Projecto 1: ECS Motor de Videojuegos (Obelisk Defender).

Generated by Doxygen 1.9.8

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

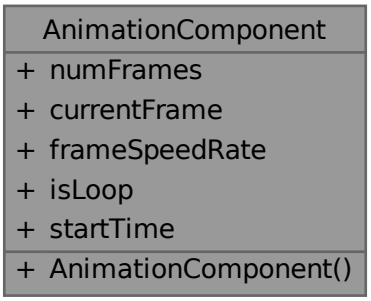
Class Documentation

4.1 AnimationComponent Struct Reference

Holds data for sprite-based animations, including frame count, speed, looping, and timing.

`#include <AnimationComponent.hpp>`

Collaboration diagram for AnimationComponent:



Public Member Functions

- AnimationComponent (int numFrames=1, int frameSpeedRate=1, bool isLoop=true)

  Constructs a new AnimationComponent.

Public Attributes

- int numFrames
- int currentFrame
- int frameSpeedRate
- bool isLoop
- int startTime

4.1.1 Detailed Description

Holds data for sprite-based animations, including frame count, speed, looping, and timing.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 AnimationComponent()

```
AnimationComponent::AnimationComponent (
            int numFrames = 1,
            int frameSpeedRate = 1,
            bool isLoop = true )  [inline]
```

Constructs a new AnimationComponent.

Parameters

numFrames	Number of frames in the animation. Default is 1.
frameSpeedRate	Time per frame in milliseconds. Default is 1.
isLoop	Whether the animation loops. Default is true.

4.1.3 Member Data Documentation

4.1.3.1 currentFrame

```
int AnimationComponent::currentFrame
```

The current frame being displayed.

4.1.3.2 frameSpeedRate

```
int AnimationComponent::frameSpeedRate
```

The speed rate of the animation (in ms per frame).

4.1.3.3 isLoop

```
bool AnimationComponent::isLoop
```

Whether the animation should loop when it ends.

4.1.3.4 numFrames

```
int AnimationComponent::numFrames
```

Total number of frames in the animation.

4.1.3.5 startTime

```
int AnimationComponent::startTime
```

The time when the animation started (in ms since SDL initialization).

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

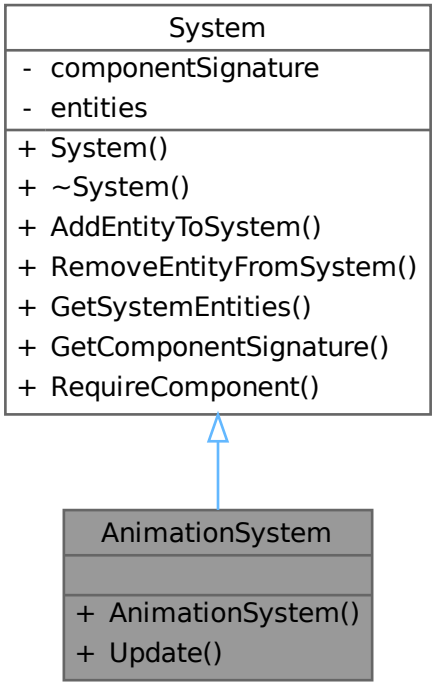
- src/Components/AnimationComponent.hpp

4.2 AnimationSystem Class Reference

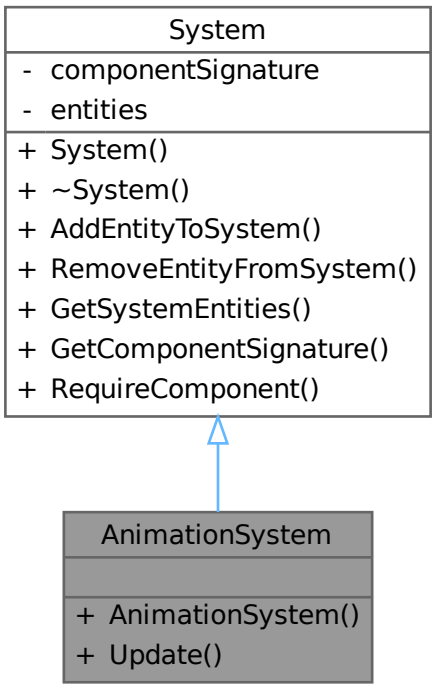
System responsible for updating the animation frames of entities.

```
#include <AnimationSystem.hpp>
```

Inheritance diagram for AnimationSystem:



Collaboration diagram for AnimationSystem:



Public Member Functions

- AnimationSystem ()

  Constructs the AnimationSystem and sets required components.
- void Update ()

  Updates all entities in the system by advancing their animation frames.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.2.1 Detailed Description

System responsible for updating the animation frames of entities.

This system requires entities to have both AnimationComponent and SpriteComponent. It updates the current animation frame based on time and adjusts the source rectangle of the sprite accordingly.

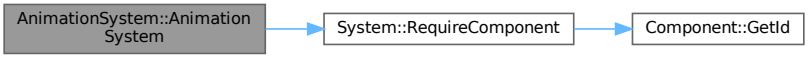
4.2.2 Constructor & Destructor Documentation

4.2.2.1 AnimationSystem()

```
AnimationSystem::AnimationSystem ( )  [inline]
```

Constructs the AnimationSystem and sets required components.

Here is the call graph for this function:



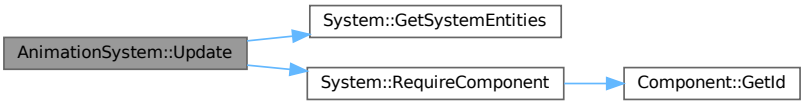
4.2.3 Member Function Documentation

4.2.3.1 Update()

```
void AnimationSystem::Update ( )  [inline]
```

Updates all entities in the system by advancing their animation frames.

Calculates the current animation frame based on the elapsed time, the animation frame speed rate, and the total number of frames. Updates the sprite's source rectangle x position to reflect the current frame. Here is the call graph for this function:



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

- src/Systems/AnimationSystem.hpp

4.3 AssetManager Class Reference

Manages textures, fonts, and music assets using SDL2.

`#include <AssetManager.hpp>`

Collaboration diagram for AssetManager:

Public Member Functions

- AssetManager ()

  Constructs a new AssetManager instance.
- ~AssetManager ()

  Destroys the AssetManager and releases all loaded assets.
- void ClearAssets ()

  Frees all textures, fonts, and music.
- void AddTexture (SDL_Renderer *renderer, const std::string &textureId, const std::string &filePath)

  Loads and stores a texture from file.
- SDL_Texture * GetTexture (const std::string &textureId)

  Retrieves a previously loaded texture.
- void AddFont (const std::string &fontId, const std::string &filePath, int fontSize)

  Loads and stores a font from file.
- TTF_Font * GetFont (const std::string &fontId)

Retrieves a previously loaded font.

- void LoadMusic (const std::string &musicId, const std::string &filePath)

  Loads a music track from file.
- void PlayMusic (const std::string &musicId, int loops=-1)

  Plays a loaded music track.
- void StopMusic ()

  Stops the currently playing music.
- void ResumeMusic ()

  Resumes paused music playback.
- void PauseMusic ()

  Pauses the currently playing music.
- void ClearMusic ()

  Frees all loaded music assets.

Private Attributes

- std::map< std::string, SDL_Texture * > textures

  Map of texture IDs to SDL_Texture pointers.
- std::map< std::string, TTF_Font * > fonts

  Map of font IDs to TTF_Font pointers.
- std::map< std::string, Mix_Music * > musics

  Map of music IDs to Mix_Music pointers.
- Mix_Music * currentMusic = nullptr

  Currently playing music.

4.3.1 Detailed Description

Manages textures, fonts, and music assets using SDL2.

This class handles loading, storing, retrieving, and clearing of SDL_Texture, TTF_Font, and Mix_Music assets. It centralizes asset management to avoid redundant loads and ensures proper cleanup.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 AssetManager()

```
AssetManager::AssetManager ( )
```

Constructs a new AssetManager instance.

4.3.2.2 ~AssetManager()

```
AssetManager::~AssetManager ( )
```

Destroys the AssetManager and releases all loaded assets.

4.3.3 Member Function Documentation

4.3.3.1 AddFont()

```
void AssetManager::AddFont (
            const std::string & fontId,
            const std::string & filePath,
            int fontSize )
```

Loads and stores a font from file.

Parameters

fontId	The unique ID to associate with the font.
filePath	The path to the font file.
fontSize	The font size to use.

4.3.3.2 AddTexture()

```
void AssetManager::AddTexture (
            SDL_Renderer * renderer,
            const std::string & textureId,
            const std::string & filePath )
```

Loads and stores a texture from file.

Parameters

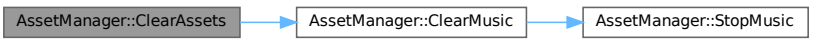
renderer	The SDL_Renderer used to create the texture.
textureId	The unique ID to associate with the texture.
filePath	The path to the texture file.

4.3.3.3 ClearAssets()

```
void AssetManager::ClearAssets ( )
```

Frees all textures, fonts, and music.

Here is the call graph for this function:



4.3.3.4 ClearMusic()

```
void AssetManager::ClearMusic ( )
```

Frees all loaded music assets.

Here is the call graph for this function:

Here is the caller graph for this function:

4.3.3.5 GetFont()

```
TTF_Font * AssetManager::GetFont (
            const std::string & fontId )
```

Retrieves a previously loaded font.

Parameters

fontId	The ID of the font to retrieve.

Returns

TTF_Font* Pointer to the font, or nullptr if not found.

4.3.3.6 GetTexture()

```
SDL_Texture * AssetManager::GetTexture (
            const std::string & textureId )
```

Retrieves a previously loaded texture.

Parameters

textureId	The ID of the texture to retrieve.

Returns

SDL_Texture* Pointer to the texture, or nullptr if not found.

4.3.3.7 LoadMusic()

```
void AssetManager::LoadMusic (
            const std::string & musicId,
            const std::string & filePath )
```

Loads a music track from file.

Parameters

musicId	The unique ID to associate with the music.
filePath	The path to the music file.

4.3.3.8 PauseMusic()

```
void AssetManager::PauseMusic ( )
```

Pauses the currently playing music.

4.3.3.9 PlayMusic()

```
void AssetManager::PlayMusic (
            const std::string & musicId,
            int loops = -1 )
```

Plays a loaded music track.

Parameters

musicId	The ID of the music to play.
loops	Number of times the music should loop. Default is -1 (infinite).

4.3.3.10 ResumeMusic()

```
void AssetManager::ResumeMusic ( )
```

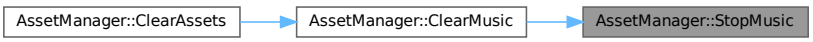
Resumes paused music playback.

4.3.3.11 StopMusic()

```
void AssetManager::StopMusic ( )
```

Stops the currently playing music.

Here is the caller graph for this function:



4.3.4 Member Data Documentation

4.3.4.1 currentMusic

```
Mix_Music* AssetManager::currentMusic = nullptr  [private]
```

Currently playing music.

4.3.4.2 fonts

```
std::map<std::string, TTF_Font*> AssetManager::fonts  [private]
```

Map of font IDs to TTF_Font pointers.

4.3.4.3 musics

```
std::map<std::string, Mix_Music*> AssetManager::musics  [private]
```

Map of music IDs to Mix_Music pointers.

4.3.4.4 textures

```
std::map<std::string, SDL_Texture*> AssetManager::textures  [private]
```

Map of texture IDs to SDL_Texture pointers.

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

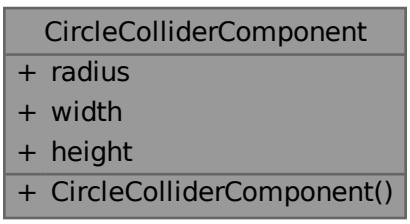
- src/AssetManager/AssetManager.hpp
- src/AssetManager/AssetManager.cpp

4.4 CircleColliderComponent Struct Reference

Component that defines the properties of a circular collider.

```
#include <CircleColliderComponent.hpp>
```

Collaboration diagram for CircleColliderComponent:



Public Member Functions

- CircleColliderComponent (double radius=0, double width=0, double height=0)

  Constructs a new CircleColliderComponent.

Public Attributes

- double radius
- double width
- double height

4.4.1 Detailed Description

Component that defines the properties of a circular collider.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 CircleColliderComponent()

```
CircleColliderComponent::CircleColliderComponent (
            double radius = 0,
            double width = 0,
            double height = 0 )  [inline]
```

Constructs a new CircleColliderComponent.

Parameters

radius	Radius of the circular collider. Default is 0.
width	Width of the bounding box. Default is 0.
height	Height of the bounding box. Default is 0.

4.4.3 Member Data Documentation

4.4.3.1 height

```
double CircleColliderComponent::height
```

The height of the bounding box (optional, for non-perfect circles).

4.4.3.2 radius

```
double CircleColliderComponent::radius
```

The radius of the circular collider.

4.4.3.3 width

```
double CircleColliderComponent::width
```

The width of the bounding box (optional, for non-perfect circles).

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

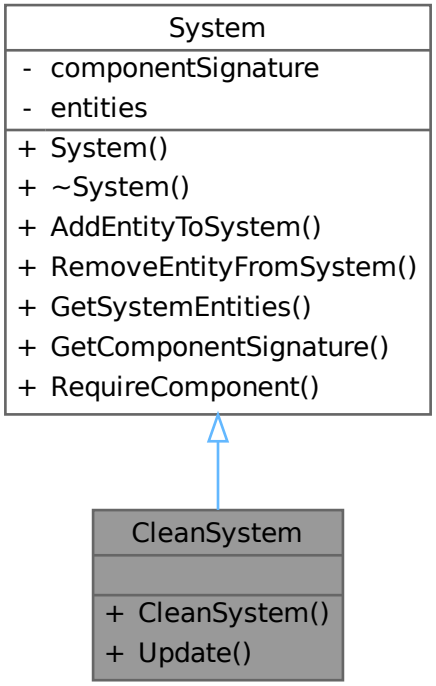
- src/Components/CircleColliderComponent.hpp

4.5 CleanSystem Class Reference

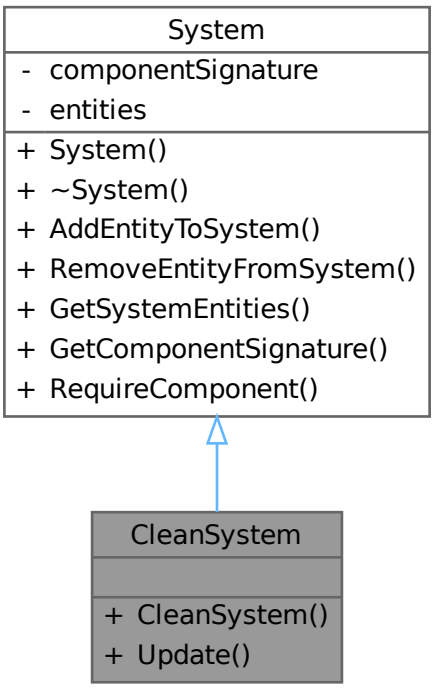
System responsible for removing entities that move outside defined bounds.

```
#include <CleanSystem.hpp>
```

Inheritance diagram for CleanSystem:



Collaboration diagram for CleanSystem:



Public Member Functions

- CleanSystem ()

  Constructs the CleanSystem and sets required components.
- void Update ()

  Updates all entities by removing those outside the horizontal bounds.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.5.1 Detailed Description

System responsible for removing entities that move outside defined bounds.

This system requires entities to have both RigidBodyComponent and TransformComponent. It checks the x position of the entity's transform, and if it is outside the range [-400, 2000], it kills the entity and logs the removal.

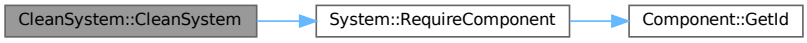
4.5.2 Constructor & Destructor Documentation

4.5.2.1 CleanSystem()

```
CleanSystem::CleanSystem ( )  [inline]
```

Constructs the CleanSystem and sets required components.

Here is the call graph for this function:



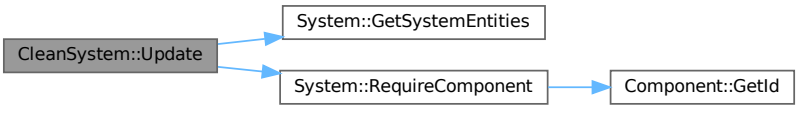
4.5.3 Member Function Documentation

4.5.3.1 Update()

```
void CleanSystem::Update ( )  [inline]
```

Updates all entities by removing those outside the horizontal bounds.

Iterates through system entities and kills those whose x position is greater than 2000 or less than -400, printing a removal message. Here is the call graph for this function:



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

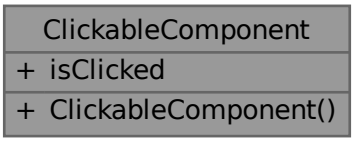
- src/Systems/CleanSystem.hpp

4.6 ClickableComponent Struct Reference

Component that indicates if an entity has been clicked.

`#include <ClickableComponent.hpp>`

Collaboration diagram for ClickableComponent:



Public Member Functions

- ClickableComponent ()

  Constructs a new ClickableComponent with isClicked initialized to false.

Public Attributes

- bool isClicked

4.6.1 Detailed Description

Component that indicates if an entity has been clicked.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 ClickableComponent()

`ClickableComponent::ClickableComponent ( )  [inline]`

Constructs a new ClickableComponent with isClicked initialized to false.

4.6.3 Member Data Documentation

4.6.3.1 isClicked

`bool ClickableComponent::isClicked`

True if the entity has been clicked, false otherwise.

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

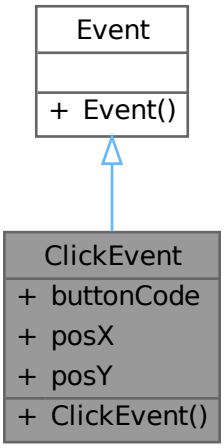
- src/Components/ClickableComponent.hpp

4.7 ClickEvent Class Reference

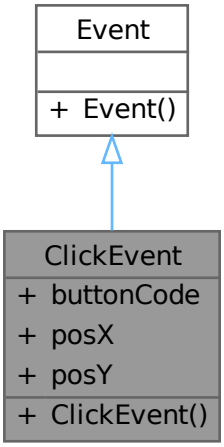
Represents a mouse click event.

`#include <ClickEvent.hpp>`

Inheritance diagram for ClickEvent:



Collaboration diagram for ClickEvent:



Public Member Functions

- ClickEvent (int buttonCode=0, int posX=0, int posY=0)

  Constructs a ClickEvent with optional button code and position.

Public Member Functions inherited from Event

- Event ()=default

  Default constructor.

Public Attributes

- int buttonCode

  Code of the mouse button clicked.
- int posX

  X position of the mouse click.
- int posY

  Y position of the mouse click.

4.7.1 Detailed Description

Represents a mouse click event.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 ClickEvent()

```
ClickEvent::ClickEvent (
            int buttonCode = 0,
            int posX = 0,
            int posY = 0 )  [inline]
```

Constructs a ClickEvent with optional button code and position.

Parameters

buttonCode	Code of the button clicked (default 0).
posX	X coordinate of the click (default 0).
posY	Y coordinate of the click (default 0).

4.7.3 Member Data Documentation

4.7.3.1 buttonCode

```
int ClickEvent::buttonCode
```

Code of the mouse button clicked.

4.7.3.2 posX

```
int ClickEvent::posX
```

X position of the mouse click.

4.7.3.3 posY

```
int ClickEvent::posY
```

Y position of the mouse click.

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

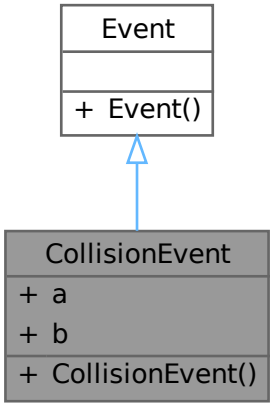
- src/Events/ClickEvent.hpp

4.8 CollisionEvent Class Reference

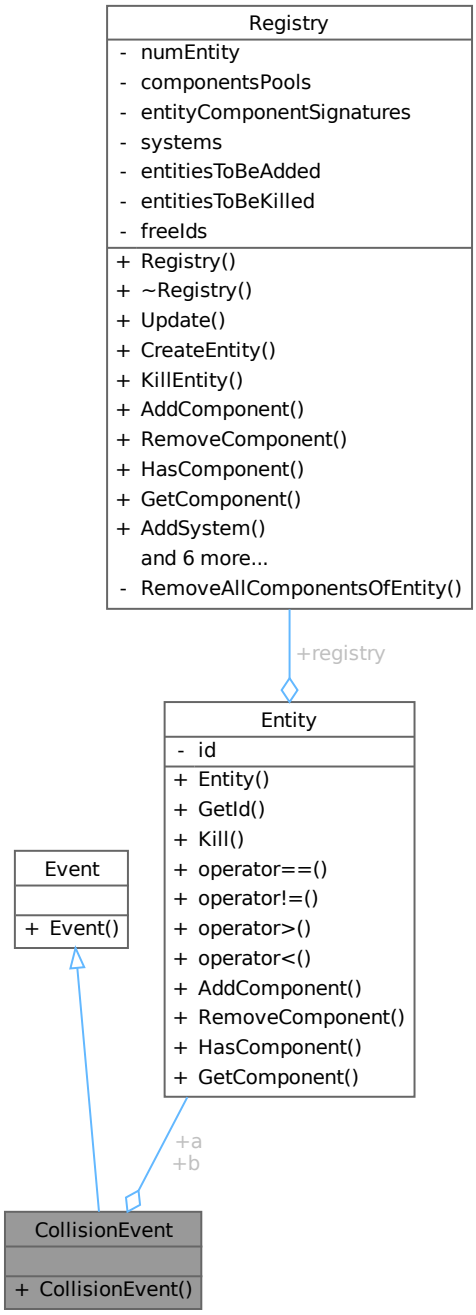
Event triggered when two entities collide.

```
#include <CollisionEvent.hpp>
```

Inheritance diagram for CollisionEvent:



Collaboration diagram for CollisionEvent:



Public Member Functions

- CollisionEvent (Entity a, Entity b)

  Constructs a CollisionEvent with two entities.

Public Member Functions inherited from Event

- Event ()=default

Default constructor.

Public Attributes

- Entity a

  First entity involved in the collision.
- Entity b

  Second entity involved in the collision.

4.8.1 Detailed Description

Event triggered when two entities collide.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 CollisionEvent()

```
CollisionEvent::CollisionEvent (
            Entity a,
            Entity b )  [inline]
```

Constructs a CollisionEvent with two entities.

Parameters

a	First entity involved.
b	Second entity involved.

4.8.3 Member Data Documentation

4.8.3.1 a

```
Entity CollisionEvent::a
```

First entity involved in the collision.

4.8.3.2 b

```
Entity CollisionEvent::b
```

Second entity involved in the collision.

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

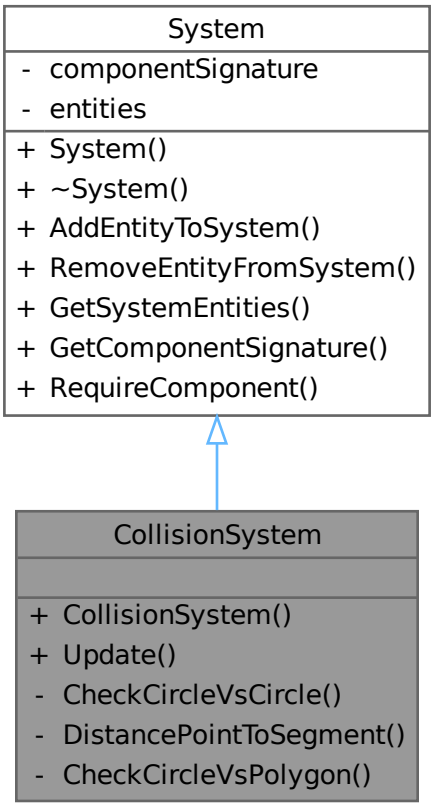
- src/Events/CollisionEvent.hpp

4.9 CollisionSystem Class Reference

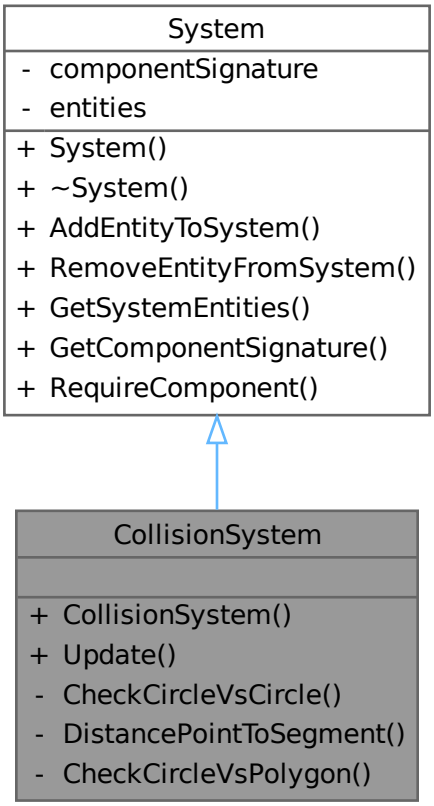
System responsible for detecting collisions between entities and emitting collision events.

`#include <CollisionSystem.hpp>`

Inheritance diagram for CollisionSystem:



Collaboration diagram for CollisionSystem:



Public Member Functions

- CollisionSystem ()

  Constructs the CollisionSystem and sets required components.
- void Update (std::unique_ptr< EventManager > &eventManager)

  Updates the system by checking collisions among all entities and emitting events.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

Returns the component signature required by this system.

- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

Private Member Functions

- bool CheckCircleVsCircle (Entity a, Entity b, TransformComponent aTransform, TransformComponent bTransform)

  Checks collision between two circle-collider entities.
- float DistancePointToSegment (glm::vec2 p, glm::vec2 a, glm::vec2 b)

  Calculates the shortest distance between point p and line segment ab.
- bool CheckCircleVsPolygon (Entity circleEntity, Entity polygonEntity, TransformComponent circleTransform)

  Checks collision between a circle entity and a polygon entity.

4.9.1 Detailed Description

System responsible for detecting collisions between entities and emitting collision events.

This system requires entities to have a TransformComponent. It supports collision detection between entities with CircleColliderComponent and PolygonColliderComponent.

The system detects collisions using circle-vs-circle and circle-vs-polygon algorithms. Upon detecting a collision, it emits a CollisionEvent through the EventManager.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 CollisionSystem()

```
CollisionSystem::CollisionSystem ( )  [inline]
```

Constructs the CollisionSystem and sets required components.

Here is the call graph for this function:

4.9.3 Member Function Documentation

4.9.3.1 CheckCircleVsCircle()

```
bool CollisionSystem::CheckCircleVsCircle (
            Entity a,
            Entity b,
            TransformComponent aTransform,
            TransformComponent bTransform )  [inline], [private]
```

Checks collision between two circle-collider entities.

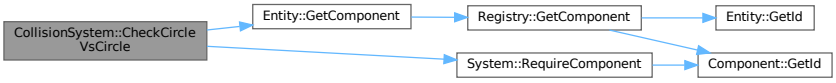
Parameters

a	First entity.
b	Second entity.
aTransform	Transform component of first entity.
bTransform	Transform component of second entity.

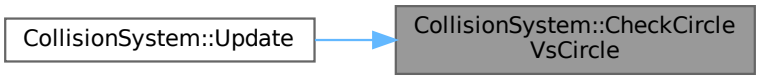
Returns

True if the two circles collide, false otherwise.

Collision is determined by comparing the distance between circle centers with the sum of their radii, and additionally comparing their scale levels. Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.2 CheckCircleVsPolygon()

```
bool CollisionSystem::CheckCircleVsPolygon (
            Entity circleEntity,
            Entity polygonEntity,
            TransformComponent circleTransform )  [inline], [private]
```

Checks collision between a circle entity and a polygon entity.

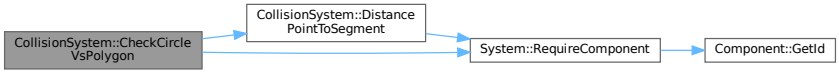
Parameters

circleEntity	Entity with a circle collider.
polygonEntity	Entity with a polygon collider.
circleTransform	Transform component of the circle entity.

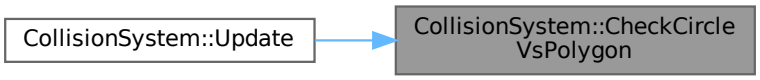
Returns

True if the circle collides with the polygon, false otherwise.

The collision is determined by checking if the distance from the circle center to any polygon edge is less than or equal to the circle radius. Here is the call graph for this function:



Here is the caller graph for this function:



4.9.3.3 DistancePointToSegment()

```
float CollisionSystem::DistancePointToSegment (
            glm::vec2 p,
            glm::vec2 a,
            glm::vec2 b )  [inline], [private]
```

Calculates the shortest distance between point p and line segment ab.

Parameters

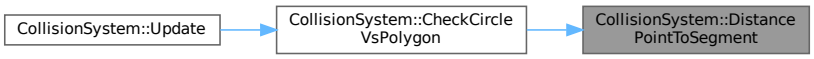
p	The point.
a	Start point of the line segment.
b	End point of the line segment.

Returns

The shortest distance from point p to the segment ab.

Here is the call graph for this function:

Here is the caller graph for this function:



4.9.3.4 Update()

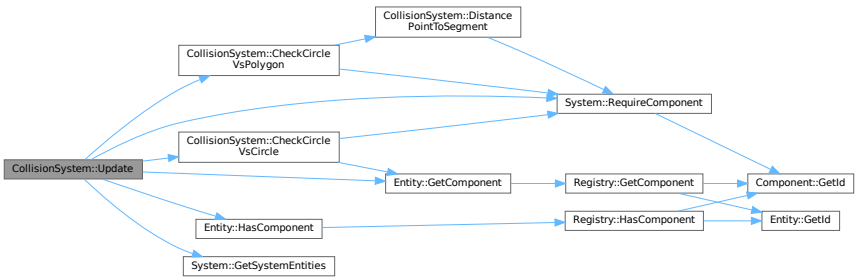
```
void CollisionSystem::Update (
            std::unique_ptr< EventManager > & eventManager ) [inline]
```

Updates the system by checking collisions among all entities and emitting events.

Parameters

eventManager	A unique pointer reference to the EventManager used to emit events.

Iterates through all pairs of entities, checks collision conditions (circle vs circle, circle vs polygon), and emits a CollisionEvent if a collision is detected. Here is the call graph for this function:



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

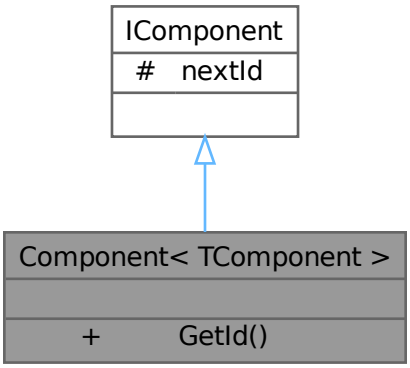
- src/Systems/CollisionSystem.hpp

4.10 Component< TComponent > Class Template Reference

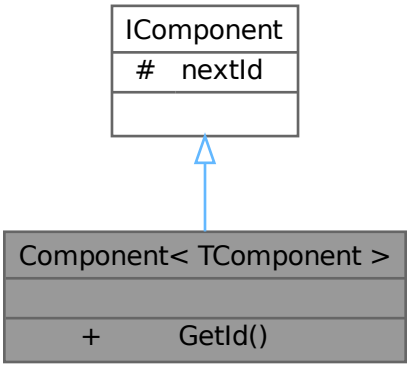
Template to generate unique component IDs per component type.

```
#include <ECS.hpp>
```

Inheritance diagram for Component< TComponent >:



Collaboration diagram for Component< TComponent >:



Static Public Member Functions

- static int GetId ()

  Returns the unique ID associated with this component type.

Additional Inherited Members

Static Protected Attributes inherited from IComponent

- static int nextId = 0

  Next available component ID.

4.10.1 Detailed Description

template<typename TComponent>
class Component< TComponent >

Template to generate unique component IDs per component type.

Template Parameters

TComponent	The component type.

4.10.2 Member Function Documentation

4.10.2.1 GetId()

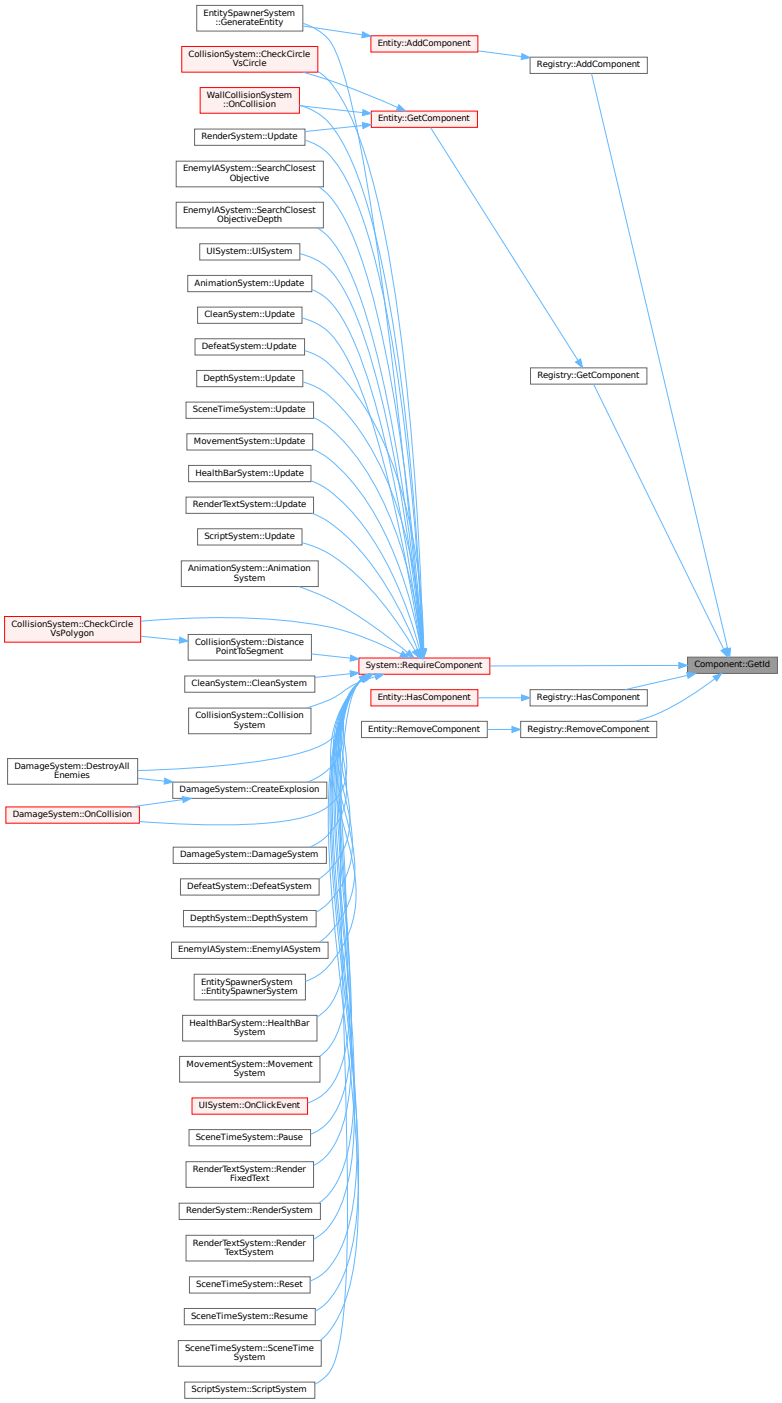
```
template<typename TComponent >
static int Component< TComponent >::GetId ( )  [inline], [static]
```

Returns the unique ID associated with this component type.

Returns

Unique component ID.

Here is the caller graph for this function:



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

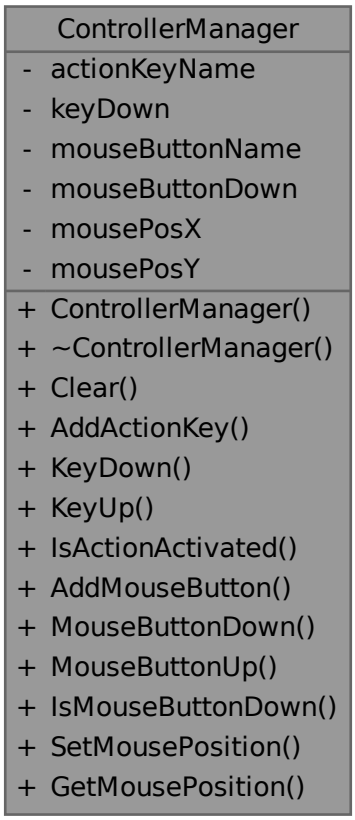
- src/ECS/ECS.hpp

4.11 ControllerManager Class Reference

Handles keyboard and mouse input mapping and state tracking.

`#include <ControllerManager.hpp>`

Collaboration diagram for ControllerManager:



Public Member Functions

- ControllerManager ()

  Constructor that initializes the controller manager.
- ~ControllerManager ()

  Destructor.
- void Clear ()

  Clears all stored key and action mappings.
- void AddActionKey (const std::string &action, int keyCode)

  Maps an action name to a keyboard key code.
- void KeyDown (int keyCode)

  Marks the specified key code as pressed.
- void KeyUp (int keyCode)

  Marks the specified key code as released.
- bool IsActionActivated (const std::string &action)

  Checks if the specified action is currently active (key pressed).
- void AddMouseButton (const std::string &name, int buttonCode)

Maps a mouse button name to a mouse button code.

- void MouseButtonDown (int buttonCode)

  Marks the specified mouse button code as pressed.
- void MouseButtonUp (int buttonCode)

  Marks the specified mouse button code as released.
- bool IsMouseButtonDown (const std::string &name)

  Checks if the specified mouse button is currently pressed.
- void SetMousePosition (int x, int y)

  Sets the current mouse position.
- std::tuple< int, int > GetMousePosition ()

  Gets the current mouse position.

Private Attributes

- std::map< std::string, int > actionKeyName

  Maps action names to keyboard key codes.
- std::map< int, bool > keyDown

  Tracks key pressed states by key code.
- std::map< std::string, int > mouseButtonName

  Maps mouse button names to button codes.
- std::map< int, bool > mouseButtonDown

  Tracks mouse button pressed states by button code.
- int mousePosX

  Current mouse X position.
- int mousePosY

  Current mouse Y position.

4.11.1 Detailed Description

Handles keyboard and mouse input mapping and state tracking.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 ControllerManager()

```
ControllerManager::ControllerManager ( )
```

Constructor that initializes the controller manager.

4.11.2.2 ~ControllerManager()

```
ControllerManager::~ControllerManager ( )
```

Destructor.

4.11.3 Member Function Documentation

4.11.3.1 AddActionKey()

```
void ControllerManager::AddActionKey (
            const std::string & action,
            int keyCode )
```

Maps an action name to a keyboard key code.

Parameters

action	The name of the action (e.g., "Jump").
keyCode	The SDL key code to bind to the action.

4.11.3.2 AddMouseButton()

```
void ControllerManager::AddMouseButton (
            const std::string & name,
            int buttonCode )
```

Maps a mouse button name to a mouse button code.

Parameters

name	The name of the mouse button (e.g., "LeftClick").
buttonCode	The SDL mouse button code.

4.11.3.3 Clear()

```
void ControllerManager::Clear ( )
```

Clears all stored key and action mappings.

4.11.3.4 GetMousePosition()

```
std::tuple< int, int > ControllerManager::GetMousePosition ( )
```

Gets the current mouse position.

Returns

A tuple with the x and y coordinates of the mouse.

4.11.3.5 IsActionActivated()

```
bool ControllerManager::IsActionActivated (
            const std::string & action )
```

Checks if the specified action is currently active (key pressed).

Parameters

action	The name of the action.

Returns

True if the key bound to the action is pressed; otherwise false.

4.11.3.6 IsMouseButtonDown()

```
bool ControllerManager::IsMouseButtonDown (
            const std::string & name )
```

Checks if the specified mouse button is currently pressed.

Parameters

name	The name of the mouse button.

Returns

True if the mouse button is pressed; otherwise false.

4.11.3.7 KeyDown()

```
void ControllerManager::KeyDown (
            int keyCode )
```

Marks the specified key code as pressed.

Parameters

keyCode	The SDL key code pressed.

4.11.3.8 KeyUp()

```
void ControllerManager::KeyUp (
            int keyCode )
```

Marks the specified key code as released.

Parameters

keyCode	The SDL key code released.

4.11.3.9 MouseButtonDown()

```
void ControllerManager::MouseButtonDown (
            int buttonCode )
```

Marks the specified mouse button code as pressed.

Parameters

buttonCode	The mouse button code pressed.

4.11.3.10 MouseButtonUp()

```
void ControllerManager::MouseButtonUp (
            int buttonCode )
```

Marks the specified mouse button code as released.

Parameters

buttonCode	The mouse button code released.

4.11.3.11 SetMousePosition()

```
void ControllerManager::SetMousePosition (
            int x,
            int y )
```

Sets the current mouse position.

Parameters

x	The x-coordinate of the mouse.
y	The y-coordinate of the mouse.

4.11.4 Member Data Documentation

4.11.4.1 actionKeyName

```
std::map<std::string, int> ControllerManager::actionKeyName  [private]
```

Maps action names to keyboard key codes.

4.11.4.2 keyDown

```
std::map<int, bool> ControllerManager::keyDown  [private]
```

Tracks key pressed states by key code.

4.11.4.3 mouseButtonDown

```
std::map<int, bool> ControllerManager::mouseButtonDown  [private]
```

Tracks mouse button pressed states by button code.

4.11.4.4 mouseButtonName

```
std::map<std::string, int> ControllerManager::mouseButtonName  [private]
```

Maps mouse button names to button codes.

4.11.4.5 mousePosX

```
int ControllerManager::mousePosX  [private]
```

Current mouse X position.

4.11.4.6 mousePosY

```
int ControllerManager::mousePosY  [private]
```

Current mouse Y position.

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

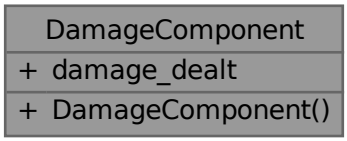
- src/ControllerManager/ControllerManager.hpp
- src/ControllerManager/ControllerManager.cpp

4.12 DamageComponent Struct Reference

Component that holds the amount of damage dealt.

```
#include <DamageComponent.hpp>
```

Collaboration diagram for DamageComponent:



Public Member Functions

- DamageComponent (int damage_dealt=0)

  Constructs a new DamageComponent.

Public Attributes

- int damage_dealt

4.12.1 Detailed Description

Component that holds the amount of damage dealt.

4.12.2 Constructor & Destructor Documentation

4.12.2.1 DamageComponent()

```
DamageComponent::DamageComponent (
            int damage_dealt = 0 )  [inline]
```

Constructs a new DamageComponent.

Parameters

damage_dealt	The damage amount (default is 0).

4.12.3 Member Data Documentation

4.12.3.1 damage_dealt

```
int DamageComponent::damage_dealt
```

The amount of damage dealt by the entity.

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

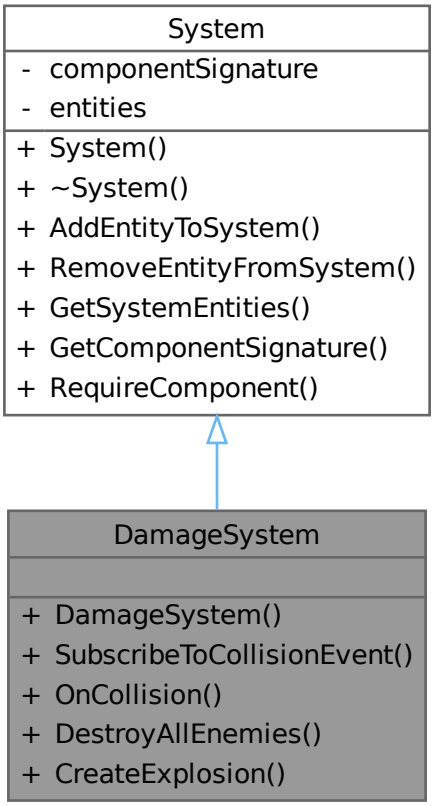
- src/Components/DamageComponent.hpp

4.13 DamageSystem Class Reference

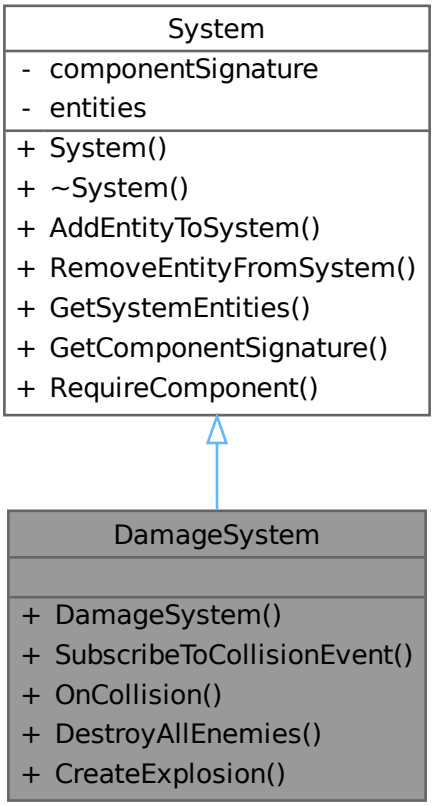
Handles damage application between entities when collisions occur.

```
#include <DamageSystem.hpp>
```

Inheritance diagram for DamageSystem:



Collaboration diagram for DamageSystem:



Public Member Functions

- DamageSystem ()

  Constructs the DamageSystem and requires entities to have CircleColliderComponent.
- void SubscribeToCollisionEvent (std::unique_ptr< EventManager > &eventManager)

  Subscribes the DamageSystem to CollisionEvent notifications from the EventManager.
- void OnCollision (CollisionEvent &e)

  Called when a CollisionEvent is emitted. Processes damage between entities involved in the collision.
- void DestroyAllEnemies ()

  Destroys all entities tagged as enemies (excluding players), triggering explosions.
- void CreateExplosion (Entity entity, int num, double scale)

  Creates an explosion entity at the position of the given entity.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

Adds an entity to this system.

- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.

- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.

- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.

- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.13.1 Detailed Description

Handles damage application between entities when collisions occur.

This system listens for collision events and processes damage interactions between entities based on their components and tags (e.g., enemy, projectile, objective). It also manages entity destruction and explosion creation when life reaches zero.

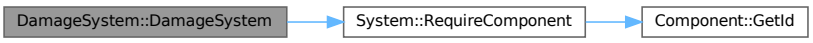
4.13.2 Constructor & Destructor Documentation

4.13.2.1 DamageSystem()

```
DamageSystem::DamageSystem ( )  [inline]
```

Constructs the DamageSystem and requires entities to have CircleColliderComponent.

Here is the call graph for this function:



4.13.3 Member Function Documentation

4.13.3.1 CreateExplosion()

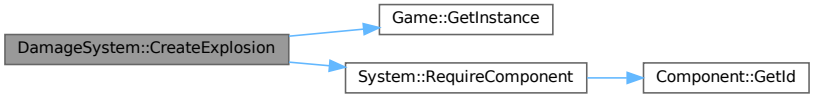
```
void DamageSystem::CreateExplosion (
            Entity entity,
            int num,
            double scale )  [inline]
```

Creates an explosion entity at the position of the given entity.

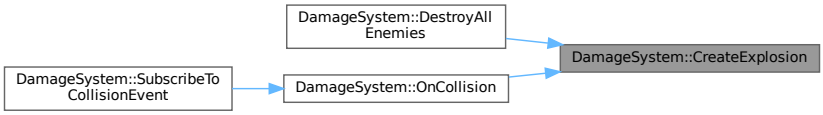
Parameters

entity	The entity at whose position the explosion is spawned.
num	The number of entities or particles to spawn for the explosion.
scale	The scale factor for the explosion visual size.

Here is the call graph for this function:



Here is the caller graph for this function:

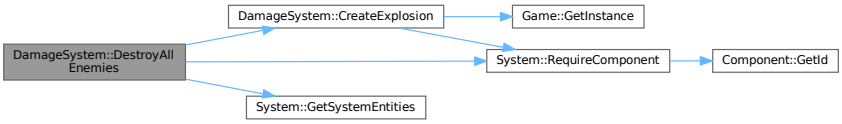


4.13.3.2 DestroyAllEnemies()

```
void DamageSystem::DestroyAllEnemies ( )  [inline]
```

Destroys all entities tagged as enemies (excluding players), triggering explosions.

Here is the call graph for this function:



4.13.3.3 OnCollision()

```
void DamageSystem::OnCollision (
            CollisionEvent & e )  [inline]
```

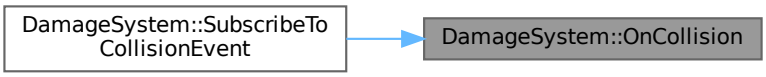
Called when a CollisionEvent is emitted. Processes damage between entities involved in the collision.

Parameters

e	Reference to the CollisionEvent containing the colliding entities.

Here is the call graph for this function:

Here is the caller graph for this function:



4.13.3.4 SubscribeToCollisionEvent()

```
void DamageSystem::SubscribeToCollisionEvent (
            std::unique_ptr< EventManager > & eventManager )  [inline]
```

Subscribes the DamageSystem to CollisionEvent notifications from the EventManager.

Parameters

eventManager	A unique pointer to the EventManager instance.

Here is the call graph for this function:

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

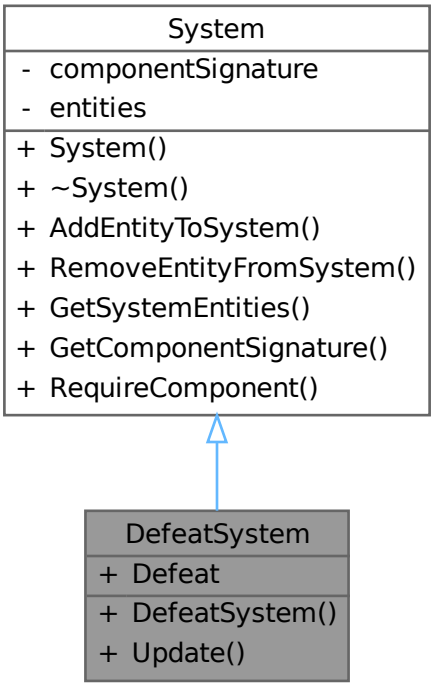
- src/Systems/DamageSystem.hpp

4.14 DefeatSystem Class Reference

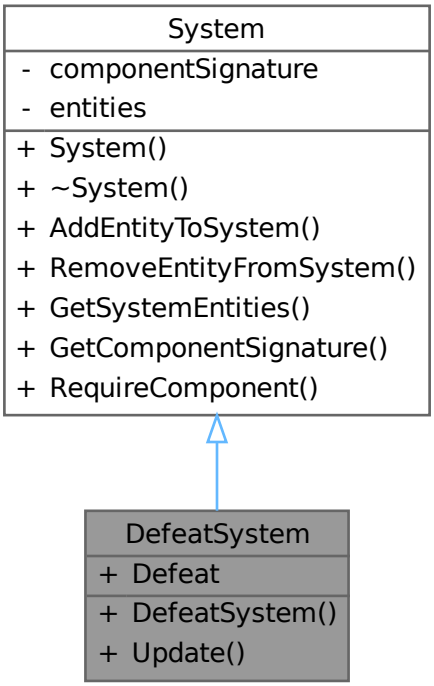
Checks if any objective entity is defeated (life <= 0).

`#include <DefeatSystem.hpp>`

Inheritance diagram for DefeatSystem:



Collaboration diagram for DefeatSystem:



Public Member Functions

- DefeatSystem ()
- void Update ()

  Updates the system, checking all objective entities for defeat.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

Public Attributes

- bool Defeat = false

  Constructs the DefeatSystem requiring LifeComponent and TagObjectiveComponent.

4.14.1 Detailed Description

Checks if any objective entity is defeated (life <= 0).

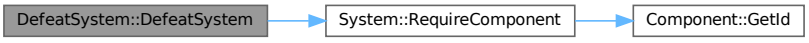
If any objective entity without the TagProjectileComponent reaches zero or less life, the Defeat flag is set to true.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 DefeatSystem()

```
DefeatSystem::DefeatSystem ( )  [inline]
```

Here is the call graph for this function:



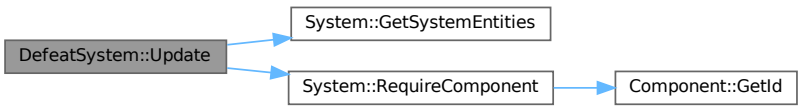
4.14.3 Member Function Documentation

4.14.3.1 Update()

```
void DefeatSystem::Update ( )  [inline]
```

Updates the system, checking all objective entities for defeat.

Sets Defeat to true if any objective entity without TagProjectileComponent has life_count <= 0. Here is the call graph for this function:



4.14.4 Member Data Documentation

4.14.4.1 Defeat

```
bool DefeatSystem::Defeat = false
```

Constructs the DefeatSystem requiring LifeComponent and TagObjectiveComponent.

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

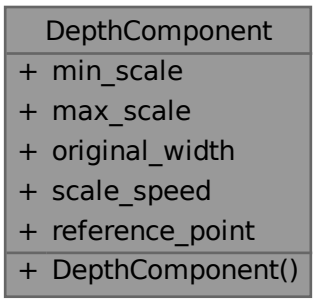
- src/Systems/DefeatSystem.hpp

4.15 DepthComponent Struct Reference

Component that manages depth-related scaling parameters for an entity.

```
#include <DepthComponent.hpp>
```

Collaboration diagram for DepthComponent:



Public Member Functions

- DepthComponent (float min_scale=0, float max_scale=0, float original_width=0, float scale_speed=0, float reference_point=0)

  Constructs a new DepthComponent.

Public Attributes

- float min_scale
- float max_scale
- float original_width
- float scale_speed
- float reference_point

4.15.1 Detailed Description

Component that manages depth-related scaling parameters for an entity.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 DepthComponent()

```
DepthComponent::DepthComponent (
            float min_scale = 0,
            float max_scale = 0,
            float original_width = 0,
            float scale_speed = 0,
            float reference_point = 0 )  [inline]
```

Constructs a new DepthComponent.

Parameters

min_scale	Minimum scale value (default 0).
max_scale	Maximum scale value (default 0).
original_width	Original width before scaling (default 0).
scale_speed	Speed of scale changes (default 0).
reference_point	Reference point for depth (default 0).

4.15.3 Member Data Documentation

4.15.3.1 max_scale

```
float DepthComponent::max_scale
```

Maximum scale value for depth effect.

4.15.3.2 min_scale

```
float DepthComponent::min_scale
```

Minimum scale value for depth effect.

4.15.3.3 original_width

```
float DepthComponent::original_width
```

Original width of the entity before scaling.

4.15.3.4 reference_point

```
float DepthComponent::reference_point
```

Reference point used for depth calculation.

4.15.3.5 scale_speed

```
float DepthComponent::scale_speed
```

Speed at which scaling changes over time.

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

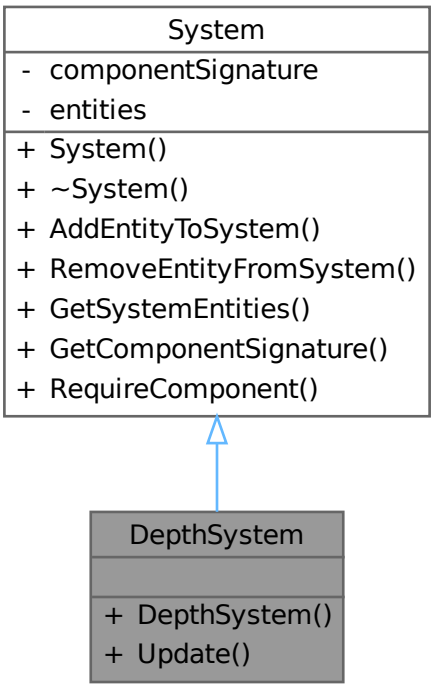
- src/Components/DepthComponent.hpp

4.16 DepthSystem Class Reference

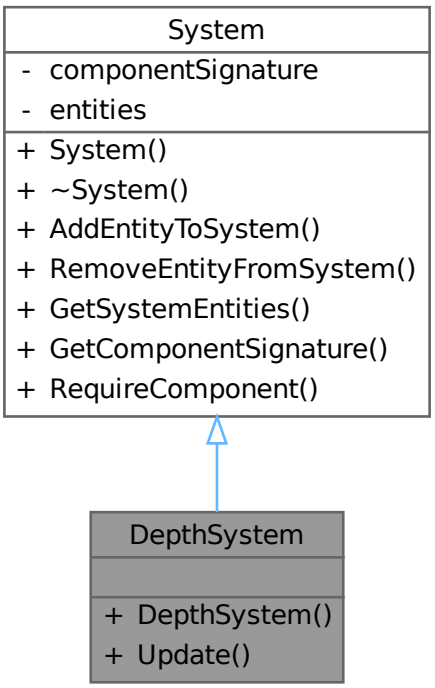
System to handle the scaling of entities based on their vertical velocity, simulating depth.

```
#include <DepthSystem.hpp>
```

Inheritance diagram for DepthSystem:



Collaboration diagram for DepthSystem:



Public Member Functions

- DepthSystem ()

  Constructor that requires necessary components for the system.
- void Update ()

  Updates all entities in the system, adjusting their scale and position based on velocity.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.16.1 Detailed Description

System to handle the scaling of entities based on their vertical velocity, simulating depth.

This system adjusts the scale of entities with DepthComponent, TransformComponent, and RigidBodyComponent to simulate a 3D depth effect by scaling entities smaller when moving up and larger when moving down. It also repositions entities to keep their center stable during scaling.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 DepthSystem()

```
DepthSystem::DepthSystem ( )  [inline]
```

Constructor that requires necessary components for the system.

Here is the call graph for this function:

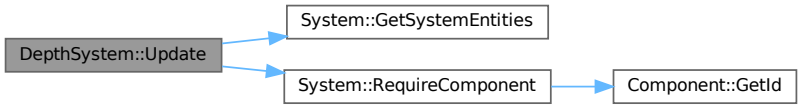
4.16.3 Member Function Documentation

4.16.3.1 Update()

```
void DepthSystem::Update ( )  [inline]
```

Updates all entities in the system, adjusting their scale and position based on velocity.

If the entity moves upward (negative y velocity), its scale decreases down to a minimum. If it moves downward (positive y velocity), its scale increases up to a maximum. The position is adjusted to maintain the entity's center point during scaling. Here is the call graph for this function:



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

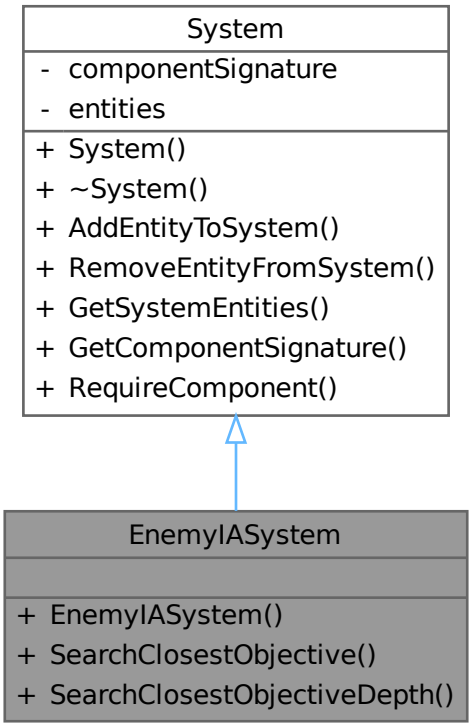
- src/Systems/DepthSystem.hpp

4.17 EnemyIASystem Class Reference

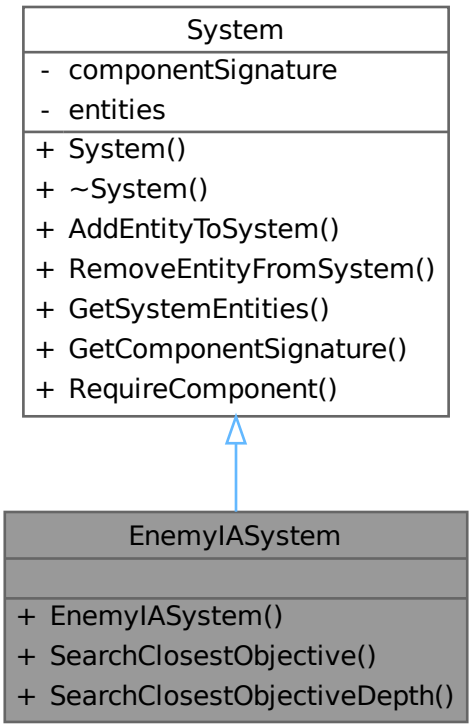
System responsible for enemy AI logic, specifically searching for closest objectives.

`#include <EnemyIASystem.hpp>`

Inheritance diagram for EnemyIASystem:



Collaboration diagram for EnemyIASystem:



Public Member Functions

- EnemyIASystem ()

  Constructor requiring the TransformComponent.
- TransformComponent SearchClosestObjective (Entity enemy, bool isPlayerIncl)

  Finds the closest objective's TransformComponent relative to the enemy.
- DepthComponent SearchClosestObjectiveDepth (Entity enemy, bool isPlayerIncl)

  Finds the closest objective's DepthComponent relative to the enemy.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.17.1 Detailed Description

System responsible for enemy AI logic, specifically searching for closest objectives.

This system provides functionality to find the closest objective to an enemy entity, optionally including or excluding player entities.

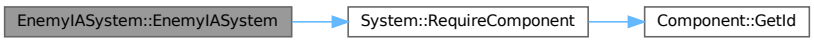
4.17.2 Constructor & Destructor Documentation

4.17.2.1 EnemyIASystem()

```
EnemyIASystem::EnemyIASystem ( )  [inline]
```

Constructor requiring the TransformComponent.

Here is the call graph for this function:



4.17.3 Member Function Documentation

4.17.3.1 SearchClosestObjective()

```
TransformComponent EnemyIASystem::SearchClosestObjective (
            Entity enemy,
            bool isPlayerIncl )  [inline]
```

Finds the closest objective's TransformComponent relative to the enemy.

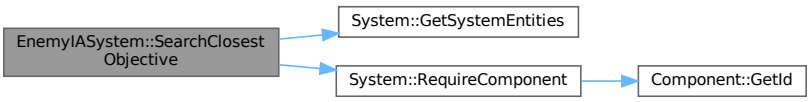
Parameters

enemy	The enemy entity performing the search.
isPlayerIncl	Whether to include player entities as objectives.

Returns

TransformComponent of the closest objective found.

Here is the call graph for this function:



4.17.3.2 SearchClosestObjectiveDepth()

```
DepthComponent EnemyIASystem::SearchClosestObjectiveDepth (
            Entity enemy,
            bool isPlayerIncl )  [inline]
```

Finds the closest objective's DepthComponent relative to the enemy.

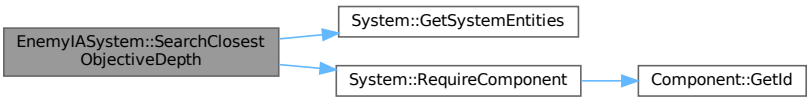
Parameters

enemy	The enemy entity performing the search.
isPlayerIncl	Whether to include player entities as objectives.

Returns

DepthComponent of the closest objective found.

Here is the call graph for this function:



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

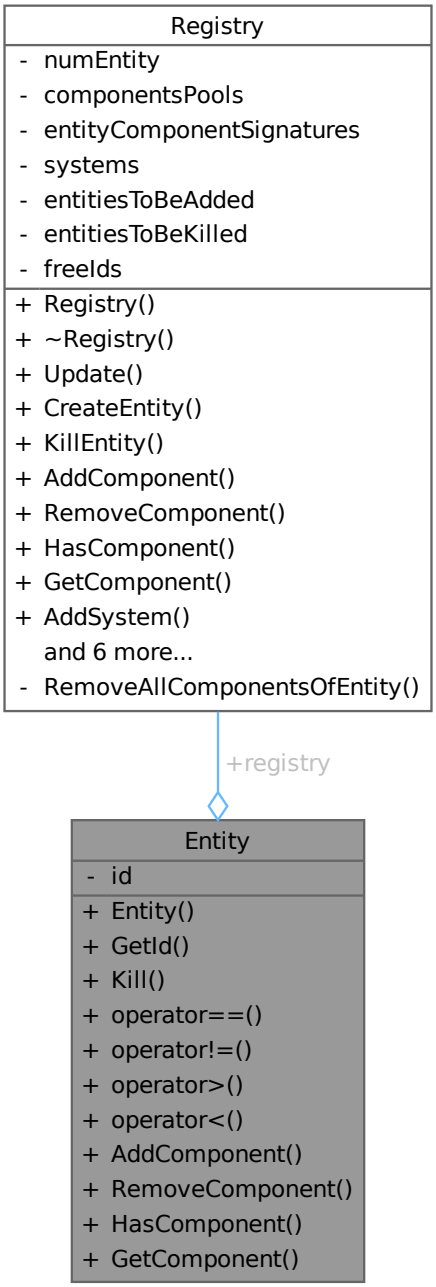
- src/Systems/EnemyIASystem.hpp

4.18 Entity Class Reference

Represents an entity in the ECS, identified by a unique ID.

`#include <ECS.hpp>`

Collaboration diagram for Entity:



Public Member Functions

- Entity (int id)

  Constructs an entity with the given ID.
- int GetId () const

  Returns the unique ID of the entity.
- void Kill ()

  Marks this entity for removal.
- bool operator== (const Entity &other) const
- bool operator!= (const Entity &other) const
- bool operator> (const Entity &other) const
- bool operator< (const Entity &other) const
- template<typename TComponent , typename... TArgs>
  void AddComponent (TArgs &&... args)

  Adds a component of type TComponent to the entity.
- template<typename TComponent >
  void RemoveComponent ()

  Removes a component of type TComponent from the entity.
- template<typename TComponent >
  bool HasComponent () const

  Checks if the entity has a component of type TComponent.
- template<typename TComponent >
  TComponent & GetComponent () const

  Returns a reference to the component of type TComponent.

Public Attributes

- class Registry * registry

  Pointer to the registry managing this entity.

Private Attributes

- int id

  Unique ID of the entity.

4.18.1 Detailed Description

Represents an entity in the ECS, identified by a unique ID.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 Entity()

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

Constructs an entity with the given ID.

Parameters

id	Unique identifier for the entity.

4.18.3 Member Function Documentation

4.18.3.1 AddComponent()

```
template<typename TComponent , typename...  TArgs>
void Entity::AddComponent (
            TArgs &&...  args )
```

Adds a component of type TComponent to the entity.

Template Parameters

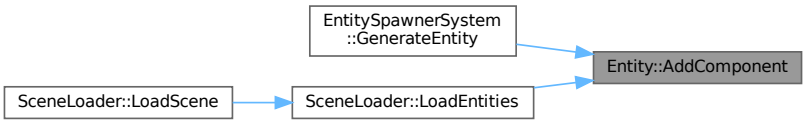
TComponent	The component type.
TArgs	Constructor arguments for the component.

Parameters

args	Arguments to forward to the component's constructor.

Here is the call graph for this function:

Here is the caller graph for this function:



4.18.3.2 GetComponent()

```
template<typename TComponent >
TComponent & Entity::GetComponent ( ) const
```

Returns a reference to the component of type TComponent.

Template Parameters

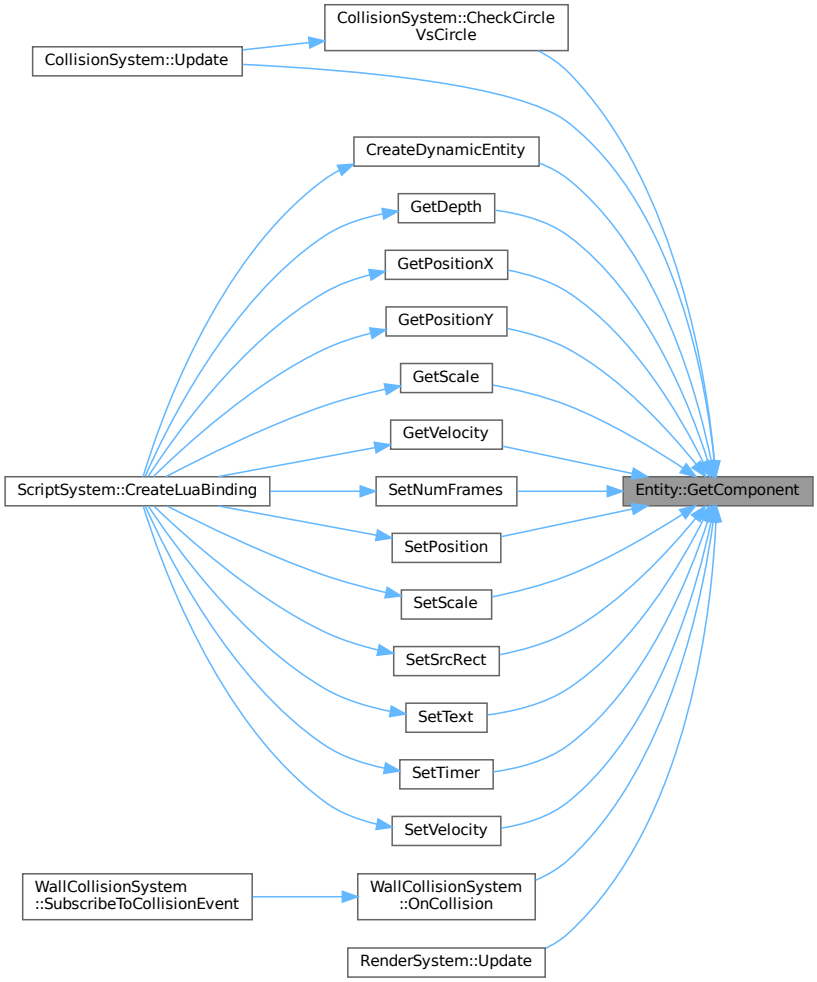
TComponent	The component type.

Returns

Reference to the component instance.

Here is the call graph for this function:

Here is the caller graph for this function:



4.18.3.3 GetId()

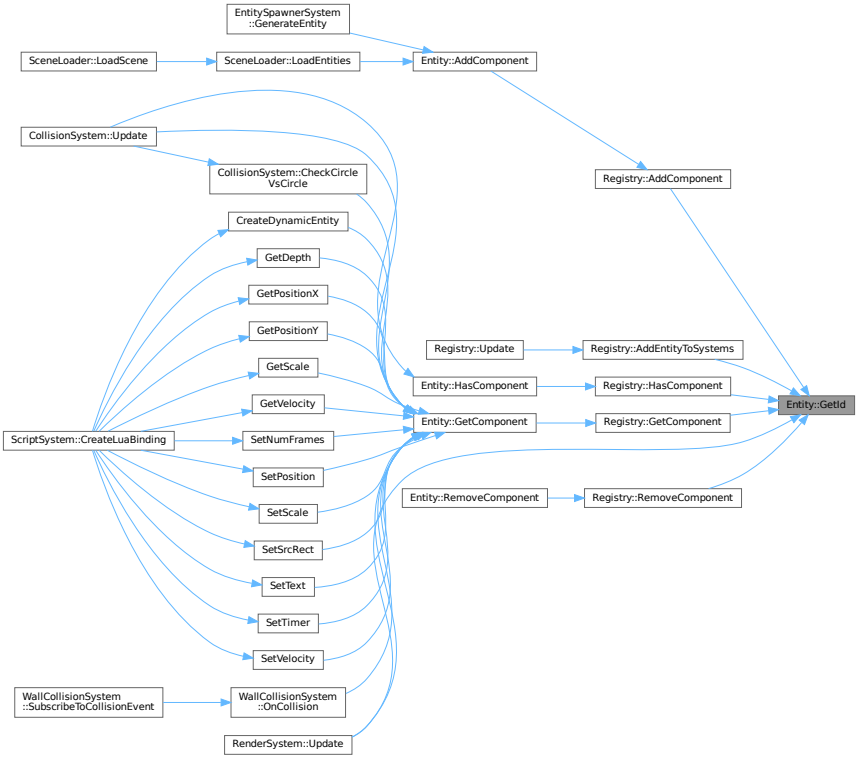
```
int Entity::GetId ( ) const
```

Returns the unique ID of the entity.

Returns

The entity ID.

Here is the caller graph for this function:



4.18.3.4 HasComponent()

```
template<typename TComponent >
bool Entity::HasComponent ( ) const
```

Checks if the entity has a component of type TComponent.

Template Parameters

TComponent	The component type to check.

Returns

True if the entity has the component, false otherwise.

Here is the call graph for this function:

Here is the caller graph for this function:

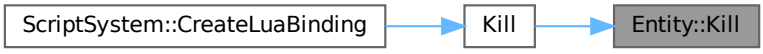
4.18.3.5 Kill()

```
void Entity::Kill ( )
```

Marks this entity for removal.

Here is the call graph for this function:

Here is the caller graph for this function:



4.18.3.6 operator"!=()

```
bool Entity::operator!= (
            const Entity & other ) const  [inline]
```

4.18.3.7 operator<()

```
bool Entity::operator< (
            const Entity & other ) const  [inline]
```

4.18.3.8 operator==()

```
bool Entity::operator== (
            const Entity & other ) const  [inline]
```

4.18.3.9 operator>()

```
bool Entity::operator> (
            const Entity & other ) const  [inline]
```

4.18.3.10 RemoveComponent()

```
template<typename TComponent >
void Entity::RemoveComponent ( )
```

Removes a component of type TComponent from the entity.

Template Parameters

TComponent	The component type to remove.

Here is the call graph for this function:

4.18.4 Member Data Documentation

4.18.4.1 id

```
int Entity::id  [private]
```

Unique ID of the entity.

4.18.4.2 registry

```
class Registry* Entity::registry
```

Pointer to the registry managing this entity.

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

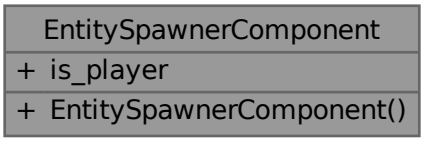
- src/ECS/ECS.hpp
- src/ECS/ECS.cpp

4.19 EntitySpawnerComponent Struct Reference

Component that marks an entity as a player or non-player spawner.

```
#include <EntitySpawnerComponent.hpp>
```

Collaboration diagram for EntitySpawnerComponent:



Public Member Functions

- EntitySpawnerComponent (bool is_player=false)
  Construct a new EntitySpawnerComponent.

Public Attributes

- bool is_player

4.19.1 Detailed Description

Component that marks an entity as a player or non-player spawner.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 EntitySpawnerComponent()

```
EntitySpawnerComponent::EntitySpawnerComponent (
            bool is_player = false )  [inline]
```

Construct a new EntitySpawnerComponent.

Parameters

is_player	Boolean indicating if spawner is for player (default false).

4.19.3 Member Data Documentation

4.19.3.1 is_player

```
bool EntitySpawnerComponent::is_player
```

Flag to indicate if this spawner is for the player (true) or not (false).

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

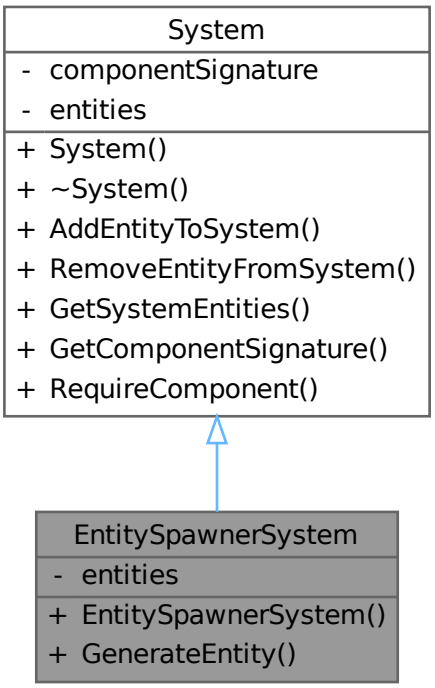
- src/Components/EntitySpawnerComponent.hpp

4.20 EntitySpawnerSystem Class Reference

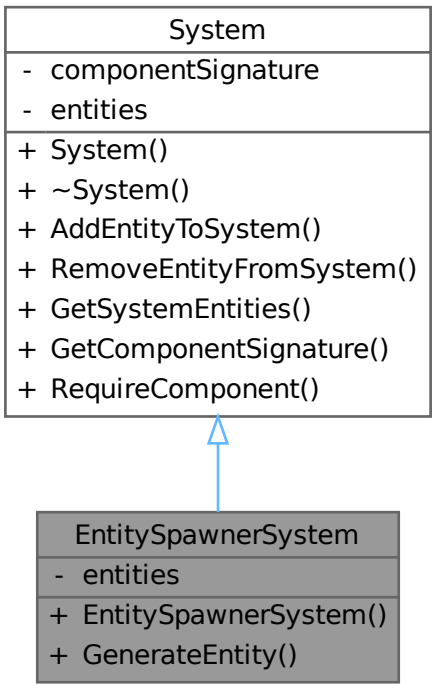
Handles the spawning of entities based on Lua scene configuration.

```
#include <EntitySpawnerSystem.hpp>
```

Inheritance diagram for EntitySpawnerSystem:



Collaboration diagram for EntitySpawnerSystem:



Public Member Functions

- EntitySpawnerSystem (const std::string &scenePath, sol::state &lua)

  Constructs the EntitySpawnerSystem, loads the Lua scene file and entities table.
- Entity GenerateEntity (std::unique_ptr< Registry > &registry, int idEntity, sol::state &lua)

  Generates an entity from the Lua entity table at the given index.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

Private Attributes

- sol::table entities

4.20.1 Detailed Description

Handles the spawning of entities based on Lua scene configuration.

This system loads entity definitions from a Lua script file and generates entities with the components defined in the Lua tables.

4.20.2 Constructor & Destructor Documentation

4.20.2.1 EntitySpawnerSystem()

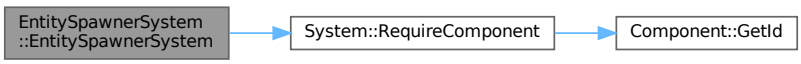
```
EntitySpawnerSystem::EntitySpawnerSystem (
            const std::string & scenePath,
            sol::state & lua )  [inline]
```

Constructs the EntitySpawnerSystem, loads the Lua scene file and entities table.

Parameters

scenePath	Path to the Lua scene file.
lua	Reference to the Lua state.

Here is the call graph for this function:



4.20.3 Member Function Documentation

4.20.3.1 GenerateEntity()

```
Entity EntitySpawnerSystem::GenerateEntity (
            std::unique_ptr< Registry > & registry,
            int idEntity,
            sol::state & lua )  [inline]
```

Generates an entity from the Lua entity table at the given index.

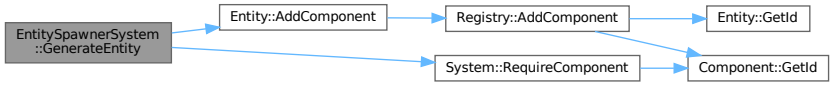
Parameters

registry	Unique pointer to the Registry managing entities.
idEntity	The index of the entity in the Lua entities table.
lua	Reference to the Lua state.

Returns

The generated Entity with components added as specified in Lua.

Here is the call graph for this function:



4.20.4 Member Data Documentation

4.20.4.1 entities

`sol::table EntitySpawnerSystem::entities [private]`

Lua table holding the entities loaded from the scene script

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

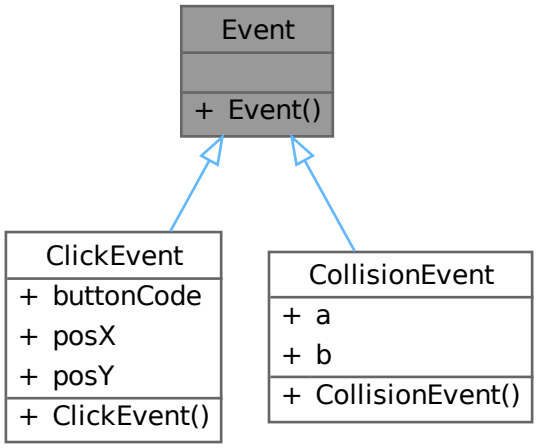
- src/Systems/EntitySpawnerSystem.hpp

4.21 Event Class Reference

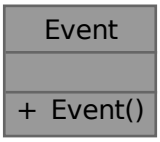
Base class for events in the system.

`#include <Event.hpp>`

Inheritance diagram for Event:



Collaboration diagram for Event:



Public Member Functions

- Event ()=default

  Default constructor.

4.21.1 Detailed Description

Base class for events in the system.

This class serves as a generic event type that can be extended to create specific event types for the event system.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 Event()

```
Event::Event ( )  [default]
```

Default constructor.

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

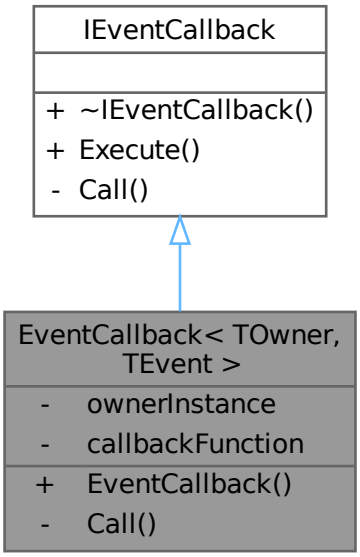
- src/EventManager/Event.hpp

4.22 EventCallback< TOwner, TEvent > Class Template Reference

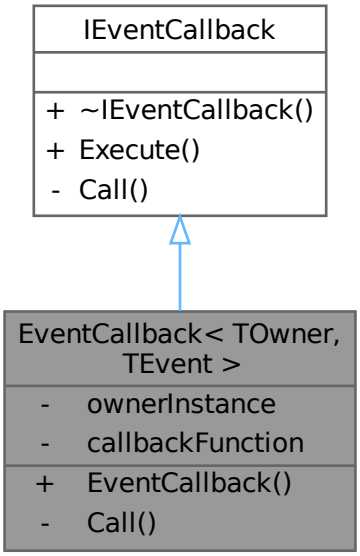
Template event callback handler connecting an owner and event type.

`#include <EventManager.hpp>`

Inheritance diagram for EventCallback< TOwner, TEvent >:



Collaboration diagram for EventCallback< TOwner, TEvent >:



Public Member Functions

- EventCallback (TOwner *ownerInstance, CallbackFunction callbackFunction)

  Constructs the callback handler with owner and function pointer.

Public Member Functions inherited from IEventCallback

- virtual ~IEventCallback ()=default

  Virtual destructor.
- void Execute (Event &e)

  Executes the callback with the given event.

Private Types

- typedef void(TOwner::* CallbackFunction) (TEvent &)

Private Member Functions

- virtual void Call (Event &e) override

  Calls the owner callback with the event, casting it to TEvent.

Private Attributes

- TOwner * ownerInstance

  Pointer to the owner instance.
- CallbackFunction callbackFunction

  Member function pointer to the callback.

4.22.1 Detailed Description

template<typename TOwner, typename TEvent>
class EventCallback< TOwner, TEvent >

Template event callback handler connecting an owner and event type.

Template Parameters

TOwner	Class type of the owner handling the event.
TEvent	Event type to handle.

4.22.2 Member Typedef Documentation

4.22.2.1 CallbackFunction

```
template<typename TOwner , typename TEvent >
typedef void(TOwner::* EventCallback< TOwner, TEvent >::CallbackFunction) (TEvent &)  [private]
```

4.22.3 Constructor & Destructor Documentation

4.22.3.1 EventCallback()

```
template<typename TOwner , typename TEvent >
EventCallback< TOwner, TEvent >::EventCallback (
            TOwner * ownerInstance,
            CallbackFunction callbackFunction )  [inline]
```

Constructs the callback handler with owner and function pointer.

Parameters

ownerInstance	Pointer to the owner object.
callbackFunction	Member function pointer to call on event.

4.22.4 Member Function Documentation

4.22.4.1 Call()

```
template<typename TOwner , typename TEvent >
virtual void EventCallback< TOwner, TEvent >::Call (
            Event & e )  [inline], [override], [private], [virtual]
```

Calls the owner callback with the event, casting it to TEvent.

Parameters

e	Reference to the base Event to cast and pass.

Implements IEventCallback.

4.22.5 Member Data Documentation

4.22.5.1 callbackFunction

```
template<typename TOwner , typename TEvent >
CallbackFunction EventCallback< TOwner, TEvent >::callbackFunction  [private]
```

Member function pointer to the callback.

4.22.5.2 ownerInstance

```
template<typename TOwner , typename TEvent >
TOwner* EventCallback< TOwner, TEvent >::ownerInstance  [private]
```

Pointer to the owner instance.

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

- src/EventManager/EventManager.hpp

4.23 EventManager Class Reference

Manages event subscription and emission.

```
#include <EventManager.hpp>
```

Collaboration diagram for EventManager:

Public Member Functions

- EventManager ()

  Constructs an EventManager.
- ~EventManager ()

  Destructor.
- void Reset ()

  Clears all subscribers, effectively unsubscribing all.
- template<typename TEvent , typename TOwner >
  void SubscribeToEvent (TOwner *ownerInstance, void(TOwner::*callbackFunction)(TEvent &))

  Subscribes a member function of an owner to a specific event type.
- template<typename TEvent , typename... TArgs>
  void EmitEvent (TArgs &&... args)

  Emits an event to all subscribers of the event's type.

Private Attributes

- std::map< std::type_index, std::unique_ptr< HandlerList > > subscribers

4.23.1 Detailed Description

Manages event subscription and emission.

Allows objects to subscribe member functions to specific event types and emits events to all registered subscribers.

4.23.2 Constructor & Destructor Documentation

4.23.2.1 EventManager()

```
EventManager::EventManager ( )  [inline]
```

Constructs an EventManager.

4.23.2.2 ~EventManager()

```
EventManager::~EventManager ( )  [inline]
```

Destructor.

4.23.3 Member Function Documentation

4.23.3.1 EmitEvent()

```
template<typename TEvent , typename...  TArgs>
void EventManager::EmitEvent (
            TArgs &&...  args )  [inline]
```

Emits an event to all subscribers of the event's type.

Template Parameters

TEvent	The event type to emit.
TArgs	Argument types to forward to the event constructor.

Parameters

args	Arguments forwarded to the event constructor.

4.23.3.2 Reset()

```
void EventManager::Reset ( )  [inline]
```

Clears all subscribers, effectively unsubscribing all.

4.23.3.3 SubscribeToEvent()

```
template<typename TEvent , typename TOwner >
void EventManager::SubscribeToEvent (
            TOwner * ownerInstance,
            void(TOwner::*)(TEvent &) callbackFunction )  [inline]
```

Subscribes a member function of an owner to a specific event type.

Template Parameters

TEvent	The event type to subscribe to.
TOwner	The class type of the owner handling the event.

Parameters

ownerInstance	Pointer to the owner instance.
callbackFunction	Member function pointer to the callback.

4.23.4 Member Data Documentation

4.23.4.1 subscribers

```
std::map<std::type_index, std::unique_ptr<HandlerList> > EventManager::subscribers  [private]
```

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

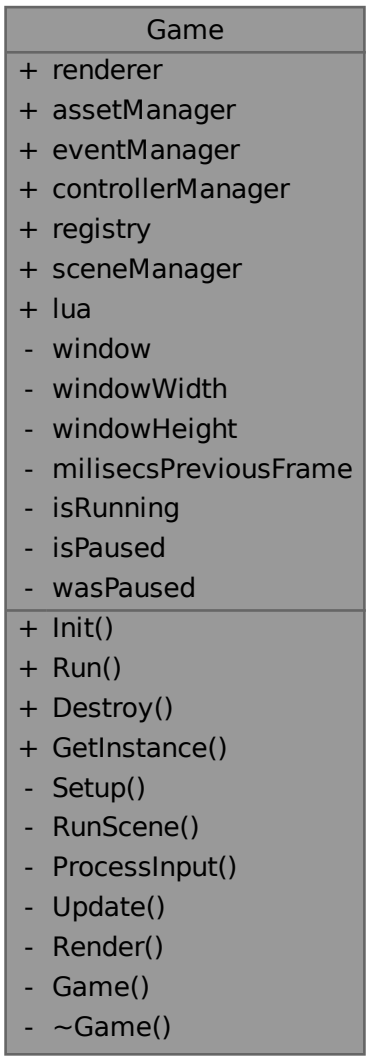
- src/EventManager/EventManager.hpp

4.24 Game Class Reference

Core class that manages the entire game lifecycle.

```
#include <Game.hpp>
```

Collaboration diagram for Game:



Public Member Functions

- void Init ()

  Initializes the game (window, subsystems, resources).
- void Run ()

  Runs the main game loop.
- void Destroy ()

  Cleans up and destroys all game resources.

Static Public Member Functions

- static Game & GetInstance ()

  Provides global access to the single Game instance.

Public Attributes

- SDL_Renderer * renderer = nullptr
  SDL renderer pointer.
- std::unique_ptr< AssetManager > assetManager
  Manages game assets.
- std::unique_ptr< EventManager > eventManager
  Manages events and event listeners.
- std::unique_ptr< ControllerManager > controllerManager
  Manages input controllers.
- std::unique_ptr< Registry > registry
  ECS registry for entities and components.
- std::unique_ptr< SceneManager > sceneManager
  Manages game scenes.
- sol::state lua
  Lua scripting state.

Private Member Functions

- void Setup ()
  Setup initial game state and resources.
- void RunScene ()
  Execute the current scene's logic.
- void ProcessInput ()
  Handle input processing.
- void Update ()
  Update game logic.
- void Render ()
  Render the current frame.
- Game ()
  Private constructor for singleton pattern.
- ~Game ()
  Destructor.

Private Attributes

- SDL_Window * window = nullptr
  The SDL window.
- int windowWidth = 0
  Window width in pixels.
- int windowHeight = 0
  Window height in pixels.
- int milisecsPreviousFrame = 0
  Timestamp of previous frame in milliseconds.
- bool isRunning = false
  Flag to check if game loop is running.
- bool isPaused = false
  Flag to check if the game is currently paused.
- bool wasPaused = false
  Flag to check if the game was paused in the last frame.

4.24.1 Detailed Description

Core class that manages the entire game lifecycle.

This class handles initialization, the main game loop, input processing, scene management, updates, rendering, and resource cleanup. It follows the singleton pattern to ensure a single instance throughout the game.

4.24.2 Constructor & Destructor Documentation

4.24.2.1 Game()

```
Game::Game ( )  [private]
```

Private constructor for singleton pattern.

4.24.2.2 ~Game()

```
Game::~Game ( )  [private]
```

Destructor.

4.24.3 Member Function Documentation

4.24.3.1 Destroy()

```
void Game::Destroy ( )
```

Cleans up and destroys all game resources.

Here is the caller graph for this function:

4.24.3.2 GetInstance()

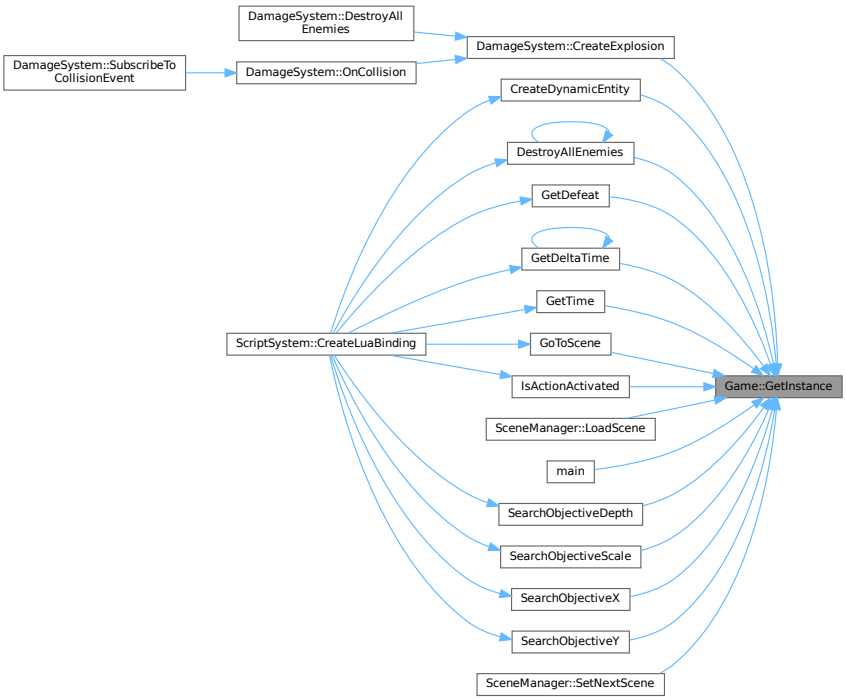
```
Game & Game::GetInstance ( )  [static]
```

Provides global access to the single Game instance.

