pytmx Documentation

Release

Author

October 19, 2016

Contents

PyTMX is a map loader for python/pygame designed for games. It provides smart tile loading with a fast and efficient storage base. Not only will does it correctly handle most Tiled object types, it also will load metadata for them, so you can modify your maps and objects in Tiled, instead of modifying your source code.

New support for pysdl2 and pyglet! Check it out!

Because PyTMX was built with games in mind, it differs slightly from Tiled in a few minor aspects:

- Layers not aligned to the grid are not supported.
- Some object metadata attribute names are not supported (see "Reserved Names")

PyTMX strives to balance performance and flexibility. Feel free to use the classes provided in pytmx.py as superclasses for your own maps, or simply load the data with PyTMX and copy the data into your own classes with the api.

Finally, there is no save feature. Once the map is loaded, it will be up to you to provide a way to save changes to the map. I've used the pickle module with good results.

CHAPTER 1

Getting Help

For bugs or feature requests, please use the issues feature of github. For all other general questions, join me on IRC at freennode.net #pygame.

CHAPTER 2

Installation

Install from pip

```
pip install pytmx
```

You can also manually install it

```
python setup.py install
```

CHAPTER 3

Basic use

From a file:

```
>>> import pytmx
>>> tmxdata = pytmx.TiledMap("map.tmx")
```

From a XML string:

```
>>> import pytmx
>>> tmxdata = pytmx.TiledMap.fromstring(xml_string)
```

Load with pygame surfaces:

```
>>> from pytmx.util_pygame import load_pygame
>>> tmxdata = load_pygame("map.tmx")
```

Load with pysdl2 images (experimental):

```
>>> from pytmx.util_pysdl2 import load_pysdl2
>>> tmx_data = load_pysdl2('map.tmx')
```

Load with pyglet images (experimental):

```
>>> from pytmx.util_pyglet import pyglet_image_loader
>>> tmx_data = load_pygame('map.tmx')
```

Getting the tile image:

```
>>> image = tmx_data.get_tile_image(x, y, layer)
>>> screen.blit(image, position)
```

CHAPTER 4

Tile, Object, and Map Properties

Properties are a powerful feature of Tiled that allows the level designer to assign key/value data to individual maps, tilesets, tiles, and objects. Pytmx includes full support for reading this data so you can set parameters for stuff in Tiled, instead of maintaining external data files, or even values in source.

Individual tile properties are accessed through the the parent map object:

```
>>> tmxdata = TiledMap('level1.tmx')
>>> props = txmdata.get_tile_properties(x, y, layer)
>>> props = tmxdata.get_tile_properties_by_gid(tile_gid)
```

All other objects, including the map, layer, objects, etc. are in an python dictionary attribute called “properties”:

```
>>> tmxdata = TiledMap('level1.tmx')
>>> tmxdata.properties['name']
>>> for layer in tmxdata.visible_layers:
>>>     layer.properties['movement_speed']
```

CHAPTER 5

Scrolling Demo

I have another repo with a working demo of a proper scrolling map using Tiled maps and pygame. Please feel free to test drive it. It isn't limited to Tiled maps, you can use any data structure you want, as long as PyGame is used.

https://github.com/bitcraft/pyscroll

CHAPTER 6

Import Notice for PyGame Users

The loader will correctly convert() or convert_alpha() each tile image, so you shouldn't attempt to circumvent the loading mechanisms.

CHAPTER 7

API Documentation

7.1 pytmx package

7.1.1 Submodules

7.1.2 pytmx.pytmx module

class pytmx.pytmx.**TiledElement**
Bases: object

Base class for all pytmx types

allow_duplicate_names = False

classmethod from_xml_string(*xml_string*)
Return a TileElement object from a xml string

Parameters **xml_string** – string containing xml data

Return type TiledElement instance

class pytmx.pytmx.**TiledMap**(*filename=None*, *image_loader=<function default_image_loader>*, ***kwargs*)
Bases: *pytmx.pytmx.TiledElement*

Contains the layers, objects, and images from a Tiled TMX map

This class is meant to handle most of the work you need to do to use a map.

add_layer(*layer*)
Add a layer (TileTileLayer, TiledImageLayer, or TiledObjectGroup)

Parameters **layer** – TileTileLayer, TiledImageLayer, TiledObjectGroup object

add_tileset(*tileset*)
Add a tileset to the map

Parameters **tileset** – TiledTileset

get_layer_by_name(*name*)
Return a layer by name

Parameters **name** – Name of layer. Case-sensitive.

Return type Layer object if found, otherwise ValueError

get_object_by_name(*name*)
Find an object

Parameters **name** – Name of object. Case-sensitive.

Return type Object if found, otherwise ValueError

get_tile_gid(*x*, *y*, *layer*)
Return the tile image GID for this location

Parameters

- **x** – x coordinate
- **y** – y coordinate
- **layer** – layer number

Return type surface if found, otherwise ValueError

get_tile_image(*x*, *y*, *layer*)
Return the tile image for this location

Parameters

- **x** – x coordinate
- **y** – y coordinate
- **layer** – layer number

Return type surface if found, otherwise 0

get_tile_image_by_gid(*gid*)
Return the tile image for this location

Parameters **gid** – GID of image

Return type surface if found, otherwise ValueError

get_tile_locations_by_gid(*gid*)
Search map for tile locations by the GID

Note: Not a fast operation. Cache results if used often.

