

R-Type - Engine

Generated by Doxygen 1.9.1

1 Engine	1
1.1 Compilation	1
1.1.1 Linux	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 Class Documentation	7
4.1 Archetypes Class Reference	7
4.2 Audio Class Reference	7
4.3 Components Class Reference	7
4.3.1 Detailed Description	8
4.3.2 Constructor & Destructor Documentation	8
4.3.2.1 Components()	8
4.3.2.2 ~Components()	9
4.3.3 Member Function Documentation	9
4.3.3.1 init()	9
4.4 DrawableComponent Class Reference	10
4.4.1 Detailed Description	10
4.4.2 Constructor & Destructor Documentation	10
4.4.2.1 ~DrawableComponent()	10
4.4.3 Member Function Documentation	11
4.4.3.1 draw()	11
4.5 Entity Class Reference	11
4.5.1 Detailed Description	13
4.5.2 Constructor & Destructor Documentation	13
4.5.2.1 Entity() [1/2]	13
4.5.2.2 Entity() [2/2]	14
4.5.2.3 ~Entity()	14
4.5.3 Member Function Documentation	14
4.5.3.1 addComponent()	15
4.5.3.2 addDrawable()	15
4.5.3.3 drawEntity()	15
4.5.3.4 getComponent()	16
4.5.3.5 getComponentArrays()	16
4.5.3.6 getComponentBitset()	17
4.5.3.7 getComponentTypeID()	17
4.5.3.8 getDrawableComponents()	17
4.5.3.9 getName()	18
4.5.3.10 initEntity()	18

4.5.3.11 setName()	18
4.6 EntityManager Class Reference	19
4.6.1 Constructor & Destructor Documentation	20
4.6.1.1 EntityManager()	20
4.6.1.2 ~EntityManager()	21
4.6.2 Member Function Documentation	21
4.6.2.1 addEntity()	21
4.6.2.2 getEntities()	22
4.6.2.3 getEntity()	22
4.6.2.4 getEntityMap()	22
4.6.2.5 initEntityManager()	23
4.7 EntityManagerTest Class Reference	23
4.8 EntityTest Class Reference	25
4.9 EventEngine Class Reference	26
4.9.1 Detailed Description	26
4.9.2 Constructor & Destructor Documentation	27
4.9.2.1 EventEngine()	27
4.9.2.2 ~EventEngine()	27
4.9.3 Member Function Documentation	27
4.9.3.1 addKeyPressed()	27
4.9.3.2 addMouseButtonPressed()	28
4.9.3.3 addMouseMoved()	28
4.9.3.4 getEvent()	29
4.9.3.5 getKeyPressedMap()	29
4.9.3.6 getMouseButtonPressedMap()	29
4.9.3.7 getMouseMovedMap()	30
4.9.3.8 init()	30
4.10 EventTest Class Reference	30
4.11 GameEngine Class Reference	31
4.11.1 Detailed Description	34
4.11.2 Constructor & Destructor Documentation	34
4.11.2.1 GameEngine() [1/2]	34
4.11.2.2 GameEngine() [2/2]	35
4.11.2.3 ~GameEngine()	35
4.11.3 Member Function Documentation	36
4.11.3.1 addWorld()	36
4.11.3.2 eventGameEngine()	36
4.11.3.3 getCurrentWorld()	36
4.11.3.4 getEventEngine()	37
4.11.3.5 getFilesTexture()	37
4.11.3.6 getMapTexture()	37
4.11.3.7 getWindow()	38

4.11.3.8	getWorld()	38
4.11.3.9	getWorldMap()	38
4.11.3.10	initialize()	39
4.11.3.11	initializeSprite()	39
4.11.3.12	initializeTexture()	40
4.11.3.13	initializeWorldMap()	40
4.11.3.14	isWindowOpen()	40
4.11.3.15	renderGameEngine()	41
4.11.3.16	run() [1/2]	41
4.11.3.17	run() [2/2]	41
4.11.3.18	setCurrentWorld()	42
4.11.3.19	setWindow()	42
4.11.3.20	updateGameEngine()	42
4.12	GameEngineTest Class Reference	43
4.13	Rect< T > Class Template Reference	45
4.13.1	Detailed Description	46
4.13.2	Constructor & Destructor Documentation	46
4.13.2.1	Rect()	46
4.13.2.2	~Rect()	47
4.13.3	Member Function Documentation	47
4.13.3.1	contains()	47
4.13.3.2	getHeight()	48
4.13.3.3	getLeft()	48
4.13.3.4	getRect()	49
4.13.3.5	getTop()	49
4.13.3.6	getWidth()	49
4.14	Script Class Reference	50
4.15	Sprite Class Reference	50
4.15.1	Detailed Description	52
4.15.2	Constructor & Destructor Documentation	52
4.15.2.1	Sprite() [1/2]	52
4.15.2.2	Sprite() [2/2]	53
4.15.2.3	~Sprite()	53
4.15.3	Member Function Documentation	53
4.15.3.1	applyDeferredSprite()	53
4.15.3.2	createSprite() [1/3]	54
4.15.3.3	createSprite() [2/3]	54
4.15.3.4	createSprite() [3/3]	54
4.15.3.5	draw()	55
4.15.3.6	getBit()	55
4.15.3.7	getSprite()	55
4.15.3.8	getTexture()	57

4.15.3.9	initSprite()	57
4.15.3.10	isTextureLoaded()	57
4.15.3.11	setDeferredSprite()	58
4.15.3.12	setPosition() [1/2]	58
4.15.3.13	setPosition() [2/2]	58
4.15.3.14	setRotation() [1/2]	59
4.15.3.15	setRotation() [2/2]	59
4.15.3.16	setScale() [1/2]	59
4.15.3.17	setScale() [2/2]	60
4.15.3.18	setSprite() [1/2]	60
4.15.3.19	setSprite() [2/2]	60
4.15.3.20	setTexture()	61
4.15.3.21	setTransformSprite() [1/2]	61
4.15.3.22	setTransformSprite() [2/2]	62
4.16	SpriteTest Class Reference	62
4.17	TestWorld Class Reference	63
4.18	toSFML Class Reference	64
4.18.1	Detailed Description	65
4.18.2	Constructor & Destructor Documentation	65
4.18.2.1	toSFML()	65
4.18.2.2	~toSFML()	66
4.18.3	Member Function Documentation	66
4.18.3.1	toSFMLRect()	66
4.19	Transform Class Reference	67
4.19.1	Detailed Description	68
4.19.2	Constructor & Destructor Documentation	68
4.19.2.1	Transform()	68
4.19.2.2	~Transform()	69
4.19.3	Member Function Documentation	69
4.19.3.1	getBit()	69
4.19.3.2	getPosition()	69
4.19.3.3	getRotation()	70
4.19.3.4	getScale()	70
4.19.3.5	getTransformStruct()	70
4.19.3.6	init()	71
4.19.3.7	setTransform()	71
4.19.3.8	setTransformPosition()	71
4.19.3.9	setTransformRotation()	72
4.19.3.10	setTransformScale()	72
4.20	TransformTest Class Reference	73
4.21	Vector2< T > Class Template Reference	73
4.21.1	Detailed Description	74

