ROME Game Engine Documentation

S192255

Updated as of Friday May 3 2019

# Table of Contents

[Hierarchical Index 4](#_Toc7753594)

[Class Hierarchy 4](#_Toc7753595)

[Class Index 5](#_Toc7753596)

[Class List 5](#_Toc7753597)

[Class Documentation 6](#_Toc7753598)

[App Class Reference 6](#_Toc7753599)

[Public Member Functions 6](#_Toc7753601)

[Constructor & Destructor Documentation 6](#_Toc7753602)

[Ball Class Reference 7](#_Toc7753603)

[Public Member Functions 7](#_Toc7753605)

[ContactListener Class Reference 8](#_Toc7753606)

[Public Member Functions 8](#_Toc7753608)

[EventManager Class Reference 9](#_Toc7753609)

[Public Member Functions 9](#_Toc7753611)

[Static Public Member Functions 9](#_Toc7753612)

[Public Attributes 9](#_Toc7753613)

[Game Class Reference 10](#_Toc7753614)

[Public Member Functions 10](#_Toc7753616)

[Public Attributes 10](#_Toc7753617)

[GameObject Class Reference 11](#_Toc7753618)

[Public Member Functions 11](#_Toc7753620)

[Public Attributes 11](#_Toc7753621)

[InputManagement Class Reference 13](#_Toc7753622)

[Public Types 13](#_Toc7753624)

[Public Member Functions 13](#_Toc7753625)

[Public Attributes 13](#_Toc7753626)

[Platform Class Reference 14](#_Toc7753627)

[Public Member Functions 14](#_Toc7753629)

[Additional Inherited Members 14](#_Toc7753630)

[Player Class Reference 15](#_Toc7753631)

[Public Types 15](#_Toc7753633)

[Public Member Functions 15](#_Toc7753634)

[Public Attributes 15](#_Toc7753635)

[ResourceManager Class Reference 16](#_Toc7753636)

[Public Member Functions 16](#_Toc7753638)

[Static Public Member Functions 16](#_Toc7753639)

[Public Attributes 16](#_Toc7753640)

[SceneManager Class Reference 17](#_Toc7753641)

[Classes 17](#_Toc7753643)

[Public Member Functions 17](#_Toc7753644)

[Public Attributes 17](#_Toc7753645)

[SceneManager::GameObjectDef Struct Reference 18](#_Toc7753646)

[Public Attributes 18](#_Toc7753648)

[TimeHandler Class Reference 19](#_Toc7753649)

[Public Member Functions 19](#_Toc7753651)

[Static Public Member Functions 19](#_Toc7753652)

[Public Attributes 19](#_Toc7753653)

[Window Class Reference 20](#_Toc7753654)

[Public Member Functions 20](#_Toc7753656)

[Public Attributes 20](#_Toc7753657)

[References/Bibliography 21](#_Toc7753658)

# Hierarchical Index

## Class Hierarchy

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

App 6

b2ContactListener

ContactListener 8

EventManager 9

Game 10

InputManagement 13

ResourceManager 16

SceneManager 17

SceneManager::GameObjectDef 18

TimeHandler 19

Transformable

GameObject 11

Ball 7

Platform 14

Player 15

Window 20

# Class Index

## Class List

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

**App (Overarching container class)** 6

**Ball (a moveable ball object )** 7

**ContactListener (handles the Physics of collision between GameObjects )** 8

**EventManager (Polls and stores events from the SFML Game Window )** 9

**Game (Stores and runs all of the gameObjects in the world/scene )** 10

**GameObject (Base GameObject class definition)** 11

**InputManagement (Handles the keyboard events)** 13

**Platform (a static collidable platform Object )** 14

**Player (Controllable playable entity GameObject )** 15

**ResourceManager (Handles the loading and storing of texture resources)** 16

**SceneManager (Parser for a JSON text file)** 17

**SceneManager::GameObjectDef ( Definition for GameObjects in the Level JSON)** 18

**TimeHandler (responsible for maintaining a global clock )** 19

**Window (defines and handles the drawing and processing of sprites)** 20

# Class Documentation

## App Class Reference

### Detailed Description

an overarching container class to hold different parts of the Engine/Managers and the **Game** in tandem

