

## 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	7
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 initEntity()	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
4.7 EntityManagerTest Class Reference	15
4.8 EntityTest Class Reference	15
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 <a href="#">getEvent()</a>	17
4.9.3.3 <a href="#">getKeyPressedMap()</a>	18
4.9.3.4 <a href="#">init()</a>	18
4.10 <a href="#">EventTest Class Reference</a>	19
4.11 <a href="#">GameEngine Class Reference</a>	19
4.11.1 <a href="#">Detailed Description</a>	20
4.11.2 <a href="#">Constructor &amp; Destructor Documentation</a>	20
4.11.2.1 <a href="#">GameEngine() [1/2]</a>	20
4.11.2.2 <a href="#">GameEngine() [2/2]</a>	21
4.11.2.3 <a href="#">~GameEngine()</a>	21
4.11.3 <a href="#">Member Function Documentation</a>	22
4.11.3.1 <a href="#">addWorld()</a>	22
4.11.3.2 <a href="#">eventGameEngine()</a>	22
4.11.3.3 <a href="#">getCurrentWorld()</a>	22
4.11.3.4 <a href="#">getEventEngine()</a>	23
4.11.3.5 <a href="#">getMapTexture()</a>	23
4.11.3.6 <a href="#">getWindow()</a>	23
4.11.3.7 <a href="#">getWorld()</a>	24
4.11.3.8 <a href="#">getWorldMap()</a>	24
4.11.3.9 <a href="#">initialize()</a>	24
4.11.3.10 <a href="#">initializeSprite()</a>	25
4.11.3.11 <a href="#">initializeTexture()</a>	25
4.11.3.12 <a href="#">initializeWorldMap()</a>	26
4.11.3.13 <a href="#">isWindowOpen()</a>	26
4.11.3.14 <a href="#">renderGameEngine()</a>	26
4.11.3.15 <a href="#">run() [1/2]</a>	27
4.11.3.16 <a href="#">run() [2/2]</a>	27
4.11.3.17 <a href="#">setCurrentWorld()</a>	27
4.11.3.18 <a href="#">setWindow()</a>	28
4.11.3.19 <a href="#">updateGameEngine()</a>	28
4.12 <a href="#">Sprite Class Reference</a>	28
4.12.1 <a href="#">Detailed Description</a>	29
4.12.2 <a href="#">Constructor &amp; Destructor Documentation</a>	29
4.12.2.1 <a href="#">Sprite() [1/2]</a>	29
4.12.2.2 <a href="#">Sprite() [2/2]</a>	30
4.12.2.3 <a href="#">~Sprite()</a>	30
4.12.3 <a href="#">Member Function Documentation</a>	30
4.12.3.1 <a href="#">applyDeferredSprite()</a>	31
4.12.3.2 <a href="#">createSprite() [1/3]</a>	31
4.12.3.3 <a href="#">createSprite() [2/3]</a>	31
4.12.3.4 <a href="#">createSprite() [3/3]</a>	32
4.12.3.5 <a href="#">draw()</a>	32

4.12.3.6 getBit()	32
4.12.3.7 getSprite()	33
4.12.3.8 getTexture()	33
4.12.3.9 initSprite()	33
4.12.3.10 isTextureLoaded()	34
4.12.3.11 setDeferredSprite()	34
4.12.3.12 setSprite() [1/2]	34
4.12.3.13 setSprite() [2/2]	35
4.12.3.14 setTexture()	35
4.13 SpriteTest Class Reference	36
4.14 TestWorld Class Reference	36
4.15 Transform Class Reference	36
4.15.1 Detailed Description	37
4.15.2 Constructor & Destructor Documentation	37
4.15.2.1 Transform() [1/2]	37
4.15.2.2 Transform() [2/2]	37
4.15.2.3 ~Transform()	37
4.15.3 Member Function Documentation	38
4.15.3.1 getBit()	38
4.15.3.2 getPositionVector()	38
4.15.3.3 getRotationVector()	38
4.15.3.4 getScaleVector()	40
4.15.3.5 setTransform()	40
4.16 TransformTest Class Reference	41
4.17 World Class Reference	41
4.17.1 Detailed Description	42
4.17.2 Constructor & Destructor Documentation	42
4.17.2.1 World()	42
4.17.2.2 ~World()	42
4.17.3 Member Function Documentation	42
4.17.3.1 addEntityManager()	43
4.17.3.2 createEntities()	43
4.17.3.3 getEntityManager()	43
4.17.3.4 getEntityManagerMap()	44
4.17.3.5 getNameWorld()	44
4.17.3.6 initWorld()	44
4.17.3.7 setNameWorld()	45