4.21.2 Constructor & Destructor Documentation	74
4.21.2.1 Vector2()	74
4.21.2.2 ~Vector2()	75
4.21.3 Member Function Documentation	75
4.21.3.1 getVector2Struct()	75
4.21.3.2 getX()	76
4.21.3.3 getY()	76
4.22 World Class Reference	76
4.22.1 Detailed Description	78
4.22.2 Constructor & Destructor Documentation	78
4.22.2.1 World()	78
4.22.2.2 ~World()	79
4.22.3 Member Function Documentation	79
4.22.3.1 addEntityManager()	79
4.22.3.2 createEntities()	79
4.22.3.3 getEntityManager()	80
4.22.3.4 getEntityManagerMap()	80
4.22.3.5 getNameWorld()	80
4.22.3.6 initWorld()	81
4.22.3.7 setNameWorld()	81
Index	83

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
Audio	7
Components	7
Entity	11
EntityManager	19
World	76
GameEngine	31
Transform	67
Rect< T >	45
Sprite	50
DrawableComponent	10
Sprite	50
EventEngine	26
GameEngine	31
Script	50
testing::Test	
EntityManagerTest	23
EntityTest	25
EventTest	30
GameEngineTest	43
SpriteTest	62
TestWorld	63
TransformTest	73
toSFML	64
Sprite	50
Vector2< T >	73
Vector2< float >	73

Chapter 3

Class Index

3.1 Class List

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

Archetypes	7
Audio	7
Components	
Components class: Components is a class that represents a component in the game	7
DrawableComponent	
DrawableComponent class: DrawableComponent is a class that represents a drawable component in the game	10
Entity	
Entity class: Entity is a class that represents an entity in the game	11
EntityManager	19
EntityManagerTest	23
EntityTest	25
EventEngine	
EventEngine class: EventEngine is a class that represents the event engine of the game	26
EventTest	30
GameEngine	
GameEngine class: GameEngine is a class that represents the game engine	31
GameEngineTest	43
Rect< T >	
Rect class: Rect is a class that represents a rectangle	45
Script	50
Sprite	
Sprite class: Sprite is a class that represents the rendering properties of a Component	50
SpriteTest	62
TestWorld	63
toSFML	
toSFML class: toSFML is a class that convert some class into SFML class	64
Transform	
Transform class: Transform is a class that represents the transform of a Component	67
TransformTest	73
Vector2< T >	
Vector class: Vector is a class that represents a vector in 2 dimensions	73
World	
World class: World is a class that represents the world of the game	76

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:

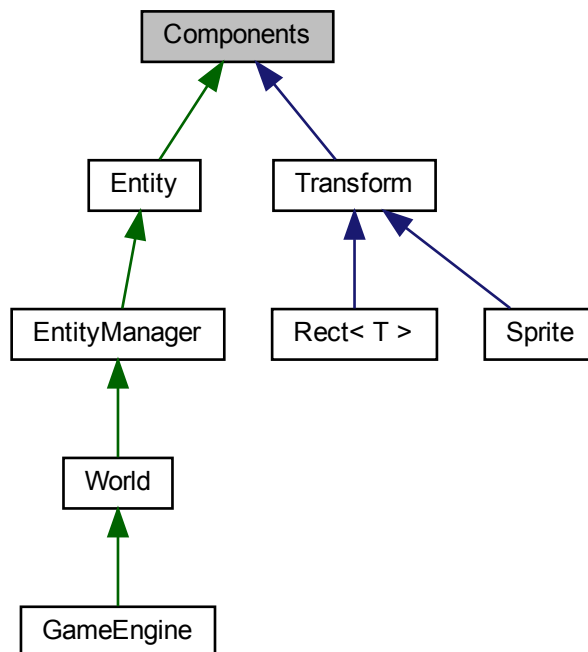
- `src/Components/all_components/include/Audio.h`

4.3 Components Class Reference

[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](#) ()
[init](#)(): Initialize the component
- virtual void [update](#) (sf::Time timeDelta)=0

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 init()**

```
virtual bool Components::init ( ) [inline], [virtual]
```

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

Parameters

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

Returns

bool: true if the component is initialized, false otherwise

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

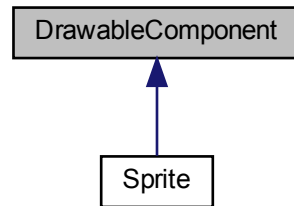
- `src/Components/include/Components.h`

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

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

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 [Sprite](#).

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

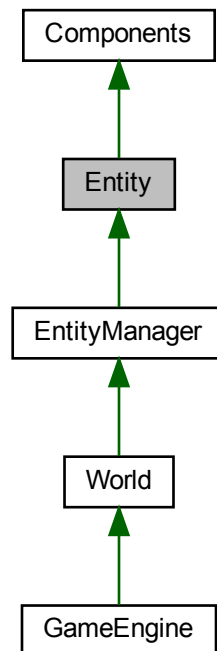
- src/Components/include/DrawableComponent.h

4.5 Entity Class Reference

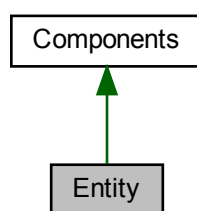
[Entity](#) class: [Entity](#) is a class that represents an entity in the game.