Returns

Reference to the Game instance.

Here is the caller graph for this function:

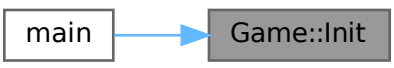


4.24.3.3 Init()

```
void Game::Init ( )
```

Initializes the game (window, subsystems, resources).

Here is the caller graph for this function:



4.24.3.4 ProcessInput()

```
void Game::ProcessInput ( )  [private]
```

Handle input processing.

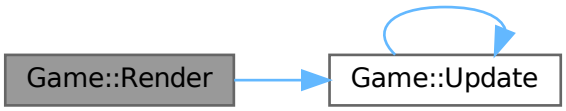
Here is the caller graph for this function:

4.24.3.5 Render()

```
void Game::Render ( )  [private]
```

Render the current frame.

Here is the call graph for this function:



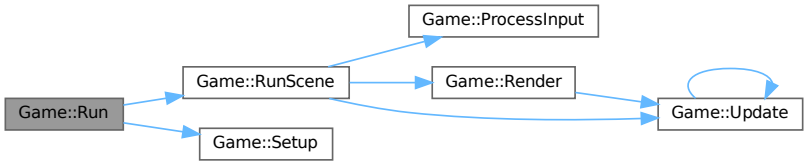
Here is the caller graph for this function:

4.24.3.6 Run()

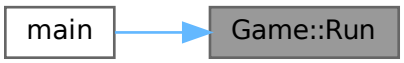
```
void Game::Run ( )
```

Runs the main game loop.

Here is the call graph for this function:



Here is the caller graph for this function:

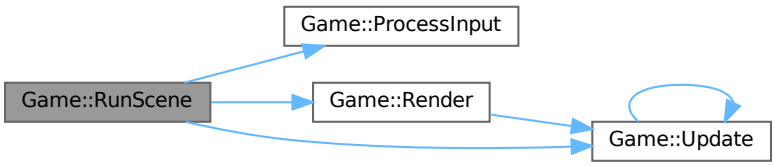


4.24.3.7 RunScene()

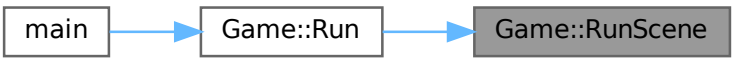
```
void Game::RunScene ( )  [private]
```

Execute the current scene's logic.

Here is the call graph for this function:



Here is the caller graph for this function:

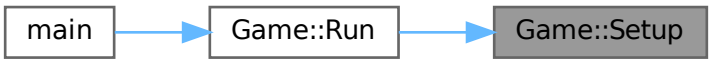


4.24.3.8 Setup()

```
void Game::Setup ( )  [private]
```

Setup initial game state and resources.

Here is the caller graph for this function:



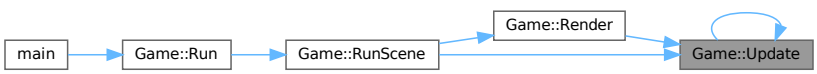
4.24.3.9 Update()

```
void Game::Update ( )  [private]
```

Update game logic.

Here is the call graph for this function:

Here is the caller graph for this function:



4.24.4 Member Data Documentation

4.24.4.1 assetManager

```
std::unique_ptr<AssetManager> Game::assetManager
```

Manages game assets.

4.24.4.2 controllerManager

```
std::unique_ptr<ControllerManager> Game::controllerManager
```

Manages input controllers.

4.24.4.3 eventManager

```
std::unique_ptr<EventManager> Game::eventManager
```

Manages events and event listeners.

4.24.4.4 isPaused

```
bool Game::isPaused = false  [private]
```

Flag to check if the game is currently paused.

4.24.4.5 isRunning

```
bool Game::isRunning = false  [private]
```

Flag to check if game loop is running.

4.24.4.6 lua

```
sol::state Game::lua
```

Lua scripting state.

4.24.4.7 milisecsPreviousFrame

```
int Game::milisecsPreviousFrame = 0  [private]
```

Timestamp of previous frame in milliseconds.

4.24.4.8 registry

```
std::unique_ptr<Registry> Game::registry
```

ECS registry for entities and components.

4.24.4.9 renderer

```
SDL_Renderer* Game::renderer = nullptr
```

SDL renderer pointer.

4.24.4.10 sceneManager

```
std::unique_ptr<SceneManager> Game::sceneManager
```

Manages game scenes.

4.24.4.11 wasPaused

```
bool Game::wasPaused = false  [private]
```

Flag to check if the game was paused in the last frame.

4.24.4.12 window

```
SDL_Window* Game::window = nullptr  [private]
```

The SDL window.

4.24.4.13 windowHeight

```
int Game::windowHeight = 0  [private]
```

Window height in pixels.

4.24.4.14 windowWidth

```
int Game::windowWidth = 0  [private]
```

Window width in pixels.

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

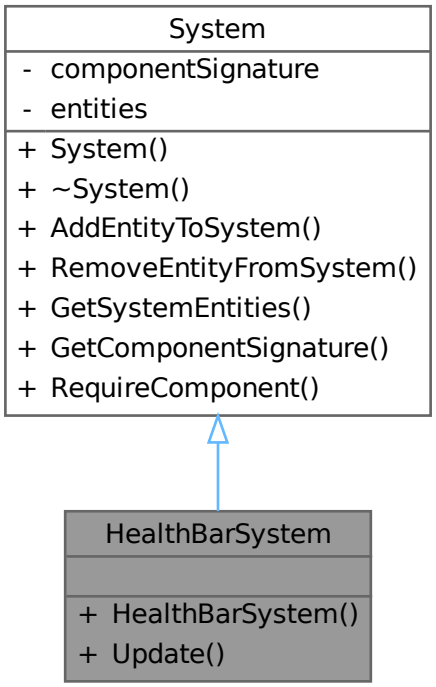
- src/Game/Game.hpp
- src/Game/Game.cpp

4.25 HealthBarSystem Class Reference

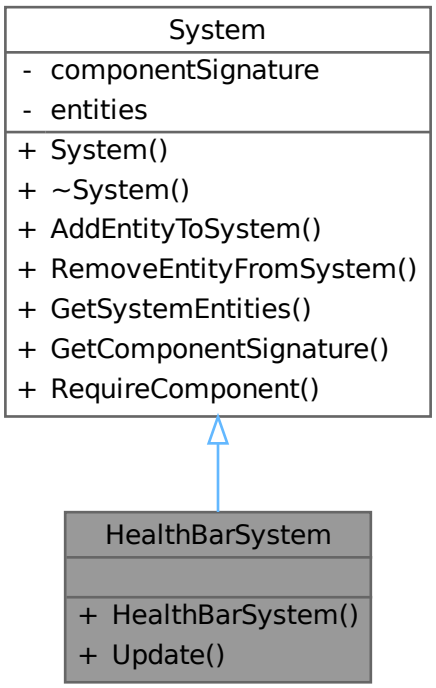
System responsible for rendering health bars for entities with LifeComponent and TagObjectiveComponent.

`#include <HealthBarSystem.hpp>`

Inheritance diagram for HealthBarSystem:



Collaboration diagram for HealthBarSystem:



Public Member Functions

- HealthBarSystem ()

  Constructs a HealthBarSystem and requires LifeComponent and TagObjectiveComponent.
- void Update (SDL_Renderer *renderer)

  Updates and renders health bars for all entities in the system.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.25.1 Detailed Description

System responsible for rendering health bars for entities with LifeComponent and TagObjectiveComponent.

This system draws the health bars differently depending on whether the entity has a TagPlayerComponent or not. For player entities, the health bar is drawn at position (100, 50) with a green fill. For other objective entities (e.g., obelisks), the health bar is drawn at position (100, 80) with a blue fill.

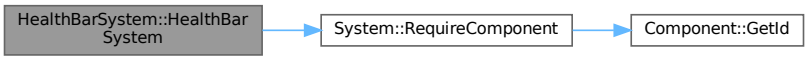
4.25.2 Constructor & Destructor Documentation

4.25.2.1 HealthBarSystem()

```
HealthBarSystem::HealthBarSystem ( )  [inline]
```

Constructs a HealthBarSystem and requires LifeComponent and TagObjectiveComponent.

Here is the call graph for this function:



4.25.3 Member Function Documentation

4.25.3.1 Update()

```
void HealthBarSystem::Update (
            SDL_Renderer * renderer )  [inline]
```

Updates and renders health bars for all entities in the system.

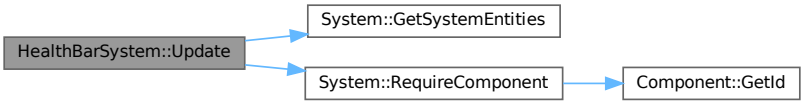
Parameters

renderer	Pointer to the SDL_Renderer used for drawing the health bars.

For each entity:

- Retrieves the LifeComponent to get current and maximum life.
- Draws a white border rectangle.
- Fills the rectangle proportionally to the entity's current life.
- Uses green color for player entities and blue for others.

Here is the call graph for this function:



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

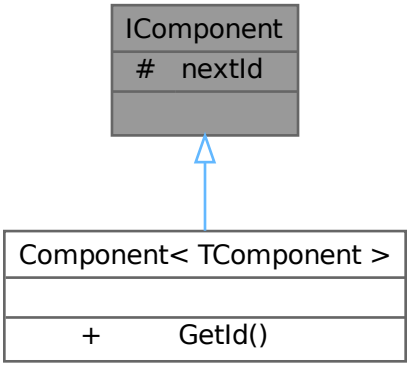
- src/Systems/HealthBarSystem.hpp

4.26 IComponent Struct Reference

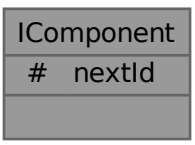
Base class for all components to generate unique IDs.

`#include <ECS.hpp>`

Inheritance diagram for IComponent:



Collaboration diagram for IComponent:



Static Protected Attributes

- static int nextId = 0

  Next available component ID.

4.26.1 Detailed Description

Base class for all components to generate unique IDs.

4.26.2 Member Data Documentation

4.26.2.1 nextId

```
int IComponent::nextId = 0  [static], [protected]
```

Next available component ID.

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

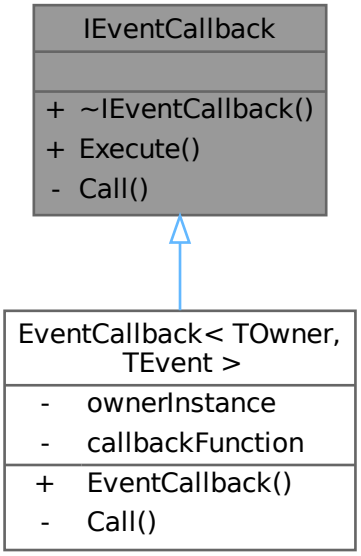
- src/ECS/ECS.hpp
- src/ECS/ECS.cpp

4.27 IEventCallback Class Reference

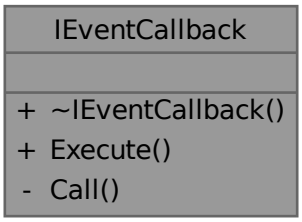
Abstract interface for event callback handlers.

```
#include <EventManager.hpp>
```

Inheritance diagram for IEventCallback:



Collaboration diagram for IEventCallback:



Public Member Functions

- virtual ~IEventCallback ()=default
  Virtual destructor.
- void Execute (Event &e)
  Executes the callback with the given event.

Private Member Functions

- virtual void Call (Event &e)=0
  Calls the stored callback with the given event.

4.27.1 Detailed Description

Abstract interface for event callback handlers.

4.27.2 Constructor & Destructor Documentation

4.27.2.1 ~IEventCallback()

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

Virtual destructor.

4.27.3 Member Function Documentation

4.27.3.1 Call()

```
virtual void IEventCallback::Call (
            Event & e )  [private], [pure virtual]
```

Calls the stored callback with the given event.

Parameters

e	Reference to the event to process.

Implemented in EventCallback< TOwner, TEvent >.

Here is the caller graph for this function:

4.27.3.2 Execute()

```
void IEventCallback::Execute (
            Event & e )  [inline]
```

Executes the callback with the given event.

Parameters

e	Reference to the event to handle.

Here is the call graph for this function:

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

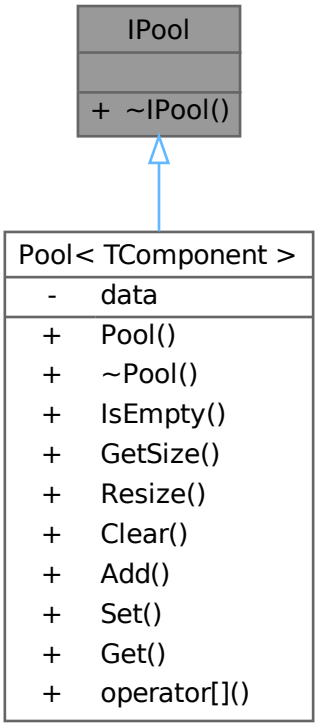
- src/EventManager/EventManager.hpp

4.28 IPool Class Reference

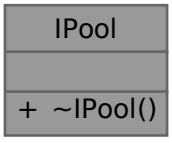
Interface for component pools.

`#include <Pool.hpp>`

Inheritance diagram for IPool:



Collaboration diagram for IPool:



Public Member Functions

- virtual ~IPool ()=default

  Virtual destructor for cleanup in derived classes.

4.28.1 Detailed Description

Interface for component pools.

4.28.2 Constructor & Destructor Documentation

4.28.2.1 ~IPool()

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

Virtual destructor for cleanup in derived classes.

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

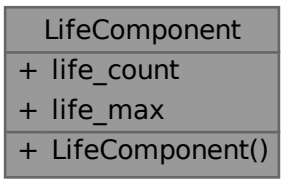
- src/Utils/Pool.hpp

4.29 LifeComponent Struct Reference

Represents the current and maximum life of an entity.

```
#include <LifeComponent.hpp>
```

Collaboration diagram for LifeComponent:



Public Member Functions

- LifeComponent (int life_count=0, int life_max=0)

  Construct a new LifeComponent object.

Public Attributes

- int life_count
- int life_max

4.29.1 Detailed Description

Represents the current and maximum life of an entity.

4.29.2 Constructor & Destructor Documentation

4.29.2.1 LifeComponent()

```
LifeComponent::LifeComponent (
            int life_count = 0,
            int life_max = 0 )  [inline]
```

Construct a new LifeComponent object.

Parameters

life_count	Initial life count (default 0).
life_max	Maximum life count (default 0).

4.29.3 Member Data Documentation

4.29.3.1 life_count

```
int LifeComponent::life_count
```

Current life count of the entity.

4.29.3.2 life_max

```
int LifeComponent::life_max
```

Maximum life the entity can have.

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

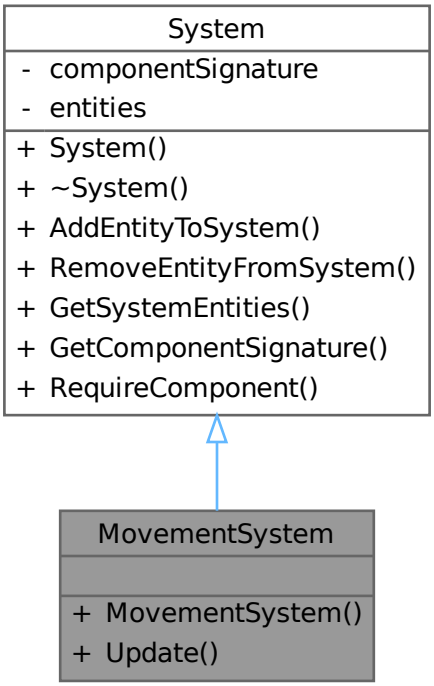
- src/Components/LifeComponent.hpp

4.30 MovementSystem Class Reference

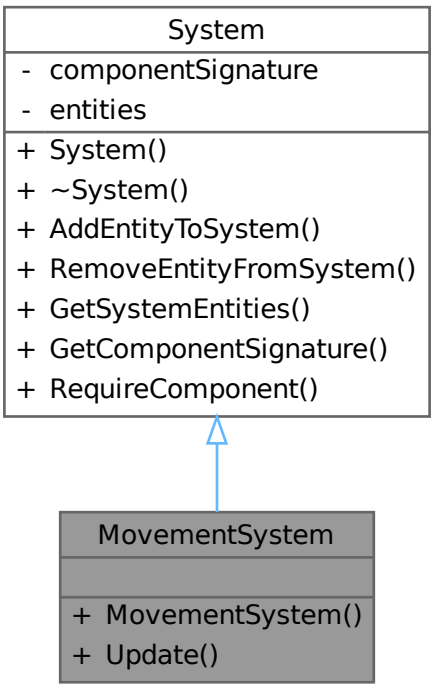
System responsible for updating entity positions based on their velocity.

```
#include <MovementSystem.hpp>
```

Inheritance diagram for MovementSystem:



Collaboration diagram for MovementSystem:



Public Member Functions

- MovementSystem ()

  Constructs a MovementSystem and requires RigidBodyComponent and TransformComponent.
- void Update (double dt)

  Updates the position of each entity based on its velocity and the delta time.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.30.1 Detailed Description

System responsible for updating entity positions based on their velocity.

This system requires entities to have both RigidBodyComponent and TransformComponent. It updates the position of each entity every frame according to its velocity and the elapsed time.

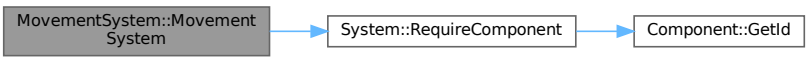
4.30.2 Constructor & Destructor Documentation

4.30.2.1 MovementSystem()

```
MovementSystem::MovementSystem ( )  [inline]
```

Constructs a MovementSystem and requires RigidBodyComponent and TransformComponent.

Here is the call graph for this function:



4.30.3 Member Function Documentation

4.30.3.1 Update()

```
void MovementSystem::Update (
            double dt )  [inline]
```

Updates the position of each entity based on its velocity and the delta time.

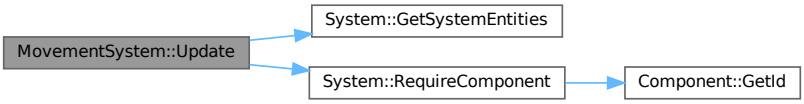
Parameters

dt	The delta time (time elapsed since last update), used to scale velocity.

For each entity:

- Retrieves the velocity from the RigidBodyComponent.
- Updates the position in the TransformComponent by velocity ∗ dt.

Here is the call graph for this function:



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

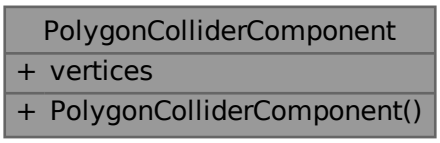
- src/Systems/MovementSystem.hpp

4.31 PolygonColliderComponent Struct Reference

Stores the vertices that define a polygon collider shape.

`#include <PolygonColliderComponent.hpp>`

Collaboration diagram for PolygonColliderComponent:



Public Member Functions

- PolygonColliderComponent (std::vector< glm::vec2 > vertices={})
  Construct a new PolygonColliderComponent object.

Public Attributes

- std::vector< glm::vec2 > vertices

4.31.1 Detailed Description

Stores the vertices that define a polygon collider shape.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 PolygonColliderComponent()

```
PolygonColliderComponent::PolygonColliderComponent (
            std::vector< glm::vec2 > vertices = {} )  [inline]
```

Construct a new PolygonColliderComponent object.

Parameters

vertices	Vector of 2D points representing the polygon vertices (default empty).

4.31.3 Member Data Documentation

4.31.3.1 vertices

```
std::vector<glm::vec2> PolygonColliderComponent::vertices
```

List of vertices that form the polygon collider.

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

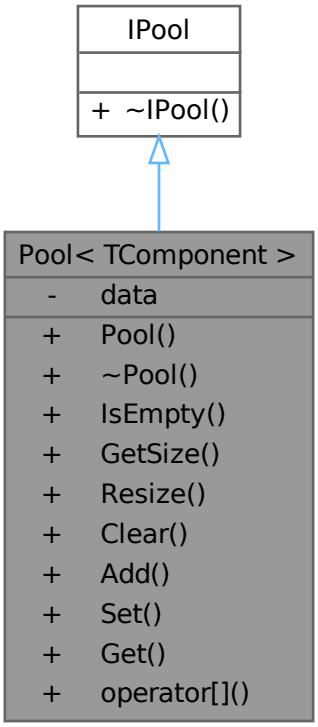
- src/Components/PolygonColliderComponent.hpp

4.32 Pool< TComponent > Class Template Reference

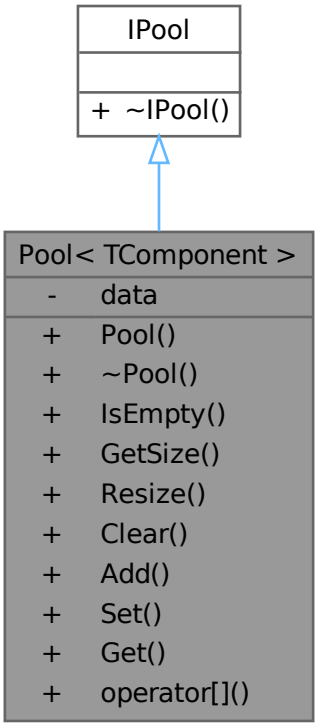
Template class managing a pool of components of type TComponent.

```
#include <Pool.hpp>
```

Inheritance diagram for Pool< TComponent >:



Collaboration diagram for Pool< TComponent >:



Public Member Functions

- Pool (int size=1000)

  Constructs a Pool with an optional initial size.
- virtual ~Pool ()=default

  Default destructor.
- bool IsEmpty () const

  Checks if the pool is empty.
- int GetSize () const

  Gets the current number of components in the pool.
- void Resize (int n)

  Resizes the pool to hold a specified number of components.
- void Clear ()

  Clears all components from the pool.
- void Add (TComponent object)

  Adds a new component to the end of the pool.
- void Set (int index, TComponent object)

  Sets the component at a given index.
- TComponent & Get (unsigned int index)

  Accesses the component at a given index.
- TComponent & operator[ ] (unsigned int index)

  Overloaded operator[] for component access.

Public Member Functions inherited from IPool

- virtual ~IPool ()=default

  Virtual destructor for cleanup in derived classes.

Private Attributes

- std::vector< TComponent > data

  Internal storage for components.

4.32.1 Detailed Description

template<typename TComponent>
class Pool< TComponent >

Template class managing a pool of components of type TComponent.

Template Parameters

TComponent	Type of component stored in the pool.

4.32.2 Constructor & Destructor Documentation

4.32.2.1 Pool()

```
template<typename TComponent >
Pool< TComponent >::Pool (
            int size = 1000 )  [inline]
```

Constructs a Pool with an optional initial size.

Parameters

size	Initial number of components in the pool (default 1000).

4.32.2.2 ~Pool()

```
template<typename TComponent >
virtual Pool< TComponent >::~Pool ( )  [virtual], [default]
```

Default destructor.

4.32.3 Member Function Documentation

4.32.3.1 Add()

```
template<typename TComponent >
void Pool< TComponent >::Add (
            TComponent object )  [inline]
```

Adds a new component to the end of the pool.

Parameters

object	Component to add.

4.32.3.2 Clear()

```
template<typename TComponent >
void Pool< TComponent >::Clear ( )  [inline]
```

Clears all components from the pool.

4.32.3.3 Get()

```
template<typename TComponent >
TComponent & Pool< TComponent >::Get (
            unsigned int index )  [inline]
```

Accesses the component at a given index.

Parameters

index	Index of the component.

Returns

Reference to the component.

4.32.3.4 GetSize()

```
template<typename TComponent >
int Pool< TComponent >::GetSize ( ) const  [inline]
```

Gets the current number of components in the pool.

Returns

Number of components.

4.32.3.5 IsEmpty()

```
template<typename TComponent >
bool Pool< TComponent >::IsEmpty ( ) const  [inline]
```

Checks if the pool is empty.

Returns

true if the pool has no components, false otherwise.

4.32.3.6 operator[]()

```
template<typename TComponent >
TComponent & Pool< TComponent >::operator[] (
            unsigned int index )  [inline]
```

Overloaded operator[] for component access.

Parameters

index	Index of the component.

Returns

Reference to the component.

4.32.3.7 Resize()

```
template<typename TComponent >
void Pool< TComponent >::Resize (
            int n )  [inline]
```

Resizes the pool to hold a specified number of components.

Parameters

n	New size of the pool.

4.32.3.8 Set()

```
template<typename TComponent >
void Pool< TComponent >::Set (
            int index,
            TComponent object )  [inline]
```

Sets the component at a given index.

Parameters

index	Index to set the component at.
object	Component to set.

4.32.4 Member Data Documentation

4.32.4.1 data

```
template<typename TComponent >
std::vector<TComponent> Pool< TComponent >::data  [private]
```

Internal storage for components.

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

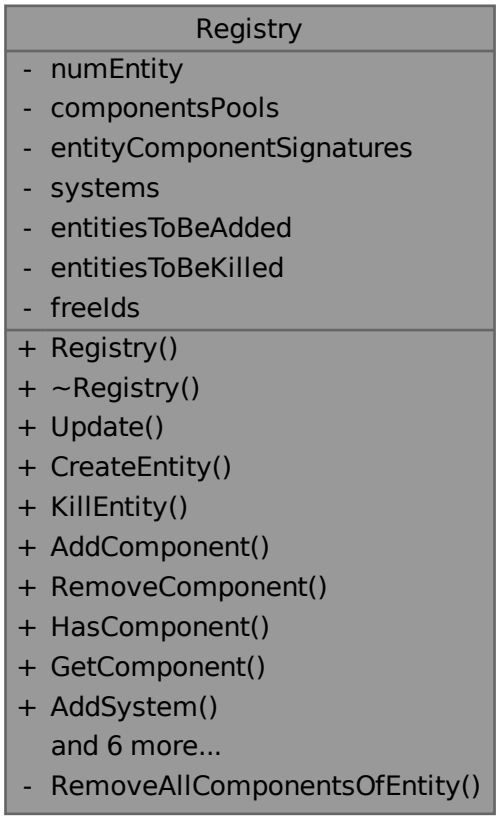
- src/Utils/Pool.hpp

4.33 Registry Class Reference

Manages entities, components, and systems in the ECS.

```
#include <ECS.hpp>
```

Collaboration diagram for Registry:



Public Member Functions

- Registry ()
- ~Registry ()
- void Update ()

  Processes entity addition and removal, updating systems accordingly.
- Entity CreateEntity ()

  Creates and returns a new entity.
- void KillEntity (Entity entity)

  Marks an entity for removal.
- template<typename TComponent , typename... TArgs>
  void AddComponent (Entity entity, TArgs &&... args)

  Adds a component of type TComponent to an entity.
- template<typename TComponent >
  void RemoveComponent (Entity entity)

  Removes a component of type TComponent from an entity.
- template<typename TComponent >
  bool HasComponent (Entity entity) const

  Checks if an entity has a component of type TComponent.

- template<typename TComponent >
  TComponent & GetComponent (Entity entity) const

  Returns a reference to the component of type TComponent attached to an entity.
- template<typename TSystem , typename... TArgs>
  void AddSystem (TArgs &&... args)

  Adds a system of type TSystem to the registry.
- template<typename TSystem >
  void RemoveSystem ()

  Removes a system of type TSystem from the registry.
- template<typename TSystem >
  bool HasSystem () const

  Checks if a system of type TSystem is registered.
- template<typename TSystem >
  TSystem & GetSystem () const

  Returns a reference to the system of type TSystem.
- void AddEntityToSystems (Entity entity)

  Adds an entity to all systems whose signatures match the entity's components.
- void RemoveEntityFromSystems (Entity entity)

  Removes an entity from all systems.
- void ClearAllEntities ()

  Removes all entities and clears the registry.

Private Member Functions

- void RemoveAllComponentsOfEntity (Entity entity)

  Removes all components of a given entity.

Private Attributes

- int numEntity = 0

  Number of entities currently created.
- std::vector< std::shared_ptr< IPool > > componentsPools

  Pools storing components indexed by component ID.
- std::vector< Signature > entityComponentSignatures

  Signatures of each entity indicating which components they have.
- std::unordered_map< std::type_index, std::shared_ptr< System > > systems

  Systems managed by the registry indexed by type.
- std::set< Entity > entitiesToBeAdded

  Entities pending addition.
- std::set< Entity > entitiesToBeKilled

  Entities pending removal.
- std::deque< int > freeIds

  Queue of free entity IDs for reuse.

4.33.1 Detailed Description

Manages entities, components, and systems in the ECS.

4.33.2 Constructor & Destructor Documentation

4.33.2.1 Registry()

```
Registry::Registry ( )
```

4.33.2.2 ~Registry()

```
Registry::~Registry ( )
```

4.33.3 Member Function Documentation

4.33.3.1 AddComponent()

```
template<typename TComponent , typename...  TArgs>
void Registry::AddComponent (
            Entity entity,
            TArgs &&...  args )
```

Adds a component of type TComponent to an entity.

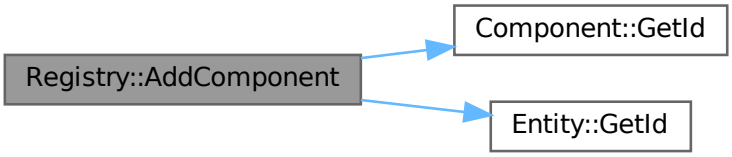
Template Parameters

TComponent	The component type.
TArgs	Arguments to forward to the component's constructor.

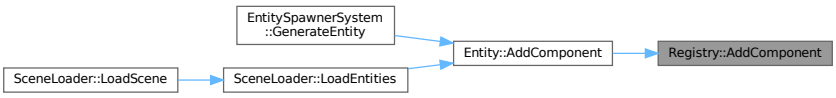
Parameters

entity	The entity to add the component to.
args	Constructor arguments for the component.

Here is the call graph for this function:



Here is the caller graph for this function:



4.33.3.2 AddEntityToSystems()

```
void Registry::AddEntityToSystems (
            Entity entity )
```

Adds an entity to all systems whose signatures match the entity's components.

Parameters

entity	The entity to add.

Here is the call graph for this function:

Here is the caller graph for this function:

4.33.3.3 AddSystem()

```
template<typename TSystem , typename...  TArgs>
void Registry::AddSystem (
            TArgs &&...  args )
```

Adds a system of type TSystem to the registry.

Template Parameters

TSystem	The system type.
TArgs	Arguments to forward to the system's constructor.

Parameters

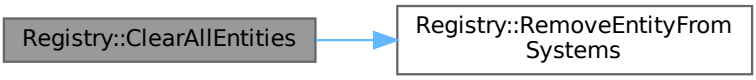
args	Constructor arguments for the system.

4.33.3.4 ClearAllEntities()

```
void Registry::ClearAllEntities ( )
```

Removes all entities and clears the registry.

Here is the call graph for this function:



4.33.3.5 CreateEntity()

```
Entity Registry::CreateEntity ( )
```

Creates and returns a new entity.

Returns

The newly created entity.

4.33.3.6 GetComponent()

```
template<typename TComponent >
TComponent & Registry::GetComponent (
             Entity entity ) const
```

Returns a reference to the component of type TComponent attached to an entity.

Template Parameters

TComponent	The component type.

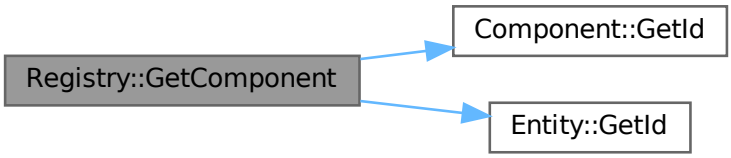
Parameters

entity	The entity.

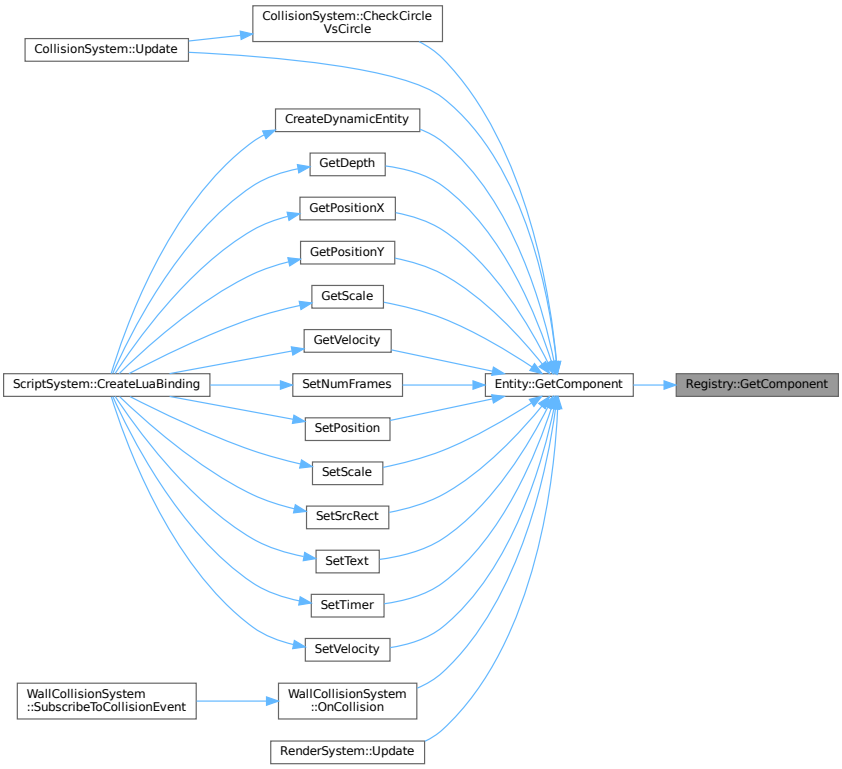
Returns

Reference to the component.

Here is the call graph for this function:



Here is the caller graph for this function:



4.33.3.7 GetSystem()

```
template<typename TSystem >
TSystem & Registry::GetSystem ( ) const
```

Returns a reference to the system of type TSystem.

Template Parameters

TSystem	The system type.

Returns

Reference to the system.

4.33.3.8 HasComponent()

```
template<typename TComponent >
bool Registry::HasComponent (
            Entity entity ) const
```

Checks if an entity has a component of type TComponent.

Template Parameters

TComponent	The component type.

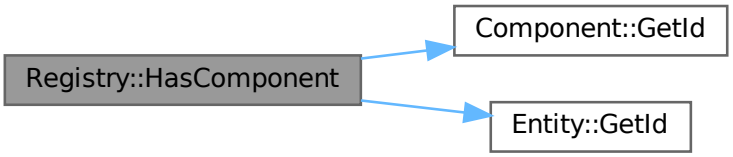
Parameters

entity	The entity to check.

Returns

True if the entity has the component, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:

4.33.3.9 HasSystem()

```
template<typename TSystem >
bool Registry::HasSystem ( ) const
```

Checks if a system of type TSystem is registered.

Template Parameters

TSystem	The system type.

Returns

True if the system is present, false otherwise.

4.33.3.10 KillEntity()

```
void Registry::KillEntity (
            Entity entity )
```

Marks an entity for removal.

Parameters

entity	The entity to kill.

Here is the caller graph for this function:

4.33.3.11 RemoveAllComponentsOfEntity()

```
void Registry::RemoveAllComponentsOfEntity (
            Entity entity )  [private]
```

Removes all components of a given entity.

Parameters

entity	The entity whose components to remove.

4.33.3.12 RemoveComponent()

```
template<typename TComponent >
void Registry::RemoveComponent (
            Entity entity )
```

Removes a component of type TComponent from an entity.

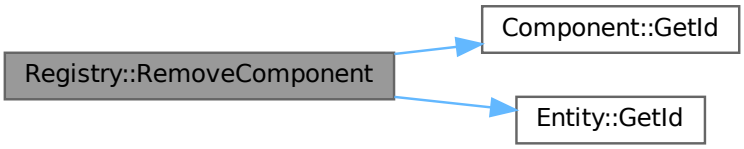
Template Parameters

TComponent	The component type.

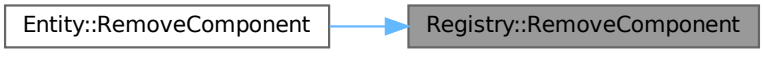
Parameters

entity	The entity to remove the component from.

Here is the call graph for this function:



Here is the caller graph for this function:



4.33.3.13 RemoveEntityFromSystems()

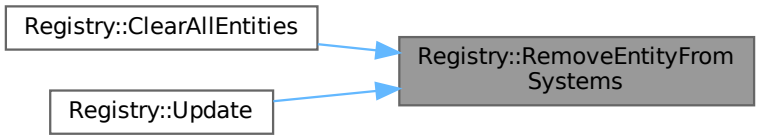
```
void Registry::RemoveEntityFromSystems (
            Entity entity )
```

Removes an entity from all systems.

Parameters

entity	The entity to remove.

Here is the caller graph for this function:



4.33.3.14 RemoveSystem()

```
template<typename TSystem >
void Registry::RemoveSystem ( )
```

Removes a system of type TSystem from the registry.

Template Parameters

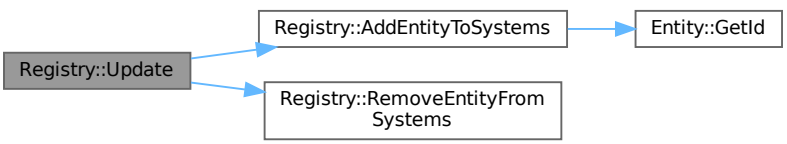
TSystem	The system type.

4.33.3.15 Update()

```
void Registry::Update ( )
```

Processes entity addition and removal, updating systems accordingly.

Here is the call graph for this function:



4.33.4 Member Data Documentation

4.33.4.1 componentsPools

```
std::vector<std::shared_ptr<IPool> > Registry::componentsPools  [private]
```

Pools storing components indexed by component ID.

4.33.4.2 entitiesToBeAdded

```
std::set<Entity> Registry::entitiesToBeAdded  [private]
```

Entities pending addition.

4.33.4.3 entitiesToBeKilled

```
std::set<Entity> Registry::entitiesToBeKilled  [private]
```

Entities pending removal.

4.33.4.4 entityComponentSignatures

```
std::vector<Signature> Registry::entityComponentSignatures  [private]
```

Signatures of each entity indicating which components they have.

4.33.4.5 freeIds

```
std::deque<int> Registry::freeIds  [private]
```

Queue of free entity IDs for reuse.

4.33.4.6 numEntity

```
int Registry::numEntity = 0  [private]
```

Number of entities currently created.

4.33.4.7 systems

```
std::unordered_map<std::type_index, std::shared_ptr<System> > Registry::systems  [private]
```

Systems managed by the registry indexed by type.

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

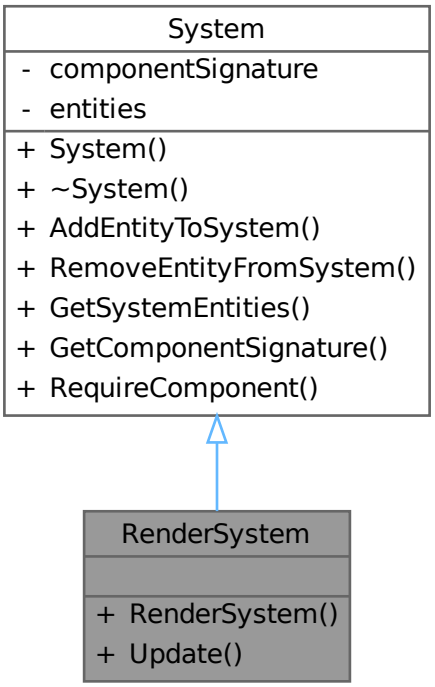
- src/ECS/ECS.hpp
- src/ECS/ECS.cpp

4.34 RenderSystem Class Reference

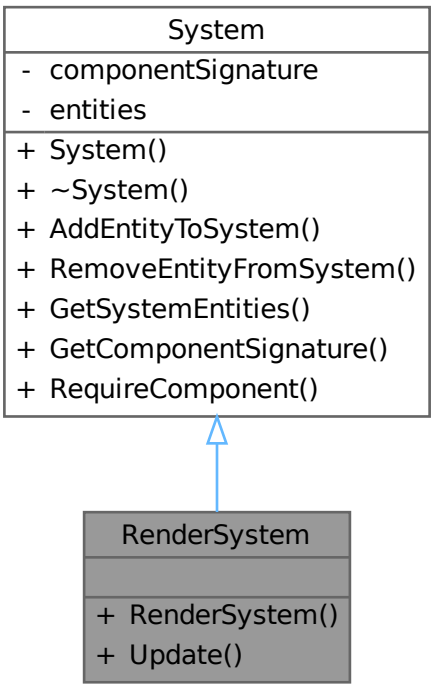
System responsible for rendering entities with sprites on the screen.

`#include <RenderSystem.hpp>`

Inheritance diagram for RenderSystem:



Collaboration diagram for RenderSystem:



Public Member Functions

- RenderSystem ()

  Constructs a RenderSystem and requires SpriteComponent and TransformComponent.
- void Update (SDL_Renderer *renderer, const std::unique_ptr< AssetManager > &AssetManager)

  Renders all entities with SpriteComponent and TransformComponent.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.34.1 Detailed Description

System responsible for rendering entities with sprites on the screen.

This system requires entities to have SpriteComponent and TransformComponent. It sorts entities based on their DepthComponent (if present) to render in the correct order.

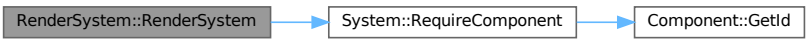
4.34.2 Constructor & Destructor Documentation

4.34.2.1 RenderSystem()

```
RenderSystem::RenderSystem ( )  [inline]
```

Constructs a RenderSystem and requires SpriteComponent and TransformComponent.

Here is the call graph for this function:



4.34.3 Member Function Documentation

4.34.3.1 Update()

```
void RenderSystem::Update (
            SDL_Renderer * renderer,
            const std::unique_ptr< AssetManager > & AssetManager )  [inline]
```

Renders all entities with SpriteComponent and TransformComponent.

Entities are sorted by their DepthComponent (if any) to respect drawing order.

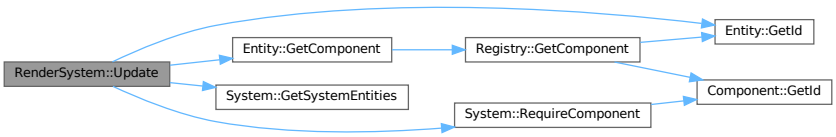
Parameters

renderer	Pointer to the SDL_Renderer used for rendering.
AssetManager	Shared pointer to the AssetManager for texture access.

For each entity:

- Calculate destination rectangle using position, scale, and sprite size.
- Render the sprite texture with rotation.

Here is the call graph for this function:



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

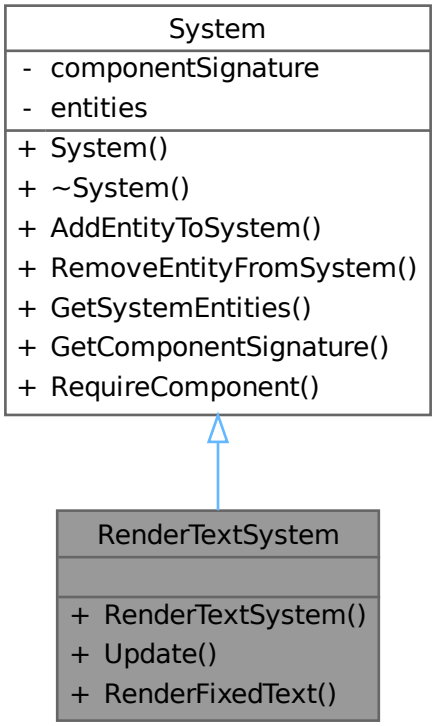
- src/Systems/RenderSystem.hpp

4.35 RenderTextSystem Class Reference

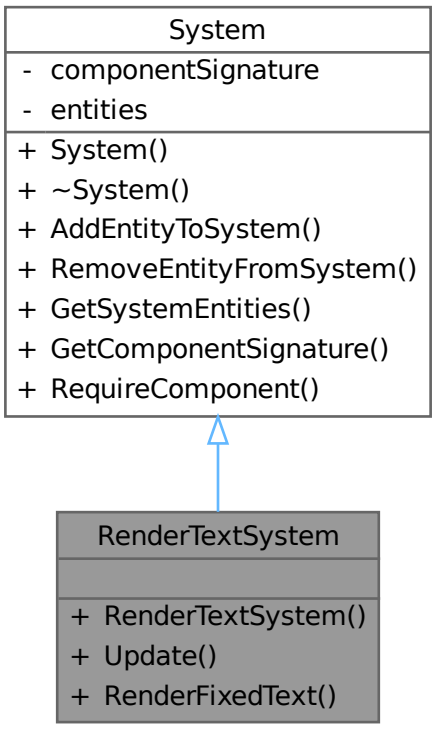
System responsible for rendering text components in the ECS.

`#include <RenderTextSystem.hpp>`

Inheritance diagram for RenderTextSystem:



Collaboration diagram for RenderTextSystem:



Public Member Functions

- RenderTextSystem ()

  Constructs a RenderTextSystem and requires TextComponent and TransformComponent.
- void Update (SDL_Renderer *renderer, const std::unique_ptr< AssetManager > &assetManager)

  Updates and renders all text entities.
- void RenderFixedText (SDL_Renderer *renderer, TTF_Font *font, const std::string &text, SDL_Color color, int x, int y, float scaleX=1.0f, float scaleY=1.0f)

  Renders fixed text at specified screen coordinates with optional scaling.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

Returns the component signature required by this system.

- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.35.1 Detailed Description

System responsible for rendering text components in the ECS.

Requires entities to have TextComponent and TransformComponent.

4.35.2 Constructor & Destructor Documentation

4.35.2.1 RenderTextSystem()

```
RenderTextSystem::RenderTextSystem ( )  [inline]
```

Constructs a RenderTextSystem and requires TextComponent and TransformComponent.

Here is the call graph for this function:

4.35.3 Member Function Documentation

4.35.3.1 RenderFixedText()

```
void RenderTextSystem::RenderFixedText (
            SDL_Renderer * renderer,
            TTF_Font * font,
            const std::string & text,
            SDL_Color color,
            int x,
            int y,
            float scaleX = 1.0f,
            float scaleY = 1.0f )  [inline]
```

Renders fixed text at specified screen coordinates with optional scaling.

This function does not depend on ECS entities.

Parameters

renderer	Pointer to the SDL_Renderer.
font	Pointer to the TTF_Font to use.
text	The text string to render.
color	The SDL_Color to render the text with.
x	The x position on screen.
y	The y position on screen.
scaleX	Optional horizontal scale factor (default 1.0)

Here is the call graph for this function:

4.35.3.2 Update()

```
void RenderTextSystem::Update (
            SDL_Renderer * renderer,
            const std::unique_ptr< AssetManager > & assetManager )  [inline]
```

Updates and renders all text entities.

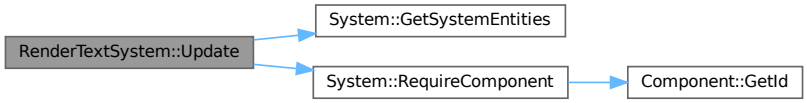
For each entity:

- Renders the text surface with the font and color specified in TextComponent.
- Creates a texture from the surface.
- Uses the TransformComponent for positioning and scaling.
- Renders the texture on the screen.
- Frees the texture and surface after rendering.

Parameters

renderer	Pointer to the SDL_Renderer used for rendering.
assetManager	Shared pointer to AssetManager to access fonts.

Here is the call graph for this function:



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

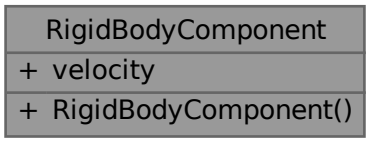
- src/Systems/RenderTextSystem.hpp

4.36 RigidBodyComponent Struct Reference

Stores the velocity vector of an entity's rigid body.

`#include <RigidBodyComponent.hpp>`

Collaboration diagram for RigidBodyComponent:



Public Member Functions

- RigidBodyComponent (glm::vec2 velocity=glm::vec2(0.0))
  Construct a new RigidBodyComponent object.

Public Attributes

- glm::vec2 velocity

4.36.1 Detailed Description

Stores the velocity vector of an entity's rigid body.

4.36.2 Constructor & Destructor Documentation

4.36.2.1 RigidBodyComponent()

```
RigidBodyComponent::RigidBodyComponent (
            glm::vec2 velocity = glm::vec2(0.0) )  [inline]
```

Construct a new RigidBodyComponent object.

Parameters

velocity	Initial velocity vector (default is zero vector).

4.36.3 Member Data Documentation

4.36.3.1 velocity

```
glm::vec2 RigidBodyComponent::velocity
```

Velocity vector (x, y) of the rigid body

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

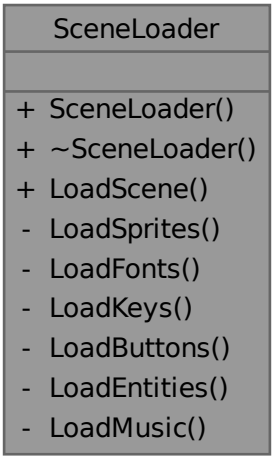
- src/Components/RigidBodyComponent.hpp

4.37 SceneLoader Class Reference

Loads a game scene from a Lua script.

```
#include <SceneLoader.hpp>
```

Collaboration diagram for SceneLoader:



Public Member Functions

- SceneLoader ()
- ~SceneLoader ()
- void LoadScene (const std::string &scenePath, sol::state &lua, std::unique_ptr< AssetManager > &asset↩ Manager, std::unique_ptr< ControllerManager > &controllerManager, std::unique_ptr< Registry > &registry, SDL_Renderer *renderer)

  Loads a full scene from a Lua file path, setting up assets, controls, entities, and music.

Private Member Functions

- void LoadSprites (SDL_Renderer *render, const sol::table &sprites, std::unique_ptr< AssetManager > &assetManager)

  Loads sprite assets from Lua table into the AssetManager.
- void LoadFonts (const sol::table &fonts, std::unique_ptr< AssetManager > &assetManager)

  Loads font assets from Lua table into the AssetManager.
- void LoadKeys (const sol::table &keys, std::unique_ptr< ControllerManager > &controllerManager)

  Loads key bindings from Lua table into the ControllerManager.
- void LoadButtons (const sol::table &buttons, std::unique_ptr< ControllerManager > &controllerManager)

  Loads button bindings from Lua table into the ControllerManager.
- void LoadEntities (sol::state &lua, const sol::table &entites, std::unique_ptr< Registry > &registry)

  Loads entities from Lua table into the ECS Registry.
- void LoadMusic (const sol::table &musicTable, std::unique_ptr< AssetManager > &assetManager)

  Loads music assets from Lua table into the AssetManager.

4.37.1 Detailed Description

Loads a game scene from a Lua script.

This class is responsible for parsing Lua scripts to load game assets, entities, key and button bindings, and music into the appropriate managers and systems (AssetManager, ControllerManager, Registry, etc.).

4.37.2 Constructor & Destructor Documentation

4.37.2.1 SceneLoader()

```
SceneLoader::SceneLoader ( )
```

4.37.2.2 ~SceneLoader()

```
SceneLoader::~SceneLoader ( )
```

4.37.3 Member Function Documentation

4.37.3.1 LoadButtons()

```
void SceneLoader::LoadButtons (
            const sol::table & buttons,
            std::unique_ptr< ControllerManager > & controllerManager )  [private]
```

Loads button bindings from Lua table into the ControllerManager.

Parameters

buttons	Lua table with button mappings.
controllerManager	Unique pointer to ControllerManager instance.

Here is the caller graph for this function:

4.37.3.2 LoadEntities()

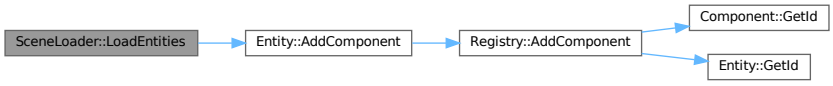
```
void SceneLoader::LoadEntities (
            sol::state & lua,
            const sol::table & entites,
            std::unique_ptr< Registry > & registry )  [private]
```

Loads entities from Lua table into the ECS Registry.

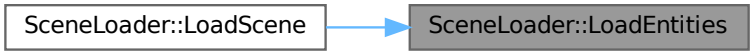
Parameters

lua	Reference to Lua state.
entities	Lua table containing entity definitions.
registry	Unique pointer to ECS Registry.

Here is the call graph for this function:



Here is the caller graph for this function:



4.37.3.3 LoadFonts()

```
void SceneLoader::LoadFonts (
            const sol::table & fonts,
            std::unique_ptr< AssetManager > & assetManager )  [private]
```

Loads font assets from Lua table into the AssetManager.

Parameters

fonts	Lua table containing font definitions.
assetManager	Unique pointer to the AssetManager instance.

Here is the caller graph for this function:

4.37.3.4 LoadKeys()

```
void SceneLoader::LoadKeys (
            const sol::table & keys,
            std::unique_ptr< ControllerManager > & controllerManager )  [private]
```

Loads key bindings from Lua table into the ControllerManager.

Parameters

keys	Lua table with key mappings.
controllerManager	Unique pointer to ControllerManager instance.

Here is the caller graph for this function:

4.37.3.5 LoadMusic()

```
void SceneLoader::LoadMusic (
            const sol::table & musicTable,
            std::unique_ptr< AssetManager > & assetManager )  [private]
```

Loads music assets from Lua table into the AssetManager.

Parameters

musicTable	Lua table containing music definitions.
assetManager	Unique pointer to the AssetManager instance.

Here is the caller graph for this function:

4.37.3.6 LoadScene()

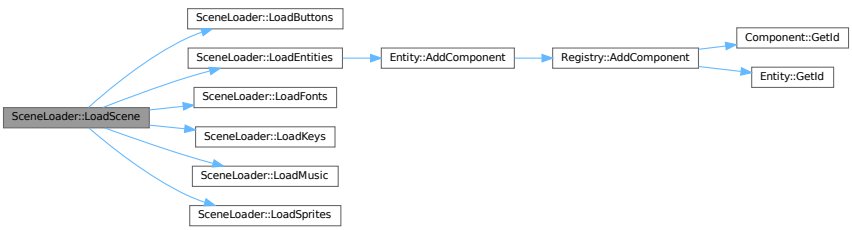
```
void SceneLoader::LoadScene (
            const std::string & scenePath,
            sol::state & lua,
            std::unique_ptr< AssetManager > & assetManager,
            std::unique_ptr< ControllerManager > & controllerManager,
            std::unique_ptr< Registry > & registry,
            SDL_Renderer * renderer )
```

Loads a full scene from a Lua file path, setting up assets, controls, entities, and music.

Parameters

scenePath	File path to the Lua scene script.
lua	Reference to the Lua scripting state.
assetManager	Unique pointer to AssetManager.
controllerManager	Unique pointer to ControllerManager.
registry	Unique pointer to ECS Registry.
renderer	SDL renderer pointer for texture creation.

Here is the call graph for this function:



4.37.3.7 LoadSprites()

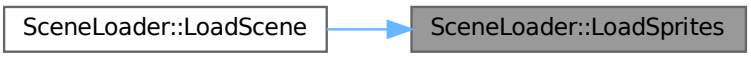
```
void SceneLoader::LoadSprites (
            SDL_Renderer * render,
            const sol::table & sprites,
            std::unique_ptr< AssetManager > & assetManager )  [private]
```

Loads sprite assets from Lua table into the AssetManager.

Parameters

render	SDL renderer used for texture creation.
sprites	Lua table containing sprite definitions.
assetManager	Unique pointer to the AssetManager instance.

Here is the caller graph for this function:



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

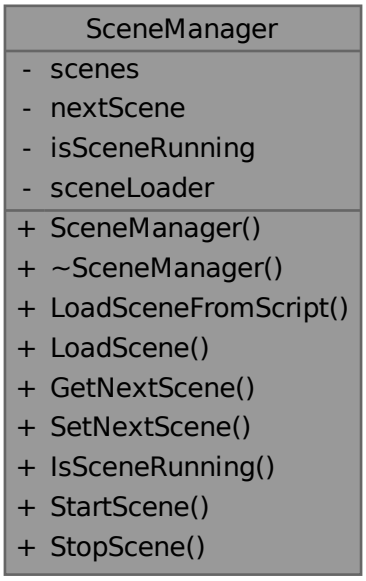
- src/SceneManager/SceneLoader.hpp
- src/SceneManager/SceneLoader.cpp

4.38 SceneManager Class Reference

Manages scenes and handles scene transitions.

```
#include <SceneManager.hpp>
```

Collaboration diagram for SceneManager:



Public Member Functions

- SceneManager ()

  Pointer to the SceneLoader used for loading scenes.
- ~SceneManager ()

  Destructs the SceneManager.
- void LoadSceneFromScript (const std::string &path, sol::state &lua)

  Loads a scene from a Lua script file.
- void LoadScene ()

  Loads the next scene based on internal state.
- std::string GetNextScene () const

  Returns the name of the next scene to be loaded.
- void SetNextScene (const std::string &nextScene)

  Sets the name of the next scene to be loaded.
- bool IsSceneRunning () const

  Checks whether a scene is currently running.
- void StartScene ()

  Marks the beginning of a scene's execution.
- void StopScene ()

  Stops the currently running scene.

Private Attributes

- std::map< std::string, std::string > scenes
- std::string nextScene

  Map of scene names to script paths.
- bool isSceneRunning = false

  Name of the next scene to load.
- std::unique_ptr< SceneLoader > sceneLoader

  Flag indicating if a scene is currently running.

4.38.1 Detailed Description

Manages scenes and handles scene transitions.

4.38.2 Constructor & Destructor Documentation

4.38.2.1 SceneManager()

```
SceneManager::SceneManager ( )
```

Pointer to the SceneLoader used for loading scenes.

Constructs the SceneManager.

4.38.2.2 ~SceneManager()

```
SceneManager::~SceneManager ( )
```

Destructs the SceneManager.

4.38.3 Member Function Documentation

4.38.3.1 GetNextScene()

```
std::string SceneManager::GetNextScene ( ) const
```

Returns the name of the next scene to be loaded.

Returns

The next scene's name.

4.38.3.2 IsSceneRunning()

```
bool SceneManager::IsSceneRunning ( ) const
```

Checks whether a scene is currently running.

Returns

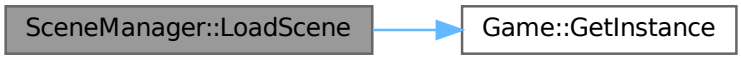
True if a scene is running, false otherwise.

4.38.3.3 LoadScene()

```
void SceneManager::LoadScene ( )
```

Loads the next scene based on internal state.

Here is the call graph for this function:



4.38.3.4 LoadSceneFromScript()

```
void SceneManager::LoadSceneFromScript (
            const std::string & path,
            sol::state & lua )
```

Loads a scene from a Lua script file.

Parameters

path	The path to the Lua script.
lua	The Lua state.

4.38.3.5 SetNextScene()

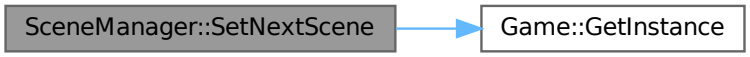
```
void SceneManager::SetNextScene (
            const std::string & nextScene )
```

Sets the name of the next scene to be loaded.

Parameters

nextScene	The next scene's name.

Here is the call graph for this function:



4.38.3.6 StartScene()

```
void SceneManager::StartScene ( )
```

Marks the beginning of a scene's execution.

4.38.3.7 StopScene()

```
void SceneManager::StopScene ( )
```

Stops the currently running scene.

4.38.4 Member Data Documentation

4.38.4.1 isSceneRunning

```
bool SceneManager::isSceneRunning = false  [private]
```

Name of the next scene to load.

4.38.4.2 nextScene

```
std::string SceneManager::nextScene  [private]
```

Map of scene names to script paths.

4.38.4.3 sceneLoader

```
std::unique_ptr<SceneLoader> SceneManager::sceneLoader  [private]
```

Flag indicating if a scene is currently running.

4.38.4.4 scenes

```
std::map<std::string, std::string> SceneManager::scenes  [private]
```

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

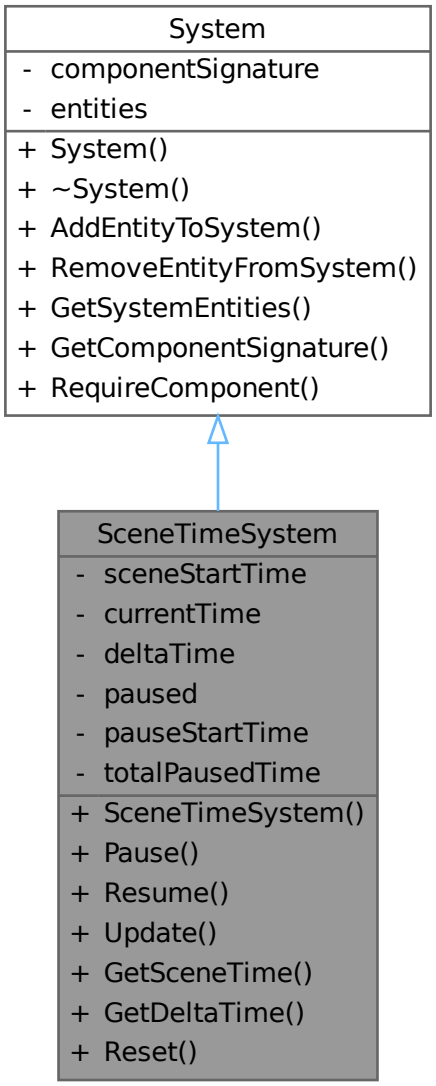
- src/SceneManager/SceneManager.hpp
- src/SceneManager/SceneManager.cpp

4.39 SceneTimeSystem Class Reference

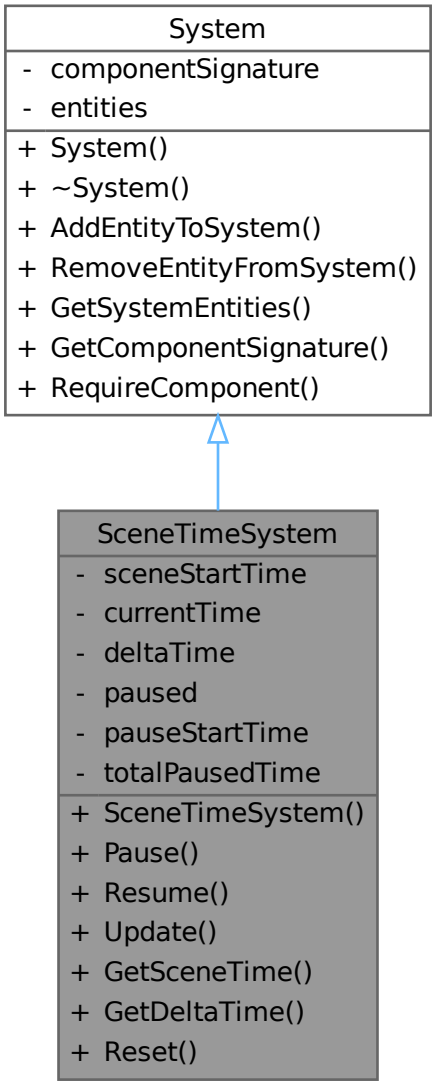
Manages scene timing including pause, resume, delta time, and total elapsed time.

```
#include <SceneTimeSystem.hpp>
```

Inheritance diagram for SceneTimeSystem:



Collaboration diagram for SceneTimeSystem:



Public Member Functions

- SceneTimeSystem ()

  Constructs the SceneTimeSystem and initializes timing variables.
- void Pause ()

  Pauses the scene timer.
- void Resume ()

  Resumes the scene timer after being paused.
- void Update ()

  Updates the delta time and current time if not paused.
- int GetSceneTime () const

Gets the total elapsed scene time, excluding paused durations.

- int GetDeltaTime () const

  Gets the delta time between last two updates.

- void Reset ()

  Resets the timer and all related variables.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.

- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.

- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.

- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.

- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

Private Attributes

- int sceneStartTime

  Time when the scene started (in milliseconds).

- int currentTime

  Current time (in milliseconds).

- int deltaTime

  Time difference between last two updates (in milliseconds).

- bool paused

  Flag indicating if the timer is paused.

- int pauseStartTime

  Time when the pause started.

- int totalPausedTime

  Total time spent paused.

4.39.1 Detailed Description

Manages scene timing including pause, resume, delta time, and total elapsed time.

Keeps track of elapsed time since scene start, accounting for paused durations.

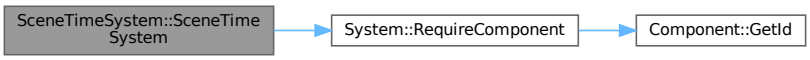
4.39.2 Constructor & Destructor Documentation

4.39.2.1 SceneTimeSystem()

```
SceneTimeSystem::SceneTimeSystem ( )  [inline]
```

Constructs the SceneTimeSystem and initializes timing variables.

Here is the call graph for this function:



4.39.3 Member Function Documentation

4.39.3.1 GetDeltaTime()

```
int SceneTimeSystem::GetDeltaTime ( ) const  [inline]
```

Gets the delta time between last two updates.

Returns

The delta time in milliseconds.

4.39.3.2 GetSceneTime()

```
int SceneTimeSystem::GetSceneTime ( ) const  [inline]
```

Gets the total elapsed scene time, excluding paused durations.

Returns

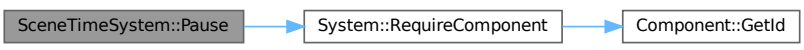
The elapsed time since scene start in milliseconds.

4.39.3.3 Pause()

```
void SceneTimeSystem::Pause ( )  [inline]
```

Pauses the scene timer.

Records the time when pause starts. Here is the call graph for this function:

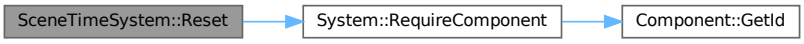


4.39.3.4 Reset()

```
void SceneTimeSystem::Reset ( )  [inline]
```

Resets the timer and all related variables.

Here is the call graph for this function:



4.39.3.5 Resume()

```
void SceneTimeSystem::Resume ( )  [inline]
```

Resumes the scene timer after being paused.

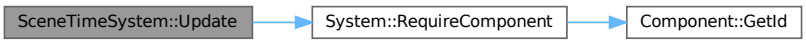
Updates total paused time and current time accordingly. Here is the call graph for this function:

4.39.3.6 Update()

```
void SceneTimeSystem::Update ( )  [inline]
```

Updates the delta time and current time if not paused.

If paused, delta time is set to zero. Here is the call graph for this function:



4.39.4 Member Data Documentation

4.39.4.1 currentTime

```
int SceneTimeSystem::currentTime  [private]
```

Current time (in milliseconds).

4.39.4.2 deltaTime

```
int SceneTimeSystem::deltaTime  [private]
```

Time difference between last two updates (in milliseconds).

4.39.4.3 paused

```
bool SceneTimeSystem::paused  [private]
```

Flag indicating if the timer is paused.

4.39.4.4 pauseStartTime

```
int SceneTimeSystem::pauseStartTime  [private]
```

Time when the pause started.

4.39.4.5 sceneStartTime

```
int SceneTimeSystem::sceneStartTime  [private]
```

Time when the scene started (in milliseconds).

4.39.4.6 totalPausedTime

```
int SceneTimeSystem::totalPausedTime  [private]
```

Total time spent paused.

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

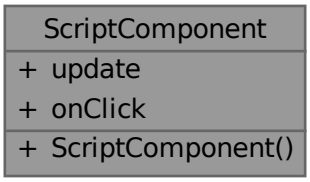
- src/Systems/SceneTimeSystem.hpp

4.40 ScriptComponent Struct Reference

Holds Lua functions for updating and click handling scripts.

`#include <ScriptComponent.hpp>`

Collaboration diagram for ScriptComponent:



Public Member Functions

- ScriptComponent (sol::function update=sol::lua_nil, sol::function onClick=sol::lua_nil)

  Construct a new ScriptComponent object.

Public Attributes

- sol::function update
- sol::function onClick

4.40.1 Detailed Description

Holds Lua functions for updating and click handling scripts.

4.40.2 Constructor & Destructor Documentation

4.40.2.1 ScriptComponent()

```
ScriptComponent::ScriptComponent (
            sol::function update = sol::lua_nil,
            sol::function onClick = sol::lua_nil )  [inline]
```

Construct a new ScriptComponent object.

Parameters

update	Lua function for updating the entity (default nil).
onClick	Lua function for click event handling (default nil).

4.40.3 Member Data Documentation

4.40.3.1 onClick

```
sol::function ScriptComponent::onClick
```

Lua function called when the entity is clicked

4.40.3.2 update

```
sol::function ScriptComponent::update
```

Lua function called every frame to update entity

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

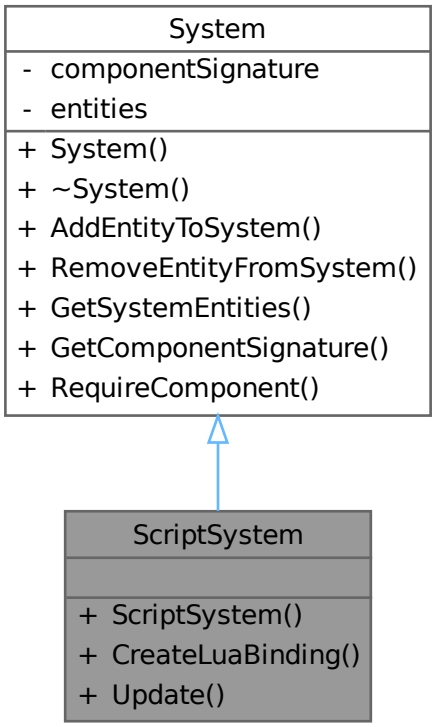
- src/Components/ScriptComponent.hpp

4.41 ScriptSystem Class Reference

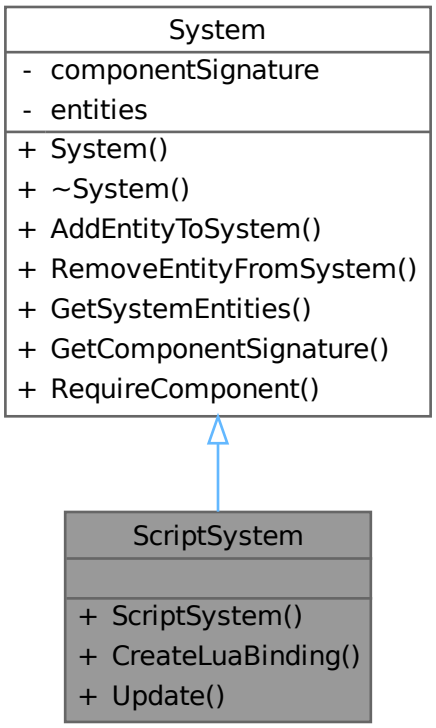
Manages entities with scripts and handles Lua binding and script updates.

`#include <ScriptSystem.hpp>`

Inheritance diagram for ScriptSystem:



Collaboration diagram for ScriptSystem:



Public Member Functions

- ScriptSystem ()

  Constructs the ScriptSystem and requires ScriptComponent.
- void CreateLuaBinding (sol::state &lua)

  Creates the Lua binding for functions and classes accessible from Lua scripts.
- void Update (sol::state &lua)

  Updates all entities with scripts by calling their Lua update functions.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.41.1 Detailed Description

Manages entities with scripts and handles Lua binding and script updates.

This system requires entities to have a ScriptComponent. It binds C++ functions and classes to Lua scripts and updates the script logic each frame.

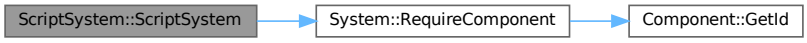
4.41.2 Constructor & Destructor Documentation

4.41.2.1 ScriptSystem()

```
ScriptSystem::ScriptSystem ( )  [inline]
```

Constructs the ScriptSystem and requires ScriptComponent.

Here is the call graph for this function:



4.41.3 Member Function Documentation

4.41.3.1 CreateLuaBinding()

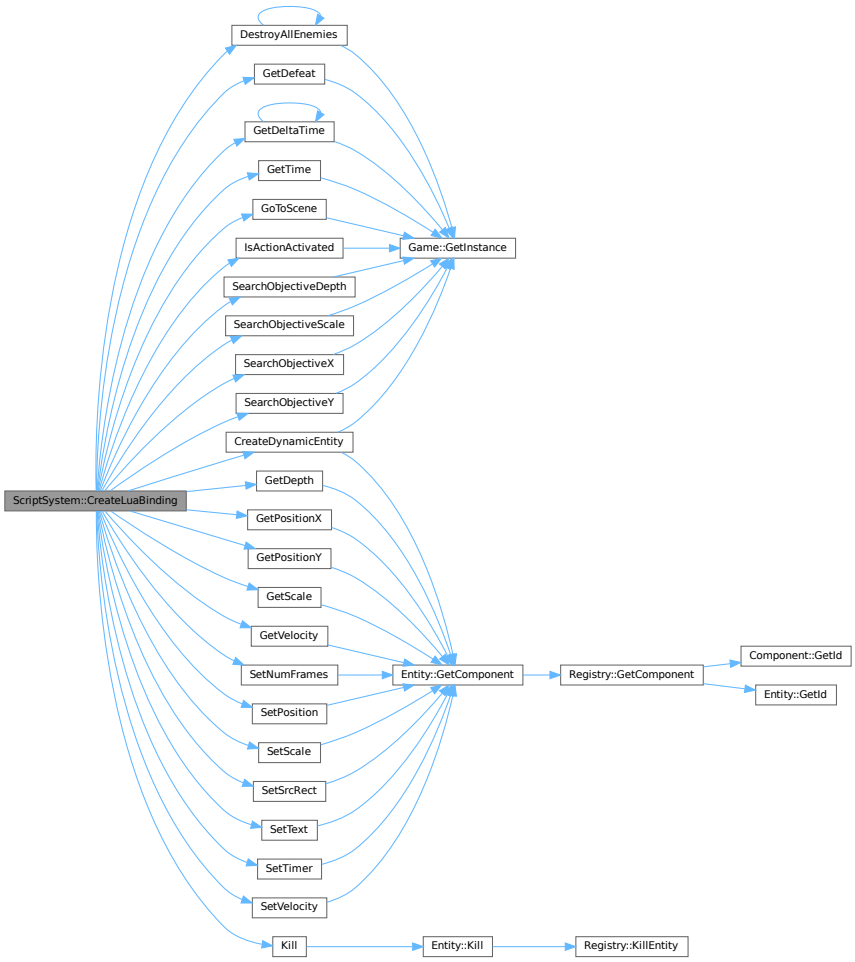
```
void ScriptSystem::CreateLuaBinding (
            sol::state & lua )  [inline]
```

Creates the Lua binding for functions and classes accessible from Lua scripts.

Parameters

lua	Reference to the Lua state to bind functions and classes.

Here is the call graph for this function:



4.41.3.2 Update()

```
void ScriptSystem::Update (
            sol::state & lua )  [inline]
```

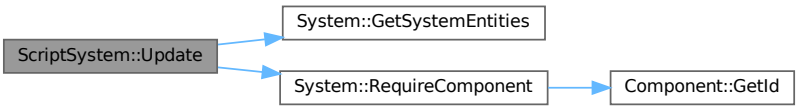
Updates all entities with scripts by calling their Lua update functions.

Sets the Lua global variable "this" to the current entity before calling the update function.

Parameters

lua	Reference to the Lua state used to run scripts.

Here is the call graph for this function:



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

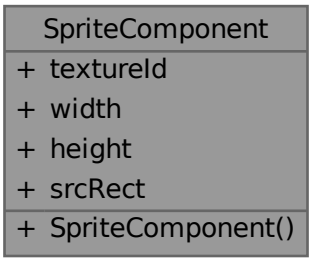
- src/Systems/ScriptSystem.hpp

4.42 SpriteComponent Struct Reference

Holds data for rendering a sprite, including texture ID and source rectangle.

`#include <SpriteComponent.hpp>`

Collaboration diagram for SpriteComponent:



Public Member Functions

- SpriteComponent (const std::string &textureId="none", int width=0, int height=0, int srcRectX=0, int srcRectY=0)

  Construct a new SpriteComponent object.

Public Attributes

- std::string textureId
- int width
- int height
- SDL_Rect srcRect

4.42.1 Detailed Description

Holds data for rendering a sprite, including texture ID and source rectangle.

4.42.2 Constructor & Destructor Documentation

4.42.2.1 SpriteComponent()

```
SpriteComponent::SpriteComponent (
            const std::string & textureId = "none",
            int width = 0,
            int height = 0,
            int srcRectX = 0,
            int srcRectY = 0 )  [inline]
```

Construct a new SpriteComponent object.

Parameters

textureId	Identifier of the texture to use (default "none")
width	Width of the sprite (default 0)
height	Height of the sprite (default 0)
srcRectX	X coordinate of the source rectangle in the texture (default 0)
srcRectY	Y coordinate of the source rectangle in the texture (default 0)

4.42.3 Member Data Documentation

4.42.3.1 height

```
int SpriteComponent::height
```

Height of the sprite

4.42.3.2 srcRect

```
SDL_Rect SpriteComponent::srcRect
```

Source rectangle from the texture to render

4.42.3.3 textureId

```
std::string SpriteComponent::textureId
```

Identifier for the texture resource

4.42.3.4 width

```
int SpriteComponent::width
```

Width of the sprite

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

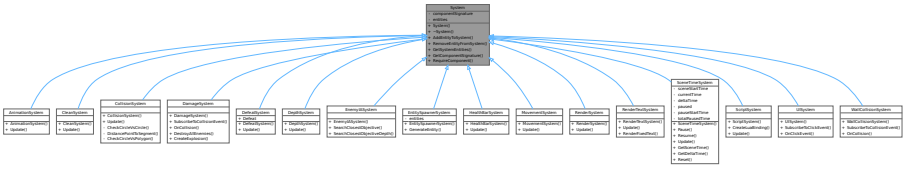
- src/Components/SpriteComponent.hpp

4.43 System Class Reference

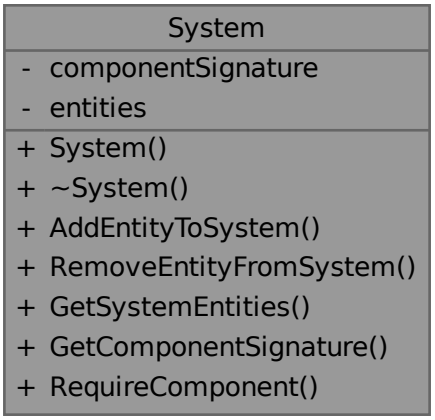
Base class for systems that operate on entities with specific components.

```
#include <ECS.hpp>
```

Inheritance diagram for System:



Collaboration diagram for System:



Public Member Functions

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

Private Attributes

- Signature componentSignature

  Signature representing the required components for this system.
- std::vector< Entity > entities

  List of entities currently managed by this system.

4.43.1 Detailed Description

Base class for systems that operate on entities with specific components.

4.43.2 Constructor & Destructor Documentation

4.43.2.1 System()

```
System::System ( )  [default]
```

4.43.2.2 ~System()

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

4.43.3 Member Function Documentation

4.43.3.1 AddEntityToSystem()

```
void System::AddEntityToSystem (
            Entity entity )
```

Adds an entity to this system.

Parameters

entity	The entity to add.

4.43.3.2 GetComponentSignature()

```
const Signature & System::GetComponentSignature ( ) const
```

Returns the component signature required by this system.

Returns

Component signature bitset.

4.43.3.3 GetSystemEntities()

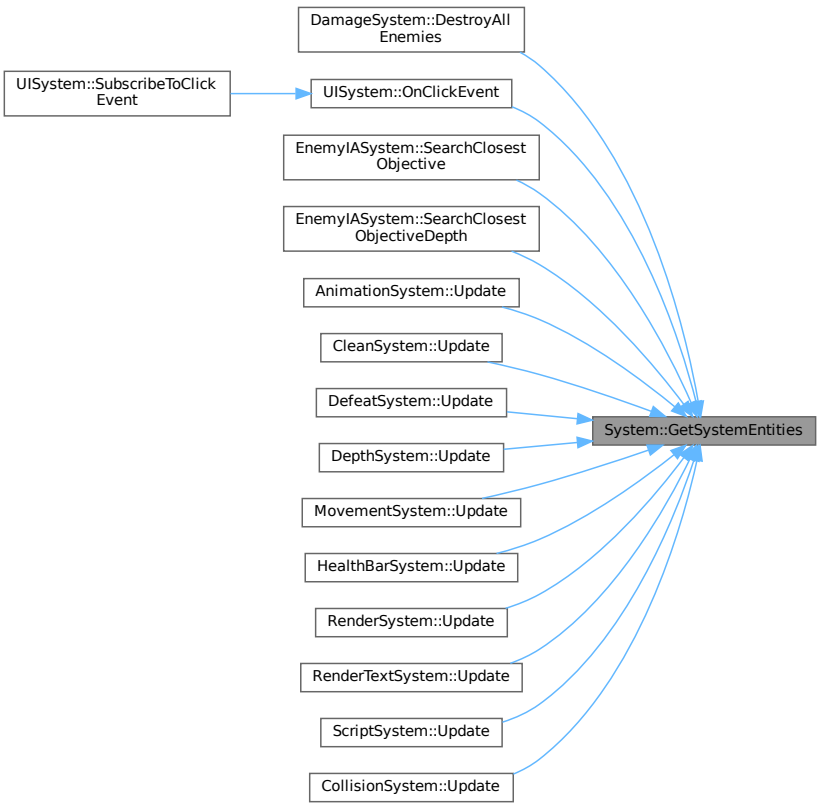
```
std::vector< Entity > System::GetSystemEntities ( ) const
```

Returns all entities currently registered in this system.

Returns

Vector of entities.

Here is the caller graph for this function:



4.43.3.4 RemoveEntityFromSystem()

```
void System::RemoveEntityFromSystem (
            Entity entity )
```

Removes an entity from this system.

Parameters

entity	The entity to remove.

4.43.3.5 RequireComponent()

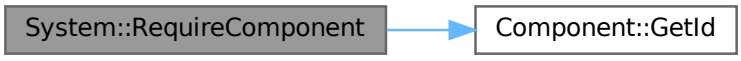
```
template<typename TComponent >
void System::RequireComponent ( )
```

Adds a required component type to the system's signature.

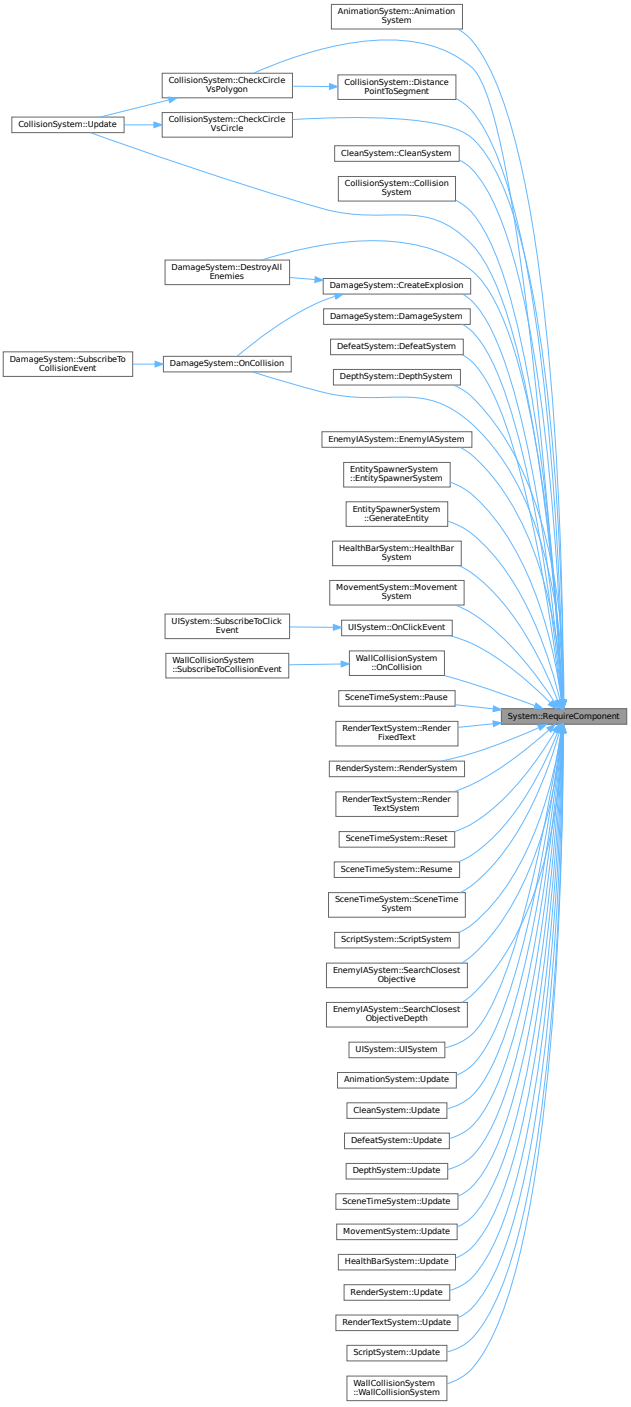
Template Parameters

TComponent	The component type required.

Here is the call graph for this function:



Here is the caller graph for this function:



4.43.4 Member Data Documentation

4.43.4.1 componentSignature

```
Signature System::componentSignature  [private]
```

Signature representing the required components for this system.

4.43.4.2 entities

```
std::vector<Entity> System::entities  [private]
```

List of entities currently managed by this system.

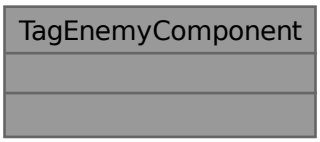
The documentation for this class was generated from the following files:

- src/ECS/ECS.hpp
- src/ECS/ECS.cpp

4.44 TagEnemyComponent Struct Reference

```
#include <TagEnemyComponent.hpp>
```

Collaboration diagram for TagEnemyComponent:



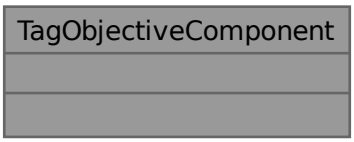
The documentation for this struct was generated from the following file:

- src/Components/TagEnemyComponent.hpp

4.45 TagObjectiveComponent Struct Reference

```
#include <TagObjectiveComponent.hpp>
```

Collaboration diagram for TagObjectiveComponent:



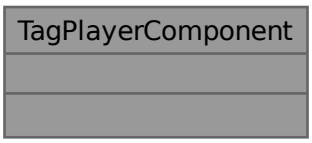
The documentation for this struct was generated from the following file:

- src/Components/TagObjectiveComponent.hpp

4.46 TagPlayerComponent Struct Reference

`#include <TagPlayerComponent.hpp>`

Collaboration diagram for TagPlayerComponent:



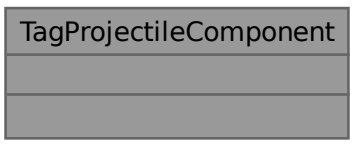
The documentation for this struct was generated from the following file:

- src/Components/TagPlayerComponent.hpp

4.47 TagProjectileComponent Struct Reference

`#include <TagProjectileComponent.hpp>`

Collaboration diagram for TagProjectileComponent:



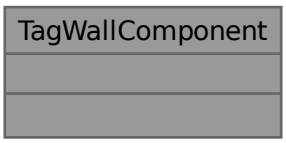
The documentation for this struct was generated from the following file:

- src/Components/TagProjectileComponent.hpp

4.48 TagWallComponent Struct Reference

`#include <TagWallComponent.hpp>`

Collaboration diagram for TagWallComponent:



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

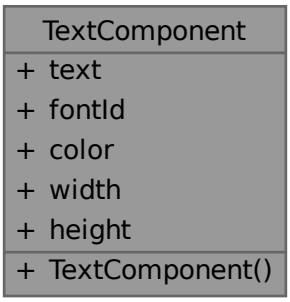
- src/Components/TagWallComponent.hpp

4.49 TextComponent Struct Reference

Component to handle text rendering attributes.

`#include <TextComponent.hpp>`

Collaboration diagram for TextComponent:



Public Member Functions

- TextComponent (const std::string &text="", const std::string &fontId="", u_char r=0, u_char g=0, u_char b=0, u_char a=0)

  Constructor for TextComponent.

Public Attributes

- std::string text

  The text content to render.
- std::string fontId

  Identifier for the font used.
- SDL_Color color

  Color of the text (RGBA)
- int width

  Width of the rendered text.
- int height

  Height of the rendered text.

4.49.1 Detailed Description

Component to handle text rendering attributes.

Contains the text string, font identifier, text color, and dimensions (width and height) of the rendered text.

4.49.2 Constructor & Destructor Documentation

4.49.2.1 TextComponent()

```
TextComponent::TextComponent (
            const std::string & text = "",
            const std::string & fontId = "",
            u_char r = 0,
            u_char g = 0,
            u_char b = 0,
            u_char a = 0 )  [inline]
```

Constructor for TextComponent.

Parameters

text	Initial text string (default empty)
fontId	Font identifier string (default empty)
r	Red component of text color (default 0)
g	Green component of text color (default 0)
b	Blue component of text color (default 0)
a	Alpha component of text color (default 0)

4.49.3 Member Data Documentation

4.49.3.1 color

```
SDL_Color TextComponent::color
```

Color of the text (RGBA)

4.49.3.2 fontId

```
std::string TextComponent::fontId
```

Identifier for the font used.

4.49.3.3 height

```
int TextComponent::height
```

Height of the rendered text.

4.49.3.4 text

```
std::string TextComponent::text
```

The text content to render.

4.49.3.5 width

```
int TextComponent::width
```

Width of the rendered text.

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

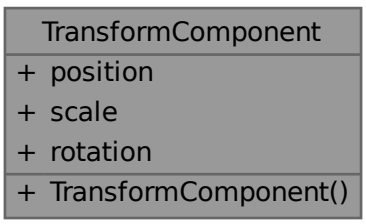
- src/Components/TextComponent.hpp

4.50 TransformComponent Struct Reference

Component to represent the transform of an entity.

```
#include <TransformComponent.hpp>
```

Collaboration diagram for TransformComponent:



Public Member Functions

- TransformComponent (glm::vec2 position=glm::vec2(0.0, 0.0), glm::vec2 scale=glm::vec2(1.0, 1.0), double rotation=0.0)

  Constructor for TransformComponent.

Public Attributes

- glm::vec2 position

  Position of the entity in 2D space.
- glm::vec2 scale

  Scale factors on the x and y axes.
- double rotation

  Rotation angle in degrees or radians (depending on use)

4.50.1 Detailed Description

Component to represent the transform of an entity.

Contains position, scale, and rotation.

4.50.2 Constructor & Destructor Documentation

4.50.2.1 TransformComponent()

```
TransformComponent::TransformComponent (
            glm::vec2 position = glm::vec2(0.0, 0.0),
            glm::vec2 scale = glm::vec2(1.0, 1.0),
            double rotation = 0.0 )  [inline]
```

Constructor for TransformComponent.

Parameters

position	Initial position (default is (0,0))
scale	Initial scale (default is (1,1))
rotation	Initial rotation (default is 0)

4.50.3 Member Data Documentation

4.50.3.1 position

```
glm::vec2 TransformComponent::position
```

Position of the entity in 2D space.

4.50.3.2 rotation

```
double TransformComponent::rotation
```

Rotation angle in degrees or radians (depending on use)

4.50.3.3 scale

```
glm::vec2 TransformComponent::scale
```

Scale factors on the x and y axes.

