tdl Documentation

Release 1.6.0

Kyle Benesch

February 08, 2017

Contents

Current documentation is a work in progress.

It will still be a while for the docs to be ported from the old epydoc format.

Until then you should be using the old docs found at: https://pythonhosted.org/tdl/

Contents:

CHAPTER 1

tdl

This is the official documentation for python-tdl. A Pythonic port of U{libtcod<http://roguecentral.org/doryen/libtcod/>}.

You can find the project page on GitHub U{here<https://github.com/HexDecimal/python-tdl>}.

Report any bugs or issues to the GitHub issue tracker U{here<https://github.com/HexDecimal/python-tdl/issues>}.

1.1 Getting Started

Once the library is imported you can load the font you want to use with L{tdl.set_font}. This is optional and when skipped will use a decent default font.

After that you call L{tdl.init} to set the size of the window and get the root console in return. This console is the canvas to what will appear on the screen.

1.2 Indexing Consoles

For most methods taking a position you can use Python-style negative indexes to refer to the opposite side of a console with (-1, -1) starting at the bottom right. You can also check if a point is part of a console using containment logic i.e. ((x, y) in console).

You may also iterate over a console using a for statement. This returns every x,y coordinate available to draw on but it will be extremely slow to actually operate on every coordinate individualy. Try to minimize draws by using an offscreen L{Console}, only drawing what needs to be updated, and using L{Console.blit}.

1.3 Drawing and Colors

Once you have the root console from L{tdl.init} you can start drawing on it using a method such as L{Console.draw_char}. When using this method you can have the char parameter be an integer or a single character string.

The fg and bg parameters expect a variety of types. The parameters default to Ellipsis which will tell the function to use the colors previously set by the L{Console.set_colors} method. The colors set by L{Console.set_colors} are per each L{Console}/L{Window} and default to white on black. You can use a 3-item list/tuple of [red, green, blue] with integers in the 0-255 range with [0, 0, 0] being black and [255, 255, 255] being white. You can even use a single integer of 0xRRGGBB if you like.

Using None in the place of any of the three parameters (char, fg, bg) will tell the function to not overwrite that color or character.

After the drawing functions are called a call to L{tdl.flush} will update the screen.

@undocumented: style

class `tdl.`**`Console`**(*width*, *height*)
Contains character and color data and can be drawn to.

The console created by the L{tdl.init} function is the root console and is the console that is rendered to the screen with L{flush}.

Any console created from the Console class is an off-screen console that can be drawn on before being L{blit} to the root console.

@undocumented: getChar

@ivar tcod_console: Public interface to the cffi TCOD_console_t object of this instance.

Feel free to pass this variable to libtcod-cffi calls but keep in mind that as soon as Console instance is garbage collected the tcod_console will be deleted.

`getChar`(**args*, ***kargs*)
Deprecated version of the function L{get_char}, you should prefer calling that function instead of this one.

exception `tdl.`**`TDLError`**
The catch all for most TDL specific errors.

class `tdl.`**`Window`**(*console*, *x*, *y*, *width*, *height*)
A Window contains a small isolated part of a Console.

Drawing on the Window draws on the Console.

Making a Window and setting its width or height to None will extend it to the edge of the console.

@undocumented: getChar

`drawChar`(**args*, ***kargs*)
Deprecated version of the function L{draw_char}, you should prefer calling that function instead of this one.

`drawFrame`(**args*, ***kargs*)
Deprecated version of the function L{draw_frame}, you should prefer calling that function instead of this one.

`drawRect`(**args*, ***kargs*)
Deprecated version of the function L{draw_rect}, you should prefer calling that function instead of this one.

`getChar`(**args*, ***kargs*)
Deprecated version of the function L{get_char}, you should prefer calling that function instead of this one.

`tdl.`**`flush`**()
Make all changes visible and update the screen.

Remember to call this function after drawing operations. Calls to flush will enfore the frame rate limit set by L{tdl.set_fps}.

This function can only be called after L{tdl.init}

`tdl.`**`set_title`**(*title*)
Change the window title.

@type title: string

tdl.**init**(*width*, *height*, *title=None*, *fullscreen=False*, *renderer=u'OPENGL'*)
Start the main console with the given width and height and return the root console.

Call the consoles drawing functions. Then remember to use L{tdl.flush} to make what's drawn visible on the console.