#include <App.h>

### Public Member Functions

**App** (std::string winName)

*Construct the* ***App*** *given a name that will be used throughout.*

**~App** ()

***App*** *deconstructor, delete any pointer references to clean memory.*

void **Update** ()

*Update function for the* ***App*** *that will call the respective Update functions for each component.*

**Window** \* **getWindow** ()

*Return a pointer to the SFML* ***Window****.*

**Game** \* **getGame** ()

*Return a pointer to the running* ***Game****.*

**EventManager** \* **getEventManager** ()

*Return a pointer to Event Manager for polling.*

float **getDeltaTime** ()

*Return Delta-Time as calculated in the Time Handler.*

### Constructor & Destructor Documentation

#### App::App (std::string *winName*)

Construct the **App** given a name that will be used throughout.

#### App::~App ()

**App** deconstructor, delete any pointer references to clean memory.

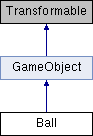
## Ball Class Reference

### Detailed Description

Derived from a base **GameObject**, create a moveable ball object.

#include <Ball.h>

Inheritance diagram for Ball:



### Public Member Functions

**Ball** (int posX, int posY, float rot)

*Construct a ball given the position and rotation.*

**~Ball** ()

*Deconstructor.*

void **GameObject::Load** (b2World \*World)

*Load the* ***GameObject*** *into the world.*

void **GameObject::Update** ()

*Update the* ***GameObject*** *of all functionality.*

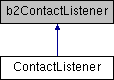
## ContactListener Class Reference

### Detailed Description

Derived from Box2D **ContactListener** - handles the Physics of collision between GameObjects.

#include <ContactListener.h>

Inheritance diagram for ContactListener:



### Public Member Functions

void **BeginContact** (b2Contact \*contact)

*Callback when Objects begin touching.*

void **EndContact** (b2Contact \*contact)

*Callback when Objects stop colliding.*

void **PostSolve** (b2Contact \*contact, const b2ContactImpulse \*impulse)

*Post solving of Object Collision.*

void **PreSolve** (b2Contact \*contact, const b2Manifold \*oldManifold)

*Pre solving of Object Collision.*

## EventManager Class Reference

### Detailed Description

polls and stores events from the SFML **Game** **Window**

#include <EventManager.h>

### Public Member Functions

**EventManager** ()

*Constructor.*

**~EventManager** ()

*Deconstructor.*

void **pollEvents** ()

*called every frame to poll the window for events*

void **storeEvent** (sf::Event currentEvent)

*store an event into the list of events for the input manager to handle*

void **setWindowPtr** (**Window** \*window)

*set the pointer for the window the game is running in*

### Static Public Member Functions

static **EventManager** \* **instance** ()

### Public Attributes

std::vector< sf::Event > **events**

*the list of events that have been polled*

## Game Class Reference

### Detailed Description

stores and runs the gameObjects in the world/scene

#include <Game.h>

### Public Member Functions

**Game** (sf::RenderWindow \*window, **TimeHandler** \*time)

*Construct a* ***Game*** *passing the* ***Window*** *and Time Manager.*

**~Game** ()

*Deconstructor.*

void **start** ()

*Start of the game, define the world and the assets needed.*

void **update** ()

*Update all gameObjects within the world.*

void **loadScene** (std::string levelFileDir)

*calling the scene Manager to load a scene from a given file*

void **givePlayerInput** (**EventManager** \*eventManager)