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

Inheritance diagram for Entity:



Collaboration diagram for Entity:



Public Member Functions

- [Entity](#) ()=default
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
 - genName(): Get the name of the entity*
- void `update` (sf::Time deltaTime) override
- void `setName` (std::string newName)
 - setName(): Set the name of the entity*
- void `addDrawable` (Components *component)
 - addDrawable(): Add 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>
 - T & `getComponent` ()
 - getComponent(): Get a component from the entity*
- template<typename T>
 - std::size_t `getComponentTypeID` () noexcept
 - getComponentTypeID(): Get the ID of a component*
- std::bitset< 3 > `getComponentBitset` () const
 - getComponentBitset(): Get the bitset of the components*
- std::vector< DrawableComponent * > `getDrawableComponents` () const
 - getDrawableComponents(): Get the drawable components of the entity*
- std::array< Components *, 3 > `getComponentArrays` () const
 - getComponentArrays(): Get the array of components*

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 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 drawEntity()

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

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

Parameters

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

Returns

void

4.5.3.4 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

4.5.3.5 GetComponentArrays()

```
std::array<Components*, 3> Entity::GetComponentArrays ( ) const [inline]
```

[GetComponentArrays\(\)](#): Get the array of components

Parameters

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

Returns

std::array<Components*, 3>: array of components

4.5.3.6 GetComponentBitset()

```
std::bitset<3> Entity::GetComponentBitset ( ) const [inline]
```

[GetComponentBitset\(\)](#): Get the bitset of the components

Parameters

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

Returns

std::bitset<3>: bitset of the components

4.5.3.7 GetComponentTypeID()

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

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

Template Parameters

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

Parameters

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

Returns

std::size_t: ID of the component

4.5.3.8 getDrawableComponents()

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

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

Parameters

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

Returns

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

4.5.3.9 getName()

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

getName(): Get the name of the entity

Parameters

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

Returns

std::string: name of the entity

4.5.3.10 initEntity()

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

init(): Initialize the entity

Parameters

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

Returns

bool: true if the entity is initialized, false otherwise

4.5.3.11 setName()

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

setName(): Set the name of the entity

Parameters

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

Returns

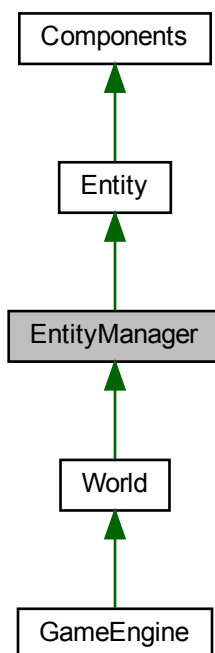
void

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

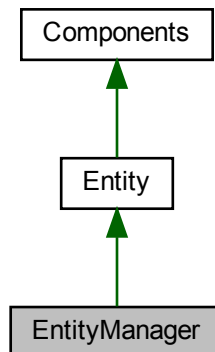
- src/Entity/include/entity.h
- src/Entity/entity.cpp

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

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.

4.6.2.5 initEntityManager()

```
bool EntityManager::initEntityManager ( ) [inline]
```

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

Parameters

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

Returns

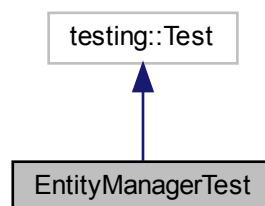
bool: true if the [EntityManager](#) is initialized, false otherwise.

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

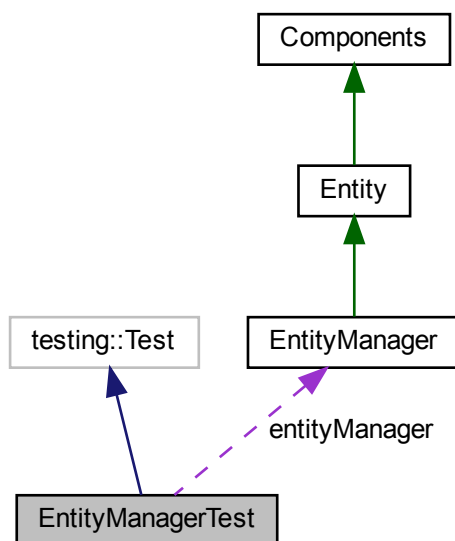
- src/Entity/include/entityManager.h
- src/Entity/entityManager.cpp

4.7 EntityManagerTest Class Reference

Inheritance diagram for EntityManagerTest:



Collaboration diagram for EntityManagerTest:



Protected Member Functions

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

Protected Attributes

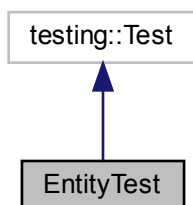
- [EntityManager](#) **entityManager** {}

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

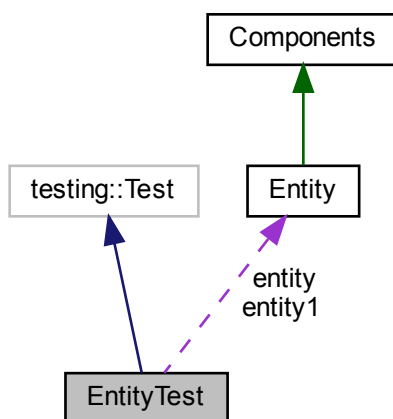
- tests/Entity/TestEntityManager.cpp

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:

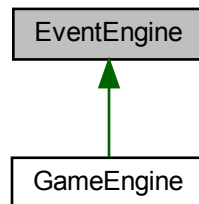
- tests/Entity/TestEntity.cpp

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.
- void [addMouseButtonPressed](#) (sf::Mouse::Button mouse, std::function< void()> function)
[addMouseButtonPressed\(\)](#): Add a mouse button pressed to the map.
- void [addMouseMoved](#) (std::string nameEntity, 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.

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 addMouseButtonPressed()

