**Game Engine & Tools Documentation**

Stefan Simtea

Contents

[Class list 4](#_Toc110565692)

[Classes: 4](#_Toc110565693)

[Structs: 5](#_Toc110565694)

[File list 5](#_Toc110565695)

[**Class Documentation** 5](#_Toc110565696)

[bitmap – Class reference 5](#_Toc110565697)

[**Description** 5](#_Toc110565698)

[**Reference** 5](#_Toc110565699)

[GameObject – Class reference 8](#_Toc110565700)

[**Description**: 8](#_Toc110565701)

[**Reference**: 9](#_Toc110565702)

[Sprite – Class reference 15](#_Toc110565703)

[**Public Member Functions** 16](#_Toc110565704)

[**Public Attributes** 16](#_Toc110565705)

[**Constructor & Destructor Documentation** 16](#_Toc110565706)

[**Member Function Documentation** 17](#_Toc110565707)

[**Member Data Documentation** 19](#_Toc110565708)

[input – Class reference 19](#_Toc110565709)

[**Public Member Functions** 19](#_Toc110565710)

[**Constructor & Destructor Documentation** 19](#_Toc110565711)

[**Member Function Documentation** 20](#_Toc110565712)

[ResourceManager – Class reference 20](#_Toc110565713)

[**Public Member Functions** 20](#_Toc110565714)

[**Static Public Member Functions** 20](#_Toc110565715)

[**Public Attributes** 20](#_Toc110565716)

[**Constructor & Destructor Documentation** 20](#_Toc110565717)

[**Member Function Documentation** 20](#_Toc110565718)

[**Member Data Documentation** 21](#_Toc110565719)

[Scene – Class reference 21](#_Toc110565720)

[**Public Member Functions** 21](#_Toc110565721)

[**Public Attributes** 21](#_Toc110565722)

[**Constructor & Destructor Documentation** 21](#_Toc110565723)

[**Member Function Documentation** 22](#_Toc110565724)

[**Member Data Documentation** 22](#_Toc110565725)

[SceneManager – Class reference 23](#_Toc110565726)

[**Public Member Functions** 23](#_Toc110565727)

[**Public Attributes** 23](#_Toc110565728)

[**Constructor & Destructor Documentation** 23](#_Toc110565729)

[**Member Function Documentation** 23](#_Toc110565730)

[**Member Data Documentation** 24](#_Toc110565731)

[**File Documentation** 24](#_Toc110565732)

[main.cpp 24](#_Toc110565733)

[Code: 24](#_Toc110565734)

[bitmap.h 25](#_Toc110565735)

[Classes: 25](#_Toc110565736)

[Code: 25](#_Toc110565737)

[bitmap.cpp 26](#_Toc110565738)

[Code: 26](#_Toc110565739)

[Debug.h 27](#_Toc110565740)

[Classes: 27](#_Toc110565741)

[Frametime Struct Reference: 27](#_Toc110565742)

[Code: 27](#_Toc110565743)

[GameObject.h 28](#_Toc110565744)

[Classes 28](#_Toc110565745)

[Enumerations 28](#_Toc110565746)

[Enumeration Type Documentation 28](#_Toc110565747)

[coords Struct Reference 28](#_Toc110565748)

[Code: 29](#_Toc110565749)

[GameObject.cpp 30](#_Toc110565750)

[Code: 30](#_Toc110565751)

[input.h 35](#_Toc110565752)

[Classes: 35](#_Toc110565753)

[Enumerations: 35](#_Toc110565754)

[Enumeration Type Documentation 35](#_Toc110565755)

[Code: 35](#_Toc110565756)

[input.cpp 36](#_Toc110565757)

[Code: 36](#_Toc110565758)

[ResourceManager.h 38](#_Toc110565759)

[Classes: 39](#_Toc110565760)

[Code: 39](#_Toc110565761)

[ResourceManager.cpp 39](#_Toc110565762)

[Code: 39](#_Toc110565763)

[Scene.h 41](#_Toc110565764)

[Classes: 41](#_Toc110565765)

[Code: 41](#_Toc110565766)

[Scene.cpp 42](#_Toc110565767)

[Code: 42](#_Toc110565768)

[SceneManager.h 44](#_Toc110565769)

[Classes: 44](#_Toc110565770)

[Code: 44](#_Toc110565771)

[SceneManager.cpp 45](#_Toc110565772)

[Code: 45](#_Toc110565773)

[Sprite.h 47](#_Toc110565774)

[Classes: 47](#_Toc110565775)

[Code: 47](#_Toc110565776)

[Sprite.cpp 48](#_Toc110565777)

[Code: 48](#_Toc110565778)

[**Tools Programming** 49](#_Toc110565779)

[In GameObject class: 49](#_Toc110565780)

[Code: 50](#_Toc110565781)

[In bitmap class: 50](#_Toc110565782)

[Code: 50](#_Toc110565783)

[I\_GUI class: 50](#_Toc110565784)

[I\_GUI Class Reference 50](#_Toc110565785)

[File Reference 51](#_Toc110565786)

[In Scene class: 52](#_Toc110565787)

[Scene.h 52](#_Toc110565788)

[Scene.cpp 52](#_Toc110565789)

[**Additional Information** 54](#_Toc110565790)

[Game scene 55](#_Toc110565791)

[Editor scene 56](#_Toc110565792)

[**References** 56](#_Toc110565793)

# Class list

List of classes and structs with brief descriptions:

## Classes:

* bitmap – Class that handles bitmaps
* GameObject – Class that handles
* Sprite – Class that handles sprites which are bitmaps with more functionality
* input – Class that handles user input
* ResourceManager – Class that helps load images into the engine
* Scene – Class that represents the actual game scene
* SceneManager – Class that manages scenes (loads, unloads, saves)

## Structs:

* BitmapData – Found in ResourceManager.h; Used to easily access a bitmap’s texture and surface data
* coords – Found in GameObject.h; Used to represent a game object’s x and y coordinates
* Frametime – Found in Debug.h; Allows storing of frametimes

# File list

* **Game Engine/Tutorial 7/bitmap.cpp**
* **Game Engine/Tutorial 7/bitmap.h**
* **Game Engine/Tutorial 7/Debug.h**
* **Game Engine/Tutorial 7/GameObject.cpp**
* **Game Engine/Tutorial 7/GameObject.h**
* **Game Engine/Tutorial 7/input.cpp**
* **Game Engine/Tutorial 7/input.h**
* **Game Engine/Tutorial 7/main.cpp**
* **Game Engine/Tutorial 7/ResourceManager.cpp**
* **Game Engine/Tutorial 7/ResourceManager.h**
* **Game Engine/Tutorial 7/Scene.cpp**
* **Game Engine/Tutorial 7/Scene.h**
* **Game Engine/Tutorial 7/SceneManager.cpp**
* **Game Engine/Tutorial 7/SceneManager.h**
* **Game Engine/Tutorial 7/Sprite.cpp**
* **Game Engine/Tutorial 7/Sprite.h**

# **Class Documentation**

## bitmap – Class reference

### **Description**

Class that is responsible for allowing the user to create bitmap type objects that take in an SDL\_Renderer, a filename, an xpos that represents the texture’s position on the x axis, an ypos that represents the texture’s position on the y axis and a boolean useTransparency that determines if the texture is going to contain transparent parts or not.

The bitmap class is used as a class to hold textures and nothing more as the sprite class handles other attributes and functions such as moving the texture around the scene.

### **Reference**

#include <bitmap.h>

#### **Public Member Functions**

* **Bitmap** (SDL\_Renderer \*renderer, std::string fileName, int xpos, int ypos, bool useTransparency=false)

*Constructor. Creates the bitmap's surface and texture.*

* ~Bitmap ()

*Frees the memory allocated to the bitmap texture and surface.*

* SDL\_Texture \* **getTexture** ()

*Gets texture file of the object that called the function.*

* string **getFile** ()

*Returns the image's path used for the bitmap.*

* void **setX** (int x)

*Set's the x axis value of the bitmap.*

* void **setY** (int y)

*Set's the y axis value of the bitmap.*

* int **getX** ()

*Returns the x axis position of the bitmap.*

* int **getY** ()

*Returns the y axis position of the bitmap.*

* void **draw** ()

*Renders the bitmap at x,y coords of the bitmap object.*

#### **Constructor & Destructor Documentation**

##### Bitmap::Bitmap (SDL\_Renderer \* *renderer*, std::string *fileName*, int *xpos*, int *ypos*, bool *useTransparency* = false)

Constructor. Creates the bitmap's surface and texture.

###### Parameters

|  |  |
| --- | --- |
| *renderer* | - Renderer that will be responsible for drawing our bitmaps |
| *fileName* | - Name of file that we will load to use as our bitmap |
| *xpos* | - x Position in the scene |
| *ypos* | - y Position in the scene |
| *useTransparency* | - Boolean that checks if the bitmap is using transparency. Default - false |

##### Bitmap::~Bitmap ()

Frees the memory allocated to the bitmap texture and surface.

#### **Member Function Documentation**

##### *void Bitmap::draw ()*

Renders the bitmap at x,y coords of the bitmap object.