## 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 . . . . .	41
GameEngine . . . . .	19
Sprite . . . . .	28
Transform . . . . .	36
DrawableComponent . . . . .	7
Sprite . . . . .	28
EventEngine . . . . .	16
GameEngine . . . . .	19
testing::Test	
EntityManagerTest . . . . .	15
EntityTest . . . . .	15
EventTest . . . . .	19
SpriteTest . . . . .	36
TestWorld . . . . .	36
TransformTest . . . . .	41



## 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>	7
<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>	15
<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>	
<a href="#">GameEngine</a> class: <a href="#">GameEngine</a> is a class that represents the game engine	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	28
<a href="#">SpriteTest</a>	36
<a href="#">TestWorld</a>	36
<a href="#">Transform</a>	
<a href="#">Transform</a> class: <a href="#">Transform</a> is a class that represents the transform of a Component	36
<a href="#">TransformTest</a>	41
<a href="#">World</a>	
<a href="#">World</a> class: <a href="#">World</a> is a class that represents the world of the game	41



## 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 Audio Class Reference

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

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

### 4.3 Components Class Reference

Inheritance diagram for Components:

#### Public Member Functions

- virtual bool **init** ()
- virtual void **update** ()

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

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

### 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/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  
*Default [Entity](#) constructor.*
- [Entity](#) (std::string nameEntity, [Archetypes](#) newArchetype=[Archetypes](#)())  
*[Entity](#) constructor.*
- [~Entity](#) () override=default  
*[Entity](#) destructor.*
- bool [initEntity](#) ()  
*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 [addDrawable](#) ([Components](#) \*component)
- void [drawEntity](#) (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*
- template<typename T >  
std::size\_t [getComponentTypeID](#) () noexcept
- 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>
template Sprite & Entity::addComponent< Sprite > (
    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 >
template Sprite & Entity::getComponent< Sprite > ( )
```

[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
```

getName(): Get the name of the entity

## Parameters

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

## Returns

std::string: name of the entity

**4.5.3.4 initEntity()**

```
bool Entity::initEntity ( )
```

init(): Initialize the entity

## Parameters

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

## Returns

bool: true if the entity is initialized, false otherwise

#### 4.5.3.5 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

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](#) ()=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 [initEntityManager](#) ()

### 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() )
```

[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 (
    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 [inline]
```

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

##### Parameters

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

#### Returns

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

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.*
- 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

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

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/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  
*< [EventEngine](#) class which manages the events.*
- [GameEngine](#) (sf::VideoMode mode, std::string type, sf::String title, sf::Uint32 style=sf::Style::Default, const sf::ContextSettings &settings=sf::ContextSettings())  
*[GameEngine](#) constructor with parameters.*
- [~GameEngine](#) ()=default  
*[GameEngine](#) destructor.*
- void [run](#) (std::map< std::string, std::unique\_ptr< [World](#) >> mapWorld, std::map< std::string, std::string > pathResources, std::string firstScene)  
*[run\(\)](#): Run the game engine (with parameters).*
- void [run](#) ()  
*[run\(\)](#): Run the game engine (without parameters).*
- void [renderGameEngine](#) ()  
*[renderGameEngine\(\)](#): Render the game engine.*
- void [eventGameEngine](#) ()  
*[eventGameEngine\(\)](#): Manage the events of the game engine.*
- bool [isWindowOpen](#) ()  
*[isWindowOpen\(\)](#): Check if the window is open.*
- void [updateGameEngine](#) ()  
*[updateGameEngine\(\)](#): Update the game engine.*
- void [initialize](#) (std::map< std::string, std::unique\_ptr< [World](#) >> mapWorld, std::map< std::string, std::string > pathResources, std::string firstScene)