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

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

Parameters

<i>nameEntity</i>	: Name of the Entity 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 ( ) [inline]
```

[`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 ( ) [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.

4.9.3.6 `getMouseButtonPressedMap()`

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

[`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.

4.9.3.7 getMouseMovedMap()

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

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.

4.9.3.8 init()

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

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

Parameters

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

Returns

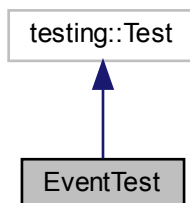
bool: True if the [EventEngine](#) is initialized, false otherwise.

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

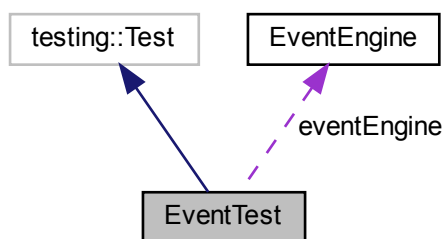
- src/Event/include/eventEngine.h
- src/Event/eventEngine.cpp

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:

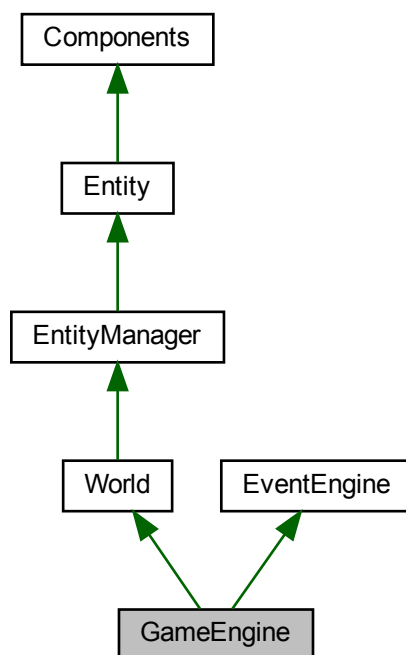
- `tests/Event/TestEvent.cpp`

4.11 GameEngine Class Reference

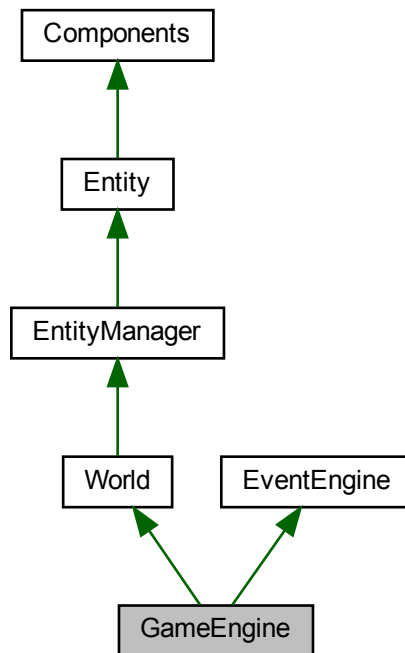
[GameEngine](#) class: [GameEngine](#) is a class that represents the game engine.

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

Inheritance diagram for GameEngine:



Collaboration diagram for GameEngine:



Public Member Functions

- `GameEngine()`=default
< Time of the game. Using with the Clock.
- `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.
- `std::vector< std::string > getFilesTexture(std::string pathDirectory)`

- *getFilesTexture(): Get all the textures files in the given directory.*
- 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 the textures with their 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, 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.*

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,
    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 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 ( ) [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>	World 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 [getFilesTexture\(\)](#)