###### Usage example:

bitmap->draw()

Will draw the bitmap on the object’s coordinates and will tell the renderer to render it.

##### *string Bitmap::getFile ()*

Returns the image's path used for the bitmap.

###### Usage example:

cout<< bitmap->getFile();

Will print:

/assets/asset.png

###### Returns

##### *SDL\_Texture \* Bitmap::getTexture ()*

Gets texture file of the object that called the function.

###### Returns

- Returns an SDL\_Texture type pointer of the object's texture

##### *int Bitmap::getX ()*

Returns the x axis position of the bitmap.

###### Returns

- Returns the x axis of the bitmap

###### Usage example:

Bitmap\* bitmap;

bitmap->getX(10)

*Returns the position on the x axis of the bitmap*

##### *int Bitmap::getY ()*

Returns the y axis position of the bitmap.

###### Returns

- Returns the y axis of the bitmap

###### Usage example:

Bitmap\* bitmap;

bitmap->getY(10)

*Returns the position on the y axis of the bitmap*

##### *void Bitmap::setX (int x)*

Set's the x axis value of the bitmap.

###### Parameters

|  |  |
| --- | --- |
| *x* | - x axis value |

###### Usage example:

Bitmap\* bitmap;

bitmap->setX(10)

*bitmap’s value on the x axis will now be 10.*

##### *void Bitmap::setY (int y)*

Set's the y axis value of the bitmap.

###### Parameters

|  |  |
| --- | --- |
| *y* | - y axis value |

###### Usage example:

Bitmap\* bitmap;

bitmap->setY(10)

*bitmap’s value on the y axis will now be 10.*

## GameObject – Class reference

#include <GameObject.h>

### **Description**:

Class that unites the various type of objects that can be present in a scene under one type.

The class can take either a Sprite, Text or Trigger type object and is responsible for initialising, rendering, destroying and manipulating these objects using its member function. The class has different initialisation methods for each type of object as each of them require different variables for the initialisation to happen.

### **Reference**:

#### **Public Member Functions**

* **GameObject** (SDL\_Renderer \*m\_game, **Type** objType, string objName)

*Constructor. Creates a* ***GameObject*** *object that takes in an SDL Renderer, a type and a name*

* void **addObject** (std::vector< **GameObject** \* > &Objects)

*Adds the game object into the vector of game objects*

* void **removeObject** (std::vector< **GameObject** \* > &Objects)

*Removes the game object from a vector of game objects*

* void **initSprite** (string path, int x, int y, bool transparent, int **layer**)

*Initializez a sprite object.*

* void **initTrigger** (int x, int y)

*Initializez a trigger object.*

* void **initText** (const char \*)

*Initializez the text with a given font.*

* void **setCoords** (int x, int y)

*Sets the value of x and y of the game object.*

* void **offsetPosition** (int x, int y)

*Offset the position by an ammount*

* void **autoMoveYAxis** (bool dir)

*Automatically moves an object on the Y Axis on one direction until "dir" tells it to go the other direction.*

* bool **checkCollision** (**GameObject** \*object)

*Checks collision between object that called the function and another object. Works for SPRITE and TRIGGER type objects. TRIGGER objects don't have a sprite so their sprite values will be set to 0.*

* void **UpdateText** (string msg, int x, int y, int fontSize, SDL\_Color colour)

*Updates text on screen.*

* void **render** ()

*Calls according function based on the object's type. For SPRITE type objects calls draw() For TRIGGER type objects prints created location. For TEXT type objects doesn't do anything. Call* ***UpdateText()*** *instead.*

* void **destroy** ()

*Function that acts like a destructor. Checks the type of* ***GameObject*** *it is called from and calls their corresponding functions that destroys them*

* void **handleInput** (**Input** \*m\_input)

*Used to dictate how much the player's position will offset when pressing a specified key.*

* void **updateInput** ()

*Not used. For debugging.*

* string **getPathToTexture** ()

*Returns path to the image used for the sprite.*

#### **Public Attributes**

* **Type** **type**
* **coords** **objCoords**
* int **layer**

#### **Constructor & Destructor Documentation**

##### GameObject::GameObject (SDL\_Renderer \* *m\_game*, Type *objType*, string *objName*)

Constructor. Creates a **GameObject** object that takes in an SDL Renderer, a type and a name

##### Parameters

|  |  |
| --- | --- |
| *m\_game* | - Renderer that will render the object |
| *objType* | - Type that can be either **Sprite**, Trigger or Text |
| *objName* | - Name of the created object for reference |

#### **Member Function Documentation**

##### void GameObject::addObject (std::vector< GameObject \* > & *Objects*)

Adds the game object into the vector of game objects

###### Parameters

|  |  |
| --- | --- |
| *Objects* | - Vector of game objects to take in the new object |

###### Usage:

GameObject\* object;

vector<GameObject\*> objects;

object->addObject(objects)

Adds the GameObject “object” to the vector “objects”

##### void GameObject::autoMoveYAxis (bool *dir*)

Automatically moves an object on the Y Axis on one direction until "dir" tells it to go the other direction.

###### Parameters

|  |  |
| --- | --- |
| *dir* | - Dictates the direction the object will head: up or down |

##### bool GameObject::checkCollision (GameObject \* *object*)

Checks collision between object that called the function and another object. Works for SPRITE and TRIGGER type objects. TRIGGER objects don't have a sprite so their sprite values will be set to 0.

###### Parameters

|  |  |
| --- | --- |
| *object* | - Object that collides with object that called checkCollision |

###### Returns

* true if collision between objects occurred
* false otherwise

###### Usage:

GameObject\* player;

GameObject\* enemy;

player->checkCollisiont(enemy)

Returns true if player collided with enemy or false otherwise

##### void GameObject::destroy ()

Function that acts like a destructor. Checks the type of **GameObject** it is called from and calls their corresponding functions that destroys them

##### string GameObject::getPathToTexture ()

Returns path to the image used for the sprite.

###### Returns

- Returns string that holds the path to the image used for the sprite

###### Usage:

string s;

GameObject\* object;

s = object->getPathToTexture()

s now holds the path to the object’s texture

##### void GameObject::handleInput (Input \* *m\_input*)

Used to dictate how much the player's position will offset when pressing a specified key.

###### Parameters

|  |  |
| --- | --- |
| *m\_input* | - Takes in an input variable to use to handle key press events |

##### void GameObject::initSprite (string *path*, int *x*, int *y*, bool *transparent*, int *layer*)

Initializes a sprite object.

###### Usage:

GameObject\* spriteObj;

spriteObj->initSprite(“/assets/asset.bmp”, 10, 20, true, 1)

Will initialize a sprite with the image “asset.bmp” at 10 on the x axis and 20 on the y axis with transparency applied to the image on the layer 1 of the scene.

###### Parameters

|  |  |
| --- | --- |
| *path* | - Path to the image for the sprite |
| *x* | - Location on the x Axis of the scene |
| *y* | - Location on the y Axis of the scene |
| *transparent* | - Use transparency on the image ? |
| *layer* | - Layer to be drawn to |

##### void GameObject::initText (const char \* *path*)

Initializes the text with a given font.

*path* - Path to the font used for the text

###### Usage:

GameObject\* text;

text->initText(“assets/sans.ttf”)

Will load a font from the given path

##### void GameObject::initTrigger (int *x*, int *y*)

Initializes a trigger object.

###### Parameters

|  |  |
| --- | --- |
| *x* | - Location on the x Axis of the scene |
| *y* | - Location on the y Axis of the scene |

###### Usage:

GameObject\* trigger

trigger->initTrigger(10, 20)

This will initialize a trigger object at 10 on the x axis and 20 on the y axis.

##### void GameObject::offsetPosition (int *x*, int *y*)

Offset the position by an amount

###### Parameters

|  |  |
| --- | --- |
| *x* | - x ammount of offset on the x Axis |
| *y* | - y ammount of offset on the y Axis |

###### Usage:

GameObject\* object;

object->offsetPosition(10,20);

Object’s position on the x axis will be increased by 10 and on the y axis by 20

##### void GameObject::removeObject (std::vector< GameObject \* > & *Objects*)

Removes the game object from a vector of game objects

###### Parameters

|  |  |
| --- | --- |
| *Objects* | - Vector of game objects to remove the object from |
| *name* |  |

###### Usage:

GameObject\* object;

vector<GameObject\*> objects;

object->removeObject(objects)

Removes the GameObject “object” to the vector “objects”

##### void GameObject::render ()

Calls according function based on the object's type. For SPRITE type objects calls draw() For TRIGGER type objects prints created location. For TEXT type objects doesn't do anything. Call **UpdateText()** instead.

##### void GameObject::setCoords (int *x*, int *y*)

Sets the value of x and y of the game object.

###### Parameters

|  |  |
| --- | --- |
| *x* | - Sets the value of x Axis to "x" |
| *y* | - Sets the value of y Axis to "y" |

###### Usage:

GameObject\* object;

object->setPosition(10,20);

Object’s position on the x axis will be 10 and on the y axis 20

##### void GameObject::updateInput ()

Not used. For debugging.