@type width: int @param width: width of the root console (in tiles)

@type height: int @param height: height of the root console (in tiles)

@type title: string @param title: Text to display as the window title.

If left None it defaults to the running scripts filename.

@type fullscreen: boolean @param fullscreen: Can be set to True to start in fullscreen mode.

@type renderer: string @param renderer: Can be one of 'GLSL', 'OPENGL', or 'SDL'.

Due to way Python works you're unlikely to see much of an improvement by using 'GLSL' over 'OPENGL' as most of the time Python is slow interacting with the console and the rendering itself is pretty fast even on 'SDL'.

@rtype: L{Console} @return: The root console. Only what is drawn on the root console is

what's visible after a call to L{tdl.flush}. After the root console is garbage collected, the window made by this function will close.

@see: L{Console}, L{set_font}

tdl.**set_fps**(*frameRate*)
Set the maximum frame rate.

@type frameRate: int @param frameRate: Further calls to L{tdl.flush} will limit the speed of

the program to run at <frameRate> frames per second. Can also be set to 0 to run without a limit.

Defaults to None.

tdl.**set_fullscreen**(*fullscreen*)
Changes the fullscreen state.

@type fullscreen: boolean

tdl.**screenshot**(*path=None*)
Capture the screen and save it as a png file

@type path: string @param path: The filepath to save the screenshot.

If path is None then the image will be placed in the current folder with the names: screenshot001.png, screenshot002.png, ...

tdl.**force_resolution**(*width*, *height*)
Change the fullscreen resoulution

@type width: int @type height: int

tdl.**set_font**(*path*, *columns=None*, *rows=None*, *columnFirst=False*, *greyscale=False*, *altLayout=False*)
Changes the font to be used for this session. This should be called before L{tdl.init}

If the font specifies its size in its filename (i.e. font_NxN.png) then this function can auto-detect the tileset formatting and the parameters columns and rows can be left None.

While it's possible you can change the font mid program it can sometimes break in rare circumstances. So use caution when doing this.

@type path: string @param path: Must be a string filepath where a bmp or png file is found.

@type columns: int @param columns: Number of columns in the tileset.

Can be left None for auto-detection.

@type rows: int @param rows: Number of rows in the tileset.

Can be left None for auto-detection.

@type columnFirst: boolean @param columnFirst: Defines if the characer order goes along the rows or

colomns. It should be True if the charater codes 0-15 are in the first column. And should be False if the characters 0-15 are in the first row.

@type greyscale: boolean @param greyscale: Creates an anti-aliased font from a greyscale bitmap.

Otherwise it uses the alpha channel for anti-aliasing.

Unless you actually need anti-aliasing from a font you know uses a smooth greyscale channel you should leave this on False.

@type altLayout: boolean @param altLayout: An alternative layout with space in the upper left

corner. The colomn parameter is ignored if this is True, find examples of this layout in the font/libtcod/ directory included with the python-tdl source.

@raise TDLError: Will be raised if no file is found at path or if auto- detection fails.

@note: A png file that's been optimized can fail to load correctly on MAC OS X creating a garbled mess when rendering. Don't use a program like optipng or just use bmp files instead if you want your program to work on macs.

`tdl.`**`get_fullscreen`**`()`

Returns True if program is fullscreen.

@rtype: boolean @return: Returns True if the window is in fullscreen mode.

Otherwise returns False.

`tdl.`**`get_fps`**`()`

Return the current frames per second of the running program set by L{set_fps}

@rtype: int @return: Returns the frameRate set by set_fps.

If set to no limit, this will return 0.

CHAPTER 2

tdl.event

This module handles user input.

To handle user input you will likely want to use the L{event.get} function or create a subclass of L{event.App}.

- L{event.get} iterates over recent events.
- L{event.App} passes events to the overridable methods: ev_* and key_*.

But there are other options such as L{event.keyWait} and L{event.isWindowClosed}.

A few event attributes are actually string constants. Here's a reference for those:

- L{Event.type}

  'QUIT', 'KEYDOWN', 'KEYUP', 'MOUSEDOWN', 'MOUSEUP', or 'MOUSEMOTION.'