- *initialize(): Initialize the game engine.*
- void **initializeSprite** ()
  - initializeSprite(): Initialize the sprites.*
- void **initializeTexture** (std::string path)
  - initializeTexture(): Initialize a texture with its path.*
- void **initializeWorldMap** (std::map< std::string, std::unique\_ptr< **World** >> mapWorld)
  - initializeWorldMap(): Initialize the world map.*
- const auto & **getWindow** ()
  - getWindow(): Get the window.*
- void **setWindow** ()
  - setWindow(): Set the window.*
- **EventEngine** & **getEventEngine** ()
  - getEventEngine(): Get the event engine.*
- void **setCurrentWorld** (**World** \*world)
  - setCurrentWorld(): Set **GameEngine**'s current world.*
- **World** \* **getCurrentWorld** ()
  - getCurrentWorld(): Get **GameEngine**'s current world.*
- **World** & **addWorld** (std::string nameWorld, std::unique\_ptr< **World** > world)
  - addWorld(): Add a world to the world map.*
- **World** & **getWorld** (std::string nameWorld)
  - getWorld(): Get a world from the world map with its name.*
- std::map< std::string, 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.*

## 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]
```

< **EventEngine** class which manages the events.

Default **GameEngine** constructor.

## Parameters

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

## Returns

void

**4.11.2.2 GameEngine()** [2/2]

```
GameEngine::GameEngine (
    sf::VideoMode mode,
    std::string type,
    sf::String title,
    sf::Uint32 style = sf::Style::Default,
    const sf::ContextSettings & settings = sf::ContextSettings() ) [explicit]
```

[GameEngine](#) constructor with parameters.

## Parameters

<i>mode</i>	Video mode.
<i>type</i>	Type of the window.
<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 ( ) [default]
```

[GameEngine](#) destructor.

## Parameters

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

## Returns

void

### 4.11.3 Member Function Documentation

#### 4.11.3.1 addWorld()

```
World & GameEngine::addWorld (
    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 getCurrentWorld()

```
World* GameEngine::getCurrentWorld ( ) [inline]
```

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

##### Parameters

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

**Returns**

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

**4.11.3.4 `getEventEngine()`**

```
EventEngine& GameEngine::getEventEngine ( ) [inline]
```

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

**Parameters**

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

**Returns**

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

**4.11.3.5 `getMapTexture()`**

```
std::map<std::string, sf::Texture> GameEngine::getMapTexture ( ) const [inline]
```

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

**Parameters**

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

**Returns**

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

**4.11.3.6 `getWindow()`**

```
const auto& GameEngine::getWindow ( ) [inline]
```

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

**Parameters**

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

**Returns**

`std::variant<std::unique_ptr<sf::Window>, std::unique_ptr<sf::RenderWindow>>`: The [GameEngine](#)'s window

**4.11.3.7 `getWorld()`**

```
World & GameEngine::getWorld (
    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.

**4.11.3.8 `getWorldMap()`**

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

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

**Parameters**

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

**Returns**

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

**4.11.3.9 `initialize()`**

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

[initialize\(\)](#): Initialize the game engine.

## 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

**4.11.3.10 initializeSprite()**

```
void GameEngine::initializeSprite ( )
```

[initializeSprite\(\)](#): Initialize the sprites.

## Parameters

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

## Returns

void

**4.11.3.11 initializeTexture()**

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

[initializeTexture\(\)](#): Initialize a texture with its path.

## Parameters

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

## Returns

void

#### 4.11.3.12 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

#### 4.11.3.13 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.

#### 4.11.3.14 renderGameEngine()

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

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

##### Parameters

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

##### Returns

void



#### 4.11.3.15 run() [1/2]

```
void GameEngine::run ( )
```

**run()**: Run the game engine (without parameters).

##### Parameters

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

##### Returns

void

#### 4.11.3.16 run() [2/2]

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

#### 4.11.3.17 setCurrentWorld()

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

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

##### Parameters

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

**Returns**

void

**4.11.3.18 setWindow()**

```
void GameEngine::setWindow ( )
```

[setWindow\(\)](#): Set the window.