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

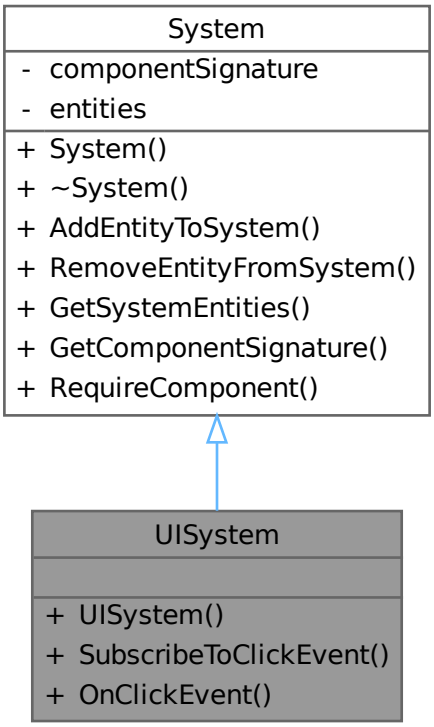
- src/Components/TransformComponent.hpp

4.51 UISystem Class Reference

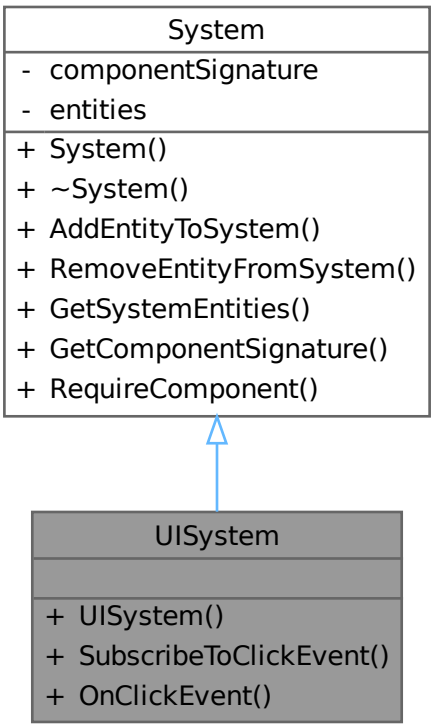
Handles UI elements that can be clicked and processes click events.

```
#include <UISystem.hpp>
```

Inheritance diagram for UISystem:



Collaboration diagram for UISystem:



Public Member Functions

- UISystem ()

  Constructs UISystem and requires necessary components.
- void SubscribeToClickEvent (std::unique_ptr< EventManager > &eventManager)

  Subscribes this system to the ClickEvent in the EventManager.
- void OnClickEvent (ClickEvent &e)

  Handles a ClickEvent by checking if the click position intersects with any UI element.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.51.1 Detailed Description

Handles UI elements that can be clicked and processes click events.

This system requires entities to have ClickableComponent, TextComponent, and TransformComponent. It subscribes to click events and triggers Lua script callbacks when UI elements are clicked.

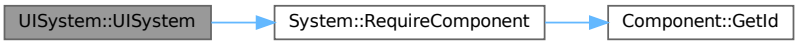
4.51.2 Constructor & Destructor Documentation

4.51.2.1 UISystem()

```
UISystem::UISystem ( )  [inline]
```

Constructs UISystem and requires necessary components.

Here is the call graph for this function:



4.51.3 Member Function Documentation

4.51.3.1 OnClickEvent()

```
void UISystem::OnClickEvent (
            ClickEvent & e )  [inline]
```

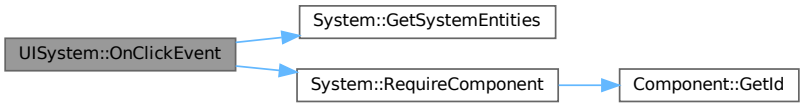
Handles a ClickEvent by checking if the click position intersects with any UI element.

If an entity contains a ScriptComponent with an onClick Lua callback, it will be called.

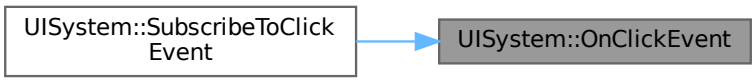
Parameters

e	Reference to the ClickEvent containing click position data.

Here is the call graph for this function:



Here is the caller graph for this function:



4.51.3.2 SubscribeToClickEvent()

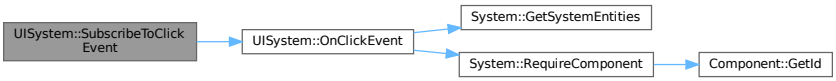
```
void UISystem::SubscribeToClickEvent (
            std::unique_ptr< EventManager > & eventManager )  [inline]
```

Subscribes this system to the ClickEvent in the EventManager.

Parameters

eventManager	Unique pointer reference to the EventManager to subscribe to.

Here is the call graph for this function:



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

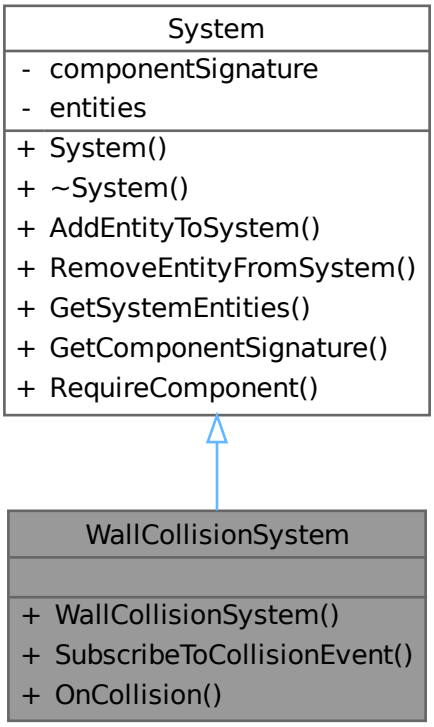
- src/Systems/UISystem.hpp

4.52 WallCollisionSystem Class Reference

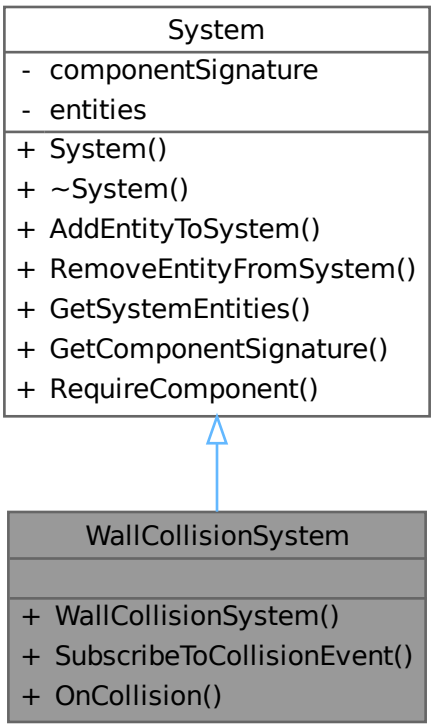
Handles collisions between wall entities and player entities.

```
#include <WallCollisionSystem.hpp>
```

Inheritance diagram for WallCollisionSystem:



Collaboration diagram for WallCollisionSystem:



Public Member Functions

- WallCollisionSystem ()

  Constructs WallCollisionSystem and requires TagWallComponent.
- void SubscribeToCollisionEvent (std::unique_ptr< EventManager > &eventManager)

  Subscribes this system to the CollisionEvent in the EventManager.
- void OnCollision (CollisionEvent &e)

  Handles a collision event between entities, resolving collisions involving walls and players.

Public Member Functions inherited from System

- System ()=default
- ~System ()=default
- void AddEntityToSystem (Entity entity)

  Adds an entity to this system.
- void RemoveEntityFromSystem (Entity entity)

  Removes an entity from this system.
- std::vector< Entity > GetSystemEntities () const

  Returns all entities currently registered in this system.
- const Signature & GetComponentSignature () const

  Returns the component signature required by this system.
- template<typename TComponent >
  void RequireComponent ()

  Adds a required component type to the system's signature.

4.52.1 Detailed Description

Handles collisions between wall entities and player entities.

This system requires entities with the TagWallComponent. It listens for collision events and when a player collides with a wall, it adjusts the player's position to prevent overlapping and stops their movement by zeroing velocity.

4.52.2 Constructor & Destructor Documentation

4.52.2.1 WallCollisionSystem()

```
WallCollisionSystem::WallCollisionSystem ( )  [inline]
```

Constructs WallCollisionSystem and requires TagWallComponent.

Here is the call graph for this function:

4.52.3 Member Function Documentation

4.52.3.1 OnCollision()

```
void WallCollisionSystem::OnCollision (
            CollisionEvent & e )  [inline]
```

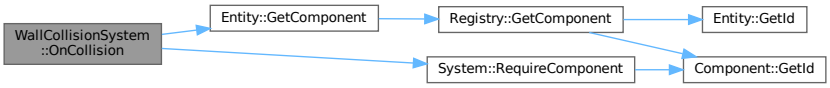
Handles a collision event between entities, resolving collisions involving walls and players.

If one entity is a wall and the other a player, adjusts the player's position to avoid overlap and resets the player's velocity.

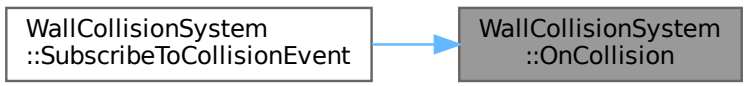
Parameters

e	Reference to the CollisionEvent containing involved entities.

Here is the call graph for this function:



Here is the caller graph for this function:



4.52.3.2 SubscribeToCollisionEvent()

```
void WallCollisionSystem::SubscribeToCollisionEvent (
            std::unique_ptr< EventManager > & eventManager )  [inline]
```

Subscribes this system to the CollisionEvent in the EventManager.

Parameters

eventManager	Unique pointer reference to the EventManager to subscribe to.

Here is the call graph for this function:

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

- src/Systems/WallCollisionSystem.hpp

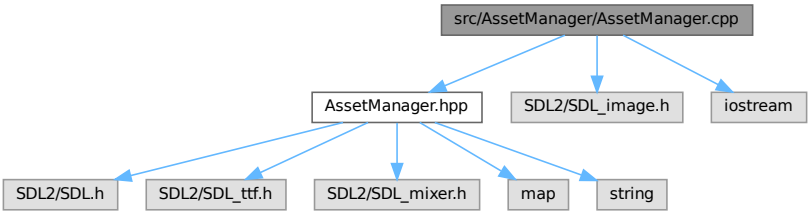
Chapter 5

File Documentation

5.1 src/AssetManager/AssetManager.cpp File Reference

```
#include "AssetManager.hpp"
#include <SDL2/SDL_image.h>
#include <iostream>
```

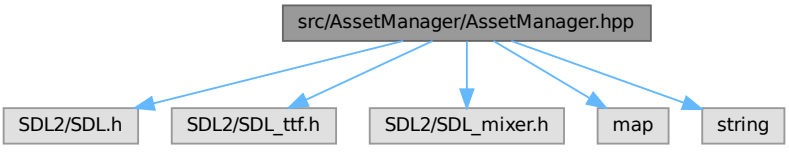
Include dependency graph for AssetManager.cpp:



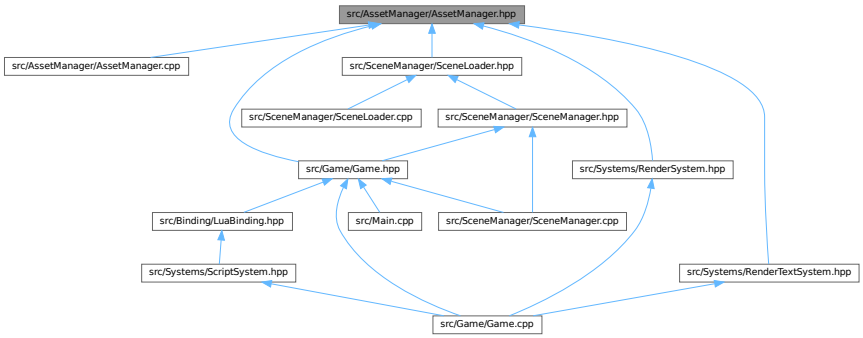
5.2 src/AssetManager/AssetManager.hpp File Reference

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_mixer.h>
#include <map>
#include <string>
```

Include dependency graph for AssetManager.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class AssetManager

  Manages textures, fonts, and music assets using SDL2.

5.3 AssetManager.hpp

Go to the documentation of this file.

```
00001 #ifndef ASSETMANAGER_HPP
00002 #define ASSETMANAGER_HPP
00003
00004 #include <SDL2/SDL.h>
00005 #include <SDL2/SDL_ttf.h>
00006 #include <SDL2/SDL_mixer.h>
00007 #include <map>
00008 #include <string>
00009
00018 class AssetManager {
00019  private:
00021   std::map<std::string, SDL_Texture*> textures;
00022
00024   std::map<std::string, TTF_Font*> fonts;
00025
00027   std::map<std::string, Mix_Music*> musics;
00028
00030   Mix_Music* currentMusic = nullptr;
00031
00032  public:
00036   AssetManager();
00037
00041   ~AssetManager();
00042
00046   void ClearAssets();
00047
00055   void AddTexture(SDL_Renderer* renderer, const std::string& textureId,
00056                   const std::string& filePath);
00057
00064   SDL_Texture* GetTexture(const std::string& textureId);
00065
00073   void AddFont(const std::string& fontId, const std::string& filePath, int fontSize);
00074
00081   TTF_Font* GetFont(const std::string& fontId);
00082
00089   void LoadMusic(const std::string& musicId, const std::string& filePath);
00090
00097   void PlayMusic(const std::string& musicId, int loops = -1);
00098
00102   void StopMusic();
00103
00107   void ResumeMusic();
00108
00112   void PauseMusic();
00113
00117   void ClearMusic();
00118 };
00119
00120 #endif // ASSETMANAGER_HPP
```

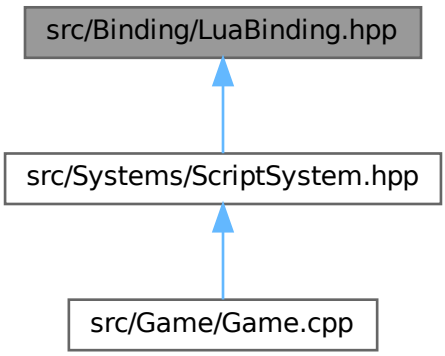
5.4 src/Binding/LuaBinding.hpp File Reference

Contains Lua bindings for interacting with ECS components and systems.

```
#include <string>
#include <glm/glm.hpp>
#include "../Components/RigidBodyComponent.hpp"
#include "../Components/TransformComponent.hpp"
#include "../Components/AnimationComponent.hpp"
#include "../Components/SpriteComponent.hpp"
#include "../Systems/EntitySpawnerSystem.hpp"
#include "../Systems/SceneTimeSystem.hpp"
#include "../Systems/DefeatSystem.hpp"
#include "../Systems/EnemyIASystem.hpp"
#include "../Systems/DamageSystem.hpp"
#include "../ECS/ECS.hpp"
#include "../Game/Game.hpp"
```

Include dependency graph for LuaBinding.hpp:

This graph shows which files directly or indirectly include this file:



Functions

- bool IsActionActivated (const std::string &action)

  Checks if a specific action is currently activated in the controller.
- void SetVelocity (Entity entity, float x, float y)

  Sets the velocity of an entity's RigidBodyComponent.
- int GetVelocity (Entity entity)

  Gets the horizontal velocity (x) of an entity's RigidBodyComponent.
- float GetScale (Entity entity)

Gets the horizontal scale of an entity's TransformComponent.

- void SetScale (Entity entity, float x, float y)

  Sets the scale of an entity's TransformComponent.
- void SetPosition (Entity entity, float x, float y)

  Sets the position of an entity's TransformComponent.
- float GetPositionX (Entity entity)

  Gets the x position of an entity's TransformComponent.
- float GetPositionY (Entity entity)

  Gets the y position of an entity's TransformComponent.
- double GetDepth (Entity entity)

  Gets the max depth scale from an entity's DepthComponent.
- void SetSrcRect (Entity entity, int width=0, int height=0, int srcRectX=0, int srcRectY=0)

  Sets the source rectangle for the SpriteComponent of an entity.
- void SetNumFrames (Entity entity, int numFrames)

  Sets the number of frames for the AnimationComponent of an entity.
- void CreateDynamicEntity (Entity entity, double dir, int num, double scale)

  Creates a new entity using the EntitySpawnerSystem and initializes its position and velocity.
- int GetDeltaTime ()

  Gets the delta time of the current scene from SceneTimeSystem.
- int GetTime ()

  Gets the total scene time from SceneTimeSystem.
- void SetTimer (Entity entity, int newTime)

  Sets a timer text value in a TextComponent from the given time.
- bool GetDefeat ()

  Checks whether the defeat condition has been triggered.
- void SetText (Entity entity, int newText)

  Sets the text of a TextComponent using an integer value.
- void GoToScene (const std::string &sceneName)

  Switches to a different scene by name.
- double SearchObjectiveX (Entity entity, bool player)

  Searches for the nearest target entity and returns its X position.
- double SearchObjectiveY (Entity entity, bool player)

  Searches for the nearest target entity and returns its Y position.
- double SearchObjectiveScale (Entity entity, bool player)

  Searches for the nearest target entity and returns its scale.
- double SearchObjectiveDepth (Entity entity, bool player)

  Searches for the nearest target entity and returns its depth.
- void DestroyAllEnemies ()

  Destroys all enemy entities via the DamageSystem.
- void Kill (Entity entity)

  Marks an entity as killed.

5.4.1 Detailed Description

Contains Lua bindings for interacting with ECS components and systems.

5.4.2 Function Documentation

5.4.2.1 CreateDynamicEntity()

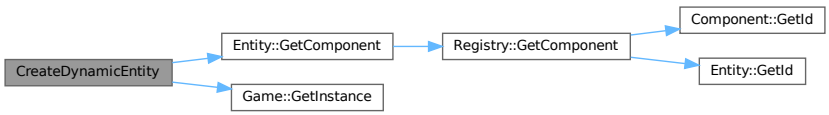
```
void CreateDynamicEntity (
            Entity entity,
            double dir,
            int num,
            double scale )
```

Creates a new entity using the EntitySpawnerSystem and initializes its position and velocity.

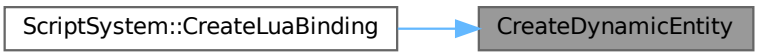
Parameters

entity	The source entity.
dir	Direction multiplier for velocity.
num	Identifier for the entity type.
scale	Scale to apply to the new entity.

Here is the call graph for this function:



Here is the caller graph for this function:

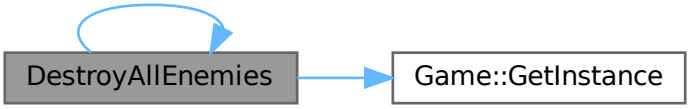


5.4.2.2 DestroyAllEnemies()

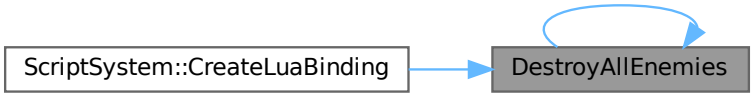
```
void DestroyAllEnemies ( )
```

Destroys all enemy entities via the DamageSystem.

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.3 GetDefeat()

```
bool GetDefeat ( )
```

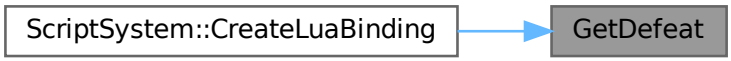
Checks whether the defeat condition has been triggered.

Returns

True if defeat is active, false otherwise.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.4 GetDeltaTime()

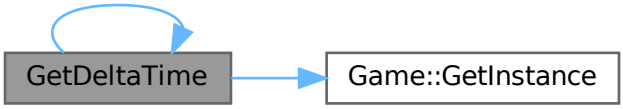
```
int GetDeltaTime ( )
```

Gets the delta time of the current scene from SceneTimeSystem.

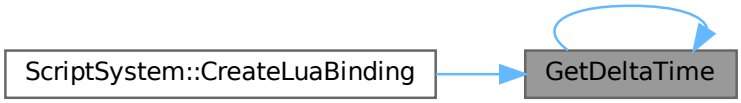
Returns

Delta time in milliseconds.

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.5 GetDepth()

```
double GetDepth (
            Entity entity )
```

Gets the max depth scale from an entity's DepthComponent.

Parameters

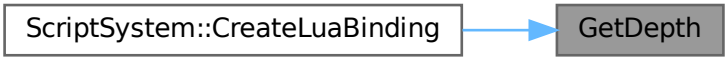
entity	The target entity.

Returns

The max depth scale.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.6 GetPositionX()

```
float GetPositionX (
            Entity entity )
```

Gets the x position of an entity's TransformComponent.

Parameters

entity	The target entity.

Returns

The x position.

Here is the call graph for this function:

Here is the caller graph for this function:

5.4.2.7 GetPositionY()

```
float GetPositionY (
            Entity entity )
```

Gets the y position of an entity's TransformComponent.

Parameters

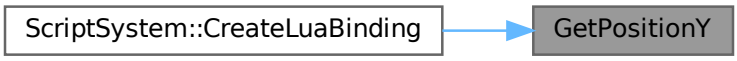
entity	The target entity.

Returns

The y position.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.8 GetScale()

```
float GetScale (
            Entity entity )
```

Gets the horizontal scale of an entity's TransformComponent.

Parameters

entity	The target entity.

Returns

The scale on the x-axis.

Gets the horizontal scale of an entity's TransformComponent.

Parameters

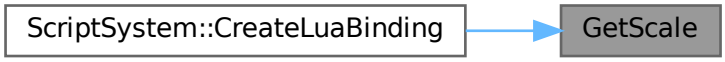
entity	The target entity.

Returns

The scale on the x-axis.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.9 GetTime()

```
int GetTime ( )
```

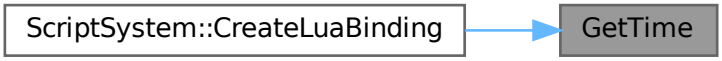
Gets the total scene time from SceneTimeSystem.

Returns

Scene time in milliseconds.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.10 GetVelocity()

```
int GetVelocity (
            Entity entity )
```

Gets the horizontal velocity (x) of an entity's RigidBodyComponent.

Parameters

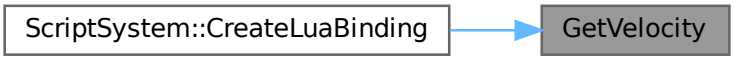
entity	The target entity.

Returns

The x velocity.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.11 GoToScene()

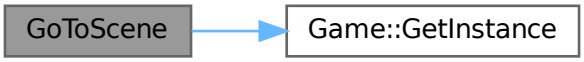
```
void GoToScene (
            const std::string & sceneName )
```

Switches to a different scene by name.

Parameters

sceneName	Name of the next scene.

Here is the call graph for this function:



Here is the caller graph for this function:

5.4.2.12 IsActionActivated()

```
bool IsActionActivated (
            const std::string & action )
```

Checks if a specific action is currently activated in the controller.

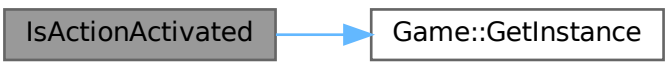
Parameters

action	The action name.

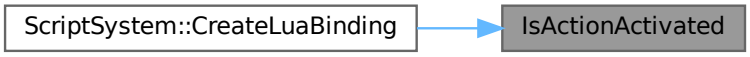
Returns

True if the action is activated, false otherwise.

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.13 Kill()

```
void Kill (
            Entity entity )
```

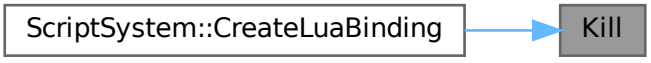
Marks an entity as killed.

Parameters

entity	The entity to kill.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.14 SearchObjectiveDepth()

```
double SearchObjectiveDepth (
            Entity entity,
            bool player )
```

Searches for the nearest target entity and returns its depth.

Parameters

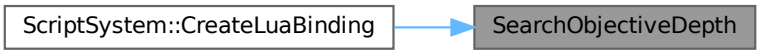
entity	The querying entity.
player	Whether to search for a player (true) or enemy (false).

Returns

The depth of the closest objective.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.15 SearchObjectiveScale()

```
double SearchObjectiveScale (
            Entity entity,
            bool player )
```

Searches for the nearest target entity and returns its scale.

Parameters

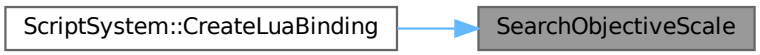
entity	The querying entity.
player	Whether to search for a player (true) or enemy (false).

Returns

The scale of the closest objective.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.16 SearchObjectiveX()

```
double SearchObjectiveX (
            Entity entity,
            bool player )
```

Searches for the nearest target entity and returns its X position.

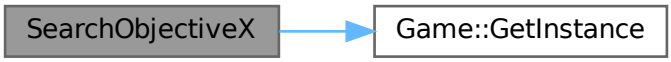
Parameters

entity	The querying entity.
player	Whether to search for a player (true) or enemy (false).

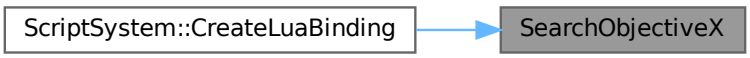
Returns

The X position of the closest objective.

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.17 SearchObjectiveY()

```
double SearchObjectiveY (
            Entity entity,
            bool player )
```

Searches for the nearest target entity and returns its Y position.

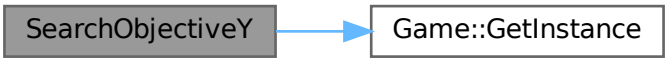
Parameters

entity	The querying entity.
player	Whether to search for a player (true) or enemy (false).

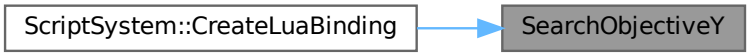
Returns

The Y position of the closest objective.

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.18 SetNumFrames()

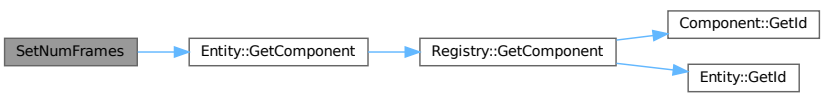
```
void SetNumFrames (
            Entity entity,
            int numFrames )
```

Sets the number of frames for the AnimationComponent of an entity.

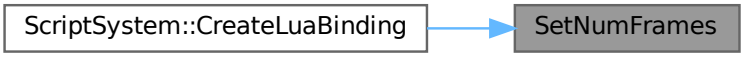
Parameters

entity	The target entity.
numFrames	Number of animation frames.

Here is the call graph for this function:



Here is the caller graph for this function:



5.4.2.19 SetPosition()

```
void SetPosition (
            Entity entity,
            float x,
            float y )
```

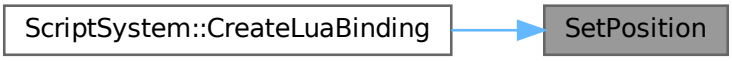
Sets the position of an entity's TransformComponent.

Parameters

entity	The target entity.
x	Position on the x-axis.
y	Position on the y-axis.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.20 SetScale()

```
void SetScale (
            Entity entity,
            float x,
            float y )
```

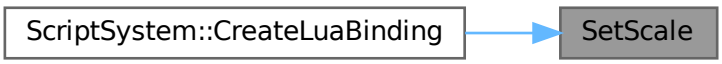
Sets the scale of an entity's TransformComponent.

Parameters

entity	The target entity.
x	Scale on the x-axis.
y	Scale on the y-axis.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.21 SetSrcRect()

```
void SetSrcRect (
            Entity entity,
            int width = 0,
            int height = 0,
            int srcRectX = 0,
            int srcRectY = 0 )
```

Sets the source rectangle for the SpriteComponent of an entity.

Parameters

entity	The target entity.
width	Width of the source rect.
height	Height of the source rect.
srcRectX	X offset of the source rect.
srcRectY	Y offset of the source rect.

Here is the call graph for this function:

Here is the caller graph for this function:

5.4.2.22 SetText()

```
void SetText (
            Entity entity,
            int newText )
```

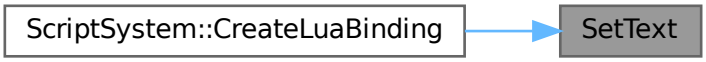
Sets the text of a TextComponent using an integer value.

Parameters

entity	The target entity.
newText	Integer value to set as text.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.23 SetTimer()

```
void SetTimer (
            Entity entity,
            int newTime )
```

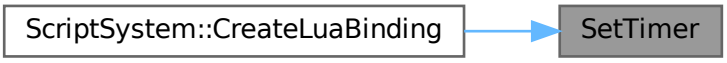
Sets a timer text value in a TextComponent from the given time.

Parameters

entity	The target entity.
newTime	New time value in milliseconds.

Here is the call graph for this function:

Here is the caller graph for this function:



5.4.2.24 SetVelocity()

```
void SetVelocity (
            Entity entity,
```

```
            float x,
            float y )
```

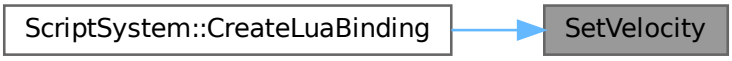
Sets the velocity of an entity's RigidBodyComponent.

Parameters

entity	The target entity.
x	Velocity in x-axis.
y	Velocity in y-axis.

Here is the call graph for this function:

Here is the caller graph for this function:



5.5 LuaBinding.hpp

Go to the documentation of this file.

```
00001 #ifndef LUABINDING_HPP
00002 #define LUABINDING_HPP
00003
00004 #include <string>
00005 #include <glm/glm.hpp>
00006
00007 #include "../Components/RigidBodyComponent.hpp"
00008 #include "../Components/TransformComponent.hpp"
00009 #include "../Components/AnimationComponent.hpp"
00010 #include "../Components/SpriteComponent.hpp"
00011 #include "../Systems/EntitySpawnerSystem.hpp"
00012 #include "../Systems/SceneTimeSystem.hpp"
00013 #include "../Systems/DefeatSystem.hpp"
00014 #include "../Systems/EnemyIASystem.hpp"
00015 #include "../Systems/DamageSystem.hpp"
00016 #include "../ECS/ECS.hpp"
00017 #include "../Game/Game.hpp"
00018
00019
00025 //Controls
00026
00032 bool IsActionActivated(const std::string& action){
00033     return Game::GetInstance().controllerManager->IsActionActivated(action);
00034 }
00035
```

```
00036 // RigidBodyComponent
00037
00044 void SetVelocity(Entity entity, float x, float y){
00045     auto& rigidbody = entity.GetComponent<RigidBodyComponent>();
00046     rigidbody.velocity.x = x;
00047     rigidbody.velocity.y = y;
00048 }
00049
00055 int GetVelocity(Entity entity){
00056     auto& rigidbody = entity.GetComponent<RigidBodyComponent>();
00057     return rigidbody.velocity.x;
00058 }
00059
00066 // TransformComponent
00067
00073 float GetScale(Entity entity) {
00074      auto& transform = entity.GetComponent<TransformComponent>();
00075      return transform.scale.x;
00076 }
00077
00084 void SetScale(Entity entity, float x, float y) {
00085      auto& transform = entity.GetComponent<TransformComponent>();
00086      transform.scale.x = x;
00087      transform.scale.y = y;
00088 }
00089
00096 void SetPosition(Entity entity, float x, float y) {
00097      auto& transform = entity.GetComponent<TransformComponent>();
00098      transform.position.x = x;
00099      transform.position.y = y;
00100 }
00101
00107 float GetPositionX(Entity entity) {
00108      auto& transform = entity.GetComponent<TransformComponent>();
00109      return transform.position.x;
00110 }
00111
00117 float GetPositionY(Entity entity) {
00118      auto& transform = entity.GetComponent<TransformComponent>();
00119      return transform.position.y;
00120 }
00121
00122 // DepthComponent
00123
00129 double GetDepth(Entity entity){
00130     auto& depth = entity.GetComponent<DepthComponent>();
00131     return depth.max_scale;
00132 }
00133
00134 // SpriteComponent
00135
00144 void SetSrcRect(Entity entity,int width = 0
00145     , int height = 0, int srcRectX = 0, int srcRectY = 0){
00146     auto& sprite = entity.GetComponent<SpriteComponent>();
00147     sprite.srcRect = {srcRectX, srcRectY, width, height};
00148 }
00149
00150 //AnimationComponent
00151
00157 void SetNumFrames(Entity entity,int numFrames){
00158     auto& animation = entity.GetComponent<AnimationComponent>();
00159     animation.numFrames = numFrames;
00160
00161 }
00162
00163 //EntitySpawnerComponent
00164
00165
00173 void CreateDynamicEntity(Entity entity, double dir, int num, double scale){
00174     auto& transform = entity.GetComponent<TransformComponent>();
00175     Entity newEntity = Game::GetInstance().registry->GetSystem<EntitySpawnerSystem>().GenerateEntity(
00176         Game::GetInstance().registry,num,Game::GetInstance().lua
00177     );
00178     if(scale > 0){
00179         auto& transformNew = newEntity.GetComponent<TransformComponent>();
00180         auto& rigidBodyNew= newEntity.GetComponent<RigidBodyComponent>();
00181         transformNew.position = transform.position;
00182         transformNew.scale.x = scale;
00183         transformNew.scale.y = scale;
00184         rigidBodyNew.velocity.x = rigidBodyNew.velocity.x*dir;}
00185
00186
00187 }
00188
00189 //Time
00190
00191
```

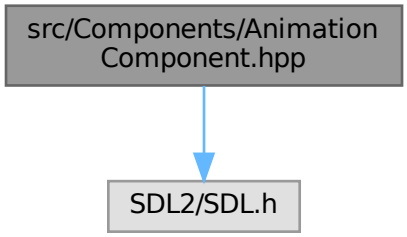
```
00196 int GetDeltaTime(){
00197     int time = Game::GetInstance().registry->GetSystem<SceneTimeSystem>().GetDeltaTime();
00198     return time;
00199 }
00200
00205 int GetTime(){
00206     int time = Game::GetInstance().registry->GetSystem<SceneTimeSystem>().GetSceneTime();
00207     return time;
00208 }
00209
00215 void SetTimer(Entity entity, int newTime){
00216     std::string timer = std::to_string(newTime / 1000);
00217     entity.GetComponent<TextComponent>().text = timer;
00218
00219 }
00220
00221 //Defeat
00222
00227 bool GetDefeat(){
00228     return Game::GetInstance().registry->GetSystem<DefeatSystem>().Defeat;
00229 }
00230
00231 //Text
00232
00238 void SetText(Entity entity, int newText){
00239     entity.GetComponent<TextComponent>().text = std::to_string(newText);
00240
00241 }
00242
00243 // Scenes
00244
00249 void GoToScene(const std::string& sceneName){
00250     Game::GetInstance().sceneManager->SetNextScene(sceneName);
00251     Game::GetInstance().sceneManager->StopScene();
00252 }
00253
00254 //EnemyIASystem
00255
00262 double SearchObjectiveX(Entity entity, bool player){
00263     TransformComponent transform =
      Game::GetInstance().registry->GetSystem<EnemyIASystem>().SearchClosestObjective(entity,player);
00264     return transform.position.x;
00265 }
00266
00273 double SearchObjectiveY(Entity entity, bool player){
00274     TransformComponent transform =
      Game::GetInstance().registry->GetSystem<EnemyIASystem>().SearchClosestObjective(entity,player);
00275     return transform.position.y;
00276 }
00277
00284 double SearchObjectiveScale(Entity entity, bool player){
00285     TransformComponent transform =
      Game::GetInstance().registry->GetSystem<EnemyIASystem>().SearchClosestObjective(entity,player);
00286     return transform.scale.x;
00287 }
00288
00295 double SearchObjectiveDepth(Entity entity, bool player){
00296     DepthComponent depth =
      Game::GetInstance().registry->GetSystem<EnemyIASystem>().SearchClosestObjectiveDepth(entity,player);
00297     return depth.max_scale;
00298 }
00299
00300 // Damage System
00301
00305 void DestroyAllEnemies(){
00306     Game::GetInstance().registry->GetSystem<DamageSystem>().DestroyAllEnemies();
00307 }
00308
00309 //Kill entity
00310
00315 void Kill(Entity entity){
00316     entity.Kill();
00317 }
00318 #endif
```

5.6 src/Components/AnimationComponent.hpp File Reference

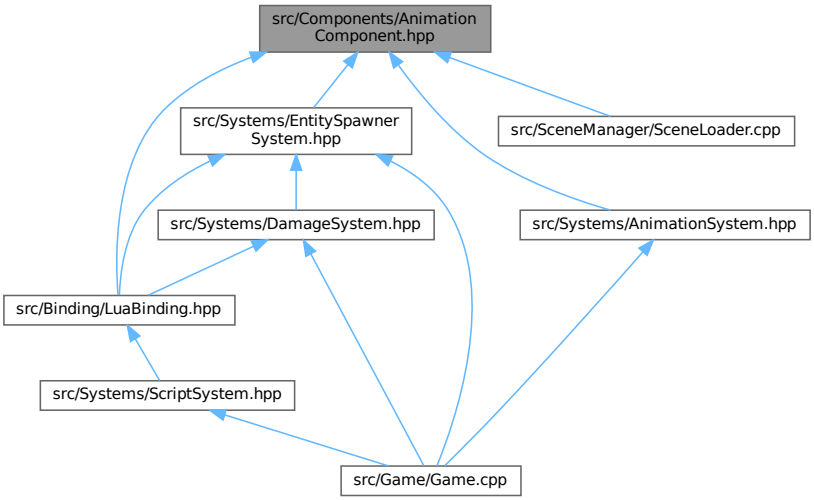
Defines the AnimationComponent used for animated entities.

```
#include <SDL2/SDL.h>
```

Include dependency graph for AnimationComponent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct AnimationComponent

  Holds data for sprite-based animations, including frame count, speed, looping, and timing.

5.6.1 Detailed Description

Defines the AnimationComponent used for animated entities.

5.7 AnimationComponent.hpp

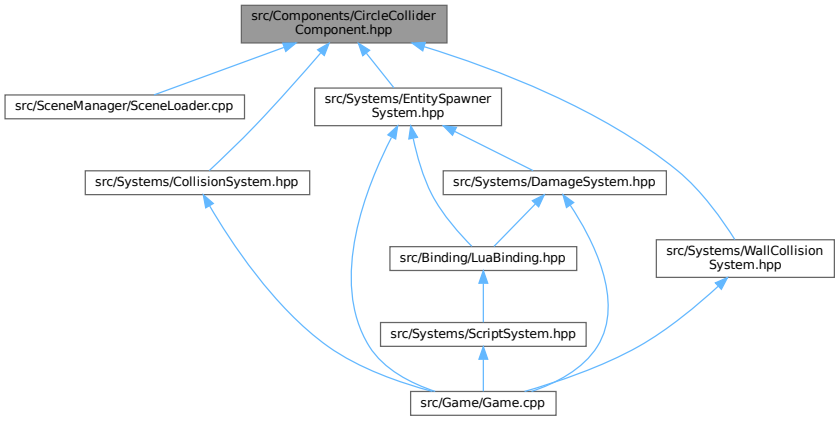
Go to the documentation of this file.

```
00001 #ifndef ANIMATIONCOMPONENT_HPP
00002 #define ANIMATIONCOMPONENT_HPP
00003
00004 #include <SDL2/SDL.h>
00005
00015 struct AnimationComponent {
00016     int numFrames;
00017     int currentFrame;
00018     int frameSpeedRate;
00019     bool isLoop;
00020     int startTime;
00028     AnimationComponent(int numFrames = 1, int frameSpeedRate = 1, bool isLoop = true){
00029         this->numFrames = numFrames;
00030         this->currentFrame = 1;
00031         this->frameSpeedRate = frameSpeedRate;
00032         this->isLoop = isLoop;
00033         this->startTime = SDL_GetTicks();
00034     }
00035 };
00036
00037 #endif
```

5.8 src/Components/CircleColliderComponent.hpp File Reference

Defines the CircleColliderComponent used for circular collision detection.

This graph shows which files directly or indirectly include this file:



Classes

- struct CircleColliderComponent

  Component that defines the properties of a circular collider.

5.8.1 Detailed Description

Defines the CircleColliderComponent used for circular collision detection.

5.9 CircleColliderComponent.hpp

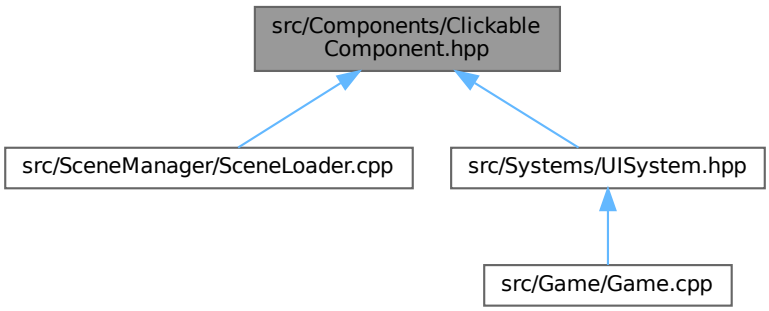
Go to the documentation of this file.

```
00001 #ifndef CIRCLECOLLIDERCOMPONENT_HPP
00002 #define CIRCLECOLLIDERCOMPONENT_HPP
00003
00013 struct CircleColliderComponent {
00014     double radius;
00015     double width;
00016     double height;
00024     CircleColliderComponent(double radius = 0, double width = 0, double height = 0) {
00025         this->radius = radius;
00026         this->width = width;
00027         this->height = height;
00028     }
00029 };
00030
00031 #endif
```

5.10 src/Components/ClickableComponent.hpp File Reference

Defines the ClickableComponent to track if an entity was clicked.

This graph shows which files directly or indirectly include this file:



Classes

- struct ClickableComponent

  Component that indicates if an entity has been clicked.

5.10.1 Detailed Description

Defines the ClickableComponent to track if an entity was clicked.

5.11 ClickableComponent.hpp

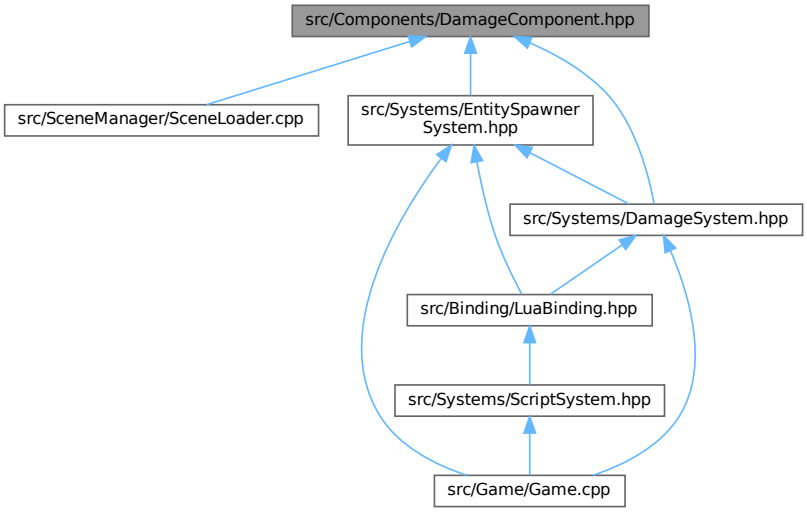
Go to the documentation of this file.

```
00001 #ifndef CLICKABLECOMPONENT_HPP
00002 #define CLICKABLECOMPONENT_HPP
00003
00013 struct ClickableComponent {
00014     bool isClicked;
00019     ClickableComponent(){
00020         isClicked = false;
00021     }
00022 };
00023
00024 #endif
```

5.12 src/Components/DamageComponent.hpp File Reference

Defines the DamageComponent which stores damage values.

This graph shows which files directly or indirectly include this file:



Classes

- struct DamageComponent

  Component that holds the amount of damage dealt.

5.12.1 Detailed Description

Defines the DamageComponent which stores damage values.

5.13 DamageComponent.hpp

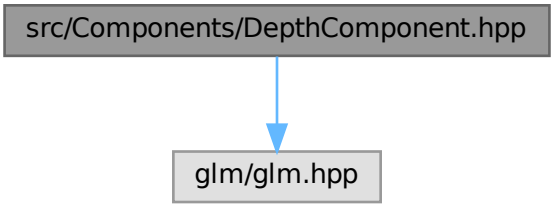
Go to the documentation of this file.

```
00001 #ifndef DAMAGECOMPONENT_HPP
00002 #define DAMAGECOMPONENT_HPP
00003
00013 struct DamageComponent {
00014     int damage_dealt;
00020     DamageComponent(int damage_dealt = 0) {
00021         this->damage_dealt = damage_dealt;
00022     }
00023 };
00024
00025 #endif
```

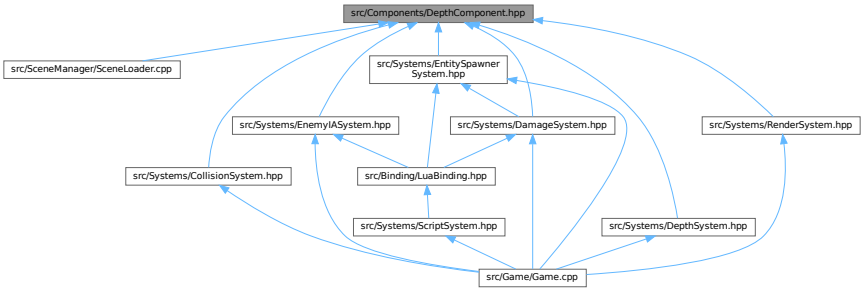
5.14 src/Components/DepthComponent.hpp File Reference

Defines the DepthComponent for handling scaling and depth effects.

`#include <glm/glm.hpp>`
Include dependency graph for DepthComponent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct DepthComponent

  Component that manages depth-related scaling parameters for an entity.

5.14.1 Detailed Description

Defines the DepthComponent for handling scaling and depth effects.

5.15 DepthComponent.hpp

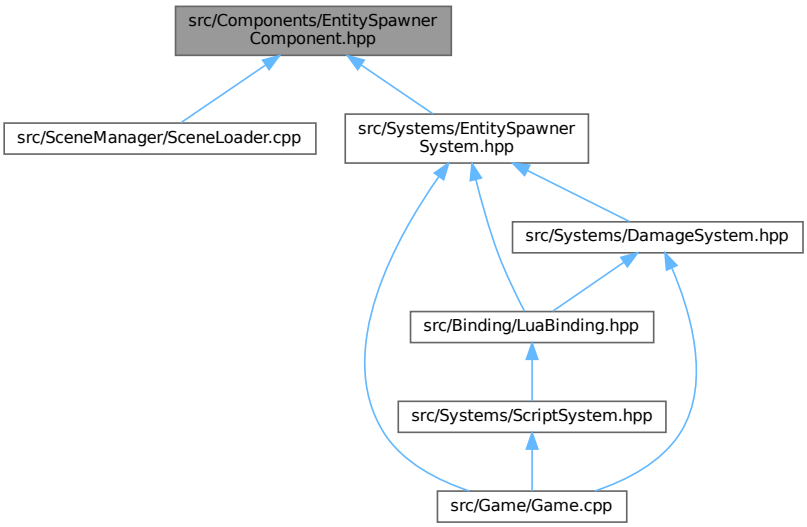
Go to the documentation of this file.