- L{MouseButtonEvent.button} (found in L{MouseDown} and L{MouseUp} events)

  'LEFT', 'MIDDLE', 'RIGHT', 'SCROLLUP', 'SCROLLDOWN'

- L{KeyEvent.key} (found in L{KeyDown} and L{KeyUp} events)

  'NONE', 'ESCAPE', 'BACKSPACE', 'TAB', 'ENTER', 'SHIFT', 'CONTROL', 'ALT', 'PAUSE', 'CAPSLOCK', 'PAGEUP', 'PAGEDOWN', 'END', 'HOME', 'UP', 'LEFT', 'RIGHT', 'DOWN', 'PRINTSCREEN', 'INSERT', 'DELETE', 'LWIN', 'RWIN', 'APPS', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'KP0', 'KP1', 'KP2', 'KP3', 'KP4', 'KP5', 'KP6', 'KP7', 'KP8', 'KP9', 'KPADD', 'KPSUB', 'KPDIV', 'KPMUL', 'KPDEC', 'KPENTER', 'F1', 'F2', 'F3', 'F4', 'F5', 'F6', 'F7', 'F8', 'F9', 'F10', 'F11', 'F12', 'NUMLOCK', 'SCROLLLOCK', 'SPACE', 'CHAR'

class `tdl.event.`**`Quit`**
Fired when the window is closed by the user.

class `tdl.event.`**`MouseUp`**(*button*, *pos*, *cell*)
Fired when a mouse button is released.

class `tdl.event.`**`KeyDown`**(*key*, *char*, *lalt*, *lctrl*, *ralt*, *rctrl*, *shift*)
Fired when the user presses a key on the keyboard or a key repeats.

class `tdl.event.`**`MouseMotion`**(*pos*, *cell*, *motion*, *cellmotion*)
Fired when the mouse is moved.

`cell` = None
(x, y) position of the mouse snapped to a cell on the root console. type: (int, int)

`cellmotion` = None
(x, y) mostion of the mouse moving over cells on the root console. type: (int, int)

motion = None
(x, y) motion of the mouse on the screen. type: (int, int)

pos = None
(x, y) position of the mouse on the screen. type: (int, int)

tdl.event.**key_wait**()
Waits until the user presses a key. Then returns a L{KeyDown} event.

Key events will repeat if held down.

A click to close the window will be converted into an Alt+F4 KeyDown event.

@rtype: L{KeyDown}

tdl.event.**set_key_repeat**(*delay=500*, *interval=0*)
Change or disable key repeat.

@type delay: int @param delay: Milliseconds before a held key begins to repeat.

Key repeat can be disabled entirely by setting a delay of zero.

@type interval: int @param interval: Milliseconds between key repeats.

An interval of zero will repeat every frame.

tdl.event.**get**()
Flushes the event queue and returns the list of events.

This function returns L{Event} objects that can be identified by their type attribute or their class.

@rtype: iterator @return: Returns an iterable of objects derived from L{Event} or anything

put in a L{push} call. If the iterator is deleted or otherwise interrupted before finishing the excess items are preserved for the next call.

class tdl.event.**MouseDown**(*button*, *pos*, *cell*)
Fired when a mouse button is pressed.

class tdl.event.**App**
Application framework.

- ev_*: Events are passed to methods based on their L{Event.type} attribute. If an event type is 'KEYDOWN' the ev_KEYDOWN method will be called with the event instance as a parameter.
- key_*: When a key is pressed another method will be called based on the L{KeyEvent.key} attribute. For example the 'ENTER' key will call key_ENTER with the associated L{KeyDown} event as its parameter.
- L{update}: This method is called every loop. It is passed a single parameter detailing the time in seconds since the last update (often known as deltaTime.)

  You may want to call drawing routines in this method followed by L{tdl.flush}.

ev_KEYDOWN(*event*)
Override this method to handle a L{KeyDown} event.

ev_KEYUP(*event*)
Override this method to handle a L{KeyUp} event.

ev_MOUSEDOWN(*event*)
Override this method to handle a L{MouseDown} event.

ev_MOUSEMOTION(*event*)
Override this method to handle a L{MouseMotion} event.

ev_MOUSEUP(*event*)
Override this method to handle a L{MouseUp} event.

ev_QUIT(*event*)
Unless overridden this method raises a SystemExit exception closing the program.

run()
Delegate control over to this App instance. This function will process all events and send them to the special methods ev_* and key_*.