Parameters **gid** – GID to be searched for

Return type generator of tile locations

get_tile_properties(*x*, *y*, *layer*)
Return the tile image GID for this location

Parameters

- **x** – x coordinate
- **y** – y coordinate
- **layer** – layer number

Return type python dict if found, otherwise None

get_tile_properties_by_gid(*gid*)
Get the tile properties of a tile GID

Parameters **gid** – GID

Return type python dict if found, otherwise None

get_tile_properties_by_layer(*layer*)
Get the tile properties of each GID in layer

Parameters **layer** – layer number

Return type iterator of (gid, properties) tuples

get_tileset_from_gid(*gid*)
Return tileset that owns the gid

Note: this is a slow operation, so if you are expecting to do this often, it would be worthwhile to cache the results of this.

Parameters **gid** – gid of tile image

Return type TiledTileset if found, otherwise ValueError

map_gid(*tiled_gid*)
Used to lookup a GID read from a TMX file's data

Parameters **tiled_gid** – GID that is found in TMX data

Return type (GID, flags) for the the GID passed, None if not found

objectgroups
Return iterator of all object groups

Return type Iterator

objects
Return iterator of all the objects associated with this map

Return type Iterator

parse_xml(*node*)
Parse a map from ElementTree xml node

Parameters **node** – ElementTree xml node

Returns self

register_gid(*tiled_gid*, *flags=None*)
Used to manage the mapping of GIDs between the tmx and pytmx

Parameters **tiled_gid** – GID that is found in TMX data

Return type GID that pytmx uses for the the GID passed

reload_images()
Load the map images from disk

This method will use the image loader passed in the constructor to do the loading or will use a generic default, in which case no images will be loaded.

Returns None

set_tile_properties(*gid*, *properties*)
Set the tile properties of a tile GID

Parameters

- **gid** – GID
- **properties** – python dict of properties for GID

visible_layers
Return iterator of Layer objects that are set 'visible'

Return type Iterator

visible_object_groups
Return iterator of object group indexes that are set 'visible'

Return type Iterator

visible_tile_layers
Return iterator of layer indexes that are set 'visible'

Return type Iterator

class pytmx.pytmx.**TiledTileset**(*parent*, *node*)
Bases: *pytmx.pytmx.TiledElement*

Represents a Tiled Tileset

External tilesets are supported. GID/ID's from Tiled are not guaranteed to be the same after loaded.

parse_xml(*node*)
Parse a Tileset from ElementTree xml element

A bit of mangling is done here so that tilesets that have external TSX files appear the same as those that don't

Parameters **node** – ElementTree element

Returns self

class pytmx.pytmx.**TiledTileLayer**(*parent*, *node*)
Bases: *pytmx.pytmx.TiledElement*

Represents a TileLayer

To just get the tile images, use TiledTileLayer.tiles()

iter_data()
Iterate over layer data

Yields X, Y, GID tuples for each tile in the layer

Returns Generator

parse_xml(*node*)
Parse a Tile Layer from ElementTree xml node

Parameters **node** – ElementTree xml node

Returns self

tiles()
Iterate over tile images of this layer

This is an optimised generator function that returns (tile_x, tile_y, tile_image) tuples,

Return type Generator

Returns (x, y, image) tuples

class pytmx.pytmx.**TiledObject**(*parent*, *node*)
Bases: *pytmx.pytmx.TiledElement*

Represents a any Tiled Object

Supported types: Box, Ellipse, Tile Object, Polyline, Polygon

image

parse_xml(*node*)
Parse an Object from ElementTree xml node

Parameters **node** – ElementTree xml node

Returns self

class pytmx.pytmx.**TiledObjectGroup**(*parent*, *node*)
Bases: *pytmx.pytmx.TiledElement*, list

Represents a Tiled ObjectGroup

Supports any operation of a normal list.

parse_xml(*node*)
Parse an Object Group from ElementTree xml node

Parameters **node** – ElementTree xml node

Returns self

class pytmx.pytmx.**TiledImageLayer**(*parent*, *node*)
Bases: *pytmx.pytmx.TiledElement*

Represents Tiled Image Layer

The image associated with this layer will be loaded and assigned a GID.

image

parse_xml(*node*)
Parse an Image Layer from ElementTree xml node

Parameters **node** – ElementTree xml node

Returns self

class pytmx.pytmx.**TileFlags**(*flipped_horizontally*, *flipped_vertically*, *flipped_diagonally*)
Bases: tuple

flipped_diagonally
Alias for field number 2

flipped_horizontally
Alias for field number 0

flipped_vertically
Alias for field number 1

pytmx.pytmx.**convert_to_bool**(*text*)
Convert a few common variations of "true" and "false" to boolean

Parameters **text** – string to test

Returns boolean

Raises ValueError

pytmx.pytmx.**parse_properties**(*node*)
Parse a Tiled xml node and return a dict that represents a tiled "property"

Parameters **node** – etree element

Returns dict

7.1.3 pytmx.util_pygame module

7.1.4 pytmx.util_pyglet module

7.1.5 pytmx.util_pysdl2 module

7.1.6 Module contents

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

p

V