**Parameters**

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

**Returns**

void

**4.11.3.19 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 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 initSprite ()` 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, std::shared_ptr< 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.*

### 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]
```

[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 <a href="#">RenderWindow</a> 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 initSprite()**

```
bool Sprite::initSprite ( ) 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, std::shared_ptr< 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/include/Sprite.h
- src/Components/all\_components/Sprite.cpp

## 4.13 SpriteTest Class Reference

Inheritance diagram for SpriteTest:

## 4.14 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

## 4.15 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](#) (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*

### 4.15.1 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.

### 4.15.2 Constructor & Destructor Documentation

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

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

Default [Transform](#) constructor.

Parameters

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

Returns

void

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

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

[Transform](#) constructor.

Parameters

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

Returns

void

#### 4.15.2.3 ~Transform()

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

[Transform](#) destructor.

**Parameters**

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

**Returns**

void

### 4.15.3 Member Function Documentation

#### 4.15.3.1 `getBit()`

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

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

**Parameters**

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

**Returns**

int: bitmask of the component

#### 4.15.3.2 `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

#### 4.15.3.3 `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

**4.15.3.4 `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

**4.15.3.5 `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 ( <code>std::string</code> , <code>std::vector&lt;float&gt;</code> ).
---------------------	--

**Returns**

`void`

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

- `src/Components/all_components/include/Transform.h`
- `src/Components/all_components/Transform.cpp`

## 4.16 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

## 4.17 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 [initWorld](#) ()  
    *init(): Initialize the World.*

## Additional Inherited Members

### 4.17.1 Detailed Description

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

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

### 4.17.2 Constructor & Destructor Documentation

#### 4.17.2.1 World()

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

< Name of the world.

Default [World](#) constructor.

##### Parameters

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

##### Returns

void

#### 4.17.2.2 ~World()

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

[World](#) destructor.

##### Parameters

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

##### Returns

void

### 4.17.3 Member Function Documentation



#### 4.17.3.1 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.

#### 4.17.3.2 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

#### 4.17.3.3 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.

**4.17.3.4 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.

**4.17.3.5 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.

**4.17.3.6 initWorld()**

```
bool World::initWorld ( ) [inline]
```

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

**Parameters**

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

**Returns**

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

**4.17.3.7 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/include/world.h
- src/World/world.cpp



# Index

- ~Entity
  - Entity, [9](#)
- ~EntityManager
  - EntityManager, [13](#)
- ~EventEngine
  - EventEngine, [17](#)
- ~GameEngine
  - GameEngine, [21](#)
- ~Sprite
  - Sprite, [30](#)
- ~Transform
  - Transform, [37](#)
- ~World
  - World, [42](#)
- addComponent
  - Entity, [10](#)
- addEntity
  - EntityManager, [13](#)
- addEntityManager
  - World, [42](#)
- addKeyPressed
  - EventEngine, [17](#)
- addWorld
  - GameEngine, [22](#)
- applyDeferredSprite
  - Sprite, [30](#)
- Archetypes, [7](#)
- Audio, [7](#)
- Components, [7](#)
- createEntities
  - World, [43](#)
- createSprite
  - Sprite, [31](#)
- draw
  - Sprite, [32](#)
- DrawableComponent, [7](#)
- Entity, [8](#)
  - ~Entity, [9](#)
  - addComponent, [10](#)
  - Entity, [9](#)
  - getComponent, [10](#)
  - getName, [11](#)
  - initEntity, [11](#)
  - setName, [11](#)
- EntityManager, [12](#)
  - ~EntityManager, [13](#)
- addEntity, [13](#)
- EntityManager, [12](#)
- getEntities, [13](#)
- getEntity, [14](#)
- getEntityMap, [14](#)
- EntityManagerTest, [15](#)
- EntityTest, [15](#)
- EventEngine, [16](#)
  - ~EventEngine, [17](#)
  - addKeyPressed, [17](#)
  - EventEngine, [16](#)
  - getEvent, [17](#)
  - getKeyPressedMap, [18](#)
  - init, [18](#)
- eventGameEngine
  - GameEngine, [22](#)
- EventTest, [19](#)
- GameEngine, [19](#)
  - ~GameEngine, [21](#)
  - addWorld, [22](#)
  - eventGameEngine, [22](#)
  - GameEngine, [20](#), [21](#)
  - getCurrentWorld, [22](#)
  - getEventEngine, [23](#)
  - getMapTexture, [23](#)
  - getWindow, [23](#)
  - getWorld, [24](#)
  - getWorldMap, [24](#)
  - initialize, [24](#)
  - initializeSprite, [25](#)
  - initializeTexture, [25](#)
  - initializeWorldMap, [25](#)
  - isWindowOpen, [26](#)
  - renderGameEngine, [26](#)
  - run, [26](#), [27](#)
  - setCurrentWorld, [27](#)
  - setWindow, [28](#)
  - updateGameEngine, [28](#)
- getBit
  - Sprite, [32](#)
  - Transform, [38](#)
- getComponent
  - Entity, [10](#)
- getCurrentWorld
  - GameEngine, [22](#)
- getEntities
  - EntityManager, [13](#)
- getEntity
  - EntityManager, [14](#)