```
00001 #ifndef DEPTHCOMPONENT_HPP
00002 #define DEPTHCOMPONENT_HPP
00003 #include <glm/glm.hpp>
00004
00014 struct DepthComponent {
00015     float min_scale;
00016     float max_scale;
00017     float original_width;
00018     float scale_speed;
00019     float reference_point;
00030     DepthComponent(float min_scale = 0, float max_scale = 0,
00031                    float original_width = 0, float scale_speed = 0,
00032                    float reference_point = 0) {
00033         this->min_scale = min_scale;
00034         this->max_scale = max_scale;
00035         this->original_width = original_width;
00036         this->scale_speed = scale_speed;
00037         this->reference_point = reference_point;
00038     }
00039 };
00040 #endif
```

5.16 src/Components/EntitySpawnerComponent.hpp File Reference

Component to indicate if an entity is a player spawner.

This graph shows which files directly or indirectly include this file:



Classes

- struct EntitySpawnerComponent

  Component that marks an entity as a player or non-player spawner.

5.16.1 Detailed Description

Component to indicate if an entity is a player spawner.

5.17 EntitySpawnerComponent.hpp

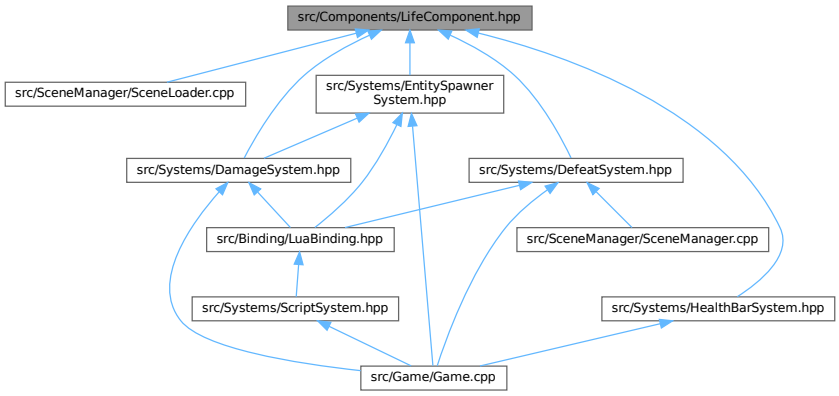
Go to the documentation of this file.

```
00001 #ifndef ENTITYSPAWNERCOMPONENT_HPP
00002 #define ENTITYSPAWNERCOMPONENT_HPP
00003
00013 struct EntitySpawnerComponent {
00014     bool is_player;
00021     EntitySpawnerComponent(bool is_player = false) {
00022         this->is_player = is_player;
00023     }
00024 };
00025
00026 #endif
```

5.18 src/Components/LifeComponent.hpp File Reference

Component to manage an entity's life count and maximum life.

This graph shows which files directly or indirectly include this file:



Classes

- struct LifeComponent

  Represents the current and maximum life of an entity.

5.18.1 Detailed Description

Component to manage an entity's life count and maximum life.

5.19 LifeComponent.hpp

Go to the documentation of this file.

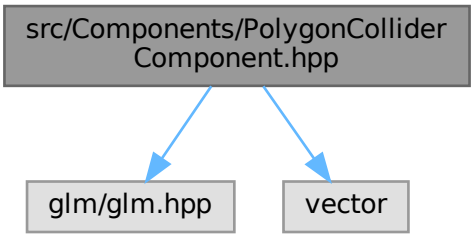
```
00001 #ifndef LIFECOMPONENT_HPP
00002 #define LIFECOMPONENT_HPP
00003
00013 struct LifeComponent {
00014     int life_count;
00015     int life_max;
00023     LifeComponent(int life_count = 0, int life_max = 0) {
00024         this->life_count = life_count;
00025         this->life_max = life_max;
00026     }
00027 };
00028
00029 #endif
```

5.20 src/Components/PolygonColliderComponent.hpp File Reference

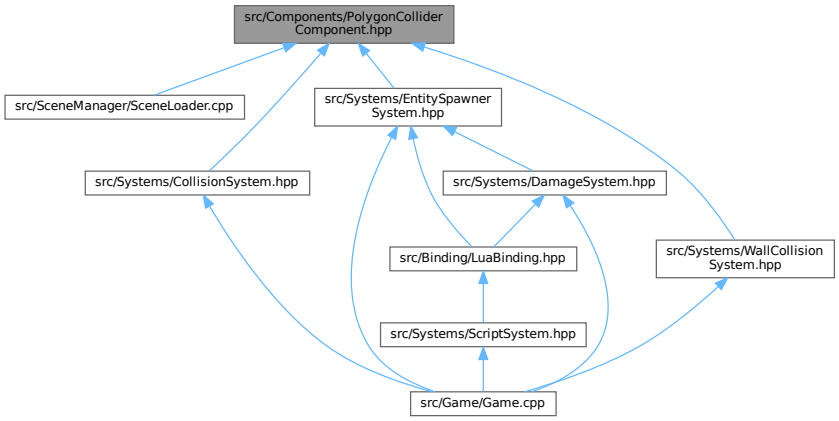
Component that holds polygon collider vertices for collision detection.

```
#include <glm/glm.hpp>
#include <vector>
```

Include dependency graph for PolygonColliderComponent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct PolygonColliderComponent

  Stores the vertices that define a polygon collider shape.

5.20.1 Detailed Description

Component that holds polygon collider vertices for collision detection.

5.21 PolygonColliderComponent.hpp

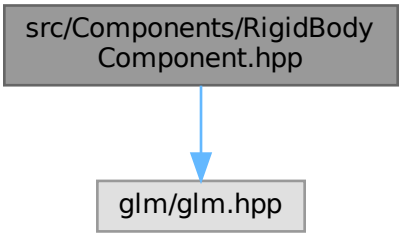
Go to the documentation of this file.

```
00001 #ifndef POLYGONCOLLIDERCOMPONENT_HPP
00002 #define POLYGONCOLLIDERCOMPONENT_HPP
00003
00004 #include <glm/glm.hpp>
00005 #include <vector>
00006
00016 struct PolygonColliderComponent {
00017     std::vector<glm::vec2> vertices;
00024     PolygonColliderComponent(std::vector<glm::vec2> vertices = {}) {
00025         this->vertices = vertices;
00026     }
00027 };
00028
00029 #endif
```

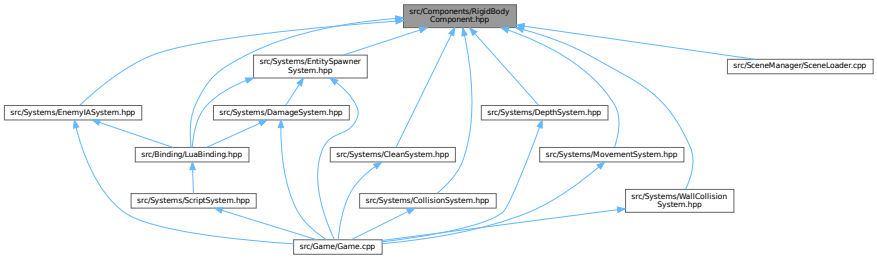
5.22 src/Components/RigidBodyComponent.hpp File Reference

Component representing a rigid body's velocity.

`#include <glm/glm.hpp>`
Include dependency graph for RigidBodyComponent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct RigidBodyComponent

  Stores the velocity vector of an entity's rigid body.

5.22.1 Detailed Description

Component representing a rigid body's velocity.

5.23 RigidBodyComponent.hpp

Go to the documentation of this file.

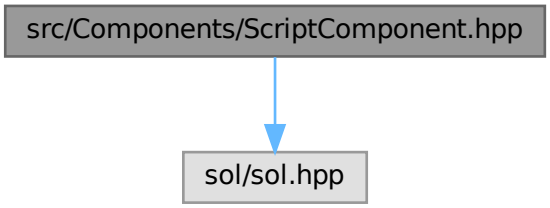
```
00001 #ifndef RIGIDBODYCOMPONENT_HPP
00002 #define RIGIDBODYCOMPONENT_HPP
00003
00004 #include <glm/glm.hpp>
00005
00015 struct RigidBodyComponent{
00016     glm::vec2 velocity;
00023     RigidBodyComponent(glm::vec2 velocity = glm::vec2(0.0)){
00024         this->velocity = velocity;
00025     }
00026 };
00027
00028 #endif
```

5.24 src/Components/ScriptComponent.hpp File Reference

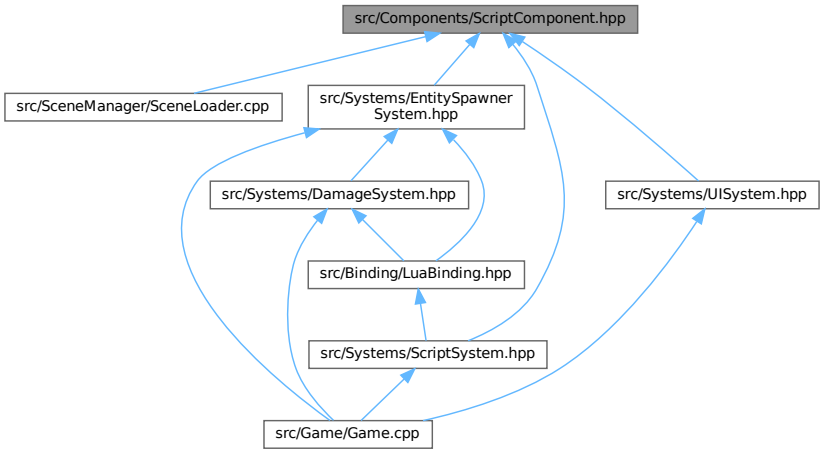
Component to hold Lua script functions for entity behavior.

```
#include <sol/sol.hpp>
```

Include dependency graph for ScriptComponent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct ScriptComponent

  Holds Lua functions for updating and click handling scripts.

5.24.1 Detailed Description

Component to hold Lua script functions for entity behavior.

5.25 ScriptComponent.hpp

Go to the documentation of this file.

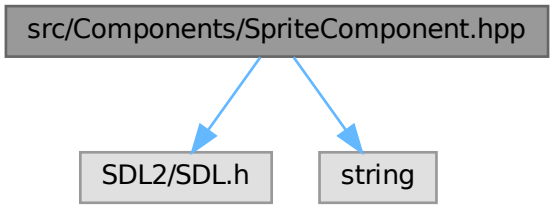
```
00001 #ifndef SCRIPTCOMPONENT_HPP
00002 #define SCRIPTCOMPONENT_HPP
00003
00004 #include <sol/sol.hpp>
00005
00015 struct ScriptComponent {
00016     sol::function update;
00017     sol::function onClick;
00025     ScriptComponent(sol::function update = sol::lua_nil,
00026                     sol::function onClick = sol::lua_nil){
00027         this->update = update;
00028         this->onClick = onClick;
00029     }
00030 };
00031
00032 #endif
```

5.26 src/Components/SpriteComponent.hpp File Reference

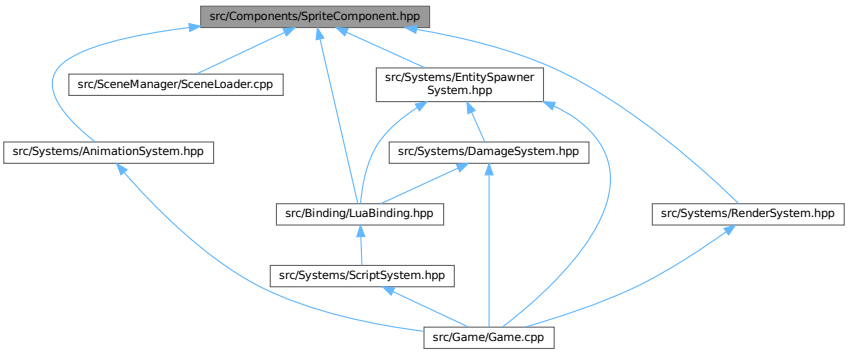
Component for rendering a sprite with a texture and source rectangle.

```
#include <SDL2/SDL.h>
#include <string>
```

Include dependency graph for SpriteComponent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct SpriteComponent

  Holds data for rendering a sprite, including texture ID and source rectangle.

5.26.1 Detailed Description

Component for rendering a sprite with a texture and source rectangle.

5.27 SpriteComponent.hpp

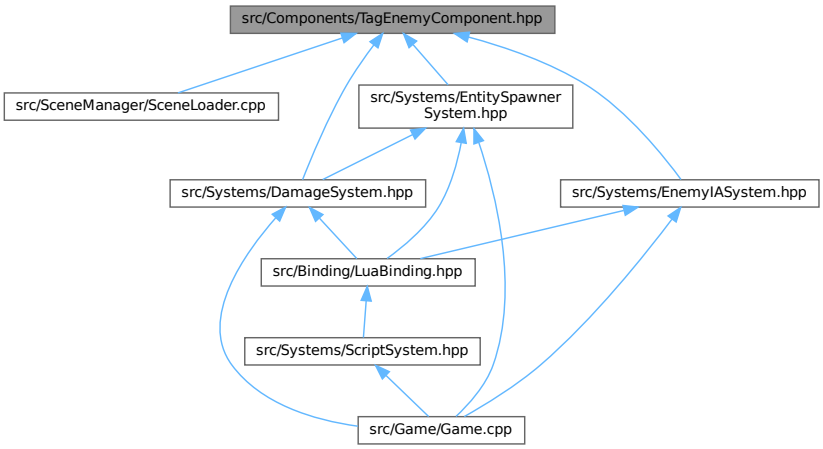
Go to the documentation of this file.

```
00001 #ifndef SPRITECOMPONENT_HPP
00002 #define SPRITECOMPONENT_HPP
00003
00004 #include <SDL2/SDL.h>
00005 #include <string>
00006
00016 struct SpriteComponent {
00017     std::string textureId;
00018     int width;
00019     int height;
00020     SDL_Rect srcRect;
00031     SpriteComponent(const std::string& textureId = "none", int width = 0,
00032                     int height = 0, int srcRectX = 0, int srcRectY = 0){
00033         this->textureId = textureId;
00034         this->width = width;
00035         this->height = height;
00036         this->srcRect = {srcRectX, srcRectY, width, height};
00037     }
00038 };
00039
00040 #endif
```

5.28 src/Components/TagEnemyComponent.hpp File Reference

Empty component used to tag entities as enemies.

This graph shows which files directly or indirectly include this file:



Classes

- struct TagEnemyComponent

5.28.1 Detailed Description

Empty component used to tag entities as enemies.

This component acts as a marker without data to identify enemy entities in the ECS.

5.29 TagEnemyComponent.hpp

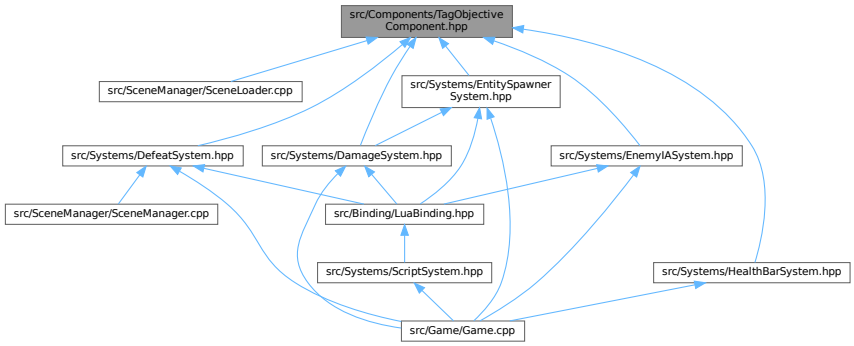
Go to the documentation of this file.

```
00001 #ifndef TAGENEMYCOMPONENT_HPP
00002 #define TAGENEMYCOMPONENT_HPP
00003
00010 struct TagEnemyComponent {
00011
00012 };
00013
00014 #endif
```

5.30 src/Components/TagObjectiveComponent.hpp File Reference

Empty component used to tag entities as objectives.

This graph shows which files directly or indirectly include this file:



Classes

- struct TagObjectiveComponent

5.30.1 Detailed Description

Empty component used to tag entities as objectives.

This component serves as a marker without data to identify objective entities in the ECS.

5.31 TagObjectiveComponent.hpp

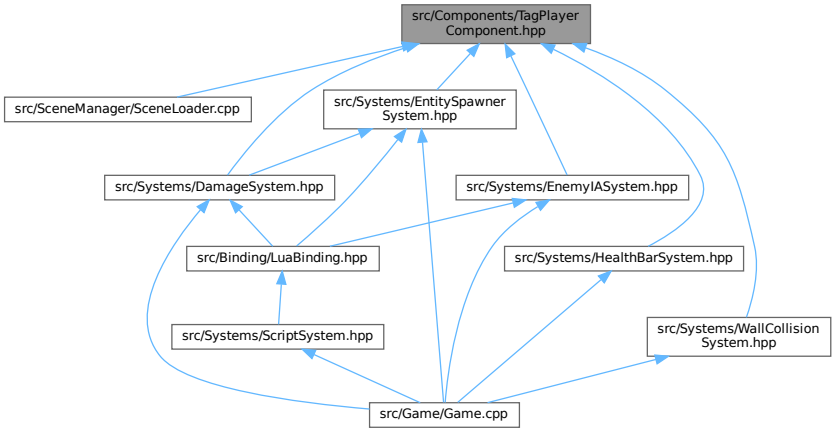
Go to the documentation of this file.

```
00001 #ifndef TAGOBJECTIVECOMPONENT_HPP
00002 #define TAGOBJECTIVECOMPONENT_HPP
00003
00010 struct TagObjectiveComponent {
00011
00012 };
00013
00014 #endif
```

5.32 src/Components/TagPlayerComponent.hpp File Reference

Empty component used to tag entities as players.

This graph shows which files directly or indirectly include this file:



Classes

- struct TagPlayerComponent

5.32.1 Detailed Description

Empty component used to tag entities as players.

This component acts as a marker with no data, used to identify player entities in the ECS.

5.33 TagPlayerComponent.hpp

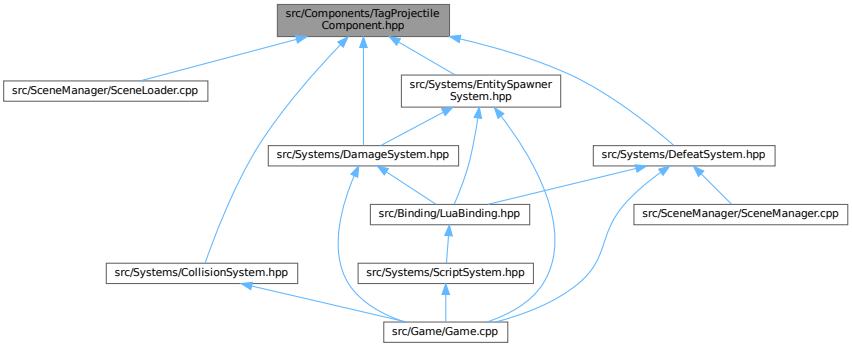
Go to the documentation of this file.

```
00001 #ifndef TAGPLAYERCOMPONENT_HPP
00002 #define TAGPLAYERCOMPONENT_HPP
00003
00010 struct TagPlayerComponent {
00011
00012 };
00013
00014 #endif
```

5.34 src/Components/TagProjectileComponent.hpp File Reference

Empty component used to tag entities as projectiles.

This graph shows which files directly or indirectly include this file:



Classes

- struct TagProjectileComponent

5.34.1 Detailed Description

Empty component used to tag entities as projectiles.

This component acts as a marker with no data, used to identify projectile entities in the ECS.

5.35 TagProjectileComponent.hpp

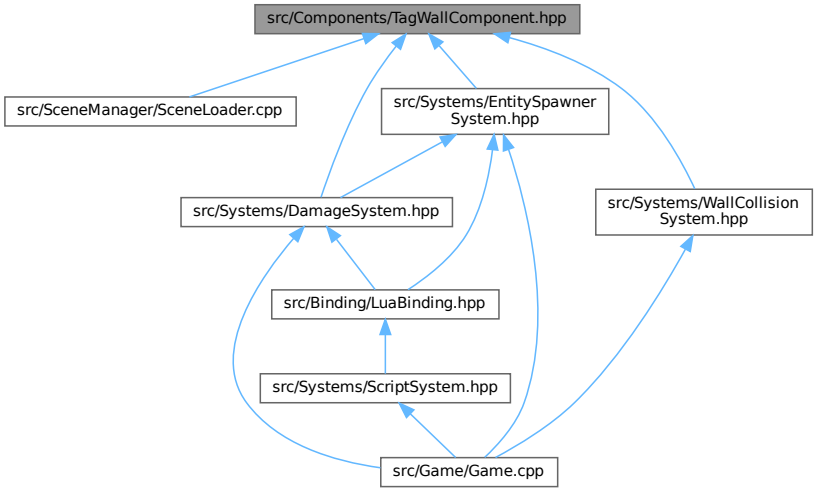
Go to the documentation of this file.

```
00001 #ifndef TAGPROJECTILECOMPONENT_HPP
00002 #define TAGPROJECTILECOMPONENT_HPP
00003
00010 struct TagProjectileComponent {
00011
00012 };
00013
00014 #endif
```

5.36 src/Components/TagWallComponent.hpp File Reference

Empty component used to tag entities as walls.

This graph shows which files directly or indirectly include this file:



Classes

- struct TagWallComponent

5.36.1 Detailed Description

Empty component used to tag entities as walls.

This component serves as a marker to identify wall entities within the ECS.

5.37 TagWallComponent.hpp

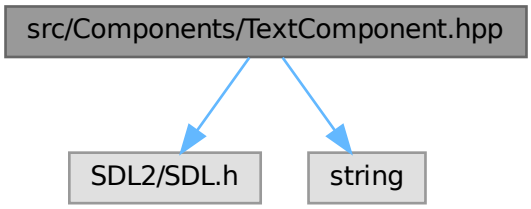
Go to the documentation of this file.

```
00001 #ifndef TAGWALLCOMPONENT_HPP
00002 #define TAGWALLCOMPONENT_HPP
00003
00010 struct TagWallComponent {
00011
00012 };
00013
00014 #endif
```

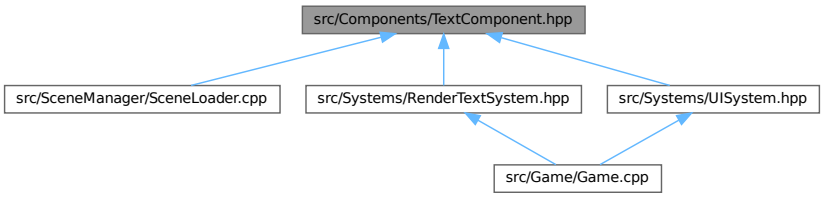
5.38 src/Components/TextComponent.hpp File Reference

```
#include <SDL2/SDL.h>
#include <string>
```

Include dependency graph for TextComponent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct TextComponent

  Component to handle text rendering attributes.

5.39 TextComponent.hpp

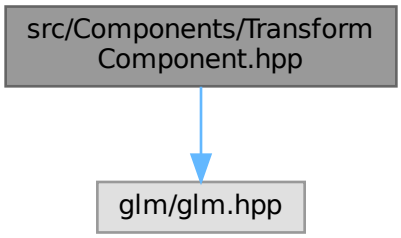
Go to the documentation of this file.

```
00001 #ifndef TEXTCOMPONENT_HPP
00002 #define TEXTCOMPONENT_HPP
00003
00004 #include <SDL2/SDL.h>
00005 #include <string>
00006
00013 struct TextComponent{
00014     std::string text;
00015     std::string fontId;
00016     SDL_Color color;
00017     int width;
00018     int height;
00019
00030     TextComponent(const std::string& text = "", const std::string& fontId = "",
00031                   u_char r = 0, u_char g = 0, u_char b = 0, u_char a = 0) {
```

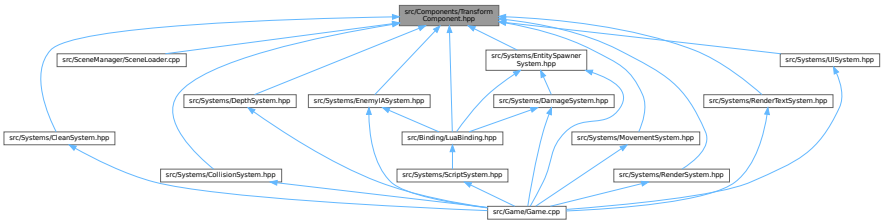
```
00032         this->text = text;
00033         this->fontId = fontId;
00034         this->color.r = r;
00035         this->color.g = g;
00036         this->color.b = b;
00037         this->color.a = a;
00038         this->width = 0;
00039         this->height = 0;
00040     }
00041 };
00042
00043 #endif
```

5.40 src/Components/TransformComponent.hpp File Reference

`#include <glm/glm.hpp>`
Include dependency graph for TransformComponent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct TransformComponent

  Component to represent the transform of an entity.

5.41 TransformComponent.hpp

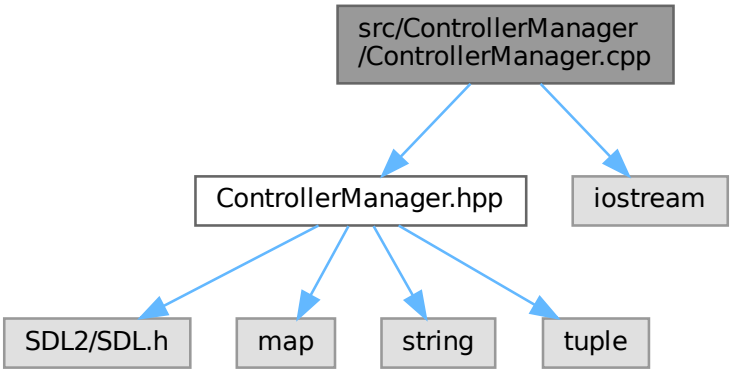
Go to the documentation of this file.

```
00001 #ifndef TRANSFORMCOMPONENT_HPP
00002 #define TRANSFORMCOMPONENT_HPP
00003
00004 #include <glm/glm.hpp>
00005
00011 struct TransformComponent {
00012     glm::vec2 position;
00013     glm::vec2 scale;
00014     double rotation;
00015
00023     TransformComponent(glm::vec2 position = glm::vec2(0.0, 0.0),
00024                        glm::vec2 scale = glm::vec2(1.0, 1.0),
00025                        double rotation = 0.0) {
00026       this->position = position;
00027       this->scale = scale;
00028       this->rotation = rotation;
00029     }
00030 };
00031
00032 #endif
```

5.42 src/ControllerManager/ControllerManager.cpp File Reference

```
#include "ControllerManager.hpp"
#include <iostream>
```

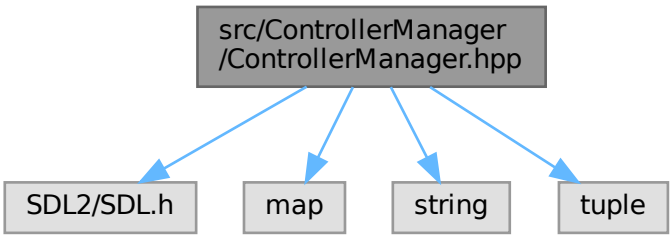
Include dependency graph for ControllerManager.cpp:



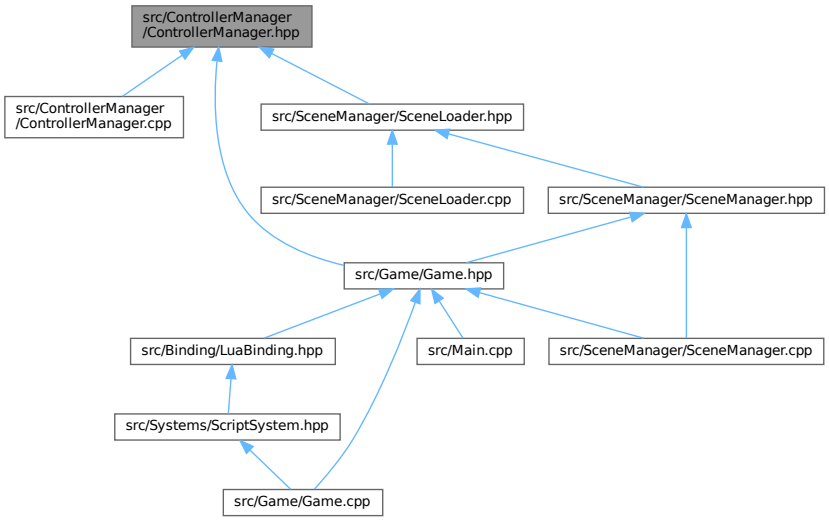
5.43 src/ControllerManager/ControllerManager.hpp File Reference

```
#include <SDL2/SDL.h>
#include <map>
#include <string>
```

`#include <tuple>`
Include dependency graph for ControllerManager.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class ControllerManager

  Handles keyboard and mouse input mapping and state tracking.

5.44 ControllerManager.hpp

Go to the documentation of this file.

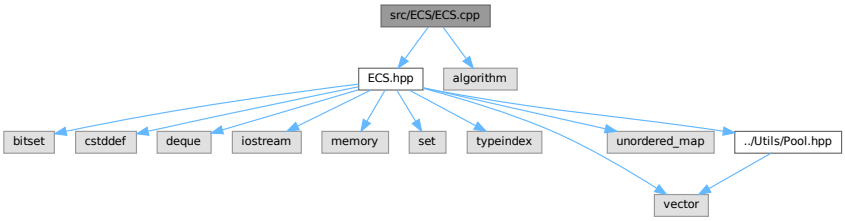
```
00001 #ifndef CONTROLLERMANAGER_HPP
00002 #define CONTROLLERMANAGER_HPP
00003
00004 #include <SDL2/SDL.h>
00005 #include <map>
```

```
00006 #include <string>
00007 #include <tuple>
00008
00013 class ControllerManager {
00014  private:
00016   std::map<std::string, int> actionKeyName;
00017
00019   std::map<int, bool> keyDown;
00020
00022   std::map<std::string, int> mouseButtonName;
00023
00025   std::map<int, bool> mouseButtonDown;
00026
00028   int mousePosX;
00029
00031   int mousePosY;
00032
00033  public:
00037   ControllerManager();
00038
00042   ~ControllerManager();
00043
00047   void Clear();
00048
00049   // Keyboard input methods
00050
00056   void AddActionKey(const std::string& action, int keyCode);
00057
00062   void KeyDown(int keyCode);
00063
00068   void KeyUp(int keyCode);
00069
00075   bool IsActionActivated(const std::string& action);
00076
00077   // Mouse input methods
00078
00084   void AddMouseButton(const std::string& name, int buttonCode);
00085
00090   void MouseButtonDown(int buttonCode);
00091
00096   void MouseButtonUp(int buttonCode);
00097
00103   bool IsMouseButtonDown(const std::string& name);
00104
00110   void SetMousePosition(int x, int y);
00111
00116   std::tuple<int, int> GetMousePosition();
00117 };
00118
00119 #endif
```

5.45 src/ECS/ECS.cpp File Reference

```
#include "ECS.hpp"
#include <algorithm>
```

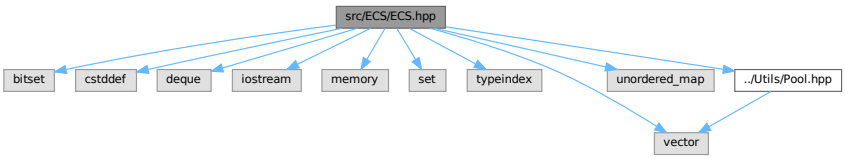
Include dependency graph for ECS.cpp:



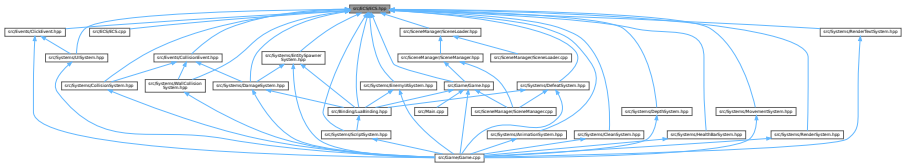
5.46 src/ECS/ECS.hpp File Reference

```
#include <bitset>
#include <cstddef>
#include <deque>
#include <iostream>
#include <memory>
#include <set>
#include <typeindex>
#include <vector>
#include <unordered_map>
#include "../Utils/Pool.hpp"
```

Include dependency graph for ECS.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- struct IComponent

  Base class for all components to generate unique IDs.
- class Component< TComponent >

  Template to generate unique component IDs per component type.
- class Entity

  Represents an entity in the ECS, identified by a unique ID.
- class System

  Base class for systems that operate on entities with specific components.
- class Registry

  Manages entities, components, and systems in the ECS.

Typedefs

- typedef std::bitset< MAX_COMPONENTS > Signature

  Signature type used to represent the set of components an entity has. Uses a bitset of MAX_COMPONENTS size.

Variables

- const unsigned int MAX_COMPONENTS = 64

  Maximum number of components supported by the ECS.

5.46.1 Typedef Documentation

5.46.1.1 Signature

```
typedef std::bitset<MAX_COMPONENTS> Signature
```

Signature type used to represent the set of components an entity has. Uses a bitset of MAX_COMPONENTS size.

5.46.2 Variable Documentation

5.46.2.1 MAX_COMPONENTS

```
const unsigned int MAX_COMPONENTS = 64
```

Maximum number of components supported by the ECS.

5.47 ECS.hpp

Go to the documentation of this file.

```
00001 #ifndef ECS_HPP
00002 #define ECS_HPP
00003
00004 #include <bitset>
00005 #include <cstddef>
00006 #include <deque>
00007 #include <iostream>
00008 #include <memory>
00009 #include <set>
00010 #include <typeindex>
00011 #include <vector>
00012 #include <unordered_map>
00013 #include "../Utils/Pool.hpp"
00014
00018 const unsigned int MAX_COMPONENTS = 64;
00019
00024 // Signature
00025 typedef std::bitset<MAX_COMPONENTS> Signature;
00026
00030 struct IComponent {
00031  protected:
00033   static int nextId;
00034 };
00035
00040 template <typename TComponent>
00041 class Component : public IComponent {
00042  public:
00047  static int GetId(){
00048     static int id = nextId++;
00049     return id;
00050  }
00051 };
00052
00057 class Entity{
00058  private:
00060   int id;
00061
00062  public:
00067   Entity(int id): id(id){}
00072   int GetId() const;
```

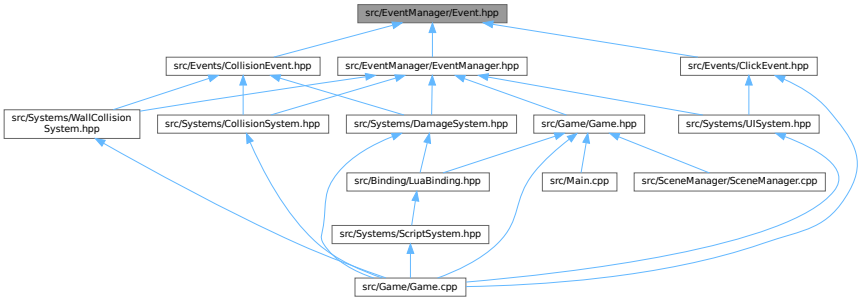
```
00076   void Kill();
00077
00078   // Comparison operators based on entity ID.
00079   bool operator ==(const Entity& other) const {return id == other.id; }
00080   bool operator !=(const Entity& other) const {return id != other.id; }
00081   bool operator >(const Entity& other) const {return id > other.id; }
00082   bool operator <(const Entity& other) const {return id < other.id; }
00083
00090   template <typename TComponent, typename... TArgs>
00091   void AddComponent(TArgs&&... args);
00092
00097   template <typename TComponent>
00098   void RemoveComponent();
00099
00105   template <typename TComponent>
00106   bool HasComponent() const;
00107
00113   template <typename TComponent>
00114   TComponent& GetComponent() const;
00115
00117   class Registry* registry;
00118 };
00119
00124 class System{
00125  private:
00127   Signature componentSignature;
00129   std::vector<Entity> entities;
00130
00131 public:
00132  System() = default;
00133  ~System() = default;
00134
00139  void AddEntityToSystem(Entity entity);
00140
00145  void RemoveEntityFromSystem(Entity entity);
00146
00151  std::vector<Entity> GetSystemEntities() const;
00152
00157  const Signature& GetComponentSignature() const;
00158
00163  template <typename TComponent>
00164  void RequireComponent();
00165
00166 };
00167
00172 class Registry {
00173  private:
00175   int numEntity = 0;
00176
00178   std::vector<std::shared_ptr<IPool» componentsPools;
00180   std::vector<Signature> entityComponentSignatures;
00181
00183   std::unordered_map<std::type_index, std::shared_ptr<System» systems;
00184
00186   std::set<Entity> entitiesToBeAdded;
00188   std::set<Entity> entitiesToBeKilled;
00189
00191   std::deque<int> freeIds;
00196   void RemoveAllComponentsOfEntity(Entity entity);
00197
00198   public:
00199    Registry();
00200    ~Registry();
00201
00202
00206    void Update();
00207
00208    //Entity Managment
00209
00214    Entity CreateEntity();
00219    void KillEntity(Entity entity);
00220
00221    //Component Managment
00222
00230    template <typename TComponent, typename... TArgs>
00231    void AddComponent(Entity entity, TArgs&&... args);
00232
00238    template <typename TComponent>
00239    void RemoveComponent(Entity entity);
00240
00247    template <typename TComponent>
00248    bool HasComponent(Entity entity) const;
00249
00256    template <typename TComponent>
00257    TComponent& GetComponent(Entity entity) const;
00258
00259     //System Managment
```

```
00260
00261
00268     template <typename TSystem, typename... TArgs>
00269     void AddSystem(TArgs&&... args);
00270
00275     template <typename TSystem>
00276     void RemoveSystem();
00277
00283     template <typename TSystem>
00284     bool HasSystem() const;
00285
00291     template <typename TSystem>
00292     TSystem& GetSystem() const;
00293
00294     // Add and remove entities to systems
00295
00300     void AddEntityToSystems(Entity entity);
00301
00306     void RemoveEntityFromSystems(Entity entity);
00307
00308     //Reset registry
00309
00310
00314     void ClearAllEntities();
00315
00316
00317 };
00318
00319
00320 template <typename TComponent>
00321 void System::RequireComponent(){
00322   const auto componentId = Component<TComponent>::GetId();
00323   componentSignature.set(componentId);
00324 }
00325
00326 template <typename TComponent, typename... TArgs>
00327 void Registry::AddComponent(Entity entity, TArgs&&... args){
00328     const int componentId = Component<TComponent>::GetId();
00329     const int entityId = entity.GetId();
00330
00331     if(static_cast<long unsigned int>(componentId) >= componentsPools.size()){
00332         componentsPools.resize(componentId + 10, nullptr);
00333     }
00334
00335     if(!componentsPools[componentId]){
00336         std::shared_ptr<Pool<TComponent» newComponentPool
00337         = std::make_shared<Pool<TComponent»();
00338         componentsPools[componentId] = newComponentPool;
00339     }
00340
00341     std::shared_ptr<Pool<TComponent» componentPool
00342       = std::static_pointer_cast<Pool<TComponent»(componentsPools[componentId]);
00343
00344     if(entityId >= componentPool->GetSize()){
00345         componentPool->Resize(numEntity + 100);
00346     }
00347
00348     TComponent newComponent(std::forward<TArgs>(args)...);
00349
00350     componentPool->Set(entityId, newComponent);
00351     entityComponentSignatures[entityId].set(componentId);
00352
00353     std::cout « "[Registry] Se agrega componente "«componentId
00354     «" a la entidad "« entityId «std::endl;
00355 }
00356
00357 template <typename TComponent>
00358 void Registry::RemoveComponent(Entity entity){
00359     const int componentId = Component<TComponent>::GetId();
00360     const int entityId = entity.GetId();
00361
00362     entityComponentSignatures[entityId].set(componentId, false);
00363 }
00364
00365 template <typename TComponent>
00366 bool Registry::HasComponent(Entity entity) const{
00367     const int componentId = Component<TComponent>::GetId();
00368     const int entityId = entity.GetId();
00369
00370     return entityComponentSignatures[entityId].test(componentId);
00371 }
00372
00373 template <typename TComponent>
00374 TComponent& Registry::GetComponent(Entity entity) const{
00375     const int componentId = Component<TComponent>::GetId();
00376     const int entityId = entity.GetId();
00377
```

```
00378     auto componentPool
00379      = std::static_pointer_cast<Pool<TComponent»(componentsPools[componentId]);
00380
00381     return componentPool->Get(entityId);
00382 }
00383
00384 template <typename TSystem, typename... TArgs>
00385 void Registry::AddSystem(TArgs&&... args){
00386     std::shared_ptr<TSystem>  newSystem
00387      = std::make_shared<TSystem>(std::forward<TArgs>(args)...);
00388     systems.insert(std::make_pair(std::type_index(typeid(TSystem)), newSystem));
00389 }
00390
00391 template <typename TSystem>
00392 void Registry::RemoveSystem(){
00393     auto system = system.find(std::type_index(typeid(TSystem)));
00394     systems.erase(system);
00395 }
00396
00397 template <typename TSystem>
00398 bool Registry::HasSystem() const{
00399     return systems.find(std::type_index(typeid(TSystem))) != systems.end();
00400
00401 }
00402
00403 template <typename TSystem>
00404 TSystem& Registry::GetSystem() const{
00405     auto system = systems.find(std::type_index(typeid(TSystem)));
00406     return *(std::static_pointer_cast<TSystem>(system->second));
00407
00408 }
00409
00410 template <typename TComponent, typename... TArgs>
00411 void Entity::AddComponent(TArgs&&... args){
00412     registry->AddComponent<TComponent>(*this, std::forward<TArgs>(args)...);
00413 }
00414
00415 template <typename TComponent>
00416 void Entity::RemoveComponent(){
00417     registry->RemoveComponent<TComponent>(*this);
00418 }
00419
00420 template <typename TComponent>
00421 bool Entity::HasComponent() const{
00422     return registry->HasComponent<TComponent>(*this);
00423 }
00424
00425 template <typename TComponent>
00426 TComponent& Entity::GetComponent() const{
00427     return registry->GetComponent<TComponent>(*this);
00428 }
00429
00430
00431 #endif
```

5.48 src/EventManager/Event.hpp File Reference

This graph shows which files directly or indirectly include this file:



Classes

- class Event

  Base class for events in the system.

5.49 Event.hpp

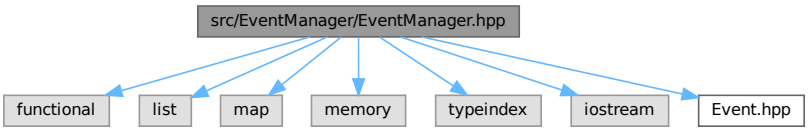
Go to the documentation of this file.

