

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()	9
4.3.2.2 ~Components()	10
4.3.3 Member Function Documentation	10
4.3.3.1 init()	10
4.3.3.2 update()	10
4.4 DrawableComponent Class Reference	12
4.4.1 Detailed Description	12
4.4.2 Constructor & Destructor Documentation	12
4.4.2.1 ~DrawableComponent()	12
4.4.3 Member Function Documentation	13
4.4.3.1 draw()	13
4.5 Entity Class Reference	13
4.5.1 Detailed Description	15
4.5.2 Constructor & Destructor Documentation	15
4.5.2.1 Entity() [1/2]	15
4.5.2.2 Entity() [2/2]	16
4.5.2.3 ~Entity()	16
4.5.3 Member Function Documentation	16
4.5.3.1 addComponent()	16
4.5.3.2 addDrawable()	17
4.5.3.3 drawEntity()	17
4.5.3.4 getComponent()	18
4.5.3.5 getComponentArrays()	18
4.5.3.6 getComponentBitset()	18
4.5.3.7 getComponentTypeID()	19
4.5.3.8 getDrawableComponents()	19
4.5.3.9 getName()	19

4.5.3.10 initEntity()	20
4.5.3.11 setName()	20
4.6 EntityManager Class Reference	21
4.6.1 Constructor & Destructor Documentation	22
4.6.1.1 EntityManager()	22
4.6.1.2 ~EntityManager()	22
4.6.2 Member Function Documentation	23
4.6.2.1 addEntity()	23
4.6.2.2 getEntities()	23
4.6.2.3 getEntity()	23
4.6.2.4 getEntityMap()	24
4.6.2.5 initEntityManager()	24
4.7 EntityManagerTest Class Reference	25
4.8 EntityTest Class Reference	26
4.9 EventEngine Class Reference	27
4.9.1 Detailed Description	27
4.9.2 Constructor & Destructor Documentation	27
4.9.2.1 EventEngine()	27
4.9.2.2 ~EventEngine()	28
4.9.3 Member Function Documentation	28
4.9.3.1 addKeyPressed()	28
4.9.3.2 getEvent()	29
4.9.3.3 getKeyPressedMap()	29
4.9.3.4 init()	29
4.10 EventTest Class Reference	30
4.11 GameEngine Class Reference	30
4.11.1 Detailed Description	33
4.11.2 Constructor & Destructor Documentation	33
4.11.2.1 GameEngine() [1/2]	33
4.11.2.2 GameEngine() [2/2]	34
4.11.2.3 ~GameEngine()	34
4.11.3 Member Function Documentation	35
4.11.3.1 addWorld()	35
4.11.3.2 eventGameEngine()	35
4.11.3.3 getCurrentWorld()	35
4.11.3.4 getEventEngine()	36
4.11.3.5 getFilesTexture()	36
4.11.3.6 getMapTexture()	36
4.11.3.7 getWindow()	37
4.11.3.8 getWorld()	37
4.11.3.9 getWorldMap()	37
4.11.3.10 initialize()	38

4.11.3.11 initializeSprite()	38
4.11.3.12 initializeTexture()	39
4.11.3.13 initializeWorldMap()	39
4.11.3.14 isWindowOpen()	39
4.11.3.15 renderGameEngine()	40
4.11.3.16 run() [1/2]	40
4.11.3.17 run() [2/2]	40
4.11.3.18 setCurrentWorld()	41
4.11.3.19 setWindow()	41
4.11.3.20 updateGameEngine()	41
4.12 GameEngineTest Class Reference	42
4.13 Sprite Class Reference	44
4.13.1 Detailed Description	45
4.13.2 Constructor & Destructor Documentation	45
4.13.2.1 Sprite() [1/2]	45
4.13.2.2 Sprite() [2/2]	46
4.13.2.3 ~Sprite()	46
4.13.3 Member Function Documentation	46
4.13.3.1 applyDeferredSprite()	46
4.13.3.2 createSprite() [1/3]	47
4.13.3.3 createSprite() [2/3]	47
4.13.3.4 createSprite() [3/3]	47
4.13.3.5 draw()	48
4.13.3.6 getBit()	48
4.13.3.7 getSprite()	48
4.13.3.8 getTexture()	50
4.13.3.9 initSprite()	50
4.13.3.10 isTextureLoaded()	50
4.13.3.11 setDeferredSprite()	51
4.13.3.12 setSprite() [1/2]	51
4.13.3.13 setSprite() [2/2]	51
4.13.3.14 setTexture()	52
4.14 SpriteTest Class Reference	53
4.15 TestWorld Class Reference	54
4.16 Transform Class Reference	55
4.16.1 Detailed Description	56
4.16.2 Constructor & Destructor Documentation	56
4.16.2.1 Transform() [1/2]	56
4.16.2.2 Transform() [2/2]	56
4.16.2.3 ~Transform()	57
4.16.3 Member Function Documentation	57
4.16.3.1 getBit()	57