- getEntityManager
  - World, [43](#)
- getEntityManagerMap
  - World, [44](#)
- getEntityMap
  - EntityManager, [14](#)
- getEvent
  - EventEngine, [17](#)
- getEventEngine
  - GameEngine, [23](#)
- getKeyPressedMap
  - EventEngine, [18](#)
- getMapTexture
  - GameEngine, [23](#)
- getName
  - Entity, [11](#)
- getNameWorld
  - World, [44](#)
- getPositionVector
  - Transform, [38](#)
- getRotationVector
  - Transform, [38](#)
- getScaleVector
  - Transform, [40](#)
- getSprite
  - Sprite, [33](#)
- getTexture
  - Sprite, [33](#)
- getWindow
  - GameEngine, [23](#)
- getWorld
  - GameEngine, [24](#)
- getWorldMap
  - GameEngine, [24](#)
- init
  - EventEngine, [18](#)
- initEntity
  - Entity, [11](#)
- initialize
  - GameEngine, [24](#)
- initializeSprite
  - GameEngine, [25](#)
- initializeTexture
  - GameEngine, [25](#)
- initializeWorldMap
  - GameEngine, [25](#)
- initSprite
  - Sprite, [33](#)
- initWorld
  - World, [44](#)
- isTextureLoaded
  - Sprite, [34](#)
- isWindowOpen
  - GameEngine, [26](#)
- renderGameEngine
  - GameEngine, [26](#)
- run
  - GameEngine, [26, 27](#)
- setCurrentWorld
  - GameEngine, [27](#)
- setDeferredSprite
  - Sprite, [34](#)
- setName
  - Entity, [11](#)
- setNameWorld
  - World, [45](#)
- setSprite
  - Sprite, [34, 35](#)
- setTexture
  - Sprite, [35](#)
- setTransform
  - Transform, [40](#)
- setWindow
  - GameEngine, [28](#)
- Sprite, [28](#)
  - ~Sprite, [30](#)
  - applyDeferredSprite, [30](#)
  - createSprite, [31](#)
  - draw, [32](#)
  - getBit, [32](#)
  - getSprite, [33](#)
  - getTexture, [33](#)
  - initSprite, [33](#)
  - isTextureLoaded, [34](#)
  - setDeferredSprite, [34](#)
  - setSprite, [34, 35](#)
  - setTexture, [35](#)
  - Sprite, [29, 30](#)
- SpriteTest, [36](#)
- TestWorld, [36](#)
- Transform, [36](#)
  - ~Transform, [37](#)
  - getBit, [38](#)
  - getPositionVector, [38](#)
  - getRotationVector, [38](#)
  - getScaleVector, [40](#)
  - setTransform, [40](#)
  - Transform, [37](#)
- TransformTest, [41](#)
- updateGameEngine
  - GameEngine, [28](#)
- World, [41](#)
  - ~World, [42](#)
  - addEntityManager, [42](#)
  - createEntities, [43](#)
  - getEntityManager, [43](#)
  - getEntityManagerMap, [44](#)
  - getNameWorld, [44](#)
  - initWorld, [44](#)
  - setNameWorld, [45](#)
  - World, [42](#)