##### void GameObject::UpdateText (string *msg*, int *x*, int *y*, int *fontSize*, SDL\_Color *colour*)

Updates text on screen.

###### Parameters

|  |  |
| --- | --- |
| *msg* | - Text to be shown |
| *x* | - Location on the x axis of the text |
| *y* | - Location on the y axis of the text |
| *fontSize* | - Font size |
| *colour* | - Color of the text |

###### Usage:

GameObject\* text;

text->initText(“font.ttf”);

text->UpdateText(“hello” , 10, 20, 10, {255, 0 ,0})

This will initialize a text game object and will update it to print out at 10 on the x axis and 20 on the y axis the word “hello” with a font dimension of 10 and the color red.

#### **Member Data Documentation**

* int GameObject::layer

* coords GameObject::objCoords

* Type GameObject::type

## Sprite – Class reference

#include <Sprite.h>

Class that is responsible for holding information about Sprite type objects from the scene. This class works hand-in-hand with bitmap meaning that the bitmap class is mainly responsible for loading the texture file and passing it through to this class. Sprite class handles movement functionality.

### **Public Member Functions**

* **Sprite** (SDL\_Renderer \*m\_gameRenderer, string path, int x, int y, bool transparent)

*Constructor. Creates a bitmap from the asset given in the "path" variable and set's it's position according to the "x" and "y" variables. Sets if the image will use transparency using the "transparency" boolean variable.*

* void **draw** ()

*Draws the sprite on screen by calling the draw function of the bitmap.*

* void **destroy** ()

*Destroys the bitmap and the input members when called.*

* void **updateInput** ()

*Used for debugging. Not used at the moment.*

* void **setPosition** (int x, int y)

*Sets the value of x and y of the sprite.*

* void **offsetPosition** (int x, int y)

*Offset the position by an ammount*

* vector< int > **getPosition** ()

*Returns a vector with two values positioned on the [0] and [1] positions inside the vector. [0] - position on the x axis [1] - position on the y axis*

* SDL\_Point **getsize** ()

*Gets size of the sprite's texture.*

* string **getPathToTexture** ()

*Returns path to the image used for the sprite.*

### **Public Attributes**

* int **xCoord**
* int **yCoord**

### **Constructor & Destructor Documentation**

#### Sprite::Sprite (SDL\_Renderer \* m\_gameRenderer, string path, int x, int y, bool transparent)

Constructor. Creates a bitmap from the asset given in the "path" variable and set's it's position according to the "x" and "y" variables. Sets if the image will use transparency using the "transparency" boolean variable.

##### Parameters

|  |  |
| --- | --- |
| *m\_gameRenderer* | - Renderer responsible for drawing elements |
| *path* | - Path to file to be used for the bitmap |
| *x* | - Position on x axis |
| *y* | - Position on y axis |
| *transparent* | - Use transparency ? |

### **Member Function Documentation**

#### void Sprite::destroy ()

Destroys the bitmap and the input members when called.

##### Usage:

sprite->destroy()

#### void Sprite::draw ()

Draws the sprite on screen by calling the draw function of the bitmap.

##### Usage:

sprite->draw()

Will draw the sprite using the bitmap’s draw method

#### string Sprite::getPathToTexture ()

Returns path to the image used for the sprite.

##### Returns

- Returns string that holds the path to the image used for the sprite

##### Usage:

string s = sprite-> getPathToTexture ()

Will put into s the path to the spirte’s texture.

#### vector< int > Sprite::getPosition ()

Returns a vector with two values positioned on the [0] and [1] positions inside the vector. [0] - position on the x axis [1] - position on the y axis

##### Returns

Returns a vector with two values positioned on the [0] and [1] positions inside the vector.

##### Usage:

vector<int> coords = sprite->getPosition();

int x = coords[0];

int y = coords[1];

x and y will now store the x and y values of the position of the sprite.

#### SDL\_Point Sprite::getsize ()

Gets size of the sprite's texture.

##### Returns

Returns an SDL\_Point type object that will store the width and the height of the texture: x holds the widht and y holds the height

##### Usage:

SDL\_Point size = sprite->getsize();

//size.x will hold the width of the sprite

//size y will hold the height of the sprite

#### void Sprite::offsetPosition (int x, int y)

Offset the position by an ammount

##### Parameters

|  |  |
| --- | --- |
| *x* | - x ammount of offset on the x Axis |
| *y* | - y ammount of offset on the y Axis |

##### Usage:

sprite->offsetPosition(10,20)

Sprite’s position on the x axis will be increased by 10 and on the y axis by 20

#### void Sprite::setPosition (int x, int y)

Sets the value of x and y of the sprite.

##### Parameters

|  |  |
| --- | --- |
| *x* | - Sets the value of x Axis to "x" |
| *y* | - Sets the value of y Axis to "y" |

##### Usage:

sprite->setPosition(10,20)

Sprite’s position on the x axis will be 10 and on the y axis 20

#### void Sprite::updateInput ()

Used for debugging. Not used at the moment.

### **Member Data Documentation**

#### int Sprite::xCoord

#### int Sprite::yCoord

## input – Class reference

#include <input.h>

Class that handles the user input taken from the input. This class allows the user to modify the inputs for the game scene and set functionality for each of them as desired. It is used each frame to update the input and act accordingly.

### **Public Member Functions**

* **Input** ()

*Constructor. Goes through each key in KEYS\_PRESSED\_LIST and sets their 'pressed' status to false.*

* **~Input** ()

*Destructor. Currently empty.*

* void **Update** (void)

*Checks for a key press event among every key from KEYS\_PRESSED\_LIST and updates their 'pressed' status.*

* void **HandleKeys** ()

*Debugging function currently out of use*

* bool **KeyIsPressed** (**KEYS\_PRESSED\_LIST** key)

*Returns if a given key from the KEYS\_PRESSED\_LIST is currently being pressed. Returns true if is being pressed and false otherwise.*

### **Constructor & Destructor Documentation**

#### Input::Input ()

Constructor. Goes through each key in KEYS\_PRESSED\_LIST and sets their 'pressed' status to false.

#### Input::~Input ()

Destructor. Currently empty.

### **Member Function Documentation**

#### void Input::HandleKeys ()

Debugging function currently out of use

#### bool Input::KeyIsPressed (KEYS\_PRESSED\_LIST key)

Returns if a given key from the KEYS\_PRESSED\_LIST is currently being pressed. Returns true if is being pressed and false otherwise.

##### Parameters

|  |  |
| --- | --- |
| *key* | - Key from the keys list |

##### Returns

- Returns if the key given is currently pressed

#### void Input::Update (void )

Checks for a key press event among every key from KEYS\_PRESSED\_LIST and updates their 'pressed' status.

## ResourceManager – Class reference

#include <ResourceManager.h>

Class that loads image files into the engine to be used by the objects inside of it that require them. Allows to load an image into a bitmap object more easily.

### **Public Member Functions**

* **ResourceManager** ()
* **~ResourceManager** ()
* **BitmapData** \* **Load** (const std::string fileName, bool useTransparancy, SDL\_Renderer \*PRenderer)
* void **Unload** (const std::string fileName)

### **Static Public Member Functions**

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

### **Public Attributes**

* map< string, **BitmapData** \* > **ResourceManager\_TextureMap**

### **Constructor & Destructor Documentation**

#### ResourceManager::ResourceManager ()

#### ResourceManager::~ResourceManager ()

### **Member Function Documentation**

#### static ResourceManager \* ResourceManager::instance ()[inline], [static]

#### BitmapData \* ResourceManager::Load (const std::string fileName, bool useTransparancy, SDL\_Renderer \* PRenderer)

#### void ResourceManager::Unload (const std::string fileName)

### **Member Data Documentation**

#### map<string, BitmapData\*> ResourceManager::ResourceManager\_TextureMap

## Scene – Class reference

#include <Scene.h>

Class that represents the demo game for the engine.

### **Public Member Functions**

* **Scene** (**Input** \*m\_input, SDL\_Renderer \*s\_Renderer, int window\_w, int window\_h)

*Constructor. Creates a new* ***Scene*** *that takes in an* ***Input*** *variable to handle user input, a Renderer variable that will get passed data to render and the current window's width and height.*

* **~Scene** ()

*Destructor. Unloads and destroys scene objects to free memory and allow a clean exit from the scene.*

* void **destroyScene** ()

*Destroys every element of sceneObjects vector and cleans the memory.*

* void **saveScene** (vector< **GameObject** \* > savedSceneObjects)

*Save the scene elements inside a given vector.*

* void **Update** ()

*Update handles rendering of every frame. Anything put in here will run every frame.*

### **Public Attributes**

* SDL\_Renderer \* **m\_Renderer**
* int **OldTime**
* **Frametime** **frameTimes**
* bool **AIdirection**
* int **offset**
* int **initialPos**
* std::vector< **GameObject** \* > **sceneObjects**
* **GameObject** \* **sceneText**
* **GameObject** \* **background**
* **GameObject** \* **m\_monster**
* **GameObject** \* **m\_monsterTrans**
* **GameObject** \* **m\_monsterTransKeyed**
* **GameObject** \* **m\_movingMonster**
* string **myString** = " "

### **Constructor & Destructor Documentation**

#### Scene::Scene (Input \* m\_input, SDL\_Renderer \* s\_Renderer, int window\_w, int window\_h)

Constructor. Creates a new **Scene** that takes in an **Input** variable to handle user input, a Renderer variable that will get passed data to render and the current window's width and height.

