTextureLoaded
  - Sprite, [25](#)
- setDeferredSprite
  - Sprite, [25](#)
- setName
  - Entity, [11](#)
- setNameWorld
  - World, [35](#)
- setSprite
  - Sprite, [26](#)
- setTexture
  - Sprite, [26](#)
- setTransform
  - Transform, [30](#)
- Sprite, [20](#)
  - ~Sprite, [21](#)
  - applyDeferredSprite, [22](#)
  - createSprite, [22](#), [23](#)
  - draw, [23](#)
  - getBit, [24](#)
  - getSprite, [24](#)
  - getTexture, [24](#)
  - init, [25](#)
  - isTextureLoaded, [25](#)
  - setDeferredSprite, [25](#)
  - setSprite, [26](#)
  - setTexture, [26](#)
  - Sprite, [21](#)
- SpriteTest, [27](#)
- Transform, [27](#)
  - ~Transform, [28](#)
  - getBit, [29](#)
  - getPositionVector, [29](#)
  - getRotationVector, [30](#)
  - getScaleVector, [30](#)
  - setTransform, [30](#)
  - Transform, [28](#)
- TransformTest, [31](#)
- World, [31](#)
  - ~World, [32](#)
  - addEntityManager, [32](#)
  - createEntities, [33](#)
  - getEntityManager, [33](#)
  - getEntityManagerMap, [34](#)
  - getNameWorld, [34](#)
  - init, [34](#)
  - setNameWorld, [35](#)
  - World, [32](#)