

# Arch Game Engine

0.1

Generated by Doxygen 1.8.11



# Contents

<b>1</b>	<b>Hierarchical Index</b>	<b>1</b>
1.1	Class Hierarchy . . . . .	1
<b>2</b>	<b>Class Index</b>	<b>3</b>
2.1	Class List . . . . .	3
<b>3</b>	<b>Class Documentation</b>	<b>5</b>
3.1	Background Class Reference . . . . .	5
3.1.1	Detailed Description . . . . .	6
3.2	Collision Class Reference . . . . .	6
3.2.1	Detailed Description . . . . .	6
3.3	Engine Class Reference . . . . .	7
3.3.1	Detailed Description . . . . .	9
3.4	Entity Class Reference . . . . .	9
3.4.1	Detailed Description . . . . .	11
3.5	Image Class Reference . . . . .	11
3.5.1	Detailed Description . . . . .	12
3.6	Input Class Reference . . . . .	12
3.6.1	Detailed Description . . . . .	14
3.7	Tileset::layer Struct Reference . . . . .	14
3.7.1	Detailed Description . . . . .	15
3.8	Object Class Reference . . . . .	15
3.8.1	Detailed Description . . . . .	17
3.9	Physics Class Reference . . . . .	18
3.9.1	Detailed Description . . . . .	18
3.10	Tile Class Reference . . . . .	18
3.10.1	Detailed Description . . . . .	19
3.11	Tileset::tile Struct Reference . . . . .	20
3.11.1	Detailed Description . . . . .	20
3.12	Tileset Class Reference . . . . .	21
3.12.1	Detailed Description . . . . .	24
	<b>Index</b>	<b>25</b>



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

Collision . . . . .	6
Engine . . . . .	7
Image . . . . .	11
Input . . . . .	12
Tileset::layer . . . . .	14
Object . . . . .	15
Background . . . . .	5
Entity . . . . .	9
Tile . . . . .	18
Physics . . . . .	18
Tileset::tile . . . . .	20
Tileset . . . . .	21



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">Background</a> . . . . .	5
<a href="#">Collision</a>	
Class used for calculating different types of collision between given Objects . . . . .	6
<a href="#">Engine</a>	
Class for declaring an engine, which does basic SDL commands like creating the window and renderer . . . . .	7
<a href="#">Entity</a>	
Class for storing health, emotion, team, etc. of an <a href="#">Object</a> . . . . .	9
<a href="#">Image</a>	
Class for loading in SDL Textures . . . . .	11
<a href="#">Input</a>	
Class for checking and storing keyboard and mouse input . . . . .	12
<a href="#">Tileset::layer</a>	
Contains a set of tiles, the width and height of the set, the x and y coordinate of the set, and the Tiles width and height . . . . .	14
<a href="#">Object</a> . . . . .	15
<a href="#">Physics</a>	
Class for doing physics functions . . . . .	18
<a href="#">Tile</a>	
An <a href="#">Object</a> class that stores the a tile value and name . . . . .	18
<a href="#">Tileset::tile</a>	
Contains the <a href="#">Tile</a> and its x and y coordinate . . . . .	20
<a href="#">Tileset</a>	
Class for loading in maps, tileset images, and then displaying them . . . . .	21



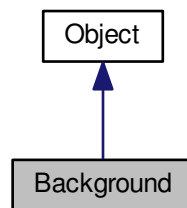


## Chapter 3

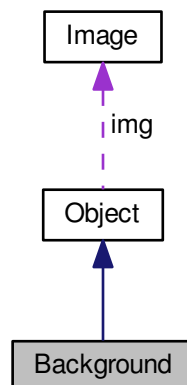
# Class Documentation

### 3.1 Background Class Reference

Inheritance diagram for Background:



Collaboration diagram for Background:



## Public Member Functions

- void **setBackground** (string file, int w, int h, SDL\_Renderer \*ren)

### 3.1.1 Detailed Description

Definition at line 6 of file background.h.

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

- background.h
- background.cpp

## 3.2 Collision Class Reference

Class used for calculating different types of collision between given Objects.

```
#include <collision.h>
```

## Public Member Functions

- bool **isTouching** (Object a, Object b)  
*Check if two objects are touching.*
- bool **outOfBoundsOf** (Object a, Object b)  
*Check if two object are not touching.*
- bool **isAbove** (Object a, Object b)  
*Check if the first object is above the second object.*
- bool **isBelow** (Object a, Object b)  
*Check if the first object is below the second object.*
- bool **isRightOf** (Object a, Object b)  
*Check if the first object is to the right of the second object.*
- bool **isLeftOf** (Object a, Object b)  
*Check if the first object is to the left of the second object.*

### 3.2.1 Detailed Description

Class used for calculating different types of collision between given Objects.

Definition at line 7 of file collision.h.

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

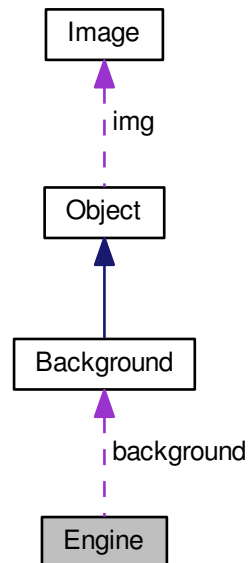
- collision.h
- collision.cpp

### 3.3 Engine Class Reference

Class for declaring an engine, which does basic SDL commands like creating the window and renderer.

```
#include <engine.h>
```

Collaboration diagram for Engine:



#### Public Member Functions

- `~Engine ()`  
*Deconstructs renderer and window and then quits SDL.*
- `SDL_Renderer * init (string s, const int &w, const int &h, int flag)`  
*Create a window with a given name, width, height, and anyother SDL\_Window flags.*
- `SDL_Renderer * init (string s, const int &w, const int &h, int flag, int it)`  
*Create a window with a given name, width, height, SDL\_Window flags, and specified SDL\_Init flags.*
- `SDL_Renderer * init (string s, const int &w, const int &h, int flag, int x, int y)`  
*Create a window with a given name, width, height, SDL\_Window flags, and specified x and y coordinate.*
- `SDL_Renderer * init (string s, const int &w, const int &h, int flag, int x, int y, int it)`  
*Create a window with a given name, width, height, SDL\_Window flags, specified x and y coordinate, and SDL\_Init flags.*
- `void setName (string s)`  
*Set window name.*
- `void setPos (int x, int y)`  
*Set window position.*
- `void setSize (int w, int y)`  
*Set window size.*

- `SDL_Renderer *` `getRenderer ()`  
*Returns screen renderer.*
- `void` `setColor` (`Uint8` r, `Uint8` g, `Uint8` b)  
*Sets SDL color.*
- `void` `render ()`  
*Call this at the end of the game loop to render.*
- `bool` `FPS ()` `const`  
*Get fps.*
- `void` `update ()`  
*Update loop time.*
- `void` `setBackground` (`Background` b)  
*Set background.*
- `void` `setBackground` (string filename)  
*Set background with filename.*
- `Background` `getBackground ()` `const`  
*Get background.*
- `void` `drawBackground ()`  
*Draw background.*
- `void` `draw` (`Object` obj)  
*Draw an object on the screen.*
- `void` `draw` (`Object` obj, int key)
- `void` `splash ()`  
*Calls splashscreen at the beginning of the game. This is automatically called unless deactivated.*
- `void` `bypassSplash` (int key)  
*Deactivates the splashscreen, requires key.*
- `bool` `hasSplashed ()`  
*Check if the splashscreen has occurred.*
- `bool` `runCustomSplash ()`  
*Run custom splashscreen. This is automatically called after splash if there is a custom splashscreen.*
- `void` `customSplash` (string file, double time, int w, int h)  
*Create a custom game splashscreen to be shown after the engine splashscreen by passing in the path to the image, the duration for it be displayed, and the size of the image.*

## Private Attributes

- `SDL_Renderer *` `engren`  
*SDL Renderer.*
- `SDL_Window *` `engwin`  
*SDL Window.*
- `int` `WIDTH`
- `int` `HEIGHT`  
*Width and height of the window.*
- `int` `simulationTime`
- `int` `realTime`  
*Timestamps used for fps loop.*
- `bool` `fps`  
*Boolean for loop.*
- `bool` `bkg`  
*Boolean for if there is a set background.*
- `Background` `background`

*Background to display.*

- Uint8 **red**
- Uint8 **green**
- Uint8 **blue**

*Colors for background.*

- bool **splashed**
- bool **custom**

*Boolean that shows if the splashscreen has occurred.*

- string **cf**

*Custom splashscreen file path.*

- double **ct**

*Custom splashscreen duration.*

- int **cw**
- int **ch**

*Custom splashscreen width and height.*

### 3.3.1 Detailed Description

Class for declaring an engine, which does basic SDL commands like creating the window and renderer.

Definition at line 26 of file engine.h.

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

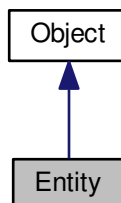
- engine.h
- engine.cpp

## 3.4 Entity Class Reference

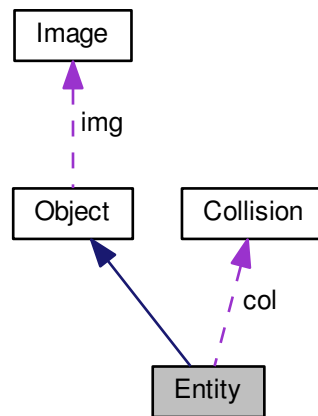
Class for storing health, emotion, team, etc. of an [Object](#).

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

Inheritance diagram for Entity:



Collaboration diagram for Entity:



## Public Member Functions

- double `getHealth` () const  
*Get `Entity`'s health.*
- void `setHealth` (double h)  
*Set the `Entity`'s health. If the health is higher then the max health it will set it to the max health.*
- double `getMaxHealth` () const  
*Get max health.*
- void `setMaxHealth` (double mh)  
*Set max health.*
- void `damage` (double d)  
*Deal damage. Subtracted from health. If health is less then zero it kills the entity.*
- void `heal` (double h)  
*Give health to the `Entity`.*
- int `getEmotion` () const  
*Get current emotion state.*
- void `setEmotion` (int e)  
*Set current emotion state.*
- int `getTeam` () const  
*Get `Entity`'s team.*
- void `setTeam` (int t)  
*Set `Entity`'s team.*
- bool `isActive` () const  
*Check if `Entity` is active.*
- void `kill` ()  
*Sets health to zero and deactivates the `Entity`.*
- void `deactivate` ()  
*Sets active to false.*
- void `activate` ()

*Sets active to true.*

- void **checkDisplayable** ([Object](#) screen)
- `SDL_Rect` **getDetect** () const
- void **setDetect** (`SDL_Rect` d)
- void **setDetectRange** (int r)
- void **setDetectRange** (int w, int h)

### Private Attributes

- double [health](#)  
*Int for the [Entity](#)'s health.*
- double [maxHealth](#)  
*Int for the [Entity](#)'s max health.*
- int [emotion](#)  
*Int for creating a range of emotional states.*
- int [team](#)  
*Int for setting the team the [Entity](#) is on.*
- bool [active](#)  
*Boolean for declaring if an entity is active.*
- [Collision](#) **col**
- `SDL_Rect` **detect**

#### 3.4.1 Detailed Description

Class for storing health, emotion, team, etc. of an [Object](#).

Definition at line 9 of file entity.h.

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

- entity.h
- entity.cpp

## 3.5 Image Class Reference

Class for loading in SDL Textures.

```
#include <image.h>
```

### Public Member Functions

- void [loadImage](#) (string file, `SDL_Renderer` \*ren)  
*Load in either a BMP or PNG file with the path and renderer..*
- void [loadPNG](#) (string file, `SDL_Renderer` \*ren)  
*Load in a PNG image with the path to the PNG file and the renderer.*
- void [loadBMP](#) (string file, `SDL_Renderer` \*ren)  
*Load in a BMP image with the path to the BMP file and the renderer.*
- `SDL_Texture` \* [getTexture](#) ()  
*Get `SDL_Texture`.*
- void [setImage](#) (`SDL_Texture` \*t)  
*Set new, preloaded texture, to [Image](#).*
- string [getFile](#) () const  
*Get path file of the image.*
- void [setFile](#) (string f)  
*Set path file to the image.*

## Private Attributes

- SDL\_Texture \* [tex](#)  
*SDL\_Texture for the image.*
- string [filename](#)  
*Path file to the image.*

### 3.5.1 Detailed Description

Class for loading in SDL Textures.

Definition at line 11 of file image.h.

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

- image.h
- image.cpp

## 3.6 Input Class Reference

Class for checking and storing keyboard and mouse input.

```
#include <input-tmp.h>
```

## Public Member Functions

- void [logPress](#) ()  
*Log all current keys and buttons being pressed.*
- bool [checkKey](#) (int [k](#))  
*Check if a key has been pressed using a given key from this class. Ex: [Input](#) i; i.checkKey(i.up);.*
- void [reset](#) ()  
*Reset all pressed keystrokes and other inputs to false. Automatically down at the beginning of each [logPress\(\)](#).*

## Public Attributes

- int [left](#)  
*Log ID for left.*
- int [right](#)  
*Log ID for right.*
- int [up](#)  
*Log ID for up.*
- int [down](#)  
*Log ID for down.*
- int [q](#)  
*Log ID for q.*
- int [w](#)  
*Log ID for w.*



- int **e**  
*Log ID for e.*
- int **r**  
*Log ID for r.*
- int **t**  
*Log ID for t.*
- int **y**  
*Log ID for y.*
- int **u**  
*Log ID for u.*
- int **i**  
*Log ID for i.*
- int **o**  
*Log ID for o.*
- int **p**  
*Log ID for p.*
- int **a**  
*Log ID for a.*
- int **s**  
*Log ID for s.*
- int **d**  
*Log ID for d.*
- int **f**  
*Log ID for f.*
- int **g**  
*Log ID for g.*
- int **h**  
*Log ID for h.*
- int **j**  
*Log ID for j.*
- int **k**  
*Log ID for k.*
- int **l**  
*Log ID for l.*
- int **z**  
*Log ID for z.*
- int **x**  
*Log ID for x.*
- int **c**  
*Log ID for c.*
- int **v**  
*Log ID for v.*
- int **b**  
*Log ID for b.*
- int **n**  
*Log ID for n.*
- int **m**  
*Log ID for m.*
- int **lshift**  
*Log ID for left shift.*
- int **rshift**

- Log ID for right shift.*
- int [shift](#)  
*Shift ID for shift.*
- int [quit](#)  
*Log ID for quit.*
- int [esc](#)  
*Log ID for esc.*
- int [mouseleft](#)  
*Log ID for left mouse click.*
- int [mousemiddle](#)  
*Log ID for middle mouse click.*
- int [mouseright](#)  
*Log ID for right mouse click.*
- int [mouseup](#)  
*Log ID for scroll up on mouse wheel.*
- int [mousedown](#)  
*Log ID for scroll down on mouse wheel.*
- int [mousex](#)  
*Log ID for mouse x coordinate.*
- int [mousey](#)  
*Log ID for mouse y coordinate.*

## Private Attributes

- bool [keys](#) [50]  
*Array that stores what buttons are down.*

### 3.6.1 Detailed Description

Class for checking and storing keyboard and mouse input.

Definition at line 9 of file input-tmp.h.

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

- input-tmp.h
- input-tmp.cpp

## 3.7 Tileset::layer Struct Reference

Contains a set of tiles, the width and height of the set, the x and y coordinate of the set, and the Tiles width and height.

## Public Attributes

- int `width` = 0  
*layers width*
- int `height` = 0  
*layers height*
- double `x` = 0  
*layers x coordinate*
- double `y` = 0  
*layers y coordinate*
- int `tw` = 0  
*Tiles width.*
- int `th` = 0  
*Tiles height.*
- vector< `tile` > `tiles`  
*Vector of tiles.*

### 3.7.1 Detailed Description

Contains a set of tiles, the width and height of the set, the x and y coordinate of the set, and the Tiles width and height.

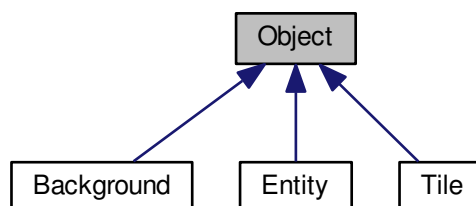
Definition at line 91 of file `tileset-tmp.h`.

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

- `tileset-tmp.h`

## 3.8 Object Class Reference

Inheritance diagram for Object:



Collaboration diagram for Object:



## Public Member Functions

- void **setDisplayable** (bool d)
- bool **isDisplayable** ([Object](#) screen)
- virtual void **checkDisplayable** ([Object](#) screen)
- void **setCoord** (double x, double y)
- void **setX** (double x)
- void **setY** (double y)
- void **move** (double x, double y)
- void **moveX** (double x)
- void **moveY** (double y)
- double **getX** () const
- double **getY** () const
- [Image](#) **getImage** () const
- void **setImage** (string file, SDL\_Renderer \*ren)
- double **getAngle** () const
- void **setAngle** (double a)
- void **center** (int w, int h)
- SDL\_Rect **getFrame** () const
- SDL\_Rect **getDest** () const
- SDL\_Rect **getPos** () const
- void **setFrame** (SDL\_Rect i)
- void **setDest** (SDL\_Rect i)
- void **setPos** (SDL\_Rect i)
- void **setFrame** (int x, int y, int w, int h)
- void **setFrameCoord** (int x, int y)
- void **setFrameSize** (int w, int h)
- void **setFrameX** (int x)
- void **setFrameY** (int y)
- void **setFrameW** (int w)
- void **setFrameH** (int h)
- int **getFrameX** () const
- int **getFrameY** () const
- int **getFrameW** () const
- int **getFrameH** () const
- void **setDest** (int x, int y, int w, int h)
- void **setDestCoord** (int x, int y)

- void **setDestSize** (int w, int h)
- void **setDestX** (int x)
- void **setDestY** (int y)
- void **setDestW** (int w)
- void **setDestH** (int h)
- int **getDestX** () const
- int **getDestY** () const
- int **getDestW** () const
- int **getDestH** () const
- void **setPos** (int x, int y, int w, int h)
- void **setPosCoord** (int x, int y)
- void **setPosSize** (int w, int h)
- void **setPosX** (int x)
- void **setPosY** (int y)
- void **setPosW** (int w)
- void **setPosH** (int h)
- int **getPosX** () const
- int **getPosY** () const
- int **getPosW** () const
- int **getPosH** () const
- void **moveFrame** (int x, int y)
- void **moveFrameX** (int x)
- void **moveFrameY** (int y)
- void **moveDest** (int x, int y)
- void **moveDestX** (int x)
- void **moveDestY** (int y)
- void **movePos** (int x, int y)
- void **movePosX** (int x)
- void **movePosY** (int y)
- void **setName** (string s)
- string **getName** ()

### Private Attributes

- [Image](#) **img**
- **SDL\_Rect frame**
- **SDL\_Rect dest**
- **SDL\_Rect pos**
- double **angle**
- string **name**
- double **x**
- double **y**
- bool **displayable**

#### 3.8.1 Detailed Description

Definition at line 9 of file object.h.

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

- object.h
- object.cpp

## 3.9 Physics Class Reference

Class for doing physics functions.

```
#include <physics-tmp.h>
```

### Public Member Functions

- [Object](#) `moveTowards` ([Object](#) cur, [Object](#) des)  
*Returns modified first [Object](#) that is moving towards the second object (I THINK).*

### 3.9.1 Detailed Description

Class for doing physics functions.

Definition at line 23 of file physics-tmp.h.

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

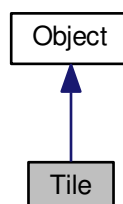
- physics-tmp.h
- physics-tmp.cpp

## 3.10 Tile Class Reference

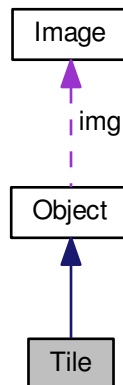
An [Object](#) class that stores the a tile value and name.

```
#include <tile.h>
```

Inheritance diagram for Tile:



Collaboration diagram for Tile:



### Public Member Functions

- void `setValue` (int v)  
*Set value of the tile. This is used when reading from a map file, etc.*
- int `getValue` ()  
*Get the value of the tile.*

### Private Attributes

- int `value`  
*Tiles value. Used for reading from a map file, etc.*

#### 3.10.1 Detailed Description

An `Object` class that stores the a tile value and name.

Definition at line 7 of file `tile.h`.

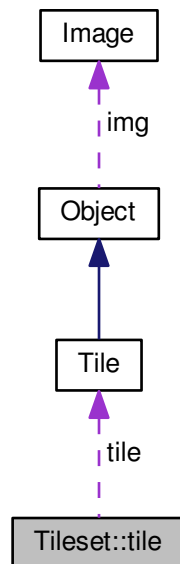
The documentation for this class was generated from the following files:

- `tile.h`
- `tile.cpp`

### 3.11 Tileset::tile Struct Reference

Contains the [Tile](#) and its x and y coordinate.

Collaboration diagram for Tileset::tile:



#### Public Attributes

- double `x` = 0  
*tile x coordinate*
- double `y` = 0  
*tile y coordinate*
- [Tile](#) `tile`  
*tile's [Tile](#)*

#### 3.11.1 Detailed Description

Contains the [Tile](#) and its x and y coordinate.

Definition at line 82 of file `tileset-tmp.h`.

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

- `tileset-tmp.h`

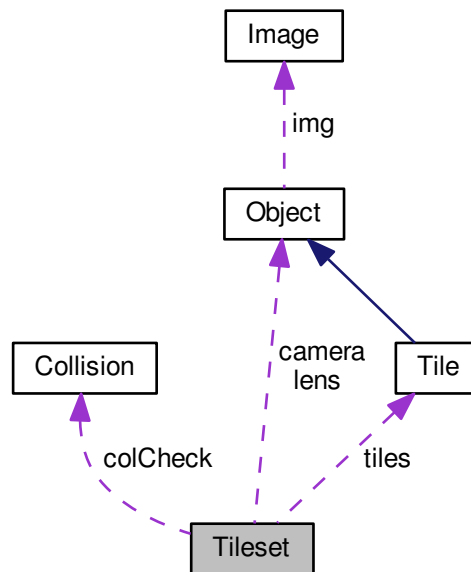


## 3.12 Tileset Class Reference

Class for loading in maps, tileset images, and then displaying them.

```
#include <tileset-tmp.h>
```

Collaboration diagram for Tileset:



### Classes

- struct **layer**  
Contains a set of tiles, the width and height of the set, the x and y coordinate of the set, and the Tiles width and height.
- struct **tile**  
Contains the **Tile** and its x and y coordinate.

### Public Member Functions

- **Tileset** (int amount)  
Amount of types of tiles.
- void **setAng** (int ang)  
Set the angle of all the tiles. Calls **pushAng()**.
- void **pushAng** ()  
Sets all tiles to the angle (I don't think this is working yet).
- void **setCoord** (double ix, double iy)  
Set the coordinate with a given x and y.
- void **setCoord** (double ix, double iy, double mx, double my)

- Set the coordinate with the given x and y and the amount to move by on the x and y.*

  - void [setWindowSize](#) (int ww, int wh)

*Set the window width and height.*
- double [getX](#) ()

*Gets the current x coordinate.*
- double [getY](#) ()

*Gets the current y coordnate.*
- vector< [Tile](#) > [loadMaps](#) (string name, string map, string img, SDL\_Renderer \*ren, int width, int height, int r, int count)

*Load in a map file with the name for all the tiles, the path to the map file, path to the tileset image, the SDL renderer, width and height of a tile, row to begin from on the image, how many tiles there are in the image.*
- vector< [Tile](#) > [loadMaps](#) (string name, string map, string img, SDL\_Renderer \*ren, int width, int height, int r, int rcount, int count)

*Load a map with a given name for the tiles, the file path to the map, the path to the tileset image, SDL renderer, width and height of a tile, row to begin on in the image, how many tiles on a certain row in the image, total amount of tiles in the image.*
- vector< [Tile](#) > [genMap](#) (string name, string map, string img, SDL\_Renderer \*ren, int width, int height, int r, int count)

*Load in a map file with the name for all the tiles, the path to the map file, path to the tileset image, the SDL renderer, width and height of a tile, row to begin from on the image, how many tiles there are in the image.*
- vector< [Tile](#) > [genMap](#) (string name, string map, string img, SDL\_Renderer \*ren, int width, int height, int r, int rcount, int count)

*Load a map with a given name for the tiles, the file path to the map, the path to the tileset image, SDL renderer, width and height of a tile, row to begin on in the image, how many tiles on a certain row in the image, total amount of tiles in the image.*
- void [loadTiles](#) (string filename, int iw, int ih)

*Read in map file with given path to the file and width and height of the tiles.*
- void [addTile](#) ([Tile](#) t)

*Push [Tile](#) in tile with given [Tile](#).*
- [Tile](#) [addTile](#) (string name, string file, SDL\_Renderer \*ren, int value, int r, int c, int width, int height)

*Generate and push [Tile](#) with tile name, path tot he tile image, SDL renderer, tile value, row and columng the tile as on in the image, the tiles width and height.*
- [Tile](#) [addTile](#) (string name, string file, SDL\_Renderer \*ren, int value, int width, int height)

*Generate and push [Tile](#) with a given name, path to image file, SDL renderer, given value, and tile width and height.*
- [Tile](#) [addTile](#) (string name, string file, SDL\_Renderer \*ren, int value, int size)

*Generate and push [Tile](#) with a given name, path to the image, SDL renderer, value, and size (used for width and height).*
- vector< [Tile](#) > [getTilesToRender](#) ()

*Get Tiles to renderer based on screen size and location.*
- vector< [Tile](#) > [getTilesToRender](#) (int w, int h)

*Get tiles to renderer based on screen size, location, and given tile width and height.*
- vector< [Tile](#) > [renderTiles](#) ([Engine](#) e)

*Get Tiles to renderer based on screen size and location and then it renders them given the [Engine](#).*
- vector< [Tile](#) > [renderTiles](#) (int w, int h, [Engine](#) e)

*Get tiles to renderer based on screen size, location, and given tile width and height and then it renders them given the [Engine](#).*
- void [move](#) (double mx, double my)

*Move map x and y amount.*
- [Object](#) [move](#) (double mx, double my, [Object](#) p)

*Given x and y amount to move and a given [Object](#) that also need to be moved, this function calculates the movement based on the Camera and Lens and then moves the [Object](#) and map if needed then returns the modified [Object](#).*
- void [calcPos](#) (double mx, double my)

*Moves all tilesets by looping through them and calling [calcSetPos\(\)](#) given movement on the x and y coordinate.*

- void [calcSetPos](#) (int i, double mx, double my)  
*Moves all tiles in a given tileset by looping through and calling [calcTilesPos\(\)](#) given the point in the array the tileset is and the movement on the x and coordinates.*
- void [calcTilesPos](#) (int i, double mx, double my)  
*Moves tile given the point on the array the tile is and the movement on the x and y coordinate.*
- void [setCameraMargin](#) (int wm, int hm)  
*Set the width and height of the Camera.*
- void [centerCamera](#) (int percentage)  
*Set the Camera size based on percentage of window to cover.*
- [Object getCamera](#) ()  
*Get the Camera.*
- void [setLensMargin](#) (int wm, int hm)  
*Set the width and height of the Lens.*
- void [centerLens](#) (int percentage)  
*Set the Lens based on the percentage of the Camera to cover.*
- [Object getLens](#) ()  
*Get the Lens.*

### Private Attributes

- int [angle](#)  
*Display angle.*
- double [x](#)  
*Display/map coordinates.*
- double [y](#)
- bool [set](#)  
*Boolean that says if the maps coordinates has already been set.*
- int [winWidth](#)  
*Window width and height.*
- int [winHeight](#)
- int [layersize](#)  
*Layer, x, and y size. (I don't think these are actually used).*
- int [xsize](#)
- int [ysize](#)
- vector< [layer](#) > [tileset](#)  
*Vector of layers (tilesets).*
- [Tile](#) \* [tiles](#)  
*Array of tiles with undeclared size.*
- [Object camera](#)  
*Camera object. When in the Camera, the map and [Object](#) move.*
- [Object lens](#)  
*Lens object. When in the Lens, only the given [Object](#) moves.*
- [Collision colCheck](#)  
*Instance of [Collision](#).*
- bool [activeCam](#)  
*Boolean that shows if the Camera and Lens have been activated.*
- bool [activeLens](#)

### 3.12.1 Detailed Description

Class for loading in maps, tileset images, and then displaying them.

Definition at line 13 of file tileset-tmp.h.

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

- tileset-tmp.h
- tileset-tmp.cpp

# Index

Background, [5](#)

Collision, [6](#)

Engine, [7](#)

Entity, [9](#)

Image, [11](#)

Input, [12](#)

Object, [15](#)

Physics, [18](#)

Tile, [18](#)

Tileset, [21](#)

Tileset::layer, [14](#)

Tileset::tile, [20](#)