4.16.3.2 getPositionVector()	57
4.16.3.3 getRotationVector()	58
4.16.3.4 getScaleVector()	58
4.16.3.5 init()	58
4.16.3.6 setTransform()	59
4.17 TransformTest Class Reference	59
4.18 World Class Reference	60
4.18.1 Detailed Description	62
4.18.2 Constructor & Destructor Documentation	62
4.18.2.1 World()	62
4.18.2.2 ~World()	63
4.18.3 Member Function Documentation	63
4.18.3.1 addEntityManager()	63
4.18.3.2 createEntities()	63
4.18.3.3 getEntityManager()	64
4.18.3.4 getEntityManagerMap()	64
4.18.3.5 getNameWorld()	64
4.18.3.6 initWorld()	65
4.18.3.7 setNameWorld()	65
Index	67

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
```


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	13
EntityManager	21
World	60
GameEngine	30
Sprite	44
Transform	55
DrawableComponent	12
Sprite	44
EventEngine	27
GameEngine	30
testing::Test	
EntityManagerTest	25
EntityTest	26
EventTest	30
GameEngineTest	42
SpriteTest	53
TestWorld	54
TransformTest	59

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	12
Entity	
Entity class: Entity is a class that represents an entity in the game	13
EntityManager	21
EntityManagerTest	25
EntityTest	26
EventEngine	
EventEngine class: EventEngine is a class that represents the event engine of the game	27
EventTest	30
GameEngine	
GameEngine class: GameEngine is a class that represents the game engine	30
GameEngineTest	42
Sprite	
Sprite class: Sprite is a class that represents the rendering properties of a Component	44
SpriteTest	53
TestWorld	54
Transform	
Transform class: Transform is a class that represents the transform of a Component	55
TransformTest	59
World	
World class: World is a class that represents the world of the game	60

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:

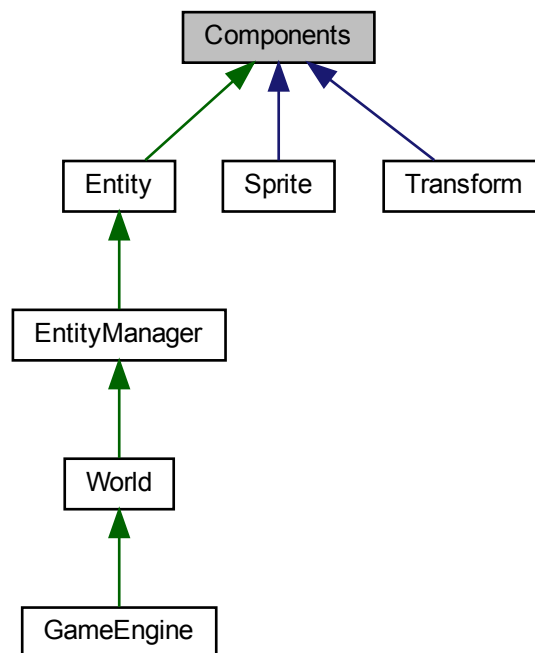
- `src/Components/all_components/include/Audio.h`

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 ()`
`update()`: Update the component

4.3.1 Detailed Description

`Components` class: `Components` is a class that represents a component in the game.

`Components` are the building blocks of the game. They are attached to entities and define their behavior.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 Components()

`Components::Components () [default]`

Default [Components](#) constructor.

Parameters

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

Returns

void

4.3.2.2 ~Components()

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

[Components](#) destructor.

Parameters

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

Returns

void

4.3.3 Member Function Documentation**4.3.3.1 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

4.3.3.2 update()