```
00001 #ifndef EVENT_HPP
00002 #define EVENT_HPP
00003
00010 class Event {
00011   public:
00015     Event() = default;
00016
00017 };
00018 #endif
```

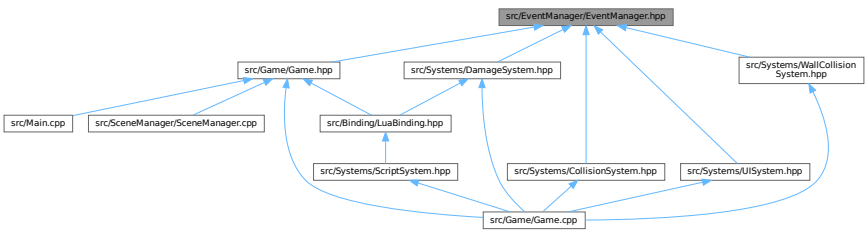
5.50 src/EventManager/EventManager.hpp File Reference

```
#include <functional>
#include <list>
#include <map>
#include <memory>
#include <typeindex>
#include <iostream>
#include "Event.hpp"
```

Include dependency graph for EventManager.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class IEventCallback

  Abstract interface for event callback handlers.
- class EventCallback< TOwner, TEvent >

  Template event callback handler connecting an owner and event type.
- class EventManager

  Manages event subscription and emission.

Typedefs

- typedef std::list< std::unique_ptr< IEventCallback > > HandlerList

5.50.1 Typedef Documentation

5.50.1.1 HandlerList

```
typedef std::list<std::unique_ptr<IEventCallback> > HandlerList
```

5.51 EventManager.hpp

Go to the documentation of this file.

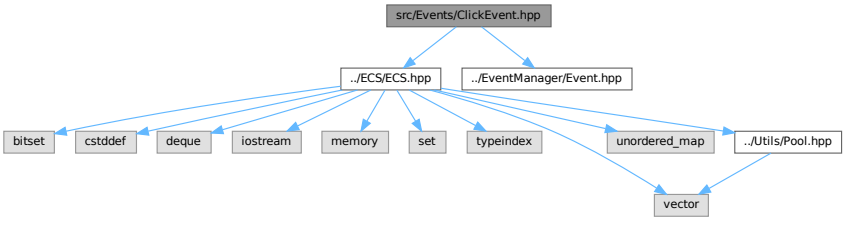
```
00001 #ifndef EVENTMANAGER_HPP
00002 #define EVENTMANAGER_HPP
00003
00004 #include <functional>
00005 #include <list>
00006 #include <map>
00007 #include <memory>
00008 #include <typeindex>
00009 #include <iostream>
00010 #include "Event.hpp"
00011
00012
00016 class IEventCallback {
00017   private:
00022    virtual void Call(Event& e) = 0;
00023
00024   public:
00028    virtual ~IEventCallback() = default;
00033    void Execute(Event& e){
00034     Call(e);
00035    }
00036 };
00037
00044 template <typename TOwner, typename TEvent>
00045 class EventCallback: public IEventCallback {
00046  private:
00047   typedef void (TOwner::*CallbackFunction)(TEvent&);
00048
00049   TOwner* ownerInstance;
00050   CallbackFunction callbackFunction;
00051
00056   virtual void Call(Event& e) override {
00057     std::invoke(callbackFunction, ownerInstance, static_cast<TEvent&>(e));
00058   }
00059
00060   public:
00066   EventCallback(TOwner* ownerInstance, CallbackFunction callbackFunction){
00067     this->callbackFunction = callbackFunction;
00068     this->ownerInstance = ownerInstance;
00069   }
00070 };
00071
00072 typedef std::list<std::unique_ptr<IEventCallback» HandlerList;
00073
```

```
00080 class EventManager {
00081  private:
00082   std::map<std::type_index, std::unique_ptr<HandlerList» subscribers;
00083
00084  public:
00088   EventManager(){
00089     std::cout« "[EventManager] Se ejecuta constructor" « std::endl;
00090   }
00091
00095  ~EventManager(){
00096     std::cout« "[EventManager] Se ejecuta destructor" « std::endl;
00097   }
00098
00099
00103   void Reset(){
00104     subscribers.clear();
00105   }
00106
00115   template <typename TEvent, typename TOwner>
00116   void SubscribeToEvent(TOwner* ownerInstance,
00117   void (TOwner::*callbackFunction)(TEvent&)) {
00118     if(!subscribers[typeid(TEvent)].get()){
00119         subscribers[typeid(TEvent)] = std::make_unique<HandlerList>();
00120     }
00121     auto subscriber = std::make_unique<EventCallback<TOwner, TEvent»(
00122         ownerInstance, callbackFunction);
00123     subscribers[typeid(TEvent)]->push_back(std::move(subscriber));
00124   }
00125
00133   template <typename TEvent, typename... TArgs>
00134   void EmitEvent(TArgs&&... args) {
00135     auto handlers = subscribers[typeid(TEvent)].get();
00136     if(handlers){
00137         for (auto it = handlers->begin(); it != handlers->end(); it++){
00138             auto handler = it->get();
00139             TEvent event (std::forward<TArgs>(args)...);
00140             handler->Execute(event);
00141         }
00142     }
00143   }
00144 };
00145 #endif
```

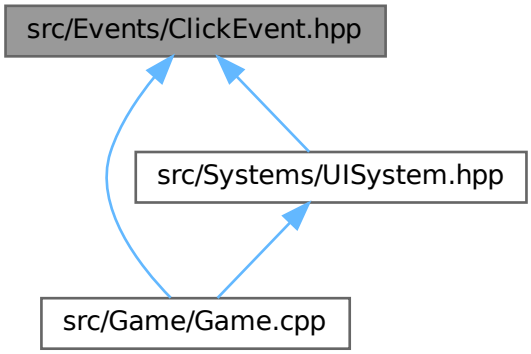
5.52 src/Events/ClickEvent.hpp File Reference

```
#include "../ECS/ECS.hpp"
#include "../EventManager/Event.hpp"
```

Include dependency graph for ClickEvent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class ClickEvent

  Represents a mouse click event.

5.53 ClickEvent.hpp

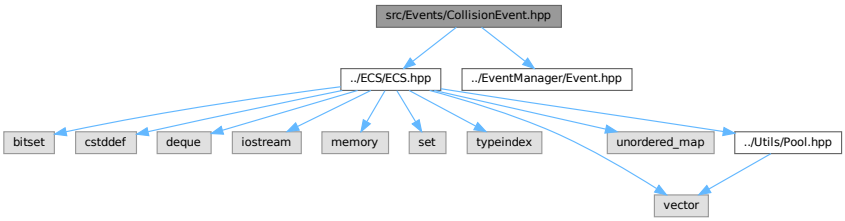
Go to the documentation of this file.

```
00001 #ifndef CLICKEVENT_HPP
00002 #define CLICKEVENT_HPP
00003
00004 #include "../ECS/ECS.hpp"
00005 #include "../EventManager/Event.hpp"
00006
00010 class ClickEvent : public Event {
00011     public:
00012     int buttonCode;
00013     int posX;
00014     int posY;
00015
00022     ClickEvent(int buttonCode = 0, int posX = 0, int posY = 0){
00023         this->buttonCode = buttonCode;
00024         this->posX = posX;
00025         this->posY = posY;
00026     }
00027 };
00028
00029 #endif
```

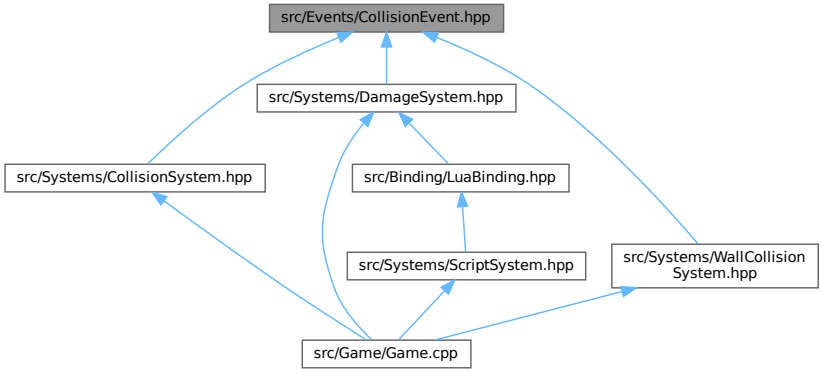
5.54 src/Events/CollisionEvent.hpp File Reference

```
#include "../ECS/ECS.hpp"
#include "../EventManager/Event.hpp"
```

Include dependency graph for CollisionEvent.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CollisionEvent

  Event triggered when two entities collide.

5.55 CollisionEvent.hpp

Go to the documentation of this file.

```
00001 #ifndef COLLISIONEVENT_HPP
00002 #define COLLISIONEVENT_HPP
00003
00004 #include "../ECS/ECS.hpp"
00005 #include "../EventManager/Event.hpp"
00006
00007
00011 class CollisionEvent : public Event {
00012  public:
00013  Entity a;
00014  Entity b;
00015
00021  CollisionEvent(Entity a, Entity b) : a(a), b(b) {}
00022 };
00023 #endif
```

5.56 src/Game/Game.cpp File Reference

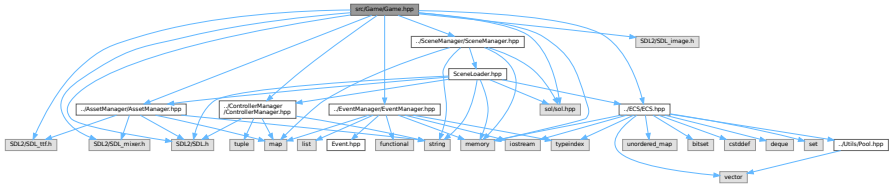
```
#include "Game.hpp"
#include <iostream>
#include "../Events/ClickEvent.hpp"
#include "../Systems/CollisionSystem.hpp"
#include "../Systems/AnimationSystem.hpp"
#include "../Systems/DamageSystem.hpp"
#include "../Systems/WallCollisionSystem.hpp"
#include "../Systems/RenderSystem.hpp"
#include "../Systems/MovementSystem.hpp"
#include "../Systems/DepthSystem.hpp"
#include "../Systems/RenderTextSystem.hpp"
#include "../Systems/ScriptSystem.hpp"
#include "../Systems/UISystem.hpp"
#include "../Systems/SceneTimeSystem.hpp"
#include "../Systems/EntitySpawnerSystem.hpp"
#include "../Systems/CleanSystem.hpp"
#include "../Systems/DefeatSystem.hpp"
#include "../Systems/EnemyIASystem.hpp"
#include "../Systems/HealthBarSystem.hpp"
```

Include dependency graph for Game.cpp:

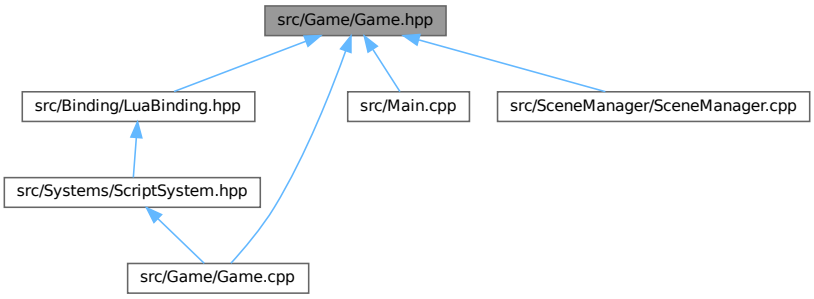
5.57 src/Game/Game.hpp File Reference

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_image.h>
#include <SDL2/SDL_ttf.h>
#include <SDL2/SDL_mixer.h>
#include <memory>
#include <sol/sol.hpp>
#include "../AssetManager/AssetManager.hpp"
#include "../ControllerManager/ControllerManager.hpp"
#include "../EventManager/EventManager.hpp"
#include "../ECS/ECS.hpp"
#include "../SceneManager/SceneManager.hpp"
```

Include dependency graph for Game.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class Game

  Core class that manages the entire game lifecycle.

Variables

- const int FPS = 30

  Frames per second target.
- const int MILISECS_PER_FRAME = 1000 / FPS

  Milliseconds per frame.

5.57.1 Variable Documentation

5.57.1.1 FPS

```
const int FPS = 30
```

Frames per second target.

5.57.1.2 MILISECS_PER_FRAME

```
const int MILISECS_PER_FRAME = 1000 / FPS
```

Milliseconds per frame.

5.58 Game.hpp

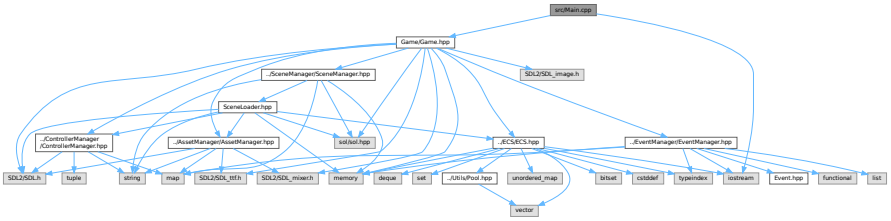
Go to the documentation of this file.

```
00001 #ifndef GAME_HPP
00002 #define GAME_HPP
00003
00004 #include <SDL2/SDL.h>
00005 #include <SDL2/SDL_image.h>
00006 #include <SDL2/SDL_ttf.h>
00007 #include <SDL2/SDL_mixer.h>
00008
00009 #include <memory>
00010 #include <sol/sol.hpp>
00011
00012 #include "../AssetManager/AssetManager.hpp"
00013 #include "../ControllerManager/ControllerManager.hpp"
00014 #include "../EventManager/EventManager.hpp"
00015 #include "../ECS/ECS.hpp"
00016 #include "../SceneManager/SceneManager.hpp"
00017
00018 const int FPS = 30;
00019 const int MILISECS_PER_FRAME = 1000 / FPS;
00020
00028 class Game {
00029 private:
00030     SDL_Window* window = nullptr;
00031
00032
00033     int windowWidth = 0;
00034     int windowHeight = 0;
00035
00036     int milisecsPreviousFrame = 0;
00037
00038     bool isRunning = false;
00039     bool isPaused = false;
00040     bool wasPaused = false;
00041
00042
00043 public:
00044     SDL_Renderer* renderer = nullptr;
00045     std::unique_ptr<AssetManager> assetManager;
00046     std::unique_ptr<EventManager> eventManager;
00047     std::unique_ptr<ControllerManager> controllerManager;
00048     std::unique_ptr<Registry> registry;
00049     std::unique_ptr<SceneManager> sceneManager;
00050     sol::state lua;
00051
00052 private:
00053     void Setup();
00054     void RunScene();
00055     void ProcessInput();
00056     void Update();
00057     void Render();
00058
00059      Game();
00060     ~Game();
00061
00062 public:
00067     static Game& GetInstance();
00071     void Init();
00075     void Run();
00079     void Destroy();
00080 };
00081 #endif
```

5.59 src/Main.cpp File Reference

```
#include <iostream>
#include "Game/Game.hpp"
```

Include dependency graph for Main.cpp:



Functions

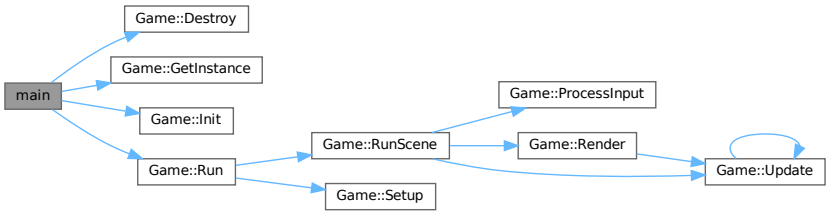
- int main (int argc, char ∗argv[ ])

5.59.1 Function Documentation

5.59.1.1 main()

```
int main (
            int argc,
            char * argv[] )
```

Here is the call graph for this function:



5.60 src/SceneManager/SceneLoader.cpp File Reference

```
#include "SceneLoader.hpp"
#include <glm/glm.hpp>
#include <vector>
#include <iostream>
#include "../Components/CircleColliderComponent.hpp"
#include "../Components/PolygonColliderComponent.hpp"
#include "../Components/DepthComponent.hpp"
#include "../Components/AnimationComponent.hpp"
#include "../Components/EntitySpawnerComponent.hpp"
#include "../Components/ClickableComponent.hpp"
#include "../Components/RigidBodyComponent.hpp"
#include "../Components/SpriteComponent.hpp"
```

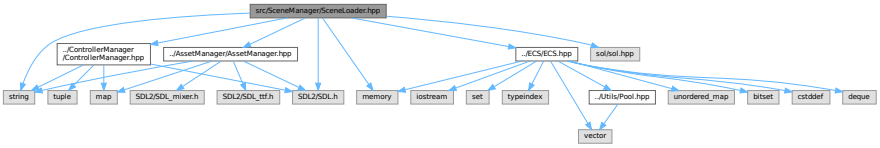
```
#include "../Components/ScriptComponent.hpp"
#include "../Components/TransformComponent.hpp"
#include "../Components/TextComponent.hpp"
#include "../Components/TagWallComponent.hpp"
#include "../Components/TagObjectiveComponent.hpp"
#include "../Components/TagEnemyComponent.hpp"
#include "../Components/TagPlayerComponent.hpp"
#include "../Components/TagProjectileComponent.hpp"
#include "../Components/LifeComponent.hpp"
#include "../Components/DamageComponent.hpp"
```

Include dependency graph for SceneLoader.cpp:

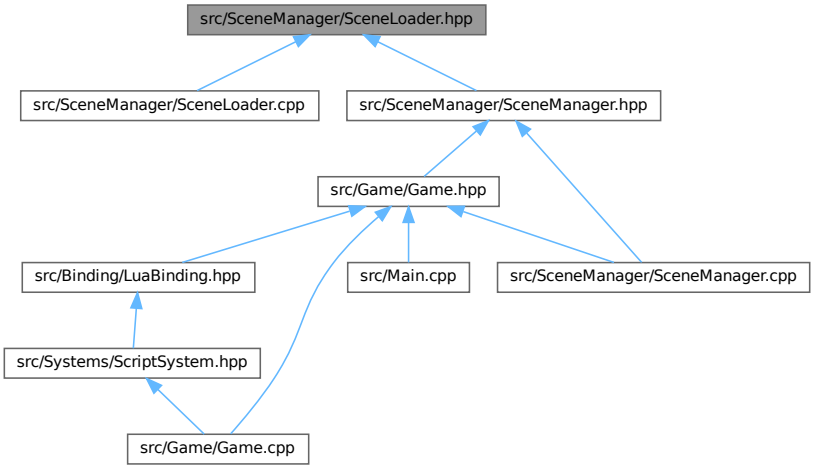
5.61 src/SceneManager/SceneLoader.hpp File Reference

```
#include <SDL2/SDL.h>
#include <memory>
#include <sol/sol.hpp>
#include <string>
#include "../AssetManager/AssetManager.hpp"
#include "../ControllerManager/ControllerManager.hpp"
#include "../ECS/ECS.hpp"
```

Include dependency graph for SceneLoader.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class SceneLoader

  Loads a game scene from a Lua script.

5.62 SceneLoader.hpp

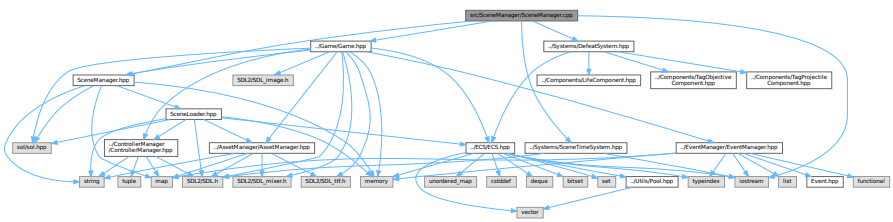
Go to the documentation of this file.

```
00001 #ifndef SCENELOADER_HPP
00002 #define SCENELOADER_HPP
00003
00004 #include <SDL2/SDL.h>
00005
00006 #include <memory>
00007 #include <sol/sol.hpp>
00008 #include <string>
00009
00010 #include "../AssetManager/AssetManager.hpp"
00011 #include "../ControllerManager/ControllerManager.hpp"
00012 #include "../ECS/ECS.hpp"
00013
00021 class SceneLoader {
00022   private:
00029   void LoadSprites(SDL_Renderer* render, const sol::table& sprites
00030   , std::unique_ptr<AssetManager>& assetManager);
00031
00037    void LoadFonts(const sol::table& fonts
00038   , std::unique_ptr<AssetManager>& assetManager);
00039
00045   void LoadKeys(const sol::table& keys, std::unique_ptr<ControllerManager>&
00046   controllerManager);
00047
00053   void LoadButtons(const sol::table& buttons, std::unique_ptr<ControllerManager>&
00054   controllerManager);
00055
00062   void LoadEntities(sol::state& lua, const sol::table& entites,
00063   std::unique_ptr<Registry>& registry);
00064
00070   void LoadMusic(const sol::table& musicTable, std::unique_ptr<AssetManager>& assetManager);
00071
00072   public:
00073   SceneLoader();
00074   ~SceneLoader();
00075
00085   void LoadScene(const std::string& scenePath, sol::state& lua
00086   , std::unique_ptr<AssetManager>& assetManager
00087   , std::unique_ptr<ControllerManager>& controllerManager
00088   , std::unique_ptr<Registry>& registry, SDL_Renderer* renderer);
00089 };
00090
00091 #endif
```

5.63 src/SceneManager/SceneManager.cpp File Reference

```
#include "SceneManager.hpp"
#include <iostream>
#include "../Game/Game.hpp"
#include "../Systems/SceneTimeSystem.hpp"
#include "../Systems/DefeatSystem.hpp"
```

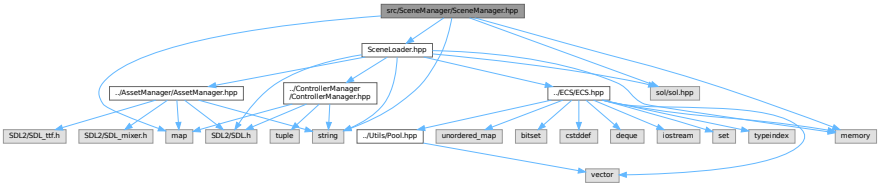
Include dependency graph for SceneManager.cpp:



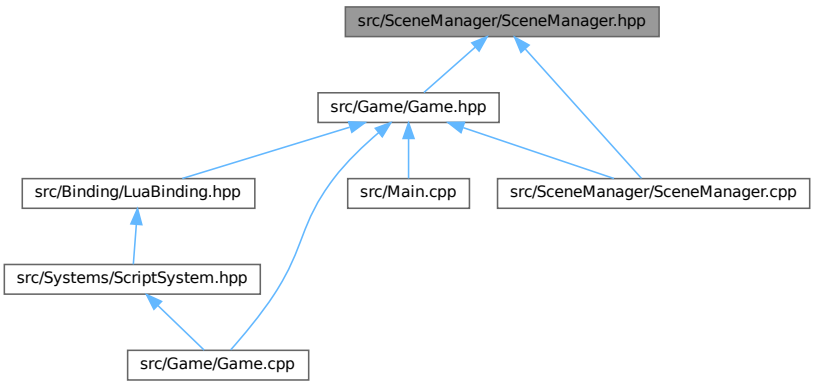
5.64 src/SceneManager/SceneManager.hpp File Reference

```
#include <map>
#include <memory>
#include <sol/sol.hpp>
#include <string>
#include "SceneLoader.hpp"
```

Include dependency graph for SceneManager.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class SceneManager

  Manages scenes and handles scene transitions.

5.65 SceneManager.hpp

Go to the documentation of this file.

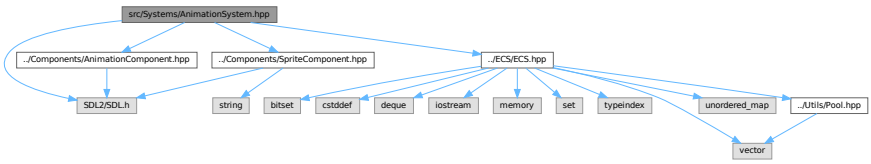
```
00001 #ifndef SCENEMANAGER_HPP
00002 #define SCENEMANAGER_HPP
00003
00004 #include <map>
00005 #include <memory>
00006 #include <sol/sol.hpp>
00007 #include <string>
00008
00009 #include "SceneLoader.hpp"
00010
```

```
00011
00015 class SceneManager{
00016  private:
00017   std::map<std::string, std::string> scenes;
00018   std::string nextScene;
00019   bool isSceneRunning = false;
00020   std::unique_ptr<SceneLoader> sceneLoader;
00021
00022 public:
00026  SceneManager();
00030  ~SceneManager();
00031
00037  void LoadSceneFromScript(const std::string& path, sol::state& lua);
00041  void LoadScene();
00042
00047  std::string GetNextScene() const;
00048
00053  void SetNextScene(const std::string& nextScene);
00058  bool IsSceneRunning() const;
00062  void StartScene();
00063
00067  void StopScene();
00068 };
00069 #endif
```

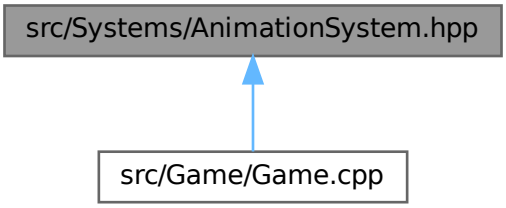
5.66 src/Systems/AnimationSystem.hpp File Reference

```
#include <SDL2/SDL.h>
#include "../Components/AnimationComponent.hpp"
#include "../Components/SpriteComponent.hpp"
#include "../ECS/ECS.hpp"
```

Include dependency graph for AnimationSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class AnimationSystem

  System responsible for updating the animation frames of entities.

5.67 AnimationSystem.hpp

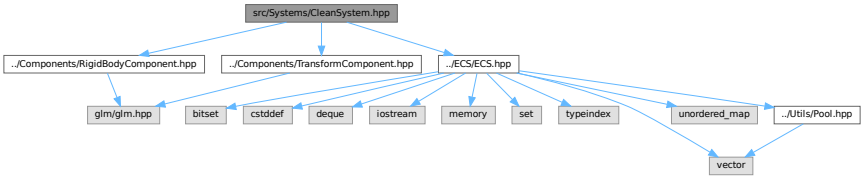
Go to the documentation of this file.

```
00001 #ifndef ANIMATIONSYSTEM_HPP
00002 #define ANIMATIONSYSTEM_HPP
00003
00004 #include <SDL2/SDL.h>
00005
00006 #include "../Components/AnimationComponent.hpp"
00007 #include "../Components/SpriteComponent.hpp"
00008 #include "../ECS/ECS.hpp"
00009
00018 class AnimationSystem : public System{
00019   public:
00023    AnimationSystem() {
00024     RequireComponent<AnimationComponent>();
00025     RequireComponent<SpriteComponent>();
00026    }
00027
00035    void Update(){
00036     for(auto entity : GetSystemEntities()){
00037         auto& animation = entity.GetComponent<AnimationComponent>();
00038         auto& sprite = entity.GetComponent<SpriteComponent>();
00039
00040         animation.currentFrame = ((SDL_GetTicks() - animation.startTime) *
      animation.frameSpeedRate/1000)
00041         % animation.numFrames;
00042
00043         sprite.srcRect.x = animation.currentFrame * sprite.width;
00044     }
00045    }
00046 };
00047 #endif
```

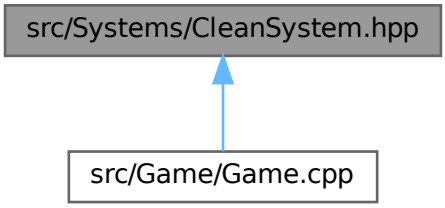
5.68 src/Systems/CleanSystem.hpp File Reference

```
#include "../Components/RigidBodyComponent.hpp"
#include "../Components/TransformComponent.hpp"
#include "../ECS/ECS.hpp"
```

Include dependency graph for CleanSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CleanSystem

  System responsible for removing entities that move outside defined bounds.

Macros

- #define CLEANTSYSTEM_HPP

5.68.1 Macro Definition Documentation

5.68.1.1 CLEANTSYSTEM_HPP

```
#define CLEANTSYSTEM_HPP
```

5.69 CleanSystem.hpp

Go to the documentation of this file.

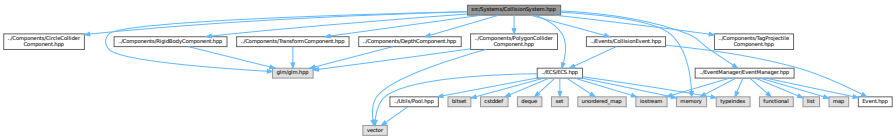
```
00001 #ifndef CLEANSYSTEM_HPP
00002 #define CLEANTSYSTEM_HPP
00003
00004 #include "../Components/RigidBodyComponent.hpp"
00005 #include "../Components/TransformComponent.hpp"
00006 #include "../ECS/ECS.hpp"
00007
00016 class CleanSystem : public System{
00017     public:
00021      CleanSystem(){
00022         RequireComponent<RigidBodyComponent>();
00023         RequireComponent<TransformComponent>();
00024
00025      }
00032     void Update() {
00033         for(auto entity : GetSystemEntities()){
00034             auto& transform = entity.GetComponent<TransformComponent>();
00035             if(transform.position.x > 2000 ||transform.position.x < -400){
00036                 entity.Kill();
00037                 std::cout«"[CleanSystem] Se elimina entidad "«entity.GetId()
00038                 «std::endl;
00039             }
00040         }
00041     }
00042 };
00043
00044 #endif
```

5.70 src/Systems/CollisionSystem.hpp File Reference

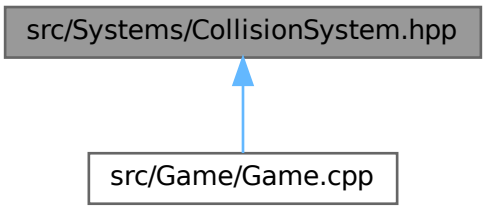
```
#include "../Components/CircleColliderComponent.hpp"
#include "../Components/PolygonColliderComponent.hpp"
#include "../Components/RigidBodyComponent.hpp"
#include "../Components/TransformComponent.hpp"
#include "../Components/TagProjectileComponent.hpp"
#include "../Components/DepthComponent.hpp"
#include "../ECS/ECS.hpp"
#include "../EventManager/EventManager.hpp"
#include "../Events/CollisionEvent.hpp"
#include <glm/glm.hpp>
```

```
#include <memory>
```

Include dependency graph for CollisionSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class CollisionSystem

  System responsible for detecting collisions between entities and emitting collision events.

Macros

- #define COLLISIONSYSTEM_HPP

5.70.1 Macro Definition Documentation

5.70.1.1 COLLISIONSYSTEM_HPP

```
#define COLLISIONSYSTEM_HPP
```

5.71 CollisionSystem.hpp

Go to the documentation of this file.

```
00001 #ifndef COLLISIONSYSTEM_HPP
00002 #define COLLISIONSYSTEM_HPP
00003
00004 #include "../Components/CircleColliderComponent.hpp"
00005 #include "../Components/PolygonColliderComponent.hpp"
00006 #include "../Components/RigidBodyComponent.hpp"
00007 #include "../Components/TransformComponent.hpp"
00008 #include "../Components/TagProjectileComponent.hpp"
00009 #include "../Components/DepthComponent.hpp"
00010 #include "../ECS/ECS.hpp"
00011 #include "../EventManager/EventManager.hpp"
00012 #include "../Events/CollisionEvent.hpp"
00013
00014 #include <glm/glm.hpp>
00015
00016 #include <memory>
00017
00028 class CollisionSystem : public System {
00029 public:
00033     CollisionSystem() {
00034         RequireComponent<TransformComponent>();
00035     }
00036
00044     void Update(std::unique_ptr<EventManager>& eventManager) {
00045         auto entities = GetSystemEntities();
00046
00047         for (auto i = entities.begin(); i != entities.end(); i++) {
00048             Entity a = *i;
00049             auto aTransform = a.GetComponent<TransformComponent>();
00050
00051             for (auto j = i; j != entities.end(); j++) {
00052                 Entity b = *j;
00053                 if (a == b) {
00054                     continue;
00055                 }
00056                 auto bTransform = b.GetComponent<TransformComponent>();
00057
00058                 bool collision = false;
00059
00060                 bool aHasCircle = a.HasComponent<CircleColliderComponent>();
00061                 bool aHasPolygon = a.HasComponent<PolygonColliderComponent>();
00062                 bool bHasCircle = b.HasComponent<CircleColliderComponent>();
00063                 bool bHasPolygon = b.HasComponent<PolygonColliderComponent>();
00064
00065                 if (aHasCircle && bHasCircle) {
00066                     collision = CheckCircleVsCircle(a, b, aTransform, bTransform);
00067                 } else if (bHasCircle && aHasPolygon) {
00068                     collision = CheckCircleVsPolygon(b, a, bTransform);
00069                 } else if (aHasCircle && bHasPolygon) {
00070                     collision = CheckCircleVsPolygon(a, b, aTransform);
00071                 }
00072
00073                 if (collision) {
00074                     eventManager->EmitEvent<CollisionEvent>(a, b);
00075                 }
00076             }
00077         }
00078     }
00079
00080 private:
00092     bool CheckCircleVsCircle(Entity a, Entity b, TransformComponent aTransform, TransformComponent
    bTransform) {
00093         auto aCollider = a.GetComponent<CircleColliderComponent>();
00094         auto bCollider = b.GetComponent<CircleColliderComponent>();
00095         auto aDepth = a.GetComponent<DepthComponent>();
00096         auto bDepth = b.GetComponent<DepthComponent>();
00097
00098         glm::vec2 aCenterPos = glm::vec2(
00099                 aTransform.position.x - (aCollider.width / 2)* aTransform.scale.x,
00100                 aTransform.position.y - (aCollider.height / 2) * aTransform.scale.y
00101             );
00102
00103         glm::vec2 bCenterPos = glm::vec2(
00104                 bTransform.position.x - (bCollider.width / 2) * bTransform.scale.x,
00105                 bTransform.position.y - (bCollider.height / 2)* bTransform.scale.y
00106             );
00107
00108         int aRadius = aCollider.radius * aTransform.scale.x;
00109         int bRadius = bCollider.radius * bTransform.scale.x;
00110
00111         glm::vec2 dif = aCenterPos - bCenterPos;
00112         double length = glm::sqrt((dif.x * dif.x) + (dif.y * dif.y));
```

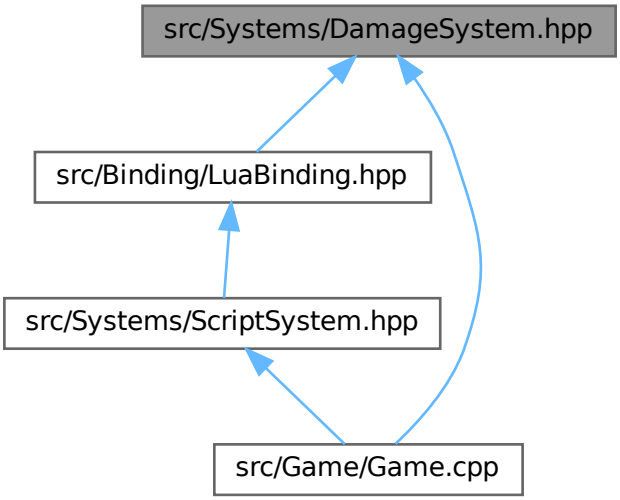
```
00113
00114
00115         int aScale = glm::floor(aTransform.scale.y*10/aDepth.max_scale);
00116         int bScale = glm::floor(bTransform.scale.y*10/bDepth.max_scale);
00117
00118         return (aRadius + bRadius) >= length && aScale == bScale;
00119     }
00120
00128     float DistancePointToSegment(glm::vec2 p, glm::vec2 a, glm::vec2 b) {
00129         glm::vec2 ab = b - a;
00130         glm::vec2 ap = p - a;
00131         float t = glm::dot(ap, ab) / glm::dot(ab, ab);
00132         t = glm::clamp(t, 0.0f, 1.0f);
00133         glm::vec2 closest = a + t * ab;
00134         return glm::length(p - closest);
00135     }
00146 bool CheckCircleVsPolygon(Entity circleEntity, Entity polygonEntity, TransformComponent
      circleTransform) {
00147     auto circleCollider = circleEntity.GetComponent<CircleColliderComponent>();
00148     auto polygonCollider = polygonEntity.GetComponent<PolygonColliderComponent>();
00149
00150     glm::vec2 circleCenter = circleTransform.position;
00151
00152
00153     circleCenter.x += (circleCollider.width * circleTransform.scale.x) / 2.0f;
00154     circleCenter.y += (circleCollider.height * circleTransform.scale.y) / 2.0f;
00155
00156
00157     float radius = circleCollider.radius * circleTransform.scale.x;
00158
00159     int count = polygonCollider.vertices.size();
00160     for (int i = 0, j = count - 1; i < count; j = i++) {
00161         if (DistancePointToSegment(circleCenter, polygonCollider.vertices[j],
      polygonCollider.vertices[i]) <= radius) {
00162             return true;
00163         }
00164     }
00165
00166
00167     return false;
00168 }
00169
00170 };
00171
00172 #endif
```

5.72 src/Systems/DamageSystem.hpp File Reference

```
#include <iostream>
#include <memory>
#include "../Components/TagWallComponent.hpp"
#include "../Components/TagObjectiveComponent.hpp"
#include "../Components/TagEnemyComponent.hpp"
#include "../Components/TagProjectileComponent.hpp"
#include "../Components/TagPlayerComponent.hpp"
#include "../Components/LifeComponent.hpp"
#include "../Components/DamageComponent.hpp"
#include "../Components/DepthComponent.hpp"
#include "../Systems/EntitySpawnerSystem.hpp"
#include "../ECS/ECS.hpp"
#include "../EventManager/EventManager.hpp"
#include "../Events/CollisionEvent.hpp"
```

Include dependency graph for DamageSystem.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class DamageSystem

  Handles damage application between entities when collisions occur.

5.73 DamageSystem.hpp

Go to the documentation of this file.

```
00001 #ifndef DAMAGESYSTEM_HPP
00002 #define DAMAGESYSTEM_HPP
00003
00004 #include <iostream>
00005 #include <memory>
00006
00007 #include "../Components/TagWallComponent.hpp"
00008 #include "../Components/TagObjectiveComponent.hpp"
00009 #include "../Components/TagEnemyComponent.hpp"
00010 #include "../Components/TagProjectileComponent.hpp"
00011 #include "../Components/TagPlayerComponent.hpp"
00012 #include "../Components/LifeComponent.hpp"
00013 #include "../Components/DamageComponent.hpp"
00014 #include "../Components/DepthComponent.hpp"
00015 #include "../Systems/EntitySpawnerSystem.hpp"
00016 #include "../ECS/ECS.hpp"
00017 #include "../EventManager/EventManager.hpp"
00018 #include "../Events/CollisionEvent.hpp"
00019
00028 class DamageSystem : public System {
00029  public:
00033   DamageSystem(){
00034     RequireComponent<CircleColliderComponent>();
00035   }
00040   void SubscribeToCollisionEvent(std::unique_ptr<EventManager>& eventManager){
00041     eventManager->SubscribeToEvent<CollisionEvent, DamageSystem>(this,
00042     &DamageSystem::OnCollision);
00043   }
00050 void OnCollision(CollisionEvent& e){
00051     std::cout« "[DamageSystem] Colisión de la entidad " « e.a.GetId()
```

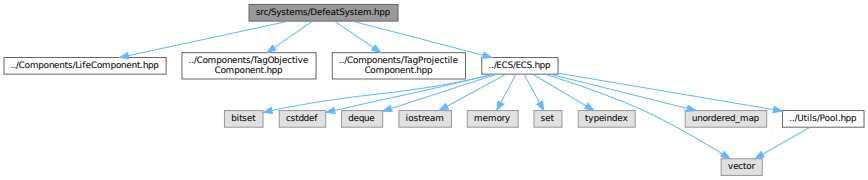
```
00052     « " y " « e.b.GetId() « std::endl;
00053
00054     if(e.a.HasComponent<LifeComponent>() && e.b.HasComponent<LifeComponent>()){
00055       auto& alife = e.a.GetComponent<LifeComponent>().life_count;
00056       auto& blife = e.b.GetComponent<LifeComponent>().life_count;
00057       auto& aScale = e.a.GetComponent<TransformComponent>().scale.x;
00058       auto& bScale = e.b.GetComponent<TransformComponent>().scale.x;
00059       auto& aMaxScale = e.a.GetComponent<DepthComponent>().max_scale;
00060       auto& bMaxScale = e.b.GetComponent<DepthComponent>().max_scale;
00061
00062       if((e.a.HasComponent<TagEnemyComponent>() && !e.b.HasComponent<TagEnemyComponent>() &&
00063       e.b.HasComponent<TagProjectileComponent>() )
00064       || (e.b.HasComponent<TagEnemyComponent>() && !e.a.HasComponent<TagEnemyComponent>()
00065       && e.a.HasComponent<TagProjectileComponent>()) ){
00066           alife -= e.b.GetComponent<DamageComponent>().damage_dealt;
00067           blife -= e.a.GetComponent<DamageComponent>().damage_dealt;
00068       }
00069
00070       if(e.a.HasComponent<TagEnemyComponent>() && e.b.HasComponent<TagObjectiveComponent>()
00071       && !e.b.HasComponent<TagProjectileComponent>()){
00072           blife -= e.a.GetComponent<DamageComponent>().damage_dealt;
00073           if(e.a.HasComponent<TagEnemyComponent>()&& e.a.HasComponent<TagProjectileComponent>()){
00074               alife -= 1;
00075           }
00076       }
00077       if(e.b.HasComponent<TagEnemyComponent>() && e.a.HasComponent<TagObjectiveComponent>()
00078       && !e.a.HasComponent<TagProjectileComponent>()){
00079           alife -= e.b.GetComponent<DamageComponent>().damage_dealt;
00080           if(e.b.HasComponent<TagEnemyComponent>()&& e.b.HasComponent<TagProjectileComponent>()){
00081               blife -= 1;
00082           }
00083       }
00084     bool oneExplosion = true;
00085       if(alife <= 0){
00086           double scale = (aScale*6)/aMaxScale;
00087           CreateExplosion(e.a,10,scale);
00088           e.a.Kill();
00089           oneExplosion = false;
00090       }
00091       if(blife <= 0){
00092         if(oneExplosion){
00093            double scale = (bScale*6)/bMaxScale;
00094            CreateExplosion(e.b,10,scale);
00095         }
00096           e.b.Kill();
00097       }
00098
00099
00100
00101     }
00102
00103 }
00104
00108 void DestroyAllEnemies (){
00109        for(auto entity : GetSystemEntities()){
00110             if(entity.HasComponent<TagEnemyComponent>() &&
      !entity.HasComponent<TagPlayerComponent>()){
00111                 auto& Scale = entity.GetComponent<TransformComponent>().scale.x;
00112                 auto& MaxScale = entity.GetComponent<DepthComponent>().max_scale;
00113                 double scale = (Scale*6)/MaxScale;
00114                 CreateExplosion(entity,10,scale);
00115                 entity.Kill();
00116             }
00117         }
00118 }
00126 void CreateExplosion(Entity entity, int num, double scale){
00127     auto& transform = entity.GetComponent<TransformComponent>();
00128     Entity newEntity = Game::GetInstance().registry->GetSystem<EntitySpawnerSystem>().GenerateEntity(
00129         Game::GetInstance().registry,num,Game::GetInstance().lua
00130     );
00131         auto& transformNew = newEntity.GetComponent<TransformComponent>();
00132         transformNew.position = transform.position;
00133         transformNew.scale.x = scale;
00134         transformNew.scale.y = scale;
00135 }
00136 };
00137 #endif
```

5.74 src/Systems/DefeatSystem.hpp File Reference

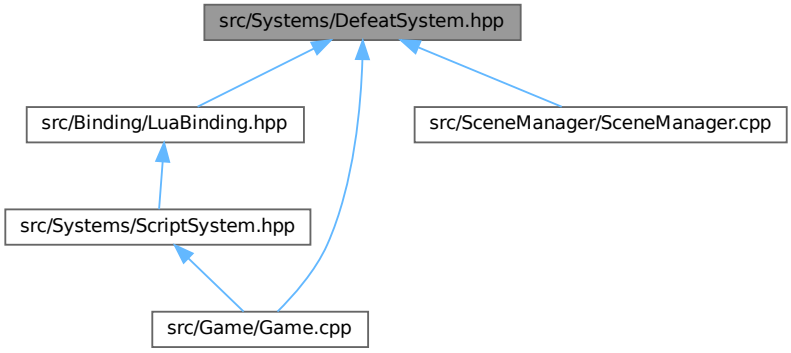
```
#include "../Components/LifeComponent.hpp"
#include "../Components/TagObjectiveComponent.hpp"
```

```
#include "../Components/TagProjectileComponent.hpp"
#include "../ECS/ECS.hpp"
```

Include dependency graph for DefeatSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class DefeatSystem

  Checks if any objective entity is defeated (life <= 0).

5.75 DefeatSystem.hpp

Go to the documentation of this file.

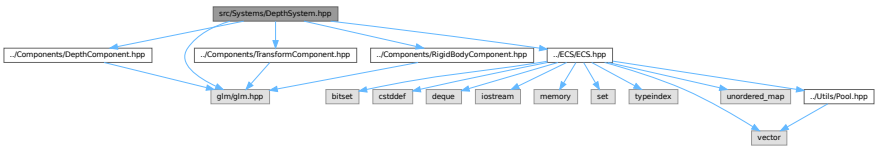
```
00001 #ifndef DEFEATYSTEM_HPP
00002 #define DEFEATYSTEM_HPP
00003
00004 #include "../Components/LifeComponent.hpp"
00005 #include "../Components/TagObjectiveComponent.hpp"
00006 #include "../Components/TagProjectileComponent.hpp"
00007 #include "../ECS/ECS.hpp"
00008
00016 class DefeatSystem : public System{
00017     public:
00021      bool Defeat = false;
00022      DefeatSystem(){
00023         RequireComponent<LifeComponent>();
00024         RequireComponent<TagObjectiveComponent>();
00025
00026      }
00033     void Update() {
00034         for(auto entity : GetSystemEntities()){
00035             const auto& life = entity.GetComponent<LifeComponent>();
```

```
00036             if(life.life_count <= 0){
00037                 if(!entity.HasComponent<TagProjectileComponent>()){
00038                     Defeat = true;
00039                     break;
00040                 }
00041             }
00042 
00043         }
00044     }
00045 };
00046 
00047 #endif
```

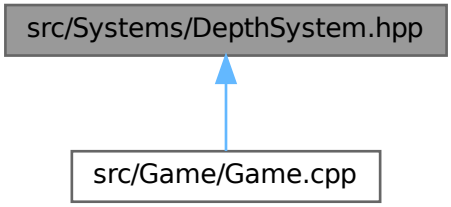
5.76 src/Systems/DepthSystem.hpp File Reference

```
#include "../Components/DepthComponent.hpp"
#include "../Components/TransformComponent.hpp"
#include "../Components/RigidBodyComponent.hpp"
#include <glm/glm.hpp>
#include "../ECS/ECS.hpp"
```

Include dependency graph for DepthSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class DepthSystem

  System to handle the scaling of entities based on their vertical velocity, simulating depth.

Macros

- #define DEPTHTSYSTEM_HPP

5.76.1 Macro Definition Documentation

5.76.1.1 DEPTHTSYSTEM_HPP

```
#define DEPTHTSYSTEM_HPP
```

5.77 DepthSystem.hpp

Go to the documentation of this file.