### Public Attributes

**Player** \* **player**

*Store a reference the player controller* ***GameObject****.*

b2World \* **world**

*Store a reference of the Physics based world.*

sf::RenderWindow \* **gameWindow**

*Store a reference of the SFML* ***Window****.*

**ResourceManager** \* **m\_resMan**

*Store a reference to the Resource Manager.*

**SceneManager** \* **m\_sceneMan**

*Store a reference to the Scene Manager.*

**TimeHandler** \* **m\_timeHandler**

*Store a reference to the Time Handler.*

std::vector< **GameObject** \* > **gameObjectList** = std::vector<**GameObject**\*>()

*Stores a list of all the GameObjects within the game world.*

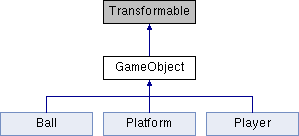
## GameObject Class Reference

### Detailed Description

Base **GameObject** class definition derived from a SFML Transformable.

#include <GameObject.h>

Inheritance diagram for GameObject:



### Public Member Functions

virtual void **Load** (b2World \*World)=0

*Virtual load function to be declared by derived Objects called on creation.*

virtual void **Update** ()=0

*Virtual Update function to be declared by derived Objects called every tick.*

void **Destroy** ()

*Remove the* ***GameObject*** *in it's entirety from the GameWorld.*

void **SetSprite** ()

*Sets the texture of the sprite.*

void **givePhysicsBody** (b2World \*World, b2BodyType physType)

*Make the* ***GameObject*** *a physical interactable Object.*

void **gravity** ()

*Apply gravity where applicable.*

void **ApplyVelocity** ()

*Move the* ***GameObject****.*

void **UpdatePhysics** ()

*Update all of the Physics of the* ***GameObject*** *each frame.*

### Public Attributes

b2Vec2 **initalPos**

*Define the* ***GameObject*** *initial position on load.*

float **initalRot**

*Define the* ***GameObject*** *initial rotation on load.*

b2Vec2 **velocity**

*Storing of velocity for Physics calculations.*

b2Body \* **body**

*Storing it's physical collide-able body/mass.*

sf::Texture **texture**

*Store the material for the Object.*

sf::Sprite **sprite**

*Store the sprite for the Object.*

## InputManagement Class Reference

### Detailed Description

Handles the keyboard events polled from the Event Manager.

#include <InputManagement.h>

### Public Types

enum **playerActions** { **moveRight**, **moveLeft**, **jump**, **stop**, **no** }

*Defining the list of actions that the* ***Player*** *can do*

### Public Member Functions

**InputManagement** ()

*Constructor function.*

**~InputManagement** ()

*Deconstruct.*

void **Listen** ()

*await for events to be polled by the Event Manager and handle them as such*

### Public Attributes

**EventManager** \* **eventHandler**

*Store a reference to the Event Management System.*

sf::Event **currentEvent**

*Temporary storage of the current event being handled.*

**playerActions** **currentAction**

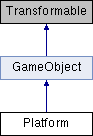
## Platform Class Reference

### Detailed Description

Derived from a base **GameObject**, create a static collide-able platform Object.

#include <Platform.h>

Inheritance diagram for Platform:



### Public Member Functions

**Platform** (int posX, int posY, float rot)

*the constructor definition for the platform*

**~Platform** ()

*Deconstructor.*

void **GameObject::Load** (b2World \*World)

*Load the* ***GameObject*** *into the world.*

void **GameObject::Update** ()

*Update the* ***GameObject*** *of all functionality.*

### Additional Inherited Members

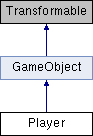
## Player Class Reference

### Detailed Description

Controllable playable entity **GameObject**.