```
virtual void Components::update ( ) [inline], [virtual]
```


`update()`: Update the component

Parameters

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

Returns

void

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

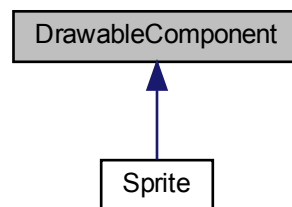
- src/Components/include/Components.h

4.4 DrawableComponent Class Reference

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

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

Inheritance diagram for DrawableComponent:



Public Member Functions

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

4.4.1 Detailed Description

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

DrawableComponents are components that can be drawn on the screen.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 ~DrawableComponent()

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

Default [DrawableComponent](#) constructor.

Parameters

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

Returns

void

4.4.3 Member Function Documentation

4.4.3.1 draw()

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

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

Parameters

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

Returns

void

Implemented in [Sprite](#).

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

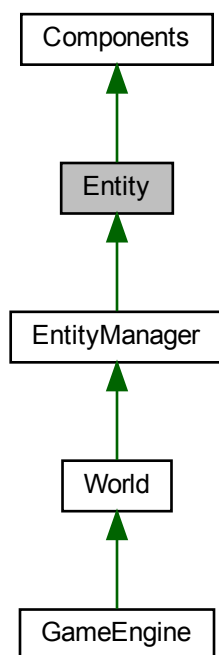
- `src/Components/include/DrawableComponent.h`

4.5 Entity Class Reference

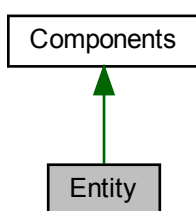
[Entity](#) class: [Entity](#) is a class that represents an entity in the game.

```
#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 `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

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

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

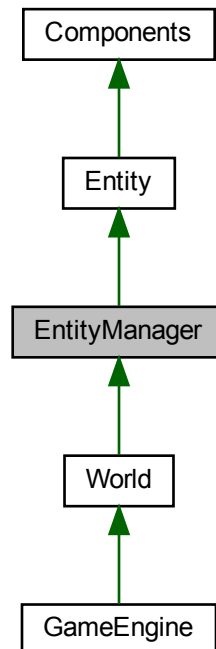
void

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

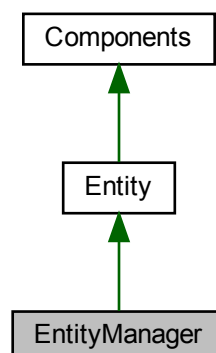
- src/Entity/include/entity.h
- src/Entity/entity.cpp

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

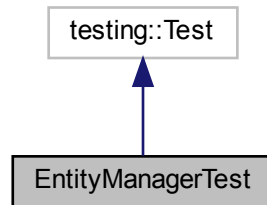
bool: true if the [EntityManager](#) is initialized, false otherwise.

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

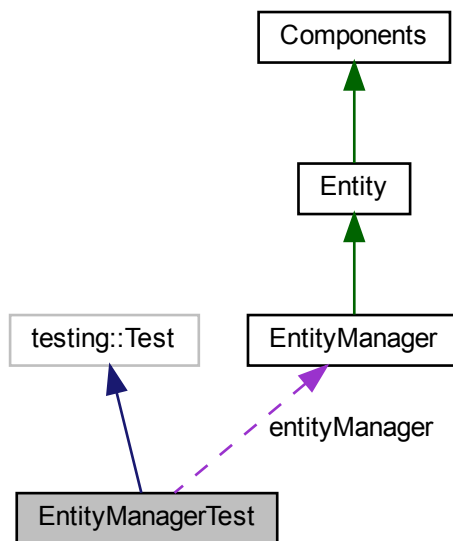
- src/Entity/include/entityManager.h
- src/Entity/entityManager.cpp

4.7 EntityManagerTest Class Reference

Inheritance diagram for EntityManagerTest:



Collaboration diagram for EntityManagerTest:



Protected Member Functions

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

Protected Attributes

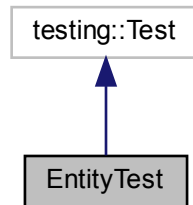
- [EntityManager](#) entityManager {}

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

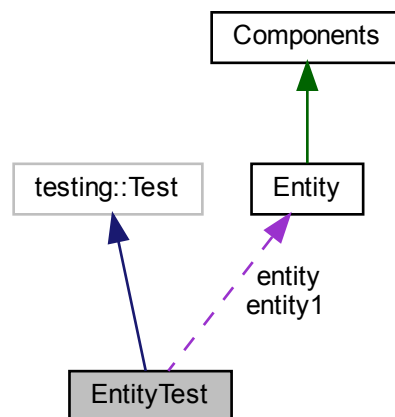
- tests/Entity/TestEntityManager.cpp

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:

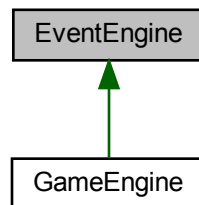
- tests/Entity/TestEntity.cpp

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.
- `std::map< sf::Keyboard::Key, std::function< void()> > & getKeyPressedMap ()`
`getKeyPressedMap()`: Get the map of the key pressed.

4.9.1 Detailed Description

`EventEngine` class: `EventEngine` is a class that represents the event engine of the game.

The `EventEngine` class manages the events of the game.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 EventEngine()

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

Default `EventEngine` constructor.

Parameters

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

Returns

void

4.9.2.2 ~EventEngine()

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

[EventEngine](#) destructor.

Parameters

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

Returns

void

4.9.3 Member Function Documentation**4.9.3.1 addKeyPressed()**

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

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

Parameters

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

Returns

void

4.9.3.2 `getEvent()`

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

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

Parameters

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

Returns

sf::Event: The SFML Event.

4.9.3.3 `getKeyPressedMap()`

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

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

Parameters

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

Returns

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

4.9.3.4 `init()`

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

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

Parameters

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

Returns

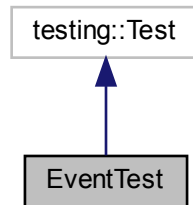
bool: True if the [EventEngine](#) is initialized, false otherwise.

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

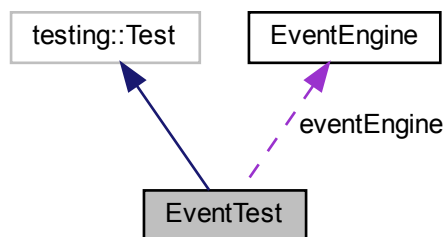
- src/Event/include/eventEngine.h
- src/Event/eventEngine.cpp

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:

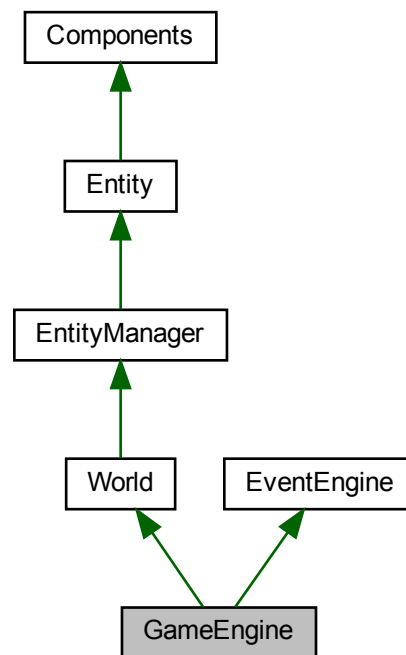
- `tests/Event/TestEvent.cpp`

4.11 GameEngine Class Reference

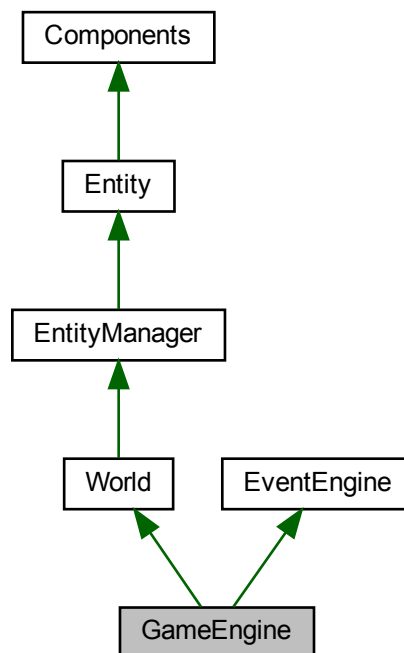
[GameEngine](#) class: [GameEngine](#) is a class that represents the game engine.