```
std::vector< std::string > GameEngine::getFilesTexture (
    std::string pathDirectory )
```

[getFilesTexture\(\)](#): Get all the textures files in the given directory.

Parameters

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

Returns

std::vector<std::string>: Vector of the textures files' names.

4.11.3.6 [getMapTexture\(\)](#)

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

[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.

4.11.3.7 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

4.11.3.8 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.

4.11.3.9 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.

4.11.3.10 initialize()

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

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

Parameters

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

Returns

`void`

4.11.3.11 initializeSprite()

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

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

Parameters

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

Returns

`void`

4.11.3.12 initializeTexture()

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

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

Parameters

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

Returns

void

4.11.3.13 initializeWorldMap()

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

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

Parameters

<i>mapWorld</i>	Map of World classes' unique pointers.
-----------------	--

Returns

void

4.11.3.14 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.

4.11.3.15 renderGameEngine()

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

renderGameEngine(): Render the game engine.

Parameters

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

Returns

void

4.11.3.16 run() [1/2]

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

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

Parameters

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

Returns

void

4.11.3.17 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 World classes' unique pointers.
<i>pathResources</i>	Map of the path of the ressources (assets).
<i>firstScene</i>	Name of the first scene.

Returns

void

4.11.3.18 setCurrentWorld()

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

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

Parameters

<i>world</i>	World to set.
--------------	-------------------------------

Returns

void

4.11.3.19 setWindow()

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

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

Parameters

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

Returns

void

4.11.3.20 updateGameEngine()

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

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

Parameters

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

Returns

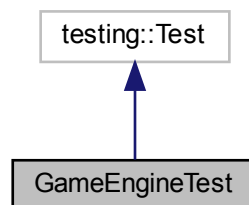
void

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

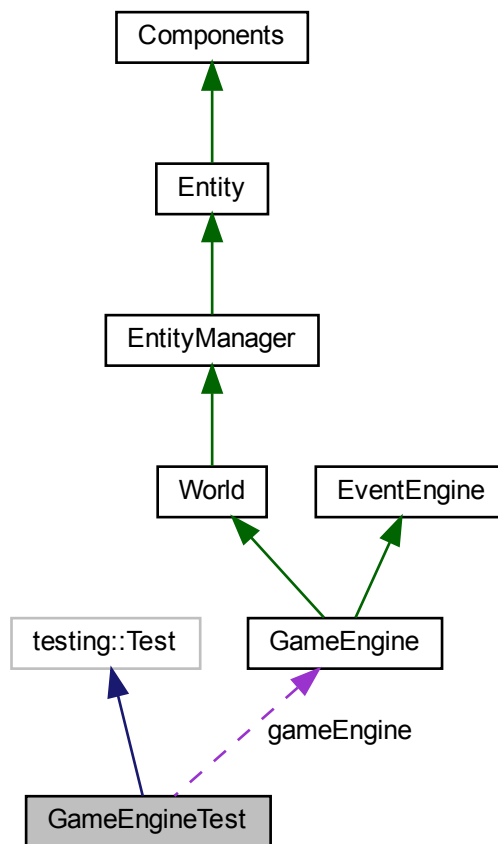
- src/GameEngine/include/gameEngine.h
- src/GameEngine/gameEngine.cpp

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:

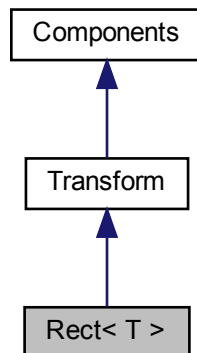
- tests/GameEngine/TestGameEngine.cpp

4.13 Rect< T > Class Template Reference

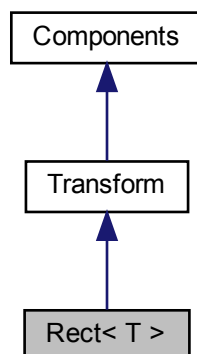
Rect class: Rect is a class that represents a rectangle.

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

Inheritance diagram for Rect< T >:



Collaboration diagram for Rect< T >:



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.

4.13.1 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.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 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.

Parameters

<i>width</i>	Width of your rectangle.
<i>height</i>	Height of your rectangle.

Returns

void

4.13.2.2 ~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

4.13.3 Member Function Documentation**4.13.3.1 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,...).

4.13.3.2 getHeight()

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

[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,...).

4.13.3.3 getLeft()

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

[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,...).

4.13.3.4 getRect()

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

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

Parameters

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

Returns

[Rect](#)

4.13.3.5 getTop()

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

[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,...).

4.13.3.6 getWidth()

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

[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

4.14 Script Class Reference

Public Member Functions

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

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

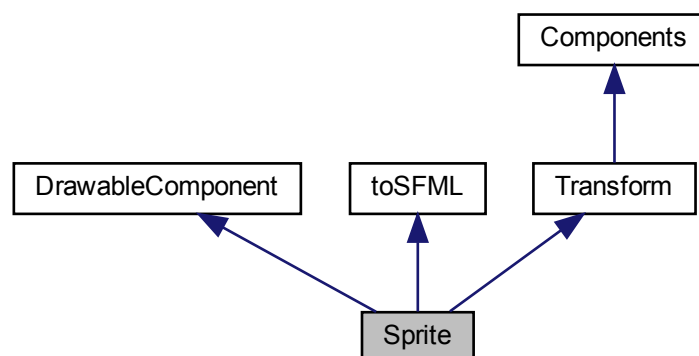
- src/Script/include/Script.h

4.15 Sprite Class Reference

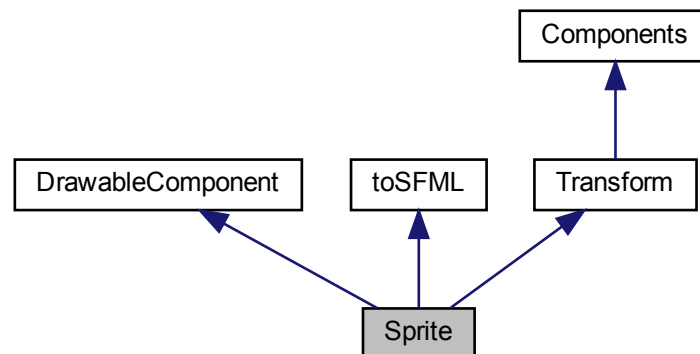
[Sprite](#) class: [Sprite](#) is a class that represents the rendering properties of a Component.

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

Inheritance diagram for Sprite:



Collaboration diagram for Sprite:



Public Member Functions

- `Sprite ()`=default
< Doing the animation.
- `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 update (sf::Time deltaTime)` override
- `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, bool animate=false, std::vector< Rect< int >> newFrames=std::vector< Rect< int >>(), int durationOfFrame=100)`

- 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 [setTransformSprite](#) ([Vector2](#)< float > newPosition, float newRotation, [Vector2](#)< float > newScale)
 - setTransformSprite(): Set the sprite transform with new value and set the value on the [Transform](#) component.*
- void [setTransformSprite](#) ()
 - setTransformSprite(): Set the transform of the sprite based on the [Transform](#) component value.*
- void [setPosition](#) ([Vector2](#)< float > newPosition)
 - setPosition(): Set the position of the sprite with new value.*
- void [setPosition](#) ()
 - setPosition(): Set the position of the sprite based on the [Transform](#) component value.*
- void [setRotation](#) (float newRotation)
 - setRotation(): Set the rotation of the sprite with new value.*
- void [setRotation](#) ()
 - setRotation(): Set the rotation of the sprite based on the [Transform](#) component value.*
- void [setScale](#) ([Vector2](#)< float > newScale)
 - setScale(): Set the the scale of the sprite with new value.*
- void [setScale](#) ()
 - setScale(): Set the scale of the sprite based on the [Transform](#) component value.*
- 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.*
- [Rect](#)< float > [getBounds](#) () const

4.15.1 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.

4.15.2 Constructor & Destructor Documentation

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

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

< Doing the animation.

Default [Sprite](#) constructor.

Parameters

void	
------	--

Returns

void

4.15.2.2 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

4.15.2.3 ~Sprite()

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

[Sprite](#) destructor.

Parameters

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

Returns

void

4.15.3 Member Function Documentation**4.15.3.1 applyDeferredSprite()**

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

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

Parameters

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

Returns

void

4.15.3.2 createSprite() [1/3]

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

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

Parameters

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

Returns

void

4.15.3.3 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

4.15.3.4 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

4.15.3.5 draw()

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

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

Parameters

<i>window</i>	SFML RenderWindow where the Sprite will be drawn.
---------------	---

Returns

void

Implements [DrawableComponent](#).

4.15.3.6 getBit()

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

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

Parameters

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

Returns

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

4.15.3.7 getSprite()

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

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

Parameters

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

Returns

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

4.15.3.8 `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

4.15.3.9 `initSprite()`

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

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

Parameters

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

Returns

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

4.15.3.10 `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.

4.15.3.11 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

4.15.3.12 setPosition() [1/2]

```
void Sprite::setPosition ( )
```

[setPosition\(\)](#): Set the position of the sprite based on the [Transform](#) component value.

Parameters

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

Returns

void

4.15.3.13 setPosition() [2/2]

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

[setPosition\(\)](#): Set the position of the sprite with new value.

Parameters

<i>newPosition</i>	The new Vector2<float> position.
--------------------	--

Returns

void

4.15.3.14 setRotation() [1/2]

```
void Sprite::setRotation ( )
```

[setRotation\(\)](#): Set the rotation of the sprite based on the [Transform](#) component value.

Parameters

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

Returns

void

4.15.3.15 setRotation() [2/2]

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

[setRotation\(\)](#): Set the rotation of the sprite with new value.

Parameters

<i>newRotation</i>	The new float rotation.
--------------------	-------------------------

Returns

void

4.15.3.16 setScale() [1/2]

```
void Sprite::setScale ( )
```

[setScale\(\)](#): Set the scale of the sprite based on the [Transform](#) component value.

Parameters

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

Returns

void

4.15.3.17 setScale() [2/2]

```
void Sprite::setScale (
    Vector2< float > newScale )
```

setScale(): Set the the scale of the sprite with new value.

Parameters

<i>newScale</i>	The new Vector2<float> scale.
-----------------	---

Returns

void

4.15.3.18 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 Sprite for rendering
---------------	---

Returns

void

4.15.3.19 setSprite() [2/2]

```
void Sprite::setSprite (
    std::map< std::string, std::shared_ptr< sf::Texture >> mapTexture,
```

```
std::string nameTexture,  
bool animate = false,  
std::vector< Rect< int >> newFrames = std::vector<Rect<int>>(),  
int durationOfFrame = 100 )
```

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

4.15.3.20 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

4.15.3.21 setTransformSprite() [1/2]

```
void Sprite::setTransformSprite ( )
```

setTransformSprite(): Set the transform of the sprite based on the [Transform](#) component value.

Parameters

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

Returns

void

4.15.3.22 setTransformSprite() [2/2]