##### Parameters

|  |  |
| --- | --- |
| *m\_input* | - Takes in input from user and passes it around objects that require it |
| *s\_Renderer* | - Renderer that will handle rendering of scene data |
| *window\_w* | - Current window's width |
| *window\_h* | - Current window's height |

#### Scene::~Scene ()

Destructor. Unloads and destroys scene objects to free memory and allow a clean exit from the scene.

### **Member Function Documentation**

#### void Scene::destroyScene ()

Destroys every element of sceneObjects vector and cleans the memory.

#### void Scene::saveScene (vector< GameObject \* > savedSceneObjects)

Save the scene elements inside a given vector.

##### Parameters

|  |  |
| --- | --- |
| *savedSceneObjects* | - Vector that will store the current scene's objects |

#### void Scene::Update (void )

Update handles rendering of every frame. Anything put in here will run every frame.

### **Member Data Documentation**

#### bool Scene::AIdirection

#### GameObject\* Scene::background

#### Frametime Scene::frameTimes

#### int Scene::initialPos

#### GameObject\* Scene::m\_monster

#### GameObject\* Scene::m\_monsterTrans

#### GameObject\* Scene::m\_monsterTransKeyed

#### GameObject\* Scene::m\_movingMonster

#### SDL\_Renderer\* Scene::m\_Renderer

#### string Scene::myString = " "

#### int Scene::offset

#### int Scene::OldTime

#### std::vector<GameObject\*> Scene::sceneObjects

#### GameObject\* Scene::sceneText

## SceneManager – Class reference

#include <SceneManager.h>

Class that can Load, Unload and Save scenes into the engine.

### **Public Member Functions**

* **SceneManager** (**Input** \*m\_input, int w, int h)

*Constructor. Initializes SDL and creates a window.*

* **~SceneManager** ()

*Destroys created window.*

* void **LoadScene** ()

*Creates a renderer and ties it to a window then creates the scene "Scene".*

* void **UnloadScene** ()

*Saves current scene's objects inside savedSceneObjects then destroys the scene.*

### **Public Attributes**

* **Scene** \* **currentScene**

### **Constructor & Destructor Documentation**

#### SceneManager::SceneManager (Input \* m\_input, int w, int h)

Constructor. Initializes SDL and creates a window.

##### Parameters

|  |  |
| --- | --- |
| *m\_input* | - **Input** variable that is going to be passed to scenes and objects when required |
| *w* | - Window width |
| *h* | - Window height |

#### SceneManager::~SceneManager ()

Destroys created window.

### **Member Function Documentation**

#### void SceneManager::LoadScene ()

Creates a renderer and ties it to a window then creates the scene "Scene".

##### Usage:

sceneManager->LoadScene()

Will load scene named Scene

#### void SceneManager::UnloadScene ()

Saves current scene's objects inside savedSceneObjects then destroys the scene.

### **Member Data Documentation**

#### Scene\* SceneManager::currentScene

# **File Documentation**

## main.cpp

#include <iostream>

#include "input.h"

#include "SceneManager.h"

### Code:

#include <iostream>

#include "input.h"

#include "SceneManager.h"

**int** main(**int** argc, **char**\* argv[])

{

Input\* input = **new** Input();

SceneManager\* game = **new** SceneManager(input, 1270, 720); // We create a window of 1270x720 resolution

game->LoadScene(); // We load our game scene

**while** (game && input)

{

**while** (!input->KeyIsPressed(KEY\_ESCAPE))

{

// We update the game every 33 frames

game->currentScene->Update();

input->Update();

SDL\_Delay(33);

}

// We cleanup

game->UnloadScene();

**delete** game;

game = **nullptr**;

**delete** input;

input = **nullptr**;

SDL\_Quit();

}

// Exit

**return** 0;

}

## bitmap.h

#include <string>

#include “SDL\_render.h”

#include “ResourceManager.h”

### Classes:

* struct SDL\_Surface
* struct SDL\_Texture
* struct SDL\_Renderer
* class Bitmap

### Code:

1 #pragma once

2

3 #include<string>

4 #include "SDL\_render.h"

5 #include "ResourceManager.h"

6

7

8 struct SDL\_Surface;

9 struct SDL\_Texture;

10 struct SDL\_Renderer;

11

12

13

14 class Bitmap

15 {

16 private:

17

18 //------------------Variables------------------

19

20 SDL\_Surface\* m\_pbitmapSurface;

21 SDL\_Texture\* m\_pbitmapTexture;

22 SDL\_Renderer\* m\_pRenderer;

23

24 int m\_x, m\_y; // auxiliary coords

25 string file;

26

27 public:

28

29 //------------------Constructor and destructor------------------

30

39 Bitmap(SDL\_Renderer\* renderer, std::string fileName, int xpos, int ypos, bool useTransparency = false);

40

44 ~Bitmap();

45

46 //------------------Methods------------------

47

52 SDL\_Texture\* getTexture();

53

58 string getFile();

59

64 void setX(int x);

65

70 void setY(int y);

71

76 int getX();

77

82 int getY();

83

87 void draw();

88 };

## bitmap.cpp

### Code:

#include "bitmap.h"

**using** **namespace** std;

Bitmap::Bitmap(SDL\_Renderer\* renderer, string fileName, **int** xpos, **int** ypos, **bool** useTransparency)

{

// Storing the renderer for future configuring and drawing

m\_pRenderer = renderer;

// Create the bitmap surface and texture

BitmapData\* Data = ResourceManager::instance()->Load(fileName.c\_str(),useTransparency,m\_pRenderer);

m\_pbitmapSurface = Data->surf;

m\_pbitmapTexture = Data->tex;

// Coords

m\_x = xpos;

m\_y = ypos;

// Filename of bitmap

file = fileName;

}

Bitmap::~Bitmap()

{

**if** (m\_pbitmapTexture)

SDL\_DestroyTexture(m\_pbitmapTexture);

**if** (m\_pbitmapSurface)

SDL\_FreeSurface(m\_pbitmapSurface);

}

**void** Bitmap::draw()

{

// Render the bitmap at the x/y coords

**if** (m\_pbitmapTexture)

{

SDL\_Rect destRect = { m\_x,m\_y,m\_pbitmapSurface->w,m\_pbitmapSurface->h };

SDL\_RenderCopy(m\_pRenderer, m\_pbitmapTexture, NULL, &destRect);

}

}

**void** Bitmap::setX(**int** x)

{

m\_x = x;

}

**void** Bitmap::setY(**int** y)

{

m\_y = y;

}

SDL\_Texture\* Bitmap::getTexture()

{

**return** m\_pbitmapTexture;

}

**int** Bitmap::getX()

{

**return** m\_x;

}

**int** Bitmap::getY()

{

**return** m\_y;

}

string Bitmap::getFile()

{

**return** file;

}

## Debug.h

#include <vector>

### Classes:

* struct Frametime

### Frametime Struct Reference:

#include <Debug.h>

#### Public Member Functions

* void **push** (float frameTime)
* float **averageTime** ()

#### Public Attributes

* std::vector< float > **FrameTimeQueue**
* uint32\_t **capacity** = 240

#### Member Function Documentation

##### float Frametime::averageTime ()[inline]

##### void Frametime::push (float *frameTime*)[inline]

#### Member Data Documentation

##### uint32\_t Frametime::capacity = 240

##### std::vector<float> Frametime::FrameTimeQueue

### Code:

1 #pragma once

2 #include <vector>

3

4 struct Frametime

5 {

6 public:

7 std::vector<float> FrameTimeQueue; // Vector that holds the frames

8

9 uint32\_t capacity = 240; // How many frames to store at a time

10 void push(float frameTime)

11 {

12 if (FrameTimeQueue.size() >= capacity) // If we hit the limit

13 {

14 FrameTimeQueue.erase(FrameTimeQueue.begin()); // We clear the first element

15 FrameTimeQueue.push\_back(frameTime); // And we add another one over on the back

16 }

17 else

18 {

19 FrameTimeQueue.push\_back(frameTime); // We push a frame if the limit is not reached

20 }

21

22 }

23

24 float averageTime() // Calculates an average of the frames in the vector

25 {

26 float average = 0;

27 for (int i = 0; i < FrameTimeQueue.size(); i++)

28 {

29 average += FrameTimeQueue[i];

30 }

31

32 return average / FrameTimeQueue.size(); // Returns the average value

33 }

34

35 };

## GameObject.h

#include <iostream>

#include "vector"

#include "string"

#include "Sprite.h"

#include "SDL\_ttf.h"

### Classes

* struct **coords**
* class GameObject

### Enumerations

* enum Type { SPRITE, TRIGGER, TEXT, NUM\_ELEMENTS }

### Enumeration Type Documentation

#### enum Type

##### Enumerator:

|  |  |
| --- | --- |
| SPRITE | Object is a Sprite |
| TRIGGER | Object is a Trigger |
| TEXT | Object is text |
| NUM\_ELEMENTS | Number of elements in the enumerator |

### coords Struct Reference

#include <GameObject.h>

#### Public Attributes

* int **x**
* int **y**

#### Member Data Documentation

##### int coords::x

##### int coords::y

### Code:

1 #pragma once

2 #include <iostream>

3

4 #include "vector"

5 #include "string"

6

7 #include "Sprite.h"

8 #include "SDL\_ttf.h"

9

10 using namespace std;

11

12 // Structure to store object's coords

13 struct coords

14 {

15 int x;

16 int y;

17 };

18

19 // Enum to store type of objects

20 enum Type

21 {

22 SPRITE,

23 TRIGGER,

24 TEXT,

25 NUM\_ELEMENTS

26 };

27

28 class GameObject

29 {

30 private:

31

32 //------------------Variables------------------

33

34 Sprite\* m\_Sprite; // Sprite of the object if applicable

35 Input\* input; // Input

36 string name; // Name of the object

37 SDL\_Renderer\* game; // Renderer

38

39 // Font

40 TTF\_Font\* defaultFont;

41 const char\* fontPath;

42 int fontSize;

43

44

45 public:

46

47 //------------------Constructor------------------

48

49

57 GameObject(SDL\_Renderer\* m\_game, Type objType, string objName);

58

59

60 //------------------Variables------------------

61

62 Type type; // Determines type of the object (SPRITE, TRIGGER, TEXT)

63 coords objCoords; // Srores object's coords

64 int layer; // Determines the layer where the object is situated

65

66

67 //------------------Methods------------------

68

73 void addObject(std::vector<GameObject\*>& Objects);

74

80 void removeObject(std::vector<GameObject\*>& Objects);

81

82

83

92 void initSprite(string path, int x, int y, bool transparent, int layer);

93

99 void initTrigger(int x, int y);

100

105 void initText(const char\*);

106

107

108

109

115 void setCoords(int x, int y);

116

122 void offsetPosition(int x, int y);

123

128 void autoMoveYAxis(bool dir);

129

137 bool checkCollision(GameObject\* object);

138

139

140

141

150 void UpdateText(string msg, int x, int y, int fontSize, SDL\_Color colour);

151

158 void render();

159

164 void destroy();

165

166

167

172 void handleInput(Input\* m\_input);

173

177 void updateInput();

178

179

180

185 string getPathToTexture();

186

187

188

189 };

190

## GameObject.cpp

### Code:

#include "GameObject.h"

GameObject::GameObject(SDL\_Renderer\* m\_game, Type objType, string objName)

{

game = m\_game;

type = objType;

name = objName;

layer = -1;

}

**void** GameObject::destroy()

{

**if** (type == SPRITE)

m\_Sprite->destroy();

**if** (type == TEXT)

{

TTF\_CloseFont(defaultFont);

}

}

**void** GameObject::render()

{

**switch** (type)

{

**case** SPRITE: m\_Sprite->draw(); **break**;

**case** TRIGGER: cout << "Trigger created at (x,y): "<<objCoords.x<<" "<< objCoords.y << endl; **break**;

**case** TEXT: **break**;

**default**: cout << "Unrecognized game object type !" << endl; **break**;

}

}

**void** GameObject::updateInput()

{

m\_Sprite->updateInput();

}

string GameObject::getPathToTexture()

{

**return** m\_Sprite->getPathToTexture();

}

**void** GameObject::initSprite(string path, **int** x, **int** y, **bool** transparent, **int** layer)

{

**if** (type == SPRITE)

{

**this**->layer = layer;

**this**->m\_Sprite = **new** Sprite(**this**->game, path, x, y, transparent);

**this**->objCoords.x = x;

**this**->objCoords.y = y;

}

**else**

cout << "Object is not a sprite/n";

}

**void** GameObject::initTrigger(**int** x, **int** y)

{

**if** (type == TRIGGER)

{

**this**->objCoords.x = x;

**this**->objCoords.y = y;

}

}

**void** GameObject::initText(**const** **char**\* path)

{

TTF\_Init();

fontPath = path;

}

**void** GameObject::UpdateText(string msg, **int** x, **int** y, **int** fontSize, SDL\_Color colour)

{

{

TTF\_Font\* font;

font = TTF\_OpenFont(fontPath, fontSize);

SDL\_Surface\* surface = **nullptr**;

SDL\_Texture\* texture = **nullptr**;

**int** texW = 0;

**int** texH = 0;

surface = TTF\_RenderText\_Solid(font, msg.c\_str(), colour);

**if** (!surface)

{

//surface not loaded? output the error

printf("SURFACE for font not loaded !\n");

printf("%s\n", SDL\_GetError());

}

**else**

{

texture = SDL\_CreateTextureFromSurface(game, surface);

**if** (!texture)

{

printf("SURFACE for font not loaded ! \n");

printf("%s\n", SDL\_GetError());

}

**else**

{

SDL\_QueryTexture(texture, NULL, NULL, &texW, &texH);

SDL\_Rect textRect = { x, y, texW, texH };

SDL\_RenderCopy(game, texture, NULL, &textRect);

}

}

**if** (texture)

SDL\_DestroyTexture(texture);

**if** (surface)

SDL\_FreeSurface(surface);

}

}

**void** GameObject::handleInput(Input\* m\_input)

{

**if** (type == SPRITE)

{

**this**->input = m\_input;

**if** (input->KeyIsPressed(KEY\_W))

**this**->offsetPosition(0, -10);

**if** (input->KeyIsPressed(KEY\_S))

**this**->offsetPosition(0, 10);

**if** (input->KeyIsPressed(KEY\_A))

**this**->offsetPosition(-10, 0);

**if** (input->KeyIsPressed(KEY\_D))

**this**->offsetPosition(10, 0);

}

**else**

cout << "Only sprites can handle input.\n";

}

**void** GameObject::offsetPosition(**int** x, **int** y)

{

**this**->objCoords.x = m\_Sprite->xCoord;

**this**->objCoords.y = m\_Sprite->yCoord;

m\_Sprite->offsetPosition(x, y);

}

**void** GameObject::autoMoveYAxis(**bool** dir)

{

**bool** changeDir = 0;

**if** (type == SPRITE)

{

**if** (dir == 0)

{

**this**->objCoords.y = **this**->objCoords.y + 10;

**this**->setCoords(**this**->objCoords.x, **this**->objCoords.y);

}

**else**

{

**this**->objCoords.y = **this**->objCoords.y - 10;

**this**->setCoords(**this**->objCoords.x, **this**->objCoords.y);

}

}

**else**

cout << "Behaviour only allowed by SPRITE type game objects\n";

}

**void** GameObject::setCoords(**int** x, **int** y)

{

**this**->objCoords.x = m\_Sprite->xCoord;

**this**->objCoords.y = m\_Sprite->yCoord;

m\_Sprite->setPosition(x, y);

}

**bool** GameObject::checkCollision(GameObject\* object)

{

SDL\_Point thisObj; // variable that holds the width and height of the current object's sprite texture

SDL\_Point otherObj; // variable that holds the width and height of the compared object's sprite texture

// if the object type is trigger then the width and height of the object are set to 0 as they dont have a sprite

**if** ((**this**->type == SPRITE) && (object->type == SPRITE))

{

thisObj = **this**->m\_Sprite->getsize();

otherObj = object->m\_Sprite->getsize();

}

**else** **if** ((**this**->type == TRIGGER) && (object->type == SPRITE))

{

thisObj.x = 0;

thisObj.y = 0;

otherObj = object->m\_Sprite->getsize();

}

**else** **if** ((**this**->type == SPRITE) && (object->type == TRIGGER))

{

thisObj = **this**->m\_Sprite->getsize();

otherObj.x = 0;

otherObj.y = 0;

}

**else** **if** ((**this**->type == TRIGGER) && (object->type == TRIGGER))

{

thisObj.x = 0;

thisObj.y = 0;

otherObj.x = 0;

otherObj.y = 0;

}

**else**

{

cout << "Collision can only happen between sprites and triggers\n";

**return** **false**;

}

**if** (**this**->layer <= object->layer)

{

**if** (((**this**->objCoords.x + thisObj.x >= object->objCoords.x) && (**this**->objCoords.x + thisObj.x <= object->objCoords.x + otherObj.x)) ||

((**this**->objCoords.x <= object->objCoords.x + otherObj.x) && (**this**->objCoords.x >= object->objCoords.x + otherObj.x)))

{

**if** (((**this**->objCoords.y + thisObj.y >= object->objCoords.y) && (**this**->objCoords.y + thisObj.y <= object->objCoords.y + otherObj.y)) ||

((**this**->objCoords.y <= object->objCoords.y + otherObj.y) && (**this**->objCoords.y >= object->objCoords.y)))

{

cout << "\n" << **this**->name;

printf(" Colliding with ");

cout << object->name << "\n";

**return** **true**;

}

}

}

**return** **false**;

}

**void** GameObject::addObject(std::vector<GameObject\*>& Objects)

{

Objects.push\_back(**this**);

}

**void** GameObject::removeObject(std::vector<GameObject\*>& Objects)

{

**for** (**int** i = 0; i < Objects.size(); i++)

**if** (Objects[i]->name == **this**->name)

Objects.erase(Objects.begin()+i);

}

## input.h

#include "SDL.h"

### Classes:

* class **Input**

### Enumerations:

* enum KEYS\_PRESSED\_LIST { KEY\_ESCAPE, KEY\_R, KEY\_G, KEY\_B, KEY\_W, KEY\_A, KEY\_S, KEY\_D, SIZE\_OF\_KEYS\_PRESSED\_ENUM }

### Enumeration Type Documentation

#### enum KEYS\_PRESSED\_LIST

##### Enumerator:

|  |  |
| --- | --- |
| KEY\_ESCAPE | ESC key on the keyboard |
| KEY\_R | R key on the keyboard |
| KEY\_G | G key on the keyboard |
| KEY\_B | B key on the keyboard |
| KEY\_W | W key on the keyboard |
| KEY\_A | A key on the keyboard |
| KEY\_S | S key on the keyboard |
| KEY\_D | D key on the keyboard |
| SIZE\_OF\_KEYS\_PRESSED\_ENUM | Number of elements in the enumerator |

### Code:

1 #pragma once

2 #include "SDL.h"

3

4 //Global vars for keys the engine will understand

5

6 enum KEYS\_PRESSED\_LIST

7 {

8 KEY\_ESCAPE, KEY\_R, KEY\_G, KEY\_B, KEY\_W, KEY\_A, KEY\_S, KEY\_D, SIZE\_OF\_KEYS\_PRESSED\_ENUM

9 };

10

11

12 class Input

13 {

14 public:

15

16 //------------------Constructor and destructor------------------

17

21 Input();

22

26 ~Input();

27

28

29 //------------------Methods------------------

30

35 void Update(void);

36

40 void HandleKeys();

41

48 bool KeyIsPressed(KEYS\_PRESSED\_LIST key);

49

50

51

52 private:

53

54 SDL\_Event m\_event;

55 bool m\_keysPressed[SIZE\_OF\_KEYS\_PRESSED\_ENUM];

56 };

## input.cpp

### Code:

#include <SDL.h>

#include <stdio.h>

#include "input.h"

Input::Input()

{

**int** i;

**for**(i=0;i< SIZE\_OF\_KEYS\_PRESSED\_ENUM;i++)

m\_keysPressed[i] = **false**;

}

Input::~Input()

{

}

**void** Input::Update(**void**)

{

//Loop through all the events

**while** (SDL\_PollEvent(&m\_event) != NULL)

{

// If an event is found we process it.

// We check for a key down event:

**if** (m\_event.type == SDL\_KEYDOWN)

{

// Cache the code of key we pressed for easier debugging.

SDL\_Keycode keyPressed = m\_event.key.keysym.sym;

// Update the keys.

**switch** (keyPressed)

{

**case** SDLK\_ESCAPE:

m\_keysPressed[KEY\_ESCAPE] = **true**;

**break**;

**case** SDLK\_r:

m\_keysPressed[KEY\_R] = **true**;

**break**;

**case** SDLK\_g:

m\_keysPressed[KEY\_G] = **true**;

**break**;

**case** SDLK\_b:

m\_keysPressed[KEY\_B] = **true**;

**break**;

**case** SDLK\_w:

m\_keysPressed[KEY\_W] = **true**;

**break**;

**case** SDLK\_a:

m\_keysPressed[KEY\_A] = **true**;

**break**;

**case** SDLK\_s:

m\_keysPressed[KEY\_S] = **true**;

**break**;

**case** SDLK\_d:

m\_keysPressed[KEY\_D] = **true**;

**break**;

}

}

// Check for key up.

**else** **if** (m\_event.type == SDL\_KEYUP)

{

SDL\_Keycode keyPressed = m\_event.key.keysym.sym;

// Update keys.

**switch** (keyPressed)

{

**case** SDLK\_r:

m\_keysPressed[KEY\_R] = **false**;

**break**;

**case** SDLK\_g:

m\_keysPressed[KEY\_G] = **false**;

**break**;

**case** SDLK\_b:

m\_keysPressed[KEY\_B] = **false**;

**break**;

**case** SDLK\_w:

m\_keysPressed[KEY\_W] = **false**;

**break**;

**case** SDLK\_a:

m\_keysPressed[KEY\_A] = **false**;

**break**;

**case** SDLK\_s:

m\_keysPressed[KEY\_S] = **false**;

**break**;

**case** SDLK\_d:

m\_keysPressed[KEY\_D] = **false**;

**break**;

}

}

}

}

**void** Input::HandleKeys()

{

**if** (KeyIsPressed(KEY\_R))

{

//cout << "r" << endl;

}

**if** (KeyIsPressed(KEY\_G))

{

//cout << "g" << endl;

}

**if** (KeyIsPressed(KEY\_B))

{

//cout << "b" << endl;

}

**if** (KeyIsPressed(KEY\_W))

{

//cout << "w" << endl;

}

**if** (KeyIsPressed(KEY\_A))

{

//cout << "a" << endl;

}

**if** (KeyIsPressed(KEY\_S))

{

//cout << "s" << endl;

}

**if** (KeyIsPressed(KEY\_D))

{

//cout << "d" << endl;

}

}

**bool** Input::KeyIsPressed(KEYS\_PRESSED\_LIST key)

{

**return** m\_keysPressed[key];

}

## ResourceManager.h

#include <map>

#include <SDL.h>

#include <string>

### Classes:

* struct **BitmapData**
* class **ResourceManager**

### Code:

1 #pragma once

2 #include <map>

3 #include <SDL.h>

4 #include <string>

5

6 using namespace std;

7

8

9 struct BitmapData

10 {

11 SDL\_Texture\* tex;

12 SDL\_Surface\* surf;

13 };

14

15

16 class ResourceManager

17 {

18 public:

19 static ResourceManager\* instance()

20 {

21 static ResourceManager\* instance;

22

23 if (!instance)

24 instance = new ResourceManager();

25

26 return instance;

27 }

28 map<string, BitmapData\*> ResourceManager\_TextureMap;

29 ResourceManager();

30 ~ResourceManager();

31

32 BitmapData\* Load(const std::string fileName, bool useTransparancy, SDL\_Renderer\* PRenderer); // Loads a bitmap

33 void Unload(const std::string fileName); // Unloads a bitmap

34

35 };

36

## ResourceManager.cpp

### Code:

#include "ResourceManager.h"

**using** **namespace** std;

ResourceManager::ResourceManager()

{

}

ResourceManager::~ResourceManager()

{

**for** (**auto** value : ResourceManager\_TextureMap)

{

**delete** value.second;

value.second = **nullptr**;

}

**delete** instance();

//instance() = nullptr;

}

**void** ResourceManager::Unload(**const** std::string fileName)

{

}

BitmapData\* ResourceManager::Load(**const** std::string fileName, **bool** useTransparancy, SDL\_Renderer\* PRenderer)

{

BitmapData\* ReturnData =NULL;

**auto** searchresult = ResourceManager\_TextureMap.find(fileName);

**if** (searchresult != ResourceManager\_TextureMap.end())

{

ReturnData = ResourceManager\_TextureMap[fileName];

}

**else** // not found, load, save and return

{

SDL\_Texture\* m\_pbitmapTexture = **nullptr**;

//if file does not exist in map

SDL\_Surface\* pTempSurface = SDL\_LoadBMP(fileName.c\_str());

**if** (!pTempSurface)

{

printf("SURFACE for bitmap '%s' not loaded\n", fileName.c\_str());

printf("%s\n", SDL\_GetError());

}

**else**

{

**if** (useTransparancy)

{

Uint32 colourKey = SDL\_MapRGB(pTempSurface->format, 255, 0, 255);

SDL\_SetColorKey(pTempSurface, SDL\_TRUE, colourKey);

}

m\_pbitmapTexture = SDL\_CreateTextureFromSurface(PRenderer, pTempSurface);

**if** (!m\_pbitmapTexture)

{

printf("TEXTURE for bitmap '%s' not loaded!\n", fileName.c\_str());

printf("%s\n", SDL\_GetError());

}

ReturnData = **new** BitmapData();

ReturnData->tex = m\_pbitmapTexture;

ReturnData->surf = pTempSurface;

ResourceManager\_TextureMap[fileName] = ReturnData;

}

}

**return** ReturnData;

}

## Scene.h

#include <stdio.h>

#include "GameObject.h"

#include "ResourceManager.h"

#include "Debug.h"

### Classes:

* class **Scene**

### Code:

1 #pragma once

2

3 #include <stdio.h>

4

5 #include "GameObject.h"

6 #include "ResourceManager.h"

7 #include "Debug.h"

8

9

10

11

12 using namespace std;

13

14 class Scene

15 {

16 private:

17

18 Input\* input;

19 int windowWidth;

20 int windowHeight;

21

22 public:

23

24

25

26 //------------------Constructor and destructor------------------

27

36 Scene(Input\* m\_input, SDL\_Renderer\* s\_Renderer, int window\_w, int window\_h);

37

41 ~Scene();

42

43

44 //------------------Variables------------------

45

46

47 // Renderer

48 SDL\_Renderer\* m\_Renderer;

49

50 // Debugging

51 int OldTime;

52 Frametime frameTimes;

53

54 // Variables used for moving the enemy

55 bool AIdirection;

56 int offset;

57 int initialPos;

58

59 // Scene objects

60 std::vector<GameObject\*> sceneObjects;

61 GameObject\* sceneText;

62 GameObject\* background;

63 GameObject\* m\_monster;

64 GameObject\* m\_monsterTrans;

65 GameObject\* m\_monsterTransKeyed;

66 GameObject\* m\_movingMonster;

67 string myString = " ";

68

69

70 //------------------Methods------------------

71

75 void destroyScene();

76

81 void saveScene(vector<GameObject\*> savedSceneObjects);

82

87 void Update();

88 };

89

## Scene.cpp

### Code:

#include "Scene.h"

Scene::Scene(Input\* m\_input, SDL\_Renderer\* s\_Renderer, **int** window\_w, **int** window\_h)

{

// Window details

windowWidth = window\_w;

windowHeight = window\_h;

// We set the renderer and the input to the passed variables

input = m\_input;

m\_Renderer = s\_Renderer;

// We initialize our objects and we add them to a vector

sceneText = **new** GameObject(m\_Renderer, TEXT, "text");

sceneText->initText("assets/DejaVuSans.ttf");

sceneText->addObject(sceneObjects);

background = **new** GameObject(m\_Renderer, SPRITE, "background"); // background image

background->initSprite("assets/Untitled.bmp",0 , 0, **false**, 1);

background->addObject(sceneObjects);

m\_monster = **new** GameObject(m\_Renderer, SPRITE, "goal"); // monster that represents the end goal of the game

m\_monster->initSprite("assets/goal.bmp", 1100, 250, **true**, 1);

m\_monster->addObject(sceneObjects);

m\_monsterTrans = **new** GameObject(m\_Renderer, SPRITE, "pickup"); //monster that represents a pickup item

m\_monsterTrans->initSprite("assets/pickup.bmp", 700, 400, **true**, 1);

m\_monsterTrans->addObject(sceneObjects);

m\_monsterTransKeyed = **new** GameObject(m\_Renderer, SPRITE, "enemy"); //moving enemy that resets the players position if collided with

m\_monsterTransKeyed->initSprite("assets/enemy.bmp", 300, 100, **true**, 1);

m\_monsterTransKeyed->addObject(sceneObjects);

m\_movingMonster = **new** GameObject(m\_Renderer, SPRITE, "player"); //player monster

m\_movingMonster->initSprite("assets/player.bmp", 0, 0, **true**, 1);

m\_movingMonster->addObject(sceneObjects);

// Variable used to dictate when the enemy will change direction

AIdirection = 0;

offset = 400; // distance in pixels that the enemy will travel downwards before returning to initial position

initialPos = m\_monsterTransKeyed->objCoords.y; // initial position of enemy

//Frametime frameTimes;

OldTime = SDL\_GetTicks();

}

Scene::~Scene()

{

destroyScene();

}

**void** Scene::destroyScene()

{

**for** (**int** i = 0; i < sceneObjects.size(); i++)

{

sceneObjects[i]->destroy(); // call the destroy function that will act according to the object's type

}

sceneObjects.clear();

cout << "Current scene has been cleared" << endl;

}

**void** Scene::saveScene(vector<GameObject\*> savedSceneObjects)

{

**for** (**int** i = 0; i < sceneObjects.size(); i++)

{

savedSceneObjects[i] = sceneObjects[i]; // place the objects inside an auxiliary vector to save them

}

}

**void** Scene::Update()

{

// We clean the renderer to make sure it's blank before rendering our scene

SDL\_RenderClear(m\_Renderer);

// We gather frame time information for debugging

Frametime frameTimes;

**int** NewTime = SDL\_GetTicks();

**int** delta = NewTime - OldTime;

frameTimes.push(delta);

// We print the average frametime for debugging

string frameTimeAvg = to\_string(frameTimes.averageTime());

// We render all objects inside our sceneObjects vector

**for** (**int** i = 0; i < sceneObjects.size(); i++)

{

sceneObjects[i]->render();

}

// We handle the user's input and allow them to move the player around

m\_movingMonster->handleInput(input);

// We tell the enemy to change direction once they arrive at certain points

**if** (m\_monsterTransKeyed->objCoords.y == initialPos)

AIdirection = 0;

**else** **if** (m\_monsterTransKeyed->objCoords.y == initialPos+offset)

AIdirection = 1;

m\_monsterTransKeyed->autoMoveYAxis(AIdirection);

// Determine behaviour of scene objects

**if** (m\_movingMonster->checkCollision(m\_monsterTransKeyed))

m\_movingMonster->setCoords(0, 0);

**if** (m\_movingMonster->checkCollision(m\_monsterTrans))

{

myString = "Picked up item !";

m\_monsterTrans->setCoords(9999, 9999);

}

**if** (m\_movingMonster->checkCollision(m\_monster))

{

m\_monster->setCoords(10000, 10000);

myString = "YOU WON !! press ESC to exit";

}

// Take input from user each frame and handle it

m\_movingMonster->updateInput();

//Draw the text on the screen

sceneText->UpdateText("Frame time average: " + frameTimeAvg, 10, windowHeight - 20, 10, ***{*** 255, 0, 0 ***}***); // for debugging

sceneText->UpdateText(myString, 50, 70, 50, ***{*** 0,255,0 ***}***); // for various events inside the scene

// Render the frame

SDL\_RenderPresent(m\_Renderer);

SDL\_Delay(16); // Slightly delay the other frame

// Reset timer after frame is done

OldTime = NewTime;

}

## SceneManager.h

#include "Scene.h"

### Classes:

* class **SceneManager**

### Code:

1 #pragma once

2 #include "Scene.h"

3

4

5 struct SDL\_Window;

6 struct SDL\_Renderer;

7 using namespace std;

8

9 class SceneManager

10 {

11

12 private:

13

14 //------------------Variables------------------

15 Input\* input = new Input();

16 SDL\_Renderer\* m\_Renderer;

17 SDL\_Window\* m\_Window;

18 int window\_w, window\_h;

19 // Scene objects vector

20 vector<GameObject\*> sceneObjects;

21 vector<GameObject\*> savedSceneObjects;

22 // Running status of scene

23 bool m\_Running = 0;

24

25

26 public:

27

28

29 //------------------Constructor and destructor------------------

30

37 SceneManager(Input\* m\_input,int w, int h);

38

42 ~SceneManager();

43

44

45 //------------------Variables------------------

46

47 Scene\* currentScene;

48

49 //------------------Methods------------------

50

54 void LoadScene();

55

59 void UnloadScene();

60

61

62

63 };

64

## SceneManager.cpp

### Code:

#include "SceneManager.h"

SceneManager::SceneManager(Input\* m\_input, **int** w, **int** h)

{

// Setting the variables

input = m\_input; // input passed to the scene

m\_Window = **nullptr**; // window variable

m\_Renderer = **nullptr**; // renderer variable

window\_w = w; // window width

window\_h = h; // window height

currentScene = **nullptr**; // current scene variable

// SDL Initialization

SDL\_Init(SDL\_INIT\_VIDEO);

SDL\_GLContext gl\_context = SDL\_GL\_CreateContext(m\_Window);

SDL\_GL\_MakeCurrent(m\_Window, gl\_context);

SDL\_GL\_SetSwapInterval(0); // Enable vsync

// Window Initialization

m\_Window = SDL\_CreateWindow(

"Game Engine", //title

250, //initial x position

50, //initial y position

window\_w, //width, in pixels

window\_h, //height, in pixels

0 //window behaviour flags (ignore for now)

);

// Error handling

**if** (!m\_Window)

{

printf("WINDOW initialisation failed: %s\n", SDL\_GetError());

printf("Press any key to continue\n");

getchar();

**return**;

}

}

SceneManager::~SceneManager()

{

**if** (m\_Window)

{

SDL\_DestroyWindow(m\_Window);

}

}

**void** SceneManager::LoadScene()

{

// First checks if there is already a scene running and returns if true

**if** (m\_Running)

{

printf("A scene is already running, please unload it before loading a new one.\n");

printf("Press any key to continue\n");

getchar();

**return**;

}

**else**

{

m\_Running = **true**;

SDL\_Renderer\* m\_Renderer;

m\_Renderer = SDL\_CreateRenderer(m\_Window, -1, 0); // We create a renderer and tie it to the current window

// Error if something fails

**if** (!m\_Renderer)

{

printf("RENDERER initialisation failed: %s\n", SDL\_GetError());

printf("Press any key to continue\n");

getchar();

**return**;

}

currentScene = **new** Scene(input, m\_Renderer, window\_w, window\_h); // We create the game scene

}

}

**void** SceneManager::UnloadScene()

{

// Error handling

**if** (!m\_Running)

{

printf("No scene is running currently. Load a scene first.\n");

printf("Press any key to continue\n");

getchar();

**return**;

}

**else**

{

m\_Running = **false**;

**if** (m\_Renderer)

{

currentScene->saveScene(savedSceneObjects);

currentScene->destroyScene();

SDL\_Delay(33);

SDL\_DestroyRenderer(m\_Renderer);

}

}

}

## Sprite.h

#include "Input.h"

#include "bitmap.h"

#include <iostream>

#include <vector>

### Classes:

* class **Sprite**

### Code:

1 #pragma once

2 #include "Input.h"

3 #include "bitmap.h"

4 #include <iostream>

5 #include <vector>

6

7 class Sprite

8 {

9 private:

10

11 Input\* input;

12 Bitmap\* bitmap; // Bitmap used to create the sprite

13

14

15 public:

16

17 //Constructor

18

28 Sprite(SDL\_Renderer\* m\_gameRenderer, string path, int x, int y, bool transparent);

29

30 //------------------Variables------------------

31

32 // Auxiliary coords

33 int xCoord;

34 int yCoord;

35

36 //------------------Methods------------------

37

41 void draw();

42

46 void destroy();

47

51 void updateInput();

52

58 void setPosition(int x, int y);

59

65 void offsetPosition(int x, int y);

66

73 vector<int> getPosition();

74

79 SDL\_Point getsize();

80

85 string getPathToTexture();

86 };

87

## Sprite.cpp

### Code:

#include "Sprite.h"

Sprite::Sprite(SDL\_Renderer\* m\_gameRenderer, string path, **int** x, **int** y, **bool** transparent)

{

bitmap = **new** Bitmap(m\_gameRenderer, path, x, y, transparent);

xCoord = bitmap->getX();

yCoord = bitmap->getY();

input = **new** Input();

}

**void** Sprite::destroy()

{

**if** (bitmap)

{

**delete** bitmap;

bitmap = **nullptr**;

}

**if** (input)

{

**delete** input;

input = **nullptr**;

}

}

**void** Sprite::setPosition(**int** x, **int** y)

{

xCoord = x;

yCoord = y;

bitmap->setX(xCoord);

bitmap->setY(yCoord);

}

**void** Sprite::offsetPosition(**int** x, **int** y)

{

xCoord += x;

yCoord += y;

bitmap->setX(xCoord);

bitmap->setY(yCoord);

}

SDL\_Point Sprite::getsize()

{

SDL\_Texture\* texture = bitmap->getTexture();

SDL\_Point size;

SDL\_QueryTexture(texture, NULL, NULL, &size.x, &size.y);

**return** size;

}

vector<**int**> Sprite::getPosition()

{

vector<**int**> returnValues;

returnValues[0] = xCoord;

returnValues[1] = yCoord;

**return** returnValues;

}

string Sprite::getPathToTexture()

{

**return** bitmap->getFile();

}

**void** Sprite::updateInput()

{

**if**(input->KeyIsPressed(KEY\_W))

cout << "w";

}

**void** Sprite::draw()

{

bitmap->draw();

}

# **Tools Programming**

## In GameObject class:

* *int checkName(vector<GameObject\*> objects);*

Checks the number of times the name of the object is found inside a game object vector.

param: objects - Vector to search in

Returns: Number of times the name is found in the vector

### Code:

int GameObject::checkName(vector<GameObject\*> objects)

{

int number = 0;

for (GameObject\* object : objects)

if (this->name == object->name)

number++;

return number;

--Found in GameObject.cpp

## In bitmap class:

* bool isPointInRect(int x, int y, bool clicked);

Checks to see if the pointer is situated inside the bitmap's texture.

Parameters:

* x - x coord of the point
* y - y coord of th point
* clicked - Is the mouse clicked?

Returns:

* true - Returns true if the pointer is inside and clicked
* false – Otherwise

### Code:

bool Bitmap::isPointInRect(int x, int y, bool clicked)

{

SDL\_Rect rect;

rect.x = m\_x;

rect.y = m\_y;

rect.h = m\_pbitmapSurface->h;

rect.w = m\_pbitmapSurface->w;

SDL\_Point mousePoint;

mousePoint.x = x;

mousePoint.y = y;

return (SDL\_PointInRect(&mousePoint, &rect) && clicked);

}

--Found in bitmap.cpp

## I\_GUI class:

### I\_GUI Class Reference

#include <I\_GUI.h>

#### Public Member Functions

* **I\_GUI** (SDL\_Renderer \*game, SDL\_Window \*win)
* **~I\_GUI** ()
* SDL\_Window \* **getWindow** ()
* void **displayDemo** ()
* void **displayInfo** (**Sprite** \*m\_sprite)
* void **startFrame** ()
* void **endFrame** ()

#### Constructor & Destructor Documentation

##### I\_GUI::I\_GUI (SDL\_Renderer \* *game*, SDL\_Window \* *win*)

##### I\_GUI::~I\_GUI ()

#### Member Function Documentation

##### void I\_GUI::displayDemo ()

##### void I\_GUI::displayInfo (Sprite \* *m\_sprite*)

##### void I\_GUI::endFrame ()

##### SDL\_Window \* I\_GUI::getWindow ()

##### void I\_GUI::startFrame ()

### File Reference

#include "imgui.h"

#include "backends/imgui\_impl\_sdl.h"

#include "imgui\_sdl.h"

#include "imgui\_internal.h"

#include <iostream>

#include <SDL.h>

#include "Sprite.h"

#### Classes

* class **I\_GUI**

#### Code:

1 #pragma once

2 #include "imgui.h"

3 #include "imgui.h"

4 #include "backends/imgui\_impl\_sdl.h"

5 #include "imgui\_sdl.h"

6 #include "imgui\_internal.h"

7

8 #include <iostream>

9 #include <SDL.h>

10 #include "Sprite.h"

11

12 //class Bitmap;

13

14 class I\_GUI

15 {

16 private:

17

18

19 SDL\_Window\* currentWindow;

20 ImGuiIO io;

21

22 public:

23 I\_GUI(SDL\_Renderer\* game, SDL\_Window\* win);

24 ~I\_GUI();

25

26

27 SDL\_Window\* getWindow();

28 void displayDemo();

29 void displayInfo(Sprite\* m\_sprite);

30 //virtual void DrawGUI() = 0;

31 void startFrame();

32 void endFrame();

33 };

34

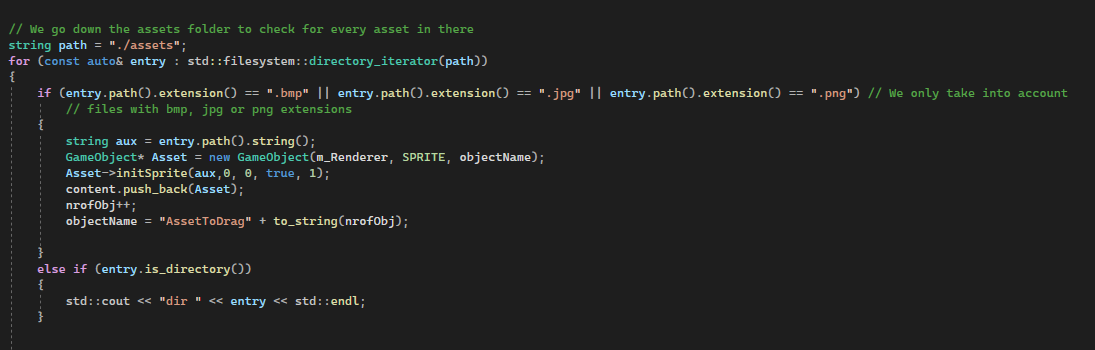
## In Scene class:

### Scene.h

Variables required for the interface


Description automatically generated

### Scene.cpp



Text

Description automatically generated

Text

Description automatically generated

A picture containing shape

Description automatically generated

# **Additional Information**

## Game scene

Graphical user interface, application, Teams

Description automatically generated

4

3

2

1

1. Player sprite

2. Enemy sprite. If collided will reset the player to starting position.

3. Pickup. If collided with it will print a message on the screen.

4. End goal of the game. If collided will display a message on the screen.

## **Editor scene**Editor scene

4**A screenshot of a video game

Description automatically generated**

3**A screenshot of a video game

Description automatically generated**

2**A screenshot of a video game

Description automatically generated**

1**A screenshot of a video game

Description automatically generated**

1. Displays the scene hierarchy

2. Displays details about a game object and allows to offset its position by dragging the sliders

3. Displays the assets inside the project’s asset folder

4. Displays debugging information (frame times)

# **References**

* Dunham, E., 2022. *Font Squirrel | DejaVu Sans Font Free by DejaVu Fonts*. [online] Fontsquirrel.com. Available at: <https://www.fontsquirrel.com/fonts/dejavu-sans>.
* Documentation helped by doxygen

Doxygen.nl. 2022. Doxygen: Doxygen. [online] Available at: <https://www.doxygen.nl/index.html> [Accessed 10 April 2022].