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

Inheritance diagram for GameEngine:



Collaboration diagram for GameEngine:



Public Member Functions

- [GameEngine](#) ()=default
Default [GameEngine](#) constructor.
- [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]
```

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

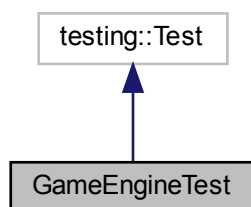
void

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

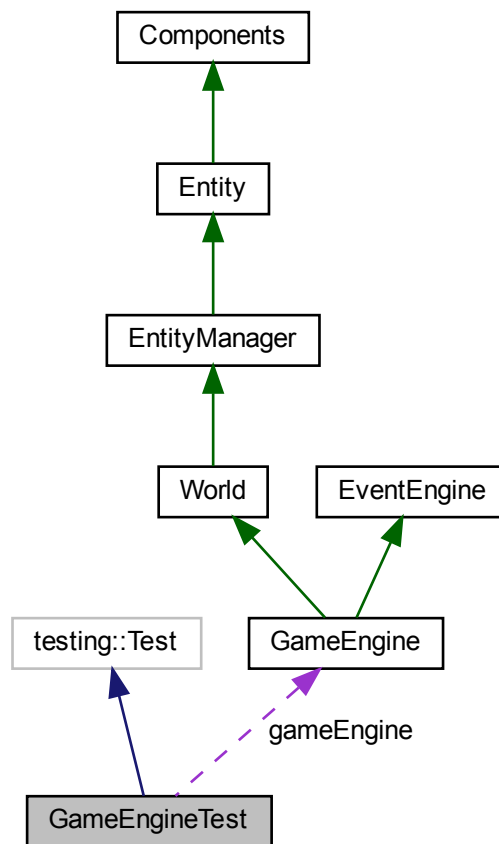
- src/GameEngine/include/gameEngine.h
- src/GameEngine/gameEngine.cpp

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:

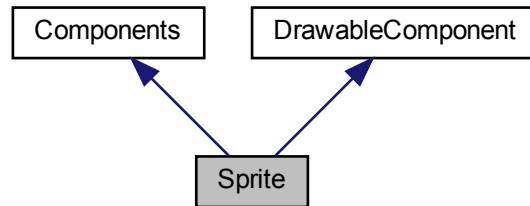
- tests/GameEngine/TestGameEngine.cpp

4.13 Sprite Class Reference

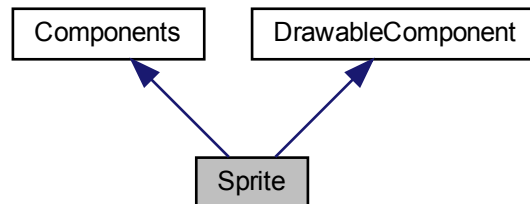
[Sprite](#) class: [Sprite](#) is a class that represents the rendering properties of a Component.

```
#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.

4.13.1 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.

4.13.2 Constructor & Destructor Documentation

4.13.2.1 *Sprite*() [1/2]

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

Default *Sprite* constructor.

Parameters

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

Returns

void

4.13.2.2 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

4.13.2.3 ~Sprite()

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

[Sprite](#) destructor.

Parameters

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

Returns

void

4.13.3 Member Function Documentation**4.13.3.1 applyDeferredSprite()**

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

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

Parameters

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

Returns

void

4.13.3.2 createSprite() [1/3]

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

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

Parameters

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

Returns

void

4.13.3.3 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

4.13.3.4 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

4.13.3.5 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](#).

4.13.3.6 getBit()

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

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

Parameters

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

Returns

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

4.13.3.7 getSprite()

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

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

Parameters

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

Returns

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

4.13.3.8 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

4.13.3.9 initSprite()

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

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

Parameters

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

Returns

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

4.13.3.10 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.