```
void Sprite::setTransformSprite (
    Vector2< float > newPosition,
    float newRotation,
    Vector2< float > newScale )
```

[setTransformSprite\(\)](#): Set the sprite transform with new value and set the value on the [Transform](#) component.

Parameters

<i>newPosition</i>	The new Vector2<float> position.
<i>newRotation</i>	The new float rotation.
<i>newScale</i>	The new Vector2<float> scale.

Returns

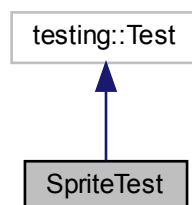
void

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

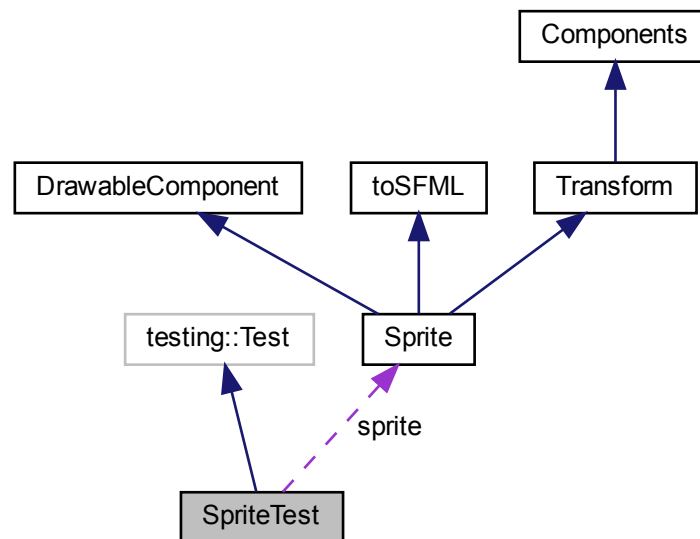
- src/Components/all_components/include/Sprite.h
- src/Components/all_components/Sprite.cpp

4.16 SpriteTest Class Reference

Inheritance diagram for SpriteTest:



Collaboration diagram for SpriteTest:



Protected Attributes

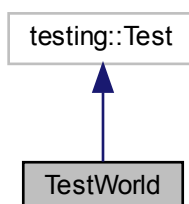
- [Sprite](#) `sprite`

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

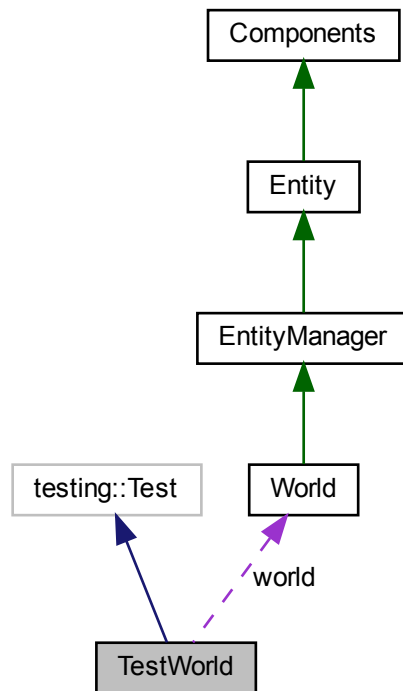
- tests/Components/all_components/TestSprite.cpp

4.17 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:

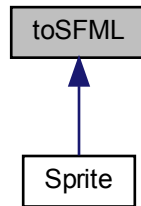
- tests/World/TestWorld.cpp

4.18 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>`.

4.18.1 Detailed Description

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

Convert some class in SFML class.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 toSFML()

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

Default `toSFML` constructor.

Parameters

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

Returns

void

4.18.2.2 ~toSFML()

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

[toSFML](#) destructor.

Parameters

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

Returns

void

4.18.3 Member Function Documentation**4.18.3.1 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:

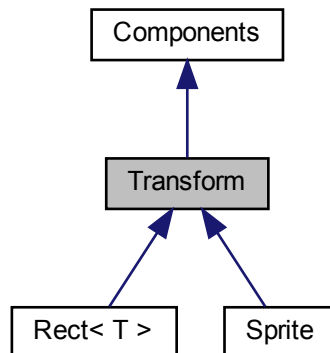
- src/toSFML/include/toSFML.h
- src/toSFML/toSFML.cpp

4.19 Transform Class Reference

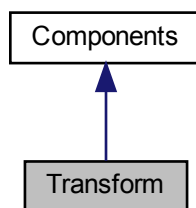
Transform class: **Transform** is a class that represents the transform of a Component.

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

Inheritance diagram for Transform:



Collaboration diagram for Transform:



Public Member Functions

- **Transform** ()
*Default **Transform** constructor.*
- bool **init** () const
***init**(): Initialize the component*
- **~Transform** () override=default
***Transform** destructor.*
- void **update** (sf::Time deltaTime) override

- int `getBit ()` const
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 `getTransformStruct ()` const
getTransformStruct(): Get the the transform of the component;
- void `setTransform (Vector2< float > newPosition, float newRotation, Vector2< float > newScale)`
setTransformStruct(): Set the transform of the component;
- void `setTransformPosition (Vector2< float > newPosition)`
setTransformPosition(): Set the transform position of the component;
- void `setTransformRotation (float newRotation)`
setTransformRotation(): Set the transform rotation of the component;
- void `setTransformScale (Vector2< float > newScale)`
setTransformScale(): Set the transform scale of the component;

4.19.1 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.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 Transform()

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

Default `Transform` constructor.

Parameters

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

Returns

`void`

4.19.2.2 ~Transform()

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

[Transform](#) destructor.

Parameters

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

Returns

void

4.19.3 Member Function Documentation

4.19.3.1 getBit()

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

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

Parameters

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

Returns

int: bitmask of the component

4.19.3.2 getPosition()

```
Vector2<float> Transform::getPosition ( ) const [inline]
```

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

Parameters

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

Returns

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

4.19.3.3 getRotation()

```
float Transform::getRotation ( ) const [inline]
```

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

Parameters

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

Returns

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

4.19.3.4 getScale()

```
Vector2<float> Transform::getScale ( ) const [inline]
```

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

Parameters

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

Returns

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

4.19.3.5 getTransformStruct()

```
TransformStruct Transform::getTransformStruct ( ) const [inline]
```

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

Parameters

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

Returns

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

4.19.3.6 init()

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

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

Parameters

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

Returns

bool: true if the component is initialized, false otherwise

4.19.3.7 setTransform()

