Direct Python Audio/Video

Release 0.0.1

Vibrant Labs

May 01, 2022

CONTENTS:

CHAPTER

ONE

DPAV

1.1 Audio

`class audio.Audio`

Bases: `object`

Handles Audio capabilities of Python Direct Platform.

Functions:

Constructor: __init__()

Functions:

play_sound(Hz, length)

If audio buffer is set: play_sound()

play_sample(string_name_of_wav_file)

Setters: set_audio_buffer(numpyarray) set_audio_device(int) set_waveform(waveform)

Getters: get_bit_number()->int get_sample_rate()->int get_audio_buffer() get_audio_device()->Returns int corresponding to audio device

Misc: list_audio_devices() wait_for_sound_end()

`get_audio_buffer()`

Returns the audio buffer of the Audio class

Description: This will return none if the audio buffer has not been set by the set_audio_buffer method.

audioobject.get_audio_buffer()

Parameters `None` –

Returns numpy array

Return type self._audio_buffer

`get_audio_device()` → int

Gets the current audio device number of the Audio Class

Description: Assuming audioobject.set_audio_device(2) is called, audioobject.get_audio_device() would return 2 [index of audio device in audioobject.list_audio_devices()]

Parameters `None` –

Returns self._audio_device: int value

Notes

Returns the integer value of the device not the device name

`get_bit_number()` → int

Gets the bit rate of the Audio class

Description: Bit rate currently locked to 16 bits

Parameters `None` –

Returns The bit rate of the Audio class - int value

Return type self._bit_number

`get_sample_rate()` → int

Gets the sample rate of the Audio class.

Description: Sample rate is currently locked to 44100

Parameters `None` –

Returns The sample rate of the audioClass - int value

Return type self._sample_rate

`list_audio_devices()` → None

Lists the output devices on your system and adds to list self._devices

Description: Run this function before using set_audio_device() to add devices to the list devices

audioobject.list_audio_devices() 0 Speakers (Realtek(R) Audio) 1 VGA248 (2-NVIDIA High Def Audio) 2 Speakers (HyperX Cloud II Wireless)

Parameters `None` –

Returns None

`play_sample`(*sample_name: str*) → None

Plays sounds that are wav, ogg or mp3 files.

Description: audioobject.play_sample(mypath.mp3) would play sounds from the file mypath.mp3

Parameters `sample_name` – String path or name of sound

Returns None

`play_sound`(*input_frequency=0*, *input_duration=0*) → None

Primary sound playing method of the audio class.

Description: Play sounds directly from this function Need to run set_audio_device() or will default to the default audio device You can use set_waveform to change the type. play_sound is somewhat overloaded to where if you have an audioBuffer set using set_audio_buffer, you can call play_sound()

and it will play whatever that audio_buffer is e.g. wav files Example in examples/custombuffer.py

play_sound(440, 1) would play an A note for one second with the sin waveform set.

Parameters

- `input_frequency` – int value - input frequency in Hz
- `input_duration` – int value - duration in seconds

Raises `TypeError` – If input_duration not a number, or < 0

Returns None

`set_audio_buffer`(*ab*) → None

Sets the audio buffer of the Audio Class.

Description: The audio buffer needs to have two rows so that way stereo works as intended. You can set the audio buffer to wav file data by fetching numpy arrays using wav or scipy, however only 16 bit waves are supported. This process can be seen in custom_buffer.py w/ the utility function sixteenWavtoRawData

Examples: # 44100 = sample rate # 32767 is 2 ^ (our bit depth -1)-1 and is essentially the number of samples per time stamp # 260 and 290 are our tones in hz # Below generates a buffer 1 second long of sin wave data-identical to the method used in house data = numpy.zeros((44100, 2), dtype=numpy.int16) for s in range(44100):

t = float(s) / 44100 data[s][0] = int(round(32767 * math.sin(2 * math.pi * 260 * t))) data[s][1] = int(round(32767 * math.sin(2 * math.pi * 290 * t)))

audioobject.set_audio_buffer(data)

Parameters `ab` – numpy array of shape(samples, channels) e.g. ab[44100][2]

Returns None

`set_audio_device`(*device: int*) → int

Sets the current audio device of the Audio class.