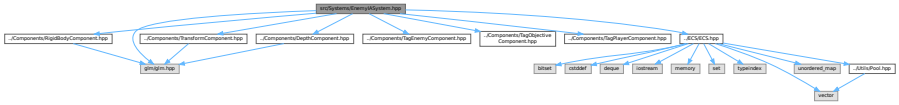
```
00001 #ifndef DEPTHSYSTEM_HPP
00002 #define DEPTHTSYSTEM_HPP
00003
00004 #include "../Components/DepthComponent.hpp"
00005 #include "../Components/TransformComponent.hpp"
00006 #include "../Components/RigidBodyComponent.hpp"
00007
00008 #include <glm/glm.hpp>
00009 #include "../ECS/ECS.hpp"
00010
00019 class DepthSystem : public System{
00020     public:
00024      DepthSystem(){
00025         RequireComponent<DepthComponent>();
00026         RequireComponent<TransformComponent>();
00027          RequireComponent<RigidBodyComponent>();
00028
00029      }
00037 void Update() {
00038     for(auto entity : GetSystemEntities()){
00039         auto& depth = entity.GetComponent<DepthComponent>();
00040         auto& rigidBody = entity.GetComponent<RigidBodyComponent>();
00041         auto& transform = entity.GetComponent<TransformComponent>();
00042
00043         float previous_scale = transform.scale.y;
00044         float current_scale = previous_scale;
00045
00046         if(rigidBody.velocity.y < 0) {
00047             // Movimiento hacia arriba = pequeno
00048             current_scale = glm::max(depth.min_scale, current_scale - depth.scale_speed);
00049         }
00050         else if(rigidBody.velocity.y > 0) {
00051             // Movimiento hacia abajo = grande
00052             current_scale = glm::min(depth.max_scale, current_scale + depth.scale_speed);
00053         }
00054
00055         if(current_scale != previous_scale) {
00056             // Reposicionar para mantener el centro
00057             float width_diff = depth.original_width * (current_scale - previous_scale);
00058             transform.position.x -= (width_diff / 2.0f);
00059
00060             transform.scale.x = current_scale;
00061             transform.scale.y = current_scale;
00062         }
00063     }
00064 }
00065 };
00066
00067 #endif
00068
00069
```

5.78 src/Systems/EnemyIASystem.hpp File Reference

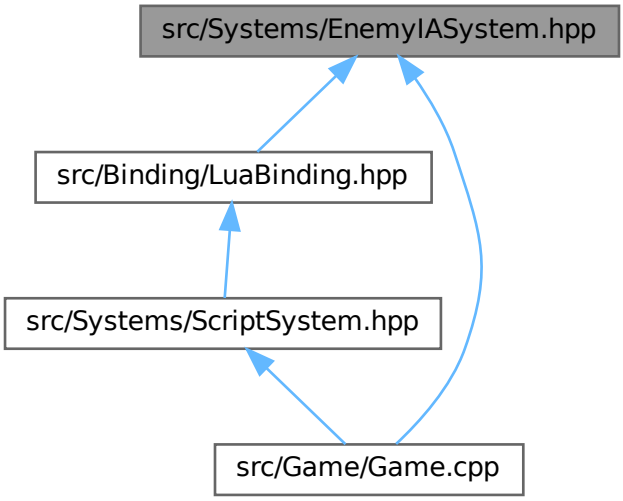
```
#include "../Components/RigidBodyComponent.hpp"
#include "../Components/TransformComponent.hpp"
#include "../Components/DepthComponent.hpp"
#include "../Components/TagEnemyComponent.hpp"
#include "../Components/TagObjectiveComponent.hpp"
#include "../Components/TagPlayerComponent.hpp"
```

```
#include "../ECS/ECS.hpp"
#include <glm/glm.hpp>
```

Include dependency graph for EnemyIASystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class EnemyIASystem

  System responsible for enemy AI logic, specifically searching for closest objectives.

5.79 EnemyIASystem.hpp

Go to the documentation of this file.

```
00001 #ifndef ENEMYIASYSTEM_HPP
00002 #define ENEMYIASYSTEM_HPP
00003
00004 #include "../Components/RigidBodyComponent.hpp"
00005 #include "../Components/TransformComponent.hpp"
00006 #include "../Components/DepthComponent.hpp"
00007 #include "../Components/TagEnemyComponent.hpp"
00008 #include "../Components/TagObjectiveComponent.hpp"
00009 #include "../Components/TagPlayerComponent.hpp"
00010 #include "../ECS/ECS.hpp"
00011 #include <glm/glm.hpp>
00012
00020 class EnemyIASystem : public System{
00021     public:
```

```
00025      EnemyIASystem(){
00026         RequireComponent<TransformComponent>();
00027
00028      }
00029
00037 TransformComponent SearchClosestObjective(Entity enemy, bool isPlayerIncl) {
00038     const auto& enemyTransform = enemy.GetComponent<TransformComponent>();
00039     glm::vec2 enemyPos = enemyTransform.position;
00040
00041     double minDistance = std::numeric_limits<double>::max();
00042     TransformComponent closestTransform;
00043
00044     for (auto entity : GetSystemEntities()) {
00045         if (!entity.HasComponent<TagObjectiveComponent>()) continue;
00046
00047         if (!isPlayerIncl && entity.HasComponent<TagPlayerComponent>()) {
00048             continue;
00049         }
00050         if (isPlayerIncl && entity.HasComponent<TagPlayerComponent>()) {
00051            const auto& targetTransform = entity.GetComponent<TransformComponent>();
00052             closestTransform = targetTransform;
00053             break;
00054         }
00055         const auto& targetTransform = entity.GetComponent<TransformComponent>();
00056         glm::vec2 targetPos = targetTransform.position;
00057
00058         double distance = glm::distance(enemyPos, targetPos);
00059
00060         if (distance < minDistance) {
00061             minDistance = distance;
00062             closestTransform = targetTransform;
00063         }
00064     }
00065
00066     return closestTransform;
00067 }
00068
00076 DepthComponent SearchClosestObjectiveDepth(Entity enemy, bool isPlayerIncl) {
00077     const auto& enemyTransform = enemy.GetComponent<TransformComponent>();
00078     glm::vec2 enemyPos = enemyTransform.position;
00079
00080     double minDistance = std::numeric_limits<double>::max();
00081     DepthComponent closestDepth;
00082
00083     for (auto entity : GetSystemEntities()) {
00084         if (!entity.HasComponent<TagObjectiveComponent>()) continue;
00085
00086         if (!isPlayerIncl && entity.HasComponent<TagPlayerComponent>()) {
00087             continue;
00088         }
00089          if (isPlayerIncl && entity.HasComponent<TagPlayerComponent>()) {
00090             const auto& targetDepth = entity.GetComponent<DepthComponent>();
00091             closestDepth = targetDepth;
00092             break;
00093         }
00094
00095         const auto& targetTransform = entity.GetComponent<TransformComponent>();
00096         const auto& targetDepth = entity.GetComponent<DepthComponent>();
00097         glm::vec2 targetPos = targetTransform.position;
00098
00099         double distance = glm::distance(enemyPos, targetPos);
00100
00101         if (distance < minDistance) {
00102             minDistance = distance;
00103             closestDepth = targetDepth;
00104         }
00105     }
00106
00107     return closestDepth;
00108 }
00109 };
00110
00111 #endif
```

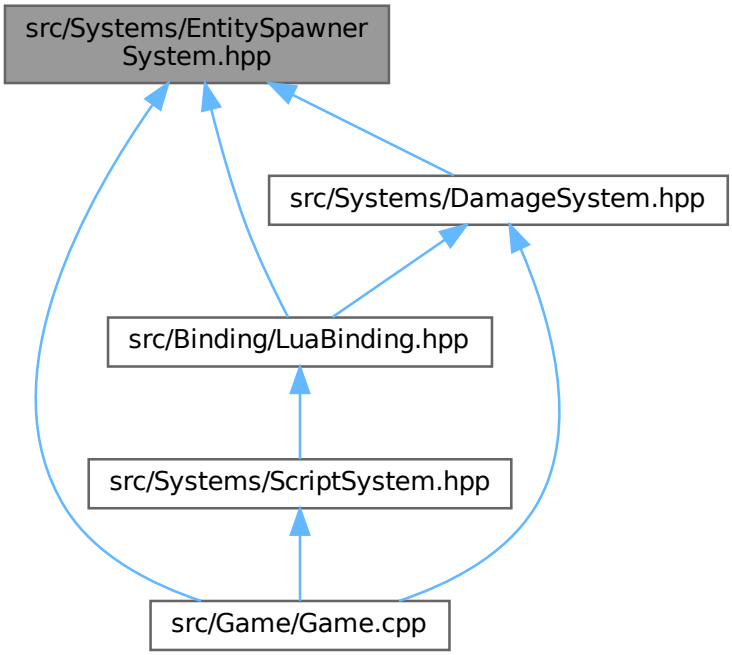
5.80 src/Systems/EntitySpawnerSystem.hpp File Reference

```
#include "../ECS/ECS.hpp"
#include "../Components/CircleColliderComponent.hpp"
#include "../Components/PolygonColliderComponent.hpp"
#include "../Components/DepthComponent.hpp"
```

```
#include "../Components/AnimationComponent.hpp"
#include "../Components/EntitySpawnerComponent.hpp"
#include "../Components/RigidBodyComponent.hpp"
#include "../Components/SpriteComponent.hpp"
#include "../Components/ScriptComponent.hpp"
#include "../Components/TransformComponent.hpp"
#include "../Components/TagWallComponent.hpp"
#include "../Components/TagObjectiveComponent.hpp"
#include "../Components/TagEnemyComponent.hpp"
#include "../Components/TagProjectileComponent.hpp"
#include "../Components/TagPlayerComponent.hpp"
#include "../Components/LifeComponent.hpp"
#include "../Components/DamageComponent.hpp"
#include <sol/sol.hpp>
#include <memory>
```

Include dependency graph for EntitySpawnerSystem.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class EntitySpawnerSystem

  Handles the spawning of entities based on Lua scene configuration.

5.81 EntitySpawnerSystem.hpp

Go to the documentation of this file.

```
00001 #ifndef ENTITYSPAWNERSYSTEM_HPP
00002 #define  ENTITYSPAWNERSYSTEM_HPP
00003
00004 #include "../ECS/ECS.hpp"
00005 #include "../Components/CircleColliderComponent.hpp"
00006 #include "../Components/PolygonColliderComponent.hpp"
00007 #include "../Components/DepthComponent.hpp"
00008 #include "../Components/AnimationComponent.hpp"
00009 #include "../Components/EntitySpawnerComponent.hpp"
00010 #include "../Components/RigidBodyComponent.hpp"
00011 #include "../Components/SpriteComponent.hpp"
00012 #include "../Components/ScriptComponent.hpp"
00013 #include "../Components/TransformComponent.hpp"
00014 #include "../Components/TagWallComponent.hpp"
00015 #include "../Components/TagObjectiveComponent.hpp"
00016 #include "../Components/TagEnemyComponent.hpp"
00017 #include "../Components/TagProjectileComponent.hpp"
00018 #include "../Components/TagPlayerComponent.hpp"
00019 #include "../Components/LifeComponent.hpp"
00020 #include "../Components/DamageComponent.hpp"
00021 #include <sol/sol.hpp>
00022 #include <memory>
00023
00031 class EntitySpawnerSystem : public System{
00032     private:
00033     sol::table entities;
00034     public:
00041      EntitySpawnerSystem(const std::string& scenePath, sol::state& lua){
00042         RequireComponent<TransformComponent>();
00043         RequireComponent<EntitySpawnerComponent>();
00044          sol::load_result script_result = lua.load_file(scenePath);
00045             if(!script_result.valid()){
00046                 sol::error err = script_result;
00047                 std::string errMessage = err.what();
00048                 std::cerr« "[EntitySpawnerSystem]"« errMessage « std::endl;
00049                 return;
00050
00051             }
00052         lua.script_file(scenePath);
00053         this->entities = lua["entities"];
00054
00055      }
00064     Entity GenerateEntity(std::unique_ptr<Registry>& registry,int idEntity,
00065      sol::state& lua){
00066         sol::table entity = entities[idEntity];
00067
00068         Entity newEntity = registry->CreateEntity();
00069
00070         sol::optional<sol::table> hasComponents = entity["components"];
00071         if(hasComponents != sol::nullopt){
00072             sol::table components = entity["components"];
00073
00074             //* AnimationComponent
00075             sol::optional<sol::table>hasAnimation =
00076             components["animation"];
00077             if(hasAnimation != sol::nullopt) {
00078                 newEntity.AddComponent<AnimationComponent>(
00079                  components["animation"]["numFrames"],
00080                  components["animation"]["frameSpeedRate"],
00081                  components["animation"]["isLoop"]
00082                 );
00083             }
00084
00085             //* CircleColiderComponent
00086             sol::optional<sol::table>hasCircleCollider =
00087             components["circle_collider"];
00088             if(hasCircleCollider != sol::nullopt) {
00089                 newEntity.AddComponent<CircleColliderComponent>(
00090                  components["circle_collider"]["radius"],
00091                  components["circle_collider"]["width"],
00092                  components["circle_collider"]["height"]
00093                 );
00094             }
00095
00096             //* RigidbodyComponent
00097             sol::optional<sol::table>hasRigidbody =
00098             components["rigidbody"];
00099             if(hasRigidbody != sol::nullopt) {
00100                 newEntity.AddComponent<RigidBodyComponent>(
00101                  glm::vec2(
00102                  components["rigidbody"]["velocity"]["x"],
00103                  components["rigidbody"]["velocity"]["y"]
```

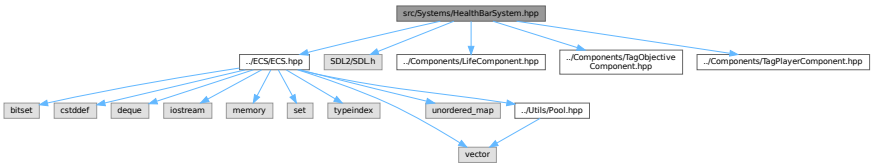
```
00104                 )
00105             );
00106         }
00107 
00108         //* ScriptComponent
00109         sol::optional<sol::table> hasScript = components["script"];
00110         if(hasScript != sol::nullopt){
00111             lua["update"] = sol::nil;
00112             lua["on_click"] = sol::nil;
00113 
00114             std::string path = components["script"]["path"];
00115             lua.script_file(path);
00116 
00117             sol::optional<sol::function> hasOnClick = lua["on_click"];
00118             sol::function onClick = sol::nil;
00119             if(hasOnClick != sol::nullopt){
00120                 onClick = lua["on_click"];
00121             }
00122 
00123             sol::optional<sol::function> hasUpdate = lua["update"];
00124             sol::function update = sol::nil;
00125             if(hasUpdate != sol::nullopt){
00126                 update = lua["update"];
00127             }
00128 
00129             newEntity.AddComponent<ScriptComponent>(update, onClick);
00130         }
00131 
00132         //* Sprite Component
00133         sol::optional<sol::table>hasSprite =
00134         components["sprite"];
00135         if(hasSprite != sol::nullopt) {
00136             newEntity.AddComponent<SpriteComponent>(
00137              components["sprite"]["assetId"],
00138              components["sprite"]["width"],
00139              components["sprite"]["height"],
00140              components["sprite"]["src_rect"]["x"],
00141              components["sprite"]["src_rect"]["y"]
00142             );
00143         }
00144         //* Transform Component
00145         sol::optional<sol::table>hasTransform =
00146         components["transform"];
00147         if(hasTransform != sol::nullopt) {
00148             newEntity.AddComponent<TransformComponent>(
00149             glm::vec2(
00150              components["transform"]["position"]["x"],
00151              components["transform"]["position"]["y"]
00152              ),
00153             glm::vec2(
00154              components["transform"]["scale"]["x"],
00155              components["transform"]["scale"]["y"]
00156              ),
00157              components["transform"]["rotation"]
00158             );
00159         }
00160 
00161          //* Depth Component
00162         sol::optional<sol::table>hasDepth =
00163         components["depth"];
00164         if(hasDepth != sol::nullopt) {
00165             newEntity.AddComponent<DepthComponent>(
00166                 components["depth"]["min_scale"],
00167                 components["depth"]["max_scale"],
00168                 components["depth"]["original_width"],
00169                 components["depth"]["scale_speed"],
00170                 components["depth"]["reference"]
00171             );
00172         }
00173          //* TagPlayerComponent
00174         sol::optional<sol::table>hasTagPlayer =
00175         components["tagplayer"];
00176         if(hasTagPlayer != sol::nullopt) {
00177             newEntity.AddComponent<TagPlayerComponent>();
00178         }
00179 
00180            //* TagEnemyComponent
00181         sol::optional<sol::table>hasTagEnemy =
00182         components["tagenemy"];
00183         if(hasTagEnemy != sol::nullopt) {
00184             newEntity.AddComponent<TagEnemyComponent>();
00185         }
00186          //* TagObjectiveComponent
00187         sol::optional<sol::table>hasTagObjective =
00188         components["tagobjective"];
00189         if(hasTagObjective != sol::nullopt) {
00190             newEntity.AddComponent<TagObjectiveComponent>();
```

```
00191             }
00192             //* TagProjectileComponent
00193             sol::optional<sol::table>hasTagProjectile =
00194             components["tagprojectile"];
00195             if(hasTagProjectile != sol::nullopt) {
00196                 newEntity.AddComponent<TagProjectileComponent>();
00197             }
00198 
00199                 //* LifeComponent
00200             sol::optional<sol::table>hasLife =
00201             components["life"];
00202             if(hasLife != sol::nullopt) {
00203                 int args = components["life"]["life_count"];
00204                 int args2 = components["life"]["life_max"];
00205                 newEntity.AddComponent<LifeComponent>(
00206                     args,args2
00207                 );
00208             }
00209               //* DamageComponent
00210             sol::optional<sol::table>hasDamage =
00211             components["damage"];
00212             if(hasDamage != sol::nullopt) {
00213                 int args = components["damage"]["damage_dealt"];
00214                 newEntity.AddComponent<DamageComponent>(
00215                     args
00216                 );
00217             }
00218 
00219         }
00220         return newEntity;
00221 
00222     }
00223 };
00224 
00225 #endif
```

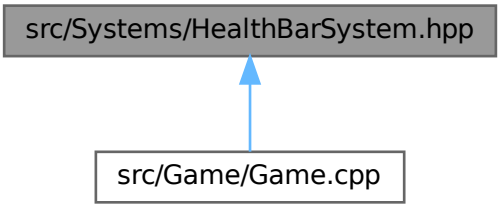
5.82 src/Systems/HealthBarSystem.hpp File Reference

```
#include "../ECS/ECS.hpp"
#include <SDL2/SDL.h>
#include "../Components/LifeComponent.hpp"
#include "../Components/TagObjectiveComponent.hpp"
#include "../Components/TagPlayerComponent.hpp"
```

Include dependency graph for HealthBarSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class HealthBarSystem

  System responsible for rendering health bars for entities with LifeComponent and TagObjectiveComponent.

5.83 HealthBarSystem.hpp

Go to the documentation of this file.

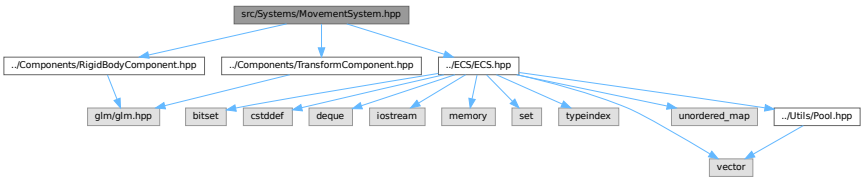
```
00001 #ifndef HEALTHBARSYSTEM_HPP
00002 #define HEALTHBARSYSTEM_HPP
00003
00004 #include "../ECS/ECS.hpp"
00005 #include <SDL2/SDL.h>
00006 #include "../Components/LifeComponent.hpp"
00007 #include "../Components/TagObjectiveComponent.hpp"
00008 #include "../Components/TagPlayerComponent.hpp"
00009
00018 class HealthBarSystem : public System {
00019 public:
00023     HealthBarSystem() {
00024         RequireComponent<LifeComponent>();
00025          RequireComponent<TagObjectiveComponent>();
00026
00027     }
00028
00040     void Update(SDL_Renderer* renderer) {
00041         for (auto entity : GetSystemEntities()) {
00042             const auto& life = entity.GetComponent<LifeComponent>();
00043
00044             SDL_Rect borderRect;
00045             SDL_Rect fillRect;
00046
00047             if (entity.HasComponent<TagPlayerComponent>()) {
00048                 // Jugador
00049                 borderRect = { 100, 50, 300, 20 };
00050                 fillRect = { 100, 50, static_cast<int>(300 * (static_cast<float>(life.life_count) /
      life.life_max)), 20 };
00051
00052                 SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
00053                 SDL_RenderDrawRect(renderer, &borderRect);
00054
00055                 SDL_SetRenderDrawColor(renderer, 0, 160, 0, 255);
00056                 SDL_RenderFillRect(renderer, &fillRect);
00057             }
00058             else {
00059                 // Obelisko
00060                 borderRect = { 100, 80, 300, 20 };
00061                 fillRect = { 100, 80, static_cast<int>(300 * (static_cast<float>(life.life_count) /
      life.life_max)), 20 };
00062
00063                 SDL_SetRenderDrawColor(renderer, 255, 255, 255, 255);
00064                 SDL_RenderDrawRect(renderer, &borderRect);
```

```
00065
00066                 SDL_SetRenderDrawColor(renderer, 0, 90, 180, 255);
00067                 SDL_RenderFillRect(renderer, &fillRect);
00068             }
00069         }
00070     }
00071 };
00072
00073 #endif
```

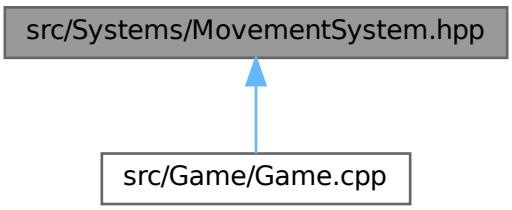
5.84 src/Systems/MovementSystem.hpp File Reference

```
#include "../Components/RigidBodyComponent.hpp"
#include "../Components/TransformComponent.hpp"
#include "../ECS/ECS.hpp"
```

Include dependency graph for MovementSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class MovementSystem

  System responsible for updating entity positions based on their velocity.

5.85 MovementSystem.hpp

Go to the documentation of this file.

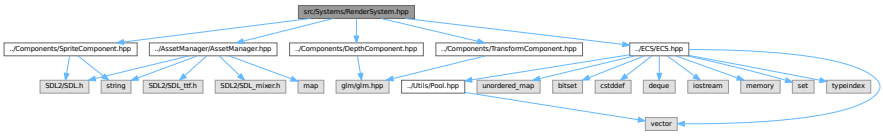
```
00001 #ifndef MOVEMENTSYSTEM_HPP
00002 #define MOVEMENTSYSTEM_HPP
00003
```

```
00004 #include "../Components/RigidBodyComponent.hpp"
00005 #include "../Components/TransformComponent.hpp"
00006 #include "../ECS/ECS.hpp"
00007
00008
00016 class MovementSystem : public System{
00017     public:
00021      MovementSystem(){
00022         RequireComponent<RigidBodyComponent>();
00023         RequireComponent<TransformComponent>();
00024
00025      }
00035     void Update(double dt) {
00036         for(auto entity : GetSystemEntities()){
00037             const auto& rigidbody = entity.GetComponent<RigidBodyComponent>();
00038             auto& transform = entity.GetComponent<TransformComponent>();
00039
00040             transform.position.x += rigidbody.velocity.x * dt;
00041             transform.position.y += rigidbody.velocity.y * dt;
00042         }
00043     }
00044 };
00045
00046 #endif
```

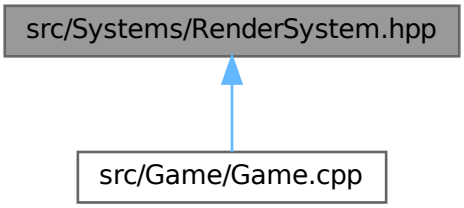
5.86 src/Systems/RenderSystem.hpp File Reference

```
#include "../AssetManager/AssetManager.hpp"
#include "../Components/SpriteComponent.hpp"
#include "../Components/TransformComponent.hpp"
#include "../Components/DepthComponent.hpp"
#include "../ECS/ECS.hpp"
```

Include dependency graph for RenderSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class RenderSystem

  System responsible for rendering entities with sprites on the screen.

5.87 RenderSystem.hpp

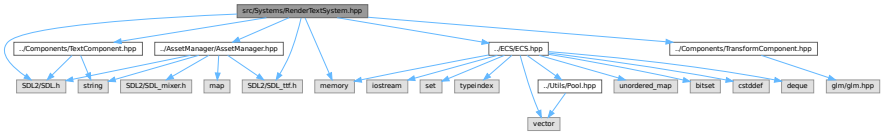
Go to the documentation of this file.

```
00001 #ifndef RENDERSYSTEM_HPP
00002 #define RENDERSYSTEM_HPP
00003
00004 #include "../AssetManager/AssetManager.hpp"
00005 #include "../Components/SpriteComponent.hpp"
00006 #include "../Components/TransformComponent.hpp"
00007 #include "../Components/DepthComponent.hpp"
00008 #include "../ECS/ECS.hpp"
00009
00017 class RenderSystem : public System {
00018   public:
00022    RenderSystem(){
00023     RequireComponent<SpriteComponent>();
00024     RequireComponent<TransformComponent>();
00025    }
00038  void Update(SDL_Renderer* renderer, const std::unique_ptr<AssetManager>& AssetManager) {
00039     std::vector<Entity> sortedEntities;
00040
00041     for (auto entity : GetSystemEntities()) {
00042         sortedEntities.push_back(entity);
00043     }
00044
00045    std::sort(sortedEntities.begin(), sortedEntities.end(), [](const Entity& a, const Entity& b) {
00046     bool aHasDepth = a.HasComponent<DepthComponent>();
00047     bool bHasDepth = b.HasComponent<DepthComponent>();
00048
00049     if (aHasDepth != bHasDepth) {
00050         return !aHasDepth;
00051     }
00052
00053     if (!aHasDepth && !bHasDepth) {
00054         return a.GetId() < b.GetId();
00055     }
00056     // Regla de 3 para escalas comparables
00057     auto aDepth = a.GetComponent<DepthComponent>();
00058     auto aTransform = a.GetComponent<TransformComponent>();
00059     aTransform.scale.y = aTransform.scale.y*10/aDepth.max_scale;
00060
00061     auto bTransform = b.GetComponent<TransformComponent>();
00062     auto bDepth = b.GetComponent<DepthComponent>();
00063     bTransform.scale.y = bTransform.scale.y*10/bDepth.max_scale;
00064
00065     return aTransform.scale.y < bTransform.scale.y;
00066 });
00067
00068     for (auto entity : sortedEntities) {
00069         const auto sprite = entity.GetComponent<SpriteComponent>();
00070         const auto transform = entity.GetComponent<TransformComponent>();
00071
00072         SDL_Rect srcRect = sprite.srcRect;
00073         SDL_Rect dstRect = {
00074             static_cast<int>(transform.position.x),
00075             static_cast<int>(transform.position.y),
00076             static_cast<int>(sprite.width * transform.scale.x),
00077             static_cast<int>(sprite.height * transform.scale.y),
00078         };
00079
00080         SDL_RenderCopyEx(
00081             renderer,
00082             AssetManager->GetTexture(sprite.textureId),
00083             &srcRect,
00084             &dstRect,
00085             transform.rotation,
00086             nullptr,
00087             SDL_FLIP_NONE
00088         );
00089     }
00090  }
00091
00092 };
00093
00094
00095
00096
00097 #endif
```

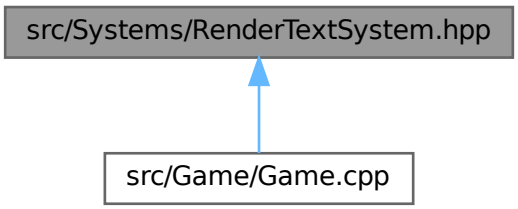
5.88 src/Systems/RenderTextSystem.hpp File Reference

```
#include <SDL2/SDL.h>
#include <SDL2/SDL_ttf.h>
#include <memory>
#include "../AssetManager/AssetManager.hpp"
#include "../Components/TextComponent.hpp"
#include "../Components/TransformComponent.hpp"
#include "../ECS/ECS.hpp"
```

Include dependency graph for RenderTextSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class RenderTextSystem

  System responsible for rendering text components in the ECS.

5.89 RenderTextSystem.hpp

Go to the documentation of this file.

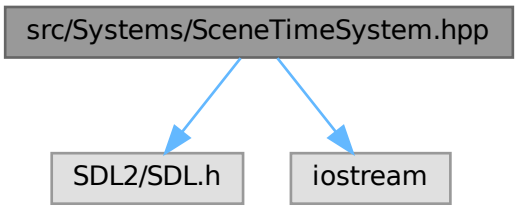
```
00001 #ifndef RENDERTEXTSYSTEM_HPP
00002 #define RENDERTEXTSYSTEM_HPP
00003
00004 #include <SDL2/SDL.h>
00005 #include <SDL2/SDL_ttf.h>
00006
00007 #include <memory>
00008
00009 #include "../AssetManager/AssetManager.hpp"
00010 #include "../Components/TextComponent.hpp"
00011 #include "../Components/TransformComponent.hpp"
00012 #include "../ECS/ECS.hpp"
00013
```

```
00014
00021 class RenderTextSystem : public System {
00022  public:
00026     RenderTextSystem(){
00027         RequireComponent<TextComponent>();
00028         RequireComponent<TransformComponent>();
00029     }
00043     void Update(SDL_Renderer* renderer, const std::unique_ptr<AssetManager>&
00044     assetManager){
00045         for(auto entity : GetSystemEntities()){
00046             auto& text = entity.GetComponent<TextComponent>();
00047             auto& transform = entity.GetComponent<TransformComponent>();
00048
00049             SDL_Surface* surface = TTF_RenderText_Blended(
00050                 assetManager->GetFont(text.fontId), text.text.c_str(), text.color);
00051             text.width = surface->w;
00052             text.height = surface->h;
00053             SDL_Texture* texture = SDL_CreateTextureFromSurface(renderer, surface);
00054             SDL_FreeSurface(surface);
00055
00056             SDL_Rect dstRect = {
00057               static_cast<int>(transform.position.x),
00058               static_cast<int>(transform.position.y),
00059               text.width * static_cast<int>(transform.scale.x),
00060               text.height * static_cast<int>(transform.scale.y)
00061             };
00062
00063             SDL_RenderCopy(renderer, texture, NULL, &dstRect);
00064             SDL_DestroyTexture(texture);
00065         }
00066     }
00081 void RenderFixedText(SDL_Renderer* renderer, TTF_Font* font, const std::string& text, SDL_Color color,
      int x, int y, float scaleX = 1.0f, float scaleY = 1.0f) {
00082
00083     SDL_Surface* surface = TTF_RenderText_Blended(font, text.c_str(), color);
00084     if (!surface) {
00085         SDL_Log("Error creando surface para texto fijo: %s", TTF_GetError());
00086         return;
00087     }
00088     SDL_Texture* texture = SDL_CreateTextureFromSurface(renderer, surface);
00089     SDL_FreeSurface(surface);
00090     if (!texture) {
00091         SDL_Log("Error creando textura para texto fijo: %s", SDL_GetError());
00092         return;
00093     }
00094
00095     SDL_Rect dstRect = {
00096         x,
00097         y,
00098         static_cast<int>(surface->w * scaleX),
00099         static_cast<int>(surface->h * scaleY)
00100     };
00101
00102     SDL_RenderCopy(renderer, texture, nullptr, &dstRect);
00103     SDL_DestroyTexture(texture);
00104 }
00105 };
00106
00107 #endif
```

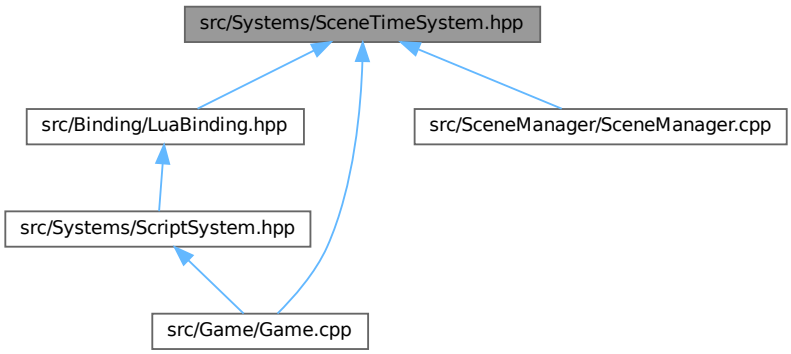
5.90 src/Systems/SceneTimeSystem.hpp File Reference

```
#include <SDL2/SDL.h>
#include <iostream>
```

Include dependency graph for SceneTimeSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class SceneTimeSystem

  Manages scene timing including pause, resume, delta time, and total elapsed time.

5.91 SceneTimeSystem.hpp

Go to the documentation of this file.

```
00001 #ifndef SCENETIMESYSTEM_HPP
00002 #define SCENETIMESYSTEM_HPP
00003
00004 #include <SDL2/SDL.h>
00005 #include <iostream>
00006
00007
00014 class SceneTimeSystem : public System {
00015 private:
00016     int sceneStartTime;
00017     int currentTime;
00018     int deltaTime;
00019     bool paused;
00020     int pauseStartTime;
```

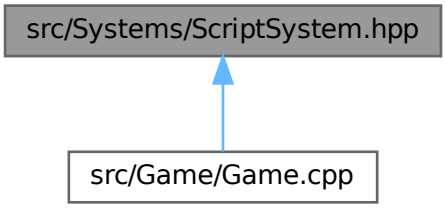
```
00021     int totalPausedTime;
00022 public:
00026     SceneTimeSystem() {
00027         sceneStartTime = SDL_GetTicks();
00028         currentTime = sceneStartTime;
00029         deltaTime = 0;
00030         paused = false;
00031         pauseStartTime = 0;
00032         totalPausedTime = 0;
00033     }
00039     void Pause() {
00040         if (!paused) {
00041             paused = true;
00042             pauseStartTime = SDL_GetTicks();
00043         }
00044     }
00050     void Resume() {
00051         if (paused) {
00052             paused = false;
00053             int pauseDuration = SDL_GetTicks() - pauseStartTime;
00054             totalPausedTime += pauseDuration;
00055             currentTime = SDL_GetTicks();
00056         }
00057     }
00063     void Update() {
00064         if (paused) {
00065             deltaTime = 0;
00066             return;
00067         }
00068
00069         int now = SDL_GetTicks();
00070         deltaTime = now - currentTime;
00071         currentTime = now;
00072     }
00077     int GetSceneTime() const {
00078         return (currentTime - sceneStartTime - totalPausedTime);
00079     }
00084     int GetDeltaTime() const {
00085         return deltaTime;
00086     }
00087
00091     void Reset() {
00092         sceneStartTime = SDL_GetTicks();
00093         currentTime = sceneStartTime;
00094         deltaTime = 0;
00095         paused = false;
00096         pauseStartTime = 0;
00097         totalPausedTime = 0;
00098     }
00099 };
00100
00101 #endif
```

5.92 src/Systems/ScriptSystem.hpp File Reference

```
#include <memory>
#include <sol/sol.hpp>
#include "../Binding/LuaBinding.hpp"
#include "../Components/ScriptComponent.hpp"
#include "../ECS/ECS.hpp"
```

Include dependency graph for ScriptSystem.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class ScriptSystem

  Manages entities with scripts and handles Lua binding and script updates.

5.93 ScriptSystem.hpp

Go to the documentation of this file.

```
00001 #ifndef SCRIPTSYSTEM_HPP
00002 #define SCRIPTSYSTEM_HPP
00003
00004 #include <memory>
00005 #include <sol/sol.hpp>
00006
00007 #include "../Binding/LuaBinding.hpp"
00008 #include "../Components/ScriptComponent.hpp"
00009 #include "../ECS/ECS.hpp"
00010
00018 class ScriptSystem : public System {
00019  public:
00023   ScriptSystem(){
00024     RequireComponent<ScriptComponent>();
00025   }
00030   void CreateLuaBinding(sol::state& lua){
00031     // Classes
00032     lua.new_usertype<Entity>("entity");
00033
00034     // Functions
00035     lua.set_function("is_action_activated", IsActionActivated);
00036     lua.set_function("set_velocity", SetVelocity);
00037     lua.set_function("get_scale", GetScale);
00038     lua.set_function("set_scale", SetScale);
00039     lua.set_function("set_position", SetPosition);
00040     lua.set_function("get_positionX", GetPositionX);
00041     lua.set_function("get_positionY", GetPositionY);
00042     lua.set_function("set_srcRect", SetSrcRect);
00043     lua.set_function("set_numFrames", SetNumFrames);
00044     lua.set_function("go_to_scene", GoToScene);
00045     lua.set_function("create_dynamic_entity",CreateDynamicEntity);
00046     lua.set_function("get_velocity", GetVelocity);
00047     lua.set_function("get_delta_time", GetDeltaTime);
00048     lua.set_function("get_time", GetTime);
00049     lua.set_function("get_defeat", GetDefeat);
00050     lua.set_function("set_timer", SetTimer);
00051     lua.set_function("set_text", SetText);
00052     lua.set_function("search_objectiveX", SearchObjectiveX);
00053     lua.set_function("search_objectiveY", SearchObjectiveY);
00054     lua.set_function("search_objectiveScale", SearchObjectiveScale);
00055     lua.set_function("search_objectiveDepth", SearchObjectiveDepth);
00056     lua.set_function("get_depth", GetDepth);
00057     lua.set_function("kill", Kill);
00058     lua.set_function("destroy_all_enemies", DestroyAllEnemies);
```

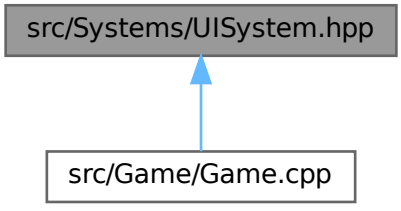
```
00059
00060   }
00067   void Update(sol::state& lua){
00068     for(auto entity : GetSystemEntities()){
00069         const auto& script = entity.GetComponent<ScriptComponent>();
00070
00071         if(script.update != sol::lua_nil){
00072             lua["this"] = entity;
00073             script.update();
00074         }
00075     }
00076   }
00077 };
00078 #endif
```

5.94 src/Systems/UISystem.hpp File Reference

```
#include <SDL2/SDL.h>
#include <memory>
#include <string>
#include <iostream>
#include "../Components/ClickableComponent.hpp"
#include "../Components/ScriptComponent.hpp"
#include "../Components/TextComponent.hpp"
#include "../Components/TransformComponent.hpp"
#include "../ECS/ECS.hpp"
#include "../EventManager/EventManager.hpp"
#include "../Events/ClickEvent.hpp"
```

Include dependency graph for UISystem.hpp:

This graph shows which files directly or indirectly include this file:



Classes

- class UISystem

  Handles UI elements that can be clicked and processes click events.

5.95 UISystem.hpp

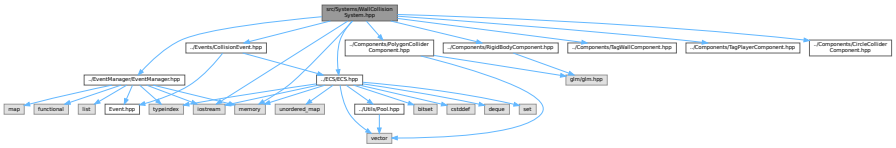
Go to the documentation of this file.

```
00001 #ifndef UISYSTEM_HPP
00002 #define UISYSTEM_HPP
00003
00004 #include <SDL2/SDL.h>
00005
00006 #include <memory>
00007 #include <string>
00008 #include <iostream>
00009
00010 #include "../Components/ClickableComponent.hpp"
00011
00012 #include "../Components/ScriptComponent.hpp"
00013 #include "../Components/TextComponent.hpp"
00014 #include "../Components/TransformComponent.hpp"
00015 #include "../ECS/ECS.hpp"
00016 #include "../EventManager/EventManager.hpp"
00017 #include "../Events/ClickEvent.hpp"
00018
00026 class UISystem : public System {
00027     public:
00031     UISystem(){
00032         RequireComponent<ClickableComponent>();
00033         RequireComponent<TextComponent>();
00034         RequireComponent<TransformComponent>();
00035     }
00040     void SubscribeToClickEvent(std::unique_ptr<EventManager>& eventManager){
00041         eventManager->SubscribeToEvent<ClickEvent, UISystem>(this,
00042         &UISystem::OnClickEvent);
00043     }
00050     void OnClickEvent(ClickEvent& e){
00051         for(auto entity : GetSystemEntities()){
00052             const auto& text = entity.GetComponent<TextComponent>();
00053             const auto& transform = entity.GetComponent<TransformComponent>();
00054
00055             if(transform.position.x < e.posX
00056             && e.posX < transform.position.x + text.width
00057             && transform.position.y < e.posY
00058             && e.posY < transform.position.y + text.height){
00059              if(entity.HasComponent<ScriptComponent>()){
00060                 const auto& script = entity.GetComponent<ScriptComponent>();
00061                 if(script.onClick != sol::nil){
00062                     script.onClick();
00063                 }
00064              }
00065             }
00066         }
00067     }
00068 };
00069
00070 #endif
```

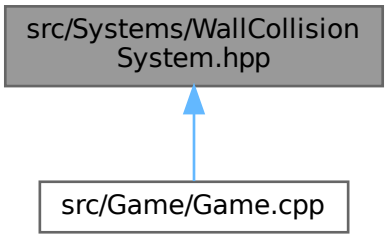
5.96 src/Systems/WallCollisionSystem.hpp File Reference

```
#include <iostream>
#include <memory>
#include "../Components/RigidBodyComponent.hpp"
#include "../Components/TagWallComponent.hpp"
#include "../Components/TagPlayerComponent.hpp"
#include "../Components/PolygonColliderComponent.hpp"
#include "../Components/CircleColliderComponent.hpp"
#include "../ECS/ECS.hpp"
#include "../EventManager/EventManager.hpp"
#include "../Events/CollisionEvent.hpp"
```

Include dependency graph for WallCollisionSystem.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class WallCollisionSystem

  Handles collisions between wall entities and player entities.

5.97 WallCollisionSystem.hpp

Go to the documentation of this file.

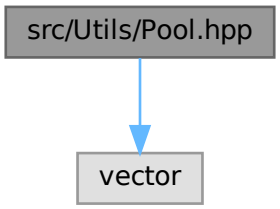
```
00001 #ifndef WALLCOLLISIONSYSTEM_HPP
00002 #define WALLCOLLISIONSYSTEM_HPP
00003
00004 #include <iostream>
00005 #include <memory>
00006
00007 #include "../Components/RigidBodyComponent.hpp"
00008 #include "../Components/TagWallComponent.hpp"
00009 #include "../Components/TagPlayerComponent.hpp"
00010 #include "../Components/PolygonColliderComponent.hpp"
00011 #include "../Components/CircleColliderComponent.hpp"
00012 #include "../ECS/ECS.hpp"
00013 #include "../EventManager/EventManager.hpp"
00014 #include "../Events/CollisionEvent.hpp"
00015
00024 class WallCollisionSystem : public System {
00025 public:
00029     WallCollisionSystem() {
00030         RequireComponent<TagWallComponent>();
00031     }
00036     void SubscribeToCollisionEvent(std::unique_ptr<EventManager>& eventManager) {
00037         eventManager->SubscribeToEvent<CollisionEvent, WallCollisionSystem>(this,
00038             &WallCollisionSystem::OnCollision);
00039     }
00047     void OnCollision(CollisionEvent& e) {
00048         bool aIsWall = e.a.HasComponent<TagWallComponent>();
00049         bool bIsWall = e.b.HasComponent<TagWallComponent>();
```

```
00050         bool aIsPlayer = e.a.HasComponent<TagPlayerComponent>();
00051         bool bIsPlayer = e.b.HasComponent<TagPlayerComponent>();
00052 
00053         if ((aIsWall || bIsWall) && (aIsPlayer||bIsPlayer)) {
00054             Entity move = aIsWall ? e.b : e.a;  // Determinamos cuál es la entidad que se mueve
00055 
00056             if (move.HasComponent<TransformComponent>() && move.HasComponent<RigidBodyComponent>()) {
00057                 auto& transform = move.GetComponent<TransformComponent>();
00058                 auto& circleCollider = move.GetComponent<CircleColliderComponent>();
00059                 auto& rigidBody = move.GetComponent<RigidBodyComponent>();
00060                 float radius = circleCollider.radius;
00061 
00062                 std::cout << "[WallCollisionSystem] Colisión de la entidad " << e.a.GetId()
00063                     << " y " << e.b.GetId() << std::endl;
00064 
00065 
00066                 if (rigidBody.velocity.x < 0) {
00067                     std::cout << "Colisión desde la derecha, moviendo a la derecha." << std::endl;
00068                     transform.position.x = e.a.GetComponent<TransformComponent>().position.x + radius;
00069 
00070                 } else if (rigidBody.velocity.x > 0) {
00071                     transform.position.x = e.a.GetComponent<TransformComponent>().position.x - radius
    ;
00072                 }
00073                 if (rigidBody.velocity.y < 0) {
00074                     transform.position.y = e.a.GetComponent<TransformComponent>().position.y + radius;
00075                 } else if (rigidBody.velocity.y > 0) {
00076                     transform.position.y = e.a.GetComponent<TransformComponent>().position.y - radius;
00077                 }
00078 
00079 
00080                 rigidBody.velocity.x = 0;
00081                 rigidBody.velocity.y = 0;
00082             }
00083         }
00084     }
00085 };
00086 
00087 #endif
```

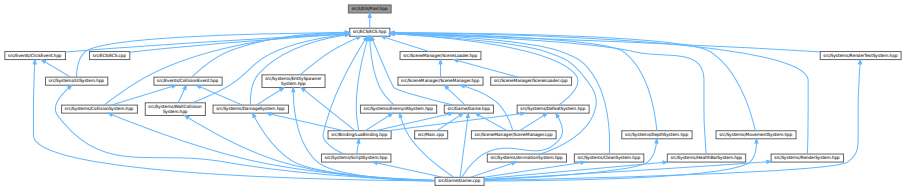
5.98 src/Utils/Pool.hpp File Reference

`#include <vector>`

Include dependency graph for Pool.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class IPool

  Interface for component pools.
- class Pool< TComponent >

  Template class managing a pool of components of type TComponent.

5.99 Pool.hpp

Go to the documentation of this file.

```
00001 #ifndef POOL_HPP
00002 #define POOL_HPP
00003
00004 #include <vector>
00005
00009 class IPool {
00010 public:
00014     virtual ~IPool() = default;
00015 };
00016
00022 template <typename TComponent>
00023 class Pool : public IPool {
00024 private:
00026     std::vector<TComponent> data;
00027
00028 public:
00034     Pool(int size = 1000) {
00035         data.resize(size);
00036     }
00037
00041     virtual ~Pool() = default;
00042
00048     bool IsEmpty() const {
00049         return data.empty();
00050     }
00051
00057     int GetSize() const {
00058         return static_cast<int>(data.size());
00059     }
00060
00066     void Resize(int n) {
00067         data.resize(n);
00068     }
00069
00073     void Clear() {
00074         data.clear();
00075     }
00076
00082     void Add(TComponent object) {
00083         data.push_back(object);
00084     }
00085
00092     void Set(int index, TComponent object) {
00093         data[index] = object;
00094     }
00095
00102     TComponent& Get(unsigned int index) {
00103         return static_cast<TComponent&>(data[index]);
00104     }
00105
```

```
00112     TComponent& operator[](unsigned int index) {
00113         return static_cast<TComponent&>(data[index]);
00114     }
00115 };
00116
00117 #endif
```

Index