```
void Transform::setTransform (
    Vector2< float > newPosition,
    float newRotation,
    Vector2< float > newScale )
```

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

Parameters

<i>newPosition</i>	: the new Vector2<float> position.
<i>newRotation</i>	: the new float rotation.
<i>newScale</i>	: the new Vector2<float> scale.

Returns

void

4.19.3.8 setTransformPosition()

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

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

Parameters

<i>newPosition</i>	: the new Vector2<float> position.
--------------------	--

Returns

void

4.19.3.9 setTransformRotation()

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

[setTransformRotation\(\)](#): Set the transform rotation of the component;

Parameters

<i>newRotation</i>	: the new float rotation.
--------------------	---------------------------

Returns

void

4.19.3.10 setTransformScale()

```
void Transform::setTransformScale (
    Vector2< float > newScale )
```

[setTransformScale\(\)](#): Set the transform scale of the component;

Parameters

<i>newScale</i>	: the new Vector2<float> scale.
-----------------	---

Returns

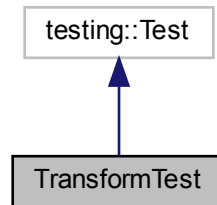
void

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

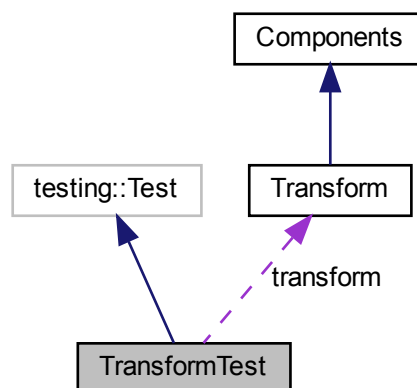
- src/Components/all_components/include/Transform.h
- src/Components/all_components/Transform.cpp

4.20 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

4.21 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](#) (T x, T y)
< Variable for using the value of the Vector2Struct.
- [~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.

4.21.1 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.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 Vector2()

```
template<typename T >
Vector2< T >::Vector2 (
    T x,
    T y ) [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

4.21.2.2 ~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

4.21.3 Member Function Documentation**4.21.3.1 getVector2Struct()**

```
template<typename T >
Vector2Struct Vector2< T >::getVector2Struct ( ) const [inline]
```

[getVector2Struct\(\)](#): Get the using Vector2Struct.

Parameters

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

Returns

Vector2Struct

4.21.3.2 getX()

```
template<typename T >
T Vector2< T >::getX ( ) const [inline]
```

[getX\(\)](#): Get x of Vector2Struct.

Template Parameters

--	--

4.21.3.3 getY()

```
template<typename T >
T Vector2< T >::getY ( ) const [inline]
```

[getY\(\)](#): Get y of Vector2Struct.

Template Parameters

--	--

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

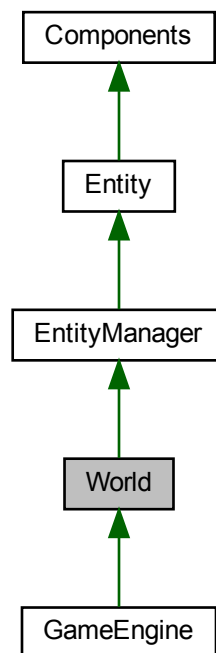
- `src/Other/include/Vector2.h`

4.22 World Class Reference

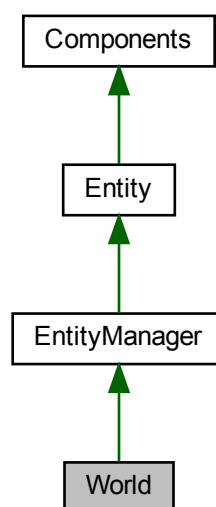
[World](#) class: [World](#) is a class that represents the world of the game.

```
#include <world.h>
```

Inheritance diagram for World:



Collaboration diagram for World:



Public Member Functions

- [World](#) ()=default
Default [World](#) constructor.
- [~World](#) () override=default
[World](#) destructor.
- void [createEntities](#) (std::map< std::string, std::pair< std::unique_ptr< [EntityManager](#) >, std::vector< std::string >>> &mapEntityManager)
[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

4.22.1 Detailed Description

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

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

4.22.2 Constructor & Destructor Documentation

4.22.2.1 World()

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

Default [World](#) constructor.

Parameters

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

Returns

void

4.22.2.2 ~World()

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

[World](#) destructor.

Parameters

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

Returns

void

4.22.3 Member Function Documentation

4.22.3.1 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.

4.22.3.2 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

4.22.3.3 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.

4.22.3.4 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.

4.22.3.5 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.

4.22.3.6 initWorld()

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

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

Parameters

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

Returns

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

4.22.3.7 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

- ~Components
 - Components, [9](#)
- ~DrawableComponent
 - DrawableComponent, [10](#)
- ~Entity
 - Entity, [14](#)
- ~EntityManager
 - EntityManager, [21](#)
- ~EventEngine
 - EventEngine, [27](#)
- ~GameEngine
 - GameEngine, [35](#)
- ~Rect
 - Rect< T >, [47](#)
- ~Sprite
 - Sprite, [53](#)
- ~Transform
 - Transform, [68](#)
- ~Vector2
 - Vector2< T >, [75](#)
- ~World
 - World, [79](#)
- ~toSFML
 - toSFML, [66](#)
- addComponent
 - Entity, [14](#)
- addDrawable
 - Entity, [15](#)
- addEntity
 - EntityManager, [21](#)
- addEntityManager
 - World, [79](#)
- addKeyPressed
 - EventEngine, [27](#)
- addMouseButtonPressed
 - EventEngine, [28](#)
- addMouseMoved
 - EventEngine, [28](#)
- addWorld
 - GameEngine, [36](#)
- applyDeferredSprite
 - Sprite, [53](#)
- Archetypes, [7](#)
- Audio, [7](#)
- Components, [7](#)
 - ~Components, [9](#)
 - Components, [8](#)
 - init, [9](#)
- contains
 - Rect< T >, [47](#)
- createEntities
 - World, [79](#)
- createSprite
 - Sprite, [54](#)
- draw
 - DrawableComponent, [11](#)
 - Sprite, [55](#)
- DrawableComponent, [10](#)
 - ~DrawableComponent, [10](#)
 - draw, [11](#)
- drawEntity
 - Entity, [15](#)
- Entity, [11](#)
 - ~Entity, [14](#)
 - addComponent, [14](#)
 - addDrawable, [15](#)
 - drawEntity, [15](#)
 - Entity, [13](#), [14](#)
 - getComponent, [16](#)
 - getComponentArrays, [16](#)
 - getComponentBitset, [16](#)
 - getComponentTypeID, [17](#)
 - getDrawableComponents, [17](#)
 - getName, [18](#)
 - initEntity, [18](#)
 - setName, [18](#)
- EntityManager, [19](#)
 - ~EntityManager, [21](#)
 - addEntity, [21](#)
 - EntityManager, [20](#)
 - getEntities, [21](#)
 - getEntity, [22](#)
 - getEntityMap, [22](#)
 - initEntityManager, [23](#)
- EntityManagerTest, [23](#)
- EntityTest, [25](#)
- EventEngine, [26](#)
 - ~EventEngine, [27](#)
 - addKeyPressed, [27](#)
 - addMouseButtonPressed, [28](#)
 - addMouseMoved, [28](#)
 - EventEngine, [27](#)
 - getEvent, [28](#)
 - getKeyPressedMap, [29](#)
 - getMouseButtonPressedMap, [29](#)
 - getMouseMovedMap, [29](#)