A call to L{App.suspend} will return the control flow back to where this function is called. And then the App can be run again. But a single App instance can not be run multiple times simultaneously.

runOnce(**args*, ***kargs*)
Deprecated version of the function L{run_once}, you should prefer calling that function instead of this one.

run_once()
Pump events to this App instance and then return.

This works in the way described in L{App.run} except it immediately returns after the first L{update} call.

Having multiple L{App} instances and selectively calling runOnce on them is a decent way to create a state machine.

suspend()
When called the App will begin to return control to where L{App.run} was called.

Some further events are processed and the L{App.update} method will be called one last time before exiting (unless suspended during a call to L{App.update}.)

update(*deltaTime*)
Override this method to handle per frame logic and drawing.

@type deltaTime: float @param deltaTime: This parameter tells the amount of time passed since

the last call measured in seconds as a floating point number.

You can use this variable to make your program frame rate independent. Use this parameter to adjust the speed of motion, timers, and other game logic.

class tdl.event.**KeyUp**(*key*, *char*, *lalt*, *lctrl*, *ralt*, *rctrl*, *shift*)
Fired when the user releases a key on the keyboard.

tdl.event.**wait**(*timeout=None*, *flush=True*)
Wait for an event.

@type timeout: int or None @param timeout: The time in seconds that this function will wait before

giving up and returning None.

With the default value of None, this will block forever.

@type flush: boolean @param flush: If True a call to L{tdl.flush} will be made before listening

for events.

@rtype: L{Event} or None @return: Returns an instance derived from L{Event}, or None if the function

has timed out. Anything added via L{push} will also be returned.

@since: 1.4.0

tdl.event.**is_window_closed**()
Returns True if the exit button on the window has been clicked and stays True afterwards.

@rtype: boolean

tdl.event.**push**(*event*)
Push an event into the event buffer.

@type event: L{Event}-like object @param event: The event will be available on the next call to L{event.get}.

An event pushed in the middle of a L{get} will not show until the next time L{get} called preventing push related infinite loops.

This object should at least have a 'type' attribute.

class tdl.event.**Event**
Base Event class.

You can easily subclass this to make your own events. Be sure to set the class attribute L{Event.type} for it to be passed to a custom L{App} ev_* method.

type = None
String constant representing the type of event.

The L{App} ev_* methods depend on this attribute.

Can be: 'QUIT', 'KEYDOWN', 'KEYUP', 'MOUSEDOWN', 'MOUSEUP', or 'MOUSEMOTION.'

CHAPTER 3

tdl.map

Rogue-like map utilitys such as line-of-sight, field-of-view, and path-finding.

class tdl.map.**Map**(*width*, *height*)

Fast field-of-view and path-finding on stored data.

Set map conditions with the walkable and transparency attributes, this object can be iterated and checked for containment similar to consoles.

For example, you can set all tiles and transparent and walkable with the following code:

```
map = tdl.map.Map(80, 60)
for x,y in map:
    map.transparent[x,y] = true
    map.walkable[x,y] = true
```

@ivar transparent: Map transparency, access this attribute with map.transparent[x,y]

Set to True to allow field-of-view rays, False will block field-of-view.

Transparent tiles only affect field-of-view.

@ivar walkable: Map accessibility, access this attribute with map.walkable[x,y]

Set to True to allow path-finding through that tile, False will block passage to that tile.

Walkable tiles only affect path-finding.

@ivar fov: Map tiles touched by a field-of-view computation, access this attribute with map.fov[x,y]

Is True if a the tile is if view, otherwise False.

You can set to this attribute if you want, but you'll typically be using it to read the field-of-view of a L{compute_fov} call.

@since: 1.5.0

compute_fov(*x*, *y*, *fov='PERMISSIVE'*, *radius=None*, *light_walls=True*, *sphere=True*, *cumulative=False*)

Compute the field-of-view of this Map and return an iterator of the points touched.

@type x: int @type y: int

@param x: x center of the field-of-view @param y: y center of the field-of-view @type fov: string @param fov: The type of field-of-view to be used. Available types are:

'BASIC', 'DIAMOND', 'SHADOW', 'RESTRICTIVE', 'PERMISSIVE', 'PERMISSIVE0', 'PERMISSIVE1', ..., 'PERMISSIVE8'

@type radius: int @param radius: Raduis of the field-of-view. @type light_walls: boolean @param light_walls: Include or exclude wall tiles in the field-of-view. @type sphere: boolean @param sphere: True for a spherical field-of-view.