Description: This can only be set ONCE per instance. To change devices, del the current instance set the new device, and continue This needs to be run after list_audio_device() in order to see list of audio devices If not run the device will default to the current device being used by the machine

audioobject.set_audio_device(2) Based on example in list_audio_devices() this would change the device to Speakers (HyperX Cloud II Wireless)

Parameters `device` – int value - see all int values for each device by running list_audio_devices()

Returns None

`set_waveform`(*wave*) → None

Sets the expression governing the wave form playing

Description: play_audio uses this in buffer generation

audioobject.set_waveform(object.wave_table.sin) This would change to the waveform sin contained in the wave_table class The wave functions need to take in a input frequency as well as a timestep parameter to solve for a particular frequency at a given time step. See wave_table for an example of this.

Parameters `Wave` – takes a mathematical expression function 'pointer' in the form of f(inputfreq, timestep)

Returns None

`wait_for_sound_end()`

Function call that is placed at the end of scripts without a pygame window instance so sounds play to their full duration without a

Description: Placed at the end of python files that do not have loops. Otherwise, sounds would be cut off prematurely.

Example: play_sound(440, 10) wait_for_sound_end() # This prevents the process from closing out before the sound ends.

Parameters `None` –

Returns None

Notes:

`class audio.wave_table`

Bases: `object`

This is a class holding waveforms for usage with the play_sound method.

There are 5 waveforms: sin saw square noise triangle

Example

waves = wave_table() sinefunc = waves.sin

`noise`(*input_frequency*, *t*)

Random white noise

Description: Warning: VERY LOUD

Parameters

- **`input_frequency`** –
- **`t`** –

Return type random.random() * input_frequency * t

`saw`(*input_frequency*, *t*)

Saw wave

Parameters

- **`input_frequency`** –
- **`t`** –

Return type t * input_frequency - math.floor(t * input_frequency)

`sin`(*input_frequency*, *t*)

Sin wave form, default for libary

Parameters

- **`input_frequency`** –
- **`t`** –

Return type math.sin(2 * math.pi * input_frequency * t) `square`(*input_frequency*, *t*)

Square wave form

Parameters

- **`input_frequency`** –
- **`t`** –

Return type round(math.sin(2 * math.pi * input_frequency * t))

`triangle`(*input_frequency*, *t*)

Triangle wave, similar in sound to saw + sin together

Parameters

- **`input_frequency`** –
- **`t`** –

Return type 2 * abs((t * input_frequency) / 1 - math.floor(((t * input_frequency) / 1) + 0.5))

1.2 VBuffer

`class vbuffer.VBuffer`(*arg1: list* | *tuple* | *numpy.ndarray = (800, 600)*)

Bases: `object`

Visual buffer for the Python Direct Platform

Holds a 2D array of hex color values. Each element represents a pixel, whose coordinates are its index. VBuffer can be loaded and displayed by the window class.

Parameters **`arg1`** (*`{(int, int)|np.ndarray(int, int)}`*) – Either array dimensions or a 2-dimensional numpy array of integers

If dimensions, will create zeroed-out 2D array of the selected dimensions. Defaults to 800x600.

If numpy array, will set buffer to the contents of that array.

`Constructor:`

__init__(self, arg1=(800, 600)) -> None

`Overloads:`

__getitem__(self, idx) -> int __setitem__(self, idx, val) -> None __len__(self) -> int

`properties:`

getter: dimensions(self) -> (int, int)

setter: dimensions(self, val) -> None

`Setter:`

write_pixel(self, coords, val) -> None set_buffer(self, buf) -> None clear(self) -> None fill(self, color: int) -> None

`Getters:`

get_pixel(self, coords) -> int get_dimensions(self) -> (int, int)

`File I/O:`

save_buffer_to_file(self, filename) -> None load_buffer_from_file(self, filename) -> None

Error Checking:

_check_numpy_arr(self,arg1,arg_name,method_name) -> None _check_coord_type(self, coords, arg_name, method_name) -> None _check_coord_vals(self, x, y, method_name) -> None

`clear()` → None

Set every pixel in buffer to 0 (hex value for black).