- init, 30
- eventGameEngine
 - GameEngine, 36
- EventTest, 30
- GameEngine, 31
 - ~GameEngine, 35
 - addWorld, 36
 - eventGameEngine, 36
 - GameEngine, 34, 35
 - getCurrentWorld, 36
 - getEventEngine, 37
 - getFilesTexture, 37
 - getMapTexture, 37
 - getWindow, 38
 - getWorld, 38
 - getWorldMap, 38
 - initialize, 39
 - initializeSprite, 39
 - initializeTexture, 39
 - initializeWorldMap, 40
 - isWindowOpen, 40
 - renderGameEngine, 40
 - run, 41
 - setCurrentWorld, 42
 - setWindow, 42
 - updateGameEngine, 42
- GameEngineTest, 43
- getBit
 - Sprite, 55
 - Transform, 69
- getComponent
 - Entity, 16
- getComponentArrays
 - Entity, 16
- getComponentBitset
 - Entity, 16
- getComponentTypeID
 - Entity, 17
- getCurrentWorld
 - GameEngine, 36
- getDrawableComponents
 - Entity, 17
- getEntities
 - EntityManager, 21
- getEntity
 - EntityManager, 22
- getEntityManager
 - World, 80
- getEntityManagerMap
 - World, 80
- getEntityMap
 - EntityManager, 22
- getEvent
 - EventEngine, 28
- getEventEngine
 - GameEngine, 37
- getFilesTexture
 - GameEngine, 37
- getHeight
 - Rect< T >, 48
- getKeyPressedMap
 - EventEngine, 29
- getLeft
 - Rect< T >, 48
- getMapTexture
 - GameEngine, 37
- getMouseButtonPressedMap
 - EventEngine, 29
- getMouseMovedMap
 - EventEngine, 29
- getName
 - Entity, 18
- getNameWorld
 - World, 80
- getPosition
 - Transform, 69
- getRect
 - Rect< T >, 49
- getRotation
 - Transform, 70
- getScale
 - Transform, 70
- getSprite
 - Sprite, 55
- getTexture
 - Sprite, 57
- getTop
 - Rect< T >, 49
- getTransformStruct
 - Transform, 70
- getVector2Struct
 - Vector2< T >, 75
- getWidth
 - Rect< T >, 49
- getWindow
 - GameEngine, 38
- getWorld
 - GameEngine, 38
- getWorldMap
 - GameEngine, 38
- getX
 - Vector2< T >, 75
- getY
 - Vector2< T >, 76
- init
 - Components, 9
 - EventEngine, 30
 - Transform, 71
- initEntity
 - Entity, 18
- initEntityManager
 - EntityManager, 23
- initialize
 - GameEngine, 39
- initializeSprite
 - GameEngine, 39

- initializeTexture
 - GameEngine, 39
- initializeWorldMap
 - GameEngine, 40
- initSprite
 - Sprite, 57
- initWorld
 - World, 81
- isTextureLoaded
 - Sprite, 57
- isWindowOpen
 - GameEngine, 40
- Rect
 - Rect< T >, 46
- Rect< T >, 45
 - ~Rect, 47
 - contains, 47
 - getHeight, 48
 - getLeft, 48
 - getRect, 49
 - getTop, 49
 - getWidth, 49
 - Rect, 46
- renderGameEngine
 - GameEngine, 40
- run
 - GameEngine, 41
- Script, 50
- setCurrentWorld
 - GameEngine, 42
- setDeferredSprite
 - Sprite, 58
- setName
 - Entity, 18
- setNameWorld
 - World, 81
- setPosition
 - Sprite, 58
- setRotation
 - Sprite, 59
- setScale
 - Sprite, 59, 60
- setSprite
 - Sprite, 60
- setTexture
 - Sprite, 61
- setTransform
 - Transform, 71
- setTransformPosition
 - Transform, 71
- setTransformRotation
 - Transform, 72
- setTransformScale
 - Transform, 72
- setTransformSprite
 - Sprite, 61, 62
- setWindow
 - GameEngine, 42
- Sprite, 50
 - ~Sprite, 53
 - applyDeferredSprite, 53
 - createSprite, 54
 - draw, 55
 - getBit, 55
 - getSprite, 55
 - getTexture, 57
 - initSprite, 57
 - isTextureLoaded, 57
 - setDeferredSprite, 58
 - setPosition, 58
 - setRotation, 59
 - setScale, 59, 60
 - setSprite, 60
 - setTexture, 61
 - setTransformSprite, 61, 62
 - Sprite, 52, 53
- SpriteTest, 62
- TestWorld, 63
- toSFML, 64
 - ~toSFML, 66
 - toSFML, 65
 - toSFMLRect, 66
- toSFMLRect
 - toSFML, 66
- Transform, 67
 - ~Transform, 68
 - getBit, 69
 - getPosition, 69
 - getRotation, 70
 - getScale, 70
 - getTransformStruct, 70
 - init, 71
 - setTransform, 71
 - setTransformPosition, 71
 - setTransformRotation, 72
 - setTransformScale, 72
 - Transform, 68
- TransformTest, 73
- updateGameEngine
 - GameEngine, 42
- Vector2
 - Vector2< T >, 74
- Vector2< T >, 73
 - ~Vector2, 75
 - getVector2Struct, 75
 - getX, 75
 - getY, 76
 - Vector2, 74
- World, 76
 - ~World, 79
 - addEntityManager, 79
 - createEntities, 79

getEntityManager, [80](#)
getEntityManagerMap, [80](#)
getNameWorld, [80](#)
initWorld, [81](#)
setNameWorld, [81](#)
World, [78](#)