#include <Player.h>

Inheritance diagram for Player:



### Public Types

enum **movementDirections** { **Left**, **Right**, **Jump**, **Stop** }

### Public Member Functions

**Player** (int posX, int posY, float rot)

*Constructor taking in positional/rotational arguments.*

**~Player** ()

*Deconstructor.*

void **GameObject::Load** (b2World \*World)

*Load the* ***GameObject*** *into the world.*

void **GameObject::Update** ()

*Update the* ***GameObject*** *of all functionality.*

void **move** (movementDirections moveType)

*move the entity based on direction using linear velocity*

### Public Attributes

b2Vec2 **speed** = b2Vec2(50, 0)

*the base speed applied when moving*

movementDirections **playerMovement**

*stores the current action the* ***Player*** *is doing based on keyboard input*

**InputManagement** \* **inputComponent**

*store a reference to the Input Management Component*

## ResourceManager Class Reference

### Detailed Description

handles the loading and storing of texture resources on the disk

#include <ResourceManager.h>

### Public Member Functions

**ResourceManager** ()

*Constructor function.*

**~ResourceManager** ()

*Deconstructor.*

sf::Texture **LoadTexture** (std::string fileDir, std::string nameToStore)

*Load the texture from file giving it an appropriate name to store in the Texture Map.*

sf::Texture **GetTextureFromMap** (std::string textureRef)

*Return a texture from the Texture Map given its key index.*

void **storeInMap** (std::string textureName, sf::Texture texture)

*Stores the loaded Texture into the Texture map with its key pairing.*

### Static Public Member Functions

static **ResourceManager** \* **instance** ()

### Public Attributes

std::map< std::string, sf::Texture > **textureMap**

*A map storing all of the loaded textures.*

## SceneManager Class Reference

### Detailed Description

a parser for a JSON text file and generates the GameObjects for the level

#include <SceneManager.h>

### Classes

struct **GameObjectDef**

*< Definition for how GameObjects in the Level JSON should be constructed*

### Public Member Functions

**SceneManager** ()

***SceneManager*** *constructor.*

**~SceneManager** ()

***SceneManager*** *deconstructor.*

void **parseSceneFromFile** (std::string levelFileDir)

*Read a provided JSON level file and parse the data.*

### Public Attributes

std::vector< **GameObjectDef** \* > **GameObjects**

*Store a list of generated* ***GameObject*** *definitions from the parsed JSON.*

## SceneManager::GameObjectDef Struct Reference

### Detailed Description

Definition for how GameObjects in the Level JSON should be constructed

#include <SceneManager.h>

### Public Attributes

std::string **type**

sf::Vector2f **position**

float **rotation**

std::string **spriteName**

## TimeHandler Class Reference

### Detailed Description

part of the **App** responsible for maintaining a global clock

#include <TimeHandler.h>

### Public Member Functions

**TimeHandler** ()

*Construct the Time Handler.*

**~TimeHandler** ()

*Deconstruction function.*

void **Update** ()

*Update and calculate DeltaTime.*

### Static Public Member Functions

static **TimeHandler** \* **instance** ()

### Public Attributes

float **deltaTime**

*Store the Delta-Time value between frames.*

sf::Clock **clock** = sf::Clock()

*Define the global clock to run.*

## Window Class Reference

### Detailed Description

**Window** class defines and handles the drawing and processing of sprites and the game world.

#include <Window.h>

### Public Member Functions

**Window** (std::string name)

*Constructor taking in name for the* ***Window****.*

**~Window** ()

*Deconstructor.*

void **CreateWindow** ()

*Create a new Render* ***Window*** *given the parameters.*

int **getWidth** ()

*return the width of the window*

int **getHeight** ()

*return the height of the window*

**Window** \* **instance** ()

*return the instance of the window*

### Public Attributes

sf::RenderWindow \* **GameWindow**

*Store a reference to the SFML Render* ***Window****.*

## References/Bibliography

Unity. (2019). *Unity - Scripting API:*. [online] Docs.unity3d.com. Available at: https://docs.unity3d.com/ScriptReference/index.html [Accessed 3 May 2019].

Docs.unrealengine.com. (2019). *Unreal Engine 4 Documentation*. [online] Available at: https://docs.unrealengine.com/en-us/ [Accessed 3 May 2019].