False for a square one.

@type cumulative: boolean @param cumulative:

@rtype: iter((x, y), ...) @return: An iterator of (x, y) points of tiles touched by the

field-of-view.

Unexpected behaviour can happen if you modify the Map while using the iterator.

You can use the Map's fov attribute as an alternative to this iterator.

compute_path(*start_x*, *start_y*, *dest_x*, *dest_y*, *diagonal_cost=1.4142135623730951*)
Get the shortest path between two points.

The start position is not included in the list.

@type diagnalCost: float @param diagnalCost: Multiplier for diagonal movement.

Can be set to zero to disable diagonal movement entirely.

@rtype: [(x, y), ...] @return: Returns a the shortest list of points to get to the destination

position from the starting position

tdl.map.**bresenham**(*x1*, *y1*, *x2*, *y2*)
Return a list of points in a bresenham line.

Implementation hastily copied from RogueBasin.

@return: Returns a list of (x, y) points, including both the start and endpoints.

tdl.map.**quick_fov**(*x*, *y*, *callback*, *fov='PERMISSIVE'*, *radius=7.5*, *lightWalls=True*, *sphere=True*)
All field-of-view functionality in one call.

Before using this call be sure to make a function, lambda, or method that takes 2 positional parameters and returns True if light can pass through the tile or False for light-blocking tiles and for indexes that are out of bounds of the dungeon.

This function is 'quick' as in no hassle but can quickly become a very slow function call if a large radius is used or the callback provided itself isn't optimized.

Always check if the index is in bounds both in the callback and in the returned values. These values can go into the negatives as well.

@type x: int @param x: x center of the field-of-view @type y: int @param y: y center of the field-of-view @type callback: function @param callback: This should be a function that takes two positional arguments x,y

and returns True if the tile at that position is transparent or False if the tile blocks light or is out of bounds.

@type fov: string @param fov: The type of field-of-view to be used. Available types are:

'BASIC', 'DIAMOND', 'SHADOW', 'RESTRICTIVE', 'PERMISSIVE', 'PERMISSIVE0', 'PERMISSIVE1', ..., 'PERMISSIVE8'

@type radius: float @param radius: Raduis of the field-of-view.

When sphere is True a floating point can be used to fine-tune the range. Otherwise the radius is just rounded up.

Be careful as a large radius has an exponential affect on how long this function takes.

@type lightWalls: boolean @param lightWalls: Include or exclude wall tiles in the field-of-view. @type sphere: boolean @param sphere: True for a spherical field-of-view. False for a square one.

@rtype: set((x, y), ...) @return: Returns a set of (x, y) points that are within the field-of-view.

class `tdl.map.`**AStar**(*width*, *height*, *callback*, *diagnalCost=1.4142135623730951*, *advanced=False*)
A* pathfinder

Using this class requires a callback detailed in L{AStar.__init__}

@undocumented: getPath

getPath(**args*, ***kargs*)
Deprecated version of the function L{get_path}, you should prefer calling that function instead of this one.

get_path(*origX*, *origY*, *destX*, *destY*)
Get the shortest path from origXY to destXY.

@rtype: [(x, y), ...] @return: Returns a list walking the path from origXY to destXY.

This excludes the starting point and includes the destination.

If no path is found then an empty list is returned.

CHAPTER 4

tdl.noise

This module provides advanced noise generation.

Noise is sometimes used for over-world generation, height-maps, and cloud/mist/smoke effects among other things.

You can see examples of the available noise algorithms in the libtcod documentation U{here<http://doryen.eptalys.net/data/libtcod/doc/1.5.1/html2/noise.html>}.

class `tdl.noise.`**`Noise`**(*algorithm='PERLIN'*, *mode='FLAT'*, *hurst=0.5*, *lacunarity=2.0*, *octaves=4.0*, *seed=None*, *dimensions=4*)

An advanced noise generator.

`getPoint`(**args*, ***kargs*)

Deprecated version of the function L{get_point}, you should prefer calling that function instead of this one.

`get_point`(**position*)

Return the noise value of a specific position.

Example usage: value = noise.getPoint(x, y, z) @type position: floats @param position:

@rtype: float @return: Returns the noise value at position.

This will be a floating point in the 0.0-1.0 range.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

t

T

U

W