4.13.3.11 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

4.13.3.12 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 Sprite for rendering
---------------	---

Returns

void

4.13.3.13 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

4.13.3.14 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

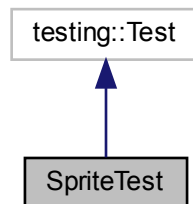
void

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

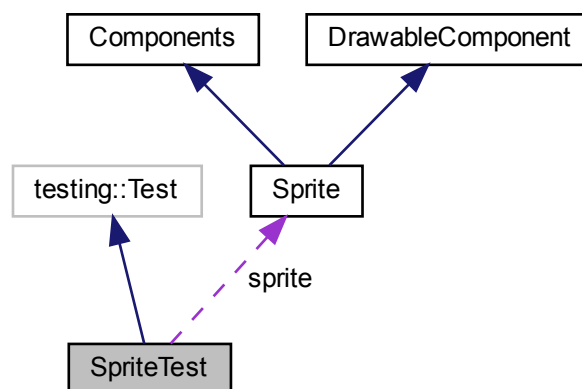
- src/Components/all_components/include/Sprite.h
- src/Components/all_components/Sprite.cpp

4.14 SpriteTest Class Reference

Inheritance diagram for SpriteTest:



Collaboration diagram for SpriteTest:



Protected Attributes

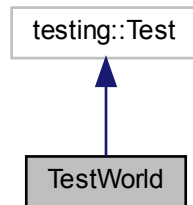
- [Sprite](#) `sprite`

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

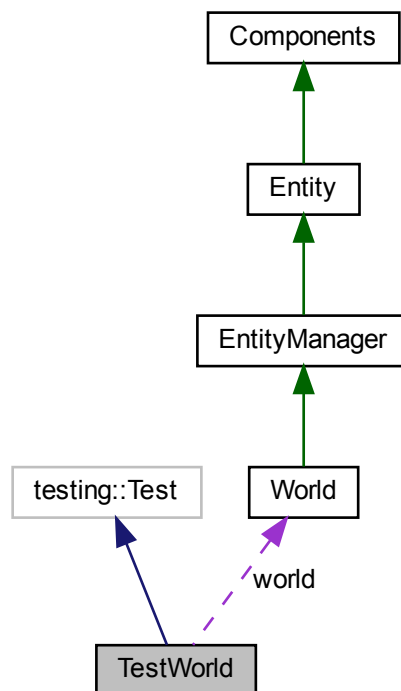
- tests/Components/all_components/TestSprite.cpp

4.15 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:

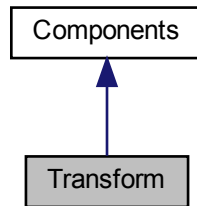
- tests/World/TestWorld.cpp

4.16 Transform Class Reference

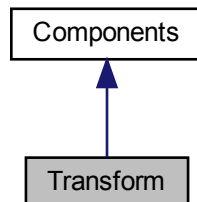
Transform class: **Transform** is a class that represents the transform of a Component.

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

Inheritance diagram for Transform:



Collaboration diagram for Transform:



Public Member Functions

- **Transform** ()=default
Default **Transform** constructor.
- bool **init** () const
init(): Initialize the component
- **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

4.16.1 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.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 Transform() [1/2]

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

Default [Transform](#) constructor.

Parameters

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

Returns

void

4.16.2.2 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

4.16.2.3 ~Transform()

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

[Transform](#) destructor.

Parameters

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

Returns

void

4.16.3 Member Function Documentation**4.16.3.1 getBit()**

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

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

Parameters

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

Returns

int: bitmask of the component

4.16.3.2 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

4.16.3.3 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

4.16.3.4 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

4.16.3.5 init()

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

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

Parameters

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

Returns

bool: true if the component is initialized, false otherwise

4.16.3.6 setTransform()

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

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

Parameters

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

Returns

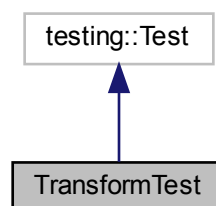
void

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

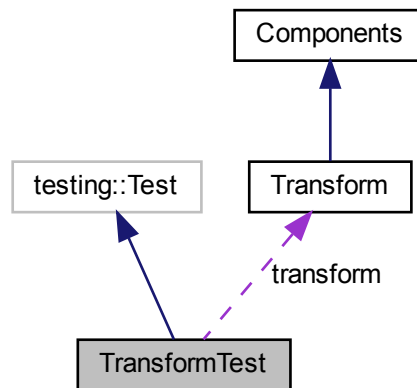
- src/Components/all_components/include/Transform.h
- src/Components/all_components/Transform.cpp