`property dimensions: list | tuple`

Return dimensions of buffer.

`fill`(*color: int*) → None

Set every pixel in the buffer to a given color.

Parameters **`color`** (*`Hex color code`*) –

`get_dimensions()` → list | tuple

Return dimensions of visual buffer array.

`get_pixel`(*coords: list* | *tuple*) → int

Return color value of chosen pixel.

Parameters **`coords`** (*`2-tuple or list containing first and second index of pixel`*) –

`load_buffer_from_file`(*filename: str*) → None

Load binary file storing buffer contents, and write it to buffer.

Parameters **`filename`** (*`Path to a binary file containing numpy array data`*) –

`save_buffer_to_file`(*filename: str*) → None

Save contents of buffer to a binary file.

Parameters **`filename`** (*`The path and name of the file to write to`*) –

`set_buffer`(*buf: numpy.ndarray*) → None

Set the visual buffer to equal a provided 2D array of pixels.

Parameters **`buf`** (*`A 2-dimensional numpy array of integer color values`*) –

`write_pixel`(*coords: list* | *tuple*, *val: int*) → None

Sets pixel at specified coordinates to specified color.

Sets pixel at coordinates coords in buffer to hex value val

Parameters

- **`coords`** (*`Pixel coordinates (an X and a Y)`*) –
- **`val`** (*`The hex value of the desired color to change the pixel with`*) –

:raises TypeError : val is not type(int): :raises ValueError : val is negative or greater than max color value (0xFFFFFF):

1.3 Window

class window.**Window**(*arg1: Optional[dpav.vbuffer.VBuffer] = None*, *scale: float = 1.0*)

Bases: `object`

Handles Window capabilites of Python Direct Platform Functions:

Constructor: __init__()

Setters: set_scale(int/float) set_vbuffer(VBuffer/np.ndarray,optional:int)

Getters: get_mouse_pos()

Misc Methods: open() is_open() close() update()

Private Methods: _update_events(pygame.event) _build_events_dict() _write_to_screen()

Public

vbuffer: active VBuffer object scale: number that scales up/down the size of the screen

(1.0 is unscaled)

events: dictionary of string:bool event pairs,

example: "l_shift": True – left shift is pressed down "l_shift": False – left shift is not pressed

eventq: list of active events that occured since last update cycle

example: ['l_shift', 'mouse', 'a', 'q']

debug_flag: boolean flag if window object should output debug info to log open_flag: boolean flag for if the window is active

Private

_keydict: int:string PyGame event mapping. PyGame events identifiers are stored as ints. This attribute is used by the public events variable to map from PyGame's integer:boolean pairs to our string:boolean pairs

_surfaces: Two PyGame Surfaces for swapping to reflect vbuffer changes and enable in-place nparray modification

_screen: PyGame.display object, used for viewing vbuffer attribute

close() → None

Closes the active instance of a pygame window

Raises `RuntimeError` – no active pygame window instances exists

get_mouse_pos() -> (<class 'int'>, <class 'int'>)

Returns the current mouse location with respect to the pygame window instance

Raises `Runtime Error` – no active pygame window instances exists

is_open() → bool

Updates events on every call, used to abstract out PyGame display calls and event loop

Example

if window.is_open(): # your code here

Returns boolean denoting if the window is currently open

`open()` → None

Creates and runs pygame window in a new thread

`set_scale`(*scale: float*) → None

Sets the window scale

`set_vbuffer`(*arg1: dpav.vbuffer.VBuffer*) → None

Sets the vbuffer/nparray object to display on screen

Parameters **`arg1`** – VBuffer/np.ndarray

Raises

- **`TypeError`** – arg1 VBuffer/np.ndarray type check
- **`TypeError`** – scale int/float type check

`update()` → None

Pygame event abstraction, called at end of pygame loop. Optional function if is_open() is used

Raises **`Runtime Error`** – No active pygame window

1.4 Utility

CHAPTER TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX