

Arch Game Engine

0.1

Generated by Doxygen 1.8.11

Contents

1	Hierarchical Index	1
1.1	Class Hierarchy	1
2	Class Index	3
2.1	Class List	3
3	Class Documentation	5
3.1	Collision Class Reference	5
3.1.1	Detailed Description	5
3.2	Engine Class Reference	6
3.3	Entity Class Reference	6
3.3.1	Detailed Description	7
3.4	Image Class Reference	7
3.4.1	Detailed Description	8
3.5	Input Class Reference	8
3.5.1	Detailed Description	10
3.6	Tileset::layer Struct Reference	10
3.6.1	Detailed Description	11
3.7	Object Class Reference	11
3.7.1	Detailed Description	14
3.8	Physics Class Reference	14
3.8.1	Detailed Description	14
3.9	Tile Class Reference	15
3.9.1	Detailed Description	15
3.10	Tileset::tile Struct Reference	15
3.10.1	Detailed Description	16
3.11	Tileset Class Reference	16
3.11.1	Detailed Description	19
	Index	21

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Collision	5
Engine	6
Image	7
Input	8
Tileset::layer	10
Object	11
Entity	6
Tile	15
Physics	14
Tileset::tile	15
Tileset	16

Chapter 2

Class Index

2.1 Class List

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

Collision	Class used for calculating different types of collision between given Objects	5
Engine	Class for declaring an engine, which does basic SDL commands like creating the window and renderer	6
Entity	Class for storing health, emotion, team, etc. of an Object	6
Image	Class for loading in SDL Textures	7
Input	Class for checking and storing keyboard and mouse input	8
Tileset::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	10
Object	Class for storing an image and the source and destination to display	11
Physics	Class for doing physics functions	14
Tile	An Object class that stores the a tile value and name	15
Tileset::tile	Contains the Tile and its x and y coordinate	15
Tileset	Class for loading in maps, tileset images, and then displaying them	16

Chapter 3

Class Documentation

3.1 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.
- `Object calibrate` (Object a, Object b, int pad)
Check if first object is colliding with the second object and then return the first objects new position based on a given padding.

3.1.1 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

3.2 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:

3.3 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.

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.

3.3.1 Detailed Description

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

Definition at line 8 of file entity.h.

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

- entity.h
- entity.cpp

3.4 Image Class Reference

Class for loading in SDL Textures.

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

Public Member Functions

- void [loadImage](#) (string file, SDL_Renderer *ren)
Load in BMP image with the path to the BMP file and the 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 * [getImage](#) ()
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.

3.4.1 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

3.5 Input Class Reference

Class for checking and storing keyboard and mouse input.

```
#include <input.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.

3.5.1 Detailed Description

Class for checking and storing keyboard and mouse input.

Definition at line 9 of file input.h.

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

- input.h
- input.cpp

3.6 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.

3.6.1 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 96 of file `tileset.h`.

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

- `tileset.h`

3.7 Object Class Reference

Class for storing an image and the source and destination to display.

```
#include <object.h>
```

Inheritance diagram for Object:

Collaboration diagram for Object:

Public Member Functions

- void [setImage](#) (string file, SDL_Renderer *ren)
Set the Objects image with a BMP image path and the renderer.
- [Image](#) [getImage](#) () const
Get the [Object's Image](#).
- SDL_Texture * [getTexture](#) ()
Get the [Image's texture](#).
- void [setSource](#) (double x, double y, int w, int h)
Set the images source with the width, height, and x and y coordinates.
- void [setDest](#) (int w, int h)
Set the display destinations width and height.
- void [setDest](#) (int w, int h, double x, double y)
Set the display destinations width, height, and x and y coordinates.
- void [setDestCoord](#) (double x, double y)
Set the display destinations x and y coordinates.
- SDL_Rect [getSource](#) ()
Get the SDL_Rect of the Objects image source.
- SDL_Rect [getDest](#) ()
Get the current SDL_Rect for the Objects destination.
- SDL_Rect [getBuff](#) ()
Get the previous display destination.
- void [setSource](#) (SDL_Rect s)
Set the image source destination with an SDL_Rect.
- void [setDest](#) (SDL_Rect d)
Set the [Object's display destination](#) with an SDL_Rect.
- void [setBuff](#) (SDL_Rect b)
Set the object's previous display destination with an SDL_Rect.
- void [setSX](#) (double x)
Set the image sources x coordinate.
- void [setSY](#) (double y)
Set the image sources y coordinate.
- void [setSW](#) (int w)
Set the image sources width.
- void [setSH](#) (int h)
Set the image sources height.
- void [setDX](#) (double x)
Set the display destinations x coordinate.
- void [setDY](#) (double y)
Set the display destinations y coordinate.
- void [setDW](#) (int w)
Set the display destinations width.
- void [setDH](#) (int h)
Set the display destinations height.
- double [getSX](#) ()
Get the image sources x coordinate.
- double [getSY](#) ()
Get the image sources y coordinate.
- double [getSW](#) ()
Get the image sources width.
- double [getSH](#) ()

- Get the image sources height.*
- double [getDX](#) ()
 - Get the display destinations x coordinate.*
- double [getDY](#) ()
 - Get the display destinations y coordinate.*
- double [getDW](#) ()
 - Get the display destinations width.*
- double [getDH](#) ()
 - Get the display destinations height.*
- void [setAng](#) (double a)
 - Set the Objects angle.*
- double [getAng](#) ()
 - Get the Objects angle.*
- void [move](#) (double mx, double my)
 - Move the [Object](#) x and y amount.*
- void [center](#) (int w, int h)
 - Center the [Object](#)'s destination by the given screens (or anythings) width and height.*
- bool [collidable](#) ()
 - Check if the [Object](#) is solid, or collidable.*
- void [setSolid](#) (bool s)
 - Set the [Object](#) to be collidable/solid.*
- bool [getSolid](#) () const
 - Check if the [Object](#) is solid.*
- int [createNewFrameSet](#) (int fCount, int r, int c, int w, int h)
 - Create a new frameset with the given framecount for the set, the row to get the frameset from, the column to start at, and the width and height of each frame. Returns an int ID for the frameset.*
- SDL_Rect [createNewFrame](#) (int x, int y, int w, int h)
 - Create a new frame with a given x and y coordinate and width and height. Automatically called from [createNewFrameSet](#)().*
- void [setCurFrameSet](#) (int fs)
 - Set the current frameset with the given frameset ID from calling [createNewFrameSet](#)().*
- void [setCurFrame](#) (int f)
 - Set current frame in the frameset.*
- void [nextFrame](#) ()
 - Change to the next frame. If it reaches its end, it restarts. Called in [setCurFrameSet](#)().*
- void [resetFrameSet](#) ()
 - Set current frame to the beginning. Called in [nextFrame](#)() when it has reached its end.*
- int [getCurFrameSet](#) () const
 - Get the current frameset.*
- int [getCurFrame](#) () const
 - Get the current frame.*

Private Attributes

- [Image](#) [img](#)
 - The Objects image.*
- SDL_Rect [rect](#)
 - The images source.*
- SDL_Rect [dest](#)
 - The display destination.*

- `SDL_Rect buff`
Previous display destination.
- `double angle`
Angle to be displayed.
- `bool solid`
Boolean for if the [Object](#) is collidable/solid.
- `vector< vector< SDL_Rect > > frameset`
2D vector of framesets that store frames.
- `int curFrameSet`
Current frameset.
- `int curFrame`
Current frame.

3.7.1 Detailed Description

Class for storing an image and the source and distination to display.

Definition at line 10 of file `object.h`.

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

- `object.h`
- `object.cpp`

3.8 Physics Class Reference

Class for doing physics functions.

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

Public Member Functions

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

3.8.1 Detailed Description

Class for doing physics functions.

Definition at line 23 of file `physics.h`.

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

- `physics.h`
- `physics.cpp`

3.9 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.
- void [setName](#) (string s)
Set the Tiles name.
- string [getName](#) ()
Get the Tiles name.

Private Attributes

- int [value](#)
Tiles value. Used for reading from a map file, etc.
- string [name](#)
Tile name.

3.9.1 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

3.10 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](#)

3.10.1 Detailed Description

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

Definition at line 87 of file `tileset.h`.

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

- `tileset.h`

3.11 Tileset Class Reference

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

```
#include <tileset.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 coordinate.
- 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 to the tile image, SDL renderer, tile value, row and column 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 render based on screen size and location.
- vector< **Tile** > **getTilesToRender** (int w, int h)
Get tiles to render based on screen size, location, and given tile width and height.
- 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.
- void **setSolid** (int s, int l)

Sets all tiles from a given start to end to solid. (I don't think this works yet).
- void **setSolid** (int l)

Sets all tiles in a given tileset from a given start to end to solid. (I don't think this works yet).
- void **setSolid** (int s, int e, int l)

Sets a certain tile value to solid.
- void **setPassable** (int s, int l)

Sets all tiles from a given start to end to passable (not solid). (I don't think this works yet).
- void **setPassable** (int s, int e, int l)

Set all tiles in a given tileset from a given start to end to passable (not solid). (I don't think this works yet).
- void **setPassable** (int l)

Set a certain tile value to passable (not solid). (I don't think this works yet).

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**

3.11.1 Detailed Description

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

Definition at line 11 of file [tileset.h](#).

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

- [tileset.h](#)
- [tileset.cpp](#)

Index

Collision, [5](#)

Engine, [6](#)

Entity, [6](#)

Image, [7](#)

Input, [8](#)

Object, [11](#)

Physics, [14](#)

Tile, [15](#)

Tileset, [16](#)

Tileset::layer, [10](#)

Tileset::tile, [15](#)