4.17 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:

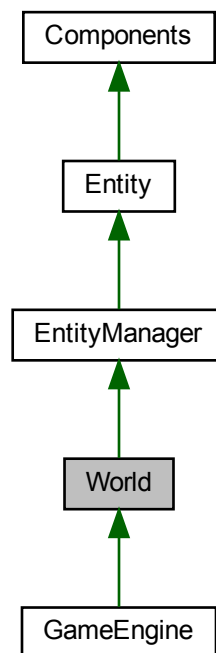
- tests/Components/all_components/TestTransform.cpp

4.18 World Class Reference

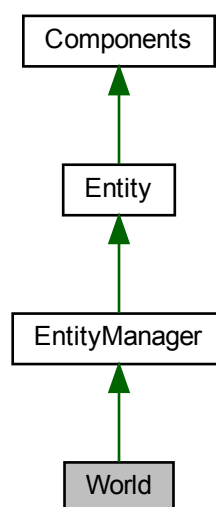
[World](#) class: [World](#) is a class that represents the world of the game.

```
#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, 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

4.18.1 Detailed Description

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

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

4.18.2 Constructor & Destructor Documentation

4.18.2.1 World()

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

Default [World](#) constructor.

Parameters

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

Returns

void

4.18.2.2 ~World()

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

[World](#) destructor.

Parameters

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

Returns

void

4.18.3 Member Function Documentation

4.18.3.1 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.

4.18.3.2 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

4.18.3.3 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.

4.18.3.4 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.

4.18.3.5 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.

4.18.3.6 initWorld()

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

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

Parameters

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

Returns

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

4.18.3.7 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, [10](#)
- ~DrawableComponent
 - DrawableComponent, [12](#)
- ~Entity
 - Entity, [16](#)
- ~EntityManager
 - EntityManager, [22](#)
- ~EventEngine
 - EventEngine, [28](#)
- ~GameEngine
 - GameEngine, [34](#)
- ~Sprite
 - Sprite, [46](#)
- ~Transform
 - Transform, [57](#)
- ~World
 - World, [63](#)
- addComponent
 - Entity, [16](#)
- addDrawable
 - Entity, [17](#)
- addEntity
 - EntityManager, [23](#)
- addEntityManager
 - World, [63](#)
- addKeyPressed
 - EventEngine, [28](#)
- addWorld
 - GameEngine, [35](#)
- applyDeferredSprite
 - Sprite, [46](#)
- Archetypes, [7](#)
- Audio, [7](#)
- Components, [7](#)
 - ~Components, [10](#)
 - Components, [8](#)
 - init, [10](#)
 - update, [10](#)
- createEntities
 - World, [63](#)
- createSprite
 - Sprite, [47](#)
- draw
 - DrawableComponent, [13](#)
 - Sprite, [48](#)
- DrawableComponent, [12](#)
 - ~DrawableComponent, [12](#)
 - draw, [13](#)
- drawEntity
 - Entity, [17](#)
- Entity, [13](#)
 - ~Entity, [16](#)
 - addComponent, [16](#)
 - addDrawable, [17](#)
 - drawEntity, [17](#)
 - Entity, [15](#), [16](#)
 - getComponent, [17](#)
 - getComponentArrays, [18](#)
 - getComponentBitset, [18](#)
 - getComponentTypeID, [19](#)
 - getDrawableComponents, [19](#)
 - getName, [19](#)
 - initEntity, [20](#)
 - setName, [20](#)
- EntityManager, [21](#)
 - ~EntityManager, [22](#)
 - addEntity, [23](#)
 - EntityManager, [22](#)
 - getEntities, [23](#)
 - getEntity, [23](#)
 - getEntityMap, [24](#)
 - initEntityManager, [24](#)
- EntityManagerTest, [25](#)
- EntityTest, [26](#)
- EventEngine, [27](#)
 - ~EventEngine, [28](#)
 - addKeyPressed, [28](#)
 - EventEngine, [27](#)
 - getEvent, [28](#)
 - getKeyPressedMap, [29](#)
 - init, [29](#)
- eventGameEngine
 - GameEngine, [35](#)
- EventTest, [30](#)
- GameEngine, [30](#)
 - ~GameEngine, [34](#)
 - addWorld, [35](#)
 - eventGameEngine, [35](#)
 - GameEngine, [33](#), [34](#)
 - getCurrentWorld, [35](#)
 - getEventEngine, [36](#)
 - getFilesTexture, [36](#)
 - getMapTexture, [36](#)
 - getWindow, [37](#)

- getWorld, 37
- getWorldMap, 37
- initialize, 38
- initializeSprite, 38
- initializeTexture, 38
- initializeWorldMap, 39
- isWindowOpen, 39
- renderGameEngine, 39
- run, 40
- setCurrentWorld, 41
- setWindow, 41
- updateGameEngine, 41
- GameEngineTest, 42
- getBit
 - Sprite, 48
 - Transform, 57
- getComponent
 - Entity, 17
- getComponentArrays
 - Entity, 18
- getComponentBitset
 - Entity, 18
- getComponentTypeID
 - Entity, 19
- getCurrentWorld
 - GameEngine, 35
- getDrawableComponents
 - Entity, 19
- getEntities
 - EntityManager, 23
- getEntity
 - EntityManager, 23
- getEntityManager
 - World, 64
- getEntityManagerMap
 - World, 64
- getEntityMap
 - EntityManager, 24
- getEvent
 - EventEngine, 28
- getEventEngine
 - GameEngine, 36
- getFilesTexture
 - GameEngine, 36
- getKeyPressedMap
 - EventEngine, 29
- getMapTexture
 - GameEngine, 36
- getName
 - Entity, 19
- getNameWorld
 - World, 64
- getPositionVector
 - Transform, 57
- getRotationVector
 - Transform, 58
- getScaleVector
 - Transform, 58
- getSprite
 - Sprite, 48
- getTexture
 - Sprite, 50
- getWindow
 - GameEngine, 37
- getWorld
 - GameEngine, 37
- getWorldMap
 - GameEngine, 37
- init
 - Components, 10
 - EventEngine, 29
 - Transform, 58
- initEntity
 - Entity, 20
- initEntityManager
 - EntityManager, 24
- initialize
 - GameEngine, 38
- initializeSprite
 - GameEngine, 38
- initializeTexture
 - GameEngine, 38
- initializeWorldMap
 - GameEngine, 39
- initSprite
 - Sprite, 50
- initWorld
 - World, 65
- isTextureLoaded
 - Sprite, 50
- isWindowOpen
 - GameEngine, 39
- renderGameEngine
 - GameEngine, 39
- run
 - GameEngine, 40
- setCurrentWorld
 - GameEngine, 41
- setDeferredSprite
 - Sprite, 51
- setName
 - Entity, 20
- setNameWorld
 - World, 65
- setSprite
 - Sprite, 51
- setTexture
 - Sprite, 52
- setTransform
 - Transform, 59
- setWindow
 - GameEngine, 41
- Sprite, 44
 - ~Sprite, 46

- [applyDeferredSprite](#), 46
 - [createSprite](#), 47
 - [draw](#), 48
 - [getBit](#), 48
 - [getSprite](#), 48
 - [getTexture](#), 50
 - [initSprite](#), 50
 - [isTextureLoaded](#), 50
 - [setDeferredSprite](#), 51
 - [setSprite](#), 51
 - [setTexture](#), 52
 - [Sprite](#), 45, 46
- [SpriteTest](#), 53
- [TestWorld](#), 54
- [Transform](#), 55
 - [~Transform](#), 57
 - [getBit](#), 57
 - [getPositionVector](#), 57
 - [getRotationVector](#), 58
 - [getScaleVector](#), 58
 - [init](#), 58
 - [setTransform](#), 59
 - [Transform](#), 56
- [TransformTest](#), 59
- [update](#)
 - [Components](#), 10
- [updateGameEngine](#)
 - [GameEngine](#), 41
- [World](#), 60
 - [~World](#), 63
 - [addEntityManager](#), 63
 - [createEntities](#), 63
 - [getEntityManager](#), 64
 - [getEntityManagerMap](#), 64
 - [getNameWorld](#), 64
 - [initWorld](#), 65
 - [setNameWorld](#), 65
 - [World](#), 62